RISC-V监管
二进制接口规范

RISC-V 平台规范工作组

版本1.0.0, 2022年3月 22: 已批准

目录

目录

前言

> *该文件已批准*
>
> *不允许更改。任何更改都可以作为后续文档的修改讨论，且修订版本号与 RISC-V 版本一致。*

版权许可信息

本 RISC-V SBI 规范：

更改日志

版本 1.0.0

- 更新批准版本

版本 1.0-rc3

- 更新调用约定
- 修复了PMU扩展名中的拼写错误
- 添加缩写表

版本 1.0-rc2

- 更新RISC-V格式
- 修改介绍部分
- 删除了所有RV32引用

版本 1.0-rc1

- 错别字修正

版本 0.3.0

- 少量错别字修正
- 文本更新LICENSE部分的超链接

版本 0.3-rc1

- 改进文档样式和命名约定
- 添加SBI系统重置扩展
- 修改SBI介绍部分
- 修改SBI hart状态管理扩展文档
- 为SBI hart状态管理扩展添加挂起
- 添加性能监控单元扩展
- 阐明一个不应局部实现的SBI扩展

版本 0.2

- 整个v0. 1 SBI已移动至旧版扩展，现已是可选扩展。

从技术上讲，这是一个向后不兼容的更改，因为旧版本的扩展是可选的，并且SBI v0.1版本不允许探测，但我们已经尽力了。

章节1 介绍

本规范阐述RISC−V Supervisor Binary Interface，简称为SBI，其允许在所有RISC−V实现上，通过定义平台（或虚拟化管理程序）特定功能的抽象，令处于监管者模式（S模式或VS模式）的软件具备可移植性。SBI的设计遵循RISC−V的一般原则，即小而精简，同时具备一组可选的模块化扩展功能。

SBI扩展作为整体是可选的，但不能部分实现。如果sbi_probe_extension()表示某个扩展可用，那么sbi_get_spec_version()报告的SBI版本中的所有函数必须符合该版本的SBI规范。

提供给监管模式软件的更高特权级软件称为SBI实现或Supervisor Execution Environment（SEE）。SBI实现（或SEE）可以是在机器模式（M-mode）下执行的平台运行时固件（参见下图1），也可以是在超级监管模式（HS-mode）下执行的某个虚拟化监管程序（参见下图2）。

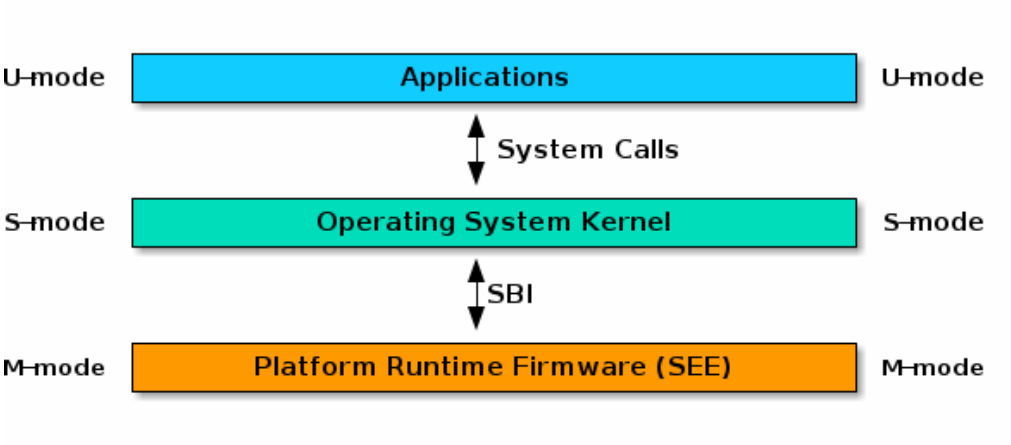

图1.无H扩展 RISC-V系统

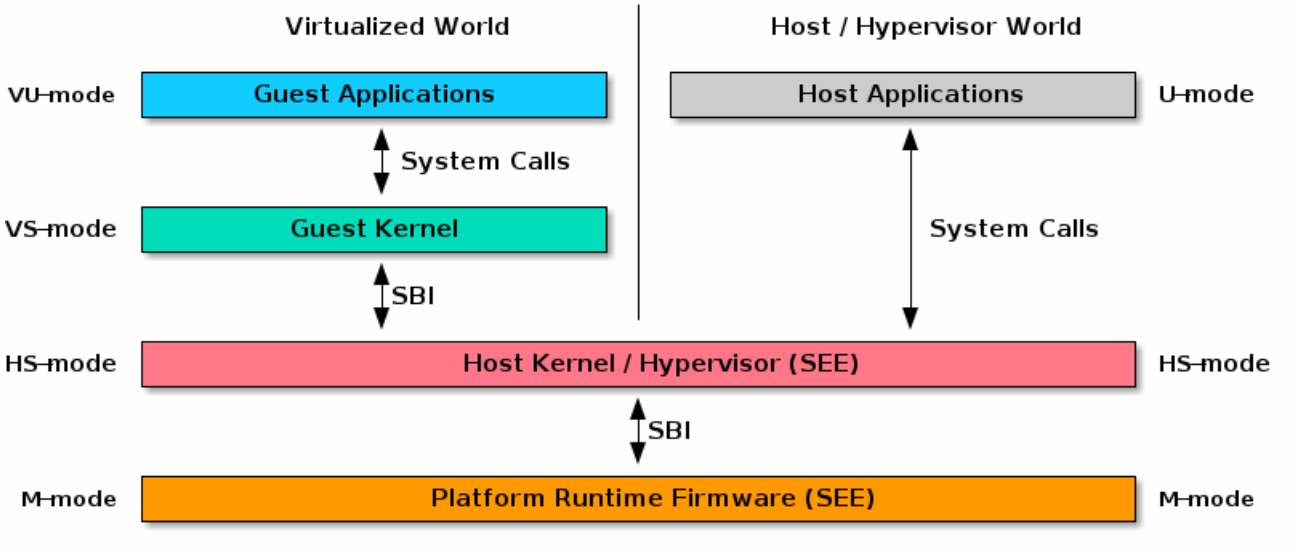

图2.带H扩展 RISC-V系统

SBI规范并未指定硬件发现的方法。监管软件必须依赖其他行业标准的硬件发现方法（例如设备树或ACPI）来实现。

章节2 术语和缩写

本规范使用了下列术语及缩写：

术语	含义
SBI	监管二进制接口
SEE	监管执行环境
EID	扩展号
FID	函数号
HSM	核心状态管理
PMU	性能监控单元
IPI	处理器核间中断
ASID	地址空间标识符
VMID	虚拟机标识符

章节3 二进制编码

所有的SBI函数共享一种二进制编码方式，这有助于混合使用SBI扩展功能。SBI规范遵循以下调用约定。

- 在监管者和SEE之间，使用ECALL作为控制传输指令。
- a7编码SBI扩展ID（EID）
- a6编码SBI函数ID（FID），对于任何在a7中编码的SBI扩展，其定义在SBI v0.2之后。
- 在SBI调用期间，除了a0和a1寄存器外，所有寄存器都必须由被调用方保留。
- SBI函数必须在a0和a1中返回一对值，其中a0返回错误代码。类似返回C结构体。

```
struct sbiret
 {
   long error;
   long value;
    };
```

为了保持兼容性，SBI扩展ID（EID）和SBI函数ID（FID）被编码为有符号的32位整数。当以寄存器形式传递时，遵循上述标准的调用约定规则。

表1列出了标准SBI错误代码。

表1. 标准SBI错误

错误类型	值
SBI_SUCCESS 成功	0
SBI_ERR_FAILED 失败	-1
SBI_ERR_NOT_SUPPORTED 不支持操作	-2
SBI_ERR_INVALID_PARAM 非法参数	-3
SBI_ERR_DENIED 拒绝	-4
SBI_ERR_INVALID_ADDRESS 非法地址	-5
SBI_ERR_ALREADY_AVAILABLE (资源)已可用	-6
SBI_ERR_ALREADY_STARTED (操作)已启动	-7
SBI_ERR_ALREADY_STOPPED (操作)已停止	-8

不支持的SBI扩展ID（EID）或SBI函数ID（FID）的ECALL必须返回错误代码SBI_ERR_NOT_SUPPORTED。

每个SBI函数应该优先选择无符号长整型unsigned long作为数据类型。这使得规范简单且易于适应所有RISC-V ISA类型。如果数据被定义为32位宽度，则更高权限的软件必须确保只使用32位数据。

如果SBI函数需要将一组hart传递给更高权限模式，它必须使用如下所定义的hart掩码。这适用于在v0.2版本之后定义的任何扩展。

任何需要hart掩码的函数需要传递以下两个参数。

- unsigned long hart_mask是一个包含hartid的标量位向量。
- unsigned long hart_mask_base 是计算位向量的起始hartid。

在单个SBI函数调用中，可以设置的最大hart数量始终为XLEN。如果较低特权模式需要传递超过XLEN个hart的信息，它应该调用多个SBI函数调用的实例。hart_mask_base可以设置为-1，表示可以忽略hart_mask并考虑所有可用的harts。

使用hart掩码的任何函数可能返回表2中列出的错误值，这些错误值是针对特定函数的错误值的补充。

表2. HART掩码错误

错误代码	描述
SBI_ERR_INVALID_PARAM 无效参数	hart_mask_base 或hart_mask 中任何一个hartid无效，即该hartid未被平台启用或对监管程序不可用。

章节4 基本扩展(EID #0x10)

基本扩展旨在尽可能简洁。因此，它仅包含用于探测可用的SBI扩展以及查询SBI版本的功能。基本扩展中的所有函数必须由所有SBI实现支持，因此没有定义错误返回值。

4.1函数:获取SBI规范版本 (FID #0)

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

返回当前的SBI规范版本。此函数必定成功。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，保留用于未来扩展。

4.2函数: 获取SBI实现标识符 (FID #1)

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

返回当前 SBI 实现的标识符，每个 SBI 实现的标识符都是不同的。这个标识符旨在让软件探测 SBI 实现的特殊问题或特点。

4.3函数:获取SBI实现版本(FID #2)

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

返回当前 SBI 实现的版本。该版本号的编码是特定于 SBI 实现的。

4.4函数: 探测SBI扩展功能 (FID #3)

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

如果给定的 SBI 扩展 ID (EID) 不可用，则返回 0；如果可用，且实现定义为其他非零值则返回 1。

4.5函数: 获取机器供应商标识符 (FID #4)

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

返回一个合法的 mvendorid CSR 值，其中 0 总是一个合法的值。mvendorid CSR 是一个用于标识机器供应商的控制状态寄存器，它用于表示底层硬件的供应商或制造商。

4. 6函数： 获取机器体系结构标识符 （FID #5）

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

返回一个在 marchid CSR 中合法的值，其中 0 总是合法的值。marchid CSR是RISC-V架构中的一个控制状态寄存器，用于标识机器体系结构。

4. 7函数： 获取机器实现标识符ID （FID #6）

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

返回一个在 mimpid CSR 中合法的值，而且对于该 CSR，0 始终是一个合法的值。

4. 8函数列表

表 3.基础函数列表

函数名	SBI 版本	FID	EID
sbi_get_sbi_spec_version	0.2	0	0x10
sbi_get_sbi_impl_id	0.2	1	0x10
sbi_get_sbi_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. 9SBI 实现标识符

表4. SBI 实现IDs

SBI实现ID	名称
0	Berkeley Boot Loader (BBL)
1	OpenSBI
2	Xvisor

SBI实现ID	名称
3	KVM
4	RustSBI
5	Diosix
6	Coffer

章节5. 旧版扩展 (EIDs #0x00-#0x0F)

传统的 SBI 扩展与 SBI v0.2（或更高版本）规范相比，遵循略微不同的调用约定，其中：

- a6 寄存器中的 SBI 函数ID 字段被忽略，因为这些被编码为多个 SBI 扩展 ID。
- a1寄存器中不返回任何值。
- 在 SBI 调用期间，除 a0 寄存器外的所有寄存器都必须由被调用者保留。
- a0 寄存器中返回的值是特定于 SBI 传统扩展的。
- SBI 实现在监管者访问内存时发生的页面和访问故障会被重定向回监管者，并且 sepc CSR指向故障的 ECALL指令。

传统 SBI 扩展已被以下扩展所取代。传统的控制台 SBI 函数（sbi_console_getchar() 和 sbi_console_putchar()）预计将被弃用；它们没有替代方案。

5.1扩展： 设置时钟 (EID #0x00)

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

设置时钟，在stime_value时间之后进行下一次事件。此功能还清除待处理的计时器中断位。

如果监管程序希望清除计时器中断但不安排下一个计时器事件，则可以将计时器中断请求设置为无限远（即（uint64_t）-1），或者通过清除sie. STIE 寄存器位来屏蔽计时器中断。

此SBI调用在成功时返回0，否则返回实现特定的负错误代码。

5.2扩展:控制台字符输出 (EID #0x01)

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

将 **ch** 中的数据写入控制台。

与sbi_console_getchar()不同的是，如果仍有任何待传输的字符或接收终端还没有准备好接收字节，则此SBI调用将阻塞。但是，如果控制台根本不存在，则字符将被丢弃。

此SBI调用在成功时返回0，否则返回实现特定的负错误代码。

5.3扩展： 控制台字符输入 （EID #0x02）

```
long  sbi_console_getchar(void)
```

从调试控制台中读一个字符。

此SBI调用在成功时返回0，否则返回实现特定的负错误代码。

5.4扩展： 清除IPI （EID #0x03）

```
long sbi_clear_ipi(void)
```

清除任何挂起的IPI（处理器核间中断）。 这个SBI调用只会清除被调用的hart（硬件线程），其他的hart不受影响。

sbi_clear_ipi()已经被弃用，因为S模式代码可以直接清除 sip.SSIP寄存器位。.

这个SBI调用会返回0，如果没有IPI被挂起，则返回一个实现特定的正值，如果有IPI被挂起。

5.5扩展： 发送IPI （EID #0x04）

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

向hart_mask指定的所有hart发送跨处理器中断。跨处理器中断在接收的hart上表现为Supervisor模式软件中断。

hart_mask是指向hart位向量的虚拟地址。该位向量表示为unsigned long无符号长整型序列，其长度等于系统中hart的数量除以unsigned long中的位数，向上取整到下一个整数。

此SBI调用在成功时返回0，或返回一个实现特定的负错误代码。

5.6扩展： 远程FENCE.I （EID #0x05）

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

该函数用于指示远程处理器执行FENCE.I指令。

此SBI调用在成功时返回0，或返回一个实现特定的负错误代码。

5.7扩展： 远程SFENCE.VMA （EID #0x06）

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

指示远程 harts 执行一个或多个 SFENCE.VMA 指令，涵盖从 start 到 size 的虚拟地址范围。其中，hart_mask 参数与 sbi_send_ipi() 函数中描述的一样。

此SBI调用在成功时返回0，或返回一个实现特定的负错误代码。

5.8扩展： 远程SFENCE.VMA（指定地址空间标识符） （EID #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hart执行一个或多个SFENCE.VMA指令，覆盖start和size之间的虚拟地址范围。此操作仅涵盖给定的ASID。

此SBI调用在成功时返回0，或返回一个实现特定的负错误代码。

5.9扩展： 系统关闭 （EID #0x08）

```
void  sbi_shutdown(void)
```

将所有hart从监管者模式的角度置于关闭状态。

此SBI调用在成功时返回0，或返回一个实现特定的负错误代码。

5.10 函数列表

表 5. 旧版函数列表

函数名	SBI 版本	FID	EID	替代 EID
sbi_set_timer	0.1	0	0x00	0x54494D45
sbi_console_putchar	0.1	0	0x01	N/A
sbi_console_getchar	0.1	0	0x02	N/A
sbi_clear_ipi	0.1	0	0x03	N/A
sbi_send_ipi	0.1	0	0x04	0x735049

函数名	SBI 版本	FID	EID	替代 EID
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
保留			0x09-0x0F	

章节6. 时钟扩展 （EID #0x54494D45 "TIME"）

这个替代扩展（EID #0x00）替代了传统的计时器扩展。它遵循在v0.2中定义的新的调用约定。

6.1函数： 时钟设定 （FID #0）

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

在 stime_value 时间之后，为下一个事件设置时钟。stime_value 以绝对时间表示。此函数还必须清除挂起的计时器中断位。

如果监管者希望在不安排下一个计时器事件的情况下清除计时器中断，可以请求一个无限远的计时器中断（即(uint64_t)-1），或者通过清除 sie.STIE 寄存器位来屏蔽计时器中断。

6.2函数列表

表6. 时钟函数列表

函数名	SBI 版本	FID	EID
sbi_set_timer	0.2	0	0x54494D45

章节7 IPI 扩展(EID #0x735049 "sPI: s-mode IPI")

该扩展替代了传统的扩展(EID #0x04)。其他与 IPI 相关的传统扩展(0x3)现已不推荐使用。该扩展中的所有函数都遵循二进制编码部分中定义的 hart_mask。

7.1函数: 发送IPI (FID #0)

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

向 hart_mask 中定义的所有 hart 发送跨处理器中断。接收 hart 上的处理器间中断将在其上表现为超级处理器软件中断。

sbiret.error返回的可能错误代码如表7所示。

表7. IPI 发送错误s

错误代码	描述
SBI_SUCCESS成功	IPI被成功发送至所有目标harts。

7.2函数列表

表8. IPI 函数列表

函数名	SBI 版本	FID	EID
sbi_send_ipi	0.2	0	0x735049

章节8. RFENCE扩展 (EID #0x52464E43 "RFNC")

该扩展定义了所有与远程屏障相关的函数，并替代了旧的扩展（EID #0x05 - #0x07）。所有的函数都遵循二进制编码部分中定义的 hart_mask。任何希望使用地址范围（即 start_addr 和 size）的函数，必须遵守以下对范围参数的约束条件。

如果以下条件满足，则远程屏障函数充当完全TLB刷新的作用：

- start_addr 和 size 都为0
- size等于2^XLEN-1

8.1函数: 远程 FENCE.I指令 (FID #0)

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

远程指示harts执行FENCE.I指令。

sbiret.error返回的可能错误代码如表9所示。

表9. RFENCE 远程FENCE.I指令错误

错误代码	描述
SBI_SUCCESS 成功	IPI被成功发送至所有目标harts。

8.2函数: 远程SFENCE.VMA指令 (FID #1)

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

指示远程hart执行一个或多个SFENCE.VMA指令，覆盖从start到size的虚拟地址范围内的地址。

sbiret.error返回的可能错误代码如表10所示。

表10. RFENCE 远程SFENCE.VMA 错误

错误代码	描述
SBI_SUCCESS成功	IPI被成功发送至所有目标harts。
SBI_ERR_INVALID_ADDRESS 非法地址	start_addr 或size非法

8.3函数： 远程SFENCE.VMA 指定 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)
```

指示远程hart执行一个或多个SFENCE.VMA指令，覆盖从start到size的虚拟地址范围内的地址。这仅涵盖给定的ASID。

sbiret.error返回的可能错误代码如表11所示。

表11. RFENCE 远程SFENCE.VMA 指定ASID 错误

错误代码	描述
SBI_SUCCESS成功	IPI被成功发送至所有目标harts。
SBI_ERR_INVALID_ADDRESS 非法地址	start_addr 或size非法

8.4函数： 远程HFENCE.GVMA 指定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)
```

指示远程hart执行一个或多个HFENCE.GVMA指令，仅覆盖给定VMID（虚拟机标识符）的start到size之间的客户机物理地址范围。此函数调用仅适用于实现了虚拟化扩展的harts。

sbiret.error返回的可能错误代码如表12所示。

表12. RFENCE 远程HFENCE.GVMA 指定VMID 错误

错误代码	描述
SBI_SUCCESS成功	IPI被成功发送至所有目标harts。
SBI_ERR_NOT_SUPPORTED （操作）不支持	由于未被实现或目标hart中不错支持虚拟化扩展，则该操作不受支持。
SBI_ERR_INVALID_ADDRESS 非法地址	start_addr 或size非法

8.5函数： 远程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)
```

指示远程 harts 执行一个或多个HFENCE.GVMA指令，覆盖从 start 到 size 之间的所有客户机的客户机物理地址范围。此函数调用仅适用于实现虚拟化扩展的 harts。

sbiret.error返回的可能错误代码如表13所示。

表13. RFENCE 远程HFENCE.GVMA 错误

错误代码	描述
SBI_SUCCESS成功	IPI被成功发送至所有目标harts。
SBI_ERR_NOT_SUPPORTED（操作）不支持	由于未被实现或目标hart中不错支持虚拟化扩展，则该操作不受支持。
SBI_ERR_INVALID_ADDRESS非法地址	start_addr 或size非法

8.6函数： 远程HFENCE.VVMA 指定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)
```

远程指示harts执行一个或多个 HFENCE.VVMA 指令，覆盖从start至size之间，指定的ASID与VMID（hgatp寄存器）下所有客户机物理地址范围。此函数调用仅适用于实现虚拟化扩展的 harts。

sbiret.error返回的可能错误代码如表14所示。

表14. RFENCE 远程HFENCE.VVMA 指定ASID 错误

错误代码	描述
SBI_SUCCESS成功	IPI被成功发送至所有目标harts。
SBI_ERR_NOT_SUPPORTED（操作）不支持	由于未被实现或目标hart中不错支持虚拟化扩展，则该操作不受支持。
SBI_ERR_INVALID_ADDRESS非法地址	start_addr 或size非法

8.7函数： 远程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)
```

指示远程 HART 执行一个或多个 HFENCE.VVMA 指令，覆盖当前调用 HART 的 VMID（在hgatp寄存器中）下的start和size之间的客户虚拟地址范围。此函数调用仅适用于实现了虚拟化扩展的 harts。

sbiret.error返回的可能错误代码如表15所示。

表15.RFENCE 远程 HFENCE.VVMA 错误

错误代码	描述
SBI_SUCCESS成功	IPI被成功发送至所有目标harts。
SBI_ERR_NOT_SUPPORTED（操作）不支持	由于未被实现或目标hart中不错支持虚拟化扩展，则该操作不受支持。
SBI_ERR_INVALID_ADDRESS非法地址	start_addr 或size非法

8.8函数列表

表16. RFENCE 函数列表

函数名	SBI 版本	FID	EID
sbi_remote_fence_i	0.2	0	0x52464E43
sbi_remote_sfence_vma	0.2	1	0x52464E43
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

章节9 Hart状态管理扩展 （EID #0x48534D ″HSM″）

Hart State Management (HSM) 扩展引入了一组 Hart 状态和一组函数，允许S-mode监管者模式下的软件请求 Hart 状态变更。

下面的表17描述了所有可能的 HSM 状态以及每个状态的唯一 HSM 状态标识符。:

表17. HSM Hart 状态

状态ID	状态名	描述
0	STARTED 启动	该 hart 处于物理上电状态，并正常执行。
1	STOPPED 结束	该 hart 没有在监管者模式S-mode或任何更低特权模式下执行。如果底层平台具有关闭 hart 的物理机制，那么它可能被 SBI 实现关闭了电源。
2	START_PENDING 等待被启动	另一个 hart 请求将该 hart 从STOPPED 状态启动（或上电），而 SBI 实现仍在努力将该 hart 转换为STARTED状态。
3	STOP_PENDING 结束等待	该 hart 在STARTED状态下请求停止（或关机），而 SBI 实现仍在努力将该 hart 转变为STOPPED状态。
4	SUSPENDED 挂起	该 hart 处于特定于平台的暂停（或低功耗）状态。
5	SUSPEND_PENDING 等待挂起	该 hart 从STARTED状态请求将自身置于特定于平台的低功耗状态，并且 SBI 实现正在努力将该 hart 转换为特定于平台的SUSPENDED状态。
6	RESUME_PENDING 等待恢复	中断或特定于平台的硬件事件导致 hart 从SUSPENDED状态恢复正常执行，而 SBI 实现正在努力将该 hart 转换为STARTED状态。

在任何时刻，hart 应该处于上述提到的 hart 状态之一。SBI 实现对 hart 状态的转换应遵循图3所示的状态机。

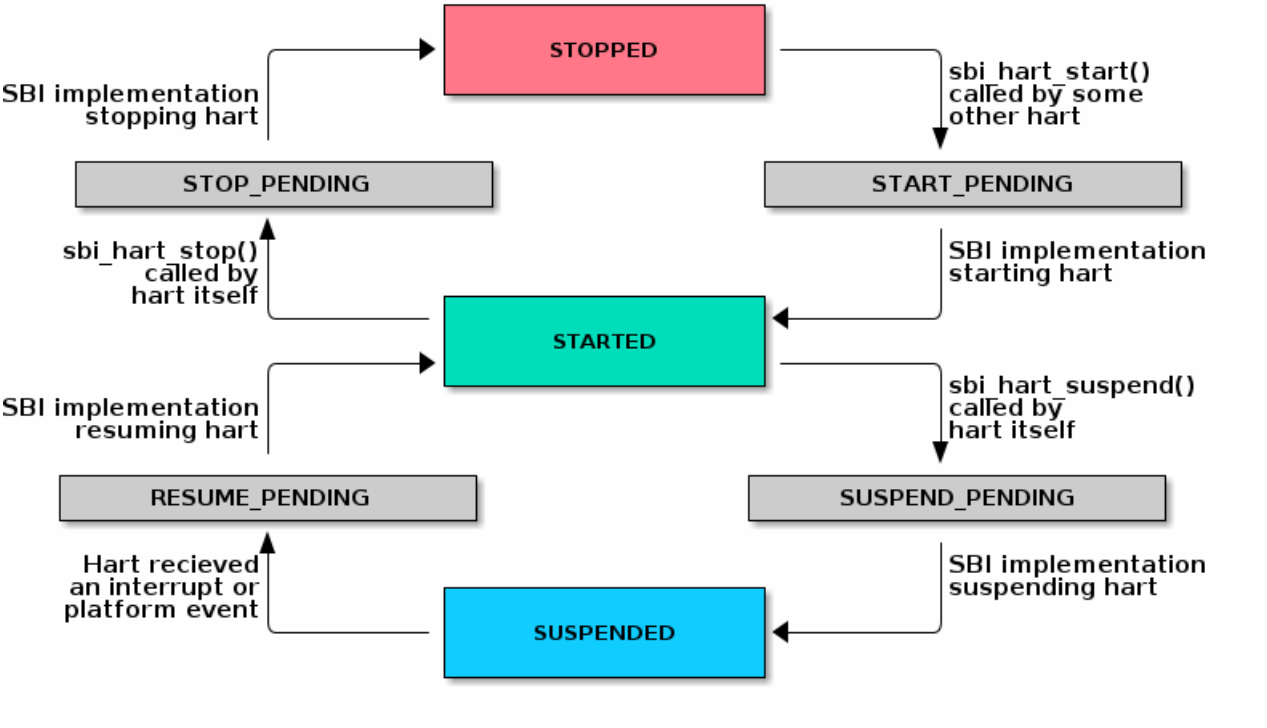

图3. SBI HSM 状态机

一个平台可以有多个 hart，这些 hart 被分组成层次拓扑组（如核心、集群、节点等），每个层次组都有独立的平台特定低功耗状态。这些层次拓扑组的平台特定低功耗状态可以表示为一个 hart 的平台特定挂起状态。SBI 实现可以利用较高层次拓扑组的挂起状态，使用以下一种方法之一：

1. Platform-coordinated **平台协调**: 在这种方法中，当一个 hart 变为空闲时，监管模式S-mode的功耗管理软件将请求该 hart 和较高层次组的最深挂起状态。SBI 实现应选择一个较高层次组的挂起状态，满足以下条件：
   a. 不比指定的挂起状态更深
   b. 唤醒延迟不高于指定挂起状态的唤醒延迟
2. OS-inititated **操作系统发起**: 在这种方法中，监管模式S-mode的功耗管理软件将在最后一个 hart 变为空闲后，直接请求较高层次组的挂起状态。当一个 hart 变为空闲时，监管模式的功耗管理软件总是选择该 hart 的挂起状态，但仅在该 hart 是组内最后一个运行的 hart 时，才为较高层次组选择一个挂起状态。SBI 实现应满足以下条件：
   a. 永远不要选择与指定暂停状态不同的较高层次组的挂起状态
   b. 总是优先选择最近请求的较高层次组的挂起状态。

9.1 函数: HART 启动 (FID #0)

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

SBI实现以特定的寄存器值在指定的start_addr地址处启动执行目标hart的超级模式。具体的寄存器值如表18所述。

表18. HSM Hart 启动的寄存器状态

寄存器名称	寄存器值
satp	0
sstatus.SIE	0
a0	hartid
a1	opaque 参数
其他寄存器仍处于未定义状态。	

此调用是异步的，具体来说，sbi_hart_start()函数可以在目标hart开始执行之前返回，只要SBI实现能够确保返回的结果准确。如果SBI实现是在机器模式（M-mode）下执行的平台运行时固件，则必须在将控制权转移给S-mode模式软件之前配置PMP和其他M-mode状态。

hartid参数指定要启动的目标hart。

start_addr参数指向一个运行时指定的物理地址，该hart可以在S-mode模式下开始执行。

opaque参数是一个XLEN位的值，当hart在start_addr处开始执行时opaque参数被设置在a1寄存器中。

sbiret.error中可能返回的错误代码在下表19中显示。

表19. HSM Hart 启动错误

错误代码	描述
SBI_SUCCESS 成功	Hart 之前处于停止状态. 它将从start_addr开始执行
SBI_ERR_INVALID_ADDRESS 非法地址	start_addr 无效，原因可能如下： * 无效物理地址。 * 该地址被PMP禁止在S-mode下执行。
SBI_ERR_INVALID_PARAM 非法参数	Hartid 是无效的物理地址，因为其相应的hart不能以S-mode启动。
SBI_ERR_ALREADY_AVAILABLE 已存在	给定hartid已启动。
SBI_ERR_FAILED 失败	未知原因造成的启动失败。

9. 2函数: HART 禁用(FID #1)

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

要求SBI实现停止以S-mode模式执行调用hart，并将其所有权返回给SBI实现。在正常情况下，不希望此调用返回。sbi_hart_stop()必须在禁用S-mode模式中断时调用。

sbiret.error可能返回的错误代码如下表20所示。

表20. HSM Hart 停止错误

错误代码	描述
SBI_ERR_FAILED 失败	当前hart停止执行失败

9.3函数： HART 获取状态 （FID #2）

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

通过sbiret.value获取给定hart的当前状态（或HSM状态ID），或通过sbiret.error获取错误信息。

hartid参数指定需要查询状态的目标hart。

sbiret.value中可能返回的状态（或HSM状态ID）值在表17中进行了描述。

sbiret.error可能返回的错误代码如下表21所示。

表21. HSM Hart 获取状态错误

错误代码	描述
SBI_ERR_INVALID_PARAM 无效参数	给定的hartid无效

由于任何并发的sbi_hart_start()、sbi_hart_stop()或sbi_hart_suspend()调用，harts可能随时转换HSM状态，因此该函数的返回值可能不代表返回值验证时hart的实际状态。

9.4函数： HART 挂起(FID #3)

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

SBI实现将调用的hart置于由suspend_type参数指定的平台特定的挂起（或低功耗）状态。当收到中断或平台特定的硬件事件时，hart将自动退出挂起状态并恢复正常执行。

一个hart的平台特定挂起状态可以是保持性的或非保持性的。保持性挂起状态将保留所有特权模式下hart的寄存器和CSR值，而非保持性挂起状态将不保留hart的寄存器和CSR值。

从保持性挂起状态恢复是比较简单的，S-mode模式的软件将看到SBI挂起调用返回而无需任何失败。在保持性挂起期间，resume_addr参数未使用。

从非保持性挂起状态恢复相对更复杂，需要软件还原各种特权模式下的hart寄存器和CSR。从非保持性挂起状态恢复后，hart将跳转到由resume_addr指定的S−mode模式地址，具体寄存器的值在表22中描述。

表22. HSM Hart 恢复寄存器状态

寄存器名称	寄存器值
satp	0
sstatus.SIE	0
a0	hartid
a1	Opaque参数
其他寄存器保持未定义状态。	

suspend_type参数是32位宽，可能的值如表23所示。

表23. HSM Hart 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	平台特定非保持性挂起
> 0xFFFFFFFF	保留

resume_addr参数指向一个在非保持性挂起后，hart可以在S-mode模式下恢复执行的运行时指定的物理地址。

opaque参数是一个XLEN位的值，当hart在非保留挂起后在resume_addr处恢复执行时，会将该值设置在a1寄存器中。

sbiret.error中可能返回的错误代码如表24所示。

表24. HSM Hart 挂起错误

错误代码	描述
SBI_SUCCESS 成功	Hart已成功地从保持性挂起状态中恢复。
SBI_ERR_INVALID_PARAM 无效参数	suspend_type无效
SBI_ERR_NOT_SUPPORTED 不支持	suspend_type 有效但未实现。

错误代码	描述
SBI_ERR_INVALID_ADDRESS 无效地址	resume_addr无效，可能是由于以下原因： * 无效的物理地址。 * 该地址被PMP禁止在S-mode模式下运行。
SBI_ERR_FAILED 失败	挂起请求因未知原因而失败。

9.5函数列表

表25. HSM 函数列表

函数名	SBI 版本	FID	EID
sbi_hart_start 启动	0.2	0	0x48534D
sbi_hart_stop 停止	0.2	1	0x48534D
sbi_hart_get_status 获取状态	0.2	2	0x48534D
sbi_hart_suspend 挂起	0.3	3	0x48534D

章节10 系统复位扩展 (EID #0x53525354 "SRST")

系统复位扩展提供了一个函数，允许S-mode模式下的软件请求系统级的重启或关闭操作。术语"系统"指的是S-mode模式下的软件的视角，底层的SBI实现可以是机器模式M-mode固件或虚拟化管理程序。

10.1函数: 系统复位 (FID #0)

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

根据提供的类型reset_type和原因reset_reason重置系统。这是一个同步调用，如果成功将不会返回。

reset_type参数的宽度为32位，其可能的取值在下面的表26中列出。

表26. SRST 系统复位类型

值	描述
0x00000000	关机
0x00000001	冷启动
0x00000002	热启动
0x00000003 - 0xEFFFFFFF	保留以便未来使用
0xF0000000 - 0xFFFFFFFF	供应商或平台特定的复位类型
> 0xFFFFFFFF	保留

reset_reason 是一个可选参数，表示系统复位的原因。该参数是32位宽，可能的取值如下所示：

表27. SRST 系统复位原因

值	描述
0x00000000	没有原因
0x00000001	系统失败
0x00000002 - 0xDFFFFFFF	保留以便未来使用
0xE0000000 - 0xEFFFFFFF	SBI实现特定的复位原因
0xF0000000 - 0xFFFFFFFF	供应商或平台特定的复位类型
> 0xFFFFFFFF	保留

当S−mode下的软件在本地运行时，SBI实现属于机器模式下的固件。在这种情况下，关机等同于整个系统的物理断电，冷启动等同于整个系统的物理断电循环（并重新上电）。此外，热重启等同于对主处理器和系统的一部分进行断电循环，而不是整个系统。例如，在具有BMC（主板管理控制器）的服务器级系统上，热重启不会对BMC进行断电循环，而冷重启则肯定会对BMC进行断电循环。

当S−mode模式下的软件在虚拟机内运行时，SBI实现是一个虚拟化管理器。关机、冷重启和热重启在功能上与本机情况相同，但可能不会导致任何物理电源变化。

sbiret.error中可能返回的错误代码如表28所示。

表28. SRST 系统复位错误s

错误代码	描述
SBI_ERR_INVALID_PARAM 无效参数	reset_type 或 reset_reason 无效
SBI_ERR_NOT_SUPPORTED 不支持	reset_type 有效但未被设定
SBI_ERR_FAILED 失败	未知原因的复位请求失败

10.2函数列表

表29. SRST 函数列表

Function Name	SBI Version	FID	EID
sbi_system_reset	0.3	0	0x53525354

章节11 性能监控单元扩展 (EID #0x504D55 "PMU")

RISC-V硬件性能计数器（如mcycle、minstret和mhpmcounterX CSRs）可以以只读方式从监管者模式S-mode使用cycle、instret和hpmcounterX CSRs进行访问。SBI性能监控单元（PMU）扩展是一个接口，供监管者模式S-mode使用，以便借助机器模式（或虚拟化模式）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从机器模式使用mcountinhibit和mhpmeventX CSRs启动、停止或配置。因此，如果RISC-V平台未实现mhpmcounterX CSR，机器模式的SBI实现可能选择禁止SBI PMU扩展。

一般情况下，RISC-V平台支持使用有限数量的硬件性能计数器（最多64位）来监控各种硬件事件。此外，SBI实现还可以提供固件性能计数器，用于监控固件事件，例如不对齐的加载/存储指令数量、RFENCE数量、IPI数量等。固件计数器始终为64位。

SBI PMU扩展提供以下功能：

1. 监管者模式软件发现和配置每个HART的硬件/固件计数器的接口
2. 具有典型perf兼容接口的硬件/固件性能计数器和事件
3. 完全访问微体系结构的原始事件编码

为了定义SBI PMU扩展调用，我们首先定义了重要的实体counter_idx、event_idx和event_data。counter_idx是分配给每个硬件/固件计数器的逻辑编号。event_idx表示硬件（或固件）事件，而event_idx为64位宽，表示硬件（或固件）事件的额外配置（或参数）。

event_idx是一个20位宽的数字，编码如下：

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

11.1. 事件:硬件通用事件 (Type #0)

event_idx.type（即事件类型）对于所有硬件通用事件应为0x0，并且每个硬件通用事件由一个唯一的event_idx.code（即事件代码）标识，如下所示的表格30中所述。

表30. PMU 硬件事件

通用事件名称	代码	描述
SBI_PMU_HW_NO_EVENT	0	未使用的事件，因为event_idx不能为零

General Event Name	Code	Description
SBI_PMU_HW_CPU_CYCLES	1	每个CPU周期的事件
SBI_PMU_HW_INSTRUCTIONS	2	每个完成的指令的事件
SBI_PMU_HW_CACHE_REFERENCES	3	缓存命中的事件
SBI_PMU_HW_CACHE_MISSES	4	缓存未命中的事件
SBI_PMU_HW_BRANCH_INSTRUCTIONS	5	分支指令的事件
SBI_PMU_HW_BRANCH_MISSES	6	分支预测错误的事件
SBI_PMU_HW_BUS_CYCLES	7	每个BUS周期的事件
SBI_PMU_HW_STALLED_CYCLES_FRONTEND	8	微架构前端阻塞周期的事件
SBI_PMU_HW_STALLED_CYCLES_BACKEND	9	微架构后端阻塞周期的事件
SBI_PMU_HW_REF_CPU_CYCLES	10	每个参考CPU周期的事件

注意：对于硬件通用事件，event_data（即事件数据）未使用，event_data的所有非零值都保留供将来使用。

注意：当RISC-V平台使用WFI指令进入WAIT状态或使用SBI HSM HART挂起调用进入平台特定的挂起状态时，CPU时钟可能会停止。

注意：SBI_PMU_HW_CPU_CYCLES事件通过cycle CSR计数CPU时钟周期。这些周期可能是可变频率的周期，在CPU时钟停止时不计数。

注意：SBI_PMU_HW_REF_CPU_CYCLES在CPU时钟未停止时计数固定频率的时钟周期。计数的固定频率可以是例如时间CSR计数的频率。

注意：SBI_PMU_HW_BUS_CYCLES计数固定频率的时钟周期。计数的固定频率可以是例如时间CSR计数的频率，或者可以是HART（及其私有缓存）与系统其他部分之间的时钟边界的频率。

11.2. 事件:硬件缓存事件 (Type #1)

对于所有硬件缓存事件，event_idx.type（即事件类型）应为0x1，并且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
```

下表显示了以下标识符可能的值：event_idx.code.cache_id（即缓存事件ID），event_idx.code.op_id（即缓存操作ID）和event_idx.code.result_id（即缓存结果ID）。

表31. 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	最后一级缓存事件
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32. PMU 缓存操作ID

缓存操作名称	操作ID	描述
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33. PMU 缓存操作结果ID

缓存结果名称	结果ID	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

注意：对于硬件缓存事件， event_data（即事件数据）未使用，所有非零值的事件数据均保留用于将来使用。

11.3. 事件: 硬件原始事件 (Type #2)

硬件原始事件类型的event_idx.type (i.e. event type)值应为0x2， 而event_idx.code(i.e. event code)值应为零。

对于具有 32 位宽度的 mhpmeventX CSRs 的 RISC-V 平台，event_data 配置（或参数）应具有要编程到mhpmeventX CSR 中的 32 位值。

对于具有 64 位宽度的mhpmeventX CSRs 的 RISC-V 平台，event_data配置（或参数）应具有要编程到mhpmeventX CSR 的低 48 位值，而 SBI 实现将确定要编程到mhpmeventX CSR 的高 16 位值。

注意：RISC-V 平台的硬件实现可能选择定义要写入mhpmeventX CSR 的硬件事件的预期值。对于硬件常规/缓存事件，为了简化起见，RISC-V 平台的硬件实现可能使用零扩展的 event_idx 作为预期值。

11.4. 事件: 固件事件 (Type #15)

所有固件事件的event_idx.type（即事件类型）应为0xf，并且每个固件事件都由唯一的event_idx.code（即事件代码）标识，其描述如下所示的 表34 中。

表34. PMU 固件事件

固件事件名称	编码	描述
SBI_PMU_FW_MISALIGNED_LOAD	0	对齐错误加载陷入事件
SBI_PMU_FW_MISALIGNED_STORE	1	对齐错误存储陷入事件
SBI_PMU_FW_ACCESS_LOAD	2	加载访问陷入事件
SBI_PMU_FW_ACCESS_STORE	3	存储访问陷入事件
SBI_PMU_FW_ILLEGAL_INSN	4	非法指令陷入事件
SBI_PMU_FW_SET_TIMER	5	设置定时器事件
SBI_PMU_FW_IPI_SENT	6	发送 IPI 给其他 HART 事件
SBI_PMU_FW_IPI_RECEIVED	7	接收来自其他 HART 的 IPI 事件
SBI_PMU_FW_FENCE_I_SENT	8	发送 FENCE.I 请求给其他 HART 事件
SBI_PMU_FW_FENCE_I_RECEIVED	9	接收来自其他 HART 的 FENCE.I 请求事件
SBI_PMU_FW_SFENCE_VMA_SENT	10	发送 SFENCE.VMA 请求给其他 HART 事件
SBI_PMU_FW_SFENCE_VMA_RECEIVED	11	接收来自其他 HART 的 SFENCE.VMA 请求事件
SBI_PMU_FW_SFENCE_VMA_ASID_SENT	12	发送带有 ASID 的 SFENCE.VMA 请求给其他 HART 事件
SBI_PMU_FW_SFENCE_VMA_ASID_RECEIVED	13	接收来自其他 HART 的带有 ASID 的 SFENCE.VMA 请求事件
SBI_PMU_FW_HFENCE_GVMA_SENT	14	发送 HFENCE.GVMA 请求给其他 HART 事件
SBI_PMU_FW_HFENCE_GVMA_RECEIVED	15	接收来自其他 HART 的 HFENCE.GVMA 请求事件
SBI_PMU_FW_HFENCE_GVMA_VMID_SENT	16	发送带有 VMID 的 HFENCE.GVMA 请求给其他 HART 事件
SBI_PMU_FW_HFENCE_GVMA_VMID_RECEIVED	17	接收来自其他 HART 的带有 VMID 的 HFENCE.GVMA 请求事件

SBI_PMU_FW_HFENCE_VVMA_SENT	18	发送 HFENCE.VVMA 请求给其他 HART 事件
SBI_PMU_FW_HFENCE_VVMA_RECEIVED	19	接收来自其他 HART 的 HFENCE.VVMA 请求事件
SBI_PMU_FW_HFENCE_VVMA_ASID_SENT	20	向其他 HART 发送带有 ASID 的 HFENCE.VVMA 请求事件
SBI_PMU_FW_HFENCE_VVMA_ASID_RECEIVED	21	R从其他 HART 接收带有 ASID 的 HFENCE.VVMA 请求事件

注意：对于固件事件，event_data（即事件数据）未使用，所有非零的event_data值都保留供将来使用。

11.5. 函数: 获取可用计数器的数量(FID #0)

```
struct sbiret sbi_pmu_num_counters()
```

返回可用计数器的数量到sbiret.value中，并始终在sbiret.error中返回SBI_SUCCESS。

11.6. 函数: 获取特定计数器的详细信息(FID #1)

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

获取有关指定计数器的详细信息，例如底层CSR编号、计数器的宽度、计数器的类型（硬件/固件）等。

此SBI调用返回的counter_info信息编码如下：

```
    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)
```

若 counter_info.type == 1，那么counter_info.csr 与counter_info.width 应该被忽略。

在sbiret.value中返回上述描述的counter_info。

sbiret.error中可能返回的错误代码如下表35所示。

表35. PMU 计数器获取信息错误

错误代码	描述
SBI_SUCCESS	成功读取counter_info
SBI_ERR_INVALID_PARAM	counter_idx指向无效的计数器。

11.7. 函数: 查找并配置匹配计数器 (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)
```

在一组计数器中查找并配置一个尚未启动（或已启用）且可以监视指定事件的计数器。其中，counter_idx_base和counter_idx_mask参数表示计数器集合，event_idx表示要监视的事件，event_data表示任何附加事件配置。

config_flags参数表示额外的计数器配置和过滤标志。config_flags参数的位定义如下所示：

表36. PMU计数器配置匹配标志

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	在计数器配置中清零（或置零）计数器值
SBI_PMU_CFG_FLAG_AUTO_START	2:2	配置匹配计数器后自动启动计数器
SBI_PMU_CFG_FLAG_SET_VUINH	3:3	VU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
RESERVED	8:(XLEN-1)	所有非零值保留供将来使用

注意：当在config_flags中设置了SBI_PMU_CFG_FLAG_SKIP_MATCH标志时，SBI实现将无条件地从由counter_idx_base和counter_idx_mask指定的计数器集合中选择第一个计数器。

注意：config_flags中的SBI_PMU_CFG_FLAG_AUTO_START标志对计数器值没有影响。

注意：config_flags[3:7]位是事件过滤提示，因此在安全性方面或由于底层RISC-V平台缺乏事件过滤支持时，SBI实现可能会忽略或覆盖这些提示。

成功时，在sbiret.value中返回counter_idx。

如果操作失败，在sbiret.error中可能返回的错误代码如下表37所示：

表 37. PMU计数器配置匹配错误

错误代码	描述
SBI_SUCCESS	计数器已成功找到并配置。
SBI_ERR_INVALID_PARAM	计数器集合中存在无效的计数器。
SBI_ERR_NOT_SUPPORTED	没有任何计数器能够监视指定的事件。

11.8. 函数: 启动一组计数器 (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)
```

启动或启用一组计数器，并设置指定的初始值。counter_idx_base和counter_idx_mask参数表示计数器集合，initial_value参数指定计数器的初始值。

start_flags参数的位定义如下表38所示：

表38. PMU 计数器启动标志

标志名称	位	描述
SBI_PMU_START_SET_INIT_VALUE	0:0	parameter根据initial_value参数设置计数器的值
RESERVED	1:(XLEN-1)	所有非零值保留供将来使用

注意：当start_flags中未设置SBI_PMU_START_SET_INIT_VALUE时，计数器值不会被修改，并且事件计数将从当前计数器值开始。

sbiret.error中可能返回的错误代码在下表表39中列出。

表39. PMU 计数器启动错误

错误代码	描述
SBI_SUCCESS	计数器启动成功

错误代码	描述
SBI_ERR_INVALID_PARAM	参数中指定的一些计数器无效。
SBI_ERR_ALREADY_STARTED	参数中指定的一些计数器已被启动。

11.9. 函数: 停止或禁用一组计数器(FID #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HART上的一组计数器。counter_idx_base和counter_idx_mask参数表示计数器的集合。stop_flags参数的位定义如下表40所示。

表40. PMU 计数器停止禁用标志

标志名称	位	描述
SBI_PMU_STOP_FLAG_RESET	0:0	重置计数器到事件的映射关系。
RESERVED	1:(XLEN-1)	所有非零值都保留供将来使用。

返回在sbiret.error中可能出现的错误代码如下表41所示。

表41. PMU 计数器停止禁用错误

错误代码	描述
SBI_SUCCESS	计数器禁用成功。
SBI_ERR_INVALID_PARAM	指定的某些计数器无效。
SBI_ERR_ALREADY_STOPPED	指定的某些计数器已经停止。

11.10. 函数: 读取固件计数器 (FID #5)

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

在sbiret.value中提供固件计数器的当前值。

sbiret.error中返回的可能错误代码如下表42所示。

表42. PMU 读取固件计数错误

错误代码	描述
SBI_SUCCESS	成功读取固件计数器的值。
SBI_ERR_INVALID_PARAM	counter_idx 指向一个硬件计数器或无效的计数器。

11.11. 函数列表

表43. PMU 函数列表

函数名	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

章节 12 SBI实验扩展空间 (EIDs #0x08000000 – #0x08FFFFFF)

未安排。

章节13. SBI供应商特定扩展空间（EIDs #0x09000000 – #0x09FFFFFF）

从 mvendorid开始的低位。

章节14. SBI固件特定扩展空间(EIDs #0x0A000000 – #0x0AFFFFFF)

低位是SBI实现ID。固件特定的SBI扩展适用于SBI实现。它提供了固件特定的SBI函数，这些函数在外部固件规范中定义。