RISC-V Supervisor Binary Interface Specification

RISC-V Platform Runtime Services Task Group

Version v3.0, 2025-07-16: Ratified

RISC-V 监督者二进制接口规范

RISC-V 平台运行时服务任务组

版本 v3.0，2025-07-16：已批准

Table of Contents

目录

Preamble

This document is in the Ratified state

No changes are allowed. Any necessary or desired modifications must be addressed through a follow-on extension. Ratified extensions are never revised.

前言

本文件处于已批准状态

不允许进行任何修改。任何必要或期望的变更必须通过后续扩展来解决。已批准的扩展永远不会被修订。

Copyright and license information

版权所有与许可信息

Contributors

This RISC-V specification has been contributed to directly or indirectly by:

Abner Chang <abner.chang@hpe.com>
Al Stone <ahs3@ahs3.net>
Andrew Jones <ajones@ventanamicro.com>
Anup Patel <apatel@ventanamicro.com>
Atish Patra <atishp04@gmail.com>
Atish Patra <atishp@rivosinc.com>
Bin Meng <bmeng.cn@gmail.com>
Chris Williams <diodesign@tuta.io>
Clément Léger <cleger@rivosinc.com>
Conor Dooley <conor.dooley@microchip.com>
Daniel Schaefer <git@danielschaefer.me>
Esteban Blanc <estblcsk@gmail.com>
hasheddan <georgedanielmangum@gmail.com>
Heinrich Schuchardt <xypron.glpk@gmx.de>
Jeff Scheel <jeff@riscv.org>
Jessica Clarke <jrtc27@jrtc27.com>
john <799433746@qq.com>
Konrad Schwarz <konrad.schwarz@siemens.com>
Luo Jia / Zhouqi Jiang <luojia@hust.edu.cn>
Nick Kossifidis <mickflemm@gmail.com>
Palmer Dabbelt <palmer@dabbelt.com>
Paolo Bonzini <pbonzini@redhat.com>
Rahul Pathak <rpathak@ventanamicro.com>
Samuel Holland <samuel.holland@sifive.com>
Sean Anderson <seanga2@gmail.com>
Stefano Stabellini <stefano.stabellini@amd.com>
Sunil V L <sunilvl@ventanamicro.com>
Tsukasa OI <research_trasio@irq.a4lg.com>
Yiting Wang <yiting.wang@windriver.com>

贡献者

本 RISC-V 规范的直接或间接贡献者包括：

Abner Chang Al Stone Andrew Jones Anup Patel
Atish Patra Atish Patra Bin Meng Chris Williams
Clément Léger Conor Dooley Daniel Schaefer
Esteban Blanc hasheddan Heinrich Schuchardt Jeff
Scheel Jessica Clarke john <799433746@qq.com>
Konrad Schwarz 罗佳/江周琦 Nick Kossifidis Palmer
Dabbelt Paolo Bonzini Rahul Pathak Samuel
Holland Sean Anderson Stefano Stabellini Sunil V L
Tsukasa OI 王艺婷

Change Log

Version 3.0

- Update the document state to Ratified

Version 3.0-rc8

- Clarifications around legacy extension and SSE events

Version 3.0-rc7

- Update the document state to Frozen.

Version 3.0-rc6

- Clarification for SSE errors.
- Formatting changes.
- Update the preamble.

Version 3.0-rc5

- Update doc-resourses
- Clarifications/minor semantic fixes in mpxy/sse.

Version 3.0-rc4

- Added landing pad and double trap related bits to the SSE extension.
- Several clarifications in SSE and MPXY extensions.
- Added a new function to retrieve the shared memory size in MPXY extension.

Version 3.0-rc3

- Added low priority RAS events in SSE.
- Miscallenous clarification around reserved bits, fwft, notification events.
- Added a dedicated error code for fwft set denial due to lock status.

Version 3.0-rc1/rc2

- Added SBI PMU event info function and new raw event type
- Added SBI MPXY extension
- Added error code SBI_ERR_TIMEOUT
- Added error code SBI_ERR_IO

变更日志

版本 3.0

- 将文档状态更新为"已批准"

版本 3.0-rc8

- 关于遗留扩展和 SSE 事件的说明澄清

版本 3.0-rc7

- 将文档状态更新为"冻结"。

版本 3.0-rc6

- 关于 SSE 错误的说明澄清。
- 格式调整
- 前言更新

版本 3.0-rc5

- 文档资源更新
- 对 mpxy/sse 扩展进行了澄清说明/细微语义修正。

版本 3.0-rc4

- 为 SSE 扩展添加了着陆垫和双重陷阱相关功能位。
- 对 SSE 和 MPXY 扩展进行了多处澄清说明。
- 在 MPXY 扩展中新增了获取共享内存大小的功能。

版本 3.0-rc3

- 在 SSE 中新增了低优先级 RAS 事件支持。
- 围绕保留位、fwft 及通知事件进行了多项说明性澄清。
- 新增了因锁定状态导致固件功能表设置被拒绝的专用错误码

版本 3.0-rc1/rc2

- 新增了 SBI 性能监控单元事件信息功能及新型原始事件类型
- 新增了 SBI MPXY 扩展
- 新增错误码 SBI_ERR_TIMEOUT
- 新增错误码 SBI_ERR_IO

- Added sse mask/unmask function and pointer masking bit in fwft
- Clarify SBI IPI and RFENCE error codes
- Clarify the description of the `set_timer` function
- Added SBI DBTR extension
- Added SBI FWFT extension
- Added SBI SSE extension
- Added error code SBI_ERR_BAD_RANGE
- Added error code SBI_ERR_INVALID_STATE

Version 2.0

- Clarification around SBI PMU set memory function
- Base extension function name typo fix
- Upate the document state to Ratified

Version 2.0-rc8

- Clarfications STA extension and counter index in the pmu snapshot.

Version 2.0-rc7

- Few clarfications around system suspend and pmu snapshot.

Version 2.0-rc6

- Few clarifications around rfence extensions
- Marks public review period complete.

Version 2.0-rc5

- Update the document state to Frozen

Version 2.0-rc4

- Added flags parameter to sbi_pmu_snapshot_set_shmem()
- Return error code SBI_ERR_NO_SHMEM in SBI PMU extension wherever applicable
- Made flags parameter of sbi_steal_time_set_shmem() as unsigned long
- Split the specification into multiple adoc files
- Add more clarification for firmware/vendor/experimental extension space.
- Fix ambiguous usage of normative statements.

Version 2.0-rc3

- 在 fwft 中新增了 sse 掩码/解掩功能及指针掩码位
- 明确了 SBI IPI 与 RFENCE 的错误码规范
- 优化了 set_timer 函数的描述说明
- 新增 SBI DBTR 扩展模块
- 新增 SBI FWFT 扩展
- 新增 SBI SSE 扩展
- 新增错误码 SBI_ERR_BAD_RANGE
- 新增错误码 SBI_ERR_INVALID_STATE

版本 2.0

- 澄清关于 SBI PMU 设置内存功能的说明
- 修正基础扩展功能名称的拼写错误
- 将文档状态更新为"已批准"

版本 2.0-rc8

- 澄清了 STA 扩展及性能监控单元快照中的计数器索引问题

版本 2.0-rc7

- 针对系统挂起和性能监控单元快照功能进行了若干说明性补充

版本 2.0-rc6

- 围绕 rfence 扩展的若干说明澄清
- 标志着公开评审阶段完成

版本 2.0-rc5

- 将文档状态更新为"冻结"

版本 2.0-rc4

- 为 sbi_pmu_snapshot_set_shmem()函数添加了 flags 参数
- 在 SBI PMU 扩展中所有适用场景下返回错误码 SBI_ERR_NO_SHMEM
- 将 sbi_steal_time_set_shmem()的 flags 参数类型改为 unsigned long
- 将规范拆分为多个 adoc 文件
- 为固件/厂商/实验性扩展空间添加更多说明
- 修正规范性表述的模糊用法

版本 2.0-rc3

- CI support added
- Fix revmark in the makefile.
- Few minor cleanups.

Version 2.0-rc2

- Added clarification for SUSP, NACL & STA extensions.
- Standardization of hart usage.
- Added an error code in SBI DBCN extension.

Version 2.0-rc1

- Added common description for shared memory physical address range parameter
- Added SBI debug console extension
- Relaxed the counter width requirement on SBI PMU firmware counters
- Added sbi_pmu_counter_fw_read_hi() in SBI PMU extension
- Reserved space for SBI implementation specific firmware events
- Added SBI system suspend extension
- Added SBI CPPC extension
- Clarified that an SBI extension can be partially implemented only if it defines a mechanism to discover implemented SBI functions
- Added error code SBI_ERR_NO_SHMEM
- Added SBI nested acceleration extension
- Added common description for a virtual hart
- Added SBI steal-time accounting extension
- Added SBI PMU snapshot extension

Version 1.0.0

- Updated the version for ratification

Version 1.0-rc3

- Updated the calling convention
- Fixed a typo in PMU extension
- Added a abbreviation table

Version 1.0-rc2

- Update to RISC-V formatting
- Improved the introduction
- Removed all references to RV32

- 新增 CI 支持
- 修正 makefile 中的 revmark 标记。
- 少量细微清理工作。

版本 2.0-rc2

- 针对 SUSP、NACL 和 STA 扩展添加了说明。
- 硬件线程使用标准化
- 在 SBI DBCN 扩展中新增错误码

版本 2.0-rc1

- 为共享内存物理地址范围参数添加通用描述
- 新增了 SBI 调试控制台扩展
- 放宽了对 SBI 性能监控单元固件计数器的位宽要求
- 在 SBI 性能监控单元扩展中新增了 sbi_pmu_counter_fw_read_hi()函数
- 为 SBI 实现特定的固件事件预留了空间
- 新增了 SBI 系统休眠扩展
- 新增了 SBI CPPC 扩展
- 明确说明 SBI 扩展可以部分实现，但必须定义发现已实现 SBI 功能的机制

- 新增错误码 SBI_ERR_NO_SHMEM
- 新增 SBI 嵌套加速扩展
- 新增虚拟硬件线程的通用描述
- 新增 SBI 窃取时间统计扩展
- 新增 SBI 性能监控单元快照扩展

版本 1.0.0

- 更新版本以进行批准

版本 1.0-rc3

- 更新了调用约定
- 修正了 PMU 扩展中的一个拼写错误
- 新增了缩略词表

版本 1.0-rc2

- 更新为 RISC-V 格式规范
- 改进了引言部分
- 移除了所有关于 RV32 的引用

Version 1.0-rc1

- A typo fix

Version 0.3.0

- Few typo fixes
- Updated the LICENSE with detailed text instead of a hyperlink

Version 0.3-rc1

- Improved document styling and naming conventions
- Added SBI system reset extension
- Improved SBI introduction section
- Improved documentation of SBI hart state management extension
- Added suspend function to SBI hart state management extension
- Added performance monitoring unit extension
- Clarified that an SBI extension shall not be partially implemented

Version 0.2

- The entire v0.1 SBI has been moved to the legacy extension, which is now an optional extension. This is technically a backwards-incompatible change because the legacy extension is optional and v0.1 of the SBI doesn't allow probing, but it's as good as we can do.

版本 1.0-rc1

- 修正一处拼写错误

版本 0.3.0

- 修正若干拼写错误
- 更新许可证文本为详细条款（原为超链接形式）

版本 0.3-rc1

- 优化了文档样式与命名规范
- 新增 SBI 系统重置扩展功能
- 完善了 SBI 介绍章节
- 完善了 SBI 硬件线程状态管理扩展的文档说明
- 在 SBI 硬件线程状态管理扩展中新增了暂停功能
- 新增性能监控单元扩展
- 明确规定了 SBI 扩展不允许部分实现

版本 0.2

- 整个 v0.1 版本的 SBI 已被移至传统扩展部分，该扩展现为可选扩展。从技术上讲这是不向后兼容的变更，因为传统扩展是可选的，而 SBI 的 v0.1 版本不允许探测功能，但这是我们能做到的最佳方案。

Chapter 1. Introduction

This specification describes the RISC-V Supervisor Binary Interface, known from here on as SBI. The SBI allows supervisor-mode (S-mode or VS-mode) software to be portable across all RISC-V implementations by defining an abstraction for platform (or hypervisor) specific functionality. The design of the SBI follows the general RISC-V philosophy of having a small core along with a set of optional modular extensions.

An SBI extension defines a set of SBI functions which provides a particular functionality to supervisor-mode software. SBI extensions as a whole are optional and cannot be partially implemented unless an SBI extension defines a mechanism to discover implemented SBI functions. If sbi_probe_extension() signals that an extension is available, all functions present in the SBI version reported by sbi_get_spec_version() must conform to that version of the SBI specification.

The higher privilege software providing SBI interface to the supervisor-mode software is referred as an SBI implementation or Supervisor Execution Environment (SEE). An SBI implementation (or SEE) can be platform runtime firmware executing in machine-mode (M-mode) (see below Figure 1) or it can be some hypervisor executing in hypervisor-mode (HS-mode) (see below Figure 2).

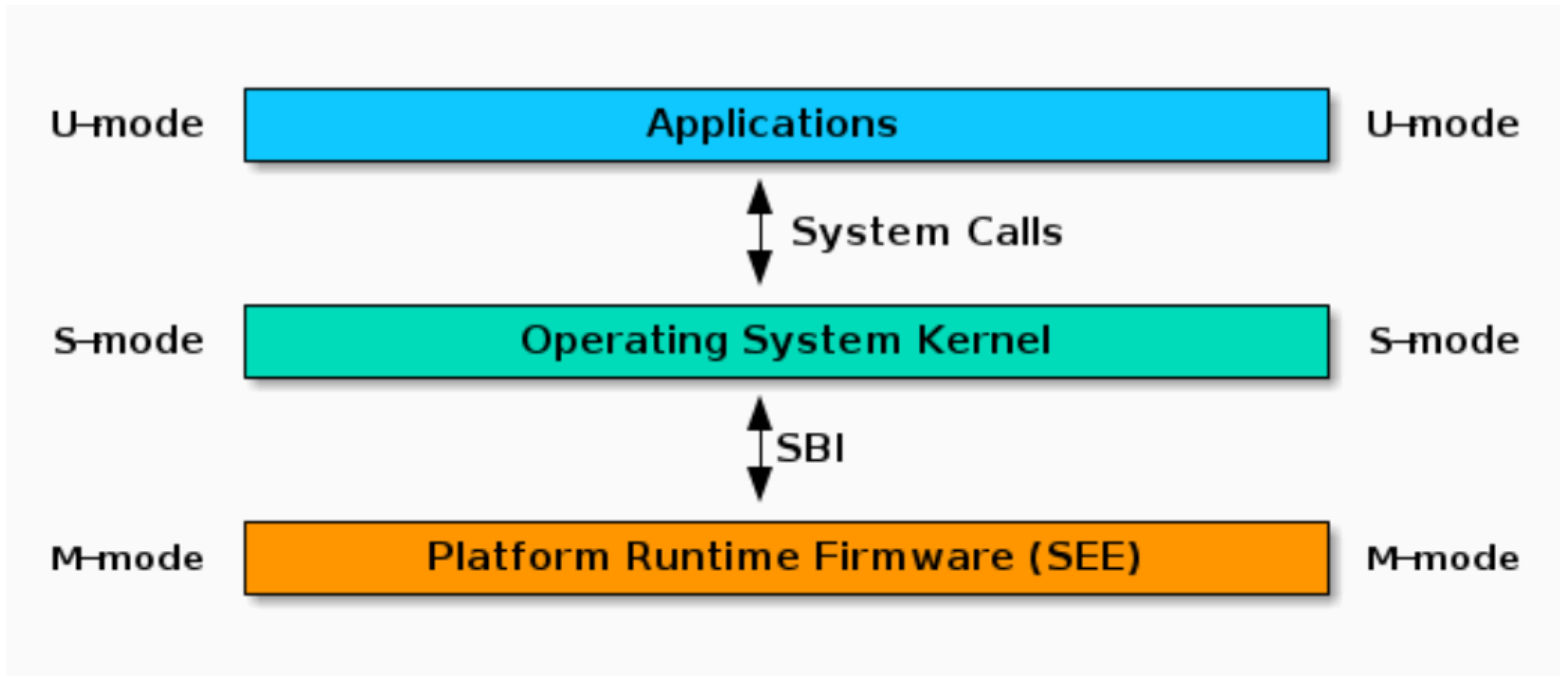


Figure 1. RISC-V System without H-extension

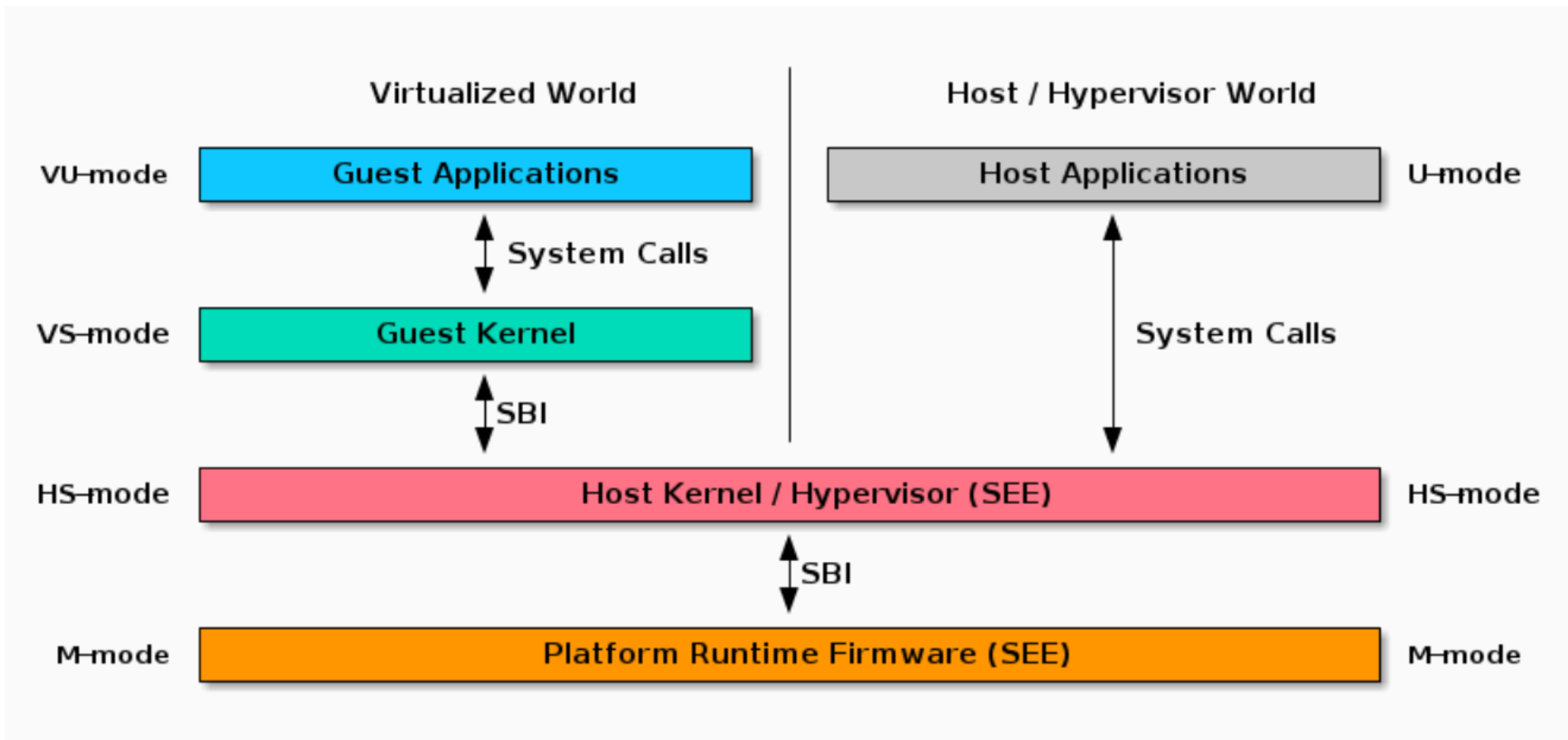


Figure 2. RISC-V System with H-extension

Harts are provisioned by the SBI implementation for supervisor-mode software. Hence, from the perspective of the SBI implementation, the S-mode hart contexts are referred to as virtual harts. In the case that the implementation is a hypervisor, virtual harts represent the VS-mode guest contexts.

第 1 章 简介

本规范描述了 RISC-V 监督者二进制接口（以下简称 SBI）。SBI 通过定义对平台（或虚拟机监控程序）特定功能的抽象，使得监督模式（S 模式或 VS 模式）软件能够在所有 RISC-V 实现中保持可移植性。SBI 的设计遵循 RISC-V 的一般理念：保持核心精简，同时提供一组可选的模块化扩展。

SBI 扩展定义了一组 SBI 功能，为监督模式软件提供特定功能。SBI 扩展整体上是可选的，除非某 SBI 扩展定义了发现已实现功能的机制，否则不能部分实现。若 sbi_probe_extension()表明某扩展可用，则 sbi_get_spec_version()报告的 SBI 版本中包含的所有功能都必须符合该版本的 SBI 规范要求。

为监督模式软件提供 SBI 接口的高特权级软件被称为 SBI 实现或监督执行环境(SEE)。SBI 实现(或 SEE)可以是在机器模式(M-mode)下运行的平台运行时固件(见下图 1)，也可以是在监督者模式(HS-mode)下运行的虚拟机监控程序(见下图 2)。

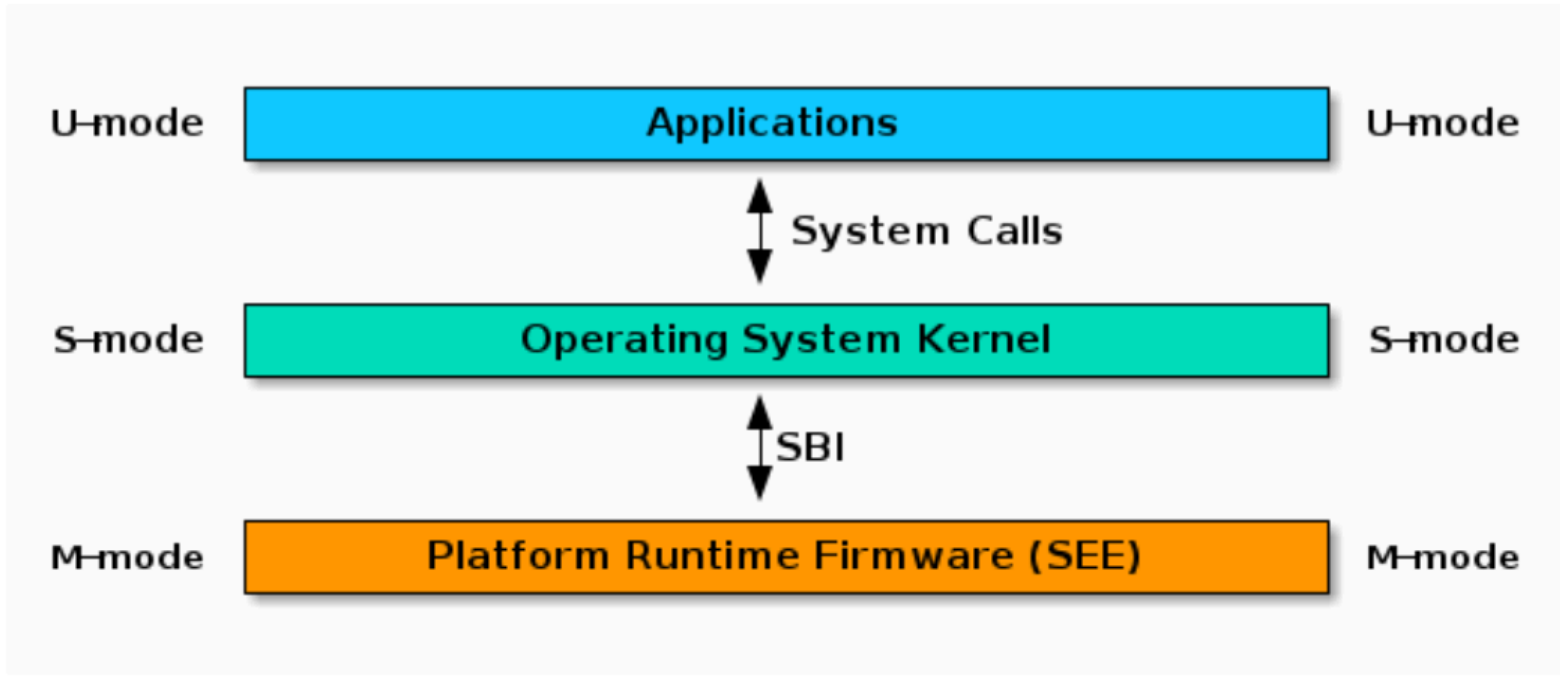


图 1. 不带 H 扩展的 RISC-V 系统

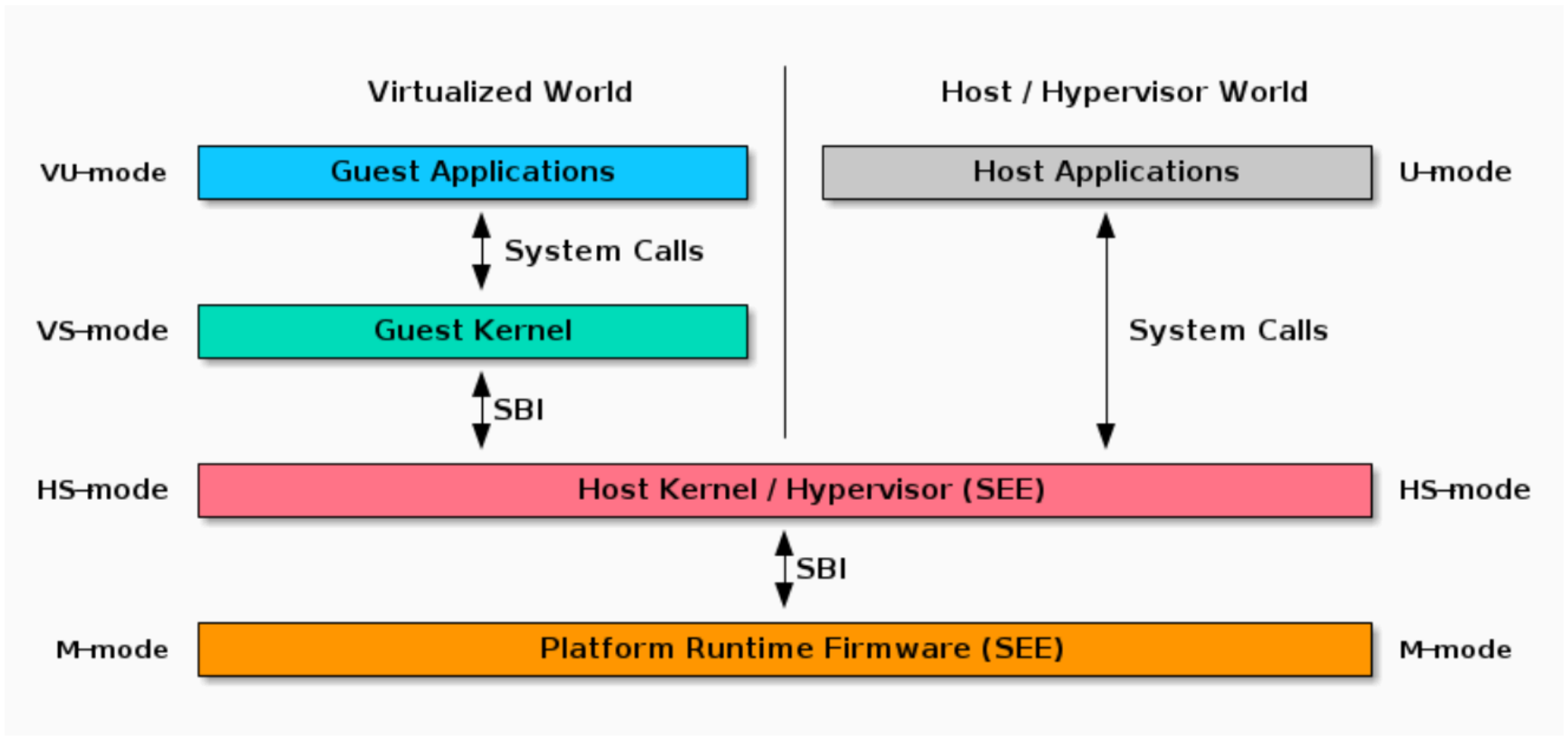


图 2. 带 H 扩展的 RISC-V 系统

SBI 实现为监督模式软件提供了硬件线程(hart)资源。因此，从 SBI 实现的角度来看，S 模式的 hart 上下文被称为虚拟 hart。当实现为虚拟机监控程序时，虚拟 hart 代表 VS 模式的客户机上下文。

The SBI specification doesn't specify any method for hardware discovery. The supervisor software must rely on the other industry standard hardware discovery methods (i.e. Device Tree or ACPI) for that.

SBI 规范并未规定任何硬件发现方法。监督模式软件必须依赖其他行业标准硬件发现方法（如设备树或 ACPI）来实现该功能。

Chapter 2. Terms and Abbreviations

This specification uses the following terms and abbreviations:

Term	Meaning
ACPI	Advanced Configuration and Power Interface
ASID	Address Space Identifier
BMC	Baseboard Management Controller
CPPC	Collaborative Processor Performance Control
EID	Extension ID
FID	Function ID
HSM	Hart State Management
IPI	Inter Processor Interrupt
PMU	Performance Monitoring Unit
SBI	Supervisor Binary Interface
SEE	Supervisor Execution Environment
VMID	Virtual Machine Identifier

第 2 章 术语与缩写

本规范使用以下术语与缩写：

术语	含义
ACPI	高级配置与电源接口
ASID	地址空间标识符
BMC	基板管理控制器
CPPC	协作处理器性能控制
EID	扩展 ID
功能标识符	函数 ID
硬件状态管理	硬件线程状态管理
处理器间中断	跨处理器中断
性能监控单元	性能监控单元
SBI	监管者二进制接口
SBI	监管执行环境
虚拟机标识符	虚拟机标识符

Chapter 3. Binary Encoding

All SBI functions share a single binary encoding, which facilitates the mixing of SBI extensions. The SBI specification follows the below calling convention.

- An `ECALL` is used as the control transfer instruction between the supervisor and the SEE.
- `a7` encodes the SBI extension ID (**EID**).
- `a6` encodes the SBI function ID (**FID**) for a given extension ID encoded in `a7` for any SBI extension defined in or after SBI v0.2.
- `a0` through `a5` contain the arguments for the SBI function call. Registers that are not defined in the SBI function call are not reserved.
- All registers except `a0` & `a1` must be preserved across an SBI call by the callee.
- SBI functions must return a pair of values in `a0` and `a1`, with `a0` returning an error code. This is analogous to returning the C structure

```
struct sbiret {
    long error;
    union {
        long value;
        unsigned long uvalue;
    };
};
```

Data type `long` in C pseudocode is XLEN bits wide.

In the name of compatibility, SBI extension IDs (**EIDs**) and SBI function IDs (**FIDs**) are encoded as signed 32-bit integers. When passed in registers these follow the standard above calling convention rules.

The Table 1 below provides a list of Standard SBI error codes.

Table 1. Standard SBI Errors

Error Type	Value	Description
SBI_SUCCESS	0	Completed successfully
SBI_ERR_FAILED	-1	Failed
SBI_ERR_NOT_SUPPORTED	-2	Not supported
SBI_ERR_INVALID_PARAM	-3	Invalid parameter(s)
SBI_ERR_DENIED	-4	Denied or not allowed
SBI_ERR_INVALID_ADDRESS	-5	Invalid address(s)
SBI_ERR_ALREADY_AVAILABLE	-6	Already available
SBI_ERR_ALREADY_STARTED	-7	Already started
SBI_ERR_ALREADY_STOPPED	-8	Already stopped
SBI_ERR_NO_SHMEM	-9	Shared memory not available
SBI_ERR_INVALID_STATE	-10	Invalid state
SBI_ERR_BAD_RANGE	-11	Bad (or invalid) range
SBI_ERR_TIMEOUT	-12	Failed due to timeout

第三章 二进制编码

所有 SBI 函数共享统一的二进制编码格式，这便于混合使用不同的 SBI 扩展。SBI 规范遵循以下调用约定。

- 使用 ECALL 指令作为监督模式执行环境(SEE)与监管者之间的控制转移指令。
- 寄存器 a7 用于编码 SBI 扩展 ID(EID)。
- 对于 SBI v0.2 及后续版本定义的任何扩展，寄存器 a6 用于编码由 a7 寄存器指定的扩展 ID 所对应的 SBI 函数 ID(FID)。
- a0 至 a5 寄存器包含 SBI 函数调用的参数。SBI 函数调用中未定义的寄存器无需保留。

- 除 a0 和 a1 外，所有寄存器在被调用方执行 SBI 调用时必须保持原值。
- SBI 函数必须通过 a0 和 a1 返回一对值，其中 a0 返回错误码。这类似于返回以下 C 结构体：

```
struct sbiret {
    长整型错误;
    联合体 {
长整型数值；无符号长整型无符号数
值； }； }；
```

C 伪代码中的长整型数据类型宽度为 XLEN 位。

出于兼容性考虑，SBI 扩展 ID（EID）和 SBI 函数 ID（FID）被编码为有符号 32 位整数。当通过寄存器传递时，这些 ID 遵循上述标准调用约定规则。

下表 1 列出了标准 SBI 错误代码清单。

表 1. 标准 SBI 错误类型

错误类型	值描述	
SBI_SUCCESS	0	成功完成
SBI_ERR_FAILED	-1	失败
SBI_ERR_NOT_SUPPORTED	-2	不支持
SBI_ERR_INVALID_PARAM	-3	无效参数
SBI_ERR_DENIED（访问被拒绝）	-4	拒绝或不允许
SBI_ERR_INVALID_ADDRESS（无效地址错误）	-5	无效地址
SBI_ERR_ALREADY_AVAILABLE	-6	已可用
SBI_ERR_ALREADY_STARTED	-7	已启动
SBI_ERR_ALREADY_STOPPED	-8	已停止
SBI_ERR_NO_SHMEM	-9	共享内存不可用
SBI_ERR_INVALID_STATE	-10	无效状态
SBI_ERR_BAD_RANGE	-11	错误（或无效）范围
SBI_ERR_TIMEOUT	-12	因超时而失败

Error Type	Value	Description
SBI_ERR_IO	-13	Input/Output error
SBI_ERR_DENIED_LOCKED	-14	Denied or not allowed due to lock status

An `ECALL` with an unsupported SBI extension ID (`EID`) or an unsupported SBI function ID (`FID`) must return the error code `SBI_ERR_NOT_SUPPORTED`.

If an SBI function call returns an error code other than `SBI_SUCCESS`, the value returned in `a1` is unspecified unless explicitly defined for that SBI function.

Every SBI function should prefer `unsigned long` as the data type. It keeps the specification simple and easily adaptable for all RISC-V ISA types. In case the data is defined as 32bit wide, higher privilege software must ensure that it only uses 32 bit data. Parameters that are 2×XLEN bits wide are passed in a pair of argument registers, with the low-order XLEN bits in the lower-numbered register and the high-order XLEN bits in the higher-numbered register.

3.1. Hart list parameter

If an SBI function caller needs to pass a list of harts to the higher privilege mode, it must use a hart mask as defined below. This is applicable to any extensions defined in or after v0.2.

Any SBI function, requiring a hart mask, must take the following two arguments:

- `unsigned long hart_mask` is a scalar bit-vector containing hartids
- `unsigned long hart_mask_base` is the starting hartid from which the bit-vector must be computed.

`hart_mask_base` *does not need to be an enabled or supervisor available hartid unless the zeroth bit of* `hart_mask` *is set.*

In a single SBI function call, the maximum number of harts that can be set is always XLEN. If a lower privilege mode needs to pass information about more than XLEN harts, it must invoke the SBI function multiple times. `hart_mask_base` can be set to `-1` to indicate that `hart_mask` shall be ignored and all available harts must be considered.

Any SBI function taking hart mask arguments may return the error values listed in the Table 2 below which are in addition to function specific error values.

Table 2. Hart Mask Errors

Error code	Description
SBI_ERR_INVALID_PARAM	At least one hartid constructed from `hart_mask_base` and `hart_mask`, is not valid, i.e. either the hartid is not enabled by the platform or is not available to the supervisor.

3.2. Shared memory physical address range parameter

If an SBI function needs to pass a shared memory physical address range to the SBI implementation (or higher privilege mode), then this physical memory address range MUST satisfy the following requirements:

- The SBI implementation MUST check that the specified physical memory range is composed of accessible physical addresses and return SBI_ERR_INVALID_ADDRESS when any address in the range is not accessible.

错误类型	值描述	
SBI_ERR_IO	-13	输入/输出错误
SBI_ERR_DENIED_LOCKED	-14	由于锁定状态被拒绝或不允许

对于包含不受支持的 SBI 扩展 ID(EID)或不受支持的 SBI 功能 ID(FID)的 ECALL 指令，必须返回错误码 SBI_ERR_NOT_SUPPORTED。
若 SBI 函数调用返回的错误码非 SBI_SUCCESS，则寄存器 a1 的返回值未作规范，除非该 SBI 函数对此有明确定义。
所有 SBI 函数应优先采用 unsigned long 作为数据类型。这能保持规范简洁性，并适配所有 RISC-V 指令集架构类型。若数据被定义为 32 位宽，高特权级软件必须确保仅使用 32 位数据。宽度为 2×XLEN 位的参数需通过一对参数寄存器传递，其中低编号寄存器存放低 XLEN 位，高编号寄存器存放高 XLEN 位。

3.1. 硬件线程列表参数

若 SBI 调用者需向更高特权级传递硬件线程列表，必须使用如下定义的硬件线程掩码。此规则适用于 v0.2 及后续版本中定义的所有扩展。

任何需要硬件线程掩码的 SBI 函数都必须接受以下两个参数：

- unsigned long hart_mask 是一个包含硬件线程 ID 的标量位向量
- unsigned long hart_mask_base 是计算位向量的起始硬件线程 ID

  ☒除非 hart_mask 的第零位被设置，否则该起始 ID 不需要是已启用或可用的监管者模式硬件线程 ID。

在一次 SBI 函数调用中，最多可设置的硬件线程数量始终为 XLEN 值。若低特权级模式需要传递超过 XLEN 数量的硬件线程信息，则必须多次调用 SBI 函数。可将 hart_mask_base 设置为-1 以表示忽略 hart_mask 参数，此时应处理所有可用硬件线程。
任何接收硬件线程掩码参数的 SBI 函数都可能返回表 2 所列的错误值，这些错误值将附加在函数特定的错误返回值之外。

表 2. 硬件线程掩码错误类型

错误代码	描述
SBI_ERR_INVALID_PARAM 当根据 hart_mask_base 和掩码构建的硬件线程 ID 中	硬件线程掩码无效，即该硬件线程 ID 未被平台启用或对监管者模式不可用。

3.2. 共享内存物理地址范围参数

若某 SBI 功能需要向 SBI 实现层（或更高特权级）传递共享内存物理地址范围，则该物理内存地址范围必须满足以下要求：

- SBI 实现层必须检查指定的物理内存范围是否由可访问的物理地址构成，当范围内任一地址不可访问时，应返回 SBI_ERR_INVALID_ADDRESS 错误码。

An accessible address is one that S-mode could reasonably expect to access per its description of the platform's physical memory layout. As an SBI implementation may further restrict the allowed range, it may return a generic SBI_ERR_FAILED (instead of SBI_ERR_INVALID_ADDRESS) when input is inaccessible with respect to its specific limits. Returning SBI_ERR_FAILED instead of SBI_ERR_INVALID_ADDRESS, in this case, is not a violation of the above specification because the SBI implementation should detect the distinct case of violating the more strict range first, making it appropriate to return the error associated with the stricter range case immediately.

- The SBI implementation MUST check that the supervisor-mode software is allowed to access the specified physical memory range with the access type requested (read and/or write).
- The SBI implementation MUST access the specified physical memory range using the PMA attributes.

If the supervisor-mode software accesses the same physical memory range using a memory type different than the PMA, then a loss of coherence or unexpected memory ordering may occur. The invoking software should follow the rules and sequences defined in the RISC-V Svpbmt specification to prevent the loss of coherence and memory ordering.

- The data in the shared memory MUST follow little-endian byte ordering.

It is recommended that a memory physical address passed to an SBI function should use at least two `unsigned long` parameters to support platforms which have memory physical addresses wider than XLEN bits.

☒ 可访问地址是指 S 模式根据其对平台物理内存布局的描述，可以合理预期访问的地址。由于 SBI 实现可能进一步限制允许的范围，当输入地址超出其特定限制而不可访问时，它可能返回通用错误码 SBI_ERR_FAILED（而非 SBI_ERR_INVALID_ADDRESS）。在这种情况下返回 SBI_ERR_FAILED 而非 SBI_ERR_INVALID_ADDRESS 并不违反上述规范，因为 SBI 实现应优先检测违反更严格范围的特殊情况，从而适合立即返回与该严格范围情况相关联的错误码。

- SBI 实现必须检查监督模式软件是否被允许以请求的访问类型（读和/或写）访问指定的物理内存范围。

- SBI 实现必须使用 PMA 属性访问指定的物理内存范围。

☒ 若监管模式软件使用与 PMA 不同的内存类型访问同一物理内存范围，则可能导致一致性丢失或意外的内存排序。调用软件应遵循 RISC-V Svpbmt 规范中定义的规则和序列，以防止一致性丢失和内存排序问题。

- 共享内存中的数据必须采用小端字节序。

建议传递给 SBI 函数的内存物理地址应至少使用两个 unsigned long 类型参数，以支持内存物理地址宽度超过 XLEN 位的平台。

Chapter 4. Base Extension (EID #0x10)

The base extension is designed to be as small as possible. As such, it only contains functionality for probing which SBI extensions are available and for querying the version of the SBI. All functions in the base extension must be supported by all SBI implementations, so there are no error returns defined.

4.1. Function: Get SBI specification version (FID #0)

```
struct sbiret sbi_get_spec_version(void);
```

Returns the current SBI specification version. This function must always succeed. The minor number of the SBI specification is encoded in the low 24 bits, with the major number encoded in the next 7 bits. Bit 31 must be 0 and is reserved for future expansion. When XLEN is greater than 32, bits 32 and above are also reserved and must be 0.

4.2. Function: Get SBI implementation ID (FID #1)

```
struct sbiret sbi_get_impl_id(void);
```

Returns the current SBI implementation ID, which is different for every SBI implementation. It is intended that this implementation ID allows software to probe for SBI implementation quirks.

4.3. Function: Get SBI implementation version (FID #2)

```
struct sbiret sbi_get_impl_version(void);
```

Returns the current SBI implementation version. The encoding of this version number is specific to the SBI implementation.

4.4. Function: Probe SBI extension (FID #3)

```
struct sbiret sbi_probe_extension(long extension_id);
```

Returns 0 if the given SBI extension ID (EID) is not available, or 1 if it is available unless defined as any other non-zero value by the implementation.

4.5. Function: Get machine vendor ID (FID #4)

```
struct sbiret sbi_get_mvendorid(void);
```

Return a value that is legal for the `mvendorid` CSR and 0 is always a legal value for this CSR.

第四章 基础扩展（EID #0x10）

基础扩展被设计得尽可能精简。因此，它仅包含探测可用 SBI 扩展和查询 SBI 版本的功能。所有基础扩展中的函数都必须被所有 SBI 实现支持，因此没有定义错误返回。

4.1. 函数：获取 SBI 规范版本（功能号#0） struct sbiret sbi_get_spec_version(void);

返回当前 SBI 规范版本。该函数必须始终执行成功。SBI 规范的次版本号编码在低 24 位中，主版本号编码在接下来的 7 位中。第 31 位必须为 0 并保留供未来扩展使用。当 XLEN 大于 32 时，第 32 位及以上位同样保留且必须为 0。

4.2. 函数：获取 SBI 实现 ID（功能号#1） struct sbiret sbi_get_impl_id(void);

返回当前 SBI 实现的唯一标识符，该标识符因不同 SBI 实现而异。此实现 ID 旨在让软件能够探测特定 SBI 实现的特殊行为。

4.3. 功能：获取 SBI 实现版本（功能号#2） struct sbiret sbi_get_impl_version(void);

返回当前 SBI 实现的版本号。该版本号的编码方式由具体 SBI 实现决定。

4.4. 功能：探测 SBI 扩展（功能号#3） struct sbiret sbi_probe_extension(long extension_id);

若给定的 SBI 扩展 ID(EID)不可用则返回 0，若可用则返回 1，除非实现中定义为其他非零值。

4.5. 功能：获取机器厂商 ID（FID #4）struct sbiret sbi_get_mvendorid(void);

返回一个对 mvendorid CSR 有效的值，0 始终是该 CSR 的有效值。

4.6. Function: Get machine architecture ID (FID #5)

```
struct sbiret sbi_get_marchid(void);
```

Return a value that is legal for the `marchid` CSR and 0 is always a legal value for this CSR.

4.7. Function: Get machine implementation ID (FID #6)

```
struct sbiret sbi_get_mimpid(void);
```

Return a value that is legal for the `mimpid` CSR and 0 is always a legal value for this CSR.

4.8. Function Listing

Table 3. Base Function List

Function Name	SBI Version	FID	EID
sbi_get_spec_version	0.2	0	0x10
sbi_get_impl_id	0.2	1	0x10
sbi_get_impl_version	0.2	2	0x10
sbi_probe_extension	0.2	3	0x10
sbi_get_mvendorid	0.2	4	0x10
sbi_get_marchid	0.2	5	0x10
sbi_get_mimpid	0.2	6	0x10

4.9. SBI Implementation IDs

Table 4. SBI Implementation IDs

Implementation ID	Name
0	Berkeley Boot Loader (BBL)
1	OpenSBI
2	Xvisor
3	KVM
4	RustSBI
5	Diosix
6	Coffer
7	Xen Project
8	PolarFire Hart Software Services
9	coreboot
10	oreboot
11	bhyve

4.6. 功能：获取机器架构 ID（功能号#5）

```
struct sbiret sbi_get_marchid(void);
```

返回一个对 marchid CSR 有效的值，0 始终是该 CSR 的合法值。

4.7. 功能：获取机器实现 ID（功能号#6） struct sbiret sbi_get_mimpid(void);

```
```

返回一个对 mimpid CSR 有效的值，0 始终是该 CSR 的合法值。

4.8. 功能列表

表 3. 基础功能列表

功能名称	SBI 版本	功能标识符	扩展标识符
sbi_get_spec_version	0.2	0	0x10
sbi_get_impl_id	0.2	1	0x10
sbi_get_impl_version	0.2	2	0x10
sbi_probe_extension	0.2	3	0x10
sbi_get_mvendorid	0.2	4	0x10
sbi_get_marchid	0.2	5	0x10
sbi_get_mimpid	0.2	6	0x10

4.9. SBI 实现标识符

表 4. SBI 实现 ID

实现 ID 名称	
0	伯克利引导加载程序(BBL)
1	OpenSBI
2	Xvisor（虚拟化监视器）
3	KVM（基于内核的虚拟机）
4	RustSBI（Rust 语言实现的 SBI 固件）
5	Diosix（轻量级虚拟化系统）
6	保险箱
7	Xen 项目
8	PolarFire 硬件线程软件服务
9	coreboot
10	oreboot
11	bhyve

Chapter 5. Legacy Extensions (EIDs #0x00 - #0x0F)

The legacy SBI extensions follow a slightly different calling convention as compared to the SBI v0.2 (or higher) specification where:

- The SBI function ID field in `a6` register is ignored because these are encoded as multiple SBI extension IDs.
- Nothing is returned in `a1` register.
- All registers except `a0` must be preserved across an SBI call by the callee.
- The value returned in `a0` register is SBI legacy extension specific.

The page and access faults taken by the SBI implementation while accessing memory on behalf of the supervisor are redirected back to the supervisor with `sepc` CSR pointing to the faulting `ECALL` instruction.

The legacy SBI extensions are deprecated in favor of the TIME, IPI, RFENCE, SRST, and DBCN extensions.

5.1. Extension: Set Timer (EID #0x00)

```
long sbi_set_timer(uint64_t stime_value)
```

Programs the clock for next event after **stime_value** time. This function also clears the pending timer interrupt bit.

If the supervisor wishes to clear the timer interrupt without scheduling the next timer event, it can either request a timer interrupt infinitely far into the future (i.e., (uint64_t)-1), or it can instead mask the timer interrupt by clearing `sie.STIE` CSR bit.

This SBI call returns 0 upon success or an implementation specific negative error code.

5.2. Extension: Console Putchar (EID #0x01)

```
long sbi_console_putchar(int ch)
```

Write data present in **ch** to debug console.

Unlike `sbi_console_getchar()`, this SBI call **will block** if there remain any pending characters to be transmitted or if the receiving terminal is not yet ready to receive the byte. However, if the console doesn't exist at all, then the character is thrown away.

This SBI call returns 0 upon success or an implementation specific negative error code.

5.3. Extension: Console Getchar (EID #0x02)

```
long sbi_console_getchar(void)
```

Read a byte from debug console.

第 5 章 传统扩展（EID #0x00 - #0x0F）

与传统 SBI v0.2（或更高版本）规范相比，这些遗留 SBI 扩展遵循稍有不同的调用约定：

- 寄存器 a6 中的 SBI 函数 ID 字段被忽略，因为这些功能被编码为多个 SBI 扩展 ID。
- 寄存器 a1 不会返回任何内容。
- 除 a0 寄存器外，所有寄存器在 SBI 调用过程中必须由被调用方保持原值不变。
- a0 寄存器返回的值是 SBI 传统扩展特有的。

SBI 实现代表监管模式访问内存时触发的页面错误和访问错误会被重定向回监管模式，此时 sepc CSR 指向引发异常的 ECALL 指令。

传统 SBI 扩展已被弃用，建议改用 TIME、IPI、RFENCE、SRST 和 DBCN 扩展。

5.1. 扩展功能：设置定时器（扩展 ID #0x00） long sbi_set_timer(uint64_t stime_value)

设置时钟在 stime_value 时间后触发下一次事件。该函数同时会清除待处理的定时器中断位。
若监管模式程序希望在不调度下次定时事件的情况下清除定时器中断，可以请求一个无限遥远的未来中断（即 (uint64_t)-1），或者通过清除 sie.STIE 控制状态寄存器位来屏蔽定时器中断。

该 SBI 调用成功时返回 0，或返回实现特定的负错误码。

5.2. 扩展功能：控制台输出字符（扩展 ID #0x01）long sbi_console_putchar(int ch)

将 ch 中存放的数据写入调试控制台。

与 sbi_console_getchar()不同，若存在待传输字符或接收终端尚未准备就绪，该 SBI 调用将进入阻塞状态。但若控制台根本不存在，则该字符会被直接丢弃。

该 SBI 调用成功时返回 0，失败时返回实现相关的负错误码。

5.3. 扩展功能：控制台获取字符（扩展 ID #0x02） long sbi_console_getchar(void)

从调试控制台读取一个字节。

The SBI call returns the byte on success, or -1 for failure.

5.4. Extension: Clear IPI (EID #0x03)

```
long sbi_clear_ipi(void)
```

Clears the pending IPIs if any. The IPI is cleared only in the hart for which this SBI call is invoked. `sbi_clear_ipi()` is deprecated because S-mode code can clear `sip.SSIP` CSR bit directly.

This SBI call returns 0 if no IPI had been pending, or an implementation specific positive value if an IPI had been pending.

5.5. Extension: Send IPI (EID #0x04)

```
long sbi_send_ipi(const unsigned long *hart_mask)
```

Send an inter-processor interrupt to all the harts defined in hart_mask. Interprocessor interrupts manifest at the receiving harts as Supervisor Software Interrupts.

hart_mask is a virtual address that points to a bit-vector of harts. The bit vector is represented as a sequence of unsigned longs whose length equals the number of harts in the system divided by the number of bits in an unsigned long, rounded up to the next integer.

This SBI call returns 0 upon success or an implementation specific negative error code.

5.6. Extension: Remote FENCE.I (EID #0x05)

```
long sbi_remote_fence_i(const unsigned long *hart_mask)
```

Instructs remote harts to execute `FENCE.I` instruction. The `hart_mask` is same as described in `sbi_send_ipi()`.

This SBI call returns 0 upon success or an implementation specific negative error code.

5.7. Extension: Remote SFENCE.VMA (EID #0x06)

```
long sbi_remote_sfence_vma(const unsigned long *hart_mask,
                           unsigned long start,
                           unsigned long size)
```

Instructs the remote harts to execute one or more `SFENCE.VMA` instructions, covering the range of virtual addresses between `start` and `start + size`.

The remote fence operation applies to the entire address space if either:

SBI 调用成功时返回字节值，失败时返回-1。

5.4. 扩展功能：清除 IPI（扩展 ID #0x03） long sbi_clear_ipi(void)

清除所有待处理的处理器间中断（IPI）。该 IPI 仅在此 SBI 调用所针对的硬件线程（hart）中被清除。
sbi_clear_ipi() 已被弃用，因为 S 模式代码可以直接清除 sip.SSIP CSR 位。

若没有待处理的 IPI，该 SBI 调用返回 0；若有待处理的 IPI，则返回实现特定的正值。

5.5. 扩展功能：发送处理器间中断（EID #0x04）
long sbi_send_ipi(const unsigned long *hart_mask)

向 hart_mask 中定义的所有硬件线程发送处理器间中断。处理器间中断在接收硬件线程上表现为监管模式软件中断。
hart_mask 是一个指向硬件线程位向量的虚拟地址。该位向量由无符号长整型序列表示，其长度等于系统中硬件线程数量除以一个无符号长整型的位数，再向上取整。

该 SBI 调用成功时返回 0，或返回实现特定的负错误码。

5.6. 扩展功能：远程 FENCE.I 指令（扩展 ID #0x05） long sbi_remote_fence_i(const unsigned long *hart_mask)

指示远程硬件线程执行 FENCE.I 指令。hart_mask 参数与
sbi_send_ipi()中的描述相同

该 SBI 调用成功时返回 0，或返回实现特定的负错误码。

5.7. 扩展功能：远程 SFENCE.VMA（扩展 ID #0x06） long sbi_remote_sfence_vma(const unsigned long *hart_mask,

```
                                unsigned long start,
                                unsigned long size)
```

指示远程硬件线程执行一条或多条 SFENCE.VMA 指令，覆盖从 start 到 start + size 范围内的虚拟地址。

远程栅栏操作在以下任一情况下适用于整个地址空间：

- `start` and `size` are both 0, or
- `size` is equal to 2^XLEN-1.

This SBI call returns 0 upon success or an implementation specific negative error code.

5.8. Extension: Remote SFENCE.VMA with ASID (EID #0x07)

```
long sbi_remote_sfence_vma_asid(const unsigned long *hart_mask,
                                unsigned long start,
                                unsigned long size,
                                unsigned long asid)
```

Instruct the remote harts to execute one or more `SFENCE.VMA` instructions, covering the range of virtual addresses between `start` and `start + size`. This covers only the given `ASID`.

The remote fence operation applies to the entire address space if either:

- `start` and `size` are both 0, or
- `size` is equal to 2^XLEN-1.

This SBI call returns 0 upon success or an implementation specific negative error code.

5.9. Extension: System Shutdown (EID #0x08)

```
void sbi_shutdown(void)
```

Puts all the harts to shutdown state from supervisor point of view.

This SBI call doesn't return irrespective whether it succeeds or fails.

5.10. Function Listing

Table 5. Legacy Function List

Function Name	SBI Version	FID	EID	Replacement EID
sbi_set_timer	0.1	0	0x00	0x54494D45
sbi_console_putchar	0.1	0	0x01	0x4442434E
sbi_console_getchar	0.1	0	0x02	0x4442434E
sbi_clear_ipi	0.1	0	0x03	N/A
sbi_send_ipi	0.1	0	0x04	0x735049
sbi_remote_fence_i	0.1	0	0x05	0x52464E43
sbi_remote_sfence_vma	0.1	0	0x06	0x52464E43
sbi_remote_sfence_vma_asid	0.1	0	0x07	0x52464E43
sbi_shutdown	0.1	0	0x08	0x53525354
RESERVED			0x09-0x0F	

- start 和 size 均为 0，或
- size 等于 2^XLEN-1。

该 SBI 调用成功时返回 0，或返回实现特定的负错误码。

5.8. 扩展功能：带 ASID 的远程 SFENCE.VMA（扩展 ID #0x07） long sbi_remote_sfence_vma_asid(const unsigned long *hart_mask,

```
                                unsigned long start,
                                unsigned long size,
                                unsigned long asid)
```

指示远程硬件线程执行一条或多条 SFENCE.VMA 指令，覆盖从 start 到 start + size 范围内的虚拟地址。此操作仅针对给定的地址空间标识符(ASID)。

当满足以下任一条件时，远程栅栏操作将作用于整个地址空间：

- start 和 size 均为 0，或
- size 等于 2^XLEN-1。

该 SBI 调用成功时返回 0，失败时返回实现特定的负错误码。

5.9. 扩展功能：系统关机（EID #0x08） void sbi_shutdown(void)

从监督者模式视角将所有硬件线程置于关机状态。

无论成功与否，该 SBI 调用都不会返回。

5.10. 函数列表

表 5. 传统函数列表

功能名称	SBI 版本	函数 ID 扩展 ID		替代扩展 ID
设置定时器	0.1	0	0x00	0x54494D45
控制台输出字符	0.1	0	0x01	0x4442434E
sbi_console_getchar	0.1	0	0x02	0x4442434E
sbi_clear_ipi	0.1	0	0x03	不适用
发送处理器间中断	0.1	0	0x04	0x735049
sbi_remote_fence_i	0.1	0	0x05	0x52464E43
sbi_远程虚拟内存屏障指令	0.1	0	0x06	0x52464E43
sbi_remote_sfence_vma_asid 0.1		0	0x07	0x52464E43
sbi_shutdown	0.1	0	0x08	0x53525354
保留字段			0x09-0x0F	

Chapter 6. Timer Extension (EID #0x54494D45 "TIME")

This replaces legacy timer extension (EID #0x00). It follows the new calling convention defined in v0.2.

6.1. Function: Set Timer (FID #0)

```
struct sbiret sbi_set_timer(uint64_t stime_value)
```

Programs the clock for next event after **stime_value** time. **stime_value** is in absolute time.

If the supervisor wishes to clear the timer interrupt without scheduling the next timer event, it may request a timer interrupt infinitely far into the future (i.e., (uint64_t)-1). Alternatively, to not receive timer interrupts, it may mask timer interrupts by clearing the `sie.STIE` CSR bit.

This function must clear the pending timer interrupt bit when **stime_value** is set to some time in the future, regardless of whether timer interrupts are masked or not.

This function always returns SBI_SUCCESS in `sbiret.error`.

6.2. Function Listing

Table 6. TIME Function List

Function Name	SBI Version	FID	EID
sbi_set_timer	0.2	0	0x54494D45

第六章 定时器扩展（扩展 ID #0x54494D45 "TIME"）

该规范取代了传统的定时器扩展（EID #0x00），并遵循 v0.2 版本定义的新调用约定。

6.1. 功能函数：设置定时器（FID #0）struct sbiret sbi_set_timer(uint64_t stime_value)

```
```

将时钟编程为在 stime_value 绝对时间后触发下一个事件。

若监管模式程序希望在不调度下次定时器事件的情况下清除定时器中断，可请求将定时器中断设置为无限远的未来时刻（即(uint64_t)-1）。或者，通过清除 sie.STIE CSR 位来屏蔽定时器中断，从而避免接收定时器中断。
该函数必须在将 stime_value 设置为未来某个时间时清除待处理的定时器中断位，无论定时器中断是否被屏蔽。

该函数在 sbiret.error 中始终返回 SBI_SUCCESS。

6.2. 函数列表

表 6. TIME 函数列表

功能名称	SBI 版本	功能标识符	扩展标识符
sbi_set_timer	0.2	0	0x54494D45

Chapter 7. IPI Extension (EID #0x735049 "sPI: s-mode IPI")

This extension replaces the legacy extension (EID #0x04). The other IPI related legacy extension(0x3) is deprecated now. All the functions in this extension follow the `hart_mask` as defined in the binary encoding section.

7.1. Function: Send IPI (FID #0)

```
struct sbiret sbi_send_ipi(unsigned long hart_mask,
                           unsigned long hart_mask_base)
```

Send an inter-processor interrupt to all the harts defined in hart_mask. Interprocessor interrupts manifest at the receiving harts as the supervisor software interrupts.

The possible error codes returned in `sbiret.error` are shown in the Table 7 below.

Table 7. IPI Send Errors

Error code	Description
SBI_SUCCESS	IPI was sent to all the targeted harts successfully.
SBI_ERR_INVALID_PARAM	At least one hartid constructed from `hart_mask_base` and `hart_mask`, is not valid, i.e. either the hartid is not enabled by the platform or is not available to the supervisor.
SBI_ERR_FAILED	The request failed for unspecified or unknown other reasons.

7.2. Function Listing

Table 8. IPI Function List

Function Name	SBI Version	FID	EID
sbi_send_ipi	0.2	0	0x735049

第七章 IPI 扩展（扩展 ID #0x735049 "sPI：s 模式 IPI"）

本扩展替代了旧版扩展（扩展 ID #0x04）。其他与 IPI 相关的旧版扩展（0x3）现已弃用。本扩展中的所有功能均遵循二进制编码章节中定义的 hart_mask 规范。

7.1. 功能：发送处理器间中断（功能 ID #0） struct sbiret sbi_send_ipi(unsigned long hart_mask,

```
unsigned long hart_mask_base)
```

向 hart_mask 定义的所有硬件线程发送处理器间中断。该中断在接收硬件线程上表现为监管模式软件中断。

sbiret.error 可能返回的错误代码如下表 7 所示。

表 7. IPI 发送错误

错误码	描述
SBI_SUCCESS	IPI 已成功发送至所有目标硬件线程。
SBI_ERR_INVALID_PARAM	由 hart_mask_base 和 hart_mask 构建的至少一个 hartid 无效，即该 hartid 未被平台启用或对监管者不可用。
SBI_ERR_FAILED	请求因未指定或其他未知原因失败。

7.2. 功能列表

表 8. IPI 功能列表

功能名称	SBI 版本	功能标识符	扩展标识符
发送处理器间中断	0.2	0	0x735049

Chapter 8. RFENCE Extension (EID #0x52464E43 "RFNC")

This extension defines all remote fence related functions and replaces the legacy extensions (EIDs #0x05 - #0x07). All the functions follow the `hart_mask` as defined in binary encoding section. Any function which accepts a range of addresses (i.e. `start_addr` and `size`) must abide by the below constraints on range parameters.

The remote fence operation applies to the entire address space if either:

- `start_addr` and `size` are both 0, or
- `size` is equal to 2^XLEN-1.

8.1. Function: Remote FENCE.I (FID #0)

```
struct sbiret sbi_remote_fence_i(unsigned long hart_mask,
                                 unsigned long hart_mask_base)
```

Instructs remote harts to execute `FENCE.I` instruction.

The possible error codes returned in `sbiret.error` are shown in the Table 9 below.

Table 9. RFENCE Remote FENCE.I Errors

Error code	Description
SBI_SUCCESS	IPI was sent to all the targeted harts successfully.
SBI_ERR_INVALID_PARAM	At least one hartid constructed from `hart_mask_base` and `hart_mask`, is not valid, i.e. either the hartid is not enabled by the platform or is not available to the supervisor.
SBI_ERR_FAILED	The request failed for unspecified or unknown other reasons.

8.2. Function: Remote SFENCE.VMA (FID #1)

```
struct sbiret sbi_remote_sfence_vma(unsigned long hart_mask,
                                    unsigned long hart_mask_base,
                                    unsigned long start_addr,
                                    unsigned long size)
```

Instructs the remote harts to execute one or more `SFENCE.VMA` instructions, covering the range of virtual addresses between `start_addr` and `start_addr + size`.

The possible error codes returned in `sbiret.error` are shown in the Table 10 below.

Table 10. RFENCE Remote SFENCE.VMA Errors

Error code	Description
SBI_SUCCESS	IPI was sent to all the targeted harts successfully.
SBI_ERR_INVALID_ADDRESS	`start_addr` or `size` is not valid.

第八章 RFENCE 扩展（扩展 ID #0x52464E43 "RFNC"）

本扩展定义了所有远程栅栏相关功能，并取代了旧版扩展（扩展 ID #0x05 和#0x07）。所有功能都遵循二进制编码章节中定义的 hart_mask 参数。任何接受地址范围（即起始地址和大小）的函数都必须遵守以下关于范围参数的约束条件。

当满足以下任一条件时，远程栅栏操作将作用于整个地址空间：

- 起始地址和大小均为 0，或
- size 等于 2^XLEN-1。

8.1. 功能：远程 FENCE.I（功能号#0） struct sbiret sbi_remote_fence_i(unsigned long hart_mask,

```
                                                    unsigned long hart_mask_base)
```

指示远程 hart 执行 FENCE.I 指令。

sbiret.error 可能返回的错误代码如下表 9 所示。

表 9. RFENCE 远程 FENCE.I 错误

错误代码	描述
SBI_SUCCESS	IPI 已成功发送至所有目标硬件线程。
SBI_ERR_INVALID_PARAM	由 hart_mask_base 和 hart_mask 构建的至少一个 hartid 无效，即该 hartid 未被平台启用或对监管者不可用。
SBI_ERR_FAILED	请求因未指定或其他未知原因失败。

8.2. 功能：远程 SFENCE.VMA（功能 ID #1） struct sbiret sbi_remote_sfence_vma(unsigned long hart_mask,

```
                                                    unsigned long hart_mask_base, 无符号长整型 hart_mask_base，无符号长整型 start_addr，无符号长整型 size
```

指示远程硬件线程执行一条或多条 SFENCE.VMA 指令，覆盖指定的虚拟地址范围
起始地址与起始地址加大小之间的地址范围。

表 10 列出了 sbiret.error 可能返回的错误代码。

表 10. RFENCE 远程 SFENCE.VMA 错误类型

错误代码	描述
SBI_SUCCESS	IPI 已成功发送至所有目标硬件线程。
SBI_ERR_INVALID_ADDRESS	起始地址或大小无效

Error code	Description
SBI_ERR_INVALID_PARAM	At least one hartid constructed from `hart_mask_base` and `hart_mask`, is not valid, i.e. either the hartid is not enabled by the platform or is not available to the supervisor.
SBI_ERR_FAILED	The request failed for unspecified or unknown other reasons.

8.3. Function: Remote SFENCE.VMA with ASID (FID #2)

```
struct sbiret sbi_remote_sfence_vma_asid(unsigned long hart_mask,
                                         unsigned long hart_mask_base,
                                         unsigned long start_addr,
                                         unsigned long size,
                                         unsigned long asid)
```

Instruct the remote harts to execute one or more `SFENCE.VMA` instructions, covering the range of virtual addresses between `start_addr` and `start_addr + size`. This covers only the given `ASID`.

The possible error codes returned in `sbiret.error` are shown in the Table 11 below.

Table 11. RFENCE Remote SFENCE.VMA with ASID Errors

Error code	Description
SBI_SUCCESS	IPI was sent to all the targeted harts successfully.
SBI_ERR_INVALID_ADDRESS	`start_addr` or `size` is not valid.
SBI_ERR_INVALID_PARAM	Either `asid`, or at least one hartid constructed from `hart_mask_base` and `hart_mask`, is not valid, i.e. either the hartid is not enabled by the platform or is not available to the supervisor.
SBI_ERR_FAILED	The request failed for unspecified or unknown other reasons.

8.4. Function: Remote HFENCE.GVMA with VMID (FID #3)

```
struct sbiret sbi_remote_hfence_gvma_vmid(unsigned long hart_mask,
                                          unsigned long hart_mask_base,
                                          unsigned long start_addr,
                                          unsigned long size,
                                          unsigned long vmid)
```

Instruct the remote harts to execute one or more `HFENCE.GVMA` instructions, covering the range of guest physical addresses between `start_addr` and `start_addr + size` only for the given `VMID`. This function call is only valid for harts implementing hypervisor extension.

The possible error codes returned in `sbiret.error` are shown in the Table 12 below.

Table 12. RFENCE Remote HFENCE.GVMA with VMID Errors

错误代码	描述
SBI_ERR_INVALID_PARAM	由 hart_mask_base 和 hart_mask 构建的至少一个 hartid 无效，即该 hartid 未被平台启用或对监管者不可用。
SBI_ERR_FAILED	请求因未指定或其他未知原因失败。

8.3. 功能：带 ASID 的远程 SFENCE.VMA 操作（功能号#2） struct sbiret sbi_remote_sfence_vma_asid(unsigned long hart_mask,

```
                                                    unsigned long hart_mask_base, unsigned
                                                    long start_addr, unsigned long size,
                                                    unsigned long asid)
```

指示远程硬件线程执行一条或多条 SFENCE.VMA 指令，覆盖从 start_addr 到 start_addr + size 范围内的虚拟地址。该操作仅针对给定的 ASID 生效。

sbiret.error 可能返回的错误代码如下表 11 所示。

表 11. RFENCE 远程 SFENCE.VMA 带 ASID 的错误类型

错误代码	描述
SBI_SUCCESS	IPI 已成功发送至所有目标硬件线程。
SBI_ERR_INVALID_ADDRESS	起始地址或大小无效。
SBI_ERR_INVALID_PARAM	ASID 无效，或由 hart_mask_base 和 hart_mask 构成的至少一个 hartid 无效（即该 hartid 未被平台启用或监管者无权访问）。
SBI_ERR_FAILED	请求因未指定或其他未知原因失败。

8.4. 功能：带 VMID 的远程 HFENCE.GVMA（功能号#3） struct sbiret sbi_remote_hfence_gvma_vmid(unsigned long hart_mask,

```
                                                    无符号长整型 hart_mask_base，无符号长整
                                                    型 start_addr，无符号长整型 size，无符号
                                                    长整型 vmid)
```

指示远程硬件线程执行一条或多条 HFENCE.GVMA 指令，仅针对给定的虚拟机标识符（VMID），覆盖从 start_addr 到 start_addr + size 范围内的客户机物理地址空间。该函数调用仅对实现了虚拟机监控程序扩展的硬件线程有效。

可能通过 sbiret.error 返回的错误代码如下表 12 所示。

表 12. 带 VMID 的远程 HFENCE.GVMA 栅栏操作错误代码

Error code	Description
SBI_SUCCESS	IPI was sent to all the targeted harts successfully.
SBI_ERR_NOT_SUPPORTED	This function is not supported as it is not implemented or one of the target hart doesn't support hypervisor extension.
SBI_ERR_INVALID_ADDRESS	**`start_addr`** or **`size`** is not valid.
SBI_ERR_INVALID_PARAM	Either **`vmid`**, or at least one hartid constructed from **`hart_mask_base`** and **`hart_mask`**, is not valid, i.e. either the hartid is not enabled by the platform or is not available to the supervisor.
SBI_ERR_FAILED	The request failed for unspecified or unknown other reasons.

8.5. Function: Remote HFENCE.GVMA (FID #4)

```
struct sbiret sbi_remote_hfence_gvma(unsigned long hart_mask,
                                     unsigned long hart_mask_base,
                                     unsigned long start_addr,
                                     unsigned long size)
```

Instruct the remote harts to execute one or more **`HFENCE.GVMA`** instructions, covering the range of guest physical addresses between **`start_addr`** and **`start_addr + size`** for all the guests. This function call is only valid for harts implementing hypervisor extension.

The possible error codes returned in **`sbiret.error`** are shown in the Table 13 below.

Table 13. RFENCE Remote HFENCE.GVMA Errors

Error code	Description
SBI_SUCCESS	IPI was sent to all the targeted harts successfully.
SBI_ERR_NOT_SUPPORTED	This function is not supported as it is not implemented or one of the target hart doesn't support hypervisor extension.
SBI_ERR_INVALID_ADDRESS	**`start_addr`** or **`size`** is not valid.
SBI_ERR_INVALID_PARAM	At least one hartid constructed from **`hart_mask_base`** and **`hart_mask`**, is not valid, i.e. either the hartid is not enabled by the platform or is not available to the supervisor.
SBI_ERR_FAILED	The request failed for unspecified or unknown other reasons.

8.6. Function: Remote HFENCE.VVMA with ASID (FID #5)

```
struct sbiret sbi_remote_hfence_vvma_asid(unsigned long hart_mask,
                                          unsigned long hart_mask_base,
                                          unsigned long start_addr,
                                          unsigned long size,
                                          unsigned long asid)
```

Instruct the remote harts to execute one or more **`HFENCE.VVMA`** instructions, covering the range of guest virtual addresses between **`start_addr`** and **`start_addr + size`** for the given **`ASID`** and current **`VMID`** (in

错误代码	描述
SBI_SUCCESS	IPI 已成功发送至所有目标硬件线程。
SBI_ERR_NOT_SUPPORTED	该功能不被支持，原因是未实现或目标硬件线程之一不支持虚拟机监控扩展。
SBI_ERR_INVALID_ADDRESS	起始地址或大小无效。
SBI_ERR_INVALID_PARAM	vmid 无效，或基于 hart_mask_base 和 hart_mask 构建的至少一个 hartid 无效，即该 hartid 未被平台启用或监管者无权访问。
SBI_ERR_FAILED	请求因未指定或其他未知原因失败。

8.5. 功能：远程 HFENCE.GVMA（功能号#4） struct sbiret sbi_remote_hfence_gvma(unsigned long hart_mask,

```
                                                    unsigned long hart_mask_base, unsigned
                                                    long start_addr, unsigned long size)
```

指示远程硬件线程执行一条或多条 HFENCE.GVMA 指令，覆盖所有虚拟机从 start_addr 到 start_addr + size 范围内的客户机物理地址。该功能调用仅对实现了虚拟机监控器扩展的硬件线程有效。

sbiret.error 中可能返回的错误代码如下表 13 所示。

表 13. RFENCE 远程 HFENCE.GVMA 错误

错误代码	描述
SBI_SUCCESS	IPI 已成功发送至所有目标硬件线程。
SBI_ERR_NOT_SUPPORTED	该功能未实现或目标硬件线程之一不支持虚拟机监控扩展，故不予支持。
SBI_ERR_INVALID_ADDRESS	起始地址或大小无效。
SBI_ERR_INVALID_PARAM	由 hart_mask_base 和 hart_mask 构建的至少一个 hartid 无效，即该 hartid 未被平台启用或对监管者不可用。
SBI_ERR_FAILED	请求因未指定或其他未知原因失败。

8.6. 功能：带 ASID 的远程 HFENCE.VVMA（功能号#5）
struct sbiret sbi_remote_hfence_vvma_asid(unsigned long hart_mask,

```
                                                    无符号长整型 hart_mask_base, 无符号长整
                                                    型 start_addr, 无符号长整型 size, 无符号
                                                    长整型 asid)
```

指示远程硬件线程执行一条或多条 HFENCE.VVMA 指令，覆盖给定 ASID 和当前 VMID 下从 start_addr 到 start_addr + size 范围内的客户机虚拟地址

`hgatp` CSR) of calling hart. This function call is only valid for harts implementing hypervisor extension.

The possible error codes returned in `sbiret.error` are shown in the Table 14 below.

Table 14. RFENCE Remote HFENCE.VVMA with ASID Errors

Error code	Description
SBI_SUCCESS	IPI was sent to all the targeted harts successfully.
SBI_ERR_NOT_SUPPORTED	This function is not supported as it is not implemented or one of the target hart doesn't support hypervisor extension.
SBI_ERR_INVALID_ADDRESS	`start_addr` or `size` is not valid.
SBI_ERR_INVALID_PARAM	Either `asid`, or at least one hartid constructed from `hart_mask_base` and `hart_mask`, is not valid, i.e. either the hartid is not enabled by the platform or is not available to the supervisor.
SBI_ERR_FAILED	The request failed for unspecified or unknown other reasons.

8.7. Function: Remote HFENCE.VVMA (FID #6)

```
struct sbiret sbi_remote_hfence_vvma(unsigned long hart_mask,
                                     unsigned long hart_mask_base,
                                     unsigned long start_addr,
                                     unsigned long size)
```

Instruct the remote harts to execute one or more `HFENCE.VVMA` instructions, covering the range of guest virtual addresses between `start_addr` and `start_addr + size` for current `VMID` (in `hgatp` CSR) of calling hart. This function call is only valid for harts implementing hypervisor extension.

The possible error codes returned in `sbiret.error` are shown in the Table 15 below.

Table 15. RFENCE Remote HFENCE.VVMA Errors

Error code	Description
SBI_SUCCESS	IPI was sent to all the targeted harts successfully.
SBI_ERR_NOT_SUPPORTED	This function is not supported as it is not implemented or one of the target hart doesn't support hypervisor extension.
SBI_ERR_INVALID_ADDRESS	`start_addr` or `size` is not valid.
SBI_ERR_INVALID_PARAM	At least one hartid constructed from `hart_mask_base` and `hart_mask`, is not valid, i.e. either the hartid is not enabled by the platform or is not available to the supervisor.
SBI_ERR_FAILED	The request failed for unspecified or unknown other reasons.

8.8. Function Listing

Table 16. RFENCE Function List

Function Name	SBI Version	FID	EID
sbi_remote_fence_i	0.2	0	0x52464E43
sbi_remote_sfence_vma	0.2	1	0x52464E43

调用 hart 的 hgatp CSR（控制状态寄存器）。此函数调用仅对实现了 hypervisor 扩展的 hart 有效。

sbiret.error 可能返回的错误代码如下表 14 所示。

表 14. 带 ASID 错误的 RFENCE 远程 HFENCE.VVMA

错误代码	描述
SBI_SUCCESS	IPI 已成功发送至所有目标硬件线程。
SBI_ERR_NOT_SUPPORTED	该功能不受支持，原因是未实现或目标硬件线程之一不支持虚拟机监控程序扩展。
SBI_ERR_INVALID_ADDRESS	起始地址或大小无效。
SBI_ERR_INVALID_PARAM	ASID 无效，或由 hart_mask_base 和 hart_mask 构建的至少一个 hartid 无效，即该 hartid 未被平台启用或监管者不可用。
SBI_ERR_FAILED	请求因未指定或其他未知原因失败。

8.7. 功能：远程 HFENCE.VVMA（功能号 #6） struct sbiret sbi_remote_hfence_vvma(unsigned long hart_mask,

```
                                                unsigned long hart_mask_base, unsigned
                                                long start_addr, unsigned long size)
```

指示远程硬件线程执行一条或多条 HFENCE.VVMA 指令，覆盖调用硬件线程当前 VMID（位于 hgatp CSR 中）对应的从 start_addr 到 start_addr + size 范围内的客户机虚拟地址。该函数调用仅对实现了虚拟机监控程序扩展的硬件线程有效。

sbiret.error 可能返回的错误代码如下表 15 所示。

表 15. RFENCE 远程 HFENCE.VVMA 错误

错误代码	描述
SBI_SUCCESS	IPI 已成功发送至所有目标硬件线程。
SBI_ERR_NOT_SUPPORTED	该功能不被支持，原因可能是未实现或目标硬件线程之一不支持虚拟机监控程序扩展。
SBI_ERR_INVALID_ADDRESS	起始地址或大小无效。
SBI_ERR_INVALID_PARAM	由 hart_mask_base 和 hart_mask 构建的至少一个 hartid 无效，即该 hartid 未被平台启用或对监管者不可用。
SBI_ERR_FAILED	请求因未指定或其他未知原因失败。

8.8. 函数列表

表 16. RFENCE 函数列表

功能名称	SBI 版本	功能标识符	扩展标识符
远程指令缓存栅栏指令	0.2	0	0x52464E43
远程虚拟内存栅栏指令	0.2	1	0x52464E43

Function Name	SBI Version	FID	EID
sbi_remote_sfence_vma_asid	0.2	2	0x52464E43
sbi_remote_hfence_gvma_vmid	0.2	3	0x52464E43
sbi_remote_hfence_gvma	0.2	4	0x52464E43
sbi_remote_hfence_vvma_asid	0.2	5	0x52464E43
sbi_remote_hfence_vvma	0.2	6	0x52464E43

功能名称	SBI 版本	功能标识符	扩展标识符
sbi_remote_sfence_vma_asid	0.2	2	0x52464E43
sbi_remote_hfence_gvma_vmid	0.2	3	0x52464E43
sbi_remote_hfence_gvma	0.2	4	0x52464E43
sbi_remote_hfence_vvma_asid	0.2	5	0x52464E43
sbi_远程硬件栅栏虚拟虚拟内存地址	0.2	6	0x52464E43

Chapter 9. Hart State Management Extension (EID #0x48534D "HSM")

The Hart State Management (HSM) Extension introduces a set of hart states and a set of functions which allow the supervisor-mode software to request a hart state change.

The Table 17 shown below describes all possible **HSM states** along with a unique **HSM state id** for each state:

Table 17. HSM Hart States

State ID	State Name	Description
0	STARTED	The hart is physically powered-up and executing normally.
1	STOPPED	The hart is not executing in supervisor-mode or any lower privilege mode. It is probably powered-down by the SBI implementation if the underlying platform has a mechanism to physically power-down harts.
2	START_PENDING	Some other hart has requested to start (or power-up) the hart from the **STOPPED** state and the SBI implementation is still working to get the hart in the **STARTED** state.
3	STOP_PENDING	The hart has requested to stop (or power-down) itself from the **STARTED** state and the SBI implementation is still working to get the hart in the **STOPPED** state.
4	SUSPENDED	This hart is in a platform specific suspend (or low power) state.
5	SUSPEND_PENDING	The hart has requested to put itself in a platform specific low power state from the **STARTED** state and the SBI implementation is still working to get the hart in the platform specific **SUSPENDED** state.
6	RESUME_PENDING	An interrupt or platform specific hardware event has caused the hart to resume normal execution from the **SUSPENDED** state and the SBI implementation is still working to get the hart in the **STARTED** state.

At any point in time, a hart should be in one of the above mentioned hart states. The hart state transitions by the SBI implementation should follow the state machine shown below in the Figure 3.

第 9 章 硬件线程状态管理扩展（EID #0x48534D "HSM"）

硬件线程状态管理（HSM）扩展引入了一组硬件线程状态及功能函数，使监管模式软件能够请求硬件线程状态变更。
下表 17 描述了所有可能的 HSM 状态，并为每个状态分配了唯一的 HSM 状态标识符：

表 17. HSM 硬件线程状态

状态 ID	状态名称	描述
0	已启动	硬件线程已物理上电并正常执行中
1	已停止	该硬件线程未处于监管模式或任何更低权限模式下运行。若底层平台具备物理关闭硬件线程的机制，该线程很可能已被 SBI 实现断电。
2	启动待处理	其他硬件线程已请求从停止状态启动（或上电）该硬件线程，且 SBI 实现仍在努力使其进入已启动状态。
3	停止待处理	该硬件线程已从 STARTED 状态请求停止（或下电），而 SBI 实现仍在努力使其进入 STOPPED 状态。
4	挂起中	该硬件线程处于平台特定的挂起（或低功耗）状态。
5	挂起待处理	该硬件线程已请求从 STARTED 状态进入平台特定的低功耗状态，而 SBI 实现仍在努力使其进入平台特定的 SUSPENDED 状态。
6	RESUME_PENDING	中断或平台特定的硬件事件导致该硬件线程从 SUSPENDED 状态恢复常规执行，而 SBI 实现仍在努力使其进入 STARTED 状态。

在任何时间点，硬件线程都应处于上述状态之一。SBI 实现引发的硬件线程状态转换应遵循图 3 所示的状态机。

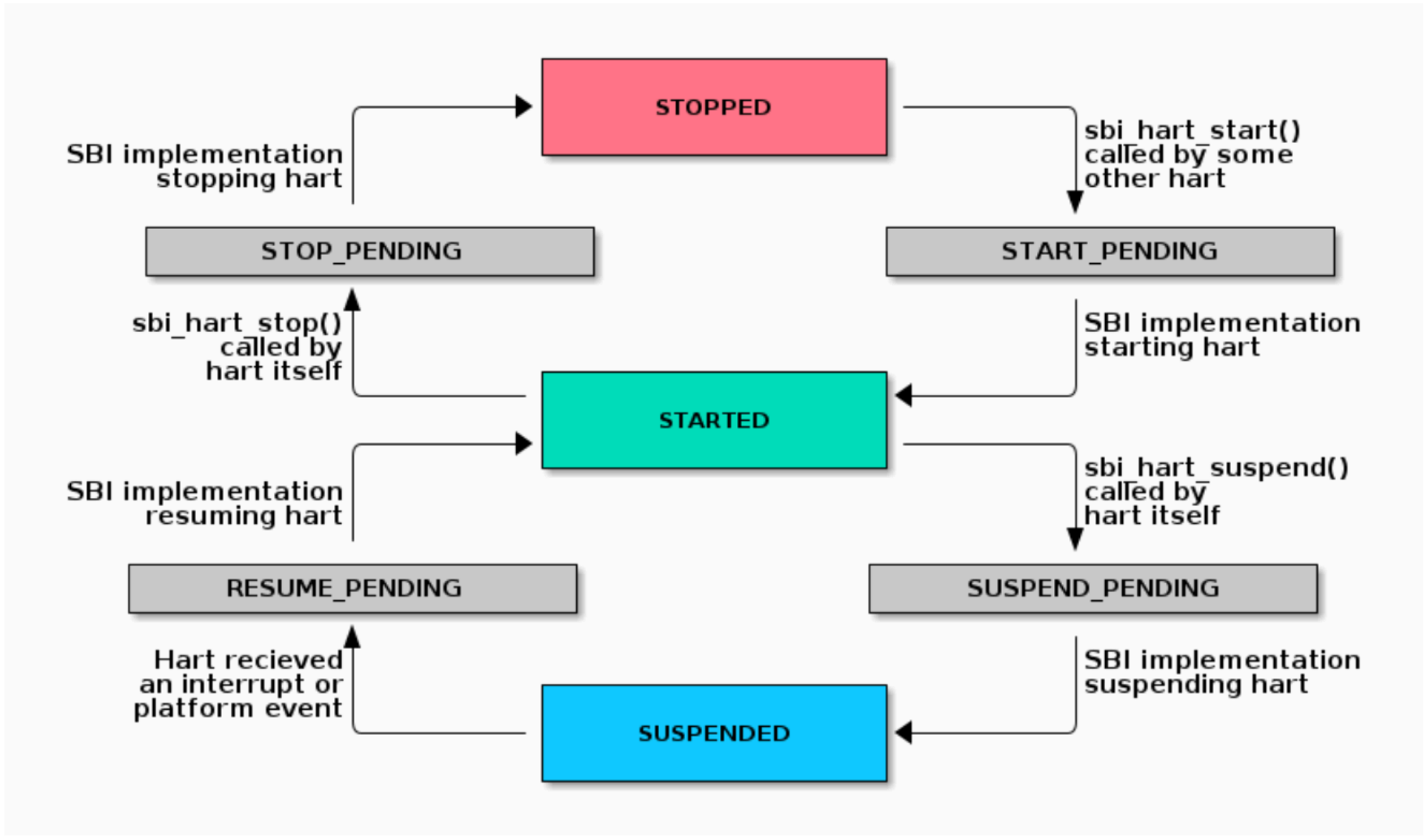


Figure 3. SBI HSM State Machine

A platform can have multiple harts grouped into hierarchical topology groups (namely cores, clusters, nodes, etc.) with separate platform specific low-power states for each hierarchical group. These platform specific low-power states of hierarchical topology groups can be represented as platform specific suspend states of a hart. An SBI implementation can utilize the suspend states of higher topology groups using one of the following approaches:

1. **Platform-coordinated:** In this approach, when a hart becomes idle the supervisor-mode power-managment software will request deepest suspend state for the hart and higher topology groups. An SBI implementation should choose a suspend state at higher topology group which is:
   a. Not deeper than the specified suspend state
   b. Wake-up latency is not higher than the wake-up latency of the specified suspend state
2. **OS-inititated:** In this approach, the supervisor-mode power-managment software will directly request a suspend state for higher topology group after the last hart in that group becomes idle. When a hart becomes idle, the supervisor-mode power-managment software will always select suspend state for the hart itself but it will select a suspend state for a higher topology group only if the hart is the last running hart in the group. An SBI implementation should:
   a. Never choose a suspend state for higher topology group different from the specified suspend state
   b. Always prefer most recent suspend state requested for higher topology group

9.1. Function: Hart start (FID #0)

```
struct sbiret sbi_hart_start(unsigned long hartid,
                             unsigned long start_addr,
                             unsigned long opaque)
```

Request the SBI implementation to start executing the target hart in supervisor-mode, at the address

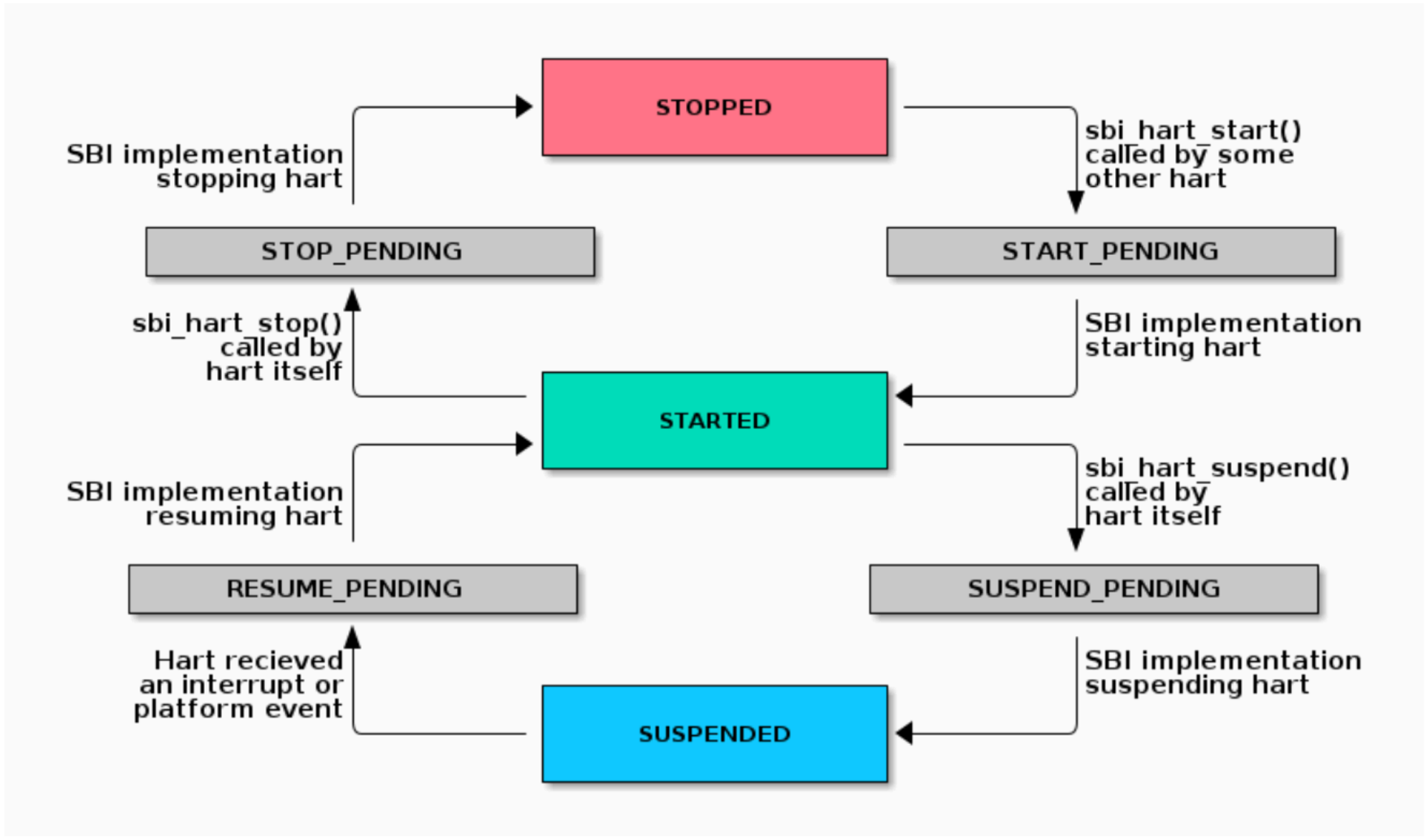


图 3. SBI HSM 状态机

一个平台可将多个硬件线程(hart)分组为层次化拓扑组(即核心、集群、节点等)，每个层次组具有独立的平台特定低功耗状态。这些层次化拓扑组的平台特定低功耗状态可表示为硬件线程的平台特定挂起状态。SBI 实现可采用以下任一方式利用更高层次拓扑组的挂起状态：

1. 平台协调式：在此方法中，当某个硬件线程进入空闲状态时，监管模式电源管理软件将请求该硬件线程及更高拓扑层级进入最深休眠状态。

   SBI 实现应选择更高拓扑组中的挂起状态，即：

   a. 不超过指定休眠状态的深度

   b. 唤醒延迟不高于指定休眠状态的唤醒延迟

2. 操作系统发起式：在这种方法中，当拓扑组中最后一个硬件线程进入空闲状态时，监管模式电源管理软件将直接为该高层拓扑组请求休眠状态。当某个硬件线程进入空闲状态时，监管模式电源管理软件总会为该硬件线程自身选择休眠状态，但只有当该硬件线程是组内最后一个运行的线程时，才会为高层拓扑组选择休眠状态。SBI 实现应当：

   a. 绝不为高层拓扑组选择与指定休眠状态不同的休眠状态

   b. 对于更高拓扑组，始终优先采用最近请求的挂起状态

9.1. 功能：Hart 启动（功能号#0） struct sbiret sbi_hart_start(unsigned long hartid,

```
                                unsigned long start_addr, unsigned
                                long opaque)
```

请求 SBI 实现以监管模式在指定地址启动目标 hart 执行

specified by `start_addr`, with the specific register values described in Table 18.

Table 18. HSM Hart Start Register State

Register Name	Register Value
satp	0
sstatus.SIE	0
a0	hartid
a1	`opaque` parameter
All other registers remain in an undefined state.	

A single `unsigned long` parameter is sufficient as `start_addr`, because the hart will start execution in supervisor-mode with the MMU off, hence `start_addr` must be less than XLEN bits wide.

This call is asynchronous — more specifically, the `sbi_hart_start()` may return before the target hart starts executing as long as the SBI implementation is capable of ensuring the return code is accurate. If the SBI implementation is a platform runtime firmware executing in machine-mode (M-mode), then it MUST configure any physical memory protection it supports, such as that defined by PMP, and other M-mode state, before transferring control to supervisor-mode software.

The `hartid` parameter specifies the target hart which is to be started.

The `start_addr` parameter points to a runtime-specified physical address, where the hart can start executing in supervisor-mode.

The `opaque` parameter is an XLEN-bit value which will be set in the `a1` register when the hart starts executing at `start_addr`.

The possible error codes returned in `sbiret.error` are shown in the Table 19 below.

Table 19. HSM Hart Start Errors

Error code	Description
SBI_SUCCESS	Hart was previously in stopped state. It will start executing from `start_addr`.
SBI_ERR_INVALID_ADDRESS	`start_addr` is not valid, possibly due to the following reasons: * It is not a valid physical address. * Executable access to the address is prohibited by a physical memory protection mechanism or H-extension G-stage for supervisor-mode.
SBI_ERR_INVALID_PARAM	`hartid` is not a valid hartid as the corresponding hart cannot be started in supervisor mode.
SBI_ERR_ALREADY_AVAILABLE	The given hartid is already started.
SBI_ERR_FAILED	The start request failed for unspecified or unknown other reasons.

9.2. Function: Hart stop (FID #1)

```
struct sbiret sbi_hart_stop(void)
```

Request the SBI implementation to stop executing the calling hart in supervisor-mode and return its ownership to the SBI implementation. This call is not expected to return under normal conditions. The

由 start_addr 指定，具体寄存器值如表 18 所述。

表 18. HSM 硬件线程启动寄存器状态

寄存器名称	寄存器值
satp	0
sstatus.SIE	0
a0	硬件线程 ID
寄存器 a1	不透明参数
其余所有寄存器均处于未定义状态。	

- 无符号长整型参数 start_addr 已足够，因为硬件线程将在 MMU 关闭的监管模式下开始执行，因此 start_addr 的位宽必须小于 XLEN。

该调用是异步的——更具体地说，只要 SBI 实现能够确保返回代码准确，sbi_hart_start()可能在目标硬件线程开始执行前就返回。若 SBI 实现是运行在机器模式(M 模式)的平台运行时固件，则必须在将控制权转移给监管模式软件前，配置其支持的所有物理内存保护机制（如 PMP 定义的机制）及其他 M 模式状态。

hartid 参数指定了要启动的目标硬件线程。

start_addr 参数指向一个运行时指定的物理地址，目标硬件线程可在该地址以监管模式开始执行。
不透明参数是一个 XLEN 位宽的值，当硬件线程(hart)开始从 start_addr 地址执行时，该值将被设置到 a1 寄存器中。

sbiret.error 可能返回的错误代码如下表 19 所示。

表 19. HSM 硬件线程启动错误

错误代码	描述
SBI_SUCCESS	该硬件线程之前处于停止状态。它将从 start_addr 地址开始执行。
SBI_ERR_INVALID_ADDRESS start_addr	地址无效，可能原因包括：* 该地址不是有效的物理地址 * 由于物理内存保护机制或 H 扩展 G 阶段对监管者模式的限制，禁止对该地址进行可执行访问
SBI_ERR_INVALID_PARAM	hartid 是无效的硬件线程 ID，因为对应的硬件线程无法在监管者模式下启动
SBI_ERR_ALREADY_AVAILABLE 给定的硬件线程 ID 已处于启动状态	
SBI_ERR_FAILED	启动请求因未指定或其他未知原因失败。

9.2. 功能：Hart 停止（FID #1） struct sbiret sbi_hart_stop(void)

请求 SBI 实现停止在监管模式下执行调用 hart，并将其所有权归还给 SBI 实现。正常情况下，此调用不应返回。

`sbi_hart_stop()` must be called with supervisor-mode interrupts disabled.

The possible error codes returned in `sbiret.error` are shown in the Table 20 below.

Table 20. HSM Hart Stop Errors

Error code	Description
SBI_ERR_FAILED	Failed to stop execution of the current hart

9.3. Function: Hart get status (FID #2)

```
struct sbiret sbi_hart_get_status(unsigned long hartid)
```

Get the current status (or HSM state id) of the given hart in `sbiret.value`, or an error through `sbiret.error`.

The `hartid` parameter specifies the target hart for which status is required.

The possible status (or HSM state id) values returned in `sbiret.value` are described in Table 17.

The possible error codes returned in `sbiret.error` are shown in the Table 21 below.

Table 21. HSM Hart Get Status Errors

Error code	Description
SBI_ERR_INVALID_PARAM	The given `hartid` is not valid.

The harts may transition HSM states at any time due to any concurrent `sbi_hart_start()` or `sbi_hart_stop()` or `sbi_hart_suspend()` calls, the return value from this function may not represent the actual state of the hart at the time of return value verification.

9.4. Function: Hart suspend (FID #3)

```
struct sbiret sbi_hart_suspend(uint32_t suspend_type,
                               unsigned long resume_addr,
                               unsigned long opaque)
```

Request the SBI implementation to put the calling hart in a platform specific suspend (or low power) state specified by the `suspend_type` parameter. The hart will automatically come out of suspended state and resume normal execution when it receives an interrupt or platform specific hardware event.

The platform specific suspend states for a hart can be either retentive or non-retentive in nature. A retentive suspend state will preserve hart register and CSR values for all privilege modes whereas a non-retentive suspend state will not preserve hart register and CSR values.

Resuming from a retentive suspend state is straight forward and the supervisor-mode software will see SBI suspend call return without any failures. The `resume_addr` parameter is unused during retentive suspend.

Resuming from a non-retentive suspend state is relatively more involved and requires software to restore various hart registers and CSRs for all privilege modes. Upon resuming from non-retentive suspend state,

调用 sbi_hart_stop()时必须禁用监管模式中断。

sbiret.error 可能返回的错误代码如下表 20 所示。

表 20. HSM 核停止错误类型

错误代码	描述
SBI_ERR_FAILED	未能停止当前核的执行

9.3. 功能：获取硬件线程状态（功能号 #2） struct sbiret sbi_hart_get_status(unsigned long hartid)

通过 sbiret.value 获取指定硬件线程的当前状态（或 HSM 状态 ID），或通过
sbiret.error 返回错误信息

参数 hartid 指定需要查询状态的目标硬件线程。

表 17 描述了 sbiret.value 中可能返回的状态（或 HSM 状态 ID）值。

sbiret.error 中可能返回的错误代码如下表 21 所示。

表 21. HSM 硬件线程状态获取错误码

错误代码	描述
SBI_ERR_INVALID_PARAM	给定的 hartid 无效。

由于任何并发的 sbi_hart_start()调用，harts 可能随时转换 HSM 状态
或 sbi_hart_stop()、sbi_hart_suspend()调用，本函数的返回值可能无法反映实际验证返回值时 hart 的状态。

9.4. 功能：Hart 挂起（功能 ID #3） struct sbiret sbi_hart_suspend(uint32_t suspend_type,

```
                                    unsigned long resume_addr, unsigned
                                    long opaque)
```

请求 SBI 实现将调用 hart 置于由 suspend_type 参数指定的平台特定挂起（或低功耗）状态。当 hart 接收到中断或平台特定硬件事件时，将自动从挂起状态恢复并继续正常执行。
硬件线程（hart）的平台特定休眠状态本质上可分为保持型与非保持型。保持型休眠状态会保留所有特权模式下的硬件线程寄存器及 CSR 值，而非保持型休眠状态则不会保留这些值。
从保持型休眠状态恢复的过程直接简单，监督模式软件将看到 SBI 休眠调用正常返回且无任何故障。在此过程中，resume_addr 参数不会被使用。
从非保持型休眠状态恢复相对复杂，需要软件为所有特权模式重新配置各类硬件线程寄存器及 CSR 值。当从非保持型休眠状态恢复时，

the hart will jump to supervisor-mode at address specified by `resume_addr` with specific registers values described in the Table 22 below.

Table 22. HSM Hart Resume Register State

Register Name	Register Value
satp	0
sstatus.SIE	0
a0	hartid
a1	`opaque` parameter
All other registers remain in an undefined state.	

A single `unsigned long` parameter is sufficient for `resume_addr`, because the hart will resume execution in supervisor-mode with the MMU off, hence `resume_addr` must be less than XLEN bits wide.

The `suspend_type` parameter is 32 bits wide and the possible values are shown in Table 23 below.

Table 23. HSM Hart Suspend Types

Value	Description
0x00000000	Default retentive suspend
0x00000001 - 0x0FFFFFFF	Reserved for future use
0x10000000 - 0x7FFFFFFF	Platform specific retentive suspend
0x80000000	Default non-retentive suspend
0x80000001 - 0x8FFFFFFF	Reserved for future use
0x90000000 - 0xFFFFFFFF	Platform specific non-retentive suspend

The `resume_addr` parameter points to a runtime-specified physical address, where the hart can resume execution in supervisor-mode after a non-retentive suspend.

The `opaque` parameter is an XLEN-bit value which will be set in the `a1` register when the hart resumes execution at `resume_addr` after a non-retentive suspend.

The possible error codes returned in `sbiret.error` are shown in the Table 24 below.

Table 24. HSM Hart Suspend Errors

Error code	Description
SBI_SUCCESS	Hart has suspended and resumed successfully from a retentive suspend state.
SBI_ERR_INVALID_PARAM	`suspend_type` is reserved or is platform-specific and unimplemented.
SBI_ERR_NOT_SUPPORTED	`suspend_type` is not reserved and is implemented, but the platform does not support it due to one or more missing dependencies.
SBI_ERR_INVALID_ADDRESS	`resume_addr` is not valid, possibly due to the following reasons: * It is not a valid physical address. * Executable access to the address is prohibited by a physical memory protection mechanism or H-extension G-stage for supervisor-mode.
SBI_ERR_FAILED	The suspend request failed for unspecified or unknown other reasons.

9.5. Function Listing

硬件线程将以表 22 中描述的特定寄存器值跳转到由 resume_addr 指定的监督模式地址。

表 22. HSM 硬件线程恢复寄存器状态

寄存器名称	寄存器值
satp	0
sstatus.SIE	0
a0	硬件线程 ID
寄存器 a1	不透明参数
其余所有寄存器均处于未定义状态。	

无符号长整型 由于硬件线程将在 MMU 关闭的管理员模式下恢复执行，因此 resume_addr 参数只需小于 XLEN 位宽即可满足要求。

suspend_type 参数为 32 位宽，其可能取值如下表 23 所示。

表 23. HSM 硬件线程挂起类型

值	描述
0x00000000	默认保持性挂起
0x00000001 - 0x0FFFFFFF	保留供未来使用
0x10000000 - 0x7FFFFFFF	平台特定的保持性挂起
0x80000000	默认非保持性挂起
0x80000001 - 0x8FFFFFFF	保留供未来使用
0x90000000 - 0xFFFFFFFF	平台特定的非保持性挂起

resume_addr 参数指向一个运行时指定的物理地址，在该地址处 hart 可在非保持性挂起后以监管模式恢复执行。opaque 参数是一个 XLEN 位值，当 hart 在非保持性挂起后于 resume_addr 恢复执行时，该值将被设置到 a1 寄存器中。

sbiret.error 中可能返回的错误代码如下表 24 所示。

表 24. HSM Hart 挂起错误

错误代码	描述
SBI_SUCCESS	硬件线程(Hart)已成功从保持性挂起状态恢复运行
SBI_ERR_INVALID_PARAM	suspend_type 为保留值或平台特定未实现类型
SBI_ERR_NOT_SUPPORTED	suspend_type 非保留且已实现，但因缺少一个或多个依赖条件导致平台不支持该类型
SBI_ERR_INVALID_ADDRESS 恢复地址(resume_addr)无效，可能出	于以下原因导致：* 该地址不是有效的物理地址 * 物理内存保护机制或 H-扩展的 G-stage 禁止以监管者模式对该地址进行可执行访问
SBI_ERR_FAILED	由于未指定或其他未知原因，挂起请求失败。

9.5. 函数列表

Table 25. HSM Function List

Function Name	SBI Version	FID	EID
sbi_hart_start	0.2	0	0x48534D
sbi_hart_stop	0.2	1	0x48534D
sbi_hart_get_status	0.2	2	0x48534D
sbi_hart_suspend	0.3	3	0x48534D

表 25. HSM 功能列表

功能名称	SBI 版本	功能标识符	扩展 ID
sbi_hart_start	0.2	0	0x48534D
sbi_hart_stop	0.2	1	0x48534D
sbi_hart_get_status	0.2	2	0x48534D
sbi_hart_suspend	0.3	3	0x48534D

Chapter 10. System Reset Extension (EID #0x53525354 "SRST")

The System Reset Extension provides a function that allow the supervisor software to request system-level reboot or shutdown. The term "system" refers to the world-view of supervisor software and the underlying SBI implementation could be provided by machine mode firmware or a hypervisor.

10.1. Function: System reset (FID #0)

```
struct sbiret sbi_system_reset(uint32_t reset_type, uint32_t reset_reason)
```

Reset the system based on provided `reset_type` and `reset_reason`. This is a synchronous call and does not return if it succeeds.

The `reset_type` parameter is 32 bits wide and it's possible values are shown in the Table 26 below.

Table 26. SRST System Reset Types

Value	Description
0x00000000	Shutdown
0x00000001	Cold reboot
0x00000002	Warm reboot
0x00000003 - 0xEFFFFFFF	Reserved for future use
0xF0000000 - 0xFFFFFFFF	Vendor or platform specific reset type

The `reset_reason` is an optional parameter representing the reason for system reset. This parameter is 32 bits wide with possible values shown in the Table 27 below

Table 27. SRST System Reset Reasons

Value	Description
0x00000000	No reason
0x00000001	System failure
0x00000002 - 0xDFFFFFFF	Reserved for future use
0xE0000000 - 0xEFFFFFFF	SBI implementation specific reset reason
0xF0000000 - 0xFFFFFFFF	Vendor or platform specific reset reason

When supervisor software is running natively, the SBI implementation is provided by machine mode firmware. In this case, shutdown is equivalent to a physical power down of the entire system and cold reboot is equivalent to a physical power cycle of the entire system. Further, warm reboot is equivalent to a power cycle of the main processor and parts of the system, but not the entire system. For example, on a server class system with a BMC (board management controller), a warm reboot will not power cycle the BMC whereas a cold reboot will definitely power cycle the BMC.

When supervisor software is running inside a virtual machine, the SBI implementation is provided by a hypervisor. Shutdown, cold reboot and warm reboot will behave functionally the same as the native case, but might not result in any physical power changes.

The possible error codes returned in `sbiret.error` are shown in the Table 28 below.

第 10 章 系统复位扩展（扩展 ID #0x53525354 "SRST"）

系统复位扩展提供了一项功能，允许监督模式软件请求系统级重启或关机。这里的"系统"是指监督模式软件的视角，底层 SBI 实现可能由机器模式固件或虚拟机监控程序提供。

10.1. 功能：系统复位（功能 ID #0）struct sbiret sbi_system_reset(uint32_t reset_type, uint32_t reset_reason)

根据提供的 reset_type 和 reset_reason 参数复位系统。这是一个同步调用，若执行成功则不会返回。

reset_type 参数为 32 位宽，其可能取值如下表 26 所示。

表 26. SRST 系统复位类型

取值	描述
0x00000000	关机
0x00000001	冷启动
0x00000002	热启动
0x00000003 - 0xEFFFFFFF	保留供未来使用
0xF0000000 - 0xFFFFFFFF	厂商或平台特定的复位类型

reset_reason 是一个可选参数，表示系统复位的原因。该参数宽度为 32 位，其可能取值如下表 27 所示

表 27. SRST 系统复位原因

值	描述
0x00000000	无原因
0x00000001	系统故障
0x00000002 - 0xDFFFFFFF	保留供未来使用
0xE0000000 - 0xEFFFFFFF	SBI 实现特定的复位原因
0xF0000000 - 0xFFFFFFFF	厂商或平台特定的复位原因

当监管模式软件原生运行时，SBI 实现由机器模式固件提供。此时，关机等同于整个系统的物理断电，冷重启等同于整个系统的物理电源循环。此外，热重启等同于主处理器及系统部分（而非整个系统）的电源循环。例如，在配备 BMC（板级管理控制器）的服务器级系统中，热重启不会对 BMC 进行电源循环，而冷重启必定会对 BMC 进行电源循环。

当监管模式软件在虚拟机内运行时，SBI 实现由虚拟机监控程序提供。关机、冷重启和热重启在功能表现上与原生情况相同，但可能不会引发任何物理电源状态变化。

表 28 列出了 sbiret.error 可能返回的错误代码。

Table 28. SRST System Reset Errors

Error code	Description
SBI_ERR_INVALID_PARAM	At least one of `reset_type` or `reset_reason` is reserved or is platform-specific and unimplemented.
SBI_ERR_NOT_SUPPORTED	`reset_type` is not reserved and is implemented, but the platform does not support it due to one or more missing dependencies.
SBI_ERR_FAILED	The reset request failed for unspecified or unknown other reasons.

10.2. Function Listing

Table 29. SRST Function List

Function Name	SBI Version	FID	EID
sbi_system_reset	0.3	0	0x53525354

表 28. SRST 系统复位错误类型

错误代码	描述
SBI_ERR_INVALID_PARAM reset_type 或	reset_reason 参数中至少有一个是保留值，或是平台特定但未实现的
SBI_ERR_NOT_SUPPORTED	reset_type 参数非保留值且已实现，但由于缺少一个或多个依赖项导致平台不支持该功能
SBI_ERR_FAILED	复位请求因未指定或其他未知原因失败。

10.2. 功能列表

表 29. SRST 功能列表

功能名称	SBI 版本	功能标识符	扩展 ID
sbi_system_reset	0.3	0	0x53525354

Chapter 11. Performance Monitoring Unit Extension (EID #0x504D55 "PMU")

The RISC-V hardware performance counters such as `mcycle`, `minstret`, and `mhpmcounterX` CSRs are accessible as read-only from supervisor-mode using `cycle`, `instret`, and `hpmcounterX` CSRs. The SBI performance monitoring unit (PMU) extension is an interface for supervisor-mode to configure and use the RISC-V hardware performance counters with assistance from the machine-mode (or hypervisor-mode). These hardware performance counters can only be started, stopped, or configured from machine-mode using `mcountinhibit` and `mhpmeventX` CSRs. Due to this, a machine-mode SBI implementation may choose to disallow SBI PMU extension if `mcountinhibit` CSR is not implemented by the RISC-V platform.

A RISC-V platform generally supports monitoring of various hardware events using a limited number of hardware performance counters which are up to 64 bits wide. In addition, a SBI implementation can also provide firmware performance counters which can monitor firmware events such as number of misaligned load/store instructions, number of RFENCEs, number of IPIs, etc. All firmware counters must have same number of bits and can be up to 64 bits wide.

The SBI PMU extension provides:

1. An interface for supervisor-mode software to discover and configure per-hart hardware/firmware counters
2. A typical Linux perf [4] compatible interface for hardware/firmware performance counters and events
3. Full access to microarchitecture's raw event encodings

To define SBI PMU extension calls, we first define important entities `counter_idx`, `event_idx`, and `event_data`. The `counter_idx` is a logical number assigned to each hardware/firmware counter. The `event_idx` represents a hardware (or firmware) event whereas the `event_data` is 64 bits wide and represents additional configuration (or parameters) for a hardware (or firmware) event.

The event_idx is a 20 bits wide number encoded as follows:

```
event_idx[19:16] = type
event_idx[15:0] = code
```

The below table describes the different types of events supported in this specification.

Table 30. PMU Event Type

Event ID Type	Value	Description
Type #0	0	Hardware general events
Type #1	1	Hardware Cache events
Type #2	2	Hardware raw events (deprecated) Bits allowed for mhpmeventX [0:47]
Type #3	3	Hardware raw events v2 Bits allowed for mhpmeventX [0:55]
Type #15	15	Firmware events

11.1. Event: Hardware general events (Type #0)

The `event_idx.type` (i.e. **event type**) should be `0x0` for all hardware general events and each hardware

第 11 章 性能监控单元扩展（扩展 ID #0x504D55 "PMU"）

RISC-V 硬件性能计数器（如 mcycle、minstret 和 mhpmcounterX 控制状态寄存器）在监督模式下可通过 cycle、instret 和 hpmcounterX 控制状态寄存器以只读方式访问。SBI 性能监控单元（PMU）扩展为监督模式提供了接口，可在机器模式（或虚拟机监控模式）的协助下配置和使用 RISC-V 硬件性能计数器。这些硬件性能计数器只能通过机器模式下的 mcountinhibit 和 mhpmeventX 控制状态寄存器进行启动、停止或配置。因此，若 RISC-V 平台未实现 mcountinhibit 控制状态寄存器，机器模式的 SBI 实现可选择禁用 SBI PMU 扩展功能。

RISC-V 平台通常支持使用有限数量的硬件性能计数器来监控各类硬件事件，这些计数器宽度可达 64 位。此外，SBI 实现还可提供固件性能计数器，用于监控固件事件（如未对齐加载/存储指令数量、RFENCE 指令数量、IPI 中断数量等）。所有固件计数器必须具有相同的位宽，且最大支持 64 位。

SBI PMU 扩展提供以下功能：

1. 为监督模式软件提供的接口，用于发现和配置每个硬件线程的硬件/固件计数器
2. 兼容 Linux 性能分析工具[4]的标准接口，用于访问硬件/固件性能计数器与事件
3. 完全支持微架构原始事件编码的访问

为定义 SBI PMU 扩展调用，我们首先定义重要实体 counter_idx（计数器索引）、event_idx（事件索引）和 event_data（事件数据）。counter_idx 是分配给每个硬件/固件计数器的逻辑编号。该 event_idx 代表硬件（或固件）事件，而 event_data 宽度为 64 位，代表硬件（或固件）事件的附加配置（或参数）。

event_idx 是一个 20 位宽的数字，编码方式如下：

```
event_idx[19:16] = 类型
event_idx[15:0] = 代码
```

下表描述了本规范支持的不同事件类型。

表 30. PMU 事件类型

事件 ID 类型	数值	描述
类型 #0	0	硬件通用事件
类型 #1	1	硬件缓存事件
类型 #2	2	硬件原生事件（已弃用） mhpmeventX 允许使用的位域[0:47]
类型#3	3	硬件原生事件 v2 版本 mhpmeventX 允许使用的位域[0:55]
类型#15	15	固件事件

11.1. 事件：硬件通用事件（类型#0）

对于所有硬件通用事件，event_idx.type（即事件类型）应设为 0x0，且每个硬件

general event is identified by an unique `event_idx.code` (i.e. **event code**) described in the Table 31 below.

Table 31. PMU Hardware Events

General Event Name	Code	Description
SBI_PMU_HW_NO_EVENT	0	Unused event because `event_idx` cannot be zero
SBI_PMU_HW_CPU_CYCLES	1	Event for each CPU cycle
SBI_PMU_HW_INSTRUCTIONS	2	Event for each completed instruction
SBI_PMU_HW_CACHE_REFERENCES	3	Event for cache hit
SBI_PMU_HW_CACHE_MISSES	4	Event for cache miss
SBI_PMU_HW_BRANCH_INSTRUCTIONS	5	Event for a branch instruction
SBI_PMU_HW_BRANCH_MISSES	6	Event for a branch misprediction
SBI_PMU_HW_BUS_CYCLES	7	Event for each BUS cycle
SBI_PMU_HW_STALLED_CYCLES_FRONTEND	8	Event for a stalled cycle in microarchitecture frontend
SBI_PMU_HW_STALLED_CYCLES_BACKEND	9	Event for a stalled cycle in microarchitecture backend
SBI_PMU_HW_REF_CPU_CYCLES	10	Event for each reference CPU cycle

The `event_data` (i.e. **event data**) is unused for hardware general events and all non-zero values of `event_data` are reserved for future use.

A RISC-V platform might halt the CPU clock when it enters WAIT state using the WFI instruction or enters platform specific SUSPEND state using the SBI HSM hart suspend call.

The SBI_PMU_HW_CPU_CYCLES *event counts CPU clock cycles as counted by the* `cycle` *CSR. These may be variable frequency cycles, and are not counted when the CPU clock is halted.*

The SBI_PMU_HW_REF_CPU_CYCLES *counts fixed-frequency clock cycles while the CPU clock is not halted. The fixed-frequency of counting might, for example, be the same frequency at which the* `time` *CSR counts.*

The SBI_PMU_HW_BUS_CYCLES *counts fixed-frequency clock cycles. The fixed-frequency of counting might be the same frequency at which the* `time` *CSR counts, or may be the frequency of the clock at the boundary between the hart (and it's private caches) and the rest of the system.*

11.2. Event: Hardware cache events (Type #1)

The `event_idx.type` (i.e. **event type**) should be `0x1` for all hardware cache events and each hardware cache event is identified by an unique `event_idx.code` (i.e. **event code**) which is encoded as follows:

```
event_idx.code[15:3] = cache_id
event_idx.code[2:1] = op_id
event_idx.code[0:0] = result_id
```

Below tables show possible values of: `event_idx.code.cache_id` (i.e. **cache event id**),

通用事件通过表 31 中描述的独特 event_idx.code（即事件代码）进行标识。

表 31. PMU 硬件事件

通用事件名称	代码描述	
SBI_PMU_HW_NO_EVENT	0	因 event_idx 不能为零而未使用的事件
SBI_PMU_HW_CPU_CYCLES	1	每个 CPU 周期对应的事件
SBI_PMU_HW_INSTRUCTIONS	2	每条已执行指令对应的事件
SBI_PMU_HW_CACHE_REFERENCES	3	缓存命中对应的事件
SBI_PMU_HW_CACHE_MISSES	4	缓存未命中事件
SBI_PMU_HW_BRANCH_INSTRUCTIONS	5	分支指令事件
SBI_PMU_HW_BRANCH_MISSES	6	分支预测错误事件
SBI_PMU_HW_BUS_CYCLES	7	每个总线周期对应的事件
SBI_PMU_HW_STALLED_CYCLES_FRONTEND	8	微架构前端停滞周期事件
SBI_PMU_HW_STALLED_CYCLES_BACKEND	9	微架构后端停滞周期事件
SBI_PMU_HW_REF_CPU_CYCLES	10	参考 CPU 周期事件

硬件通用事件的事件数据（即 event_data）未被使用，所有非零值的事件数据保留供未来使用。

☒ RISC-V 平台在执行 WFI 指令进入 WAIT 状态或通过 SBI HSM hart suspend 调用进入平台特定 SUSPEND 状态时，可能会停止 CPU 时钟。

SBI_PMU_HW_CPU_CYCLES 事件统计由 cycle CSR 记录的 CPU 时钟周期数。这些可能是可变频率周期，且当 CPU 时钟停止时不进行计数。

☒SBI_PMU_HW_REF_CPU_CYCLES 用于在 CPU 时钟未停顿时统计固定频率时钟周期。该固定计数频率可能与时间 CSR 计数频率相同。

☒ SBI_PMU_HW_BUS_CYCLES 用于统计固定频率时钟周期。该固定计数频率可能与时间 CSR 计数频率相同，也可能是硬件线程（及其私有缓存）与系统其余部分之间边界处的时钟频率。

11.2. 事件：硬件缓存事件（类型#1）

对于所有硬件缓存事件，event_idx.type（即事件类型）应设为 0x1。每个硬件缓存事件由唯一的 event_idx.code（即事件代码）标识，其编码方式如下：

```
event_idx.code[15:3] = cache_id
event_idx.code[2:1] = op_id
event_idx.code[0:0] = result_id
```

下表展示了 event_idx.code.cache_id（即缓存事件 ID）的可能取值：

`event_idx.code.op_id` (i.e. **cache operation id**) and `event_idx.code.result_id` (i.e. **cache result id**).

Table 32. PMU Cache Event ID

Cache Event Name	Event ID	Description
SBI_PMU_HW_CACHE_L1D	0	Level1 data cache event
SBI_PMU_HW_CACHE_L1I	1	Level1 instruction cache event
SBI_PMU_HW_CACHE_LL	2	Last level cache event
SBI_PMU_HW_CACHE_DTLB	3	Data TLB event
SBI_PMU_HW_CACHE_ITLB	4	Instruction TLB event
SBI_PMU_HW_CACHE_BPU	5	Branch predictor unit event
SBI_PMU_HW_CACHE_NODE	6	NUMA node cache event

Table 33. PMU Cache Operation ID

Cache Operation Name	Operation ID	Description
SBI_PMU_HW_CACHE_OP_READ	0	Read cache line
SBI_PMU_HW_CACHE_OP_WRITE	1	Write cache line
SBI_PMU_HW_CACHE_OP_PREFETCH	2	Prefetch cache line

Table 34. PMU Cache Operation Result ID

Cache Result Name	Result ID	Description
SBI_PMU_HW_CACHE_RESULT_ACCESS	0	Cache access
SBI_PMU_HW_CACHE_RESULT_MISS	1	Cache miss

The `event_data` (i.e. **event data**) is unused for hardware cache events and all non-zero values of `event_data` are reserved for future use.

11.3. Event: Hardware raw events (Type #2)

The `event_idx.type` (i.e. **event type**) should be `0x2` for all hardware raw events and `event_idx.code` (i.e. **event code**) should be zero.

On RISC-V platforms with 32 bits wide `mhpmeventX` CSRs, the `event_data` configuration (or parameter) should have the 32-bit value to to be programmed in the `mhpmeventX` CSR.

On RISC-V platforms with 64 bits wide `mhpmeventX` CSRs, the `event_data` configuration (or parameter) should have the 48-bit value to be programmed in the lower 48-bits of `mhpmeventX` CSR and the SBI implementation shall determine the value to be programmed in the upper 16 bits of `mhpmeventX` CSR.

This event type is deprecated in favor of `raw events v2`.

11.4. Event: Hardware raw events v2 (Type #3)

The `event_idx.type` (i.e. **event type**) should be `0x3` for all hardware raw events and `event_idx.code` (i.e. **event code**) should be zero.

On RISC-V platforms with 32 bits wide `mhpmeventX` CSRs, the `event_data` configuration (or parameter) should have the 32-bit value to to be programmed in the `mhpmeventX` CSR.

`event_idx.code.op_id` （即缓存操作 ID）和 event_idx.code.result_id（即缓存结果 ID）。

表 32. PMU 缓存事件 ID

缓存事件名称	事件 ID	描述
SBI_PMU_HW_CACHE_L1D	0	一级数据缓存事件
SBI_PMU_HW_CACHE_L1I	1	一级指令缓存事件
SBI_PMU_HW_CACHE_LL	2	末级缓存事件
SBI_PMU_HW_CACHE_DTLB	3	数据 TLB 事件
SBI_PMU_HW_CACHE_ITLB	4	指令 TLB 事件
SBI_PMU_HW_CACHE_BPU	5	分支预测单元事件
SBI_PMU_HW_CACHE_NODE	6	NUMA 节点缓存事件

表 33. PMU 缓存操作 ID

缓存操作名称	操作 ID 说明	
SBI_PMU_HW_CACHE_OP_READ	0	读取缓存行
SBI_PMU_HW_CACHE_OP_WRITE	1	写入缓存行
SBI_PMU_HW_CACHE_OP_PREFETCH	2	预取缓存行

表 34. PMU 缓存操作结果 ID

缓存结果名称	结果标识符	描述
SBI_PMU_HW_CACHE_RESULT_ACCESS	0	缓存访问
SBI_PMU_HW_CACHE_RESULT_MISS	1	缓存未命中

硬件缓存事件不使用 event_data（即事件数据），所有非零的 `event_data` 值均保留供未来使用。

11.3. 事件：硬件原生事件（类型#2）

所有硬件原始事件的事件索引类型（即事件类型）应为 0x2，且事件索引代码（即事件代码）应为零。

在具有 32 位宽 mhpmeventX 控制状态寄存器(CSR)的 RISC-V 平台上，事件数据配置（或参数）应包含需编程写入 mhpmeventX CSR 的 32 位数值。
在具有 64 位宽 mhpmeventX 控制状态寄存器的 RISC-V 平台上，事件数据配置（或参数）应包含需写入 mhpmeventX 寄存器低 48 位的 48 位数值，而 SBI 实现将决定需写入 mhpmeventX 寄存器高 16 位的数值。

☒ 该事件类型已弃用，建议改用原始事件 v2 版本。

11.4. 事件：硬件原生事件 v2（类型#3）

对于所有硬件原始事件，event_idx.type（即事件类型）应为 0x3，且 event_idx.code（即事件代码）应为零。

在 mhpmeventX 控制状态寄存器为 32 位宽的 RISC-V 平台上，事件数据配置（或参数）应为需编程写入 mhpmeventX 控制状态寄存器的 32 位数值。

On RISC-V platforms with 64 bits wide `mhpmeventX` CSRs, the `event_data` configuration (or parameter) should have the 56-bit value be programmed in the lower 56-bits of `mhpmeventX` CSR and the SBI implementation shall determine the value to be programmed in the upper 8 bits of `mhpmeventX` CSR based on privilege specification definition.

The RISC-V platform hardware implementation may choose to define the expected value to be written to `mhpmeventX` *CSR for a hardware event. In case of hardware general/cache events, the RISC-V platform hardware implementation may use the zero-extended* `event_idx` *as the expected value for simplicity.*

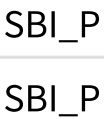
11.5. Event: Firmware events (Type #15)

The `event_idx.type` (i.e. **event type**) should be `0xf` for all firmware events and each firmware event is identified by an unique `event_idx.code` (i.e. **event code**) described in the Table 35 below.

Table 35. PMU Firmware Events

Firmware Event Name	Code	Description
SBI_PMU_FW_MISALIGNED_LOAD	0	Misaligned load trap event
SBI_PMU_FW_MISALIGNED_STORE	1	Misaligned store trap event
SBI_PMU_FW_ACCESS_LOAD	2	Load access trap event
SBI_PMU_FW_ACCESS_STORE	3	Store access trap event
SBI_PMU_FW_ILLEGAL_INSN	4	Illegal instruction trap event
SBI_PMU_FW_SET_TIMER	5	Set timer event
SBI_PMU_FW_IPI_SENT	6	Sent IPI to other hart event
SBI_PMU_FW_IPI_RECEIVED	7	Received IPI from other hart event
SBI_PMU_FW_FENCE_I_SENT	8	Sent FENCE.I request to other hart event
SBI_PMU_FW_FENCE_I_RECEIVED	9	Received FENCE.I request from other hart event
SBI_PMU_FW_SFENCE_VMA_SENT	10	Sent SFENCE.VMA request to other hart event
SBI_PMU_FW_SFENCE_VMA_RECEIVED	11	Received SFENCE.VMA request from other hart event
SBI_PMU_FW_SFENCE_VMA_ASID_SENT	12	Sent SFENCE.VMA with ASID request to other hart event
SBI_PMU_FW_SFENCE_VMA_ASID_RECEIVED	13	Received SFENCE.VMA with ASID request from other hart event
SBI_PMU_FW_HFENCE_GVMA_SENT	14	Sent HFENCE.GVMA request to other hart event
SBI_PMU_FW_HFENCE_GVMA_RECEIVED	15	Received HFENCE.GVMA request from other hart event
SBI_PMU_FW_HFENCE_GVMA_VMID_SENT	16	Sent HFENCE.GVMA with VMID request to other hart event
SBI_PMU_FW_HFENCE_GVMA_VMID_RECEIVED	17	Received HFENCE.GVMA with VMID request from other hart event
SBI_PMU_FW_HFENCE_VVMA_SENT	18	Sent HFENCE.VVMA request to other hart event

在具有 64 位宽 mhpmeventX 控制状态寄存器(CSR)的 RISC-V 平台上，事件数据配置（或参数）应将 56 位数值编程至 mhpmeventX CSR 的低 56 位，而 SBI 实现则需根据特权规范定义确定 mhpmeventX CSR 高 8 位的编程值。



RISC-V 平台硬件实现可选择为硬件事件定义需写入 mhpmeventX CSR 的预期值。对于硬件通用/缓存事件，为简化实现，RISC-V 平台硬件可采用零扩展的 event_idx 作为预期值。

11.5. 事件：固件事件（类型#15）

所有固件事件的 event_idx.type（即事件类型）应为 0xf，每个固件事件由表 35 所述唯一 event_idx.code（即事件代码）标识。

表 35. PMU 固件事件

固件事件名称	代码	描述
SBI_PMU_FW_MISALIGNED_LOAD	0	未对齐加载陷阱事件
SBI_PMU_FW_MISALIGNED_STORE	1	未对齐存储陷阱事件
SBI_PMU_FW_ACCESS_LOAD	2	加载访问陷阱事件
SBI_PMU_FW_ACCESS_STORE	3	存储访问陷阱事件
SBI_PMU_FW_ILLEGAL_INSN	4	非法指令陷阱事件
SBI_PMU_FW_SET_TIMER	5	设置定时器事件
SBI_PMU_FW_IPI_SENT	6	向其他硬件线程发送 IPI 事件
SBI_PMU_FW_IPI_RECEIVED	7	接收来自其他硬线程的 IPI 事件
SBI_PMU_FW_FENCE_I_SENT	8	向其他硬线程发送 FENCE.I 请求事件
SBI_PMU_FW_FENCE_I_RECEIVED	9	接收到来自其他硬线程的 FENCE.I 请求事件
SBI_PMU_FW_SFENCE_VMA_SENT	10	向其他硬线程发送 SFENCE.VMA 请求事件
SBI_PMU_FW_SFENCE_VMA_RECEIVED	11	接收到来自其他硬线程的 SFENCE.VMA 请求事件
SBI_PMU_FW_SFENCE_VMA_ASID_SENT	12	向其他硬线程发送带 ASID 的 SFENCE.VMA 请求事件
SBI_PMU_FW_SFENCE_VMA_ASID_RECEIVED	13	接收到来自其他硬线程的带 ASID 的 SFENCE.VMA 请求事件
SBI_PMU_FW_HFENCE_GVMA_SENT	14	向其他硬线程发送 HFENCE.GVMA 请求事件
SBI_PMU_FW_HFENCE_GVMA_RECEIVED	15	接收到来自其他 hart 的 HFENCE.GVMA 请求事件
SBI_PMU_FW_HFENCE_GVMA_VMID_SENT	16	发送带有 VMID 请求的 HFENCE.GVMA 指令至其他硬件线程事件
SBI_PMU_FW_HFENCE_GVMA_VMID_RECEIVED	17	接收到来自其他 hart 的带 VMID 请求的 HFENCE.GVMA 事件
SBI_PMU_FW_HFENCE_VVMA_SENT	18	向其他硬件线程发送 HFENCE.VVMA 请求事件

Firmware Event Name	Code	Description
SBI_PMU_FW_HFENCE_VVMA_RECEIVED	19	Received HFENCE.VVMA request from other hart event
SBI_PMU_FW_HFENCE_VVMA_ASID_SENT	20	Sent HFENCE.VVMA with ASID request to other hart event
SBI_PMU_FW_HFENCE_VVMA_ASID_RECEIVED	21	Received HFENCE.VVMA with ASID request from other hart event
Reserved	22 - 255	Reserved for future use
Implementation specific events	256 - 65534	SBI implementation specific firmware events
SBI_PMU_FW_PLATFORM	65535	RISC-V platform specific firmware events, where the `event_data` configuration (or parameter) contains the event encoding.

For all firmware events except SBI_PMU_FW_PLATFORM, the `event_data` configuration (or parameter) is unused and all non-zero values of `event_data` are reserved for future use.

11.6. Function: Get number of counters (FID #0)

```
struct sbiret sbi_pmu_num_counters()
```

Returns the number of counters (both hardware and firmware) in `sbiret.value` and always returns `SBI_SUCCESS` in sbiret.error.

11.7. Function: Get details of a counter (FID #1)

```
struct sbiret sbi_pmu_counter_get_info(unsigned long counter_idx)
```

Get details about the specified counter such as underlying CSR number, width of the counter, type of counter hardware/firmware, etc.

The `counter_info` returned by this SBI call is encoded as follows:

```
    counter_info[11:0] = CSR (12bit CSR number)
    counter_info[17:12] = Width (One less than number of bits in CSR)
    counter_info[XLEN-2:18] = Reserved for future use
    counter_info[XLEN-1] = Type (0 = hardware and 1 = firmware)
```

If `counter_info.type == 1` then `counter_info.csr` and `counter_info.width` should be ignored.

Returns the `counter_info` described above in `sbiret.value`.

The possible error codes returned in `sbiret.error` are shown in the Table 36 below.

Table 36. PMU Counter Get Info Errors

固件事件名称	代码	描述
SBI_PMU_FW_HFENCE_VVMA_RECEIVED	19	接收到来自其他硬线程的 HFENCE.VVMA 请求事件
SBI_PMU_FW_HFENCE_VVMA_ASID_SENT	20	向其他硬件线程发送带 ASID 的 HFENCE.VVMA 请求事件
SBI_PMU_FW_HFENCE_VVMA_ASID_RECEIVED 21		接收来自其他硬件线程的带 ASID 的 HFENCE.VVMA 请求事件
保留字段	22 - 255 保留供	未来使用
实现特定事件	256 - 65534 SBI	实现特定的固件事件
SBI_PMU_FW_PLATFORM	65535	RISC-V 平台特定的固件事件，其中 event_data 配置（或参数）包含事件编码。

除 SBI_PMU_FW_PLATFORM 外，所有固件事件的 event_data 配置（或参数）均未使用，所有非零的 event_data 值均保留供未来使用。

11.6. 功能：获取计数器数量（FID #0） struct sbiret sbi_pmu_num_counters()

返回 sbiret.value 中的计数器数量（包括硬件和固件计数器），且始终返回 `SBI_SUCCESS` 在 sbiret.error 中。

11.7. 功能：获取计数器详情（功能号 #1） struct sbiret sbi_pmu_counter_get_info(unsigned long counter_idx)

获取指定计数器的详细信息，如底层 CSR 编号、计数器位宽、计数器类型（硬件/固件）等。

该 SBI 调用返回的 counter_info 编码方式如下：

```
counter_info[11:0] = CSR（12 位 CSR 编号）
counter_info[17:12] = 位宽（CSR 位宽减 1 的值）
counter_info[XLEN-2:18] = 保留未来使用
counter_info[XLEN-1] = 类型（0=硬件计数器，1=固件计数器）
```

若 counter_info.type == 1，则应忽略 counter_info.csr 和 counter_info.width 字段。

通过 sbiret.value 返回上述描述的 counter_info 信息。

sbiret.error 可能返回的错误码如下表 36 所示。

表 36. PMU 计数器获取信息错误

Error code	Description
SBI_SUCCESS	`counter_info` read successfully.
SBI_ERR_INVALID_PARAM	`counter_idx` points to an invalid counter.

11.8. Function: Find and configure a matching counter (FID #2)

```
struct sbiret sbi_pmu_counter_config_matching(unsigned long counter_idx_base,
                                              unsigned long counter_idx_mask,
                                              unsigned long config_flags,
                                              unsigned long event_idx,
                                              uint64_t event_data)
```

Find and configure a counter from a set of counters which is not started (or enabled) and can monitor the specified event. The `counter_idx_base` and `counter_idx_mask` parameters represent the set of counters whereas `event_idx` represents the event to be monitored and `event_data` represents any additional event configuration.

The `config_flags` parameter represents additional counter configuration and filter flags. The bit definitions of the `config_flags` parameter are shown in the Table 37 below.

Table 37. PMU Counter Config Match Flags

Flag Name	Bits	Description
SBI_PMU_CFG_FLAG_SKIP_MATCH	0:0	Skip the counter matching
SBI_PMU_CFG_FLAG_CLEAR_VALUE	1:1	Clear (or zero) the counter value in counter configuration
SBI_PMU_CFG_FLAG_AUTO_START	2:2	Start the counter after configuring a matching counter
SBI_PMU_CFG_FLAG_SET_VUINH	3:3	Event counting inhibited in VU-mode
SBI_PMU_CFG_FLAG_SET_VSINH	4:4	Event counting inhibited in VS-mode
SBI_PMU_CFG_FLAG_SET_UINH	5:5	Event counting inhibited in U-mode
SBI_PMU_CFG_FLAG_SET_SINH	6:6	Event counting inhibited in S-mode
SBI_PMU_CFG_FLAG_SET_MINH	7:7	Event counting inhibited in M-mode
RESERVED	8:(XLEN-1)	Reserved for future use and must be zero.

When SBI_PMU_CFG_FLAG_SKIP_MATCH *is set in* `config_flags`*, the SBI implementation will unconditionally select the first counter from the set of counters specified by the* `counter_idx_base` *and* `counter_idx_mask`*.*

The SBI_PMU_CFG_FLAG_AUTO_START *flag in* `config_flags` *has no impact on the counter value.*

The `config_flags[3:7]` *bits are event filtering hints so these can be ignored or overridden by*

错误代码	描述
SBI_SUCCESS	计数器信息读取成功。
SBI_ERR_INVALID_PARAM	counter_idx 指向无效计数器。

11.8. 功能函数：查找并配置匹配的计数器（FID #2）struct sbiret sbi_pmu_counter_config_matching(unsigned long counter_idx_base,

```
                                                    unsigned long counter_idx_mask, unsigned
                                                    long config_flags, unsigned long
                                                    event_idx, uint64_t event_data)
```

从一组未启动（或未启用）的计数器中查找并配置一个能够监测指定事件的计数器。参数 counter_idx_base 和 counter_idx_mask 表示计数器集合，event_idx 表示待监测事件，event_data 表示任何额外的事件配置项。参数 config_flags 表示额外的计数器配置和过滤标志位。config_flags 参数的位定义如下表 37 所示。

表 37. PMU 计数器配置匹配标志位

标志名称	位	描述
SBI_PMU_CFG_FLAG_SKIP_MATCH	0:0	跳过计数器匹配
SBI_PMU_CFG_FLAG_CLEAR_VALUE	1:1	清除（或归零）计数器配置中的计数值
SBI_PMU_CFG_FLAG_AUTO_START	2:2	配置匹配计数器后自动启动计数
SBI_PMU_CFG_FLAG_SET_VUINH	3:3	虚拟用户模式下禁止事件计数
SBI_PMU_CFG_FLAG_SET_VSINH	4:4	VS 模式下禁止事件计数
SBI_PMU_CFG_FLAG_SET_UINH	5:5	U 模式下禁止事件计数
SBI_PMU_CFG_FLAG_SET_SINH	6:6	S 模式下禁止事件计数
SBI_PMU_CFG_FLAG_SET_MINH	7:7	M 模式下禁止事件计数
保留字段	8:(XLEN-1)	保留未来使用，必须置零

☒ 当 config_flags 中设置 SBI_PMU_CFG_FLAG_SKIP_MATCH 标志时，SBI 实现将无条件地从 counter_idx_base 和 counter_idx_mask 指定的计数器集合中

选择第一个计数器

☒config_flags 中的 SBI_PMU_CFG_FLAG_AUTO_START 标志不会影响计数器数值

配置标志位[3:7] 这些位是事件过滤提示位，因此可被忽略或覆盖

> *the SBI implementation for security concerns or due to lack of event filtering support in the underlying RISC-V platform.*

Returns the `counter_idx` in `sbiret.value` upon success.

In case of failure, the possible error codes returned in `sbiret.error` are shown in the Table 38 below.

Table 38. PMU Counter Config Match Errors

Error code	Description
SBI_SUCCESS	counter found and configured successfully.
SBI_ERR_INVALID_PARAM	set of counters has at least one invalid counter or the given flag parameter has a reserved bit set.
SBI_ERR_NOT_SUPPORTED	none of the counters can monitor the specified event.

11.9. Function: Start a set of counters (FID #3)

```
struct sbiret sbi_pmu_counter_start(unsigned long counter_idx_base,
                                    unsigned long counter_idx_mask,
                                    unsigned long start_flags,
                                    uint64_t initial_value)
```

Start or enable a set of counters on the calling hart with the specified initial value. The `counter_idx_base` and `counter_idx_mask` parameters represent the set of counters whereas the `initial_value` parameter specifies the initial value of the counter.

The bit definitions of the `start_flags` parameter are shown in the Table 39 below.

Table 39. PMU Counter Start Flags

Flag Name	Bits	Description
SBI_PMU_START_SET_INIT_VALUE	0:0	Set the value of counters based on the `initial_value` parameter
SBI_PMU_START_FLAG_INIT_SNAPSHOT	1:1	Initialize the given counters from shared memory if available.
RESERVED	2:(XLEN-1)	Reserved for future use and must be zero.

> *When* `SBI_PMU_START_SET_INIT_VALUE` *or* `SBI_PMU_START_FLAG_INIT_SNAPSHOT` *is not set in* `start_flags`*, the counter value will not be modified and the event counting will start from the current counter value.*

The shared memory address must be set during boot via `sbi_pmu_snapshot_set_shmem` before the `SBI_PMU_START_FLAG_INIT_SNAPSHOT` flag may be used. The SBI implementation must initialize all the given valid counters (to be started) from the value set in the shared snapshot memory.

> `SBI_PMU_START_SET_INIT_VALUE` *and* `SBI_PMU_START_FLAG_INIT_SNAPSHOT` *are mutually exclusive as the former is only valid for a single counter.*

The possible error codes returned in `sbiret.error` are shown in the Table 40 below.

Table 40. PMU Counter Start Errors

出于安全考虑或底层 RISC-V 平台缺乏事件过滤支持，SBI 实现可能无法提供该功能。

成功时在 sbiret.value 中返回 counter_idx。

失败时，sbiret.error 可能返回的错误代码如下表 38 所示。

表 38. PMU 计数器配置匹配错误

错误代码	描述
SBI_SUCCESS	计数器已找到并成功配置。
SBI_ERR_INVALID_PARAM	计数器集合中至少存在一个无效计数器，或给定的标志参数设置了保留位。
SBI_ERR_NOT_SUPPORTED	所有计数器均无法监控指定事件。

11.9. 功能：启动计数器集合（功能号 #3） struct sbiret sbi_pmu_counter_start(unsigned long counter_idx_base,

```
                                                unsigned long counter_idx_mask, unsigned
                                                long start_flags, uint64_t initial_value)
```

以指定的初始值在调用硬件线程上启动或启用一组计数器。counter_idx_base 和 counter_idx_mask 参数表示计数器集合，而 initial_value 参数则指定计数器的初始值。

start_flags 参数的位定义如下表 39 所示。

表 39. PMU 计数器启动标志

标志名称	位	描述
SBI_PMU_START_SET_INIT_VALUE	0:0	根据计数器设置值 initial_value 参数
SBI_PMU_START_FLAG_INIT_SNAPSHOT 1:1		若共享内存可用，则从中初始化给定计数器。
保留字段	2:(XLEN-1) 保留供	未来使用，必须置零。

SBI_PMU_START_SET_INIT_VALUE 或未设置 SBI_PMU_START_FLAG_INIT_SNAPSHOT 标志

在 start_flags 中，计数器值不会被修改，事件计数将从当前计数器值开始。

共享内存地址必须在启动时通过 sbi_pmu_snapshot_set_shmem 进行设置，之后才能使用
SBI_PMU_START_FLAG_INIT_SNAPSHOT 标志。SBI 实现必须从共享快照内存中设置的值初始化所有要启动的有效计数器。

SBI_PMU_START_SET_INIT_VALUE 和 SBI_PMU_START_FLAG_INIT_SNAPSHOT 是互斥的

前者仅对单一计数器有效，具有排他性。

sbiret.error 可能返回的错误代码如下表 40 所示。

表 40. PMU 计数器启动错误

Error code	Description
SBI_SUCCESS	counter started successfully.
SBI_ERR_INVALID_PARAM	set of counters has at least one invalid counter or the given flag parameter has a reserved bit set.
SBI_ERR_ALREADY_STARTED	set of counters includes at least one counter which is already started.
SBI_ERR_NO_SHMEM	the snapshot shared memory is not available and `SBI_PMU_START_FLAG_INIT_SNAPSHOT` is set in the flags.

11.10. Function: Stop a set of counters (FID #4)

```
struct sbiret sbi_pmu_counter_stop(unsigned long counter_idx_base,
                                   unsigned long counter_idx_mask,
                                   unsigned long stop_flags)
```

Stop or disable a set of counters on the calling hart. The `counter_idx_base` and `counter_idx_mask` parameters represent the set of counters. The bit definitions of the `stop_flags` parameter are shown in the Table 41 below.

Table 41. PMU Counter Stop Flags

Flag Name	Bits	Description
SBI_PMU_STOP_FLAG_RESET	0:0	Reset the counter to event mapping.
SBI_PMU_STOP_FLAG_TAKE_SNAPSHOT	1:1	Save a snapshot of the given counter's values in the shared memory if available.
RESERVED	2:(XLEN-1)	Reserved for future use and must be zero.

The shared memory address must be set during boot via `sbi_pmu_snapshot_set_shmem` before the `SBI_PMU_STOP_FLAG_TAKE_SNAPSHOT` flag may be used. The SBI implementation must save the current value of all the stopped counters in the shared memory if `SBI_PMU_STOP_FLAG_TAKE_SNAPSHOT` is set. The values corresponding to all other counters must not be modified. The SBI implementation must additionally update the overflown counter bitmap in the shared memory.

The possible error codes returned in `sbiret.error` are shown in the Table 42 below.

Table 42. PMU Counter Stop Errors

Error code	Description
SBI_SUCCESS	counter stopped successfully.
SBI_ERR_INVALID_PARAM	set of counters has at least one invalid counter or the given flag parameter has a reserved bit set.
SBI_ERR_ALREADY_STOPPED	set of counters includes at least one counter which is already stopped.
SBI_ERR_NO_SHMEM	the snapshot shared memory is not available and `SBI_PMU_STOP_FLAG_TAKE_SNAPSHOT` is set in the flags.

错误代码	描述
SBI_SUCCESS	计数器启动成功。
SBI_ERR_INVALID_PARAM	计数器集合中包含至少一个无效计数器，或给定的标志参数设置了保留位。
SBI_ERR_ALREADY_STARTED	计数器集合中包含至少一个已启动的计数器。
SBI_ERR_NO_SHMEM	快照共享内存不可用且 在标志位中设置了 SBI_PMU_START_FLAG_INIT_SNAPSHOT

11.10. 功能：停止一组计数器（功能号#4） struct sbiret sbi_pmu_counter_stop(unsigned long counter_idx_base,

```
                                    unsigned long counter_idx_mask, unsigned
                                    long stop_flags)
```

停止或禁用调用硬件线程上的一组计数器。counter_idx_base 和 counter_idx_mask 参数表示计数器集合。stop_flags 参数的位定义如下表 41 所示。

表 41. PMU 计数器停止标志

标志名称	位	描述
SBI_PMU_STOP_FLAG_RESET	0:0	重置计数器与事件的映射关系
SBI_PMU_STOP_FLAG_TAKE_SNAPSHOT 1:1		若共享内存可用，则保存指定计数器的数值快照
保留字段	2:(XLEN-1) 保留未来使用且必须置零	

共享内存地址必须在启动时通过 sbi_pmu_snapshot_set_shmem 设置，之后才能使用

SBI_PMU_STOP_FLAG_TAKE_SNAPSHOT 标志。若设置了 SBI_PMU_STOP_FLAG_TAKE_SNAPSHOT，SBI 实现必须将所有已停止计数器的当前值保存到共享内存中。其他所有计数器的对应值不得修改。SBI 实现还须更新共享内存中的溢出计数器位图。

sbiret.error 可能返回的错误代码如下表 42 所示。

表 42. PMU 计数器停止错误

错误代码	描述
SBI_SUCCESS	计数器已成功停止。
SBI_ERR_INVALID_PARAM	计数器集合中至少包含一个无效计数器，或给定的标志参数设置了保留位。
SBI_ERR_ALREADY_STOPPED	计数器集合中至少包含一个已停止的计数器。
SBI_ERR_NO_SHMEM	快照共享内存不可用且 在标志位中设置了 SBI_PMU_STOP_FLAG_TAKE_SNAPSHOT。

11.11. Function: Read a firmware counter (FID #5)

```
struct sbiret sbi_pmu_counter_fw_read(unsigned long counter_idx)
```

Provide the current firmware counter value in `sbiret.value`. On RV32 systems, the `sbiret.value` will only contain the lower 32 bits of the current firmware counter value.

The possible error codes returned in `sbiret.error` are shown in the Table 43 below.

Table 43. PMU Counter Firmware Read Errors

Error code	Description
SBI_SUCCESS	firmware counter read successfully.
SBI_ERR_INVALID_PARAM	`counter_idx` points to a hardware counter or an invalid counter.

11.12. Function: Read a firmware counter high bits (FID #6)

```
struct sbiret sbi_pmu_counter_fw_read_hi(unsigned long counter_idx)
```

Provide the upper 32 bits of the current firmware counter value in `sbiret.value`. This function always returns zero in `sbiret.value` for RV64 (or higher) systems.

The possible error codes returned in `sbiret.error` are shown in Table 44 below.

Table 44. PMU Counter Firmware Read High Errors

Error code	Description
SBI_SUCCESS	Firmware counter read successfully.
SBI_ERR_INVALID_PARAM	`counter_idx` points to a hardware counter or an invalid counter.

11.13. Function: Set PMU snapshot shared memory (FID #7)

```
struct sbiret sbi_pmu_snapshot_set_shmem(unsigned long shmem_phys_lo,
                                         unsigned long shmem_phys_hi,
                                         unsigned long flags)
```

Set and enable the PMU snapshot shared memory on the calling hart.

If both `shmem_phys_lo` and `shmem_phys_hi` parameters are not all-ones bitwise then `shmem_phys_lo` specifies the lower XLEN bits and `shmem_phys_hi` specifies the upper XLEN bits of the snapshot shared memory physical base address. The `shmem_phys_lo` MUST be 4096 bytes (i.e. page) aligned and the size of the snapshot shared memory must be 4096 bytes. The layout of the snapshot shared memory is described in Table 45.

If both `shmem_phys_lo` and `shmem_phys_hi` parameters are all-ones bitwise then the PMU snapshot shared memory is cleared and disabled.

11.11. 功能：读取固件计数器（功能号 #5）struct sbiret sbi_pmu_counter_fw_read(unsigned long counter_idx)

在 sbiret.value 中返回当前固件计数器的值。对于 RV32 系统，sbiret.value 仅包含当前固件计数器值的低 32 位。

sbiret.error 可能返回的错误代码如下表 43 所示。

表 43. PMU 计数器固件读取错误

错误代码	描述
SBI_SUCCESS	固件计数器读取成功。
SBI_ERR_INVALID_PARAM	counter_idx 指向一个硬件计数器或无效计数器。

11.12. 功能函数：读取固件计数器高位（FID #6）struct sbiret sbi_pmu_counter_fw_read_hi(unsigned long counter_idx)

在 sbiret.value 中返回当前固件计数器值的高 32 位。对于 RV64（或更高位）系统，本函数始终在 sbiret.value 中返回零。

表 44 下方展示了 sbiret.error 可能返回的错误代码。

表 44. PMU 计数器固件读取高位错误

错误代码	描述
SBI_SUCCESS	固件计数器读取成功。
SBI_ERR_INVALID_PARAM	counter_idx 指向硬件计数器或无效计数器。

11.13. 功能：设置 PMU 快照共享内存（功能号#7） struct sbiret sbi_pmu_snapshot_set_shmem(unsigned long shmem_phys_lo, unsigned long shmem_phys_hi, unsigned long flags)

为当前调用硬件线程设置并启用 PMU 快照共享内存。

若 shmem_phys_lo 和 shmem_phys_hi 参数不全为按位全 1 值，则 shmem_phys_lo 指定快照共享内存物理基地址的低 XLEN 位，shmem_phys_hi 指定高 XLEN 位。shmem_phys_lo 必须按 4096 字节（即页）对齐，且快照共享内存大小必须为 4096 字节。快照共享内存的布局详见表 45。
若 shmem_phys_lo 和 shmem_phys_hi 参数均为按位全 1 值，则清除并禁用 PMU 快照共享内存。

The `flags` parameter is reserved for future use and must be zero.

This is an optional function and the SBI implementation may choose not to implement it.

Table 45. SBI PMU Snapshot shared memory layout

Name	Offset	Size	Description
counter_overflow_bitmap	0x0000	8	A bitmap of all logical overflown counters relative to the `counter_idx_base`. This is valid only if the `Sscofpmf` ISA extension is available. Otherwise, it must be zero.
counter_values	0x0008	512	An array of 64-bit logical counters where each index represents the value of each logical counter associated with hardware/firmware relative to the `counter_idx_base`.
Reserved	0x0208	3576	Reserved for future use

Any future revisions to this structure should be made in a backward compatible manner and will be associated with an SBI version.

The logical counter indices in the `counter_overflow_bitmap` and `counter_values` array are relative w.r.t to `counter_idx_base` argument present in the `sbi_pmu_counter_stop` and `sbi_pmu_counter_start` functions. This allows the users to use snapshot feature for more than XLEN counters if required.

This function should be invoked only once per hart at boot time. Once configured, the SBI implementation has read/write access to the shared memory when `sbi_pmu_counter_stop` is invoked with the `SBI_PMU_STOP_FLAG_TAKE_SNAPSHOT` flag set. The SBI implementation has read only access when `sbi_pmu_counter_start` is invoked with the `SBI_PMU_START_FLAG_INIT_SNAPSHOT` flag set. The SBI implementation must not access this memory any other time.

The possible error codes returned in `sbiret.error` are shown in Table 46 below.

Table 46. PMU Setup Snapshot Area Errors

Error code	Description
SBI_SUCCESS	Shared memory was set or cleared successfully.
SBI_ERR_NOT_SUPPORTED	The SBI PMU snapshot functionality is not available in the SBI implementation.
SBI_ERR_INVALID_PARAM	The `flags` parameter is not zero or the `shmem_phys_lo` parameter is not 4096 bytes aligned.
SBI_ERR_INVALID_ADDRESS	The shared memory pointed to by the `shmem_phys_lo` and `shmem_phys_hi` parameters is not writable or does not satisfy other requirements of Section 3.2.
SBI_ERR_FAILED	The request failed for unspecified or unknown other reasons.

11.14. Function: Get PMU Event info (FID #8)

```
struct sbiret sbi_pmu_event_get_info(unsigned long shmem_phys_lo,
                                     unsigned long shmem_phys_hi,
```

flags 参数保留供未来使用，必须设置为零。

这是一个可选功能，SBI 实现可以选择不实现该功能。

表 45. SBI PMU 快照共享内存布局

名称	偏移量	大小	描述
计数器溢出位图	0x0000 8		所有逻辑溢出计数器相对于的位图 counter_idx_base。该字段有效 仅当 Sscofpmf ISA 扩展可用时。否则，该值必须为零。
计数器数值	0x0008 512		一个 64 位逻辑计数器数组，其中每个索引表示相对于 counter_idx_base 的、与硬件/固件相关联的每个逻辑计数器的值。
保留	0x0208 3576		保留供未来使用

未来对该结构的任何修订都应保持向后兼容，并将与 SBI 版本相关联。
counter_overflow_bitmap 和 counter_values 数组中的逻辑计数器索引是相对于

sbi_pmu_counter_stop 和 sbi_pmu_counter_start 函数中存在的 counter_idx_base 参数的。
这允许用户在需要时为超过 XLEN 数量的计数器使用快照功能。

该函数应在每个硬件线程启动时仅调用一次。一旦配置完成，当调用 sbi_pmu_counter_stop 并设置

SBI_PMU_STOP_FLAG_TAKE_SNAPSHOT 标志时，SBI 实现将获得共享内存的读写权限。而当
调用 sbi_pmu_counter_start 时设置了 SBI_PMU_START_FLAG_INIT_SNAPSHOT 标志位。此时 SBI
实现方案在其他任何时间都不得访问该内存区域。

可能通过 sbiret.error 返回的错误代码如下表 46 所示。

表 46. PMU 设置快照区域错误代码

错误代码	描述
SBI_SUCCESS	共享内存设置或清除成功。
SBI_ERR_NOT_SUPPORTED	该 SBI 实现中未提供 PMU 快照功能。
SBI_ERR_INVALID_PARAM	flags 参数非零或 shmem_phys_lo 参数未按 4096 字节对齐。
SBI_ERR_INVALID_ADDRESS	由 shmem_phys_lo 和 shmem_phys_hi 参数指向的共享内存不可写或不满足第 3.2 节的其他要求。
SBI_ERR_FAILED	请求因未指定或其他未知原因失败。

11.14. 功能：获取 PMU 事件信息（功能号#8）结构体 sbiret sbi_pmu_event_get_info(unsigned long shmem_phys_lo,

```
                                unsigned long shmem_phys_hi,
```

```
                                    unsigned long num_entries,
                                    unsigned long flags)
```

Get details about any PMU event via shared memory. The supervisor software can get event specific information for multiple events in one shot by writing an entry for each event in the shared memory. Each entry in the shared memory must be encoded as follows:

Table 47. Event info entry format

Word	Name	ACCESS(SBI Implementation)	Encoding
0	event_idx	RO	BIT[0:19] - Describes the `event_idx` BIT[20:31] - Reserved for the future purpose. Must be zero.
1	output	RW	BIT[0] - Boolean value to indicate `event_idx` is supported or not. The SBI implementation MUST update this entire 32-bit word if valid `event_idx` and `event_data` (if applicable) are specified in the entry. BIT[1:31] - Reserved for the future purpose. Must be zero
2-3	event_data	RO	BIT[0:63] - Valid when `event_idx.type` is either `0x2`, `0x3` or `0xf`. It describes the `event_data` for the specific event specified in `event_idx` if applicable.

The caller must initialize the shared memory and add `num_entries` of each event for which it wishes to discover information about. The `shmem_phys_lo` MUST be 16-byte aligned and the size of the share memory must be (16 * `num_entries`) bytes.

The `flags` parameter is reserved for future use and MUST be zero.

The SBI implementation MUST NOT touch the shared memory once this call returns as supervisor software may free the memory at any time.

The possible error codes returned in `sbiret.error` are shown in Table 48 below.

Table 48. PMU Get Event Info Errors

Error code	Description
SBI_SUCCESS	The output field is updated for each event.
SBI_ERR_NOT_SUPPORTED	The SBI PMU event info retrieval function is not available in the SBI implementation.
SBI_ERR_INVALID_PARAM	The `flags` parameter is not zero or the `shmem_phys_lo` parameter is not 16-bytes aligned or any reserved bit in an `event_idx` word is set.
SBI_ERR_INVALID_ADDRESS	The shared memory pointed to by the `shmem_phys_lo` and `shmem_phys_hi` parameters is not writable or does not satisfy other requirements of Section 3.2.
SBI_ERR_FAILED	The write failed for unspecified or unknown other reasons.

11.15. Function Listing

Table 49. PMU Function List

```
                                        unsigned long num_entries, unsigned
                                        long flags)
```

通过共享内存获取任意 PMU 事件的详细信息。监管者软件可以通过在共享内存中为每个事件写入一个条目，一次性获取多个事件的特定信息。共享内存中的每个条目必须按以下方式编码：

表 47. 事件信息条目格式

字名称		访问(SBI 实现)	编码
0	event_idx 只读	位[0:19] - 描述事件索引	位[20:31] 保留供未来使用。必须置零。
1	输出	读写	位[0] - 布尔值，用于指示是否支持 event_idx。若在入口中指定了有效的 event_idx 和 event_data（如适用），则 SBI 实现必须更新整个 32 位字。位[1:31] - 保留供未来使用，必须置零
2-3	事件数据 只读		BIT[0:63] - 当 event_idx.type 为 0x2、0x3 或 0xf 时有效。若适用，它描述 event_idx 中指定特定事件的 event_data。

调用者必须初始化共享内存，并为每个希望获取信息的事件添加 num_entries 条目。shmem_phys_lo 必须按 16 字节对齐，共享内存的大小必须为(16 * num_entries)字节。

flags 参数保留供未来使用，当前必须置零。

SBI 实现一旦此调用返回就不得再访问共享内存，因为监管者软件可能随时释放该内存。

sbiret.error 可能返回的错误代码如下表 48 所示。

表 48. PMU 获取事件信息错误

错误代码	描述
SBI_SUCCESS	输出字段会针对每个事件进行更新。
SBI_ERR_NOT_SUPPORTED	当前 SBI 实现中未提供 PMU 事件信息检索功能。
SBI_ERR_INVALID_PARAM	flags 参数非零，或 shmem_phys_lo 参数未按 16 字节对齐，或 event_idx 字段中任何保留位被置位。
SBI_ERR_INVALID_ADDRESS（无效地址错误）	由 shmem_phys_lo 和 shmem_phys_hi 参数指向的共享内存不可写入，或不符合第 3.2 节规定的其他要求。
SBI_ERR_FAILED	由于未指定或其他未知原因导致写入失败。

11.15. 函数列表

表 49. PMU 功能列表

Function Name	SBI Version	FID	EID
sbi_pmu_num_counters	0.3	0	0x504D55
sbi_pmu_counter_get_info	0.3	1	0x504D55
sbi_pmu_counter_config_matching	0.3	2	0x504D55
sbi_pmu_counter_start	0.3	3	0x504D55
sbi_pmu_counter_stop	0.3	4	0x504D55
sbi_pmu_counter_fw_read	0.3	5	0x504D55
sbi_pmu_counter_fw_read_hi	2.0	6	0x504D55
sbi_pmu_snapshot_set_shmem	2.0	7	0x504D55
sbi_pmu_event_get_info	3.0	8	0x504D55

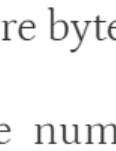
功能名称	SBI 版本 FID EID		
sbi_pmu_num_counters	0.3	0 0x504D55	
sbi_pmu_counter_get_info	0.3	1	0x504D55
sbi_pmu_counter_config_matching	0.3	2	0x504D55
sbi_pmu_counter_start	0.3	3	0x504D55
sbi_pmu_counter_stop	0.3	4	0x504D55
sbi_pmu_counter_fw_read	0.3	5	0x504D55
sbi_pmu_counter_fw_read_hi	2.0	6 0x504D55	
sbi_pmu_snapshot_set_shmem	2.0	7	0x504D55
sbi_pmu_event_get_info	3.0	8	0x504D55

Chapter 12. Debug Console Extension (EID #0x4442434E "DBCN")

The debug console extension defines a generic mechanism for debugging and boot-time early prints from supervisor-mode software.

This extension replaces the legacy console putchar (EID #0x01) and console getchar (EID #0x02) extensions. The debug console extension allows supervisor-mode software to write or read multiple bytes in a single SBI call.

If the underlying physical console has extra bits for error checking (or correction) then these extra bits should be handled by the SBI implementation.



It is recommended that bytes sent/received using the debug console extension follow UTF-8 character encoding.

12.1. Function: Console Write (FID #0)

```
struct sbiret sbi_debug_console_write(unsigned long num_bytes,
                                      unsigned long base_addr_lo,
                                      unsigned long base_addr_hi)
```

Write bytes to the debug console from input memory.

The `num_bytes` parameter specifies the number of bytes in the input memory. The physical base address of the input memory is represented by two XLEN bits wide parameters. The `base_addr_lo` parameter specifies the lower XLEN bits and the `base_addr_hi` parameter specifies the upper XLEN bits of the input memory physical base address.

This is a non-blocking SBI call and it may do partial/no writes if the debug console is not able to accept more bytes.

The number of bytes written is returned in `sbiret.uvalue` and the possible error codes returned in `sbiret.error` are shown in Table 50 below.

Table 50. Debug Console Write Errors

Error code	Description
SBI_SUCCESS	Bytes written successfully.
SBI_ERR_INVALID_PARAM	The memory pointed to by the `num_bytes`, `base_addr_lo`, and `base_addr_hi` parameters does not satisfy the requirements described in the Section 3.2
SBI_ERR_DENIED	Writes to the debug console is not allowed.
SBI_ERR_FAILED	Failed to write due to I/O errors.

12.2. Function: Console Read (FID #1)

```
struct sbiret sbi_debug_console_read(unsigned long num_bytes,
                                     unsigned long base_addr_lo,
```

第 12 章 调试控制台扩展（EID #0x4442434E "DBCN"）

调试控制台扩展定义了一种通用机制，用于调试和监督模式软件的启动早期输出。
该扩展取代了传统的控制台输出字符(EID #0x01)和控制台输入字符(EID #0x02)扩展。调试控制台扩展允许监督模式软件在单个 SBI 调用中写入或读取多个字节。
如果底层物理控制台具有用于错误检查(或纠正)的额外比特位，则这些额外比特位应由 SBI 实现处理。

建议使用调试控制台扩展发送/接收的字节遵循 UTF-8 字符编码规范。

12.1. 功能：控制台写入（功能 ID #0） struct sbiret sbi_debug_console_write(unsigned long num_bytes,

```
                                                    unsigned long base_addr_lo, unsigned
                                                    long base_addr_hi)
```

将输入内存中的字节写入调试控制台。

num_bytes 参数指定输入内存中的字节数。输入内存的物理基地址由两个 XLEN 位宽参数表示。base_addr_lo 参数指定输入内存物理基地址的低 XLEN 位，base_addr_hi 参数指定输入内存物理基地址的高 XLEN 位。
这是一个非阻塞的 SBI 调用，如果调试控制台无法接收更多字节，它可能会执行部分写入或完全不写入。
实际写入的字节数通过 sbiret.uvalue 返回，可能的错误代码

sbiret.error 如下表 50 所示。

表 50. 调试控制台写入错误

错误代码	描述
SBI_SUCCESS	成功写入的字节数。
SBI_ERR_INVALID_PARAM	由 num_bytes、base_addr_lo 和 base_addr_hi 参数指向的内存区域不符合第 3.2 节所述的要求。
SBI_ERR_DENIED（访问被拒绝）	不允许写入调试控制台。
SBI_ERR_FAILED	因 I/O 错误导致写入失败。

12.2. 功能：控制台读取（功能 ID #1）
结构体 sbiret sbi_debug_console_read(无符号长整型 num_bytes,

```
                                                    无符号长整型 base_addr_lo,
```

```
                                    unsigned long base_addr_hi)
```

Read bytes from the debug console into an output memory.

The **num_bytes** parameter specifies the maximum number of bytes which can be written into the output memory. The physical base address of the output memory is represented by two XLEN bits wide parameters. The **base_addr_lo** parameter specifies the lower XLEN bits and the **base_addr_hi** parameter specifies the upper XLEN bits of the output memory physical base address.

This is a non-blocking SBI call and it will not write anything into the output memory if there are no bytes to be read in the debug console.

The number of bytes read is returned in **sbiret.uvalue** and the possible error codes returned in **sbiret.error** are shown in Table 51 below.

Table 51. Debug Console Read Errors

Error code	Description
SBI_SUCCESS	Bytes read successfully.
SBI_ERR_INVALID_PARAM	The memory pointed to by the **num_bytes**, **base_addr_lo**, and **base_addr_hi** parameters does not satisfy the requirements described in the Section 3.2
SBI_ERR_DENIED	Reads from the debug console is not allowed.
SBI_ERR_FAILED	Failed to read due to I/O errors.

12.3. Function: Console Write Byte (FID #2)

```
struct sbiret sbi_debug_console_write_byte(uint8_t byte)
```

Write a single byte to the debug console.

This is a blocking SBI call and it will only return after writing the specified byte to the debug console. It will also return, with SBI_ERR_FAILED, if there are I/O errors.

The **sbiret.uvalue** is set to zero and the possible error codes returned in **sbiret.error** are shown in Table 52 below.

Table 52. Debug Console Write Byte Errors

Error code	Description
SBI_SUCCESS	Byte written successfully.
SBI_ERR_DENIED	Write to the debug console is not allowed.
SBI_ERR_FAILED	Failed to write the byte due to I/O errors.

12.4. Function Listing

Table 53. DBCN Function List

```
                                                    unsigned long base_addr_hi)
```

从调试控制台读取字节到输出内存中。

num_bytes 参数指定了可写入输出内存的最大字节数。输出内存的物理基地址由两个 XLEN 位宽的参数表示。base_addr_lo 参数指定输出内存物理基地址的低 XLEN 位，base_addr_hi 参数指定高 XLEN 位。
这是一个非阻塞的 SBI 调用，如果调试控制台中没有可读取的字节，则不会向输出内存写入任何内容。
实际读取的字节数通过 sbiret.uvalue 返回，可能的错误代码通过

sbiret.error 如下表 51 所示。

表 51. 调试控制台读取错误

错误代码	描述
SBI_SUCCESS	成功读取的字节数。
SBI_ERR_INVALID_PARAM	由 num_bytes、base_addr_lo 和 base_addr_hi 参数指向的内存区域不符合第 3.2 节所述的要求。
SBI_ERR_DENIED（访问被拒绝）	禁止从调试控制台读取。
SBI_ERR_FAILED	因 I/O 错误导致读取失败。

12.3. 功能：控制台写入字节（功能 ID #2） struct sbiret sbi_debug_console_write_byte(uint8_t byte)

```

```

向调试控制台写入单个字节。

这是一个阻塞式 SBI 调用，只有在将指定字节写入调试控制台后才会返回。如果出现 I/O 错误，也会返回 SBI_ERR_FAILED 错误码。
sbiret.uvalue 被置为零，sbiret.error 可能返回的错误码如下表 52 所示。

表 52. 调试控制台写字节错误码

错误代码	描述
SBI_SUCCESS	字节写入成功。
SBI_ERR_DENIED（访问被拒绝）	不允许写入调试控制台。
SBI_ERR_FAILED	由于 I/O 错误导致字节写入失败。

12.4. 函数列表

表 53. DBCN 函数列表

Function Name	SBI Version	FID	EID
sbi_debug_console_write	2.0	0	0x4442434E
sbi_debug_console_read	2.0	1	0x4442434E
sbi_debug_console_write_byte	2.0	2	0x4442434E

功能名称	SBI 版本 FID EID		
sbi_debug_console_write	2.0	0 0x4442434E	
sbi_debug_console_read	2.0	1	0x4442434E
sbi_debug_console_write_byte	2.0	2	0x4442434E

Chapter 13. System Suspend Extension (EID #0x53555350 "SUSP")

The system suspend extension defines a set of system-level sleep states and a function which allows the supervisor-mode software to request that the system transitions to a sleep state. Sleep states are identified with 32-bit wide identifiers (`sleep_type`). The possible values for the identifiers are shown in Table 54.

The term "system" refers to the world-view of the supervisor software domain invoking the call. System suspend may only suspend the part of the overall system which is visible to the invoking supervisor software domain.

The system suspend extension does not provide any way for supported sleep types to be probed. Platforms are expected to specify their supported system sleep types and per-type wake up devices in their hardware descriptions. The `SUSPEND_TO_RAM` sleep type is the one exception, and its presence is implied by that of the extension.

Table 54. SUSP System Sleep Types

Type	Name	Description
0	SUSPEND_TO_RAM	This is a “suspend to RAM” sleep type, similar to ACPI’s S2 or S3. Entry requires all but the calling hart be in the HSM `STOPPED` state and all hart registers and CSRs saved to RAM.
0x00000001 - 0x7fffffff		Reserved for future use
0x80000000 - 0xffffffff		Platform-specific system sleep types

13.1. Function: System Suspend (FID #0)

```
struct sbiret sbi_system_suspend(uint32_t sleep_type,
                                 unsigned long resume_addr,
                                 unsigned long opaque)
```

A return from a `sbi_system_suspend()` call implies an error and an error code from Table 56 will be in `sbiret.error`. A successful suspend and wake up, results in the hart which initiated the suspend, resuming from the `STOPPED` state. To resume, the hart will jump to supervisor-mode, at the address specified by `resume_addr`, with the specific register values described in Table 55.

Table 55. SUSP System Resume Register State

Register Name	Register Value
satp	0
sstatus.SIE	0
a0	hartid
a1	`opaque` parameter
All other registers remain in an undefined state.	

A single `unsigned long` parameter is sufficient for `resume_addr`, because the hart will resume execution in supervisor-mode with the MMU off, hence `resume_addr` must be less

第 13 章. 系统挂起扩展（扩展 ID #0x53555350 "SUSP"）

系统休眠扩展定义了一组系统级睡眠状态及一个功能函数，允许监管模式软件请求系统进入睡眠状态。睡眠状态通过 32 位宽标识符（sleep_type）进行标识，其可能取值如表 54 所示。

术语"系统"特指发起调用的监管软件域所观察到的世界范围。系统休眠可能仅会暂停发起调用的监管软件域可见的那部分整体系统。

系统休眠扩展未提供任何探测支持的休眠类型的方法。平台应在其硬件描述中指定其支持的系统休眠类型及每种类型的唤醒设备。SUSPEND_TO_RAM 休眠类型是个例外，其存在性由该扩展隐含表明。

表 54. SUSP 系统休眠类型

类型	名称	描述
0	SUSPEND_TO_RAM	这是一种"挂起到内存"的睡眠类型，类似于 ACPI 的 S2 或 S3 状态。进入此状态要求除调用该功能的硬件线程外，所有其他硬件线程必须处于 HSM STOPPED 状态，且所有硬件线程寄存器及 CSR 需保
0x00000001 - 0x7fffffff		保留供未来使用
0x80000000 - 0xffffffff		平台特定的系统休眠类型

13.1. 功能：系统挂起（功能号 #0） struct sbiret sbi_system_suspend(uint32_t sleep_type,

```
                                    unsigned long resume_addr, unsigned
                                    long opaque)
```

从 sbi_system_suspend()调用返回意味着出现错误，sbiret.error 中将包含表 56 中的错误代码。

成功挂起并唤醒后，发起挂起的硬件线程将从 STOPPED 状态恢复。恢复时，该硬件线程将跳转到监督模式，地址由

恢复地址 ，具体寄存器值如表 55 所述。

表 55. SUSP 系统恢复寄存器状态

寄存器名称	寄存器值
satp	0
sstatus.SIE	0
a0	硬件线程 ID
寄存器 a1	不透明参数
其余所有寄存器均处于未定义状态。	

- 无符号长整型参数对于 resume_addr 已足够，因为硬件线程将在 MMU 关闭的监管模式下恢复执行，因此 resume_addr 必须小于

> *than XLEN bits wide.*

The `resume_addr` parameter points to a runtime-specified physical address, where the hart can resume execution in supervisor-mode after a system suspend.

The `opaque` parameter is an XLEN-bit value which will be set in the `a1` register when the hart resumes execution at `resume_addr` after a system suspend.

Besides ensuring all entry criteria for the selected sleep type are met, such as ensuring other harts are in the `STOPPED` state, the caller must ensure all power units and domains are in a state compatible with the selected sleep type. The preparation of the power units, power domains, and wake-up devices used for resumption from the system sleep state is platform specific and beyond the scope of this specification.

When supervisor software is running inside a virtual machine, the SBI implementation is provided by a hypervisor. System suspend will behave similarly to the native case from the point of view of the supervisor software.

The possible error codes returned in `sbiret.error` are shown in Table 56.

Table 56. SUSP System Suspend Errors

Error code	Description
SBI_ERR_INVALID_PARAM	`sleep_type` is reserved or is platform-specific and unimplemented.
SBI_ERR_NOT_SUPPORTED	`sleep_type` is not reserved and is implemented, but the platform does not support it due to one or more missing dependencies.
SBI_ERR_INVALID_ADDRESS	`resume_addr` is not valid, possibly due to the following reasons: * It is not a valid physical address. * Executable access to the address is prohibited by a physical memory protection mechanism or H-extension G-stage for supervisor mode.
SBI_ERR_DENIED	The suspend request failed due to unsatisfied entry criteria.
SBI_ERR_FAILED	The suspend request failed for unspecified or unknown other reasons.

13.2. Function Listing

Table 57. SUSP Function List

Function Name	SBI Version	FID	EID
sbi_system_suspend	2.0	0	0x53555350

> 超过 XLEN 位宽。

resume_addr 参数指向一个运行时指定的物理地址，在该地址处 hart 可以在系统挂起后以监管模式恢复执行。
不透明参数是一个 XLEN 位值，当硬件线程在系统挂起后于 resume_addr 地址恢复执行时，该值将被设置到 a1 寄存器中。
除了确保满足所选休眠类型的所有进入条件（例如确保其他硬件线程处于 STOPPED 状态），调用者还必须确保所有电源单元和电源域都处于与所选休眠类型兼容的状态。用于从系统休眠状态恢复的电源单元、电源域以及唤醒设备的准备工作是平台特定的，不在本规范范围内。
当监督模式软件在虚拟机内运行时，SBI 实现由虚拟机监控程序提供。从监督模式软件的角度来看，系统挂起的行为将与原生情况类似。

sbiret.error 中可能返回的错误代码如表 56 所示。

表 56. SUSP 系统挂起错误类型

错误代码	描述
SBI_ERR_INVALID_PARAM	sleep_type 为保留值或平台特定未实现类型
SBI_ERR_NOT_SUPPORTED	sleep_type 非保留且已实现，但因缺少一个或多个依赖项导致平台不支持该类型
SBI_ERR_INVALID_ADDRESS resume_addr 地址无效，可能原因包括：* 非有效物理地址 * 物理内存保护机制或 H 扩展 G 级监管模式禁止对该地址进行可执行访问。	
SBI_ERR_DENIED（访问被拒绝）	由于不满足进入条件，挂起请求失败。
SBI_ERR_FAILED	由于未指定或其他未知原因，挂起请求失败。

13.2. 功能列表

表 57. SUSP 功能列表

功能名称	SBI 版本	功能标识符	扩展 ID
sbi_system_suspend	2.0	0	0x53555350

Chapter 14. CPPC Extension (EID #0x43505043 "CPPC")

ACPI defines the Collaborative Processor Performance Control (CPPC) mechanism, which is an abstract and flexible mechanism for the supervisor-mode power-management software to collaborate with an entity in the platform to manage the performance of the processors.

The SBI CPPC extension provides an abstraction to access the CPPC registers through SBI calls. The CPPC registers can be memory locations shared with a separate platform entity such as a BMC. Even though CPPC is defined in the ACPI specification, it may be possible to implement a CPPC driver based on Device Tree.

Table 58 defines 32-bit identifiers for all CPPC registers to be used by the SBI CPPC functions. The first half of the 32-bit register space corresponds to the registers as defined by the ACPI specification. The second half provides the information not defined in the ACPI specification, but is additionally required by the supervisor-mode power-management software.

Table 58. CPPC Registers

Register ID	Register	Bit Width	Attribute	Description
0x00000000	HighestPerformance	32	Read-only	ACPI Spec 6.5: 8.4.6.1.1.1
0x00000001	NominalPerformance	32	Read-only	ACPI Spec 6.5: 8.4.6.1.1.2
0x00000002	LowestNonlinearPerformance	32	Read-only	ACPI Spec 6.5: 8.4.6.1.1.4
0x00000003	LowestPerformance	32	Read-only	ACPI Spec 6.5: 8.4.6.1.1.5
0x00000004	GuaranteedPerformanceRegister	32	Read-only	ACPI Spec 6.5: 8.4.6.1.1.6
0x00000005	DesiredPerformanceRegister	32	Read / Write	ACPI Spec 6.5: 8.4.6.1.2.3
0x00000006	MinimumPerformanceRegister	32	Read / Write	ACPI Spec 6.5: 8.4.6.1.2.2
0x00000007	MaximumPerformanceRegister	32	Read / Write	ACPI Spec 6.5: 8.4.6.1.2.1
0x00000008	PerformanceReductionTolerance Register	32	Read / Write	ACPI Spec 6.5: 8.4.6.1.2.4
0x00000009	TimeWindowRegister	32	Read / Write	ACPI Spec 6.5: 8.4.6.1.2.5
0x0000000A	CounterWraparoundTime	32 / 64	Read-only	ACPI Spec 6.5: 8.4.6.1.3.1
0x0000000B	ReferencePerformanceCounterRe gister	32 / 64	Read-only	ACPI Spec 6.5: 8.4.6.1.3.1
0x0000000C	DeliveredPerformanceCounterRe gister	32 / 64	Read-only	ACPI Spec 6.5: 8.4.6.1.3.1
0x0000000D	PerformanceLimitedRegister	32	Read / Write	ACPI Spec 6.5: 8.4.6.1.3.2
0x0000000E	CPPCEnableRegister	32	Read / Write	ACPI Spec 6.5: 8.4.6.1.4
0x0000000F	AutonomousSelectionEnable	32	Read / Write	ACPI Spec 6.5: 8.4.6.1.5

第 14 章 CPPC 扩展（EID #0x43505043 "CPPC"）

ACPI 定义了协作处理器性能控制（CPPC）机制，这是一种抽象且灵活的机制，允许监管模式电源管理软件与平台中的实体协作来管理处理器性能。

SBI CPPC 扩展提供了通过 SBI 调用访问 CPPC 寄存器的抽象层。这些 CPPC 寄存器可以是与独立平台实体（如 BMC）共享的内存位置。尽管 CPPC 在 ACPI 规范中定义，但基于设备树实现 CPPC 驱动程序也是可行的。

表 58 定义了 SBI CPPC 功能使用的所有 CPPC 寄存器的 32 位标识符。32 位寄存器空间的前半部分对应 ACPI 规范定义的寄存器，后半部分则提供了 ACPI 规范未定义但监管模式电源管理软件所需的额外信息。

表 58. CPPC 寄存器

寄存器编号	寄存器	位宽	属性	说明
0x00000000	最高性能	32	只读	ACPI 规范 6.5 版：8.4.6.1.1.1
0x00000001	标称性能	32	只读	ACPI 规范 6.5 版：8.4.6.1.1.2
0x00000002	最低非线性性能 32		只读	ACPI 规范 6.5: 8.4.6.1.1.4
0x00000003	最低性能	32	只读	ACPI 规范 6.5 版：8.4.6.1.1.5
0x00000004	保证性能寄存器 32 位		只读	ACPI 规范 6.5 版：8.4.6.1.1.6
0x00000005	期望性能寄存器 32 位		读取/写入	ACPI 规范 6.5 版：8.4.6.1.2.3
0x00000006	最低性能寄存器 32 位		读取 / 写入	ACPI 规范 6.5 版：8.4.6.1.2.2
0x00000007	最大性能寄存器 32		读取 / 写入	ACPI 规范 6.5 版：8.4.6.1.2.1
0x00000008	性能降级容忍度寄存器	32	读取 / 写入	ACPI 规范 6.5 版 8.4.6.1.2.4
0x00000009	时间窗口寄存器	32	读取/写入	ACPI 规范 6.5 版：8.4.6.1.2.5
0x0000000A	计数器回绕时间	32 / 64	只读	ACPI 规范 6.5：8.4.6.1.3.1
0x0000000B	参考性能计数器寄存器	32 / 64	只读	ACPI 规范 6.5 版：8.4.6.1.3.1
0x0000000C	性能计数器交付寄存器	32 / 64	只读	ACPI 规范 6.5 版：8.4.6.1.3.1
0x0000000D	性能受限寄存器 32		读取 / 写入	ACPI 规范 6.5 版：8.4.6.1.3.2
0x0000000E	CPPC 使能寄存器	32	读取 / 写入	ACPI 规范 6.5 版：8.4.6.1.4
0x0000000F	自主选择使能 32		读取 / 写入	ACPI 规范 6.5 版：8.4.6.1.5

Register ID	Register	Bit Width	Attribute	Description
0x00000010	AutonomousActivityWindowRegister	32	Read / Write	ACPI Spec 6.5: 8.4.6.1.6
0x00000011	EnergyPerformancePreferenceRegister	32	Read / Write	ACPI Spec 6.5: 8.4.6.1.7
0x00000012	ReferencePerformance	32	Read-only	ACPI Spec 6.5: 8.4.6.1.1.3
0x00000013	LowestFrequency	32	Read-only	ACPI Spec 6.5: 8.4.6.1.1.7
0x00000014	NominalFrequency	32	Read-only	ACPI Spec 6.5: 8.4.6.1.1.7
0x00000015 - 0x7FFFFFFF				Reserved for future use.
0x80000000	TransitionLatency	32	Read-only	Provides the maximum (worst-case) performance state transition latency in nanoseconds.
0x80000001 - 0xFFFFFFFF				Reserved for future use.

14.1. Function: Probe CPPC register (FID #0)

```
struct sbiret sbi_cppc_probe(uint32_t cppc_reg_id)
```

Probe whether the CPPC register as specified by the **cppc_reg_id** parameter is implemented or not by the platform.

If the register is implemented, **sbiret.value** will contain the register width. If the register is not implemented, **sbiret.value** will be set to 0.

The possible error codes returned in **sbiret.error** are shown in Table 59.

Table 59. CPPC Probe Errors

Error code	Description
SBI_SUCCESS	Probe completed successfully.
SBI_ERR_INVALID_PARAM	**cppc_reg_id** is reserved.
SBI_ERR_FAILED	The probe request failed for unspecified or unknown other reasons.

14.2. Function: Read CPPC register (FID #1)

```
struct sbiret sbi_cppc_read(uint32_t cppc_reg_id)
```

Reads the register as specified in the **cppc_reg_id** parameter and returns the value in **sbiret.value**. When supervisor mode XLEN is 32, the **sbiret.value** will only contain the lower 32 bits of the CPPC register

寄存器 ID	寄存器	位宽	属性	描述
0x00000010	自主活动窗口寄存器	32	读取 / 写入	ACPI 规范 6.5 版：8.4.6.1.6
0x00000011	能源性能偏好寄存器	32	读取 / 写入	ACPI 规范 6.5 版：8.4.6.1.7
0x00000012	参考性能	32	只读	ACPI 规范 6.5 版：8.4.6.1.1.3
0x00000013	最低频率	32	只读	ACPI 规范 6.5 版：8.4.6.1.1.7
0x00000014	标称频率	32	只读	ACPI 规范 6.5 版：8.4.6.1.1.7
0x00000015 - 0x7FFFFFFF				保留供未来使用使用
0x80000000	转换延迟	32	只读	提供最差（最坏）情况下的性能状态转换延迟纳秒。
0x80000001 - 0xFFFFFFFF				保留供未来使用使用。

14.1. 功能：探测 CPPC 寄存器（功能号#0） struct sbiret sbi_cppc_probe(uint32_t cppc_reg_id)

探测平台是否实现了由 cppc_reg_id 参数指定的 CPPC 寄存器。
若该寄存器已实现，sbiret.value 将包含寄存器位宽；若未实现，sbiret.value 将被置为 0。

sbiret.error 可能返回的错误代码如表 59 所示。

表 59. CPPC 探测错误

错误代码	描述
SBI_SUCCESS	探测成功完成。
SBI_ERR_INVALID_PARAM	cppc_reg_id 为保留字段。
SBI_ERR_FAILED	探测请求因未指定或其他未知原因失败。

14.2. 功能：读取 CPPC 寄存器（功能号#1）struct sbiret sbi_cppc_read(uint32_t cppc_reg_id)

读取 cppc_reg_id 参数指定的寄存器，并通过 sbiret.value 返回其值。当监管模式 XLEN 为 32 位时，sbiret.value 仅包含 CPPC 寄存器的低 32 位

value.

The possible error codes returned in `sbiret.error` are shown in Table 60.

Table 60. CPPC Read Errors

Error code	Description
SBI_SUCCESS	Read completed successfully.
SBI_ERR_INVALID_PARAM	`cppc_reg_id` is reserved.
SBI_ERR_NOT_SUPPORTED	`cppc_reg_id` is not implemented by the platform.
SBI_ERR_DENIED	`cppc_reg_id` is a write-only register.
SBI_ERR_FAILED	The read request failed for unspecified or unknown other reasons.

14.3. Function: Read CPPC register high bits (FID #2)

```
struct sbiret sbi_cppc_read_hi(uint32_t cppc_reg_id)
```

Reads the upper 32-bit value of the register specified in the `cppc_reg_id` parameter and returns the value in `sbiret.value`. This function always returns zero in `sbiret.value` when supervisor mode XLEN is 64 or higher.

The possible error codes returned in `sbiret.error` are shown in Table 61.

Table 61. CPPC Read Hi Errors

Error code	Description
SBI_SUCCESS	Read completed successfully.
SBI_ERR_INVALID_PARAM	`cppc_reg_id` is reserved.
SBI_ERR_NOT_SUPPORTED	`cppc_reg_id` is not implemented by the platform.
SBI_ERR_DENIED	`cppc_reg_id` is a write-only register.
SBI_ERR_FAILED	The read request failed for unspecified or unknown other reasons.

14.4. Function: Write to CPPC register (FID #3)

```
struct sbiret sbi_cppc_write(uint32_t cppc_reg_id, uint64_t val)
```

Writes the value passed in the `val` parameter to the register as specified in the `cppc_reg_id` parameter.

The possible error codes returned in `sbiret.error` are shown in Table 62.

Table 62. CPPC Write Errors

Error code	Description
SBI_SUCCESS	Write completed successfully.
SBI_ERR_INVALID_PARAM	`cppc_reg_id` is reserved.
SBI_ERR_NOT_SUPPORTED	`cppc_reg_id` is not implemented by the platform.

值。

sbiret.error 可能返回的错误代码如表 60 所示。

表 60. CPPC 读取错误

错误代码	描述
SBI_SUCCESS	读取成功完成。
SBI_ERR_INVALID_PARAM	cppc_reg_id 保留未使用
SBI_ERR_NOT_SUPPORTED	cppc_reg_id 平台未实现该功能
SBI_ERR_DENIED（访问被拒绝）	cppc_reg_id 是一个只写寄存器。
SBI_ERR_FAILED	读取请求因未指定或其他未知原因失败。

14.3. 功能：读取 CPPC 寄存器高 32 位（功能号#2） struct sbiret sbi_cppc_read_hi(uint32_t cppc_reg_id)

读取 cppc_reg_id 参数指定寄存器的高 32 位值，并通过 sbiret.value 返回该值。当监管模式 XLEN 为 64 位或更高时，该函数始终在 sbiret.value 中返回零。

sbiret.error 可能返回的错误代码如表 61 所示。

表 61. CPPC 读取高位错误码

错误代码	描述
SBI_SUCCESS	读取成功完成。
SBI_ERR_INVALID_PARAM	cppc_reg_id 为保留字段。
SBI_ERR_NOT_SUPPORTED	平台未实现 cppc_reg_id。
SBI_ERR_DENIED（访问被拒绝）	cppc_reg_id 是一个只写寄存器。
SBI_ERR_FAILED	读取请求因未指定或其他未知原因失败。

14.4. 功能：写入 CPPC 寄存器（功能 ID #3） struct sbiret sbi_cppc_write(uint32_t cppc_reg_id, uint64_t val)

将 val 参数中的值写入 cppc_reg_id 参数指定的寄存器。

sbiret.error 中可能返回的错误代码如表 62 所示。

表 62. CPPC 写入错误

错误代码	描述
SBI_SUCCESS	写入成功完成。
SBI_ERR_INVALID_PARAM	cppc_reg_id 被保留。
SBI_ERR_NOT_SUPPORTED	cppc_reg_id 未被平台实现。

Error code	Description
SBI_ERR_DENIED	`cppc_reg_id` is a read-only register.
SBI_ERR_FAILED	The write request failed for unspecified or unknown other reasons.

14.5. Function Listing

Table 63. CPPC Function List

Function Name	SBI Version	FID	EID
sbi_cppc_probe	2.0	0	0x43505043
sbi_cppc_read	2.0	1	0x43505043
sbi_cppc_read_hi	2.0	2	0x43505043
sbi_cppc_write	2.0	3	0x43505043

错误代码	描述
SBI_ERR_DENIED（访问被拒绝）	cppc_reg_id 是一个只读寄存器。
SBI_ERR_FAILED	写入请求因未指定或其他未知原因失败。

14.5. 函数列表

表 63. CPPC 函数列表

功能名称	SBI 版本	功能标识符	扩展 ID
sbi_cppc_probe	2.0	0	0x43505043
sbi_cppc_read	2.0	1	0x43505043
sbi_cppc_read_hi	2.0	2	0x43505043
sbi_cppc_write	2.0	3	0x43505043

Chapter 15. Nested Acceleration Extension (EID #0x4E41434C "NACL")

Nested virtualization is the ability of a hypervisor to run another hypervisor as a guest. RISC-V nested virtualization requires an L0 hypervisor (running in hypervisor-mode) to trap-and-emulate the RISC-V H-extension [1] functionality (such as CSR accesses, HFENCE instructions, HLV/HSV instructions, etc.) for the L1 hypervisor (running in virtualized supervisor-mode).

The SBI nested acceleration extension defines a shared memory based interface between the SBI implementation (or L0 hypervisor) and the supervisor software (or L1 hypervisor) which allows both to collaboratively reduce traps taken by the L0 hypervisor for emulating RISC-V H-extension functionality. The nested acceleration shared memory allows the L1 hypervisor to batch multiple RISC-V H-extension CSR accesses and HFENCE requests which are then emulated by the L0 hypervisor upon an explicit synchronization SBI call.

The M-mode firmware should not implement the SBI nested acceleration extension if the underlying platform has the RISC-V H-extension implemented in hardware.

This SBI extension defines optional features which MUST be discovered by the supervisor software (or L1 hypervisor) before using the corresponding SBI functions. Each nested acceleration feature is assigned a unique ID which is an unsigned 32-bit integer. The Table 64 below provides a list of all nested acceleration features.

Table 64. Nested acceleration features

Feature ID	Feature Name	Description
0x00000000	SBI_NACL_FEAT_SYNC_CSR	Synchronize CSR
0x00000001	SBI_NACL_FEAT_SYNC_HFENCE	Synchronize HFENCE
0x00000002	SBI_NACL_FEAT_SYNC_SRET	Synchronize SRET
0x00000003	SBI_NACL_FEAT_AUTOSWAP_CSR	Autoswap CSR
> 0x00000003	RESERVED	Reserved for future use

To use the SBI nested acceleration extension, the supervisor software (or L1 hypervisor) MUST set up a nested acceleration shared memory physical address for each virtual hart at boot-time. The physical base address of the nested acceleration shared memory MUST be 4096 bytes (i.e. page) aligned and the size of the nested acceleration shared memory must be `4096 + (1024 * (XLEN / 8))` bytes. The Table 65 below shows the layout of nested acceleration shared memory.

Table 65. Nested acceleration shared memory layout

Name	Offset	Size (bytes)	Description
Scratch space	0x00000000	4096	Nested acceleration feature specific data.
CSR space	0x00001000	`XLEN * 128`	An array of 1024 XLEN-bit words where each word corresponds to a possible RISC-V H-extension CSR defined in the Table 2.1 of the RISC-V privileged specification [1].

Any nested acceleration feature may define the contents of the scratch space shown in the Table 65 above if required.

The contents of the CSR space shown in the Table 65 above is an array of RISC-V H-extension CSR values where CSR `<x>` is at index `<i> = ((<x> & 0xc00) >> 2) | (<x> & 0xff)`. The SBI implementation (or L0 hypervisor) MUST update the CSR space whenever the state of any RISC-V H-extension CSR changes

第 15 章 嵌套加速扩展（扩展 ID #0x4E41434C "NACL"）

嵌套虚拟化是指一个虚拟机监控程序（hypervisor）能够作为客户机运行另一个虚拟机监控程序的能力。RISC-V 的嵌套虚拟化要求运行在监管者模式（hypervisor-mode）的 L0 层虚拟机监控程序，能够为运行在虚拟化监管者模式（virtualized supervisor-mode）的 L1 层虚拟机监控程序捕获并模拟 RISC-V H 扩展[1]功能（例如 CSR 访问、HFENCE 指令、HLV/HSV 指令等）。

SBI 嵌套加速扩展定义了 SBI 实现（或 L0 层虚拟机监控程序）与监管者软件（或 L1 层虚拟机监控程序）之间基于共享内存的接口。该接口允许双方协同减少 L0 层虚拟机监控程序为模拟 RISC-V H 扩展功能而产生的捕获操作。嵌套加速共享内存使得 L1 层虚拟机监控程序能够批量处理多个 RISC-V H 扩展的 CSR 访问和 HFENCE 请求，这些请求随后通过显式同步 SBI 调用由 L0 层虚拟机监控程序进行模拟。

如果底层平台已通过硬件实现了 RISC-V H 扩展，则 M 模式固件不应实现 SBI 嵌套加速扩展。

该 SBI 扩展定义了可选功能，监督模式软件（或 L1 虚拟机监控程序）在使用相应 SBI 函数前必须发现这些功能。每个嵌套加速功能都被分配了一个唯一的无符号 32 位整数 ID。下表 64 列出了所有嵌套加速功能。

表 64. 嵌套加速功能列表

功能 ID	功能名称	描述
0x00000000	SBI_NACL_FEAT_SYNC_CSR	同步 CSR
0x00000001	SBI_NACL_FEAT_SYNC_HFENCE	同步 HFENCE
0x00000002	SBI_NACL_FEAT_SYNC_SRET	同步 SRET 指令
0x00000003	SBI_NACL_FEAT_AUTOSWAP_CSR	自动交换 CSR 寄存器
> 0x00000003	保留	预留供未来使用

要使用 SBI 嵌套加速扩展，监督模式软件（或 L1 虚拟机监控程序）必须在启动时为每个虚拟硬件线程设置嵌套加速共享内存的物理地址。嵌套加速共享内存的物理基地址必须按 4096 字节（即页）对齐，且共享内存大小必须为 4096 + (1024 * (XLEN / 8))字节。下表 65 展示了嵌套加速共享内存的布局结构。

表 65. 嵌套加速共享内存布局

名称	偏移量	大小（字节）	描述
暂存空间	0x00000000	4096	嵌套加速特性专用数据。
CSR 空间	0x00001000	XLEN * 128	一个由 1024 个 XLEN 位字组成的数组，其中每个单词对应 RISC-V 特权规范[1]表 2.1 中可能定义的 RISCV H-extension CSR 寄存器。

若需要，任何嵌套加速特性均可定义上文表 65 所示暂存空间的内容。

上文表 65 所示的 CSR 空间内容是一个 RISC-V H-extension CSR 值数组，其中 CSR位于索引 *= ((& 0xc00) >> 2) | (& 0xff)处。当任何 RISC-V H-extension CSR 寄存器状态发生变化时，SBI 实现（或 L0 虚拟机监控程序）必须更新该 CSR 空间。*

unless some nested acceleration feature defines a different behaviour. The Table 66 below shows CSR space index ranges for all possible 1024 RISC-V H-extension CSRs.

Table 66. Nested acceleration H-extension CSR index ranges

H-extension CSR address				SBI NACL CSR space index
[11:10]	[9:8]	[7:4]	Hex Range	Hex Range
00	10	xxxx	0x200 - 0x2ff	0x000 - 0x0ff
01	10	0xxx	0x600 - 0x67f	0x100 - 0x17f
01	10	10xx	0x680 - 0x6bf	0x180 - 0x1bf
01	10	11xx	0x6c0 - 0x6ff	0x1c0 - 0x1ff
10	10	0xxx	0xa00 - 0xa7f	0x200 - 0x27f
10	10	10xx	0xa80 - 0xabf	0x280 - 0x2bf
10	10	11xx	0xac0 - 0xaff	0x2c0 - 0x2ff
11	10	0xxx	0xe00 - 0xe7f	0x300 - 0x37f
11	10	10xx	0xe80 - 0xebf	0x380 - 0x3bf
11	10	11xx	0xec0 - 0xeff	0x3c0 - 0x3ff

15.1. Feature: Synchronize CSR (ID #0)

The synchronize CSR feature describes the ability of the SBI implementation (or LO hypervisor) to allow supervisor software (or L1 hypervisor) to write RISC-V H-extension CSRs using the CSR space.

This nested acceleration feature defines the scratch space offset range `0x0F80 - 0x0FFF` (128 bytes) as nested CSR dirty bitmap. The nested CSR dirty bitmap contains 1-bit for each possible RISC-V H-extension CSR.

To write a CSR `<x>` in nested acceleration shared memory, the supervisor software (or L1 hypervisor) MUST do the following:

1. Compute `<i> = ((<x> & 0xc00) >> 2) | (<x> & 0xff)`
2. Write a new CSR value at word with index `<i>` in the CSR space
3. Set the `<i>` bit in the nested CSR dirty bitmap

To synchronize a CSR `<x>`, the SBI implementation (or LO hypervisor) MUST do the following:

1. Compute `<i> = ((<x> & 0xc00) >> 2) | (<x> & 0xff)`
2. If bit `<i>` is not set in the nested CSR dirty bitmap then goto step 5
3. Emulate write to CSR `<x>` with the new CSR value taken from the word with index `<i>` in the CSR space
4. Clear the `<i>` bit in the nested CSR dirty bitmap
5. Write back the latest CSR value of CSR `<x>` to the word with index `<i>` in the CSR space

When synchronizing multiple CSRs, if the value of a CSR `<y>` depends on the value of some other CSR `<x>` then the SBI implementation (or LO hypervisor) MUST synchronize CSR `<x>` before CSR `<y>`. For example, the value of CSR `hip` depends on the value of the CSR `hvip`, which means `hvip` is emulated and written first, followed by `hip`.

除非某些嵌套加速特性定义了不同的行为。下表 66 展示了所有 1024 个可能的 RISC-V H 扩展 CSR 的寄存器空间索引范围。

表 66. 嵌套加速 H 扩展 CSR 索引范围

H 扩展 CSR 地址				SBI NACL CSR 空间索引
[11:10]	[9:8]	[7:4]	十六进制范围	十六进制范围
00	10	xxxx	0x200 - 0x2ff	0x000 - 0x0ff
01	10	0xxx	0x600 - 0x67f	0x100 - 0x17f
01	10	10xx	0x680 - 0x6bf	0x180 - 0x1bf
01	10	11xx	0x6c0 - 0x6ff	0x1c0 - 0x1ff
10	10	0xxx	0xa00 - 0xa7f	0x200 - 0x27f
10	10	10xx	0xa80 - 0xabf	0x280 - 0x2bf
10	10	11xx	0xac0 - 0xaff	0x2c0 - 0x2ff
11	10	0xxx	0xe00 - 0xe7f	0x300 - 0x37f
11	10	10xx	0xe80 - 0xebf	0x380 - 0x3bf
11	10	11xx	0xec0 - 0xeff	0x3c0 - 0x3ff

15.1. 功能：同步 CSR（ID #0）

同步 CSR 功能描述了 SBI 实现（或 L0 虚拟机监控程序）允许监督模式软件（或 L1 虚拟机监控程序）通过 CSR 空间写入 RISC-V H 扩展 CSR 的能力。

此嵌套加速功能将 0x0F80 - 0x0FFF（128 字节）的暂存空间偏移范围定义为嵌套 CSR 脏位图。嵌套 CSR 脏位图为每个可能的 RISC-V H 扩展 CSR 包含 1 个比特位。

若要在嵌套加速共享内存中写入 CSR，监督模式软件（或 L1 虚拟机监控程序）必须执行以下操作：

1. 计算 = ((& 0xc00) >> 2) | (& 0xff)
2. 在 CSR 空间中索引为*的字位置写入新的 CSR 值*
3. 在嵌套 CSR 脏位图中设置第*位*

为了同步 CSR，SBI 实现（或 L0 管理程序）必须执行以下操作：

1. 计算 = ((& 0xc00) >> 2) | (& 0xff)
2. 若嵌套 CSR 脏位图的第*位未置位，则跳转至步骤 5*
3. 使用 CSR 空间中索引为*的字作为新 CSR 值，模拟对 CSR的写入操作*
4. 清除嵌套 CSR 脏位图中的第*位*
5. 将 CSR的最新 CSR 值回写到 CSR 空间中索引为*的字单元*

当同步多个 CSR 时，若某个 CSR的值依赖于其他 CSR的值，则 SBI 实现（或 L0 管理程序）必须先同步 CSR再同步 CSR。例如，CSR hip 的值依赖于 CSR hvip 的值，这意味着需要先模拟并写入 hvip，随后处理 hip。

15.2. Feature: Synchronize HFENCE (ID #1)

The synchronize HFENCE feature describes the ability of the SBI implementation (or L0 hypervisor) to allow supervisor software (or L1 hypervisor) to issue HFENCE using the scratch space.

This nested acceleration feature defines the scratch space offset range `0x0800 - 0x0F7F` (1920 bytes) as an array of nested HFENCE entries. The total number of nested HFENCE entries are `3840 / XLEN` where each nested HFENCE entry consists of four XLEN-bit words.

A nested HFENCE entry is equivalent to an HFENCE over a range of guest addresses. The Table 67 below shows the nested HFENCE entry format whereas Table 68 below provides a list of nested HFENCE entry types. Upon an explicit synchronize HFENCE request from supervisor software (or L1 hypervisor), the SBI implementation (or L0 hypervisor) will process nested HFENCE entries with the `Config.Pending` bit set. After processing pending nested HFENCE entries, the SBI implementation (or L0 hypervisor) will clear the `Config.Pending` bit of these entries.

Table 67. Nested HFENCE entry format

Word	Name	Encoding

0	Config	Config information about the nested HFENCE entry
		
		BIT[XLEN-1:XLEN-1] - Pending
		BIT[XLEN-2:XLEN-4] - Reserved and must be zero
		BIT[XLEN-5:XLEN-8] - Type
		BIT[XLEN-9:XLEN-9] - Reserved and must be zero
		BIT[XLEN-10:XLEN-16] - Order
		if XLEN == 32 then
		  BIT[15:9] - VMID
		  BIT[8:0] - ASID
		else
		  BIT[29:16] - VMID
		  BIT[15:0] - ASID
		
		The page size for invalidation must be `1 << (Config.Order + 12)` bytes.

1	Page_Number	Page address right shifted by `Config.Order + 12`
2	Reserved	Reserved for future use and must be zero
3	Page_Count	Number of pages to invalidate

Table 68. Nested HFENCE entry types

Type	Name	Description
0	GVMA	Invalidate a guest physical address range across all VMIDs. The `VMID` and `ASID` fields of the `Config` word are ignored and MUST be zero.
1	GVMA_ALL	Invalidate all guest physical addresses across all VMIDs. The `Order`, `VMID` and `ASID` fields of the `Config` word are ignored and MUST be zero. The `Page_Number` and `Page_Count` words are ignored and MUST be zero.
2	GVMA_VMID	Invalidate a guest physical address range for a particular VMID. The `ASID` field of the `Config` word is ignored and MUST be zero.
3	GVMA_VMID_ALL	Invalidate all guest physical addresses for a particular VMID. The `Order` and `ASID` fields of the `Config` word are ignored and MUST be zero. The `Page_Number` and `Page_Count` words are ignored and MUST be zero.

15.2. 特性：同步 HFENCE（功能编号#1）

同步 HFENCE 功能描述了 SBI 实现（或 L0 虚拟机监控程序）允许监督模式软件（或 L1 虚拟机监控程序）通过暂存空间发起 HFENCE 的能力。

该嵌套加速特性将暂存空间偏移量范围 0x0800 - 0x0F7F（1920 字节）定义为嵌套 HFENCE 条目数组。嵌套 HFENCE 条目总数为 3840/XLEN，其中每个嵌套 HFENCE 条目由四个 XLEN 位字组成。

嵌套 HFENCE 条目等效于对客户机地址范围的 HFENCE 操作。下表 67 展示了嵌套 HFENCE 条目格式，而表 68 列出了嵌套 HFENCE 条目类型。当监督模式软件（或 L1 虚拟机监控程序）发出显式同步 HFENCE 请求时，SBI 实现（或 L0 虚拟机监控程序）将处理 Config.Pending 位被设置的嵌套 HFENCE 条目。处理完待处理的嵌套 HFENCE 条目后，SBI 实现（或 L0 虚拟机监控程序）将清除

Config.Pending 位　这些条目的位信息

表 67. 嵌套 HFENCE 条目格式

字段名称		编码方式
0	配置	关于嵌套 HFENCE 条目的配置信息 位[XLEN-1:XLEN-1] - 待处理位 位[XLEN-2:XLEN-4] - 保留位且必须为零 位[XLEN-5:XLEN-8] - 类型位 位[XLEN-9:XLEN-9] - 保留位且必须为零 位[XLEN-10:XLEN-16] - 排序位 若 XLEN == 32 则 位[15:9] - 虚拟机标识符(VMID) 位[8:0] - 地址空间标识符(ASID) 否则 位[29:16] - 虚拟机标识符(VMID) 位[15:0] - 地址空间标识符(ASID) 无效化操作的页面大小必须为 1 << (配置. 排序位 + 12)字节。
1	页号 = 页面地址右移(配置.排序位 + 12)位	
2	保留	保留未来使用，必须为零
3	页数	待无效化的页面数量

表 68. 嵌套 HFENCE 条目类型

类型名称		描述
0	GVMA	在所有 VMID 范围内使客机物理地址范围失效。配置字中的 VMID 和 ASID 字段被忽略且必须为零。
1	全局虚拟内存地址全量无效化（所有 VMID）	使所有 VMID 下的所有客户机物理地址失效。配置字中的 Order、VMID 和 ASID 字段被忽略且必须为零。页码（Page_Number）和页计数（Page_Count）字段被忽略且必须为零。
2	全局虚拟内存地址按 VMID 无效化（特定 VMID）	针对特定 VMID 的客户机物理地址范围进行无效化处理。 配置字中的 ASID 字段被忽略且必须为零。
3	GVMA_VMID_ALL	使特定 VMID 的所有客户机物理地址失效。配置字中的 Order 和 ASID 字段被忽略且必须为零。Page_Number 和 Page_Count 字段被忽略且必须为零。

Type	Name	Description
4	VVMA	Invalidate a guest virtual address range for a particular VMID. The `ASID` field of the `Config` word is ignored and MUST be zero.
5	VVMA_ALL	Invalidate all guest virtual addresses for a particular VMID. The `Order` and `ASID` fields of the `Config` word are ignored and MUST be zero. The `Page_Number` and `Page_Count` words are ignored and MUST be zero.
6	VVMA_ASID	Invalidate a guest virtual address range for a particular VMID and ASID.
7	VVMA_ASID_ALL	Invalidate all guest virtual addresses for a particular VMID and ASID. The `Order` field of the `Config` word is ignored and MUST be zero. The `Page_Number` and `Page_Count` words are ignored and MUST be zero.
> 7	Reserved	Reserved for future use.

To add a nested HFENCE entry, the supervisor software (or L1 hypervisor) MUST do the following:

1. Find an unused nested HFENCE entry with `Config.Pending == 0`
2. Update the `Page_Number` and `Page_Count` words in the nested HFENCE entry
3. Update the `Config` word in the nested HFENCE entry such that `Config.Pending` bit is set

To synchronize a nested HFENCE entry, the SBI implementation (or L0 hypervisor) MUST do the following:

1. If `Config.Pending == 0` then do nothing and skip below steps
2. Process HFENCE based on details in the nested HFENCE entry
3. Clear the `Config.Pending` bit in the nested HFENCE entry

15.3. Feature: Synchronize SRET (ID #2)

The synchronize SRET feature describes the ability of the SBI implementation (or L0 hypervisor) to do synchronization of CSRs and HFENCEs in the nested acceleration shared memory for the supervisor software (or L1 hypervisor) along with SRET emulation.

This nested acceleration feature defines the scratch space offset range `0x0000 - 0x01FF` (512 bytes) as nested SRET context. The Table 69 below shows contents of the nested SRET context.

Table 69. Nested SRET context

Offset	Name	Encoding
`0 * (XLEN / 8)`	Reserved	Reserved for future use and must be zero
`1 * (XLEN / 8)`	X1	Value to be restored in GPR X1
`2 * (XLEN / 8)`	X2	Value to be restored in GPR X2
`3 * (XLEN / 8)`	X3	Value to be restored in GPR X3
`4 * (XLEN / 8)`	X4	Value to be restored in GPR X4
`5 * (XLEN / 8)`	X5	Value to be restored in GPR X5
`6 * (XLEN / 8)`	X6	Value to be restored in GPR X6
`7 * (XLEN / 8)`	X7	Value to be restored in GPR X7

类型名称		描述
4	VVMA	使特定 VMID 的客户机虚拟地址范围失效。Config 字中的 ASID 字段被忽略且必须为零。
5	VVMA_ALL	使特定 VMID 的所有客户机虚拟地址失效。Config 字中的 Order 和 ASID 字段被忽略且必须为零。Page_Number 和 Page_Count 字被忽略且必须为零。
6	VVMA_ASID	使特定 VMID 和 ASID 对应的客户机虚拟地址范围失效。
7	VVMA_ASID_ALL	使特定 VMID 和 ASID 对应的所有客户机虚拟地址失效。Config 字段中的 Order 位被忽略且必须为零。Page_Number 和 Page_Count 字段被忽略且必须为零。
7 保留		预留未来使用

要添加嵌套的 HFENCE 条目，监控程序软件（或 L1 虚拟机监控程序）必须执行以下操作：

1. 找到一个未使用的嵌套 HFENCE 条目，且 Config.Pending == 0
2. 更新嵌套 HFENCE 条目中的 Page_Number 和 Page_Count 字段
3. 更新嵌套 HFENCE 条目中的配置字，设置 Config.Pending 位

为同步嵌套 HFENCE 条目，SBI 实现（或 L0 监控程序）必须执行以下操作：

1. 若 Config.Pending == 0 则无需处理并跳过后续步骤
2. 根据嵌套 HFENCE 条目中的详细信息处理 HFENCE
3. 清除嵌套 HFENCE 条目中的 Config.Pending 位

15.3. 功能：同步 SRET（ID #2）

同步 SRET 功能描述了 SBI 实现（或 L0 虚拟机监控程序）能够为上级软件（或 L1 虚拟机监控程序）在嵌套加速共享内存中同步 CSR 和 HFENCE，同时模拟 SRET 指令。
该嵌套加速功能定义了 0x0000 至 0x01FF（512 字节）的暂存空间偏移范围为嵌套 SRET 上下文。下表 69 展示了嵌套 SRET 上下文的具体内容。

表 69. 嵌套 SRET 上下文

偏移量	名称	编码
0 *(XLEN / 8)	保留	预留未来使用且必须置零
1 * (XLEN / 8)	X1	待恢复至通用寄存器 X1 的值
2 * (XLEN / 8)	X2	待恢复至通用寄存器 X2 的值
3 * (XLEN / 8)	X3	需恢复至 GPR X3 寄存器的值
4 * (XLEN / 8)	X4	待恢复至通用寄存器 X4 的值
5 * (XLEN / 8)	X5	需恢复至通用寄存器 X5 的值
6 * (XLEN / 8)	X6	需恢复至通用寄存器 X6 的值
7 * (XLEN / 8)	X7	需恢复至通用寄存器 X7 的值

Offset	Name	Encoding
`8 * (XLEN / 8)`	X8	Value to be restored in GPR X8
`9 * (XLEN / 8)`	X9	Value to be restored in GPR X9
`10 * (XLEN / 8)`	X10	Value to be restored in GPR X10
`11 * (XLEN / 8)`	X11	Value to be restored in GPR X11
`12 * (XLEN / 8)`	X12	Value to be restored in GPR X12
`13 * (XLEN / 8)`	X13	Value to be restored in GPR X13
`14 * (XLEN / 8)`	X14	Value to be restored in GPR X14
`15 * (XLEN / 8)`	X15	Value to be restored in GPR X15
`16 * (XLEN / 8)`	X16	Value to be restored in GPR X16
`17 * (XLEN / 8)`	X17	Value to be restored in GPR X17
`18 * (XLEN / 8)`	X18	Value to be restored in GPR X18
`19 * (XLEN / 8)`	X19	Value to be restored in GPR X19
`20 * (XLEN / 8)`	X20	Value to be restored in GPR X20
`21 * (XLEN / 8)`	X21	Value to be restored in GPR X21
`22 * (XLEN / 8)`	X22	Value to be restored in GPR X22
`23 * (XLEN / 8)`	X23	Value to be restored in GPR X23
`24 * (XLEN / 8)`	X24	Value to be restored in GPR X24
`25 * (XLEN / 8)`	X25	Value to be restored in GPR X25
`26 * (XLEN / 8)`	X26	Value to be restored in GPR X26
`27 * (XLEN / 8)`	X27	Value to be restored in GPR X27
`28 * (XLEN / 8)`	X28	Value to be restored in GPR X28
`29 * (XLEN / 8)`	X29	Value to be restored in GPR X29
`30 * (XLEN / 8)`	X30	Value to be restored in GPR X30
`31 * (XLEN / 8)`	X31	Value to be restored in GPR X31
`32 * (XLEN / 8)` - 0x1FF	Reserved	Reserved for future use

Before sending a synchronize SRET request to the SBI implementation (or L0 hypervisor), the supervisor software (or L1 hypervisor) MUST write the GPR `X<i>` values to be restored at offset `<i> * (XLEN / 8)` of the nested SRET context.

Upon a synchronize SRET request from the supervisor software (or L1 hypervisor), the SBI implementation (or L0 hypervisor) MUST do the following:

1. If SBI_NACL_FEAT_SYNC_CSR feature is available then
    a. All RISC-V H-extension CSRs implemented by the SBI implementation (or L0 hypervisor) are synchronized as described in the Section 15.1. This is equivalent to the SBI call `sbi_nacl_sync_csr(-1UL)`.
2. If SBI_NACL_FEAT_SYNC_HFENCE feature is available then
    a. All nested HFENCE entries are synchronized as described in the Section 15.2. This is equivalent to the SBI call `sbi_nacl_sync_hfence(-1UL)`.
3. Restore GPR `X<i>` registers from the nested SRET context.

偏移量	名称	编码
8 * (XLEN / 8)	X8	要恢复到通用寄存器 X8 中的值
9 * (XLEN / 8)	X9	待恢复至通用寄存器 X9 的值
10 * (XLEN / 8)	X10	需恢复至通用寄存器 X10 的值
11 * (XLEN / 8)	X11	需恢复至通用寄存器 X11 的值
12 * (XLEN / 8)	X12	需恢复至通用寄存器 X12 的值
13 * (XLEN / 8)	X13	需恢复至通用寄存器 X13 的值
14 * (XLEN / 8)	X14	需恢复至通用寄存器 X14 的值
15 * (XLEN / 8)	X15	需恢复至通用寄存器 X15 的值
16 * (XLEN / 8)	X16	待恢复至通用寄存器 X16 的值
17 * (XLEN / 8)	X17	要恢复到通用寄存器 X17 中的值
18 * (XLEN / 8)	X18	待恢复至通用寄存器 X18 的值
19 * (XLEN / 8)	X19	需恢复至 GPR X19 寄存器的值
20 * (XLEN / 8)	X20	需恢复至通用寄存器 X20 的值
21 * (XLEN / 8)	X21	待恢复至通用寄存器 X21 的值
22 * (XLEN / 8)	X22	需恢复至通用寄存器 X22 的值
23 * (XLEN / 8)	X23	需恢复至通用寄存器 X23 的值
24 * (XLEN / 8)	X24	需恢复至通用寄存器 X24 的值
25 * (XLEN / 8)	X25	需恢复至通用寄存器 X25 的值
26 * (XLEN / 8)	X26	需恢复至通用寄存器 X26 的值
27 * (XLEN / 8)	X27	需恢复至通用寄存器 X27 的值
28 * (XLEN / 8)	X28	需恢复至通用寄存器 X28 的值
29 * (XLEN / 8)	X29	需恢复至通用寄存器 X29 的值
30 * (XLEN / 8)	X30	待恢复至通用寄存器 X30 的值
31 * (XLEN / 8)	X31	待恢复至通用寄存器 X31 的值
32 * (XLEN / 8) - 0x1FF	保留	预留未来使用

在向 SBI 实现（或 L0 虚拟机监控程序）发送同步 SRET 请求前，监控态软件（或 L1 虚拟机监控程序）必须将要恢复的通用寄存器 X*值写入嵌套 SRET 上下文的偏移量*(XLEN/8)处。*

当接收到来自监控态软件（或 L1 虚拟机监控程序）的同步 SRET 请求时，SBI 实现（或 L0 虚拟机监控程序）必须执行以下操作：

1. 若 SBI_NACL_FEAT_SYNC_CSR 特性可用，则：a. SBI 实现（或 L0 虚拟机监控程序）所实现的所有 RISC-V H-extension CSR 寄存器将

   按照第 15.1 节所述进行同步。该操作等效于执行 SBI 调用 `sbi_nacl_sync_csr(-1UL)` .

2. 若 SBI_NACL_FEAT_SYNC_HFENCE 特性可用，则：a. 所有嵌套 HFENCE 条目将按照第 15.2 节所述进行同步。该操作等效于

   执行 SBI 调用 sbi_nacl_sync_hfence(-1UL)。

3. 从嵌套 SRET 上下文中恢复通用寄存器 X。

4. Emulate the SRET instruction as defined by the RISC-V Privilege specification [1].

15.4. Feature: Autoswap CSR (ID #3)

The autoswap CSR feature describes the ability of the SBI implementation (or L0 hypervisor) to automatically swap certain RISC-V H-extension CSR values from the nested acceleration shared memory in the following situations:

- Before emulating the SRET instruction for a synchronized SRET request from the supervisor software (or L1 hypervisor).
- After supervisor (or L1) virtualization state changes from ON to OFF.

The supervisor software (or L1 hypervisor) should use the autoswap CSR feature in conjunction with the synchronize SRET feature.

This nested acceleration feature defines the scratch space offset range `0x0200 - 0x027F` (128 bytes) as nested autoswap context. The Table 70 below shows contents of the nested autoswap context.

Table 70. Nested autoswap context

Offset	Name	Encoding
`0 * (XLEN / 8)`	Autoswap_Flags	Autoswap flags BIT[XLEN-1:1] - Reserved for future use and must be zero BIT[0:0] - HSTATUS
`1 * (XLEN / 8)`	HSTATUS	Value to be swapped with `HSTATUS` CSR
`2 * (XLEN / 8)` - 0x7F	Reserved	Reserved for future use.

To enable automatic swapping of CSRs from the nested autoswap context, the supervisor software (or L1 hypervisor) MUST do the following:

1. Write the `HSTATUS` swap value in the nested autoswap context.
2. Set `Autoswap_Flags.HSTATUS` bit in the nested autoswap context.

To swap CSRs from the nested autoswap context, the SBI implementation (or L0 hypervisor) MUST do the following:

1. If `Autoswap_Flags.HSTATUS` bit is set in the nested autoswap context then swap the supervisor `HSTATUS` CSR value with the `HSTATUS` value in the nested autoswap context.

15.5. Function: Probe nested acceleration feature (FID #0)

```
struct sbiret sbi_nacl_probe_feature(uint32_t feature_id)
```

Probe a nested acceleration feature. This is a mandatory function of the SBI nested acceleration extension. The `feature_id` parameter specifies the nested acceleration feature to probe. Table 64 provides a list of possible feature IDs.

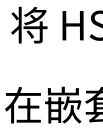
This function always returns SBI_SUCCESS in `sbiret.error`. It returns `0` in `sbiret.value` if the given `feature_id` is not available, or `1` in `sbiret.value` if it is available.

4. 按照 RISC-V 特权规范[1]模拟执行 SRET 指令。

15.4. 特性：自动交换 CSR（特性号 #3）

自动交换 CSR 特性描述了 SBI 实现（或 L0 虚拟机监控程序）在以下场景中自动从嵌套加速共享内存交换特定 RISC-V H-extension CSR 值的能力：

- 在模拟监管者软件（或 L1 虚拟机监控程序）发出的同步 SRET 请求的 SRET 指令之前。
- 当监管者（或 L1）虚拟化状态从开启变为关闭之后。



监管者软件（或 L1 虚拟机监控程序）应当将自动交换 CSR 特性与同步 SRET 特性配合使用。

此嵌套加速特性定义了 0x0200-0x027F（128 字节）的偏移地址范围作为嵌套自动交换上下文。下表 70 展示了嵌套自动交换上下文的具体内容。

表 70. 嵌套自动交换上下文

偏移量	名称	编码
0 *(XLEN / 8)	自动交换标志	自动交换标志位 比特位[XLEN-1:1] - 保留未来使用且必须置零 比特位[0:0] - HSTATUS
1 * (XLEN / 8)	HSTATUS	待与 HSTATUS CSR 交换的值
2 * (XLEN / 8) - 0x7F 保留字段		保留供未来使用

为启用嵌套自动交换上下文中 CSR 的自动交换功能，监督模式软件（或 L1 虚拟机监控程序）必须执行以下操作：

1. 将 HSTATUS 交换值写入嵌套自动交换上下文中。
2. 在嵌套自动交换上下文中设置 Autoswap_Flags.HSTATUS 标志位。

要从嵌套自动交换上下文交换 CSR 寄存器，SBI 实现（或 L0 虚拟机监控程序）必须执行以下操作：

1. 如果嵌套自动交换上下文中的 Autoswap_Flags.HSTATUS 标志位被设置，则将当前监管者模式下的 HSTATUS CSR 寄存器值与嵌套自动交换上下文中的 HSTATUS 值进行交换。

15.5. 功能：探测嵌套加速特性（功能号 #0） struct sbiret sbi_nacl_probe_feature(uint32_t feature_id)

探测嵌套加速功能。这是 SBI 嵌套加速扩展的必需功能。feature_id 参数指定要探测的嵌套加速功能。表 64 列出了可能的特性 ID。
此函数在 sbiret.error 中始终返回 SBI_SUCCESS。若给定的条件不满足，则会在 sbiret.value 中返回 0。

功能标识符　不可用时返回 0，可用时在 sbiret.value 中返回 1。

15.6. Function: Set nested acceleration shared memory (FID #1)

```
struct sbiret sbi_nacl_set_shmem(unsigned long shmem_phys_lo,
                                 unsigned long shmem_phys_hi,
                                 unsigned long flags)
```

Set and enable the shared memory for nested acceleration on the calling hart. This is a mandatory function of the SBI nested acceleration extension.

If both `shmem_phys_lo` and `shmem_phys_hi` parameters are not all-ones bitwise then `shmem_phys_lo` specifies the lower XLEN bits and `shmem_phys_hi` specifies the upper XLEN bits of the shared memory physical base address. `shmem_phys_lo` MUST be 4096 bytes (i.e. page) aligned and the size of the shared memory must be `4096 + (XLEN * 128)` bytes.

If both `shmem_phys_lo` and `shmem_phys_hi` parameters are all-ones bitwise then the nested acceleration features are disabled.

The `flags` parameter is reserved for future use and must be zero.

The possible error codes returned in `sbiret.error` are shown in Table 71.

Table 71. NACL Set Shared Memory Errors

Error code	Description
SBI_SUCCESS	Shared memory was set or cleared successfully.
SBI_ERR_INVALID_PARAM	The `flags` parameter is not zero or or the `shmem_phys_lo` parameter is not 4096 bytes aligned.
SBI_ERR_INVALID_ADDRESS	The shared memory pointed to by the `shmem_phys_lo` and `shmem_phys_hi` parameters does not satisfy the requirements described in Section 3.2.
SBI_ERR_FAILED	The request failed for unspecified or unknown other reasons.

15.7. Function: Synchronize shared memory CSRs (FID #2)

```
struct sbiret sbi_nacl_sync_csr(unsigned long csr_num)
```

Synchronize CSRs in the nested acceleration shared memory. This is an optional function which is only available if the SBI_NACL_FEAT_SYNC_CSR feature is available. The parameter `csr_num` specifies the set of RISC-V H-extension CSRs to be synchronized.

If `csr_num` is all-ones bitwise then all RISC-V H-extension CSRs implemented by the SBI implementation (or L0 hypervisor) are synchronized as described in the Section 15.1.

If `(csr_num & 0x300) == 0x200` and `csr_num < 0x1000` then only a single RISC-V H-extension CSR specified by the `csr_num` parameter is synchronized as described in the Section 15.1.

The possible error codes returned in `sbiret.error` are shown in Table 72.

Table 72. NACL Synchronize CSR Errors

15.6. 功能：设置嵌套加速共享内存（FID #1） struct sbiret sbi_nacl_set_shmem(unsigned long shmem_phys_lo,

```
                                                    unsigned long shmem_phys_hi, unsigned
                                                    long flags)
```

为当前硬件线程设置并启用嵌套加速共享内存。这是 SBI 嵌套加速扩展的必备功能。
若 shmem_phys_lo 和 shmem_phys_hi 参数不全为按位全 1 值，则 shmem_phys_lo 指定共享内存物理基地址的低 XLEN 位，shmem_phys_hi 指定高 XLEN 位。shmem_phys_lo 必须按 4096 字节（即页）对齐，且共享内存大小必须为 4096 + (XLEN * 128)字节。
若 shmem_phys_lo 和 shmem_phys_hi 参数均为按位全 1 值，则表示嵌套加速功能被禁用。

flags 参数保留供未来使用，当前必须置零。

sbiret.error 可能返回的错误代码如表 71 所示。

表 71. NACL 设置共享内存错误

错误代码	描述
SBI_SUCCESS	共享内存设置或清除成功。
SBI_ERR_INVALID_PARAM	flags 参数不为零或 shmem_phys_lo 参数未按 4096 字节对齐。
SBI_ERR_INVALID_ADDRESS 由 shmem_phys_lo 和 shmem_phys_hi 参数指向的共享内存不符合第 3.2 节所述要求。	
SBI_ERR_FAILED	请求因未指定或其他未知原因失败。

15.7. 功能：同步共享内存 CSR（FID #2） struct sbiret sbi_nacl_sync_csr(unsigned long csr_num)

同步嵌套加速共享内存中的 CSR 寄存器。此功能为可选功能，仅当 SBI_NACL_FEAT_SYNC_CSR 特性可用时才生效。参数 csr_num 指定需要同步的 RISC-V H-extension CSR 寄存器集合。
若 csr_num 为全 1 比特位，则 SBI 实现（或 L0 虚拟机监控程序）所实现的所有 RISC-V H 扩展 CSR 将按照第 15.1 节所述进行同步。
若(csr_num & 0x300) == 0x200 且 csr_num < 0x1000，则仅对由 csr_num 参数指定的单个 RISC-V H 扩展 CSR 按照第 15.1 节所述进行同步。

sbiret.error 可能返回的错误代码如表 72 所示。

表 72. NACL 同步 CSR 错误代码

Error code	Description
SBI_SUCCESS	CSRs synchronized successfully.
SBI_ERR_NOT_SUPPORTED	SBI_NACL_FEAT_SYNC_CSR feature is not available.
SBI_ERR_INVALID_PARAM	`csr_num` is not all-ones bitwise and either: * `(csr_num & 0x300) != 0x200` or * `csr_num >= 0x1000` or * `csr_num` is not implemented by the SBI implementation
SBI_ERR_NO_SHMEM	Nested acceleration shared memory not available.

15.8. Function: Synchronize shared memory HFENCEs (FID #3)

```
struct sbiret sbi_nacl_sync_hfence(unsigned long entry_index)
```

Synchronize HFENCEs in the nested acceleration shared memory. This is an optional function which is only available if the SBI_NACL_FEAT_SYNC_HFENCE feature is available. The parameter `entry_index` specifies the set of nested HFENCE entries to be synchronized.

If `entry_index` is all-ones bitwise then all nested HFENCE entries are synchronized as described in the Section 15.2.

If `entry_index < (3840 / XLEN)` then only a single nested HFENCE entry specified by the `entry_index` parameter is synchronized as described in the Section 15.2.

The possible error codes returned in `sbiret.error` are shown in Table 73.

Table 73. NACL Synchronize HFENCE Errors

Error code	Description
SBI_SUCCESS	HFENCEs synchronized successfully.
SBI_ERR_NOT_SUPPORTED	SBI_NACL_FEAT_SYNC_HFENCE feature is not available.
SBI_ERR_INVALID_PARAM	`entry_index` is not all-ones bitwise and `entry_index >= (3840 / XLEN)`.
SBI_ERR_NO_SHMEM	Nested acceleration shared memory not available.

15.9. Function: Synchronize shared memory and emulate SRET (FID #4)

```
struct sbiret sbi_nacl_sync_sret(void)
```

Synchronize CSRs and HFENCEs in the nested acceleration shared memory and emulate the SRET instruction. This is an optional function which is only available if the SBI_NACL_FEAT_SYNC_SRET feature is available.

This function is used by supervisor software (or L1 hypervisor) to do a synchronize SRET request and the SBI implementation (or L0 hypervisor) MUST handle it as described in the Section 15.3.

This function does not return upon success and the possible error codes returned in `sbiret.error` upon failure are shown in Table 74.

错误代码	描述
SBI_SUCCESS	CSR 同步成功完成。
SBI_ERR_NOT_SUPPORTED	SBI_NACL_FEAT_SYNC_CSR 功能不可用。
SBI_ERR_INVALID_PARAM	csr_num 并非全 1 位模式且满足以下任一条件： * (csr_num & 0x300) != 0x200 或 * csr_num >= 0x1000 或 * csr_num 未被当前 SBI 实现所支持
SBI_ERR_NO_SHMEM	嵌套加速共享内存不可用。

15.8. 功能：同步共享内存 HFENCE（功能号 #3） struct sbiret sbi_nacl_sync_hfence(unsigned long entry_index)

同步嵌套加速共享内存中的 HFENCE 操作。这是一个可选功能，仅当 SBI_NACL_FEAT_SYNC_HFENCE 特性可用时才生效。参数 entry_index 指定需要同步的嵌套 HFENCE 条目集合。

若 entry_index 为全 1 比特位，则按照第 15.2 节描述同步所有嵌套 HFENCE 条目。

若 entry_index < (3840 / XLEN)，则仅同步由 entry_index 参数指定的单个嵌套 HFENCE 条目，具体说明见第 15.2 节。

sbiret.error 可能返回的错误代码如表 73 所示。

表 73. NACL 同步 HFENCE 错误码

错误代码	描述
SBI_SUCCESS	HFENCE 同步成功。
SBI_ERR_NOT_SUPPORTED	SBI_NACL_FEAT_SYNC_HFENCE 特性不可用。
SBI_ERR_INVALID_PARAM	entry_index 非全 1 位且 entry_index >= (3840 / XLEN)。
SBI_ERR_NO_SHMEM	嵌套加速共享内存不可用。

15.9. 功能：同步共享内存并模拟 SRET（FID #4） struct sbiret sbi_nacl_sync_sret(void)

同步嵌套加速共享内存中的 CSR 和 HFENCE，并模拟 SRET 指令。这是一个可选功能，仅在 SBI_NACL_FEAT_SYNC_SRET 特性可用时生效。

该函数由监管者软件（或 L1 虚拟机监控程序）用于发起同步 SRET 请求，SBI 实现（或 L0 虚拟机监控程序）必须按照第 15.3 节所述进行处理。

此函数成功时不返回，失败时可能通过 sbiret.error 返回的错误代码如表 74 所示。

Table 74. NACL Synchronize SRET Errors

Error code	Description
SBI_ERR_NOT_SUPPORTED	SBI_NACL_FEAT_SYNC_SRET feature is not available.
SBI_ERR_NO_SHMEM	Nested acceleration shared memory not available.

15.10. Function Listing

Table 75. NACL Function List

Function Name	SBI Version	FID	EID
sbi_nacl_probe_feature	2.0	0	0x4E41434C
sbi_nacl_set_shmem	2.0	1	0x4E41434C
sbi_nacl_sync_csr	2.0	2	0x4E41434C
sbi_nacl_sync_hfence	2.0	3	0x4E41434C
sbi_nacl_sync_sret	2.0	4	0x4E41434C

表 74. NACL 同步 SRET 错误

错误代码	描述
SBI_ERR_NOT_SUPPORTED SBI_NACL_FEAT_SYNC_SRET	特性不可用。
SBI_ERR_NO_SHMEM	嵌套加速共享内存不可用。

15.10. 函数列表

表 75. NACL 函数列表

功能名称	SBI 版本	功能标识符	EID
sbi_nacl_probe_feature	2.0	0	0x4E41434C
sbi_nacl_set_shmem	2.0	1	0x4E41434C
sbi_nacl_sync_csr	2.0	2	0x4E41434C
sbi_nacl_sync_hfence	2.0	3	0x4E41434C
sbi_nacl_sync_sret	2.0	4	0x4E41434C

Chapter 16. Steal-time Accounting Extension (EID #0x535441 "STA")

SBI implementations may encounter situations where virtual harts are ready to run, but must be withheld from running. These situations may be, for example, when multiple SBI domains share processors or when an SBI implementation is a hypervisor and guest contexts share processors with other guest contexts or host tasks. When virtual harts are at times withheld from running, observers within the contexts of the virtual harts may need a way to account for less progress than would otherwise be expected. The time a virtual hart was ready, but had to wait, is called "stolen time" and the tracking of it is referred to as steal-time accounting. The Steal-time Accounting (STA) extension defines the mechanism in which an SBI implementation provides steal-time and preemption information, for each virtual hart, to supervisor-mode software.

16.1. Function: Set Steal-time Shared Memory Address (FID #0)

```
struct sbiret sbi_steal_time_set_shmem(unsigned long shmem_phys_lo,
                                       unsigned long shmem_phys_hi,
                                       unsigned long flags)
```

Set the shared memory physical base address for steal-time accounting of the calling virtual hart and enable the SBI implementation's steal-time information reporting.

If `shmem_phys_lo` and `shmem_phys_hi` are not all-ones bitwise, then `shmem_phys_lo` specifies the lower XLEN bits and `shmem_phys_hi` specifies the upper XLEN bits of the shared memory physical base address. `shmem_phys_lo` MUST be 64-byte aligned. The size of the shared memory must be at least 64 bytes. The SBI implementation MUST zero the first 64 bytes of the shared memory before returning from the SBI call.

If `shmem_phys_lo` and `shmem_phys_hi` are all-ones bitwise, the SBI implementation will stop reporting steal-time information for the virtual hart.

The `flags` parameter is reserved for future use and MUST be zero.

It is not expected for the shared memory to be written by the supervisor-mode software while it is in use for steal-time accounting. However, the SBI implementation MUST not misbehave if a write from supervisor-mode software occurs, however, in that case, it MAY leave the shared memory filled with inconsistent data.

The SBI implementation MUST stop writing to the shared memory when the supervisor-mode software is not runnable, such as upon system reset or system suspend.

Not writing to the shared memory when the supervisor-mode software is not runnable avoids unnecessary work and supports repeatable capture of a system image while the supervisor-mode software is suspended.

The shared memory layout is defined in Table 76

Table 76. STA Shared Memory Structure

第 16 章 窃取时间统计扩展（EID #0x535441 "STA"）

SBI 实现可能会遇到虚拟硬件线程已准备就绪但必须暂停运行的情况。这类情形可能出现在多个 SBI 域共享处理器时，或当 SBI 实现作为虚拟机监控程序且客户机上下文与其他客户机上下文或主机任务共享处理器时。当虚拟硬件线程被临时暂停运行时，其上下文中的观察者可能需要一种机制来解释实际进度低于预期的情况。虚拟硬件线程准备就绪但被迫等待的时间被称为"窃取时间"，其追踪机制则称为窃取时间统计。窃取时间统计（STA）扩展定义了 SBI 实现为每个虚拟硬件线程向监管模式软件提供窃取时间和抢占信息的机制。

16.1. 功能：设置窃取时间共享内存地址（FID #0）struct sbiret sbi_steal_time_set_shmem(unsigned long shmem_phys_lo,

```
unsigned long shmem_phys_hi, unsigned
long flags)
```

为调用虚拟硬件线程设置窃取时间统计的共享内存物理基地址，并启用 SBI 实现的窃取时间信息报告功能。

若 shmem_phys_lo 与 shmem_phys_hi 并非全 1 位模式，则 shmem_phys_lo 指定共享内存物理基地址的低 XLEN 位，shmem_phys_hi 指定其高 XLEN 位。

shmem_phys_lo 必须按 64 字节对齐。共享内存大小至少为 64 字节。SBI 实现必须在从 SBI 调用返回前，将共享内存的前 64 字节清零。

若 shmem_phys_lo 与 shmem_phys_hi 为全 1 位模式，SBI 实现将停止为该虚拟硬件线程报告 stealtime 信息。

flags 参数保留供未来使用，当前必须置零。

在用于窃取时间统计期间，不应预期监管模式软件会写入共享内存。然而，若发生监管模式软件的写入操作，SBI 实现不得出现异常行为，但在这种情况下，可能会使共享内存中保留不一致的数据。

当监管模式软件不可运行时（例如系统复位或系统挂起时），SBI 实现必须停止向共享内存写入数据。

在监管模式软件不可运行时停止写入共享内存，既可避免不必要的工作，又能支持在监管模式软件挂起期间对系统映像进行可重复的捕获。

共享内存布局定义见表 76

表 76. STA 共享内存结构

Name	Offset	Size	Description
sequence	0	4	The SBI implementation MUST increment this field to an odd value before writing the `steal` field, and increment it again to an even value after writing `steal` (i.e. an odd sequence number indicates an in-progress update). The SBI implementation SHOULD ensure that the sequence field remains odd for only very short periods of time. The supervisor-mode software MUST check this field before and after reading the `steal` field, and repeat the read if it is different or odd. *This sequence field enables the value of the steal field to be read by supervisor-mode software executing in a 32-bit environment.*
flags	4	4	Always zero. *Future extensions of the SBI call might allow the supervisor-mode software to write to some of the fields of the shared memory. Such extensions will not be enabled as long as a zero value is used for the flags argument to the SBI call.*
steal	8	8	The amount of time in which this virtual hart was not idle and scheduled out, in nanoseconds. The time during which the virtual hart is idle will not be reported as steal-time.
preempted	16	1	An advisory flag indicating whether the virtual hart which registered this structure is running or not. A non-zero value MAY be written by the SBI implementation if the virtual hart has been preempted (i.e. while the `steal` field is increasing), while a zero value MUST be written before the virtual hart starts to run again. *This preempted field can, for example, be used by the supervisor-mode software to check if a lock holder has been preempted, and, in that case, disable optimistic spinning.*
pad	17	47	Pad with zeros to a 64 byte boundary.

`sbiret.value` is set to zero and the possible error codes returned in `sbiret.error` are shown in Table 77 below.

Table 77. STA Set Steal-time Shared Memory Address Errors

Error code	Description
SBI_SUCCESS	The steal-time shared memory physical base address was set or cleared successfully.
SBI_ERR_INVALID_PARAM	The `flags` parameter is not zero or the `shmem_phys_lo` is not 64-byte aligned.
SBI_ERR_INVALID_ADDRESS	The shared memory pointed to by the `shmem_phys_lo` and `shmem_phys_hi` parameters is not writable or does not satisfy other requirements of Section 3.2.
SBI_ERR_FAILED	The request failed for unspecified or unknown other reasons.

名称	偏移量	大小	描述
序列	0	4	SBI 实现必须在写入 steal 字段前将该字段递增为奇数值，并在写入 steal 后再次递增为偶数值（即奇数序列号表示更新正在进行中）。SBI 实现应确保序列字段保持奇数状态的时间极短。 监管模式软件在读取 steal 字段前后必须检查该字段，若发现数值不同或为奇数则需重新读取。 该序列字段使得在 32 位环境中运行的监管模式软件能够正确读取 steal 字段的值。
flags	4	4	始终为零。 未来对 SBI 调用的扩展可能允许监管模式软件写入共享内存的某些字段。只要在 SBI 调用中使用零值作为 flags 参数，就不会启用此类扩展。
窃取	8	8	该虚拟硬件线程非空闲状态被调度出去的时间量，以纳秒为单位。时间 在此期间虚拟硬件线程处于空闲状态的时间不会被报告为窃取时间。
被抢占	16	1	一个建议性标志，用于指示注册此结构的虚拟硬件线程当前是否正在运行。若虚拟硬件线程被抢占（即当窃取时间字段正在增加时），SBI 实现可以写入非零值；而在虚拟硬件线程重新开始运行前，必须写入零值。 例如，管理模式软件可利用这个被抢占字段来检查锁持有者是否被抢占，若发生抢占情况，则可禁用乐观自旋策略。
填充	17		用零填充至 64 字节边界对齐

sbiret.value 被置为零，可能的错误码通过 sbiret.error 返回，如下表 77 所示

表 77. STA 设置窃取时间共享内存地址错误

错误代码	描述
SBI_SUCCESS	成功设置或清除了窃取时间共享内存的物理基地址。
SBI_ERR_INVALID_PARAM	flags 参数非零或 shmem_phys_lo 未按 64 字节对齐。
SBI_ERR_INVALID_ADDRESS	由 shmem_phys_lo 和 shmem_phys_hi 指向的共享内存不可写或不满足第 3.2 节的其他要求。
SBI_ERR_FAILED	请求因未指定或其他未知原因失败。

16.2. Function Listing

Table 78. STA Function List

Function Name	SBI Version	FID	EID
sbi_steal_time_set_shmem	2.0	0	0x535441

16.2. 功能列表

表 78. STA 功能列表

功能名称	SBI 版本	功能标识符	扩展 ID
sbi_steal_time_set_shmem 2.0		0	0x535441

Chapter 17. Supervisor Software Events Extension (EID #0x535345 "SSE")

The SBI Supervisor Software Events (SSE) extension provides a mechanism to inject software events from an SBI implementation to supervisor software such that it preempts all other traps and interrupts. The supervisor software will receive software events only on harts which are ready to receive them. A software event is delivered only after supervisor software has registered an event handler and enabled the software event.

The software events are one of two types: local or global. A local software event is local to a hart and can be handled only on that hart whereas a global software event is a system event and can be handled by any participating hart.

17.1. Software Event Identification

Each software event is identified by a unique 32-bit unsigned integer called `event_id`. The `event_id` space is divided into multiple 16-bit ranges where each 16-bit range is encoded as follows:

```
event_id[14:14] = Platform (0: Standard event, 1: Platform specific event)
event_id[15:15] = Global (0: Local event, 1: Global event)
```

The Table 79 below show the complete `event_id` space along with standard events based on the above encoding.

Table 79. SSE Event ID Space

Software Event ID	Description
Range 0x00000000 - 0x0000ffff	
0x00000000	Local High Priority RAS event
0x00000001	Local double trap event
0x00000002 - 0x00003fff	Local events reserved for future use
0x00004000 - 0x00007fff	Platform specific local events
0x00008000	Global High Priority RAS event
0x00008001 - 0x0000bfff	Global events reserved for future use
0x0000c000 - 0x0000ffff	Platform specific global events
Range 0x00010000 - 0x0001ffff	
0x00010000	Local PMU overflow event (depends on overflow IRQ)
0x00010001 - 0x00013fff	Local events reserved for future use
0x00014000 - 0x00017fff	Platform specific local events
0x00018000 - 0x0001bfff	Global events reserved for future use
0x0001c000 - 0x0001ffff	Platform specific global events
...	
Range 0x00100000 - 0x0010ffff	
0x00100000	Local Low Priority RAS event
0x00100001 - 0x00103fff	Local events reserved for future use

第 17 章 监管者软件事件扩展（EID #0x535345 "SSE"）

SBI 监督者软件事件（SSE）扩展提供了一种机制，使 SBI 实现能够向监督者软件注入软件事件，这些事件将抢占所有其他陷阱和中断。监督者软件仅会在准备接收事件的硬件线程（hart）上收到软件事件。软件事件只有在监督者软件注册了事件处理程序并启用该软件事件后才会被传递。

软件事件分为两种类型：本地事件或全局事件。本地软件事件仅与特定硬件线程相关且只能在该线程上处理，而全局软件事件属于系统事件，可由任何参与的硬件线程处理。

17.1. 软件事件标识

每个软件事件由一个称为 event_id 的 32 位无符号整数唯一标识。event_id 空间被划分为多个 16 位范围，每个 16 位范围的编码方式如下：

```
event_id[14:14] = 平台标识（0：标准事件，1：平台特定事件）event_id[15:15] = 全局标识（0：本地事件，1：全局事件）
```

下表 79 展示了基于上述编码规则的完整 event_id 空间及标准事件。

表 79. SSE 事件 ID 空间

软件事件 ID	描述
范围 0x00000000 - 0x0000ffff	
0x00000000	本地高优先级 RAS 事件
0x00000001	本地双重陷阱事件
0x00000002 - 0x00003fff	预留供未来使用的本地事件
0x00004000 - 0x00007fff	平台特定的本地事件
0x00008000	全局高优先级 RAS 事件
0x00008001 - 0x0000bfff	为未来使用保留的全局事件
0x0000c000 - 0x0000ffff	平台特定全局事件
范围 0x00010000 - 0x0001ffff	
0x00010000	本地性能监控单元溢出事件（取决于溢出中断请求）
0x00010001 - 0x00013fff	预留供未来使用的本地事件
0x00014000 - 0x00017fff	平台特定的本地事件
0x00018000 - 0x0001bfff	为未来使用保留的全局事件
0x0001c000 - 0x0001ffff	平台特定的全局事件
…	
地址范围 0x00100000 - 0x0010ffff	
0x00100000	本地低优先级 RAS 事件
0x00100001 - 0x00103fff	预留供未来使用的本地事件

Software Event ID	Description
0x00104000 - 0x00107fff	Platform specific local events
0x00108000	Global Low Priority RAS event
0x00108001 - 0x0010bfff	Global events reserved for future use
0x0010c000 - 0x0010ffff	Platform specific global events
...	
Range 0xffff0000 - 0xffffffff	
0xffff0000	Software injected local event
0xffff0001 - 0xffff3fff	Local events reserved for future use
0xffff4000 - 0xffff7fff	Platform specific local events
0xffff8000	Software injected global event
0xffff8001 - 0xffffbfff	Global events reserved for future use
0xffffc000 - 0xffffffff	Platform specific global events

Local double trap event: For SSE double trap events to be generated, supervisor software **MUST** *enable the double trap feature (*`DOUBLE_TRAP`*) via the Firmware Feature extension (Chapter 18).*

17.2. Software Event States

At any point in time, a software event **MUST** be in one of the following states:

1. **UNUSED** - Software event is not used by supervisor software
2. **REGISTERED** - Supervisor software has provided an event handler for the software event
3. **ENABLED** - Supervisor software is ready to handle the software event
4. **RUNNING** - Supervisor software is handling the software event

A **global** software event **MUST** be registered and enabled only once by any hart. By default, a global software event will be routed to any hart which is ready to receive software events but supervisor software can provide a preferred hart to handle this software event. The state of a global software event **MUST** be common to all harts.

The preferred hart assigned to a global software event by the supervisor software is only a hint about supervisor software's preference. The SBI implementation may choose a different hart for handling the global software event to avoid an inter-processor interrupt.

A **local** software event **MUST** be registered and enabled by all harts which want to handle this event. A local event is delivered to a hart only when the hart is ready to receive software events and the local event is registered and enabled on that hart. The state of a local software event **MUST** be tracked separately for each hart.

If a software event in `RUNNING` *state is signalled by the event source again, the software event will be taken only after the running event handler completes, provided that supervisor software doesn't disable the software event upon completion.*

The Figure 4 below shows the state transitions of a software event.

软件事件 ID	描述
0x00104000 - 0x00107fff	平台特定的本地事件
0x00108000	全局低优先级 RAS 事件
0x00108001 - 0x0010bfff	为未来使用保留的全局事件
0x0010c000 - 0x0010ffff	平台特定的全局事件
…	
范围 0xffff0000 - 0xffffffff	
0xffff0000	软件注入的本地事件
0xffff0001 - 0xffff3fff	预留供未来使用的本地事件
0xffff4000 - 0xffff7fff	平台特定的本地事件
0xffff8000	软件注入的全局事件
0xffff8001 - 0xffffbfff	预留供未来使用的全局事件
0xffffc000 - 0xffffffff	平台特定全局事件

本地双重陷阱事件：要生成 SSE 双重陷阱事件，监督模式软件必须通过固件特性扩展（第 18 章）启用双重陷阱功能（DOUBLE_TRAP）。

17.2. 软件事件状态

在任何时间点，软件事件必须处于以下状态之一：

1. 未使用 - 监管者软件未使用该软件事件
2. 已注册 - 监管者软件已为该软件事件提供事件处理程序
3. 已启用 - 监管者软件已准备好处理该软件事件
4. 运行中 - 监管者软件正在处理该软件事件

任何硬件线程（hart）对全局软件事件的注册与启用必须仅执行一次。默认情况下，全局软件事件会被路由至所有准备接收软件事件的硬件线程，但监督模式软件可指定首选硬件线程来处理该事件。全局软件事件的状态必须对所有硬件线程保持一致。

监督模式软件为全局软件事件指定的首选硬件线程仅作为偏好提示。SBI 实现可选择其他硬件线程来处理全局软件事件，以避免处理器间中断。

局部软件事件必须由所有需要处理该事件的硬件线程分别注册和启用。仅当硬件线程准备接收软件事件且已在该线程上注册并启用局部事件时，该事件才会被递送。局部软件事件的状态必须按每个硬件线程独立维护。

若处于运行状态的软件事件被事件源再次触发，只要监督模式软件未在事件处理完成后禁用该事件，新事件将在当前运行的事件处理程序完成后被捕获。

下图图 4 展示了软件事件的状态转换。

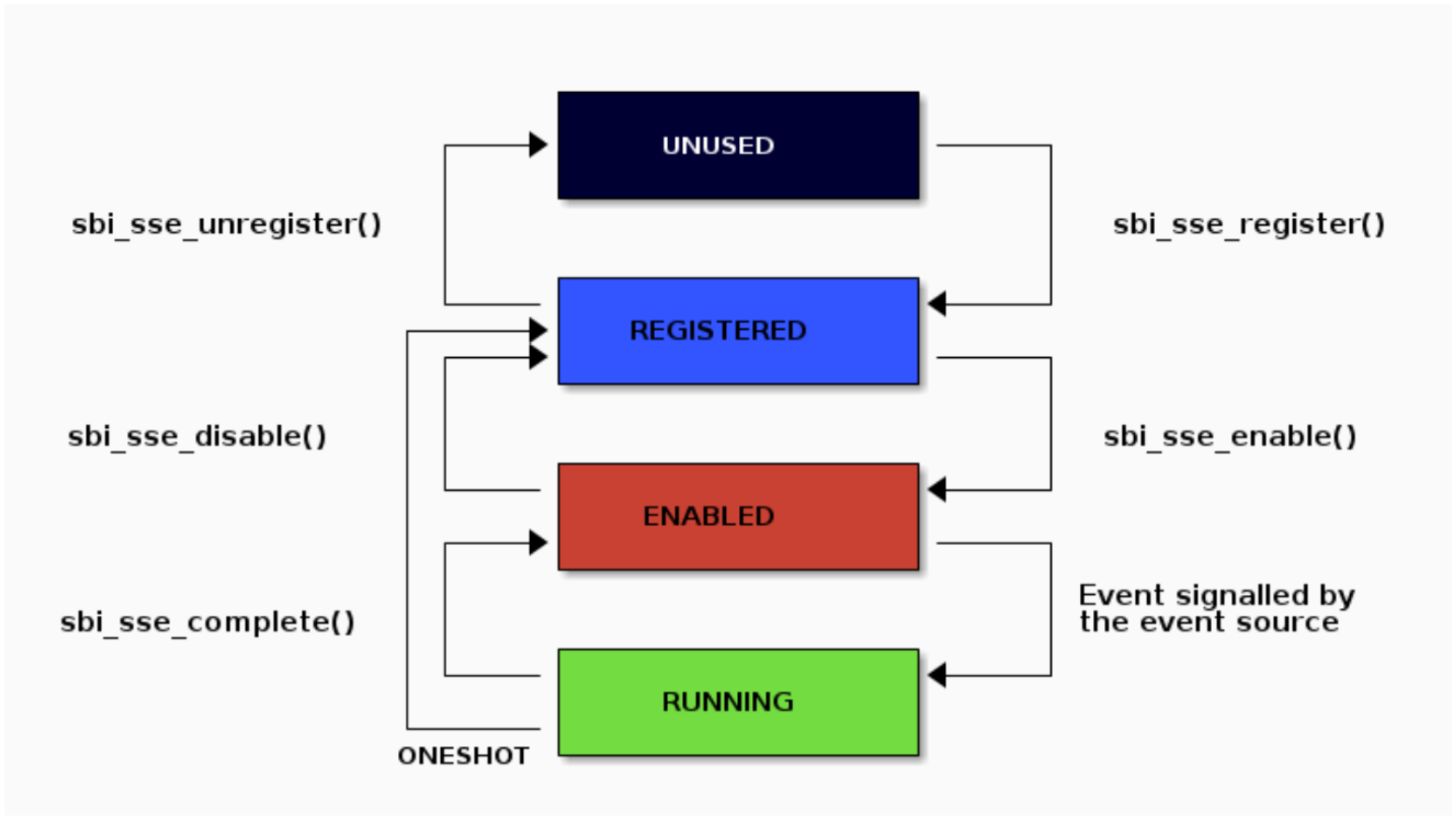


Figure 4. SBI SSE State Machine

17.3. Software Event Priority

Each software event has an associated priority (referred as `event_priority`) which is used by an SBI implementation to select a software event for injection when multiple software events are pending on the same hart.

The priority of a software event is a 32-bit unsigned integer where lower value means higher priority. By default, all software events have event priority as zero.

If two or more software events have same priority on a given hart then the SBI implementation must use `event_id` for tie-breaking where lower `event_id` has higher priority.

A higher priority software event, unless disabled by supervisor software, **always** preempts a lower priority software event in `RUNNING` state on the same hart. Once a higher priority software event is completed, the previous lower priority software event will be resumed.

17.4. Software Event Attributes

A software event can have various XLEN bits wide attributes associated to it where each event attribute is identified by a unique 32-bit unsigned integer called `attr_id`. A software event attribute has Read-Only or Read-Write access permissions. The Table 80 below provides a list event attributes.

Table 80. SSE Event Attributes

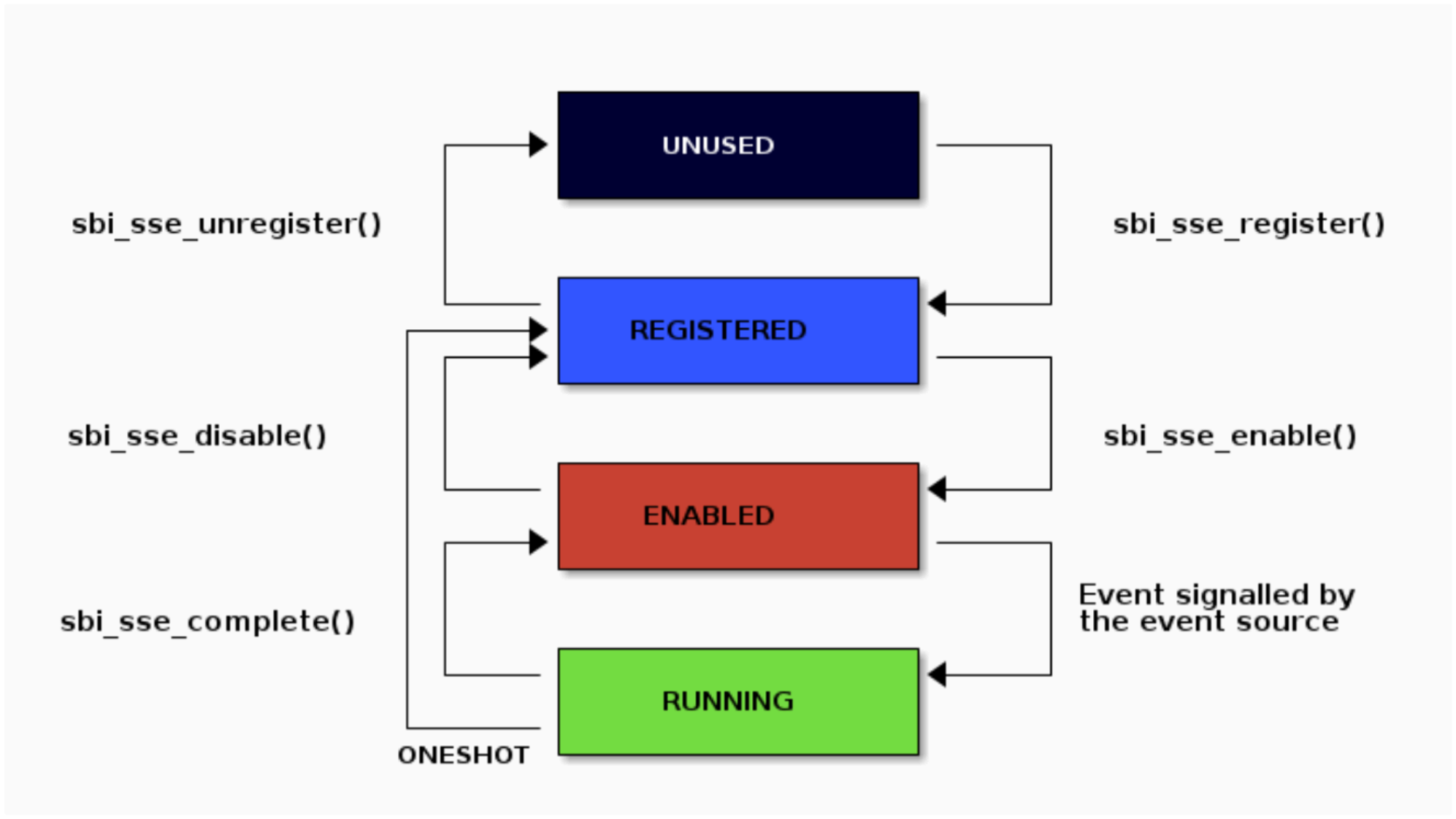


图 4. SBI SSE 状态机

17.3. 软件事件优先级

每个软件事件都关联着一个优先级（称为 event_priority），当同一硬件线程上有多个软件事件处于待处理状态时，SBI 实现将根据该优先级来选择要注入的软件事件。
软件事件优先级是一个 32 位无符号整数，数值越小表示优先级越高。默认情况下，所有软件事件的优先级均为零。
若同一硬件线程上有两个或多个软件事件具有相同优先级，则 SBI 实现必须采用

用于仲裁的事件 ID，数值较小者优先级较高。

除非被监管软件禁用，否则较高优先级的软件事件总是会抢占同一硬件线程上处于运行状态的较低优先级软件事件。一旦较高优先级的软件事件完成执行，先前被抢占的较低优先级软件事件将恢复运行。

17.4 软件事件属性

每个软件事件可关联多个 XLEN 位宽的属性，各属性通过唯一的 32 位无符号整数 attr_id 进行标识。软件事件属性具有只读或读写访问权限。下表 80 列出了相关事件属性。

表 80. SSE 事件属性

Attribute Name	Attribute ID (attr_id)	Access (RO / RW)	Description
STATUS	0x00000000	RO	Status of the software event which is encoded as follows: **bit[1:0]**: Event state with following possible values: 0 = UNUSED, 1 = REGISTERED, 2 = ENABLED, and 3 = RUNNING **bit[2:2]**: Event pending status (1 = Pending and 0 = Not Pending). This flag is set by the event source and it is cleared when the software event is moved to **RUNNING** state. **bit[3:3]**: Event injection using the **sbi_sse_inject** call (1 = Allowed and 0 = Not allowed) **bit[XLEN-1:4]**: Reserved for future use and must be zero The reset value of this attribute is zero.
PRIORITY	0x00000001	RW	Software event priority where only lower 32-bits of the value are used and other bits are always set to zero. This attribute can be updated only when the software event is in **UNUSED** or **REGISTERED** state. The reset value of this attribute is zero.
CONFIG	0x00000002	RW	Additional configuration of the software event. This attribute can be updated only when the software event is in **UNUSED** or **REGISTERED** state. The encoding of this event attribute is as follows: **bit[0:0]**: Disable software event upon **sbi_sse_complete** call (one-shot) **bit[XLEN-1:1]**: Reserved for future use and must be zero The reset value of this attribute is zero.
PREFERRED_HART	0x00000003	RW (global) RO (local)	Hart ID of the preferred hart that should handle the global software event. The value of this attribute must always be valid hart ID for both local and global software events. This attribute is read-only for local software events and for global software events it can be updated only when the software event is in **UNUSED** or **REGISTERED** state. The reset value of this attribute is SBI implementation specific.

属性名称	属性 ID (属性标识)	访问权限 (只读/读写)	描述
状态	0x00000000 只读		软件事件的状态编码如下： bit[1:0]：事件状态，可能取值如下：0=未使用，1=已注册，2=已启用，3=运行中 bit[2:2]：事件待处理状态（1=待处理，0=非待处理）。该标志由事件源设置，当软件事件转为运行状态时将被清除。 bit[3:3]: 通过 sbi_sse_inject 调用进行事件注入的权限位（1=允许，0=禁止） bit[XLEN-1:4]: 保留位（必须置零） 该属性的复位值为零。
PRIORITY 0x00000001 RW 软件事件优先级（仅使用该值的低 32 位）			其余位始终置零。该属性仅在软件事件处于 未使用或已注册状态 该属性的复位值为零。
配置	0x00000002 可读写		软件事件的附加配置。该属性仅在软件事件处于未使用或已注册状态时可更新。此事件属性的编码如下： 位[0:0]：单次触发时禁用软件事件 sbi_sse_complete 调用（单次触发） 位[XLEN-1:1]：保留未来使用且必须为零 该属性的复位值为零。
PREFERRED_HART	0x00000003 可读写（全局） 只读（局部）		处理全局软件事件的首选硬件线程 ID。该属性值对于局部和全局软件事件都必须始终是有效的硬件线程 ID。对于局部软件事件此属性为只读，而对于全局软件事件，仅当软件事件处于 未使用或已注册状态时才可更新。 该属性的复位值由 SBI 具体实现决定。

Attribute Name	Attribute ID (attr_id)	Access (RO / RW)	Description
ENTRY_PC	0x00000004	RO	Entry program counter value for handling the software event in supervisor software. The value of this event attribute MUST be 2-bytes aligned. The reset value of this attribute is zero.
ENTRY_ARG	0x00000005	RO	Entry argument (or parameter) value for handling the software event in supervisor software. This attribute value is passed to the supervisor software via `A7` GPR. The reset value of this attribute is zero.
INTERRUPTED_SEPC	0x00000006	RW	Interrupted `sepc` CSR value which is saved before handling the software event in supervisor software. This attribute can be updated only when the software event is in `RUNNING` state. For global events, only the hart executing the event handler can modify it. The reset value of this attribute is zero.
INTERRUPTED_FLAGS	0x00000007	RW	Interrupted flags which are saved before handling the software event in supervisor software. This attribute can be updated only when the software event is in `RUNNING` state. For global events, only the hart executing the event handler can modify it. The encoding of this event attribute is as follows: `bit[0:0]`: interrupted `sstatus.SPP` CSR bit value `bit[1:1]`: interrupted `sstatus.SPIE` CSR bit value `bit[2:2]`: interrupted `hstatus.SPV` CSR bit value `bit[3:3]`: interrupted `hstatus.SPVP` CSR bit value `bit[4:4]`: interrupted `sstatus.SPELP` CSR bit value if `Zicfilp` extension is available to supervisor mode `bit[5:5]`: interrupted `sstatus.SDT` CSR bit value if `Ssdbltrp` extension is available to supervisor mode `bit[XLEN-1:6]`: Reserved for future use and must be set to zero

属性名称	属性 ID (属性标识)	访问权限 (只读/读写)	描述
入口程序计数器(ENTRY_PC)	0x00000004 RO		用于监管模式软件处理软件事件的入口程序计数器值。该事件属性值必须按 2 字节对齐。 该属性的复位值为零。
入口参数(ENTRY_ARG)	0x00000005 只读		用于监管模式软件处理软件事件的入口参数值。该属性值通过 A7 通用寄存器传递给监管模式软件。 该属性的复位值为零。
INTERRUPTED_SEPC 0x00000006 RW			在处理监督模式软件的软件事件前保存的中断 sepc CSR 值。该属性仅在软件事件处于 RUNNING 状态时可更新。对于全局事件，只有执行事件处理程序的 hart 才能修改它。 该属性的复位值为零。
INTERRUPTED_FLAGS 0x00000007 RW			在处理监督模式软件的软件事件前保存的中断标志位。该属性仅在软件事件处于 RUNNING 状态时可更新。对于全局事件，只有执行事件处理程序的 hart 才能修改它。该事件属性的编码方式如下： 位[0:0]: 中断的 sstatus.SPP CSR 位值 位[1:1]: 中断的 sstatus.SPIE CSR 位值 位[2:2]: 中断时的 hstatus.SPV CSR 状态位值 位[3:3]: 中断时的 hstatus.SPVP CSR 状态位值 位[4:4]：中断时的 sstatus.SPELP 控制状态寄存器若监督者模式可用 Zicfilp 扩展时的位值 位[5:5]：中断时的 sstatus.SDT 控制状态寄存器若监督者模式可用 Ssdbltrp 扩展时的位值 bit[XLEN-1:6]：保留未来使用，必须置零

Attribute Name	Attribute ID (attr_id)	Access (RO / RW)	Description
INTERRUPTED_A6	0x00000008	RW	Interrupted `A6` GPR value which is saved before handling the software event in supervisor software. This attribute can be updated only when the software event is in `RUNNING` state. For global events, only the hart executing the event handler can modify it. The reset value of this attribute is zero.
INTERRUPTED_A7	0x00000009	RW	Interrupted `A7` GPR value which is saved before handling the software event in supervisor software. This attribute can be updated only when the software event is in `RUNNING` state. For global events, only the hart executing the event handler can modify it. The reset value of this attribute is zero.
RESERVED	> 0x00000009	---	Reserved for future use.

17.5. Software Event Injection

To inject a software event on a hart, the SBI implementation must do the following:

1. Save interrupted state of supervisor mode
    a. Set `INTERRUPTED_FLAGS` event attribute as follows:
        i. `INTERRUPTED_FLAGS[0:0]` = interrupted `sstatus.SPP` CSR bit value
        ii. `INTERRUPTED_FLAGS[1:1]` = interrupted `sstatus.SPIE` CSR bit value
        iii. if H-extension is available to supervisor mode:
            A. Set `INTERRUPTED_FLAGS[2:2]` = interrupted `hstatus.SPV` CSR bit value
            B. Set `INTERRUPTED_FLAGS[3:3]` = interrupted `hstatus.SPVP` CSR bit value
        iv. else
            A. Set `INTERRUPTED_FLAGS[3:2]` = zero
        v. if `Zicfilp` extension is available to supervisor mode:
            A. `INTERRUPTED_FLAGS[4:4]` = interrupted `sstatus.SPELP` CSR bit value
        vi. else
            A. `INTERRUPTED_FLAGS[4:4]` = zero
        vii. if `Ssdbltrp` extension is available to supervisor mode:
            A. `INTERRUPTED_FLAGS[5:5]` = interrupted `sstatus.SDT` CSR bit value
        viii. else
            A. `INTERRUPTED_FLAGS[5:5]` = zero
        ix. Set `INTERRUPTED_FLAGS[XLEN-1:6]` = zero
    b. Set `INTERRUPTED_SEPC` event attribute = interrupted `sepc` CSR

属性名称	属性 ID (属性标识)	访问权限 (只读/读写)	描述
中断 A6	0x00000008 可读写		在处理监管模式软件事件前保存的中断 A6 通用寄存器值。该属性仅在软件事件处于运行状态时可更新。对于全局事件，仅执行事件处理程序的硬件线程可修改此值。 该属性的重置值为零。
中断 A7 寄存器	0x00000009 可读写		在处理监管模式软件事件前保存的中断 A7 通用寄存器值。该属性仅在软件事件处于运行状态时可更新。对于全局事件，仅执行事件处理程序的硬件线程可修改此值。 该属性的复位值为零。
保留字段	> 0x00000009 ---		预留未来使用

17.5. 软件事件注入

要在某个硬件线程上注入软件事件，SBI 实现必须执行以下操作：

1. 保存监管者模式的中断状态
a. 按以下方式设置 INTERRUPTED_FLAGS 事件属性：

i. INTERRUPTED_FLAGS[0:0] = 被中断的 sstatus.SPP CSR 位值

ii. INTERRUPTED_FLAGS[1:1] = 被中断的 sstatus.SPIE CSR 位值

iii. 若监管者模式支持 H 扩展：

A. 设置 INTERRUPTED_FLAGS[2:2] = 被中断的 hstatus.SPV CSR 位值

B. 设置 INTERRUPTED_FLAGS[3:3] = 被中断的 hstatus.SPVP CSR 位值

iv. 否则

A. 设置 INTERRUPTED_FLAGS[3:2] = 零

五、若监督者模式支持 Zicfilp 扩展：

A. 中断标志位[4:4] = 被中断的 sstatus.SPELP CSR 位值

六、否则

A. 中断标志位[4:4] = 零

七、若监督者模式可使用 Ssdbltrp 扩展：

A. 中断标志位[5:5] = 被中断的 sstatus.SDT CSR 位值

八、否则

A. 中断标志位[5:5] = 零

ix. 将 INTERRUPTED_FLAGS[XLEN-1:6]设置为零

b. 设置 INTERRUPTED_SEPC 事件属性 = 被中断的 sepc CSR 值

   c. Set `INTERRUPTED_A6` event attribute = interrupted `A6` GPR value
   d. Set `INTERRUPTED_A7` event attribute = interrupted `A7` GPR value
2. Redirect execution to supervisor event handler
   a. Set `A6` GPR = Current Hart ID
   b. Set `A7` GPR = `ENTRY_ARG` event attribute value
   c. Set `sepc` = Interrupted program counter value
   d. Set `sstatus.SPP` CSR bit = interrupted privilege mode
   e. Set `sstatus.SPIE` CSR bit = `sstatus.SIE` CSR bit value
   f. Set `sstatus.SIE` CSR bit = zero
   g. if `Zicfilp` extension is available to supervisor mode:
      i. Set `sstatus.SPELP` = interrupted landing pad state
      ii. Set landing pad state = NO_LP_EXPECTED
   h. if H-extension is available to supervisor mode:
      i. Set `hstatus.SPV` CSR bit = interrupted virtualization state
      ii. if `hstatus.SPV` CSR bit == 1:
         A. Set `hstatus.SPVP` CSR bit = `sstatus.SPP` CSR bit value
   i. if `Ssdbltrp` extension is available to supervisor mode:
      i. Set S-mode-disable-trap = 1
   j. Set virtualization state = OFF
   k. Set privilege mode = S-mode
   l. Set program counter = `ENTRY_PC` event attribute value

17.6. Software Event Completion

After handling the software event on a hart, the supervisor software must notify the SBI implementation about completion of event handling using `sbi_sse_complete` call. The SBI implementation must do the following to resume the interrupted state for a completed event:

1. Set program counter = `sepc` CSR value
2. Set privilege mode = `sstatus.SPP` CSR bit value
3. if `Ssdbltrp` extension is available to supervisor mode:
   a. Set `sstatus.SDT` CSR bit = `INTERRUPTED_FLAGS[5:5]` event attribute value
4. if `Zicfilp` extension is available to supervisor mode:
   a. Set `sstatus.SPELP` CSR bit = `INTERRUPTED_FLAGS[4:4]` event attribute value
5. if H-extension is available to supervisor mode:
   a. Set virtualization state = `hstatus.SPV` CSR bit value
   b. Set `hstatus.SPV` CSR bit = `INTERRUPTED_FLAGS[2:2]` event attribute value
   c. Set `hstatus.SPVP` CSR bit = `INTERRUPTED_FLAGS[3:3]` event attribute value
6. Set `sstatus.SIE` CSR bit = `sstatus.SPIE` CSR bit
7. Set `sstatus.SPIE` CSR bit = `INTERRUPTED_FLAGS[1:1]` event attribute value

c. 设置 INTERRUPTED_A6 事件属性 = 被中断的 A6 通用寄存器值

d. 设置 INTERRUPTED_A7 事件属性 = 被中断的 A7 通用寄存器值

2. 重定向执行至监控程序事件处理程序
a. 设置 A6 通用寄存器 = 当前硬件线程 ID

b. 设置 A7 通用寄存器 = ENTRY_ARG 事件属性值

c. 设置 sepc = 被中断的程序计数器值

d. 设置 sstatus.SPP CSR 位 = 被中断的特权模式

e. 设置 sstatus.SPIE CSR 位 = sstatus.SIE CSR 位的值

f. 设置 sstatus.SIE CSR 位 = 零

g. 若监督者模式可用 Zicfilp 扩展：

i. 设置 sstatus.SPELP = 中断前的着陆垫状态

ii. 设置着陆垫状态 = 无预期着陆垫(NO_LP_EXPECTED)

h. 若监督者模式支持 H 扩展：

i. 设置 hstatus.SPV CSR 位 = 中断前的虚拟化状态

二、若 hstatus.SPV CSR 位 == 1：

甲、将 hstatus.SPVP CSR 位设置为 sstatus.SPP CSR 位的值

一、若监督者模式可使用 Ssdbltrp 扩展：

一、设置 S-mode-disable-trap = 1

j. 设置虚拟化状态 = 关闭

k. 设置特权模式 = S 模式

l. 设置程序计数器 = ENTRY_PC 事件属性值

17.6. 软件事件完成

在处理完某个硬件线程上的软件事件后，监督模式软件必须通过 sbi_sse_complete 调用来通知 SBI 实现已完成事件处理。对于已完成的事件，SBI 实现必须执行以下操作以恢复被中断的状态：

1. 将程序计数器设置为 sepc CSR 的值

2. 将特权模式设置为 sstatus.SPP CSR 位的值

3. 若监督模式支持 Ssdbltrp 扩展：a. 将 sstatus.SDT CSR 位设置为 INTERRUPTED_FLAGS[5:5]事件属性值

4. 若监督者模式可用 Zicfilp 扩展：a. 设置 sstatus.SPELP CSR 位 = INTERRUPTED_FLAGS[4:4]事件属性值

5. 若监督者模式可用 H 扩展：a. 设置虚拟化状态 = hstatus.SPV CSR 位值

b. 设置 hstatus.SPV CSR 位 = INTERRUPTED_FLAGS[2:2]事件属性值

c. 设置 hstatus.SPVP CSR 位 = INTERRUPTED_FLAGS[3:3]事件属性值

6. 设置 sstatus.SIE CSR 位 = sstatus.SPIE CSR 位
7. 设置 sstatus.SPIE CSR 位 = INTERRUPTED_FLAGS[1:1]事件属性值

8. Set `sstatus.SPP` CSR bit = `INTERRUPTED_FLAGS[0:0]` event attribute value
9. Set `A7` GPR = `INTERRUPTED_A7` event attribute value
10. Set `A6` GPR = `INTERRUPTED_A6` event attribute value
11. Set `sepc` = `INTERRUPTED_SEPC` event attribute value

If the supervisor software wishes to resume from a different location, it can update the event attributes of the software event before calling `sbi_sse_complete`.

17.7. Function: Read software event attributes (FID #0)

```
struct sbiret sbi_sse_read_attrs(uint32_t event_id,
                                 uint32_t base_attr_id, uint32_t attr_count,
                                 unsigned long output_phys_lo,
                                 unsigned long output_phys_hi)
```

Read a range of event attribute values from a software event.

The `event_id` parameter specifies the software event ID whereas `base_attr_id` and `attr_count` parameters specifies the range of event attribute IDs.

The event attribute values are written to a output shared memory which is specified by the `output_phys_lo` and `output_phys_hi` parameters where:

- The `output_phys_lo` parameter MUST be `XLEN / 8` bytes aligned
- The size of output shared memory is assumed to be `(XLEN / 8) * attr_count`
- The value of event attribute with ID `base_attr_id + i` should be written at offset `(XLEN / 8) * (base_attr_id + i)`

In case of an error, the possible error codes are shown in the Table 81 below:

Table 81. SSE Event Attributes Read Errors

Error code	Description
SBI_SUCCESS	Event attribute values read successfully.
SBI_ERR_NOT_SUPPORTED	`event_id` is not reserved and valid, but the platform does not support it due to one or more missing dependencies (Hardware or SBI implementation).
SBI_ERR_INVALID_PARAM	`event_id` is invalid or `attr_count` is zero.
SBI_ERR_BAD_RANGE	One of the event attribute IDs in the range specified by `base_attr_id` and `attr_count` is reserved.
SBI_ERR_INVALID_ADDRESS	The shared memory pointed to by the `output_phys_lo` and `output_phys_hi` parameters does not satisfy the requirements described in Section 3.2.
SBI_ERR_FAILED	The read failed for unspecified or unknown other reasons.

8. 将 sstatus.SPP CSR 位设置为 INTERRUPTED_FLAGS[0:0]事件属性值
9. 将 A7 通用寄存器设置为 INTERRUPTED_A7 事件属性值
10. 将 A6 通用寄存器设置为 INTERRUPTED_A6 事件属性值
11. 将 sepc 设置为 INTERRUPTED_SEPC 事件属性值

若监管模式软件希望从不同位置恢复执行，可在调用 sbi_sse_complete 前更新软件事件的事件属性。

17.7. 功能：读取软件事件属性（功能号#0）

```
struct sbiret sbi_sse_read_attrs(uint32_t event_id,
                                 uint32_t base_attr_id, uint32_t attr_count, unsigned long
                                 output_phys_lo, unsigned long output_phys_hi)
```

从软件事件中读取一组事件属性值。

event_id 参数指定软件事件 ID，而 base_attr_id 和 attr_count 参数指定事件属性 ID 的范围。
事件属性值将被写入由 output_phys_lo 和 output_phys_hi 参数指定的输出共享内存区域，其中：

- output_phys_lo 参数必须按 XLEN/8 字节对齐
- 输出共享内存的大小假定为(XLEN/8)*attr_count
- ID 为 base_attr_id + i 的事件属性值应写入偏移量(XLEN/8)*(base_attr_id + i)处

若发生错误，可能出现的错误代码如下表 81 所示：

表 81. SSE 事件属性读取错误

错误代码	描述
SBI_SUCCESS	事件属性值读取成功。
SBI_ERR_NOT_SUPPORTED	event_id 未被保留且有效，但由于缺少一个或多个依赖项（硬件或 SBI 实现），平台不支持该事件。
SBI_ERR_INVALID_PARAM	事件 ID 无效或属性计数为零。
SBI_ERR_BAD_RANGE	在指定范围内的事件属性 ID 之一 base_attr_id 和 attr_count 保留未使用
SBI_ERR_INVALID_ADDRESS	由 output_phys_lo 和 output_phys_hi 参数指向的共享内存不符合第 3.2 节所述要求
SBI_ERR_FAILED	由于未指定或其他未知原因导致读取失败

17.8. Function: Write software event attributes (FID #1)

```
struct sbiret sbi_sse_write_attrs(uint32_t event_id,
                                  uint32_t base_attr_id, uint32_t attr_count,
                                  unsigned long input_phys_lo,
                                  unsigned long input_phys_hi)
```

Write a range of event attribute values to a software event.

The `event_id` parameter specifies the software event ID whereas `base_attr_id` and `attr_count` parameters specifies the range of event attribute IDs.

The event attribute values are read from a input shared memory which is specified by the `input_phys_lo` and `input_phys_hi` parameters where:

- The `input_phys_lo` parameter MUST be `XLEN / 8` bytes aligned
- The size of input shared memory is assumed to be `(XLEN / 8) * attr_count`
- The value of event attribute with ID `base_attr_id + i` should be read from offset `(XLEN / 8) * (base_attr_id + i)`

For local events, the event attributes are updated only for the calling hart. For global events, the event attributes are updated for all the harts.

The possible error codes returned in `sbiret.error` are shown in Table 82 below. In case of errors with attribute values, the first error encountered (based on attributes ID order) is returned.

Table 82. SSE Event Attributes Write Errors

Error code	Description
SBI_SUCCESS	Event attribute values written successfully.
SBI_ERR_NOT_SUPPORTED	`event_id` is not reserved and valid, but the platform does not support it due to one or more missing dependencies (Hardware or SBI implementation).
SBI_ERR_INVALID_PARAM	Attribute write operation failed because either: - `event_id` is invalid - `attr_count` is zero - `event_id` is valid but one of the attribute values violates the legal values described in Table 80.
SBI_ERR_DENIED	`event_id` is valid but one of the attributes is read-only.
SBI_ERR_INVALID_STATE	`event_id` is valid but one of the attribute values violates the state rules described in Table 80.
SBI_ERR_BAD_RANGE	One of the event attribute IDs in the range specified by `base_attr_id` and `attr_count` is reserved.
SBI_ERR_INVALID_ADDRESS	The shared memory pointed to by the `input_phys_lo` and `input_phys_hi` parameters does not satisfy the requirements described in Section 3.2.
SBI_ERR_FAILED	The write failed for unspecified or unknown other reasons.

17.9. Function: Register a software event (FID #2)

17.8. 功能：写入软件事件属性（功能号#1） struct sbiret sbi_sse_write_attrs(uint32_t event_id,

```
                                                uint32_t base_attr_id, uint32_t attr_count, unsigned long
                                                input_phys_lo, unsigned long input_phys_hi)
```

向软件事件写入一组事件属性值。

event_id 参数指定软件事件 ID，而 base_attr_id 和 attr_count 参数则指定事件属性 ID 的范围。
事件属性值从输入共享内存中读取，该内存由 input_phys_lo 和 input_phys_hi 参数指定，其中：

- input_phys_lo 参数必须按 XLEN/8 字节对齐
- 输入共享内存的大小假定为(XLEN/8)*attr_count
- 标识符为 base_attr_id + i 的事件属性值应从偏移量(XLEN/8)*(base_attr_id + i)

对于本地事件，事件属性仅更新调用 hart 的对应值。对于全局事件，事件属性将更新所有 hart 的对应值。
可能返回的 sbiret.error 错误码如下表 82 所示。若属性值存在错误，将按属性 ID 顺序返回首个遇到的错误。

表 82. SSE 事件属性写入错误

错误代码	描述
SBI_SUCCESS	事件属性值写入成功。
SBI_ERR_NOT_SUPPORTED	event_id 未被保留且有效，但由于缺少一个或多个依赖项（硬件或 SBI 实现），平台不支持该事件。
SBI_ERR_INVALID_PARAM	属性写入操作失败，原因可能是：- event_id 无效 - attr_count 为零 - event_id 有效但某个属性值违反表 80 中描述的合法值范围。
SBI_ERR_DENIED（访问被拒绝）	事件 ID 有效但其中一个属性为只读。
SBI_ERR_INVALID_STATE	事件 ID 有效但其中一个属性值违反了表 80 中描述的状态规则。
SBI_ERR_BAD_RANGE	在 base_attr_id 和 attr_count 指定范围内的事件属性 ID 之一已被保留。
SBI_ERR_INVALID_ADDRESS	由 input_phys_lo 和 input_phys_hi 参数指向的共享内存不符合第 3.2 节所述要求。
SBI_ERR_FAILED	由于未指定或其他未知原因导致写入失败。

17.9. 功能：注册软件事件（功能 ID #2）

```
struct sbiret sbi_sse_register(uint32_t event_id,
                               unsigned long handler_entry_pc,
                               unsigned long handler_entry_arg)
```

Register an event handler for the software event.

The `event_id` parameter specifies the event ID for which an event handler is being registered. The `handler_entry_pc` parameter MUST be 2-bytes aligned and specifies the `ENTRY_PC` event attribute of the software event whereas the `handler_entry_arg` parameter specifies the `ENTRY_ARG` event attribute of the software event.

For local events, the event is registered only for the calling hart. For global events, the event is registered for all the harts.

The event MUST be in `UNUSED` state otherwise this function will fail.

It is advisable to use different values for `handler_entry_arg` for different events because higher priority events preempt lower priority events.

Upon success, the event state moves from `UNUSED` to `REGISTERED`. In case of an error, possible error codes are listed in Table 83 below.

Table 83. SSE Event Register Errors

Error code	Description
SBI_SUCCESS	Event handler is registered successfully.
SBI_ERR_NOT_SUPPORTED	`event_id` is not reserved and valid, but the platform does not support it due to one or more missing dependencies (Hardware or SBI implementation).
SBI_ERR_INVALID_STATE	`event_id` is valid but the event is not in `UNUSED` state.
SBI_ERR_INVALID_PARAM	`event_id` is invalid or `handler_entry_pc` is not 2-bytes aligned.

17.10. Function: Unregister a software event (FID #3)

```
struct sbiret sbi_sse_unregister(uint32_t event_id)
```

Unregister the event handler for given `event_id`.

For local events, the event is unregistered only for the calling hart. For global events, the event is unregistered for all the harts.

The event MUST be in `REGISTERED` state otherwise this function will fail.

Upon success, the event state moves from `REGISTERED` to `UNUSED`. In case of an error, possible error codes are listed in Table 84 below.

Table 84. SSE Event Unregister Errors

```
结构体 sbiret sbi_sse_register(uint32_t event_id,
                                        unsigned long handler_entry_pc, unsigned
                                        long handler_entry_arg)
```

为软件事件注册一个事件处理程序。

event_id 参数指定了要为其注册事件处理程序的事件 ID。
handler_entry_pc 参数必须按 2 字节对齐，用于指定软件事件的 ENTRY_PC 事件属性；而 handler_entry_arg 参数则用于指定软件事件的 ENTRY_ARG 事件属性。
对于本地事件，该事件仅注册给发起调用的 hart。对于全局事件，该事件将注册给所有 hart。

事件必须处于 UNUSED（未使用）状态，否则此函数将执行失败。

建议为不同事件设置不同的 handler_entry_arg 值，因为高优先级事件会抢占低优先级事件。

成功时，事件状态将从"未使用"转为"已注册"。若出现错误，可能的错误代码如下表 83 所示。

表 83. SSE 事件寄存器错误

错误代码	描述
SBI_SUCCESS	事件处理程序注册成功。
SBI_ERR_NOT_SUPPORTED	event_id 未被保留且有效，但由于缺少一个或多个依赖项（硬件或 SBI 实现），平台不支持该事件。
SBI_ERR_INVALID_STATE	event_id 有效但事件不处于未使用状态。
SBI_ERR_INVALID_PARAM	事件 ID 无效或处理程序入口地址未按 2 字节对齐

17.10. 功能函数：注销软件事件（功能号#3） struct sbiret sbi_sse_unregister(uint32_t event_id)

注销指定 event_id 对应的事件处理程序

对于本地事件，仅注销调用该函数的 hart 对应事件；对于全局事件，将为所有 hart 注销该事件

该事件必须处于已注册(REGISTERED)状态，否则此函数将执行失败。

成功时，事件状态将从已注册(REGISTERED)转为未使用(UNUSED)。若出现错误，可能的错误代码如下表 84 所示。

表 84. SSE 事件注销错误码

Error code	Description
SBI_SUCCESS	Event handler is unregistered successfully.
SBI_ERR_NOT_SUPPORTED	`event_id` is not reserved and valid, but the platform does not support it due to one or more missing dependencies (Hardware or SBI implementation).
SBI_ERR_INVALID_STATE	`event_id` is valid but the event is not in `REGISTERED` state.
SBI_ERR_INVALID_PARAM	`event_id` is invalid.

17.11. Function: Enable a software event (FID #4)

```
struct sbiret sbi_sse_enable(uint32_t event_id)
```

Enable the software event specified by the `event_id` parameter.

For local events, the event is enabled only for the calling hart. For global events, the event is enabled for all the harts.

The event MUST be in `REGISTERED` state otherwise this function will fail.

Upon success, the event state moves from `REGISTERED` to `ENABLED`. In case of an error, possible error codes are listed in Table 85 below.

Table 85. SSE Event Enable Errors

Error code	Description
SBI_SUCCESS	Event is successfully enabled.
SBI_ERR_NOT_SUPPORTED	`event_id` is not reserved and valid, but the platform does not support it due to one or more missing dependencies (Hardware or SBI implementation).
SBI_ERR_INVALID_PARAM	`event_id` is not valid.
SBI_ERR_INVALID_STATE	`event_id` is valid but the event is not in `REGISTERED` state.

17.12. Function: Disable a software event (FID #5)

```
struct sbiret sbi_sse_disable(uint32_t event_id)
```

Disable the software event specified by the `event_id` parameter.

For local events, the event is disabled only for the calling hart. For global events, the event is disabled for all the harts.

The event MUST be in `ENABLED` state otherwise this function will fail.

Upon success, the event state moves from `ENABLED` to `REGISTERED`. In case of an error, possible error codes are listed in Table 86 below.

Table 86. SSE Event Disable Errors

错误代码	描述
SBI_SUCCESS	事件处理程序已成功注销。
SBI_ERR_NOT_SUPPORTED	event_id 未被保留且有效，但由于缺少一个或多个依赖项（硬件或 SBI 实现），平台不支持该事件。
SBI_ERR_INVALID_STATE	event_id 有效但事件未处于已注册状态。
SBI_ERR_INVALID_PARAM	event_id 无效。

17.11. 功能：启用软件事件（FID #4） struct sbiret sbi_sse_enable(uint32_t event_id)

启用由 event_id 参数指定的软件事件。

对于本地事件，该事件仅对调用该功能的 hart 启用。对于全局事件，该事件对所有 hart 启用。

该事件必须处于已注册(REGISTERED)状态，否则此函数将执行失败。

成功时，事件状态将从已注册(REGISTERED)转为已启用(ENABLED)。若出现错误，可能的错误代码如下表 85 所示。

表 85. SSE 事件启用错误码

错误代码	描述
SBI_SUCCESS	事件已成功启用。
SBI_ERR_NOT_SUPPORTED	事件 ID 未被保留且有效，但由于缺少一个或多个依赖项（硬件或 SBI 实现），平台不支持该事件。
SBI_ERR_INVALID_PARAM	事件 ID 无效。
SBI_ERR_INVALID_STATE	事件 ID 有效但该事件未处于已注册状态。

17.12. 功能：禁用软件事件（功能号 #5）struct sbiret sbi_sse_disable(uint32_t event_id)

禁用由 event_id 参数指定的软件事件。

对于本地事件，该事件仅对调用该功能的硬件线程禁用。对于全局事件，该事件将对所有硬件线程禁用。

事件必须处于已启用状态，否则此功能将执行失败。

成功时，事件状态将从 ENABLED 转为 REGISTERED。若出现错误，可能的错误代码如下表 86 所示。

表 86. SSE 事件禁用错误

Error code	Description
SBI_SUCCESS	Event is successfully disabled.
SBI_ERR_NOT_SUPPORTED	`event_id` is not reserved and valid, but the platform does not support it due to one or more missing dependencies (Hardware or SBI implementation).
SBI_ERR_INVALID_PARAM	`event_id` is not valid.
SBI_ERR_INVALID_STATE	`event_id` is valid but the event is not in `ENABLED` state.

17.13. Function: Complete software event handling (FID #6)

```
struct sbiret sbi_sse_complete(void)
```

Complete the supervisor event handling for the highest priority event in `RUNNING` state on the calling hart.

If there were no events in `RUNNING` state on the calling hart then this function does nothing and returns `SBI_SUCCESS` otherwise it moves the highest priority event in `RUNNING` state to:

- `REGISTERED` if the event is configured as one-shot (see the `CONFIG` attribute in Table 80.)
- `ENABLED` state otherwise

It then resumes the interrupted supervisor state as described in Section 17.6.

17.14. Function: Inject a software event (FID #7)

```
struct sbiret sbi_sse_inject(uint32_t event_id, unsigned long hart_id)
```

The supervisor software can inject a software event with this function. The `event_id` paramater refers to the ID of the event to be injected.

For local events, the `hart_id` parameter refers to the hart on which the event is to be injected. For global events, the `hart_id` parameter is ignored.

An event can only be injected if it is allowed by the event attribute as described in Table 80.

If an event is injected from within an SSE event handler, if it is ready to be run, it will be handled according to the priority rules described in Section 17.3:

- If it has a higher priority than the one currently running, then it will be handled immediately, effectively preempting the currently running one.
- If it has a lower priority, it will be run after the one that is currently running completes.

In case of an error, possible error codes are listed in Table 87 below.

Table 87. SSE Event Inject Errors

Error code	Description
SBI_SUCCESS	Event is successfully injected.

错误代码	描述
SBI_SUCCESS	事件已成功禁用。
SBI_ERR_NOT_SUPPORTED	event_id 有效且未被保留，但由于缺少一个或多个依赖项（硬件或 SBI 实现），平台不支持该事件。
SBI_ERR_INVALID_PARAM	event_id 无效。
SBI_ERR_INVALID_STATE	event_id 有效但事件未处于 ENABLED 状态

17.13. 功能函数：完成软件事件处理（FID #6） struct sbiret sbi_sse_complete(void)

为调用 hart 上处于 RUNNING 状态的最高优先级事件完成监管程序事件处理

若调用 hart 上不存在处于 RUNNING 状态的事件，则该函数不执行任何操作并直接返回 SBI_SUCCESS 否则它将处于运行状态的最高优先级事件转移至：

- 已注册状态（如果该事件被配置为单次触发，参见表 80 中的 CONFIG 属性）
- 否则转移至已启用状态

随后按照第 17.6 节所述恢复被中断的管理程序状态

17.14. 功能：注入软件事件（功能号 #7） struct sbiret sbi_sse_inject(uint32_t event_id, unsigned long hart_id)

监管者软件可通过此功能注入软件事件。event_id 参数表示待注入事件的 ID。
对于本地事件，hart_id 参数指定事件将被注入的目标硬件线程。对于全局事件，hart_id 参数将被忽略。

只有当事件属性允许时（如表 80 所述）才能注入事件。

若从 SSE 事件处理程序内部注入一个事件，且该事件已准备就绪，则将根据第 17.3 节所述的优先级规则进行处理：

- 若其优先级高于当前正在运行的事件，则将立即处理该事件，从而有效抢占当前运行事件
- 若其优先级较低，则将在当前运行事件完成后处理该事件

若发生错误，下表 87 列出了可能的错误代码。

表 87. SSE 事件注入错误

错误代码	描述
SBI_SUCCESS	事件注入成功。

Error code	Description
SBI_ERR_NOT_SUPPORTED	`event_id` is not reserved and valid, but the platform does not support it due to one or more missing dependencies (Hardware or SBI implementation).
SBI_ERR_INVALID_PARAM	`event_id` or `hart_id` is invalid.
SBI_ERR_FAILED	The injection failed for unspecified or unknown other reasons.

17.15. Function: Unmask software events on a hart (FID #8)

```
struct sbiret sbi_sse_hart_unmask(void)
```

Start receiving (or unmask) software events on the calling hart. In other words, the calling hart is ready to receive software events from the SBI implementation.

The software events are masked initially on all harts so the supervisor software must explicitly unmask software events on relevant harts at boot-time.

In case of an error, possible error codes are listed in Table 88 below.

Table 88. SSE Hart Unmask Errors

Error code	Description
SBI_SUCCESS	Software events unmasked successfully on the calling hart.
SBI_ERR_ALREADY_STARTED	Software events were already unmasked on the calling hart.
SBI_ERR_FAILED	The request failed for unspecified or unknown other reasons.

17.16. Function: Mask software events on a hart (FID #9)

```
struct sbiret sbi_sse_hart_mask(void)
```

Stop receiving (or mask) software events on the calling hart. In other words, the calling hart will no longer be ready to receive software events from the SBI implementation.

In case of an error, possible error codes are listed in Table 89 below.

Table 89. SSE Hart Mask Errors

Error code	Description
SBI_SUCCESS	Software events masked successfully on the calling hart.
SBI_ERR_ALREADY_STOPPED	Software events were already masked on the calling hart.
SBI_ERR_FAILED	The request failed for unspecified or unknown other reasons.

17.17. Function Listing

Table 90. SSE Function List

错误代码	描述
SBI_ERR_NOT_SUPPORTED	event_id 未被保留且有效，但由于缺少一个或多个依赖项（硬件或 SBI 实现），平台不支持该功能。
SBI_ERR_INVALID_PARAM	事件 ID 或硬件线程 ID 无效。
SBI_ERR_FAILED	由于未指定或其他未知原因导致注入失败。

17.15. 功能：解除硬件线程上的软件事件屏蔽（功能号#8） struct sbiret sbi_sse_hart_unmask(void)

开始接收（或解除屏蔽）当前硬件线程上的软件事件。换言之，当前硬件线程已准备好接收来自 SBI 实现的软件事件。
所有硬件线程上的软件事件初始状态均为屏蔽状态，因此监督模式软件必须在启动时显式解除相关硬件线程上的软件事件屏蔽。

若出现错误，可能返回的错误代码如下表 88 所示。

表 88. SSE 硬件线程解除屏蔽错误代码

错误代码	描述
SBI_SUCCESS	调用硬件线程上的软件事件已成功解除屏蔽。
SBI_ERR_ALREADY_STARTED	调用硬件线程上的软件事件已解除屏蔽。
SBI_ERR_FAILED	请求因未指定或其他未知原因失败。

17.16. 功能：屏蔽硬件线程上的软件事件（功能号 #9） struct sbiret sbi_sse_hart_mask(void)

停止接收（或屏蔽）调用硬件线程上的软件事件。换言之，调用硬件线程将不再准备接收来自 SBI 实现的软件事件。

若发生错误，可能的错误代码列于下表 89。

表 89. SSE 硬件线程掩码错误

错误代码	描述
SBI_SUCCESS	调用硬件线程上的软件事件已成功屏蔽。
SBI_ERR_ALREADY_STOPPED	调用硬件线程上的软件事件已被屏蔽。
SBI_ERR_FAILED	请求因未指定或其他未知原因失败。

17.17. 函数列表

表 90. SSE 函数列表

Function Name	SBI Version	FID	EID
sbi_sse_read_attrs	3.0	0	0x535345
sbi_sse_write_attrs	3.0	1	0x535345
sbi_sse_register	3.0	2	0x535345
sbi_sse_unregister	3.0	3	0x535345
sbi_sse_enable	3.0	4	0x535345
sbi_sse_disable	3.0	5	0x535345
sbi_sse_complete	3.0	6	0x535345
sbi_sse_inject	3.0	7	0x535345
sbi_sse_hart_unmask	3.0	8	0x535345
sbi_sse_hart_mask	3.0	9	0x535345

功能名称	SBI 版本 FID EID		
sbi_sse_read_attrs	3.0	0 0x535345	
sbi_sse_write_attrs	3.0	1	0x535345
sbi_sse_register	3.0	2	0x535545
sbi_sse_unregister	3.0	3	0x535545
sbi_sse_enable	3.0	4	0x535545
sbi_sse_disable	3.0	5	0x535545
sbi_sse_complete	3.0	6 0x535545	
sbi_sse_inject	3.0	7	0x535545
sbi_sse_hart_unmask	3.0	8	0x535450
sbi_sse_hart_mask	3.0	9 0x535450	

Chapter 18. SBI Firmware Features Extension (EID #0x46574654 "FWFT")

The Firmware Features extension enables supervisor-mode software to manage and control specific hardware capabilities or SBI implementation features. Table 91 defines 32-bit identifiers for the features which supervisor-mode software may request to set or get.

Table 91. FWFT Feature Types

Value	Name	Description
0x00000000	MISALIGNED_EXC_DELEG	Control misaligned access exception delegation to supervisor-mode
0x00000001	LANDING_PAD	Control landing pad support for supervisor-mode.
0x00000002	SHADOW_STACK	Control shadow stack support for supervisor-mode.
0x00000003	DOUBLE_TRAP	Control double trap support.
0x00000004	PTE_AD_HW_UPDATING	Control hardware updating of PTE A/D bits.
0x00000005	POINTER_MASKING_PMLEN	Control the pointer masking length for supervisor-mode.
0x00000006 - 0x3fffffff		Local feature types reserved for future use.
0x40000000 - 0x7fffffff		Platform specific local feature types.
0x80000000 - 0xbfffffff		Global feature types reserved for future use.
0xc0000000 - 0xffffffff		Platform specific global feature types.

These features have some attributes that define their behavior and are described in Table 92. The attribute values are defined for each feature in Table 93.

Table 92. FWFT Feature Attributes

Attribute	Description
Scope	Defines if a feature is local (per-hart) or global. Global features only need to be enabled/disabled by a single hart, whereas local features need to be enabled/disabled by each hart. The status and flags of local features can be different from one hart to another.
Reset value	Reset value of the feature. Might be implementation defined.
Values	Per feature values that can be set.

During non-retentive suspend, feature values are retained and restored by the SBI when resuming operations. Upon hart reset, local feature values are not retained and reset to their default reset values according to the feature description. Upon system reset, global and local feature values are reset.

Table 93. FWFT Feature Attribute Values

第 18 章 SBI 固件特性扩展（扩展 ID #0x46574654 "FWFT"）

固件特性扩展使监管模式软件能够管理和控制特定的硬件能力或 SBI 实现特性。表 91 定义了监管模式软件可请求设置或获取的 32 位特性标识符。

表 91. FWFT 特性类型

数值	名称	描述
0x00000000	MISALIGNED_EXC_DELEG	控制未对齐访问异常委托给监管者模式
0x00000001	LANDING_PAD	控制监督模式下的着陆垫支持功能
0x00000002	影子栈	控制监督模式下的影子栈支持功能
0x00000003	双重陷阱	控制双重陷阱支持
0x00000004	PTE_AD_HW_UPDATING	控制 PTE A/D 位的硬件更新
0x00000005	POINTER_MASKING_PMLEN	控制监督模式下的指针掩码长度。
0x00000006 - 0x3fffffff		为未来使用保留的本地特性类型。
0x40000000 - 0x7fffffff		平台特定的本地功能类型。
0x80000000 - 0xbfffffff		为未来使用保留的全局特性类型。
0xc0000000 - 0xffffffff		平台特定的全局功能类型。

这些功能具有定义其行为的若干属性，详见表 92。各功能的属性值定义见表 93。

表 92. FWFT 功能属性

属性	描述
作用范围	定义某个特性是局部的（每个硬件线程独有）还是全局的。全局特性只需由单个硬件线程启用/禁用，而局部特性需要每个硬件线程分别启用/禁用。局部特性的状态和标志在不同硬件线程之间可能存在差异。
复位值	该特性的复位值。可能由具体实现定义。
数值	可设置的各特性数值

在非保持性挂起期间，特性值由 SBI 在恢复操作时保留并恢复。当 hart 复位时，本地特性值不会被保留，而是根据特性描述重置为默认复位值。系统复位时，全局和本地特性值均会被重置。

表 93. FWFT 特性属性值

Feature Name	Reset	Scope	Values	
MISALIGNED_EXC_DELEG	Implementation-defined	Local	0	Disable misaligned exception delegation.
			1	Enable misaligned exception delegation.
LANDING_PAD	0	Local	0	Disable landing pad for supervisor-mode.
			1	Enable landing pad for supervisor-mode.
SHADOW_STACK	0	Local	0	Disable shadow-stack for supervisor-mode.
			1	Enable shadow-stack for supervisor-mode.
DOUBLE_TRAP	0	Local	0	Disable double trap
			1	Enable double trap
PTE_AD_HW_UPDATING	0	Local	0	Disable hardware updating of PTE A/D bits for supervisor-mode.
			1	Enable hardware updating of PTE A/D bits for supervisor-mode.
POINTER_MASKING_PMLEN	0	Local	0	Disable pointer masking for supervisor-mode.
			N	Enable pointer masking for supervisor-mode with PMLEN = N.

18.1. Function: Firmware Features Set (FID #0)

```
struct sbiret sbi_fwft_set(uint32_t feature,
                           unsigned long value,
                           unsigned long flags)
```

A successful return from `sbi_fwft_set()` results in the requested firmware feature to be set according to the `value` and `flags` parameters for which per feature supported values are described in Table 93 and flags in Table 94.

The set operation will succeed if requested `value` *matches the existing value.*

Table 94. FWFT Firmware Features Set Flags

Name	Encoding	Description
LOCK	BIT[0]	If provided, once set, the feature value can no longer be modified until: - hart reset for feature with local scope - system reset for feature with global scope
	BIT[XLEN-1:1]	Reserved for future use and must be zero.

特性名称	复位	范围	数值
未对齐异常委托 G	实现 实现定义	本地	0 禁用未对齐异常委托。 1 启用未对齐异常委托。
着陆垫	0	本地	0 禁用监管模式的着陆垫。 1 启用监督模式着陆区
影子栈	0	局部	0 禁用监管者模式下的影子栈 1 启用监管者模式下的影子栈
双重陷阱	0	本地	0 禁用双重陷阱 1 启用双重陷阱
PTE_AD_HW_UPDATING 0		本地	0 禁用监管模式下 PTE A/D 位的硬件更新 1 启用监管模式下 PTE A/D 位的硬件更新
POINTER_MASKING_PMLEN	0	本地	0 禁用监管者模式的指针掩码功能。 N 启用监管者模式的指针掩码功能，掩码长度为 N。

18.1. 功能：固件特性设置（功能号 #0）

```
struct sbiret sbi_fwft_set(uint32_t feature,
                           unsigned long value,
                           unsigned long flags)
```

sbi_fwft_set()调用成功时，将根据 value 和 flags 参数设置请求的固件特性，各特性支持的 value 值见表 93，flags 定义见表 94。

☒ 若请求值与当前值一致，设置操作将视为成功。

表 94. FWFT 固件特性集标志位

名称	编码	描述
锁定	第 0 位	若提供该功能，一旦设置后，在以下情况前不可修改特征值： - 对于局部作用域的功能，需 hart 复位 - 对于全局作用域的功能，需系统复位
	第[XLEN-1:1]位 保留未来使用且必须置零	

In case of failure, `feature` value is not modified and the possible error codes returned in `sbiret.error` are shown in Table 95 below.

Table 95. FWFT Firmware Features Set Errors

Error code	Description
SBI_SUCCESS	`feature` was set successfully.
SBI_ERR_NOT_SUPPORTED	`feature` is not reserved and valid, but the platform does not support it due to one or more missing dependencies (Hardware or SBI implementation).
SBI_ERR_INVALID_PARAM	Provided `value` or `flags` parameter is invalid.
SBI_ERR_DENIED	`feature` set operation failed because either: - it was denied by the SBI implementation - `feature` is reserved or is platform-specific and unimplemented
SBI_ERR_DENIED_LOCKED	`feature` set operation failed because the `feature` is locked
SBI_ERR_FAILED	The set operation failed for unspecified or unknown other reasons.

The rationale for an SBI implementation to return `SBI_ERR_DENIED` is for instance to allow some hypervisors to simply passthrough the misaligned delegation state to the Guest/VM and deny any changes to that delegation state from the Guest/VM. If authorized, an SBI call would be required at each Guest/VM switch if delegation choices are different between Host and Guest/VM.

18.2. Function: Firmware Features Get (FID #1)

```
struct sbiret sbi_fwft_get(uint32_t feature)
```

A successful return from `sbi_fwft_get()` results in the firmware feature configuration value to be returned in `sbiret.value`. Possible `sbiret.value` values are described in Table 93 for each feature ID.

In case of failure, the content of `sbiret.value` is zero and the possible error codes returned in `sbiret.error` are shown in Table 96.

Table 96. FWFT Firmware Features Get Errors

Error code	Description
SBI_SUCCESS	Feature status was retrieved successfully.
SBI_ERR_NOT_SUPPORTED	`feature` is not reserved and valid, but the platform does not support it due to one or more missing dependencies (Hardware or SBI implementation).
SBI_ERR_DENIED	`feature` is reserved or is platform-specific and unimplemented.
SBI_ERR_FAILED	The get operation failed for unspecified or unknown other reasons.

18.3. Function Listing

Table 97. FWFT Function List

Function Name	SBI Version	FID	EID
sbi_fwft_set	3.0	0	0x46574654
sbi_fwft_get	3.0	1	0x46574654

若执行失败，则特征值不会被修改，且可能返回的错误码（通过 sbiret.error）如下表 95 所示。

表 95. FWFT 固件特性设置错误

错误代码	描述
SBI_SUCCESS	特性设置成功。
SBI_ERR_NOT_SUPPORTED	特性未被保留且有效，但由于缺少一个或多个依赖项（硬件或 SBI 实现），平台不支持该特性。
SBI_ERR_INVALID_PARAM	提供的值或标志参数无效。
SBI_ERR_DENIED	功能集操作失败，原因可能是：- 被 SBI 实现拒绝 - 该功能为保留项或平台特定功能且未实现
SBI_ERR_DENIED_LOCKED	功能集操作失败，因为该功能已被锁定
SBI_ERR_FAILED	设置操作因未指定或其他未知原因失败。

SBI 实现返回 SBI_ERR_DENIED 的典型场景是：允许某些虚拟机监控程序直接将未对齐的委托状态透传给客户机/虚拟机，同时禁止客户机/虚拟机修改该委托状态。若允许修改，当主机与客户机/虚拟机的委托选择不同时，每次切换客户机/虚拟机都需要执行 SBI 调用。

18.2. 功能：固件特性获取（FID #1） struct sbiret sbi_fwft_get(uint32_t feature)

sbi_fwft_get()调用成功时，固件特性配置值将通过 sbiret.value 返回。表 93 描述了各特性 ID 对应的 sbiret.value 可能取值。
调用失败时，sbiret.value 内容为零，可能返回的错误代码包括

`sbiret.error` 如表 96 所示。

表 96. FWFT 固件特性获取错误

错误代码	描述
SBI_SUCCESS	特性状态已成功获取。
SBI_ERR_NOT_SUPPORTED	该特性未被保留且有效，但由于缺少一个或多个依赖项（硬件或 SBI 实现）导致平台不支持该特性。
SBI_ERR_DENIED	该特性已被保留或是平台特定且未实现的功能。
SBI_ERR_FAILED	获取操作因未指定或其他未知原因失败。

18.3. 功能列表

表 97. FWFT 功能列表

功能名称	SBI 版本	功能标识符	扩展 ID
sbi_fwft_set	3.0	0	0x46574654
sbi_fwft_get	3.0	1	0x46574654

Chapter 19. Debug Triggers Extension (EID #0x44425452 "DBTR")

The RISC-V Sdtrig extension [2] allows machine-mode software to directly configure debug triggers which in-turn allows native (or hosted) debugging in machine-mode without any external debugger. Unfortunately, the debug triggers are only accessible to machine-mode.

The SBI debug trigger extension defines a SBI based abstraction to provide native debugging for supervisor-mode software such that it is:

1. Suitable for the rich operating systems and hypervisors running in supervisor-mode.
2. Allows Guest (VS-mode) and Hypervisor (HS-mode) to share debug triggers on a hart.

Each hart on a RISC-V platform has a fixed number of debug triggers which is referred to as `trig_max` in this SBI extension. Each debug trigger is assigned a logical index called `trig_idx` by the SBI implementation where `-1 < trig_idx < trig_max`.

> *The* `trig_max` *may vary across harts on a platform with asymmetric harts.*

The configuration of each debug trigger is expressed by three parameters `trig_tdata1`, `trig_tdata2`, and `trig_tdata3` which are encoded in the same way as the `tdata1`, `tdata2`, and `tdata3` CSRs defined by the RISC-V Sdtrig extension [2] but with the following additional constraints:

1. The `trig_tdata1.dmode` bit must always be zero.
2. The `trig_tdata1.m` bit must always be zero.

The SBI implementation MUST also maintain an additional software state for each debug trigger called `trig_state` which is encoded as shown in Table 98 below.

Table 98. Debug Trigger State Fields

Field Name	Bits	Description
hw_trig_idx	`trig_state[XLEN-1:8]`	hardware (or physical) index of the debug trigger. This field must be ignored when `trig_state.have_hw_trig == 0`.
reserved	`trig_state[7:6]`	Reserved for future use and must be zero.
have_hw_trig	`trig_state[5:5]`	When set, the hardware (or physical) debug trigger details are available.
vs	`trig_state[4:4]`	Saved copy of the `trig_tdata1.vs` bit.
vu	`trig_state[3:3]`	Saved copy of the `trig_tdata1.vu` bit.
s	`trig_state[2:2]`	Saved copy of the `trig_tdata1.s` bit.
u	`trig_state[1:1]`	Saved copy of the `trig_tdata1.u` bit.
mapped	`trig_state[0:0]`	When set, the trigger has been mapped to some HW debug trigger.

19.1. Function: Get number of triggers (FID #0)

```
struct sbiret sbi_debug_num_triggers(unsigned long trig_tdata1)
```

第 19 章 调试触发器扩展（EID #0x44425452 "DBTR"）

RISC-V Sdtrig 扩展[2]允许机器模式软件直接配置调试触发器，从而无需外部调试器即可在机器模式下实现原生（或托管）调试。
遗憾的是，调试触发器仅可由机器模式访问。

SBI 调试触发器扩展定义了基于 SBI 的抽象层，旨在为运行在监管模式下的丰富操作系统和虚拟机监控程序提供原生调试支持，其特性包括：

1. 适用于运行在监管模式下的复杂操作系统和虚拟机监控程序。
2. 允许客户机（VS 模式）与管理程序（HS 模式）共享硬件线程上的调试触发器。

RISC-V 平台上的每个硬件线程都拥有固定数量的调试触发器，本 SBI 扩展中将其称为 trig_max。每个调试触发器由 SBI 实现分配一个逻辑索引 trig_idx，其范围为-1 < trig_idx < trig_max。

⚠️ 注意：在具有非对称硬件线程的平台上，不同硬件线程间的该数值可能存在差异。

每个调试触发器的配置由三个参数表示：trig_tdata1、trig_tdata2 以及
trig_tdata3，其编码方式与 RISC-V Sdtrig 扩展[2]定义的 tdata1、tdata2 和 tdata3 控制状态寄存器相同，但需遵循以下额外约束：

1. trig_tdata1.dmode 位必须始终为零
2. trig_tdata1.m 位必须始终为零

SBI 实现还必须为每个调试触发器维护一个额外的软件状态，称为
触发器状态 其编码方式如下表 98 所示。

表 98. 调试触发器状态字段

字段名称	位	描述
硬件触发索引	trig_state[XLEN-1:8]	调试触发器的硬件（或物理）索引。当 trig_state.have_hw_trig == 0 时，该字段必须被忽略。
保留字段	触发状态[7:6]	保留供未来使用，必须置零。
硬件触发支持标志	触发器状态[5:5]	当该位被置位时，表示硬件（或物理）调试触发器的详细信息可用。
对比	触发器状态[4:4]	已保存的 trig_tdata1.vs 位副本
vu	trig_state[3:3]	已保存的 trig_tdata1.vu 位副本
s	trig_state[2:2]	trig_tdata1.s 位的保存副本
u	trig_state[1:1]	trig_tdata1.u 位的保存副本
已映射	触发状态[0:0]	当该位被置位时，表示该触发器已映射到某个硬件调试触发器上

19.1. 功能：获取触发器数量（功能号 #0）
struct sbiret sbi_debug_num_triggers(unsigned long trig_tdata1)

Get the number of debug triggers on the calling hart which can support the trigger configuration specified by `trig_tdata1` parameter.

This function always returns SBI_SUCCESS in `sbiret.error`. It will return `trig_max` in `sbiret.value` when `trig_tdata1 == 0` otherwise it will return the number of matching debug triggers in `sbiret.value`.

19.2. Function: Set trigger shared memory (FID #1)

```
struct sbiret sbi_debug_set_shmem(unsigned long shmem_phys_lo,
                                  unsigned long shmem_phys_hi,
                                  unsigned long flags)
```

Set and enable the shared memory for debug trigger configuration on the calling hart.

If both `shmem_phys_lo` and `shmem_phys_hi` parameters are not all-ones bitwise then `shmem_phys_lo` specifies the lower XLEN bits and `shmem_phys_hi` specifies the upper XLEN bits of the shared memory physical base address. The `shmem_phys_lo` MUST be `(XLEN / 8)` bytes aligned and the size of shared memory is assumed to be `trig_max * (XLEN / 2)` bytes.

If both `shmem_phys_lo` and `shmem_phys_hi` parameters are all-ones bitwise then shared memory for debug trigger configuration is disabled.

The `flags` parameter is reserved for future use and MUST be zero.

The possible error codes returned in `sbiret.error` are shown in Table 99.

Table 99. Debug Triggers Set Shared Memory Errors

Error code	Description
SBI_SUCCESS	Shared memory was set or cleared successfully.
SBI_ERR_INVALID_PARAM	The `flags` parameter is not zero or the `shmem_phys_lo` parameter is not `(XLEN / 8)` bytes aligned.
SBI_ERR_INVALID_ADDRESS	The shared memory pointed to by the `shmem_phys_lo` and `shmem_phys_hi` parameters does not satisfy the requirements described in Section 3.2.
SBI_ERR_FAILED	The request failed for unspecified or unknown other reasons.

19.3. Function: Read triggers (FID #2)

```
struct sbiret sbi_debug_read_triggers(unsigned long trig_idx_base,
                                      unsigned long trig_count)
```

Read the debug trigger state and configuration into shared memory for a range of debug triggers specified by the `trig_idx_base` and `trig_count` parameters on the calling hart.

For each debug trigger with index `trig_idx_base + i` where `-1 < i < trig_count`, the debug trigger state and configuration consisting of four XLEN-bit words are written in little-endian format at `offset = i * (XLEN / 2)` of the shared memory as follows:

获取调用硬件线程上可支持由 trig_tdata1 参数指定的触发器配置的调试触发器数量。
该函数在 sbiret.error 中始终返回 SBI_SUCCESS。当 trig_tdata1 等于 0 时，将在 sbiret.value 中返回 trig_max，否则将在 sbiret.value 中返回匹配的调试触发器数量。

19.2. 函数：设置触发器共享内存（功能号#1） struct sbiret sbi_debug_set_shmem(unsigned long shmem_phys_lo,

```
                                    unsigned long shmem_phys_hi, unsigned
                                    long flags)
```

为当前硬件线程设置并启用用于调试触发器配置的共享内存空间。

若 shmem_phys_lo 和 shmem_phys_hi 参数不全为按位全 1 值，则 shmem_phys_lo 指定共享内存物理基地址的低 XLEN 位，shmem_phys_hi 指定高 XLEN 位。shmem_phys_lo 必须按(XLEN/8)字节对齐，且假定共享内存大小为 trig_max*(XLEN/2)字节。
若 shmem_phys_lo 和 shmem_phys_hi 参数均为按位全 1 值，则禁用调试触发器配置的共享内存功能。

flags 参数保留供未来使用，当前必须置零。

sbiret.error 可能返回的错误代码如表 99 所示。

表 99. 调试触发器设置共享内存错误代码

错误代码	描述
SBI_SUCCESS	共享内存设置或清除成功。
SBI_ERR_INVALID_PARAM	flags 参数不为零，或 shmem_phys_lo 参数未按(XLEN/8)字节对齐。
SBI_ERR_INVALID_ADDRESS 由 shmem_phys_lo 和 shmem_phys_hi 参数指向的共享内存不符合第 3.2 节所述要求。	
SBI_ERR_FAILED	请求因未指定或其他未知原因失败。

19.3. 功能：读取触发器（FID #2） struct sbiret sbi_debug_read_triggers(unsigned long trig_idx_base,

```
                                    unsigned long trig_count)
```

读取调试触发器的状态和配置到共享内存中，范围由调用硬件线程上的 trig_idx_base 和 trig_count 参数指定。
对于每个索引为 trig_idx_base + i（其中-1 < i < trig_count）的调试触发器，其状态和配置（由四个 XLEN 位字组成）将以小端格式写入偏移量 = i *

(XLEN / 2)　　共享内存的使用方式如下：

```
word[0] = `trig_state` written by the SBI implementation
word[1] = `trig_tdata1` written by the SBI implementation
word[2] = `trig_tdata2` written by the SBI implementation
word[3] = `trig_tdata3` written by the SBI implementation
```

The possible error codes returned in `sbiret.error` are shown in Table 100.

Table 100. Debug Triggers Read Errors

Error code	Description
SBI_SUCCESS	State and configuration of triggers read successfully.
SBI_ERR_NO_SHMEM	Shared memory for debug triggers is disabled.
SBI_ERR_BAD_RANGE	Either `trig_idx_base >= trig_max` or `trig_idx_base + trig_count >= trig_max`

19.4. Function: Install triggers (FID #3)

```
struct sbiret sbi_debug_install_triggers(unsigned long trig_count)
```

Install debug triggers based on an array of trigger configurations in the shared memory of the calling hart. The `trig_idx` assigned to each installed trigger configuration is written back in the shared memory.

The `trig_count` parameter represents the number of trigger configuration entries in the shared memory at offset `0x0`.

The i'th trigger configuration at `offset = i * (XLEN / 2)` in the shared memory consists of four consecutive XLEN-bit words in little-endian format which are organized as follows:

```
word[0] = `trig_idx` written back by the SBI implementation
word[1] = `trig_tdata1` read by the SBI implementation
word[2] = `trig_tdata2` read by the SBI implementation
word[3] = `trig_tdata3` read by the SBI implementation
```

The SBI implementation MUST consider trigger configurations in the increasing order of the array index and starting with array index `0`. To install a debug trigger for the trigger configuration at array index `i` in the shared memory, the SBI implementation MUST do the following:

1. Map an unused HW debug trigger which matches the trigger configuration to an an unused `trig_idx`.
2. Save a copy of the `trig_tdata1.vs`, `trig_tdata1.vu`, `trig_tdata1.s`, and `trig_tdata.u` bits in `trig_state`.
3. Update the `tdata1`, `tdata2`, and `tdata3` CSRs of the HW debug trigger.
4. Write `trig_idx` at `offset = i * (XLEN / 2)` in the shared memory.

Additionally for each trigger configuration chain in the shared memory, the SBI implementation MUST assign contiguous `trig_idx` values and contiguous HW debug triggers when installing the trigger configuration chain.

```
word[0] = SBI 实现写入的`trig_state`
word[1] = SBI 实现写入的`trig_tdata1`
word[2] = SBI 实现写入的`trig_tdata2`
word[3] = SBI 实现写入的`trig_tdata3`
```

sbiret.error 可能返回的错误码如表 100 所示。

表 100. 调试触发器读取错误

错误代码	描述
SBI_SUCCESS	成功读取触发器的状态和配置。
SBI_ERR_NO_SHMEM	调试触发器的共享内存功能未启用。
SBI_ERR_BAD_RANGE	trig_idx_base >= trig_max 或 trig_idx_base + trig_count >= trig_max

19.4. 功能：安装触发器（功能号 #3） struct sbiret sbi_debug_install_triggers(unsigned long trig_count)

根据调用硬件线程共享内存中的触发器配置数组安装调试触发器。
每个已安装触发器配置分配的 trig_idx 将被写回共享内存。

参数 trig_count 表示共享内存偏移量 0x0 处触发器配置条目的数量。
共享内存中偏移量 = i * (XLEN / 2) 处的第 i 个触发器配置由四个连续的小端格式 XLEN 位字组成，其组织结构如下：

```
字[0] = SBI 实现回写的`trig_idx`
字[1] = SBI 实现读取的`trig_tdata1`
字[2] = SBI 实现读取的`trig_tdata2`

字[3] = SBI 实现读取的`trig_tdata3`
```

SBI 实现必须按照数组索引从 0 开始的递增顺序处理触发器配置。要为共享内存中数组索引 i 处的触发器配置安装调试触发器，SBI 实现必须执行以下操作：

1. 将一个未使用的硬件调试触发器映射到未使用的 trig_idx，该触发器需匹配给定的触发器配置。

2. 在 trig_state 中保存 trig_tdata1.vs、trig_tdata1.vu、trig_tdata1.s 和 trig_tdata.u 位的副本。

3. 更新硬件调试触发器的 tdata1、tdata2 和 tdata3 控制状态寄存器。4. 在共享内存偏移量=i*(XLEN/2)处写入 trig_idx 值。

此外，对于共享内存中的每个触发器配置链，SBI 实现安装该配置链时，必须分配连续的 trig_idx 值并使用连续的硬件调试触发器。

The last trigger configuration in the shared memory MUST not have `trig_tdata1.chain == 1` for `trig_tdata1.type = 2 or 6` to prevent incomplete trigger configuration chain in the shared memory.

The `sbiret.value` is set to zero upon success or if shared memory is disabled whereas `sbiret.value` is set to the array index `i` of the failing trigger configuration upon other failures.

The possible error codes returned in `sbiret.error` are shown in Table 101.

Table 101. Debug Triggers Install Errors

Error code	Description
SBI_SUCCESS	Triggers installed successfully.
SBI_ERR_NO_SHMEM	Shared memory for debug triggers is disabled.
SBI_ERR_BAD_RANGE	`trig_count >= trig_max`
SBI_ERR_INVALID_PARAM	One of the trigger configuration words `trig_tdata1`, `trig_tdata2`, or `trig_tdata3` has an invalid value.
SBI_ERR_FAILED	Failed to assign `trig_idx` or HW debug trigger for one of the trigger configurations.
SBI_ERR_NOT_SUPPORTED	One of the trigger configuration can't be programmed due to unimplemented optional bits in `tdata1`, `tdata2`, or `tdata3` CSRs.

19.5. Function: Update triggers (FID #4)

```
struct sbiret sbi_debug_update_triggers(unsigned long trig_count)
```

Update already installed debug triggers based on a trigger configuration array in the shared memory of the calling hart.

The `trig_count` parameter represents the number of trigger configuration entries in the shared memory at offset `0x0`.

The i'th trigger configuration at `offset = i * (XLEN / 2)` in the shared memory consists of four consecutive XLEN-bit words in little-endian format as follows:

```
word[0] = `trig_idx` read by the SBI implementation
word[1] = `trig_tdata1` read by the SBI implementation
word[2] = `trig_tdata2` read by the SBI implementation
word[3] = `trig_tdata3` read by the SBI implementation
```

The SBI implementation MUST consider trigger configurations in the increasing order of array index and starting with array index `0`. To update a debug trigger based on trigger configuration at array index `i` in the shared memory, the SBI implementation MUST do the following:

1. Check and fail if any of the following constraints are not satisfied:
   a. `trig_idx` represents logical index of a installed debug trigger
   b. `trig_tdata1.type` matches with original installed debug trigger
   c. `trig_tdata1.chain` matches with original installed debug trigger

共享内存中的最后一个触发配置必须满足 trig_tdata1.chain != 1
trig_tdata1.type = 2 或 6 以防止共享内存中的触发器配置链不完整。

成功时或共享内存被禁用时，sbiret.value 会被置零；而在其他失败情况下，sbiret.value 会被设置为触发配置失败的数组索引 i。

sbiret.error 可能返回的错误代码如表 101 所示。

表 101. 调试触发器安装错误

错误码	描述
SBI_SUCCESS	触发器安装成功。
SBI_ERR_NO_SHMEM	调试触发器的共享内存功能未启用。
SBI_ERR_BAD_RANGE	触发计数 >= 触发上限
SBI_ERR_INVALID_PARAM	触发器配置字 trig_tdata1、trig_tdata2 或 trig_tdata3 中存在无效值。
SBI_ERR_FAILED	未能为某个触发器配置分配 trig_idx 或硬件调试触发器。
SBI_ERR_NOT_SUPPORTED 由于 tdata1、tdata2 或 tdata3 CSR 中存在未实现的可选位，导致无法编程某个触发器配置。	

19.5. 功能：更新触发器（FID #4） struct sbiret sbi_debug_update_triggers(unsigned long trig_count)

根据调用 hart 共享内存中的触发器配置数组，更新已安装的调试触发器。
trig_count 参数表示共享内存中偏移量 0x0 处的触发器配置条目数量。
共享内存中偏移量 = i * (XLEN / 2) 处的第 i 个触发器配置由四个连续的小端格式 XLEN 位字组成，具体如下：

```
字[0] = SBI 实现读取的`trig_idx`
字[1] = SBI 实现读取的`trig_tdata1`
字[2] = SBI 实现读取的`trig_tdata2`
字[3] = SBI 实现读取的`trig_tdata3`
```

SBI 实现必须按照数组索引从 0 开始的递增顺序处理触发器配置。要根据共享内存中数组索引 i 处的触发器配置更新调试触发器，SBI 实现必须执行以下操作：

1. 检查以下约束条件，若任一条件不满足则判定失败：
a. trig_idx 表示已安装调试触发器的逻辑索引

b. trig_tdata1.type 与原始安装的调试触发器相匹配

c. trig_tdata1.chain 与原始安装的调试触发器相匹配

2. Save a copy of the `trig_tdata1.vs`, `trig_tdata1.vu`, `trig_tdata1.s`, and `trig_tdata.u` bits in `trig_state`.
3. Update the `tdata1`, `tdata2`, and `tdata3` CSRs of the HW debug trigger.

The `sbiret.value` is set to zero upon success or if shared memory is disabled whereas `sbiret.value` is set to the array index `i` of the failing trigger configuration upon other failures.

The possible error codes returned in `sbiret.error` are shown in Table 102.

Table 102. Debug Triggers Update Errors

Error code	Description
SBI_SUCCESS	Triggers updated successfully.
SBI_ERR_NO_SHMEM	Shared memory for debug triggers is disabled.
SBI_ERR_BAD_RANGE	`trig_count >= trig_max`
SBI_ERR_INVALID_PARAM	One of the trigger configuration in the shared memory has an invalid of `trig_idx` (i.e. `trig_idx >= trig_max`), `trig_tdata1`, `trig_tdata2`, or `trig_tdata3`.
SBI_ERR_FAILED	One of the trigger configurations has valid `trig_idx` but the corresponding debug trigger is not mapped to any HW debug trigger.
SBI_ERR_NOT_SUPPORTED	One of the trigger configuration can't be programmed due to unimplemented optional bits in `tdata1`, `tdata2`, or `tdata3` CSRs.

19.6. Function: Uninstall a set of triggers (FID #5)

```
struct sbiret sbi_debug_uninstall_triggers(unsigned long trig_idx_base,
                                           unsigned long trig_idx_mask)
```

Uninstall a set of debug triggers specified by the `trig_idx_base` and `trig_idx_mask` parameters on the calling hart. The `trig_idx_base` specifies the starting trigger index, while the `trig_idx_mask` is a bitmask indicating which triggers, relative to the base, are to be uninstalled. Each bit in the mask corresponds to a specific trigger, allowing for batch operations on multiple triggers simultaneously.

For each debug trigger in the specified set of debug triggers, the SBI implementation MUST:

1. Clear the `tdata1`, `tdata2`, and `tdata3` CSRs of the mapped HW debug trigger.
2. Clear the `trig_state` of the debug trigger.
3. Unmap and free the HW debug trigger and corresponding `trig_idx` for re-use in the future trigger installations.

The possible error codes returned in `sbiret.error` are shown in Table 103.

Table 103. Debug Triggers Uninstall Errors

Error code	Description
SBI_SUCCESS	Triggers uninstalled successfully.
SBI_ERR_INVALID_PARAM	One of the debug triggers with index `trig_idx` in the specified set of debug triggers either not mapped to any HW debug trigger OR has `trig_idx` >= `trig_max`.

2. 在 trig_state 中保存 trig_tdata1.vs、trig_tdata1.vu、trig_tdata1.s 和 trig_tdata.u 位的副本。

3. 更新硬件调试触发器的 tdata1、tdata2 和 tdata3 控制状态寄存器。

成功时或共享内存被禁用时，sbiret.value 会被置零；而在其他失败情况下，sbiret.value 会被设置为触发配置失败的数组索引 i。

表 102 展示了 sbiret.error 可能返回的错误代码。

表 102. 调试触发器更新错误代码

错误代码	描述
SBI_SUCCESS	触发器更新成功。
SBI_ERR_NO_SHMEM	调试触发器的共享内存功能未启用。
SBI_ERR_BAD_RANGE	触发计数 >= 触发上限
SBI_ERR_INVALID_PARAM	共享内存中的某个触发器配置存在无效值 trig_idx（即 trig_idx >= trig_max）、trig_tdata1、trig_tdata2 或 trig_tdata3。
SBI_ERR_FAILED	某个触发器配置具有有效的 trig_idx，但对应的调试触发器未映射到任何硬件调试触发器。
SBI_ERR_NOT_SUPPORTED 由于 tdata1、tdata2 或 tdata3 CSR 中存在未实现的可选位，导致无法编程某个触发器配置。	

19.6. 功能：卸载一组触发器（FID #5） struct sbiret sbi_debug_uninstall_triggers(unsigned long trig_idx_base,

```
                                                    unsigned long trig_idx_mask)
```

在调用该功能的硬件线程上，卸载由 trig_idx_base 和 trig_idx_mask 参数指定的一组调试触发器。trig_idx_base 指定起始触发器索引，而 trig_idx_mask 是一个位掩码，表示相对于基址需要卸载的触发器。掩码中的每一位对应一个特定的触发器，允许同时对多个触发器进行批量操作。

对于指定调试触发器集合中的每个调试触发器，SBI 实现必须：

1. 清除映射的硬件调试触发器对应的 tdata1、tdata2 和 tdata3 控制状态寄存器(CSR)。
2. 清除该调试触发器的 trig_state 状态。
3. 解除映射并释放该硬件调试触发器及对应的 trig_idx 索引，以供后续触发器安装时重复使用。

可能返回的错误码如 sbiret.error 所示，详见表 103。

表 103. 调试触发器卸载错误

错误代码	描述
SBI_SUCCESS	触发器卸载成功。
SBI_ERR_INVALID_PARAM	指定调试触发器集合中索引为 trig_idx 的某个触发器要么未映射到任何硬件调试触发器，要么其 trig_idx 值≥trig_max。

19.7. Function: Enable a set of triggers (FID #6)

```
struct sbiret sbi_debug_enable_triggers(unsigned long trig_idx_base,
                                        unsigned long trig_idx_mask)
```

Enable a set of debug triggers specified by the `trig_idx_base` and `trig_idx_mask` parameters on the calling hart. The `trig_idx_base` specifies the starting trigger index, while the `trig_idx_mask` is a bitmask indicating which triggers, relative to the base, are to be enabled. Each bit in the mask corresponds to a specific trigger, allowing for batch operations on multiple triggers simultaneously.

To enable a debug trigger in the specified set of debug triggers, the SBI implementation MUST restore the `vs`, `vu`, `s`, and `u` bits of the mapped HW debug trigger from their saved copy in `trig_state`.

The possible error codes returned in `sbiret.error` are shown in Table 104.

Table 104. Debug Triggers Enable Errors

Error code	Description
SBI_SUCCESS	Triggers enabled successfully.
SBI_ERR_INVALID_PARAM	One of the debug triggers with index `trig_idx` in the specified set of debug triggers either not mapped to any HW debug trigger OR has `trig_idx` >= `trig_max`.

19.8. Function: Disable a set of triggers (FID #7)

```
struct sbiret sbi_debug_disable_triggers(unsigned long trig_idx_base,
                                         unsigned long trig_idx_mask)
```

Disable a set of debug triggers specified by the `trig_idx_base` and `trig_idx_mask` parameters on the calling hart. The `trig_idx_base` specifies the starting trigger index, while the `trig_idx_mask` is a bitmask indicating which triggers, relative to the base, are to be disabled. Each bit in the mask corresponds to a specific trigger, allowing for batch operations on multiple triggers simultaneously.

To disable a debug trigger in the specified set of debug triggers, the SBI implementation MUST clear the `vs`, `vu`, `s`, and `u` bits of the mapped HW debug trigger.

The possible error codes returned in `sbiret.error` are shown in Table 105.

Table 105. Debug Triggers Disable Errors

Error code	Description
SBI_SUCCESS	Triggers disabled successfully.
SBI_ERR_INVALID_PARAM	One of the debug triggers with index `trig_idx` in the specified set of debug triggers either not mapped to any HW debug trigger OR has `trig_idx` >= `trig_max`.

19.9. Function Listing

Table 106. Debug Triggers Function List

19.7. 函数：启用一组触发器（功能号#6）struct sbiret sbi_debug_enable_triggers(unsigned long trig_idx_base,

```
                                                    unsigned long trig_idx_mask)
```

在调用该函数的硬件线程上，启用由 trig_idx_base 和 trig_idx_mask 参数指定的一组调试触发器。
trig_idx_base 指定起始触发器索引，而 trig_idx_mask 是一个位掩码，表示相对于基址需要启用的触发器。掩码中的每一位对应一个特定触发器，允许同时对多个触发器进行批量操作。
要启用指定调试触发器集合中的某个调试触发器，SBI 实现必须恢复

从 trig_state 中保存的副本恢复映射硬件调试触发器的 vs、vu、s 和 u 位。

sbiret.error 可能返回的错误代码如表 104 所示。

表 104. 调试触发器启用错误

错误码	描述
SBI_SUCCESS	触发器启用成功。
SBI_ERR_INVALID_PARAM	指定调试触发器集合中索引为 trig_idx 的某个触发器要么未映射到任何硬件调试触发器，要么其 trig_idx >= trig_max。

19.8. 功能：禁用一组触发器（FID #7） struct sbiret sbi_debug_disable_triggers(unsigned long trig_idx_base,

```
                                                    unsigned long trig_idx_mask)
```

禁用调用硬件线程上由 trig_idx_base 和 trig_idx_mask 参数指定的一组调试触发器。trig_idx_base 指定起始触发器索引，而 trig_idx_mask 是相对于基址的位掩码，指示需要禁用的触发器。掩码中的每个比特位对应一个特定触发器，允许同时对多个触发器进行批量操作。
要禁用指定调试触发器集合中的某个触发器，SBI 实现必须清除 vs 位，

映射硬件调试触发器的 vu、s 和 u 位。

sbiret.error 中可能返回的错误代码如表 105 所示。

表 105. 调试触发器禁用错误

错误码	描述
SBI_SUCCESS	触发器已成功禁用。
SBI_ERR_INVALID_PARAM	指定调试触发器集合中索引为 trig_idx 的某个触发器要么未映射到任何硬件调试触发器，要么其 trig_idx >= trig_max。

19.9. 函数列表

表 106. 调试触发器函数列表

Function Name	SBI Version	FID	EID
sbi_debug_num_triggers	3.0	0	0x44425452
sbi_debug_set_shmem	3.0	1	0x44425452
sbi_debug_read_triggers	3.0	2	0x44425452
sbi_debug_install_triggers	3.0	3	0x44425452
sbi_debug_update_triggers	3.0	4	0x44425452
sbi_debug_uninstall_triggers	3.0	5	0x44425452
sbi_debug_enable_triggers	3.0	6	0x44425452
sbi_debug_disable_triggers	3.0	7	0x44425452

功能名称	SBI 版本 FID EID		
sbi_debug_num_triggers	3.0	0 0x44425452	
sbi_debug_set_shmem	3.0	1	0x44425452
sbi_debug_read_triggers	3.0	2	0x44425452
sbi_debug_install_triggers	3.0	3	0x44425452
sbi_debug_update_triggers	3.0	4	0x44425452
sbi_debug_uninstall_triggers	3.0	5	0x44425452
sbi_debug_enable_triggers（启用调试触发器）	3.0	6 0x44425452	
sbi_debug_disable_triggers（禁用调试触发器）	3.0	7	0x44425452

Chapter 20. Message Proxy Extension (EID #0x4D505859 "MPXY")

The Message Proxy (MPXY) extension allows the supervisor software to send and receive messages through the SBI implementation. This extension defines a generic interface that allows the supervisor software to implement clients for various messaging protocols implemented by the SBI implementation (such as RPMI [3], etc). The SBI MPXY is an abstract interface and agnostic of message protocol implementations in the SBI implementation. The message format used by a client in the supervisor software to send/receive messages through the SBI MPXY extension is defined by the corresponding message protocol specification.

This extension requires a per-hart shared memory between the supervisor software and the SBI implementation for message data transfer. This per-hart shared memory is different from the message protocol specific shared memory that is used between the SBI implementation and the remote entity that implements the message protocol. The remote entity can be implemented as a system-level partition on the same hart or as firmware running on a platform microcontroller or emulated by an SBI implementation. The supervisor software MUST call the `sbi_mpxy_set_shmem` function to set up the shared memory before calling any other function defined in the extension.

20.1. SBI MPXY and Dedicated SBI extension rule

The implementation may only provide either an SBI MPXY or a dedicated SBI extension interface for a specific functionality within the specified message protocol, but never both.

20.2. Message Channels

The MPXY extension defines an abstract message channel which is identified by a unique 32 bits unsigned integer referred to as `channel_id`. The supervisor software can discover the `channel_id` of a message channel using standard hardware discovery mechanisms. The message protocol specification associated with a message channel is discovered through the standard message channel attributes defined in the following sections.

The type of message data, or the group of messages, that may be transmitted over an MPXY message channel is defined by the message protocol specification. The message protocol specification may define multiple message groups, but may allow only a selected set of messages accessible to the supervisor software via the MPXY extension.

Any `channel_id` exported to the supervisor software via the hardware discovery mechanism is implicitly associated with a particular message protocol transport. This binding is internal to the SBI implementation. To the supervisor software, a message channel is an abstract entity with associated attributes that can be accessed through the MPXY extension. The message channel attributes describe the characteristics of a message channel depending on the associated message protocol.

20.3. Message Channel Attributes

Each message channel (`channel_id`) has a set of associated attributes which are identified by a unique 32 bits unsigned integer called `attribute_id` where each attribute value is 32 bits wide.

The message channel attributes are divided into two categories: standard attributes and message protocol specific attributes. The encoding of message channel `attribute_id` is as follows:

第 20 章 消息代理扩展（EID #0x4D505859 "MPXY"）

消息代理（MPXY）扩展允许监督模式软件通过 SBI 实现发送和接收消息。该扩展定义了一个通用接口，使监督模式软件能够为 SBI 实现的各种消息协议（如 RPMI[3]等）开发客户端。SBI MPXY 是一个抽象接口，与 SBI 实现中的具体消息协议无关。监督模式软件中的客户端通过 SBI MPXY 扩展发送/接收消息时使用的消息格式，由相应的消息协议规范定义。

该扩展要求在监管软件与 SBI 实现之间为每个硬件线程(hart)建立共享内存，用于消息数据传输。这种每 hart 共享内存不同于 SBI 实现与实现消息协议的远程实体之间使用的协议专用共享内存。远程实体可以实现在同一 hart 上的系统级分区、平台微控制器上运行的固件，或由 SBI 实现模拟。监管软件在调用本扩展定义的其他任何函数之前，必须调用 sbi_mpxy_set_shmem 函数来设置共享内存。

20.1. SBI MPXY 与专用 SBI 扩展规则

对于指定消息协议中的特定功能，实现只能提供 SBI MPXY 或专用 SBI 扩展接口中的一种，绝不能同时提供两者。

20.2. 消息通道

MPXY 扩展定义了一个抽象消息通道，该通道通过一个称为 channel_id 的唯一 32 位无符号整数进行标识。监督模式软件可通过标准硬件发现机制来获取消息通道的 channel_id。与消息通道相关联的消息协议规范则通过后续章节定义的标准消息通道属性进行发现。

消息数据在 MPXY 消息通道上传输的类型或消息分组由消息协议规范定义。消息协议规范可定义多个消息组，但可能仅允许监督模式软件通过 MPXY 扩展访问特定的消息集合。

通过硬件发现机制导出给监督者软件的任意 channel_id 都隐式关联着特定的消息协议传输。这种绑定关系是 SBI 实现内部的。对监督者软件而言，消息通道是具有关联属性的抽象实体，可通过 MPXY 扩展进行访问。消息通道属性根据关联的消息协议描述通道特性。

20.3. 消息通道属性

每个消息通道(channel_id)都关联着一组属性，这些属性由称为 attribute_id 的 32 位无符号整数唯一标识，每个属性值宽度为 32 位。

消息通道属性分为两类：标准属性和消息协议特定属性。消息通道 attribute_id 的编码规则如下：

```
attribute_id[31] = 0 (Standard)
attribute_id[31] = 1 (Message protocol)
```

Standard attributes are defined by the MPXY extension and all message channels MUST support these attributes. Apart from standard attributes, a message channel may also have message protocol attributes which are defined by the message protocol specification.

Once supervisor software has verified the channel and its associated attributes, it can use the MPXY interface to send messages over the message channel where each message is identified by a 32 bit unsigned integer called `message_id`. The set of `message_id` that can be sent over an MPXY channel are defined by the message protocol specification.

Table 107. MPXY Channel Attributes

Attribute Name	Attribute ID	Access	Description
MSG_PROT_ID	0x00000000	RO	**Message Protocol Identifier** Unique ID for identifying the message protocol specification. The table Table 108 provides a list of supported message protocol specifications and their IDs.

MSG_PROT_VERSION	0x00000001	RO	**Message Protocol Version**
			Version of the message protocol specification.
			`[31:16]: Major version.`
			`[15:0]: Minor version.`
			If the message protocol specification has additional version fields or if the above encoding is not suitable, the message protocol specification may define message protocol specific attribute for discovering the version of the message protocol specification.

MSG_DATA_MAX_LEN	0x00000002	RO	**Maximum Message Data Length** Maximum message data size in bytes supported by the message channel to send or receive message.
MSG_SEND_TIMEOUT	0x00000003	RO	**Message Send Timeout** Timeout for sending a message in microseconds as supported by the message protocol specification. Functions which do not wait for response can use this timeout value.
MSG_COMPLETION_TIMEOUT	0x00000004	RO	**Message Completion Timeout** This is the aggregate of `MSG_SEND_TIMEOUT` and the response receive timeout in microseconds as supported by the message protocol specification. Functions which wait for response can use this timeout value.

```
属性 ID[31] = 0（标准） 属性 ID[31] = 1（消息协议）
```

标准属性由 MPXY 扩展定义，所有消息通道都必须支持这些属性。除标准属性外，消息通道还可能包含由消息协议规范定义的消息协议属性。

当监控模式软件验证完通道及其关联属性后，即可通过 MPXY 接口在消息通道上发送消息。每条消息由一个称为 message_id 的 32 位无符号整数标识，可通过 MPXY 通道发送的 message_id 集合由消息协议规范定义。

表 107. MPXY 通道属性

属性名称	属性 ID 访问权限 描述		
消息协议标识符	0x00000000 只读		消息协议标识符 用于标识消息协议规范的唯一标识符。表 108 提供了支持的消息协议规范列表及其对应 ID。
MSG_PROT_VERSION	0x00000001 只读		消息协议版本 消息协议规范的版本号。 [31:16]：主版本号。 [15:0]：次版本号。 若消息协议规范包含额外版本字段或上述编码方式不适用，该规范可定义特定属性用于发现协议版本信息。
MSG_DATA_MAX_LEN	0x00000002 RO		最大消息数据长度 消息通道支持发送或接收消息的最大数据字节数。
MSG_SEND_TIMEOUT	0x00000003 RO		消息发送超时 消息协议规范支持的以微秒为单位的消息发送超时值。无需等待响应的函数可使用此超时值。
消息完成超时	0x00000004 RO		消息完成超时 该值为消息协议规范支持的 MSG_SEND_TIMEOUT 与响应接收超时（以微秒为单位）的聚合值。等待响应的函数可使用此超时值。

Attribute Name	Attribute ID	Access	Description

CHANNEL_CAPABILITY	0x00000005	RO	**Channel Capabilities Bits**
			
			`[31:6]: Reserved and 0`
			
			`[5]: Get Notifications (FID #7) Support`
			
			`[4]: Send Message without Response (FID #6) Support`
			
			`[3]: Send Message with Response (FID #5) Support`
			
			`[2]: Events State Support`
			
			`[1]: SSE Event`
			
			`[0]: MSI`
			
			Any defined bit as `1` means the corresponding capability is supported.
			
			The SBI implementation only needs to support one method to indicate the availability of notifications, either MSI or SSE. If both are enabled, the MSI is preferred over the SSE event.
			
			If Get Notifications function (FID #7) is not supported then the Events State Support, SSE Event and MSI bits must be `0`.

SSE_EVENT_ID	0x00000006	RO	**SSE Event ID** Channel SSE event ID if the SSE is supported as discovered via `CHANNEL_CAPABILITY` attribute. If the SSE is not supported then this value is unspecified.

MSI_CONTROL	0x00000007	RW	**MSI Control**
			Control for MSI based indication.
			
			`0 = Disable`
			`1 = Enable`
			
			This attribute can be set to `1` if `MSI_ADDR_LOW` and `MSI_ADDR_HIGH` attributes point to a valid MSI target.
			
			If the message channel does not support MSI based indication as discovered via the `CHANNEL_CAPABILITY` attribute, then `MSI_CONTROL` ignore writes and always reads zero.
			
			The reset value of this attribute is `0`.

属性名称	属性 ID 访问权限 描述		
通道能力	0x00000005 只读		通道能力位 [31:6]: 保留位且置`0` [5]: 获取通知（功能 ID #7） 支持 [4]: 无等待发送消息 响应（功能 ID #6）支持 [3]：发送消息附带 响应（功能 ID #5）支持 [2]：事件状态支持 [1]: SSE 事件 [0]: MSI 任何定义为 1 的比特位表示支持相应功能 SBI 实现只需支持一种通知可用性的方法（MSI 或 SSE 事件）。若两者同时启用，则优先采用 MSI 而非 SSE 事件。 如果不支持获取通知功能（功能 ID #7），则事件状态支持、SSE 事件和 MSI 位必须为 0。
SSE_EVENT_ID	0x00000006 RO		SSE 事件 ID 如果通过 CHANNEL_CAPABILITY 属性发现支持 SSE，则此处为通道 SSE 事件 ID。若不支持 SSE，则该值未定义。
MSI 控制寄存器	0x00000007 可读写		MSI 控制位 基于 MSI 的中断控制配置 0 = 禁用 1 = 启用 当 MSI_ADDR_LOW 和 MSI_ADDR_HIGH 属性指向有效的 MSI 目标时，可将此属性设置为 1 如果通过 CHANNEL_CAPABILITY 属性发现消息通道不支持基于 MSI 的指示，则 MSI_CONTROL 寄存器将忽略写入操作且始终读取为零。 该属性的复位值为 0。

Attribute Name	Attribute ID	Access	Description
MSI_ADDR_LOW	0x00000008	RW	**MSI Address Low** Low 32 bits of the MSI target physical address. If the message channel does not support MSI based indication then this attribute ignores writes and always reads `0`. The reset value of this attribute is `0`.
MSI_ADDR_HIGH	0x00000009	RW	**MSI Address High** High 32 bits of the MSI target physical address. If the message channel does not support MSI based indication then this attribute ignores writes and always reads `0`. The reset value of this attribute is `0`.
MSI_DATA	0x0000000A	RW	**MSI Data** MSI data word written to the MSI target. If the message channel does not support MSI based indication then this attribute ignores writes and always reads `0`. The reset value of this attribute is `0`.
EVENTS_STATE_CONTROL	0x0000000B	RW	**Events State Control.** If the message channel supports notification events state data then this attribute can be used to enable state reporting like number of events `RETURNED`, `REMAINING` or `LOST` after a call to Get Notifications (FID #7) function. The reset value of this attribute is `0`, which means disabled. If supervisor software wants to enable events state reporting, it MUST write `1`. If the events state reporting is not supported by the channel or the Get Notifications (FID #7) function is not implemented as indicated by the `CHANNEL_CAPABILITY` attribute, then the writes to this attribute will be ignored. More details on events state data are mentioned in the function Get Notifications (FID #7) description.
RESERVED	0x0000000C - 0x7fffffff		Reserved for future use.
Message Protocol Attributes	0x80000000 - 0xffffffff		Attributes defined by the message protocol specification. Refer to message protocol specification for details.

20.4. Message Protocol IDs

Each message protocol specification supporting MPXY extension will be assigned a 32 bits identifier which is listed in the table below. New message protocol enabling support for MPXY will need to be added in the below table with its assigned ID.

属性名称	属性 ID 访问权限 描述		
MSI_ADDR_LOW	0x00000008 可读写		MSI 地址低 32 位 MSI 目标物理地址的低 32 位。 若消息通道不支持基于 MSI 的指示方式，则该属性会忽略写入操作并始终返回 0。 该属性的复位值为 0。
MSI_ADDR_HIGH	0x00000009 可读写		MSI 地址高位 MSI 目标物理地址的高 32 位。 若消息通道不支持基于 MSI 的指示，则该属性会忽略写入操作并始终读取 0 值。 该属性的复位值为 0。
MSI_DATA	0x0000000A RW		MSI 数据 写入 MSI 目标设备的 MSI 数据字。 若消息通道不支持基于 MSI 的指示方式，则该属性会忽略写入操作并始终返回 0 值。 该属性的复位值为 0。
EVENTS_STATE_CONTROL 0x0000000B RW			事件状态控制 若消息通道支持通知事件状态数据，则可通过该属性启用状态报告功能，例如在调用获取通知(FID #7)函数后，返回事件数量、剩余事件数量或丢失事件数量等状态信息。 该属性默认值为 0，表示禁用状态报告功能。若监管者软件需启用事件状态报告，必须写入 1。若通道不支持事件状态报告功能，或如 CHANNEL_CAPABILITY 属性所示未实现获取通知(FID #7)函数，则对该属性的写入操作将被忽略。 关于事件状态数据的更多细节，请参阅获取通知(FID #7)函数的描述部分。
保留字段	0x0000000C - 0x7fffffff		预留未来使用
消息协议属性 0x80000000	- 0xffffffff		消息协议规范定义的属性。详情请参阅消息协议规范。

20.4. 消息协议标识符

每个支持 MPXY 扩展的消息协议规范将被分配一个 32 位标识符，如下表所示。新增的支持 MPXY 的消息协议需要在下表中添加其分配的 ID。

Table 108. MPXY Message Protocol IDs

Message Protocol Name	MSG_PROT_ID value	Description
RPMI	0x00000000	RPMI [3]
RESERVED	0x00000001 - 0x7fffffff	
Vendor Specific	0x80000000 - 0xffffffff	Custom vendor specific message protocol

20.5. Function: Get shared memory size (FID #0)

```
struct sbiret sbi_mpxy_get_shmem_size(void)
```

Get the shared memory size in number of bytes for sending and receiving messages.

The shared memory size returned by the SBI implementation MUST satisfy the following requirements:

1. The shared memory size MUST be same for all HARTs
2. The shared memory size MUST be at least 4096 bytes
3. The shared memory size MUST be multiple of 4096 bytes
4. The shared memory size MUST not be less than the biggest MSG_DATA_MAX_LEN attribute value across all MPXY channels

This function always returns SBI_SUCCESS in `sbiret.error` and it will return the shared memory size in `sbiret.uvalue`.

20.6. Function: Set shared memory (FID #1)

```
struct sbiret sbi_mpxy_set_shmem(unsigned long shmem_phys_lo,
                                 unsigned long shmem_phys_hi,
                                 unsigned long flags)
```

Set the shared memory for sending and receiving messages on the calling hart.

If both `shmem_phys_lo` and `shmem_phys_hi` parameters are not all-ones bit-wise then the `shmem_phys_lo` parameter specifies the lower XLEN bits and the `shmem_phys_hi` parameter specifies the upper XLEN bits of the shared memory physical base address. The `shmem_phys_lo` parameter MUST be 4096 bytes aligned and the shared memory size is assumed to be same as returned by the Get shared memory size function (FID #0).

If both `shmem_phys_lo` and `shmem_phys_hi` parameters are all-ones bit-wise then shared memory is disabled.

The `flags` parameter specifies configuration for shared memory setup and it is encoded as follows:

```
flags[XLEN-1:2]: Reserved for future use and must be zero.
flags[1:0]: Shared memory setup mode (Refer table below).
```

表 108. MPXY 消息协议 ID

消息协议名称 MSG_PROT_ID 值		描述
RPMI	0x00000000	RPMI [3]
保留字段	0x00000001 - 0x7fffffff	
厂商自定义	0x80000000 - 0xffffffff 自定义厂商特定消息协议	

20.5. 功能：获取共享内存大小（功能号#0） struct sbiret sbi_mpxy_get_shmem_size(void)

获取用于发送和接收消息的共享内存大小（以字节为单位）。

SBI 实现返回的共享内存大小必须满足以下要求：

1. 所有 HART 的共享内存大小必须相同
2. 共享内存大小必须至少为 4096 字节
3. 共享内存大小必须是 4096 字节的整数倍
4. 共享内存大小不得小于所有 MPXY 通道中 MSG_DATA_MAX_LEN 属性的最大值

该函数在 sbiret.error 中始终返回 SBI_SUCCESS，并将返回共享内存的
sbiret.uvalue.

20.6. 功能：设置共享内存（功能号 #1） struct sbiret sbi_mpxy_set_shmem(unsigned long shmem_phys_lo,

```
                                    unsigned long shmem_phys_hi, unsigned
                                    long flags)
```

为当前硬件线程设置用于收发消息的共享内存区域。

若 shmem_phys_lo 和 shmem_phys_hi 参数并非全 1 位模式，则 shmem_phys_lo 参数指定共享内存物理基地址的低 XLEN 位，shmem_phys_hi 参数指定其高 XLEN 位。shmem_phys_lo 参数必须按 4096 字节对齐，且假定共享内存大小与"获取共享内存大小功能（FID #0）"返回的值相同。
若 shmem_phys_lo 和 shmem_phys_hi 参数均为全 1 位模式，则共享内存功能被禁用。

flags 参数用于指定共享内存的配置设置，其编码方式如下：

```
flags[XLEN-1:2]：保留未来使用，必须置零。
标志位[1:0]：共享内存设置模式（参见下表）。
```

Table 109. MPXY Shared Memory Setup Mode

Mode	flags[1:0]	Description
OVERWRITE	0b00	Ignore the current shared memory state and force setup the new shared memory based on the passed parameters.
OVERWRITE-RETURN	0b01	Same as `OVERWRITE` mode and additionally after the new shared memory state is enabled, the old shared memory `shmem_phys_lo` and `shmem_phys_hi` are written in the same order to the new shared memory at offset `0x0`. This flag provide provision to software layers in the supervisor software that want to send messages using the shared memory but do not know the shared memory details that has already been setup. Those software layers can temporarily setup their own shared memory on the calling hart, send messages and then restore back the previous shared memory with the SBI implementation.
RESERVED	0b10 - 0b11	Reserved for future use. Must be initialized to `0`.

The supervisor software may consist of several software layers, including an operating system and runtime firmware, which are mutually exclusive and without any provision for data exchange. Typically, a call is required to invoke the runtime firmware when required by the operating system, and once the runtime firmware has finished the task it returns control to the operating system.
The operating system may setup the shared memory per-hart using the `OVERWRITE` *flag during boot. The runtime firmware may also need to use the MPXY channel to send the message data when its invoked. In such a scenario the runtime firmware can setup its own MPXY channel shared memory on the called hart using the* `OVERWRITE-RETURN` *flag and when finished, can restore the previous shared memory before returning control to the operating system.*

The possible error codes returned in `sbiret.error` are below.

Table 110. MPXY Set Shared Memory Errors

Error code	Description
SBI_SUCCESS	Shared memory was set or cleared successfully.
SBI_ERR_INVALID_PARAM	The `flags` parameter has invalid value or the bits set are within the reserved range. Or the `shmem_phys_lo` parameter is not 4096 bytes aligned.
SBI_ERR_INVALID_ADDRESS	The shared memory pointed to by the `shmem_phys_lo` and `shmem_phys_hi` parameters does not satisfy the requirements described in Section 3.2.
SBI_ERR_FAILED	Failed due to other unspecified errors.

The supervisor software MUST call this function to setup the shared memory first before calling any other function in this extension.

表 109. MPXY 共享内存设置模式

模式	标志位[1:0]	描述
覆盖	0b00	忽略当前共享内存状态，强制根据传入参数设置新的共享内存。
覆盖返回模式 0b01		与覆盖模式相同，且在新共享内存状态启用后，旧共享内存的 shmem_phys_lo 和 shmem_phys_hi 会按相同顺序写入新共享内存的 0x0 偏移处。 该标志为监管者软件中希望使用共享内存发送消息、但不知晓已设置共享内存详情的软件层提供了支持。这些软件层可在调用硬件线程上临时设置自己的共享内存，发送消息后再通过 SBI 实现恢复之前的共享内存。
保留	0b10 - 0b11	预留未来使用。必须初始化为 0。

监督者软件可能包含多个相互独立且无数据交换机制的软件层，例如操作系统和运行时固件。通常当操作系统需要时，会通过调用指令激活运行时固件；待固件完成任务后，再将控制权交还给操作系统。

操作系统可以在启动时使用 OVERWRITE 标志为每个硬件线程设置共享内存。运行时固件在被调用时也可能需要使用 MPXY 通道发送消息数据。在这种情况下，运行时固件可以使用 OVERWRITE-RETURN 标志在被调用的硬件线程上设置自己的 MPXY 通道共享内存，并在完成后恢复先前的共享内存，再将控制权交还给操作系统。

sbiret.error 可能返回的错误代码如下。

表 110. MPXY 设置共享内存错误码

错误代码	描述
SBI_SUCCESS	共享内存设置或清除成功。
SBI_ERR_INVALID_PARAM	flags 参数值无效，或设置的位处于保留范围内。或者 shmem_phys_lo 参数未按 4096 字节对齐。
SBI_ERR_INVALID_ADDRESS 由 shmem_phys_lo 和 shmem_phys_hi 参数指向的共享内存不符合第 3.2 节所述要求。	
SBI_ERR_FAILED	由于其他未指定的错误导致失败。

监控软件在调用本扩展中的任何其他函数之前，必须先调用此函数来设置共享内存。

20.7. Function: Get Channel IDs (FID #2)

```
struct sbiret sbi_mpxy_get_channel_ids(uint32_t start_index)
```

Get channel IDs of the message channels accessible to the supervisor software in the shared memory of the calling hart. The channel IDs are returned as an array of 32 bits unsigned integers where the `start_index` parameter specifies the array index of the first channel ID to be returned in the shared memory.

The SBI implementation will return channel IDs in the shared memory of the calling hart as specified by the table below:

Table 111. MPXY Channel IDs Shared Memory Layout

Offset	Field	Description
0x0	REMAINING	Remaining number of channel IDs.
0x4	RETURNED	Number of channel IDs (N) returned in the shared memory.
0x8	CHANNEL_ID [start_index + 0]	Channel ID
0xC	CHANNEL_ID [start_index + 1]	Channel ID
0x8 + ((N-1) * 4)	CHANNEL_ID [start_index + N - 1]	Channel ID

The number of channel IDs returned in the shared memory are specified by the `RETURNED` field whereas the `REMAINING` field specifies the number of remaining channel IDs. If the `REMAINING` is not `0` then supervisor software can call this function again to get remaining channel IDs with `start_index` passed accordingly. The supervisor software may require multiple SBI calls to get the complete list of channel IDs depending on the `RETURNED` and `REMAINING` fields.

The `sbiret.uvalue` is always set to zero whereas the possible error codes returned in `sbiret.error` are below.

Table 112. MPXY Get Channel IDs Errors

Error code	Description
SBI_SUCCESS	The channel ID array has been written successfully.
SBI_ERR_INVALID_PARAM	`start_index` is invalid.
SBI_ERR_NO_SHMEM	The shared memory setup is not done or disabled for the calling hart.
SBI_ERR_DENIED	Getting channel ID array is not allowed on the calling hart.
SBI_ERR_FAILED	Failed due to other unspecified errors.

20.8. Function: Read Channel Attributes (FID #3)

```
struct sbiret sbi_mpxy_read_attributes(uint32_t channel_id,
                                       uint32_t base_attribute_id,
                                       uint32_t attribute_count)
```

Read message channel attributes. The `channel_id` parameter specifies the message channel whereas `base_attribute_id` and `attribute_count` parameters specify the range of attribute ids to be read.

20.7. 功能：获取通道 ID（功能号#2）
结构体 sbiret sbi_mpxy_get_channel_ids(无符号 32 位整型 起始索引)

获取调用硬件线程共享内存中可供监管者软件访问的消息通道 ID。通道 ID 以 32 位无符号整数数组形式返回，其中 start_index 参数指定共享内存中首个待返回通道 ID 的数组索引。
SBI 实现将按照下表规定，在调用硬件线程的共享内存中返回通道 ID：

表 111. MPXY 通道 ID 共享内存布局

偏移量	字段	描述
0x0	剩余	剩余通道 ID 数量
0x4	返回	共享内存中返回的通道 ID 数量(N)。
0x8	通道 ID [起始索引 + 0] 通道标识符	
0xC	通道 ID [起始索引 + 1]	通道标识符
0x8 + ((N-1) * 4)	通道 ID [起始索引 + N - 1] 通道标识符	

共享内存中返回的通道 ID 数量由 RETURNED 字段指定，而
REMAINING 字段则指定剩余的通道 ID 数量。若 REMAINING 不为 0，监督模式软件可再次调用此函数获取剩余通道 ID，此时需相应调整 start_index 参数。根据实际情况，监督模式软件可能需要多次 SBI 调用来获取完整的通道 ID 列表。
返回值与剩余值字段

sbiret.uvalue 始终设为零，而 sbiret.error 可能返回的错误代码如下

表 112. MPXY 获取通道 ID 错误码

错误代码	描述
SBI_SUCCESS	通道 ID 数组已成功写入
SBI_ERR_INVALID_PARAM	起始索引无效。
SBI_ERR_NO_SHMEM	共享内存设置未完成或当前调用硬件线程已禁用该功能。
SBI_ERR_DENIED（访问被拒绝）	当前硬件线程不允许获取通道 ID 数组。
SBI_ERR_FAILED	因其他未指定错误导致失败。

20.8. 功能：读取通道属性（功能号 #3） struct sbiret sbi_mpxy_read_attributes(uint32_t channel_id,

```
                                                uint32_t base_attribute_id, uint32_t
attribute_count)
```

读取消息通道属性。channel_id 参数指定消息通道，而
base_attribute_id 和 attribute_count 参数则指定要读取的属性 ID 范围。

Supervisor software MUST call this function for the contiguous attribute range where the `base_attribute_id` is the starting index of that range and `attribute_count` is the number of attributes in the contiguous range. If there are multiple such attribute ranges then multiple calls of this function may be done from supervisor software. Supervisor software MUST read the message protocol specific attributes via separate call to this function with `base_attribute_id` and `attribute_count` without any overlap with the MPXY standard attributes.

Upon calling this function the message channel attribute values are returned starting from the offset `0x0` in the shared memory of the calling hart where the value of the attribute with `attribute_id = base_attribute_id + i` is available at the shared memory offset `4 * i`.

The possible error codes returned in `sbiret.error` are shown below.

Table 113. MPXY Read Channel Attributes Errors

Error code	Description
SBI_SUCCESS	Message channel attributes has been read successfully.
SBI_ERR_INVALID_PARAM	`attribute_count` is `0`. Or the `attribute_count > (shared memory size)/4`. Or the `base_attribute_id` is not valid.
SBI_ERR_NOT_SUPPORTED	`channel_id` is not supported or invalid.
SBI_ERR_BAD_RANGE	One of the attributes in the range specified by the `base_attribute_id` and `attribute_count` do not exist.
SBI_ERR_NO_SHMEM	The shared memory setup is not done or disabled for calling hart.
SBI_ERR_FAILED	Failed due to other unspecified errors.

20.9. Function: Write Channel Attributes (FID #4)

```
struct sbiret sbi_mpxy_write_attributes(uint32_t channel_id,
                                        uint32_t base_attribute_id,
                                        uint32_t attribute_count)
```

Write message channel attributes. The `channel_id` parameter specifies the message channel whereas `base_attribute_id` and `attribute_count` parameters specify the range of attribute ids.

Supervisor software MUST call this function for the contiguous attribute range where the `base_attribute_id` is the starting index of that range and `attribute_count` is the number of attributes in the contiguous range. If there are multiple such attribute ranges then multiple calls of this function may be done from supervisor software. Apart from contiguous attribute indices, supervisor software MUST also consider the attribute access permissions and attributes with RO (Read Only) access MUST be excluded from the attribute range. Supervisor software MUST write the message protocol specific attributes via separate call to this function with `base_attribute_id` and `attribute_count` without any overlap with the MPXY standard attributes.

Before calling this function, the supervisor software must populate the shared memory of the calling hart starting from offset 0x0 with the message channel attribute values. For each attribute with attribute_id = base_attribute_id + i, the corresponding value MUST be placed at the shared memory offset 4 * i.

The possible error codes returned in `sbiret.error` are shown below.

Table 114. MPXY Write Channel Attributes Errors

监管软件必须为连续属性范围调用此函数，其中
base_attribute_id 是该范围的起始索引，attribute_count 是连续范围内的属性数量。若存在多个此类属性范围，监管软件可多次调用此函数。监管软件必须通过单独调用此函数（指定不重叠于 MPXY 标准属性的 base_attribute_id 和 attribute_count 参数）来读取消息协议特定属性。
调用此函数时，消息通道属性值将从调用硬件线程共享内存的 0x0 偏移处开始返回，其中属性 ID 为

base_attribute_id + i 的属性值将存储在共享内存偏移量 4 * i 处。

可能的错误码通过 sbiret.error 返回如下所示。

表 113. MPXY 读取通道属性错误

错误码	描述
SBI_SUCCESS	消息通道属性已成功读取。
SBI_ERR_INVALID_PARAM（无效参数错误）	属性计数为 0。 或属性计数 > (共享内存大小)/4。或基础属性 ID 无效。
SBI_ERR_NOT_SUPPORTED	channel_id 不被支持或无效。
SBI_ERR_BAD_RANGE	由 base_attribute_id 指定的范围中的某个属性 属性计数 不存在。
SBI_ERR_NO_SHMEM	共享内存设置未完成或对调用核心禁用。
SBI_ERR_FAILED	因其他未指定错误导致失败。

20.9. 功能：写入通道属性（功能号 #4） struct sbiret sbi_mpxy_write_attributes(uint32_t channel_id,

```
                                                uint32_t base_attribute_id, uint32_t
                                                attribute_count)
```

写入消息通道属性。channel_id 参数指定消息通道，而
base_attribute_id 和 attribute_count 参数指定属性 ID 的范围。
监控程序软件必须为连续的属性范围调用此函数，其中
base_attribute_id 是该范围的起始索引，attribute_count 是连续范围内的属性数量。如果存在多个这样的属性范围，则监控程序软件可以多次调用此函数。除了连续的属性索引外，监控程序软件还必须考虑属性访问权限，并且必须将具有 RO（只读）访问权限的属性排除在属性范围之外。监控程序软件必须通过对此函数的单独调用写入消息协议特定的属性，其中 base_attribute_id 和 attribute_count 不得与 MPXY 标准属性有任何重叠。
在调用此函数前，监督模式软件必须从共享内存偏移量 0x0 开始，为调用硬件的共享内存填充消息通道属性值。对于每个属性标识符为 base_attribute_id + i 的属性，其对应值必须放置在共享内存偏移量 4 * i 处。

sbiret.error 可能返回的错误代码如下所示。

表 114. MPXY 写入通道属性错误码

Error code	Description
SBI_SUCCESS	Message channel attributes has been written successfully.
SBI_ERR_INVALID_PARAM	`attribute_count` is `0`. Or the `attribute_count > (shared memory size)/4`. Or the `base_attribute_id` is not valid.
SBI_ERR_NOT_SUPPORTED	`channel_id` is not supported or invalid.
SBI_ERR_BAD_RANGE	One of the attributes in the range specified by the `base_attribute_id` and `attribute_count` do not exist or the attribute is read-only (RO). Or `base_attribute_id` and `attribute_count` result into a range which overlaps with standard and message protocol specific attributes.
SBI_ERR_NO_SHMEM	The shared memory setup is not done or disabled for calling hart.
SBI_ERR_DENIED	If any attribute write dependency is not satisfied.
SBI_ERR_FAILED	Failed due to other unspecified errors.

20.10. Function: Send Message with Response (FID #5)

```
struct sbiret
sbi_mpxy_send_message_with_response(uint32_t channel_id,
                                    uint32_t message_id,
                                    unsigned long message_data_len)
```

Send a message to the MPXY channel specified by the `channel_id` parameter and wait until a message response is received from the MPXY channel. The `message_id` parameter specifies the message protocol specific identification of the message to be sent whereas the `message_data_len` parameter represents the length of message data in bytes which is located at the offset `0x0` in the shared memory setup by the calling hart.

This function only succeeds upon receipt of a message response from the MPXY channel. In cases where complete data transfer requires multiple transmissions, the supervisor software shall send multiple messages as necessary. Details of such cases can be found in respective message protocol specifications.

Upon calling this function the SBI implementation MUST write the response message data at the offset `0x0` in the shared memory setup by the calling hart and the number of bytes written will be returned through `sbiret.uvalue`. The layout of data in case of both request and response is according to the respective message protocol specification message format.

Upon success, this function:
1) Writes the message response data at offset `0x0` of the shared memory setup by the calling hart.
2) Returns `SBI_SUCCESS` in `sbiret.error`.
3) Returns message response data length in `sbiret.uvalue`.

This function is optional. If this function is implemented, the corresponding bit in the `CHANNEL_CAPABILITY` attribute is set to `1`.

The possible error codes returned in `sbiret.error` are below.

Table 115. MPXY Send Message with Response Errors

Error code	Description
SBI_SUCCESS	Message sent and response received successfully.

错误码	描述
SBI_SUCCESS	消息通道属性已成功写入。
SBI_ERR_INVALID_PARAM	属性计数为 0。 或属性计数 > (共享内存大小)/4。 或基础属性 ID 无效。
SBI_ERR_NOT_SUPPORTED	channel_id 不被支持或无效。
SBI_ERR_BAD_RANGE	由 base_attribute_id 和 attribute_count 指定的属性范围内，存在不存在的属性或该属性为只读（RO）。 或者 base_attribute_id 和 attribute_count 形成的范围与标准和消息协议特定属性存在重叠。
SBI_ERR_NO_SHMEM	共享内存设置未完成或对调用核心禁用。
SBI_ERR_DENIED	若任何属性写入依赖关系未满足。
SBI_ERR_FAILED	由于其他未指定错误导致失败。

20.10. 功能：发送带响应消息（功能号 #5）结构体 sbiret

```
sbi_mpxy_send_message_with_response(uint32_t channel_id,
                                                uint32_t message_id, unsigned long
                                                message_data_len)
```

向 channel_id 参数指定的 MPXY 通道发送消息，并等待直至收到来自该 MPXY 通道的响应消息。message_id 参数指定待发送消息在消息协议中的特定标识，而 message_data_len 参数表示位于调用 hart 所设共享内存 0x0 偏移处的消息数据字节长度。

该函数仅在收到 MPXY 通道的响应消息时执行成功。若需多次传输才能完成数据传递，监督模式软件应按需发送多条消息。此类情形的具体细节详见各消息协议规范。

调用此函数时，SBI 实现必须将响应消息数据写入调用 hart 所设共享内存的 0x0 偏移处，并通过 sbiret.uvalue 返回写入的字节数。

请求与响应数据的布局均遵循相应消息协议规范中规定的消息格式。

调用成功时，此函数将：1）在调用 hart 设置的共享内存偏移量 0x0 处写入消息响应数据。

2）在 sbiret.error 中返回 SBI_SUCCESS 状态码。

3）在 sbiret.uvalue 中返回消息响应数据的长度。

此函数为可选实现。若实现该函数，则需在 CHANNEL_CAPABILITY 属性被设置为 1。

sbiret.error 中可能返回的错误代码如下。

表 115. MPXY 发送带响应消息的错误类型

错误代码	描述
SBI_SUCCESS	消息发送成功并收到响应

Error code	Description
SBI_ERR_INVALID_PARAM	The `message_data_len > MSG_DATA_MAX_LEN` for specified `channel_id`. Or the `message_data_len` is greater than the size of shared memory on the calling hart.
SBI_ERR_NOT_SUPPORTED	`channel_id` is not supported or invalid. Or the message represented by the `message_id` is not supported or invalid. Or this function is not supported.
SBI_ERR_NO_SHMEM	The shared memory setup is not done or disabled for calling hart.
SBI_ERR_TIMEOUT	Waiting for response timeout.
SBI_ERR_IO	Failed due to I/O error.
SBI_ERR_FAILED	Failed due to other unspecified errors.

20.11. Function: Send Message without Response (FID #6)

```
struct sbiret
sbi_mpxy_send_message_without_response(uint32_t channel_id,
                                       uint32_t message_id,
                                       unsigned long message_data_len)
```

Send a message to the MPXY channel specified by the `channel_id` parameter without waiting for a message response from the MPXY channel. The `message_id` parameter specifies the message protocol specific identification of the message to be sent whereas the `message_data_len` parameter represents the length of message data in bytes which is located at the offset `0x0` in the shared memory setup by the calling hart.

This function does not wait for message response from the channel and returns after successful message transmission. In cases where complete data transfer requires multiple transmissions, the supervisor software shall send multiple messages as necessary. Details of such cases can be found in the respective message protocol specification.

The messages which do not have an expected response as per the underlying message protocol specification are also referred to as posted messages. This function should be only used for such posted messages. The respective message protocol specification should define a mechanism to track the status of posted messages using notification events or some other message with response if required.

This function is optional. If this function is implemented, the corresponding bit in the `CHANNEL_CAPABILITY` attribute is set to `1`.

The possible error codes returned in `sbiret.error` are below.

Table 116. MPXY Send Message without Response Errors

Error code	Description
SBI_SUCCESS	Message sent successfully.
SBI_ERR_INVALID_PARAM	The `message_data_len > MSG_DATA_MAX_LEN` for specified `channel_id`. Or the `message_data_len` is greater than the size of shared memory on the calling hart.

错误码	描述
SBI_ERR_INVALID_PARAM	指定 channel_id 对应的 message_data_len 超过 MSG_DATA_MAX_LEN 限制。 或 message_data_len 超出调用 hart 共享内存区域的大小。
SBI_ERR_NOT_SUPPORTED	channel_id 不被支持或无效。 或者 message_id 所代表的消息不被支持或无效。 或者该功能不被支持。
SBI_ERR_NO_SHMEM	调用 hart 未设置或禁用了共享内存配置
SBI_ERR_TIMEOUT	等待响应超时
SBI_ERR_IO	由于 I/O 错误导致失败。
SBI_ERR_FAILED	由于其他未指定错误导致失败。

20.11. 功能：发送无响应消息（功能号 #6）结构体 sbiret

```
sbi_mpxy_send_message_without_response(uint32_t channel_id,
                                       uint32_t message_id, unsigned long
                                       message_data_len)
```

向 channel_id 参数指定的 MPXY 通道发送消息，无需等待该通道的响应消息。message_id 参数指定待发送消息的协议特定标识，而 message_data_len 参数表示位于调用 hart 所设共享内存 0x0 偏移处的消息数据字节长度。本函数不会等待通道的响应消息，成功发送后立即返回。若完整数据传输需多次发送，监管模式软件应按需发送多条消息。此类情况的详细说明请参阅相应消息协议规范。

根据底层消息协议规范无需预期响应的消息亦称为投递消息。此函数应仅用于此类投递消息。相关消息协议规范应定义相应机制，通过通知事件或必要时借助含响应的其他消息来追踪投递消息状态。

此函数为可选实现。若实现该函数，则 CHANNEL_CAPABILITY 属性中对应比特位应置为 1。

sbiret.error 可能返回的错误代码如下。

表 116. MPXY 发送无响应消息的错误代码

错误代码	描述
SBI_SUCCESS	消息发送成功。
SBI_ERR_INVALID_PARAM	指定通道 ID 的 message_data_len 超过 MSG_DATA_MAX_LEN 限制 或 message_data_len 大于调用 hart 上共享内存的大小

Error code	Description
SBI_ERR_NOT_SUPPORTED	`channel_id` is not supported or invalid. Or the message represented by the `message_id` is not supported or invalid. Or this function is not supported.
SBI_ERR_NO_SHMEM	The shared memory setup is not done or disabled for calling hart.
SBI_ERR_TIMEOUT	Message send timeout.
SBI_ERR_IO	Failed due to I/O error.
SBI_ERR_FAILED	Failed due to other unspecified errors.

20.12. Function: Get Notifications (FID #7)

```
struct sbiret sbi_mpxy_get_notification_events(uint32_t channel_id)
```

Get the message protocol specific notification events on the MPXY channel specified by the `channel_id` parameter. The events are message protocol specific and MUST be defined in the respective message protocol specification. The SBI implementation may support indication mechanisms like MSI or SSE to inform the supervisor software about the availability of events.

If the message channel does not support or is not configured for an indication mechanism, such as MSI or SSE, the supervisor software can periodically invoke the poll operation `sbi_mpxy_get_notification_events`.

Notifications are asynchronous from the perspective of the supervisor software. Any caching or buffering mechanism is specific to the SBI implementation. The supervisor software may periodically poll for notification events using this function, provided that the polling frequency is sufficient to prevent the loss of events due to potential buffer limitations in the SBI implementation.

Depending on the message protocol implementation, a channel may support events state which includes data like number of events `RETURNED`, `REMAINING` and `LOST`. Events state data is optional, and if the message protocol implementation supports it, then the channel will have the corresponding bit set in the `CHANNEL_CAPABILITY` attribute. By default the events state is disabled and supervisor software can explicitly enable it through the `EVENTS_STATE_CONTROL` attribute.

Only after enabling the events state reporting through `EVENTS_STATE_CONTROL` *attribute, the events state data will start getting accumulated by the SBI implementation. The supervisor software may enable the* `EVENTS_STATE_CONTROL` *attribute in the initialization phase if it is supported.*

In the shared memory, 16 bytes starting from offset `0x0` are used to return this state data.

Shared memory layout with events state data (each field is of 4 bytes):

```
Offset 0x0: REMAINING
Offset 0x4: RETURNED
Offset 0x8: LOST
Offset 0xC: RESERVED
Offset 0x10: Start of message protocol specific notification events data
```

错误代码	描述
SBI_ERR_NOT_SUPPORTED	channel_id 不受支持或无效。 或者 message_id 所代表的消息不受支持或无效。 或者该功能不受支持。
SBI_ERR_NO_SHMEM	共享内存设置未完成或对调用核心禁用。
SBI_ERR_TIMEOUT	消息发送超时。
SBI_ERR_IO	因 I/O 错误导致失败。
SBI_ERR_FAILED	因其他未指定错误导致失败。

20.12. 功能：获取通知事件（功能号 #7） struct sbiret sbi_mpxy_get_notification_events(uint32_t channel_id)

获取由 channel_id 参数指定的 MPXY 通道上消息协议特定的通知事件。这些事件与消息协议相关，且必须在相应的消息协议规范中定义。SBI 实现可支持如 MSI 或 SSE 等指示机制，以通知监督模式软件事件的可用性。

☒ 若消息通道不支持或未配置指示机制（如 MSI 或 SSE），监督模式软件可定期调用轮询操作 sbi_mpxy_get_notification_events .

☒ 从监管软件的角度来看，通知是异步的。任何缓存或缓冲机制都取决于具体的 SBI 实现。监管软件可以定期使用此函数轮询通知事件，前提是轮询频率足够高，以防止由于 SBI 实现中潜在的缓冲区限制而导致事件丢失。

根据消息协议的具体实现，通道可能支持包含事件状态的数据，例如已返回（RETURNED）、剩余（REMAINING）和丢失（LOST）事件的数量。事件状态数据是可选的，如果消息协议实现支持该功能，则通道将在 CHANNEL_CAPABILITY 属性中设置相应的标志位。默认情况下事件状态功能处于禁用状态，监管软件可通过 EVENTS_STATE_CONTROL 属性显式启用该功能。

☒ 只有通过 EVENTS_STATE_CONTROL 属性启用事件状态报告后，SBI 实现才会开始累积事件状态数据。若该功能受支持，监管软件可在初始化阶段启用 EVENTS_STATE_CONTROL 属性。

在共享内存中，从偏移量 0x0 开始的 16 字节用于返回此状态数据。

带事件状态数据的共享内存布局（每个字段占 4 字节）：

偏移量 0x0：剩余数量 偏移量 0x4：已返回数量 偏移量 0x8：丢失数量 偏移量 0xC：保留字段 偏移量 0x10：消息协议特定通知事件数据的起始位置

The `RETURNED` field represents the number of events which are returned in the shared memory when this function is called. The `REMAINING` field represents the number of events still remaining with SBI implementation. The supervisor software may need to call this function again until the `REMAINING` field becomes `0`.

The `LOST` field represents the number of events which are lost due to limited buffer size managed by the message protocol implementation. Details of buffering/caching of events is specific to message protocol implementation.

Upon calling this function the received notification events are written by the SBI implementation at the offset `0x10` in the shared memory setup by the calling hart irrespective of events state data reporting. If events state data reporting is disabled or not supported, then the values in events state fields are undefined. The number of the bytes written to the shared memory will be returned through `sbiret.uvalue` which is the number of bytes starting from offset `0x10`. The layout and encoding of notification events are defined by the message protocol specification associated with the message proxy channel (`channel_id`).

This function is optional. If this function is implemented, the corresponding bit in the `CHANNEL_CAPABILITY` attribute is set to `1`.

The possible error codes returned in `sbiret.error` are below.

Table 117. MPXY Get Notifications Errors

Error code	Description
SBI_SUCCESS	Notifications received successfully.
SBI_ERR_NOT_SUPPORTED	`channel_id` is not supported or invalid. Or this function is not supported.
SBI_ERR_NO_SHMEM	The shared memory setup is not done or disabled for calling hart.
SBI_ERR_IO	Failed due to I/O error.
SBI_ERR_FAILED	Failed due to other unspecified errors.

20.13. Function Listing

Table 118. MPXY Function List

Function Name	SBI Version	FID	EID
sbi_mpxy_get_shmem_size	3.0	0	0x4D505859
sbi_mpxy_set_shmem	3.0	1	0x4D505859
sbi_mpxy_get_channel_ids	3.0	2	0x4D505859
sbi_mpxy_read_attributes	3.0	3	0x4D505859
sbi_mpxy_write_attributes	3.0	4	0x4D505859
sbi_mpxy_send_message_with_response	3.0	5	0x4D505859
sbi_mpxy_send_message_without_response	3.0	6	0x4D505859
sbi_mpxy_get_notification_events	3.0	7	0x4D505859

RETURNED 字段表示调用此函数时在共享内存中返回的事件数量。REMAINING 字段表示 SBI 实现中仍剩余的事件数量。监控程序软件可能需要重复调用此函数，直到 REMAINING 字段变为 0。

LOST 字段表示由于消息协议实现管理的缓冲区大小有限而丢失的事件数量。事件缓冲/缓存的细节取决于具体的消息协议实现。

调用此函数时，SBI 实现会将接收到的通知事件写入调用 hart 设置的共享内存偏移量 0x10 处，无论事件状态数据报告是否启用。如果事件状态数据报告被禁用或不支持，则事件状态字段中的值未定义。写入共享内存的字节数将通过 sbiret.uvalue 返回，该值表示从偏移量 0x10 开始的字节数。通知事件的布局和编码由与消息代理通道(channel_id)相关联的消息协议规范定义。

此函数为可选实现。若实现了该函数，则对应通道的

CHANNEL_CAPABILITY 属性位将被置为 1。

可能的错误码通过 sbiret.error 返回如下。

表 117. MPXY 获取通知错误码

错误码	描述
SBI_SUCCESS	通知接收成功。
SBI_ERR_NOT_SUPPORTED	channel_id 不受支持或无效。 或此功能不受支持。
SBI_ERR_NO_SHMEM	共享内存设置未完成或对调用核心禁用。
SBI_ERR_IO	因 I/O 错误导致失败。
SBI_ERR_FAILED	因其他未指定错误导致失败。

20.13. 函数列表

表 118. MPXY 函数列表

函数名称	SBI 版本	FID	EID
sbi_mpxy_get_shmem_size	3.0	0	0x4D505859
sbi_mpxy_set_shmem	3.0	1	0x4D505859
sbi_mpxy_get_channel_ids	3.0	2	0x4D505859
sbi_mpxy_read_attributes	3.0	3	0x4D505859
sbi_mpxy_write_attributes	3.0	4	0x4D505859
sbi_mpxy_send_message_with_response	3.0	5	0x4D505859
sbi_mpxy_send_message_without_response	3.0	6	0x4D505859
sbi_mpxy_get_notification_events	3.0	7	0x4D505859

Chapter 21. Experimental SBI Extension Space (EIDs #0x08000000 - #0x08FFFFFF)

The SBI specification doesn't define any rules for the EID management for experimental SBI extensions.

第 21 章 实验性 SBI 扩展空间（EID 编号范围 #0x08000000 - #0x08FFFFFF）

SBI 规范未对实验性 SBI 扩展的 EID 管理制定任何规则。

Chapter 22. Vendor Specific Extension Space (EIDs #0x09000000 - #0x09FFFFFF)

The lower 24 bits of vendor specific EID must match the lower 24 bits of the `mvendorid` value.

第 22 章 厂商特定扩展空间 (扩展 ID 范围 #0x09000000 - #0x09FFFFFF)

厂商特定扩展 ID 的低 24 位必须与 mvendorid 值的低 24 位相匹配。

Chapter 23. Firmware Specific Extension Space (EIDs #0x0A000000 - #0x0AFFFFFF)

The lower 24 bits of the firmware EID must match the lower 24 bits of the SBI implementation ID. The firmware specific SBI extensions space is reserved for SBI implementations. It provides firmware specific SBI functions which are defined in the external firmware specification.

第 23 章 固件特定扩展空间（EID 范围 #0x0A000000 - #0x0AFFFFFF）

固件 EID 的低 24 位必须与 SBI 实现 ID 的低 24 位相匹配。固件特定的 SBI 扩展空间专为 SBI 实现保留，提供由外部固件规范定义的固件特定 SBI 功能。

References

[1] "The RISC-V Instruction Set Manual, Volume II: Privileged Architecture." 2021, [Online]. Available: github.com/riscv/riscv-isa-manual/releases/tag/Priv-v1.12.

[2] "The RISC-V Debug Specification." 2024, [Online]. Available: github.com/riscv/riscv-debug-spec/releases/tag/1.0.0-rc3.

[3] "The RISC-V Platform Management Interface Specification." 2024, [Online]. Available: github.com/riscv-non-isa/riscv-rpmi/blob/main/riscv-rpmi.adoc.

[4] "Perf (Linux)." 2025, [Online]. Available: en.wikipedia.org/wiki/Perf_(Linux).

参考文献

[1] 《RISC-V 指令集手册 第二卷：特权架构》. 2021 年, [在线]. 获取地址: github.com/riscv/riscv-isa-manual/releases/tag/Priv-v1.12.

[2] 《RISC-V 调试规范》. 2024 年, [在线]. 获取地址: github.com/riscv/riscv-debug-spec/releases/tag/1.0.0-rc3.

[3] 《RISC-V 平台管理接口规范》. 2024 年, [在线]. 获取地址: github.com/riscv-non-isa/riscv-rpmi/blob/main/riscv-rpmi.adoc.

[4] "Perf (Linux 性能分析工具)". 2025 年, [在线]. 参见: en.wikipedia.org/wiki/Perf_(Linux).