Requirements List of RISC-V Supervisor Binary Interface Specification,

Document Version 2.0[*]

by

Mehmet Öner

Version 1.0

*`RISC-V Supervisor Binary Interface Specification, Document Version 2.0', Contributors 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>, 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>, 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>, RISC-V International, January 2024.
https://github.com/riscv-non-isa/riscv-sbi-doc/

About

The Document

In systems engineering approach, before doing anything with the design of the system under consideration, *requirements analysis* must be completed as one of the first tasks (if not the very first). Beginning with an itemized, atomic, classified and well defined list of requirements is essential. Because, following activities at various stages of development (like design coverage analysis, testing, verification, validation …) depend on the requirement specifications stated at the beginning.

RISC-V International provides the ISA (Instruction Set Architecture) and non-ISA requirement specifications for the RISC-V architecture (https://riscv.org/technical/specifications/). These documents in general are good written technical plain text documents. However, they lack some aspects of good requirement specification practices:

- Requirements are in free text form and not itemized: Itemized list of requirements enables requirement coverage in design, test, verification and validation phases.
- Text includes comments and information statements along with requirements: Statements must be clearly labeled and categorized.
- Some statements include more than one specifications: Each specification need to be isolated.
- Some specifications such as instruction definitions are distributed throughout the text: The distributed content need to be put together to have a complete the specification.

The aim of this document is providing an edited list of requirements for `RISC-V Supervisor Binary Interface Specification, Document Version 2.0', Contributors Abner Chang, Al Stone, Andrew Jones, Anup Patel, Atish Patra, Atish Patra, Bin Meng, Chris Williams, Conor Dooley, Daniel Schaefer, Esteban Blanc, hasheddan, Heinrich Schuchardt, Jeff Scheel, Jessica Clarke, john, Konrad Schwarz, Luo Jia / Zhouqi Jiang, Nick Kossifidis, Palmer Dabbelt, Paolo Bonzini, Sean Anderson, Stefano Stabellini, Sunil V L, Tsukasa OI, Yiting Wang, RISC-V International, January 2024. https://github.com/riscv-non-isa/riscv-sbi-doc/

It is licensed under the Creative Commons Attribution 4.0 International License (CC-BY 4.0). https://creativecommons.org/licenses/by/4.0/

In particular:

- Statements were itemized and given an ID number.
- Each itemized statement was referenced to the original document to provide traceability
- Itemized statements were categorized
- Complex statements were broken into simpler atomic requirement statements when needed.
- Distributed requirement information was put together to form complete specifications.

Special attention was given to preserve original statements, even when dividing complex statements into simpler atomic statements. But occasionally, some statements were re-written as to form a formal requirement statement.

The Editor (or the systems engineer)

After 34 years of my career, I retired from my regular job in 2023. Now, I do part time consulting services to interested parties, while I do work on projects that interest me more than a regular work.

In my career I dealt with very diverse fields of engineering: Academics, C/C++ desktop programming, embedded systems, analog circuit design, DSP algorithms, VLSI/FPGA design, underwater acoustics are to name few.

I've always enjoyed designing controllers and processors with generic HDL for VLSI or FPGA. So, as my personal project to work on, I decided to design RISC-V cores with different capabilities.

Having some defense sector background, I find systems engineering approach very useful. After reading RISC-V specification documents, I decided to take the initiative and edit the documents into customer requirements list format which is a tough, tedious and time consuming work.

In case someone else could find these documents useful, I share them on github:
https://github.com/vizionerco/RISC-V

Best regards,

Mehmet Öner, Ph.D.

www.linkedin.com/in/mehmet-oner-00453733/

Table of Contents

Definitions

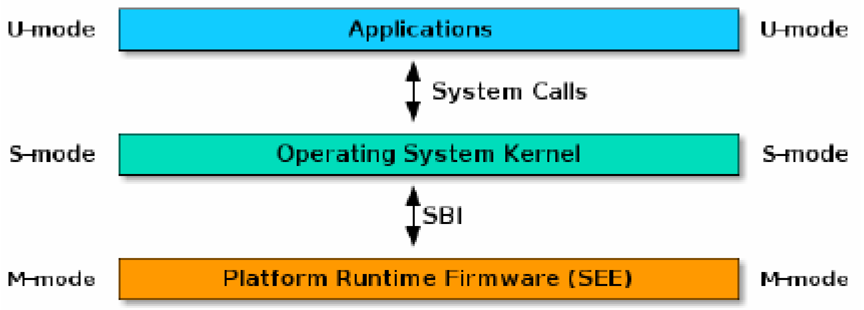
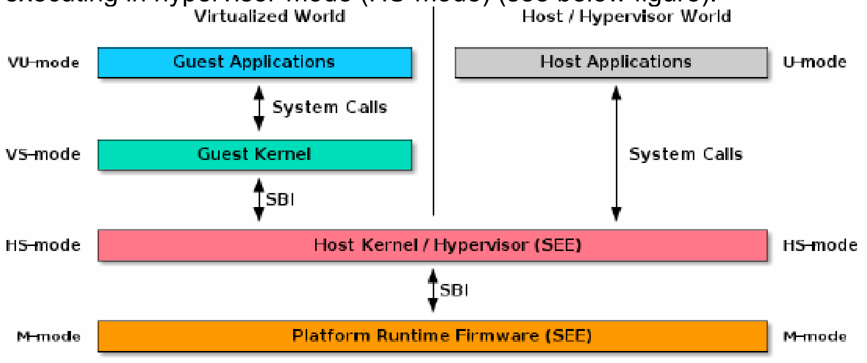
Table 1: Requirement Types

TYPE	NAME	EXPLANATION
H	Heading	Headings in the original document. Headings are included to provide context to the subsequent requirements
I	Information	These are statements that explain some aspects of the subject, but actually do not specify any requirement. For these types, the statement(s) in the text is given as is.
C	Comment	These statements are original comment statements in the RISC_V documentation which are explained as: "Commentary on our design decisions is formatted as in this paragraph. This non-normative text can be skipped if the reader is only interested in the specification itself." For these types, the statement(s) in the text is given as is.
R	Requirement	These are statements that specify a specific need to be fulfilled. For these types, the statement(s) in the text is given as is where possible. In some cases, information is collected from different tables and figures to form a complete specification. In some cases, context is added in parenthesis to make the requirement self explanatory. In some cases, the statements are broken into single statements requirement statements.
O	Optional Requirement	These are tatements that specify a property that is not mandatory to implement. However if it is chosen to fulfill, it should obey this requirement. Inclusion of the text is the same as R/Requirement type.
T	Tentative Requirement	These are requirements but are not frozen yet by the RISC-V committee. Inclusion of the text is the same as R/Requirement type.

Table 2: Abbreviations & Definitions

SHORT	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
PMA	Physical memory attributes
PMU	Performance Monitoring Unit
SBI	Supervisor Binary Interface
SEE	Supervisor Execution Environment.
VMID	Virtual Machine Identifier
XLEN	Width of an integer register in bits of an ISA

CHAPTER 1 Introduction

ID	REFERENCE	TYPE	DEFINITION
RVS.1.1	1.0 (p.7)	H	Introduction
RVS.1.2	1.0 (p.7)	I	This specification describes the RISC-V Supervisor Binary Interface, known from here on as SBI.
RVS.1.3	1.0 (p.7)	I	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.
RVS.1.4	1.0 (p.7)	I	The design of the SBI follows the general RISC-V philosophy of having a small core along with a set of optional modular extensions.
RVS.1.5	1.0 (p.7)	I	An SBI extension defines a set of SBI functions which provides a particular functionality to supervisor-mode software.
RVS.1.6	1.0 (p.7)	R	SBI extensions as a whole are optional and cannot be partially implemented unless an SBI extension defines a mechanism to discover implemented SBI functions.
RVS.1.7	1.0 (p.7)	R	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.
RVS.1.8	1.0 (p.7)	I	The higher privilege software providing SBI interface to the supervisor-mode software is referred as an SBI implementation or Supervisor Execution Environment (SEE).
RVS.1.9	1.0 (p.7) Figure 1	I	An SBI implementation (or SEE) can be platform runtime firmware executing in machine-mode (M-mode) (see below figure)  U-mode Applications U-mode System Calls S-mode Operating System Kernel S-mode SBI M-mode Platform Runtime Firmware (SEE) M-mode RISC-V System without H-extension
RVS.1.10	1.0 (p.7) Figure 2	I	(An SBI implementation (or SEE) can be ...) or it can be some hypervisor executing in hypervisor-mode (HS-mode) (see below figure).  Virtualized World Host / Hypervisor World VU-mode Guest Applications Host Applications U-mode System Calls VS-mode Guest Kernel System Calls SBI HS-mode Host Kernel / Hypervisor (SEE) HS-mode SBI M-mode Platform Runtime Firmware (SEE) M-mode RISC-V System with H-extension

ID	REFERENCE	TYPE	DEFINITION
RVS.1.11	1.0 (p.8)	I	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.
RVS.1.12	1.0 (p.8)	I	In the case that the implementation is a hypervisor, virtual harts represent the VS-mode guest contexts.
RVS.1.13	1.0 (p.8)	I	The SBI specification doesn’t specify any method for hardware discovery.
RVS.1.14	1.0 (p.8)	R	The supervisor software must rely on the other industry standard hardware discovery methods (i.e. Device Tree or ACPI) for that.

CHAPTER 3 Binary Encoding

ID	REFERENCE	TYPE	DEFINITION
RVS.3.1	3.0 (p.10)	H	Binary Encoding
RVS.3.2	3.0 (p.10)	I	All SBI functions share a single binary encoding, which facilitates the mixing of SBI extensions.

RVS.3.3	3.0 (p.10)	R	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.
			• 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;`
			`      long value;`
			`};`

RVS.3.4	3.0 (p.10)	R	In the name of compatibility, SBI extension IDs (EIDs) and SBI function IDs (FIDs) are encoded as signed 32-bit integers.
RVS.3.5	3.0 (p.10)	R	When passed in registers these follow the standard above calling convention rules.
RVS.3.6	3.0 (p.10) Table 1	R	The table below provides a list of Standard SBI error codes.

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

RVS.3.7	3.0 (p.10)	R	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.
RVS.3.8	3.0 (p.10)	R	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.
RVS.3.9	3.0 (p.10, p.11)	R	In case the data is defined as 32bit wide, higher privilege software must ensure that it only uses 32 bit data.
RVS.3.10	3.1 (p.11)	H	Hart list parameter
RVS.3.11	3.1 (p.11)	R	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.

ID	REFERENCE	TYPE	DEFINITION
RVS.3.12	3.1 (p.11)	R	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.
RVS.3.13	3.1 (p.11)	R	In a single SBI function call, the maximum number of harts that can be set is always XLEN.
RVS.3.14	3.1 (p.11)	R	If a lower privilege mode needs to pass information about more than XLEN harts, it must invoke the SBI function multiple times.
RVS.3.15	3.1 (p.11)	R	**hart_mask_base** can be set to -1 to indicate that **hart_mask** shall be ignored and all available harts must be considered.
RVS.3.16	3.1 (p.11) Table 2	R	Any SBI function taking hart mask arguments may return the error values listed in the table below which are in addition to function specific error values.

Error Code	Description
SBI_ERR_INVALID_PARAM	Either **hart_mask_base**, or at least one hartid from **hart_mask**, is not valid, i.e. either the hartid is not enabled by the platform or is not available to the supervisor.

ID	REFERENCE	TYPE	DEFINITION
RVS.3.17	3.2 (p.11)	H	Shared memory physical address range parameter
RVS.3.18	3.2 (p.11)	R	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 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. • The data in the shared memory MUST follow little-endian byte ordering.
RVS.3.19	3.2 (p.11)	C	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.
RVS.3.20	3.2 (p.11, p.12)	O	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.

CHAPTER 4 Base Extension (EID #0x10)

ID	REFERENCE	TYPE	DEFINITION
RVS.4.1	4.0 (p.13)	H	Base Extension (EID #0x10)
RVS.4.2	4.0 (p.13)	I	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.
RVS.4.3	4.0 (p.13)	R	All functions in the base extension must be supported by all SBI implementations, so there are no error returns defined.
RVS.4.4	4.1 (p.13)	H	Function: Get SBI specification version (FID #0)
RVS.4.5	4.1 (p.13)	I	`struct sbiret sbi_get_spec_version(void);`
RVS.4.6	4.1 (p.13)	R	Returns the current SBI specification version.
RVS.4.7	4.1 (p.13)	R	This function must always succeed.
RVS.4.8	4.1 (p.13)	R	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.
RVS.4.9	4.2 (p.13)	H	Function: Get SBI implementation ID (FID #1)
RVS.4.10	4.2 (p.13)	R	`struct sbiret sbi_get_impl_id(void);`
RVS.4.11	4.2 (p.13)	R	Returns the current SBI implementation ID, which is different for every SBI implementation.
RVS.4.12	4.2 (p.13)	I	It is intended that this implementation ID allows software to probe for SBI implementation quirks.
RVS.4.13	4.3 (p.13)	H	Function: Get SBI implementation version (FID #2)
RVS.4.14	4.3 (p.13)	R	`struct sbiret sbi_get_impl_version(void);`
RVS.4.15	4.3 (p.13)	R	Returns the current SBI implementation version.
RVS.4.16	4.3 (p.13)	R	The encoding of this version number is specific to the SBI implementation.
RVS.4.17	4.4 (p.13)	H	Function: Probe SBI extension (FID #3)
RVS.4.18	4.4 (p.13)	R	`struct sbiret sbi_probe_extension(long extension_id);`
RVS.4.19	4.4 (p.13)	R	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.
RVS.4.20	4.5 (p.13)	H	Function: Get machine vendor ID (FID #4)
RVS.4.21	4.5 (p.13)	R	`struct sbiret sbi_get_mvendorid(void);`
RVS.4.22	4.5 (p.14)	R	Return a value that is legal for the **mvendorid** CSR and 0 is always a legal value for this CSR.
RVS.4.23	4.6 (p.14)	H	Function: Get machine architecture ID (FID #5)
RVS.4.24	4.6 (p.14)	R	`struct sbiret sbi_get_marchid(void);`
RVS.4.25	4.6 (p.14)	R	Return a value that is legal for the **marchid** CSR and 0 is always a legal value for this CSR.
RVS.4.26	4.7 (p.14)	H	Function: Get machine implementation ID (FID #6)
RVS.4.27	4.7 (p.14)	R	`struct sbiret sbi_get_mimpid(void);`

ID	**REFERENCE**	**TYPE**	**DEFINITION**
RVS.4.28	4.7 (p.14)	R	Return a value that is legal for the mimpid CSR and 0 is always a legal value for this CSR.
RVS.4.29	4.8 (p.14)	H	Function Listing
RVS.4.30	4.8 (p.14) Table 3	R	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

RVS.4.31	4.9 (p.14)	H	SBI Implementation IDs
RVS.4.32	4.9 (p.14, p.15) Table 4	R	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

CHAPTER 5 Legacy Extensions (EIDs #0x00 - #0x0F)

ID	REFERENCE	TYPE	DEFINITION
RVS.5.1	5.0 (p.16)	H	Legacy Extensions (EIDs #0x00 - #0x0F)
RVS.5.2	5.0 (p.16)	R	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.
RVS.5.3	5.0 (p.16)	R	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.
RVS.5.4	5.0 (p.16)	I	The legacy SBI extensions is deprecated in favor of the other extensions listed below.
RVS.5.5	5.1 (p.16)	H	Extension: Set Timer (EID #0x00)
RVS.5.6	5.1 (p.16)	R	`long sbi_set_timer(uint64_t stime_value)`
RVS.5.7	5.1 (p.16)	R	Programs the clock for next event after stime_value time. This function also clears the pending timer interrupt bit.
RVS.5.8	5.1 (p.16)	R	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.
RVS.5.9	5.1 (p.16)	R	This SBI call returns 0 upon success or an implementation specific negative error code.
RVS.5.10	5.2 (p.16)	H	Extension: Console Putchar (EID #0x01)
RVS.5.11	5.2 (p.16)	R	`long sbi_console_putchar(int ch)`
RVS.5.12	5.2 (p.16)	R	Write data present in ch to debug console.
RVS.5.13	5.2 (p.16)	R	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.
RVS.5.14	5.2 (p.16)	R	However, if the console doesn’t exist at all, then the character is thrown away.
RVS.5.15	5.2 (p.16)	R	This SBI call returns 0 upon success or an implementation specific negative error code.
RVS.5.16	5.3 (p.17)	H	Extension: Console Getchar (EID #0x02)
RVS.5.17	5.3 (p.17)	R	`long sbi_console_getchar(void)`
RVS.5.18	5.3 (p.17)	R	Read a byte from debug console.
RVS.5.19	5.3 (p.17)	R	The SBI call returns the byte on success, or -1 for failure.
RVS.5.20	5.4 (p.17)	H	Extension: Clear IPI (EID #0x03)
RVS.5.21	5.4 (p.17)	R	`long sbi_clear_ipi(void)`
RVS.5.22	5.4 (p.17)	R	Clears the pending IPIs if any. The IPI is cleared only in the hart for which this SBI call is invoked.

ID	REFERENCE	TYPE	DEFINITION
RVS.5.23	5.4 (p.17)	R	**sbi_clear_ipi()** is deprecated because S-mode code can clear **sip.SSIP** CSR bit directly.
RVS.5.24	5.4 (p.17)	R	This SBI call returns 0 if no IPI had been pending, or an implementation specific positive value if an IPI had been pending.
RVS.5.25	5.5 (p.17)	H	Extension: Send IPI (EID #0x04)
RVS.5.26	5.5 (p.17)	R	`long sbi_send_ipi(const unsigned long *hart_mask)`
RVS.5.27	5.5 (p.17)	R	Send an inter-processor interrupt to all the harts defined in hart_mask.
RVS.5.28	5.5 (p.17)	R	Interprocessor interrupts manifest at the receiving harts as Supervisor Software Interrupts.
RVS.5.29	5.5 (p.17)	R	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.
RVS.5.30	5.5 (p.17)	R	This SBI call returns 0 upon success or an implementation specific negative error code.
RVS.5.31	5.6 (p.17)	H	Extension: Remote FENCE.I (EID #0x05)

RVS.5.32	5.6 (p.17)	R	`long sbi_remote_fence_i(`
			`        const unsigned long *hart_mask)`

RVS.5.33	5.6 (p.17)	R	Instructs remote harts to execute **FENCE.I** instruction. The **hart_mask** is same as described in **sbi_send_ipi()**.
RVS.5.34	5.6 (p.17)	R	This SBI call returns 0 upon success or an implementation specific negative error code.
RVS.5.35	5.7 (p.17)	H	Extension: Remote SFENCE.VMA (EID #0x06)

RVS.5.36	5.7 (p.18)	R	`long sbi_remote_sfence_vma(`
			`    const unsigned long *hart_mask,`
			`    unsigned long start, unsigned long size)`

RVS.5.37	5.7 (p.18)	R	Instructs the remote harts to execute one or more **SFENCE.VMA** instructions, covering the range of virtual addresses between **start** and **start + size**.
RVS.5.38	5.7 (p.18)	R	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.
RVS.5.39	5.7 (p.18)	R	This SBI call returns 0 upon success or an implementation specific negative error code.
RVS.5.40	5.8 (p.18)	H	Extension: Remote SFENCE.VMA with ASID (EID #0x07)

RVS.5.41	5.8 (p.18)	R	`long sbi_remote_sfence_vma_asid(`
			`    const unsigned long *hart_mask,`
			`    unsigned long start, unsigned long size,`
			`    unsigned long asid)`

RVS.5.42	5.8 (p.18)	R	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**.
RVS.5.43	5.8 (p.18)	R	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.
RVS.5.44	5.8 (p.18)	R	This SBI call returns 0 upon success or an implementation specific negative error code.

ID	REFERENCE	TYPE	DEFINITION
RVS.5.45	5.9 (p.18)	H	Extension: System Shutdown (EID #0x08)
RVS.5.46	5.9 (p.18)	R	`void sbi_shutdown(void)`
RVS.5.47	5.9 (p.18)	R	Puts all the harts to shutdown state from supervisor point of view.
RVS.5.48	5.9 (p.18)	R	This SBI call doesn’t return irrespective whether it succeeds or fails.
RVS.5.49	5.10 (p.18)	H	Function Listing
RVS.5.50	5.10 (p.19) Table 5	R	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	

CHAPTER 6 Timer Extension (EID #0x54494D45 "TIME")

ID	REFERENCE	TYPE	DEFINITION
RVS.6.1	6.0 (p.20)	H	Timer Extension (EID #0x54494D45 "TIME")
RVS.6.2	6.0 (p.20)	I	This replaces legacy timer extension (EID #0x00).
RVS.6.3	6.0 (p.20)	R	It follows the new calling convention defined in v0.2.
RVS.6.4	6.1 (p.20)	H	Function: Set Timer (FID #0)
RVS.6.5	6.1 (p.20)	R	`struct sbiret sbi_set_timer(uint64_t stime_value)`
RVS.6.6	6.1 (p.20)	R	Programs the clock for next event after **stime_value** time.
RVS.6.7	6.1 (p.20)	R	**stime_value** is in absolute time.
RVS.6.8	6.1 (p.20)	R	This function must clear the pending timer interrupt bit as well.
RVS.6.9	6.1 (p.20)	R	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.
RVS.6.10	6.2 (p.20)	H	Function Listing
RVS.6.11	6.0 (p.20) Table 6	R	TIME Function List

Function Name	SBI Version	FID	EID
sbi_set_timer	0.2	0	0x54494D45

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

ID	REFERENCE	TYPE	DEFINITION
RVS.7.1	7.0 (p.21)	H	IPI Extension (EID #0x735049 "sPI:s-mode IPI")
RVS.7.2	7.0 (p.21)	I	This extension replaces the legacy extension (EID #0x04). The other IPI related legacy extension(0x3) is deprecated now.
RVS.7.3	7.0 (p.21)	I	All the functions in this extension follow the **hart_mask** as defined in the binary encoding section.
RVS.7.4	7.1 (p.21)	H	Function: Send IPI (FID #0)

RVS.7.5	7.1 (p.21)	R	`struct sbiret sbi_send_ipi(`
			`      unsigned long hart_mask,`
			`      unsigned long hart_mask_base)`

RVS.7.6	7.1 (p.21)	R	Send an inter-processor interrupt to all the harts defined in hart_mask.
RVS.7.7	7.1 (p.21)	R	Interprocessor interrupts manifest at the receiving harts as the supervisor software interrupts.
RVS.7.8	7.1 (p.21) Table 7	R	The possible error codes returned in **sbiret.error** are shown in the table below (IPI Send Errors).

Error Code	Description
SBI_SUCCESS	IPI was sent to all the targeted harts successfully.

ID	REFERENCE	TYPE	DEFINITION
RVS.7.9	7.2 (p.21)	H	Function Listing
RVS.7.10	7.2 (p.21) Table 8	R	IPI Function List

Function Name	SBI Version	FID	EID
sbi_send_ipi	0.2	0	0x735049

CHAPTER 8 RFENCE Extension (EID #0x52464E43 "RFNC")

ID	REFERENCE	TYPE	DEFINITION
RVS.8.1	8.0 (p.22)	H	RFENCE Extension (EID #0x52464E43 "RFNC")
RVS.8.2	8.0 (p.22)	I	This extension defines all remote fence related functions and replaces the legacy extensions (EIDs #0x05 - #0x07).
RVS.8.3	8.0 (p.22)	I	All the functions follow the **hart_mask** as defined in binary encoding section.
RVS.8.4	8.0 (p.22)	R	Any function which accepts a range of addresses (i.e. **start_addr** and **size**) must abide by the below constraints on range parameters.
RVS.8.5	8.0 (p.22)	R	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.
RVS.8.6	8.1 (p.22)	H	Function: Remote FENCE.I (FID #0)

RVS.8.7	8.1 (p.22)	R	`struct sbiret sbi_remote_fence_i(`
			`      unsigned long hart_mask,`
			`      unsigned long hart_mask_base)`

RVS.8.8	8.1 (p.22)	R	Instructs remote harts to execute FENCE.I instruction.
RVS.8.9	8.1 (p.22) Table 9	R	The possible error codes returned in **sbiret.error** are shown in the table below (RFENCE Remote FENCE.I Errors).

Error Code	Description
SBI_SUCCESS	IPI was sent to all the targeted harts successfully.

ID	REFERENCE	TYPE	DEFINITION
RVS.8.10	8.2 (p.22)	H	Function: Remote SFENCE.VMA (FID #1)

RVS.8.11	8.2 (p.22)	R	`struct sbiret sbi_remote_sfence_vma(`
			`      unsigned long hart_mask,`
			`      unsigned long hart_mask_base,`
			`      unsigned long start_addr,`
			`      unsigned long size)`

RVS.8.12	8.2 (p.22)	R	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**.
RVS.8.13	8.2 (p.22) Table 10	R	The possible error codes returned in **sbiret.error** are shown in the table below (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.

ID	REFERENCE	TYPE	DEFINITION
RVS.8.14	8.3 (p.23)	H	Function: Remote SFENCE.VMA with ASID (FID #2)

RVS.8.15	8.3 (p.23)	R	`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)`

ID	REFERENCE	TYPE	DEFINITION
RVS.8.16	8.3 (p.23)	R	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.
RVS.8.17	8.3 (p.23) Table 11	R	The possible error codes returned in **sbiret.error** are shown in the table below (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.

ID	REFERENCE	TYPE	DEFINITION
RVS.8.18	8.4 (p.23)	H	Function: Remote HFENCE.GVMA with VMID (FID #3)
RVS.8.19	8.4 (p.23)	R	

```
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)
```

ID	REFERENCE	TYPE	DEFINITION
RVS.8.20	8.4 (p.23)	R	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.
RVS.8.21	8.4 (p.23) Table 12	R	The possible error codes returned in **sbiret.error** are shown in the table below (RFENCE Remote HFENCE.GVMA with VMID 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.8.22	8.5 (p.24)	H	Function: Remote HFENCE.GVMA (FID #4)
RVS.8.23	8.5 (p.24)	R	

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

ID	REFERENCE	TYPE	DEFINITION
RVS.8.24	8.5 (p.24)	R	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.
RVS.8.25	8.5 (p.24)	R	This function call is only valid for harts implementing hypervisor extension.
RVS.8.26	8.5 (p.24) Table 13	R	The possible error codes returned in **sbiret.error** are shown in the table below (RFENCE Remote HFENCE.GVMA with VMID 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.8.27	8.6 (p.24)	H	Function: Remote HFENCE.VVMA with ASID (FID #5)
RVS.8.28	8.6 (p.24)	R	`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)`
RVS.8.29	8.6 (p.24)	R	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 **hgatp** CSR) of calling hart. This function call is only valid for harts implementing hypervisor extension.
RVS.8.30	8.6 (p.24) Table 14	R	The possible error codes returned in **sbiret.error** are shown in the table below (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.

ID	REFERENCE	TYPE	DEFINITION
RVS.8.31	8.7 (p.25)	H	Function: Remote HFENCE.VVMA (FID #6)
RVS.8.32	8.7 (p.25)	R	`struct sbiret sbi_remote_hfence_vvma( unsigned long hart_mask, unsigned long hart_mask_base, unsigned long start_addr, unsigned long size)`
RVS.8.33	8.7 (p.25)	R	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.
RVS.8.34	8.7 (p.25) Table 15	R	The possible error codes returned in **sbiret.error** are shown in the table below (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.

ID	REFERENCE	TYPE	DEFINITION
RVS.8.35	8.6 (p.25)	H	Function Listing

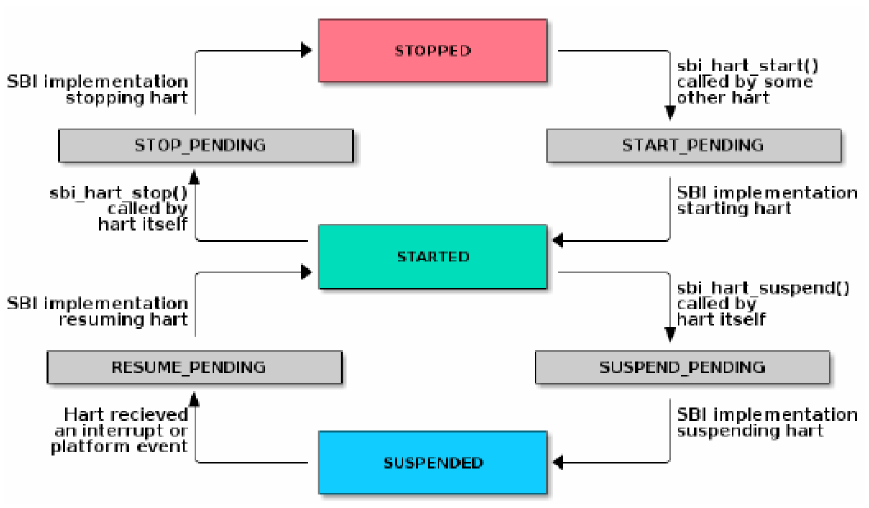
ID	REFERENCE	TYPE	DEFINITION
RVS.8.36	8.6 (p.25) Table 16	R	RFENCE Function List

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

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

ID	REFERENCE	TYPE	DEFINITION
RVS.9.1	9.0 (p.26)	H	Hart State Management Extension (EID #0x48534D "HSM")
RVS.9.2	9.0 (p.26)	I	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.
RVS.9.3	9.0 (p.26) Table 17	R	The table shown below describes all possible HSM states along with a unique HSM state id for each state (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.

ID	REFERENCE	TYPE	DEFINITION
RVS.9.4	9.0 (p.26) Figure 3	R	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  STOPPED SBI implementation stopping hart sbi_hart_start() called by some other hart STOP_PENDING START_PENDING sbi_hart_stop() called by hart itself SBI implementation starting hart STARTED SBI implementation resuming hart sbi_hart_suspend() called by hart itself RESUME_PENDING SUSPEND_PENDING Hart recieved an interrupt or platform event SBI implementation suspending hart SUSPENDED SBI HSM State Machine
RVS.9.5	9.0 (p.27)	I	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.
RVS.9.6	9.0 (p.27)	I	These platform specific low-power states of hierarchical topology groups can be represented as platform specific suspend states of a hart.
RVS.9.7	9.0 (p.27)	R	An SBI implementation can utilize the suspend states of higher topology groups using one of the following approaches:
RVS.9.8	9.0 (p.27)	O	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

ID	REFERENCE	TYPE	DEFINITION
RVS.9.9	9.0 (p.27)	O	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
RVS.9.10	9.1 (p.27)	H	Function: Hart start (FID #0)

RVS.9.11	9.1 (p.27, p.28)	R	`struct sbiret sbi_hart_start(unsigned long hartid,`
			`      unsigned long start_addr,`
			`      unsigned long opaque)`

RVS.9.12	9.1 (p.28) Table 18	R	Request the SBI implementation to start executing the target hart in supervisor-mode, at the address specified by **start_addr**, with the specific register values described in the table below.

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

ID	REFERENCE	TYPE	DEFINITION
RVS.9.13	9.1 (p.28)	C	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.
RVS.9.14	9.1 (p.28)	R	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.
RVS.9.15	9.1 (p.28)	R	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.
RVS.9.16	9.1 (p.28)	R	The **hartid** parameter specifies the target hart which is to be started.
RVS.9.17	9.1 (p.28)	R	The **start_addr** parameter points to a runtime-specified physical address, where the hart can start executing in supervisor-mode.
RVS.9.18	9.1 (p.28)	R	The **opaque** parameter is an XLEN-bit value which will be set in the a1 register when the hart starts executing at **start_addr**.

ID	REFERENCE	TYPE	DEFINITION
RVS.9.19	9.1 (p.28) Table 19	R	The possible error codes returned in **sbiret.error** are shown in the table below.

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.

ID	REFERENCE	TYPE	DEFINITION
RVS.9.20	9.2 (p.29)	H	Function: Hart stop (FID #1)
RVS.9.21	9.2 (p.29)	R	`struct sbiret sbi_hart_stop(void)`
RVS.9.22	9.2 (p.29)	R	Request the SBI implementation to stop executing the calling hart in supervisor-mode and return its ownership to the SBI implementation.
RVS.9.23	9.2 (p.29)	R	This call is not expected to return under normal conditions.
RVS.9.24	9.2 (p.29)	R	The **sbi_hart_stop()** must be called with supervisor-mode interrupts disabled.
RVS.9.25	9.2 (p.29) Table 20	R	The possible error codes returned in **sbiret.error** are shown in the table below. HSM Hart Stop Errors.

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

ID	REFERENCE	TYPE	DEFINITION
RVS.9.26	9.3 (p.29)	H	Function: Hart get status (FID #2)

RVS.9.27	9.3 (p.29)	R	`struct sbiret sbi_hart_get_status(`
			`unsigned long hartid)`

RVS.9.28	9.3 (p.29)	R	Get the current status (or HSM state id) of the given hart in **sbiret.value**, or an error through sbiret.error.
RVS.9.29	9.3 (p.29)	R	The **hartid** parameter specifies the target hart for which status is required.
RVS.9.30	9.3 (p.29)	R	The possible status (or HSM state id) values returned in **sbiret.value** are described in RVS.9.3.
RVS.9.31	9.3 (p.29) Table 21	R	The possible error codes returned in **sbiret.error** are shown in the table below. HSM Hart Get Status Errors

Error Code	Description
SBI_ERR_INVALID_PARAM	The given **hartid** is not valid.

ID	REFERENCE	TYPE	DEFINITION
RVS.9.32	9.3 (p.29)	I	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.
RVS.9.33	9.4 (p.30)	H	Function: Hart suspend (FID #3)

ID	REFERENCE	TYPE	DEFINITION
RVS.9.34	9.4 (p.30)	R	`struct sbiret sbi_hart_suspend( uint32_t suspend_type, unsigned long resume_addr, unsigned long opaque)`
RVS.9.35	9.4 (p.30)	R	Request the SBI implementation to put the calling hart in a platform specific suspend (or low power) state specified by the suspend_type parameter.
RVS.9.36	9.4 (p.30)	R	The hart will automatically come out of suspended state and resume normal execution when it receives an interrupt or platform specific hardware event.
RVS.9.37	9.4 (p.30)	I	The platform specific suspend states for a hart can be either retentive or non-retentive in nature.
RVS.9.38	9.4 (p.30)	R	A retentive suspend state will preserve hart register and CSR values for all privilege modes ...
RVS.9.39	9.4 (p.30)	R	… whereas a non-retentive suspend state will not preserve hart register and CSR values.
RVS.9.40	9.4 (p.30)	R	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.
RVS.9.41	9.4 (p.30) Table 22	R	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, the hart will jump to supervisor-mode at address specified by **resume_addr** with specific registers values described in the table below.

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

ID	REFERENCE	TYPE	DEFINITION
RVS.9.42	9.4 (p.30)	C	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.
RVS.9.43	9.4 (p.30) Table 23	R	The **suspend_type** parameter is 32 bits wide and the possible values are shown in the table below

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

ID	REFERENCE	TYPE	DEFINITION
RVS.9.44	9.4 (p.31)	R	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.

ID	REFERENCE	TYPE	DEFINITION
RVS.9.45	9.4 (p.31)	R	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.
RVS.9.46	9.4 (p.31) Table 24	R	The possible error codes returned in **sbiret.error** are shown in the table below. 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.9.47	9.5 (p.31)	H	Function Listing
RVS.9.48	9.5 (p.31) Table 25	R	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

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

ID	REFERENCE	TYPE	DEFINITION
RVS.10.1	10.0 (p.32)	H	System Reset Extension (EID #0x53525354 "SRST")
RVS.10.2	10.0 (p.32)	I	The System Reset Extension provides a function that allow the supervisor software to request system-level reboot or shutdown.
RVS.10.3	10.0 (p.32)	I	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.
RVS.10.4	10.1 (p.32)	H	Function: System reset (FID #0)
RVS.10.5	10.1 (p.32)	R	`struct sbiret sbi_system_reset( uint32_t reset_type, uint32_t reset_reason)`
RVS.10.6	10.1 (p.32)	R	Reset the system based on provided **reset_type** and **reset_reason**.
RVS.10.7	10.1 (p.32)	R	This is a synchronous call and does not return if it succeeds.
RVS.10.8	10.1 (p.32) Table 26	R	The **reset_type** parameter is 32 bits wide and it’s possible values are shown in the table below. 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

ID	REFERENCE	TYPE	DEFINITION
RVS.10.9	10.1 (p.32) Table 27	R	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 below

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

ID	REFERENCE	TYPE	DEFINITION
RVS.10.10	10.1 (p.32)	R	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.
RVS.10.11	10.1 (p.33)	R	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.

ID	REFERENCE	TYPE	DEFINITION
RVS.10.12	10.1 (p.33) Table 28	R	The possible error codes returned in **sbiret.error** are shown in the table below. SRST System Reset Errors

Error Code	Description
SBI_ERR_INVALID_PARAM	At least one of r**eset_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.

ID	REFERENCE	TYPE	DEFINITION
RVS.10.13	10.2 (p.33)	H	Function Listing
RVS.10.14	10.2 (p.33) Table 29	R	SRST Function List

Function Name	SBI Version	FID	EID
sbi_system_reset	0.3	0	0x53525354

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

ID	REFERENCE	TYPE	DEFINITION
RVS.11.1	11.0 (p.34)	H	Performance Monitoring Unit Extension (EID #0x504D55 "PMU")
RVS.11.2	11.0 (p.34)	I	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.
RVS.11.3	11.0 (p.34)	I	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).
RVS.11.4	11.0 (p.34)	R	These hardware performance counters can only be started, stopped, or configured from machine-mode using **mcountinhibit** and **mhpmeventX** CSRs. Due to this, a machinemode SBI implementation may choose to disallow SBI PMU extension if **mcountinhibit** CSR is not implemented by the RISC-V platform.
RVS.11.5	11.0 (p.34)	I	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.
RVS.11.6	11.0 (p.34)	R	All firmware counters must have same number of bits and can be up to 64 bits wide.
RVS.11.7	11.0 (p.34)	I	The SBI PMU extension provides: 1. An interface for supervisor-mode software to discover and configure per-hart hardware/firmware counters 2. A typical `perf` compatible interface for hardware/firmware performance counters and events 3. Full access to microarchitecture's raw event encodings
RVS.11.8	11.0 (p.34)	R	To define SBI PMU extension calls, we first define important entities **counter_idx**, **event_idx**, and **event_data**.
RVS.11.9	11.0 (p.34)	R	The **counter_idx** is a logical number assigned to each hardware/firmware counter.
RVS.11.10	11.0 (p.34)	R	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.

RVS.11.11	11.0 (p.34)	R	The event_idx is a 20 bits wide number encoded as follows:
			`event_idx[19:16] = type`
			`event_idx[15:0] = code`

RVS.11.12	11.1 (p.34)	H	Event: Hardware general events (Type #0)
RVS.11.13	11.1 (p.34)	R	The **event_idx.type** (i.e. event type) should be **0x0** for all hardware general events ...

ID	**REFERENCE**	**TYPE**	**DEFINITION**
RVS.11.14	11.1 (p.34) Table 30	R	… and each hardware general event is identified by an unique **event_idx.code** (i.e. event code) described in the table below.

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

RVS.11.15	11.1 (p.35)	R	The **event_data** (i.e. event data) is unused for hardware general events
RVS.11.16	11.1 (p.35)	R	… and all non-zero values of **event_data** are reserved for future use
RVS.11.17	11.1 (p.35)	C	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.
RVS.11.18	11.1 (p.35)	C	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.
RVS.11.19	11.1 (p.35)	C	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.
RVS.11.20	11.1 (p.35)	C	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.
RVS.11.21	11.2 (p.35)	H	Event: Hardware cache events (Type #1)
RVS.11.22	11.2 (p.35, p.36)	R	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`

ID	REFERENCE	TYPE	DEFINITION
RVS.11.23	11.2 (p.36) Table 31	R	Below tables show possible values of: **event_idx.code.cache_id** (i.e. cache event id). 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

ID	REFERENCE	TYPE	DEFINITION
RVS.11.24	11.2 (p.36) Table 32	R	Below table show possible values of: **event_idx.code.op_id** (i.e. cache operation id). 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

ID	REFERENCE	TYPE	DEFINITION
RVS.11.25	11.2 (p.36) Table 33	R	Below table show possible values of: **event_idx.code.result_id** (i.e. cache result id). 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

ID	REFERENCE	TYPE	DEFINITION
RVS.11.26	11.2 (p.36)	R	The **event_data** (i.e. event data) is unused for hardware cache events
RVS.11.27	11.2 (p.36)	R	... and all non-zero values of **event_data** are reserved for future use.
RVS.11.28	11.3 (p.36)	H	Event: Hardware raw events (Type #2)
RVS.11.29	11.3 (p.36)	R	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.
RVS.11.30	11.3 (p.36)	R	On RISC-V platform 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.
RVS.11.31	11.3 (p.36, p.37)	R	On RISC-V platform with 64 bits wide **mhpmeventX** CSRs, the **event_data** configuration (or parameter) should have the 48-bit value to 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.
RVS.11.32	11.3 (p.37)	C	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
RVS.11.33	11.4 (p.37)	H	Event: Firmware events (Type #15)
RVS.11.34	11.4 (p.37)	R	The **event_idx.type** (i.e. event type) should be **0xf** for all firmware events …

ID	REFERENCE	TYPE	DEFINITION
RVS.11.35	11.4 (p.37) Table 34	R	… and each firmware event is identified by an unique **event_idx.code** (i.e. event code) described in the table below. 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
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.

ID	REFERENCE	TYPE	DEFINITION
RVS.11.36	11.4 (p.38)	R	For all firmware events except SBI_PMU_FW_PLATFORM, the **event_data** configuration (or parameter) is unused ...

ID	REFERENCE	TYPE	DEFINITION
RVS.11.37	11.4 (p.38)	R	… and all non-zero values of **event_data** are reserved for future use.
RVS.11.38	11.5 (p.38)	H	Function: Get number of counters (FID #0)
RVS.11.39	11.5 (p.38)	R	`struct sbiret sbi_pmu_num_counters()`
RVS.11.40	11.5 (p.38)	R	Returns the number of counters (both hardware and firmware) in **sbiret.value** and always returns SBI_SUCCESS in sbiret.error.
RVS.11.41	11.6 (p.38)	H	Function: Get details of a counter (FID #1)
RVS.11.42	11.6 (p.38)	R	`struct sbiret sbi_pmu_counter_get_info( unsigned long counter_idx)`
RVS.11.43	11.6 (p.38)	R	Get details about the specified counter such as underlying CSR number, width of the counter, type of counter hardware/firmware, etc.
RVS.11.44	11.6 (p.39)	R	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)`
RVS.11.45	11.6 (p.39)	R	If **counter_info.type == 1** then **counter_info.csr** and **counter_info.width** should be ignored.
RVS.11.46	11.6 (p.39)	R	Returns the **counter_info** described above in **sbiret.value**.
RVS.11.47	11.6 (p.39) Table 35	R	The possible error codes returned in sbiret.error are shown in the table below. PMU Counter Get Info Errors

Error Code	Description
SBI_SUCCESS	**counter_info** read successfully.
SBI_ERR_INVALID_PARAM	**counter_idx** points to an invalid counter.

ID	REFERENCE	TYPE	DEFINITION
RVS.11.48	11.7 (p.39)	H	Function: Find and configure a matching counter (FID #2)
RVS.11.49	11.7 (p.39)	R	`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)`
RVS.11.50	11.7 (p.39)	R	Find and configure a counter from a set of counters which is not started (or enabled) and can monitor the specified event.
RVS.11.51	11.7 (p.39)	R	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.

ID	REFERENCE	TYPE	DEFINITION
RVS.11.52	11.7 (p.39) Table 36	R	The **config_flags** parameter represents additional counter configuration and filter flags. The bit definitions of the **config_flags** parameter are shown in the table below. 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)	All non-zero values are reserved for future use

ID	REFERENCE	TYPE	DEFINITION
RVS.11.53	11.7 (p.40)	C	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**.
RVS.11.54	11.7 (p.40)	C	The SBI_PMU_CFG_FLAG_AUTO_START flag in **config_flags** has no impact on the counter value.
RVS.11.55	11.7 (p.40)	C	The **config_flags[3:7]** bits are event filtering hints so these can be ignored or overridden by the SBI implementation for security concerns or due to lack of event filtering support in the underlying RISC-V platform
RVS.11.56	11.7 (p.40)	R	Returns the **counter_idx** in **sbiret.value** upon success.
RVS.11.57	11.7 (p.40) Table 37	R	In case of failure, the possible error codes returned in **sbiret.error** are shown in the table below. 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.
SBI_ERR_NOT_SUPPORTED	none of the counters can monitor the specified event.

ID	REFERENCE	TYPE	DEFINITION
RVS.11.58	11.8 (p.41)	H	Function: Start a set of counters (FID #3)
RVS.11.59	11.8 (p.41)	R	`struct sbiret sbi_pmu_counter_start(`

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

ID	REFERENCE	TYPE	DEFINITION
RVS.11.60	11.8 (p.41)	R	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.
RVS.11.61	11.8 (p.41) Table 38	R	The bit definitions of the start_flags parameter are shown in the table below. 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

ID	REFERENCE	TYPE	DEFINITION
RVS.11.62	11.8 (p.41)	C	When SBI_PMU_START_SET_INIT_VALUE is not set in start_flags, the counter value will not be modified and event counting will start from current counter value.
RVS.11.63	11.8 (p.41)	R	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.
RVS.11.64	11.8 (p.41)	R	The SBI implementation must initialize all the given valid counters (to be started) from the value set in the shared snapshot memory.
RVS.11.65	11.8 (p.41)	C	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.
RVS.11.66	11.8 (p.41) Table 39	R	The possible error codes returned in **sbiret.error** are shown in the table below. PMU Counter Start Errors

Error Code	Description
SBI_SUCCESS	counter started successfully
SBI_ERR_INVALID_PARAM	set of counters has at least one invalid counter.
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.

ID	REFERENCE	TYPE	DEFINITION
RVS.11.67	11.9 (p.42)	H	Function: Stop a set of counters (FID #4)
RVS.11.68	11.9 (p.42)	R	see below
RVS.11.69	11.9 (p.42)	R	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.

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

ID	REFERENCE	TYPE	DEFINITION
RVS.11.70	11.9 (p.42) Table 40	R	The bit definitions of the stop_flags parameter are shown in the table below. 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

ID	REFERENCE	TYPE	DEFINITION
RVS.11.71	11.9 (p.42)	R	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.
RVS.11.72	11.9 (p.42)	R	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.
RVS.11.73	11.9 (p.42)	R	The values corresponding to all other counters must not be modified.
RVS.11.74	11.9 (p.42)	R	The SBI implementation must additionally update the overflown counter bitmap in the shared memory.
RVS.11.75	11.9 (p.42) Table 41	R	The possible error codes returned in **sbiret.error** are shown in the table below. 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.
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.

ID	REFERENCE	TYPE	DEFINITION
RVS.11.76	11.10 (p.43)	H	Function: Read a firmware counter (FID #5)

RVS.11.77	11.10 (p.43)	R	`struct sbiret sbi_pmu_counter_fw_read(`
			`      unsigned long counter_idx)`

RVS.11.78	11.10 (p.43)	R	Provide the current firmware counter value in **sbiret.value**.
RVS.11.79	11.10 (p.43)	R	On RV32 systems, the **sbiret.value** will only contain the lower 32 bits of the current firmware counter value.
RVS.11.80	11.10 (p.43) Table 42	R	The possible error codes returned in **sbiret.error** are shown in the table below. 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.11.81	11.11 (p.43)	H	Function: Read a firmware counter high bits (FID #6)

RVS.11.82	11.11 (p.43)	R	`struct sbiret sbi_pmu_counter_fw_read_hi(`
			`            unsigned long counter_idx)`

ID	REFERENCE	TYPE	DEFINITION
RVS.11.83	11.11 (p.43)	R	Provide the upper 32 bits of the current firmware counter value in **sbiret.value**.
RVS.11.84	11.11 (p.43)	R	This function always returns zero in **sbiret.value** for RV64 (or higher) systems.
RVS.11.85	11.11 (p.43) Table 43	R	The possible error codes returned in **sbiret.error** are shown in the table below. 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.11.86	11.12 (p.43)	H	Function: Set PMU snapshot shared memory (FID #7)
RVS.11.87	11.12 (p.43)	R	`struct sbiret sbi_pmu_snapshot_set_shmem( unsigned long shmem_phys_lo, unsigned long shmem_phys_hi, unsigned long flags)`
RVS.11.88	11.12 (p.43)	R	Set and enable the PMU snapshot shared memory on the calling hart.
RVS.11.89	11.12 (p.43, p.44)	R	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.
RVS.11.90	11.12 (p.43)	R	The **shmem_phys_lo** MUST be 4096 bytes (i.e. page) aligned and the size of the snapshot shared memory must be 4096 bytes.
RVS.11.91	11.12 (p.44) Table 44	R	The layout of the snapshot shared memory is described in the table below. 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

ID	REFERENCE	TYPE	DEFINITION
RVS.11.92	11.12 (p.44)	R	If both **shmem_phys_lo** and **shmem_phys_hi** parameters are all-ones bitwise then the PMU snapshot shared memory is cleared and disabled.
RVS.11.93	11.12 (p.44)	R	The **flags** parameter is reserved for future use and must be zero.
RVS.11.94	11.12 (p.44)	R	This is an optional function and the SBI implementation may choose not to implement it.
RVS.11.95	11.12 (p.44)	R	Any future revisions to this structure should be made in a backward compatible manner and will be associated with an SBI version.
RVS.11.96	11.12 (p.44)	R	The logical counter indicies 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.11.97	11.12 (p.44)	R	This function should be invoked only once per hart at boot time.
RVS.11.98	11.12 (p.44)	R	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.
RVS.11.99	11.12 (p.44)	R	The SBI implementation has read only access when **sbi_pmu_counter_start** is invoked with the SBI_PMU_START_FLAG_INIT_SNAPSHOT flag set.
RVS.11.100	11.12 (p.44)	R	The SBI implementation must not access this memory any other time.
RVS.11.101	11.12 (p.44) Table 45	R	The possible error codes returned in **sbiret.error** are shown in the table below. 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.11.102	11.13 (p.45)	R	Function Listing
RVS.11.103			PMU Function List

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

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

ID	REFERENCE	TYPE	DEFINITION
RVS.12.1	12.0 (p.46)	H	Debug Console Extension (EID #0x4442434E "DBCN")
RVS.12.2	12.0 (p.46)	I	The debug console extension defines a generic mechanism for debugging and boot-time early prints from supervisor-mode software.
RVS.12.3	12.0 (p.46)	I	This extension replaces the legacy console putchar (EID #0x01) and console getchar (EID #0x02) extensions.
RVS.12.4	12.0 (p.46)	I	The debug console extension allows supervisor-mode software to write or read multiple bytes in a single SBI call.
RVS.12.5	12.0 (p.46)	R	If the underlying physical console has extra bits for error checking (or correction) then these extra bits should be handled by the SBI implementation.
RVS.12.6	12.0 (p.46)	C	It is recommended that bytes sent/received using the debug console extension follow UTF-8 character encoding.
RVS.12.7	12.1 (p.46)	H	Function: Console Write (FID #0)
RVS.12.8	12.1 (p.46)	R	`struct sbiret sbi_debug_console_write( unsigned long num_bytes, unsigned long base_addr_lo, unsigned long base_addr_hi)`
RVS.12.9	12.1 (p.46)	R	Write bytes to the debug console from input memory.
RVS.12.10	12.1 (p.46)	R	The **num_bytes** parameter specifies the number of bytes in the input memory.
RVS.12.11	12.1 (p.46)	R	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.
RVS.12.12	12.1 (p.46)	R	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.
RVS.12.13	12.1 (p.46)	R	The number of bytes written is returned in **sbiret.value** ...
RVS.12.14	12.1 (p.46) Table 47	R	… and the possible error codes returned in **sbiret.error** are shown in the table below. 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.12.15	12.2 (p.47)	H	Function: Console Read (FID #1)
RVS.12.16	12.2 (p.47)	R	`struct sbiret sbi_debug_console_read( unsigned long num_bytes, unsigned long base_addr_lo, unsigned long base_addr_hi)`
RVS.12.17	12.2 (p.47)	R	Read bytes from the debug console into an output memory.

ID	REFERENCE	TYPE	DEFINITION
RVS.12.18	12.2 (p.47)	R	The **num_bytes** parameter specifies the maximum number of bytes which can be written into the output memory.
RVS.12.19	12.2 (p.47)	R	The physical base address of the output memory is represented by two XLEN bits wide parameters.
RVS.12.20	12.2 (p.47)	R	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.
RVS.12.21	12.2 (p.47)	R	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.
RVS.12.22	12.2 (p.47)	R	The number of bytes read is returned in sbiret.value ...
RVS.12.23	12.2 (p.47) Table 48	R	… and the possible error codes returned in **sbiret.error** are shown in the table below. 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

ID	REFERENCE	TYPE	DEFINITION
RVS.12.24	12.3 (p.47)	H	Function: Console Write Byte (FID #2)
RVS.12.25	12.3 (p.47)	R	`struct sbiret sbi_debug_console_write_byte( uint8_t byte)`
RVS.12.26	12.3 (p.47)	R	Write a single byte to the debug console.
RVS.12.27	12.3 (p.47)	R	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.
RVS.12.28	12.3 (p.47)	R	The sbiret.value is set to zero
RVS.12.29	12.3 (p.47) Table 49	R	and the possible error codes returned in sbiret.error are shown in the table below. 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.12.30	12.4 (p.48)	H	Function Listing
RVS.12.31	12.4 (p.48) Table 50	R	DBCN Function List

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

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

ID	REFERENCE	TYPE	DEFINITION
RVS.13.1	13.0 (p.49)	H	System Suspend Extension (EID #0x53555350 "SUSP")
RVS.13.2	13.0 (p.49)	I	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.
RVS.13.3	13.0 (p.49) Table 51	I	Sleep states are identified with 32-bit wide identifiers (sleep_type). The possible values for the identifiers are shown in the table below. 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

ID	REFERENCE	TYPE	DEFINITION
RVS.13.4	13.1 (p.49)	H	Function: System Suspend (FID #0)
RVS.13.5	13.1 (p.49)	R	`struct sbiret sbi_system_suspend( uint32_t sleep_type, unsigned long resume_addr, unsigned long opaque)`
RVS.13.6	13.1 (p.49)	R	A return from a **sbi_system_suspend()** call implies an error and an error code from Table 53 will be in **sbiret.error**.
RVS.13.7	13.1 (p.49)	R	A successful suspend and wake up, results in the hart which initiated the suspend, resuming from the STOPPED state.
RVS.13.8	13.1 (p.49) Table 52	R	To resume, the hart will jump to supervisor-mode, at the address specified by **resume_addr**, with the specific register values described in the table below. 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.	

ID	REFERENCE	TYPE	DEFINITION
RVS.13.9	13.1 (p.50)	C	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.
RVS.13.10	13.1 (p.50)	R	The **resume_addr** parameter points to a runtime-specified physical address, where the hart can resume execution in supervisor-mode after a system suspend.
RVS.13.11	13.1 (p.50)	R	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.

ID	REFERENCE	TYPE	DEFINITION
RVS.13.12	13.1 (p.50)	R	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.
RVS.13.13	13.1 (p.50)	R	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.
RVS.13.14	13.1 (p.50)	R	When supervisor software is running inside a virtual machine, the SBI implementation is provided by a hypervisor.
RVS.13.15	13.1 (p.50)	R	System suspend will behave similarly to the native case from the point of view of the supervisor software.
RVS.13.16	13.1 (p.50) Table 53	R	The possible error codes returned in **sbiret.error** are shown in the table below. 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.13.17	13.2 (p.51)	H	Function Listing
RVS.13.18	13.2 (p.51) Table 54	R	SUSP Function List

Function Name	SBI Version	FID	EID
sbi_system_suspend	2.0	0	0x53555350

CHAPTER 14 CPPC Extension (EID #0x43505043 "CPPC")

ID	REFERENCE	TYPE	DEFINITION
RVS.14.1	14.0 (p.52)	H	CPPC Extension (EID #0x43505043 "CPPC")
RVS.14.2	14.0 (p.52)	I	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.
RVS.14.3	14.0 (p.52)	I	The SBI CPPC extension provides an abstraction to access the CPPC registers through SBI calls.
RVS.14.4	14.0 (p.52)	I	The CPPC registers can be memory locations shared with a separate platform entity such as a BMC.
RVS.14.5	14.0 (p.52)	I	Even though CPPC is defined in the ACPI specification, it may be possible to implement a CPPC driver based on Device Tree.

ID	REFERENCE	TYPE	DEFINITION
RVS.14.6	14.0 (p.52) Table 55	R	The table below defines 32-bit identifiers for all CPPC registers to be used by the SBI CPPC functions.

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	PerformanceReductionToleranceRegister	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	ReferencePerformanceCounterRegister	32/64	Read-only	ACPI Spec 6.5: 8.4.6.1.3.1
0x0000000C	DeliveredPerformanceCounterRegister	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
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 (worstcase) performance state transition latency in nanoseconds.
0x80000001-0xFFFFFFFF				Reserved for future use.

ID	REFERENCE	TYPE	DEFINITION
RVS.14.7	14.0 (p.52)	R	The first half of the 32-bit register space corresponds to the registers as defined by the ACPI specification.
RVS.14.8	14.0 (p.52)	R	The second half provides the information not defined in the ACPI specification, but is additionally required by the supervisor-mode power-management software.
RVS.14.9	14.1 (p.53)	H	Function: Probe CPPC register (FID #0)
RVS.14.10	14.1 (p.53)	R	`struct sbiret sbi_cppc_probe(uint32_t cppc_reg_id)`
RVS.14.11	14.1 (p.53)	R	Probe whether the CPPC register as specified by the **cppc_reg_id** parameter is implemented or not by the platform.
RVS.14.12	14.1 (p.53)	R	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.
RVS.14.13	14.1 (p.53, p.54)	R	If the register is not implemented, **sbiret.value** will be set to 0.
RVS.14.14	14.1 (p.54) Table 56	R	The possible error codes returned in **sbiret.error** are shown in the table below. 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.14.15	14.2 (p.54)	H	Function: Read CPPC register (FID #1)
RVS.14.16	14.2 (p.54)	R	`struct sbiret sbi_cppc_read(uint32_t cppc_reg_id)`
RVS.14.17	14.2 (p.54)	R	Reads the register as specified in the **cppc_reg_id** parameter and returns the value in **sbiret.value**.
RVS.14.18	14.2 (p.54)	R	When supervisor mode XLEN is 32, the **sbiret.value** will only contain the lower 32 bits of the CPPC register value.
RVS.14.19	14.2 (p.54) Table 57	R	The possible error codes returned in **sbiret.error** are shown in the table below. 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.14.20	14.3 (p.54)	H	Function: Read CPPC register high bits (FID #2)
RVS.14.21	14.3 (p.54)	R	`struct sbiret sbi_cppc_read_hi(uint32_t cppc_reg_id)`
RVS.14.22	14.3 (p.54)	R	Reads the upper 32-bit value of the register specified in the **cppc_reg_id** parameter and returns the value in **sbiret.value**.
RVS.14.23	14.3 (p.54)	R	This function always returns zero in **sbiret.value** when supervisor mode XLEN is 64 or higher.

ID	REFERENCE	TYPE	DEFINITION
RVS.14.24	14.3 (p.54) Table 58	R	The possible error codes returned in **sbiret.error** are shown in the table below. 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.14.25	14.4 (p.55)	H	Function: Write to CPPC register (FID #3)
RVS.14.26	14.4 (p.55)	R	`struct sbiret sbi_cppc_write( uint32_t cppc_reg_id, uint64_t val)`
RVS.14.27	14.4 (p.55)	R	Writes the value passed in the **val** parameter to the register as specified in the **cppc_reg_id** parameter.
RVS.14.28	14.4 (p.55) Table 59	R	The possible error codes returned in **sbiret.error** are shown in the table below. 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.
SBI_ERR_DENIED	**cppc_reg_id** is a read-only register.
SBI_ERR_FAILED	The write request failed for unspecified or unknown other reasons.

ID	REFERENCE	TYPE	DEFINITION
RVS.14.29	14.5 (p.55)	H	Function Listing
RVS.14.30	14.5 (p.55) Table 60	R	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

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

ID	REFERENCE	TYPE	DEFINITION
RVS.15.1	15.0 (p.56)	H	Nested Acceleration Extension (EID #0x4E41434C "NACL")
RVS.15.2	15.0 (p.56)	I	Nested virtualization is the ability of a hypervisor to run another hypervisor as a guest.
RVS.15.3	15.0 (p.56)	I	RISC-V nested virtualization requires an L0 hypervisor (running in hypervisor-mode) to trap-and-emulate the RISC-V H-extension† functionality (such as CSR accesses, HFENCE instructions, HLV/HSV instructions, etc.) for the L1 hypervisor (running in virtualized supervisor-mode).
RVS.15.4	15.0 (p.56)	I	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.
RVS.15.5	15.0 (p.56)	I	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.
RVS.15.6	15.0 (p.56)	C	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.
RVS.15.7	15.0 (p.56)	I	This SBI extension defines optional features which MUST be discovered by the supervisor software (or L1 hypervisor) before using the corresponding SBI functions.
RVS.15.8	15.0 (p.56)	R	Each nested acceleration feature is assigned a unique ID which is an unsigned 32-bit integer.
RVS.15.9	15.0 (p.56) Table 61	R	The table below below provides a list of all nested acceleration features. 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

ID	REFERENCE	TYPE	DEFINITION
RVS.15.10	15.0 (p.56)	R	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.
RVS.15.11	15.0 (p.56)	R	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 RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20211203, URL: github.com/riscv/riscv-isa-manual/releases/tag/Priv-v1.12

ID	REFERENCE	TYPE	DEFINITION
RVS.15.12	15.0 (p.56) Table 62	R	The table below shows the layout of nested acceleration shared memory.

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 RISCV privileged specification.

ID	REFERENCE	TYPE	DEFINITION
RVS.15.13	15.0 (p.57)	R	Any nested acceleration feature may define the contents of the scratch space shown in RVS.15.12 if required.
RVS.15.14	15.0 (p.57)	R	The contents of the CSR space shown in RVS.15.12 is an array of RISC-V H-extension CSR values where CSR **<x>** is at index **<i> = ((<x> & 0xc00) >> 2) | (<x> & 0xff)**.
RVS.15.15	15.0 (p.57)	R	The SBI implementation (or L0 hypervisor) MUST update the CSR space whenever the state of any RISC-V Hextension CSR changes unless some nested acceleration feature defines a different behaviour.
RVS.15.16	15.0 (p.57) Table 63	R	The table below shows CSR space index ranges for all possible 1024 RISC-V H-extension CSRs. 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`

ID	REFERENCE	TYPE	DEFINITION
RVS.15.17	15.1 (p.57)	H	Feature: Synchronize CSR (ID #0)
RVS.15.18	15.1 (p.57)	I	The synchronize CSR feature describes the ability of the SBI implementation (or L0 hypervisor) to allow supervisor software (or L1 hypervisor) to write RISC-V H-extension CSRs using the CSR space.
RVS.15.19	15.1 (p.57)	R	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 Hextension CSR.
RVS.15.20	15.1 (p.57, p.58)	R	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

ID	REFERENCE	TYPE	DEFINITION
RVS.15.21	15.1 (p.58)	R	To synchronize a CSR **<x>**, the SBI implementation (or L0 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
RVS.15.22	15.1 (p.58)	R	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 L0 hypervisor) MUST synchronize CSR **<x>** before CSR **<y>**.
RVS.15.23	15.1 (p.58)	I	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**.
RVS.15.24	15.2 (p.58)	H	Feature: Synchronize HFENCE (ID #1)
RVS.15.25	15.2 (p.58)	I	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.
RVS.15.26	15.2 (p.58)	R	This nested acceleration feature defines the scratch space offset range **0x0800 - 0x0F7F** (1920 bytes) as an array of nested HFENCE entries.
RVS.15.27	15.2 (p.58)	R	The total number of nested HFENCE entries are **3840 / XLEN** where each nested HFENCE entry consists of four XLEN-bit words.
RVS.15.28	15.2 (p.58)	R	A nested HFENCE entry is equivalent to an HFENCE over a range of guest addresses.
RVS.15.29	15.2 (p.58) Table 64	R	The table below shows the nested HFENCE entry format... 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

ID	REFERENCE	TYPE	DEFINITION
RVS.15.30	15.2 (p.58) Table 65	R	… whereas the table below provides a list of nested HFENCE entry types. 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.
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.

ID	REFERENCE	TYPE	DEFINITION
RVS.15.31	15.2 (p.58)	R	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.
RVS.15.32	15.2 (p.58)	R	After processing pending nested HFENCE entries, the SBI implementation (or L0 hypervisor) will clear the **Config.Pending** bit of these entries.
RVS.15.33	15.2 (p.60)	R	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
RVS.15.34	15.2 (p.60)	R	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
RVS.15.35	15.3 (p.60)	H	Feature: Synchronize SRET (ID #2)

ID	REFERENCE	TYPE	DEFINITION
RVS.15.36	15.3 (p.60)	I	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
RVS.15.37	15.3 (p.60)	R	This nested acceleration feature defines the scratch space offset range **0x0000 - 0x01FF** (512 bytes) as nested SRET context.
RVS.15.38	15.3 (p.60) Table 66	R	The table below shows contents of the nested SRET context. 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
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

ID	REFERENCE	TYPE	DEFINITION
RVS.15.39	15.3 (p.61)	R	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.

ID	REFERENCE	TYPE	DEFINITION
RVS.15.40	15.3 (p.61, p.62)	R	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. 4. Emulate the SRET instruction as defined by the RISC-V Privilege specification.
RVS.15.41	15.4 (p.62)	H	Feature: Autoswap CSR (ID #3)
RVS.15.42	15.4 (p.62)	I	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.
RVS.15.43	15.4 (p.62)	C	The supervisor software (or L1 hypervisor) should use the autoswap CSR feature in conjunction with the synchronize SRET feature.
RVS.15.44	15.4 (p.62)	R	This nested acceleration feature defines the scratch space offset range **0x0200 - 0x027F** (128 bytes) as nested autoswap context.
RVS.15.45	15.4 (p.62) Table 67	R	The table below shows contents of the nested autoswap context. 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.15.46	15.4 (p.62)	R	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.
RVS.15.47	15.4 (p.62, p.63)	R	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.
RVS.15.48	15.5 (p.63)	H	Function: Probe nested acceleration feature (FID #0)

RVS.15.49	15.5 (p.63)	R	`struct sbiret sbi_nacl_probe_feature(`
			`uint32_t feature_id)`

RVS.15.50	15.5 (p.63)	R	Probe a nested acceleration feature.

ID	REFERENCE	TYPE	DEFINITION
RVS.15.51	15.5 (p.63)	R	This is a mandatory function of the SBI nested acceleration extension.
RVS.15.52	15.5 (p.63)	R	The **feature_id** parameter specifies the nested acceleration feature to probe.
RVS.15.53	15.5 (p.63)	R	RVS.15.9 provides a list of possible feature IDs.
RVS.15.54	15.5 (p.63)	R	This function always returns SBI_SUCCESS in **sbiret.error**.
RVS.15.55	15.5 (p.63)	R	It returns 0 in **sbiret.value** if the given **feature_id** is not available, or 1 in **sbiret.value** if it is available.
RVS.15.56	15.6 (p.63)	H	Function: Set nested acceleration shared memory (FID #1)
RVS.15.57	15.6 (p.63)	R	`struct sbiret sbi_nacl_set_shmem(unsigned long shmem_phys_lo, unsigned long shmem_phys_hi, unsigned long flags)`
RVS.15.58	15.6 (p.63)	R	Set and enable the shared memory for nested acceleration on the calling hart.
RVS.15.59	15.6 (p.63)	R	This is a mandatory function of the SBI nested acceleration extension.
RVS.15.60	15.6 (p.63)	R	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.
RVS.15.61	15.6 (p.63)	R	**shmem_phys_lo** MUST be **4096** bytes (i.e. page) aligned and the size of the shared memory must be **4096 + (XLEN * 128)** bytes.
RVS.15.62	15.6 (p.63)	R	If both **shmem_phys_lo** and **shmem_phys_hi** parameters are all-ones bitwise then the nested acceleration features are disabled.
RVS.15.63	15.6 (p.63)	R	The flags parameter is reserved for future use and must be zero.
RVS.15.64	15.6 (p.63) Table 68	R	The possible error codes returned in **sbiret.error** are shown in the table below. 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.15.65	15.7 (p.64)	H	Function: Synchronize shared memory CSRs (FID #2)
RVS.15.66	15.7 (p.64)	R	`struct sbiret sbi_nacl_sync_csr(unsigned long csr_num)`
RVS.15.67	15.7 (p.64)	R	Synchronize CSRs in the nested acceleration shared memory.
RVS.15.68	15.7 (p.64)	R	This is an optional function which is only available if the SBI_NACL_FEAT_SYNC_CSR feature is available.
RVS.15.69	15.7 (p.64)	R	The parameter **csr_num** specifies the set of RISC-V H-extension CSRs to be synchronized.
RVS.15.70	15.7 (p.64)	R	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.

ID	REFERENCE	TYPE	DEFINITION
RVS.15.71	15.7 (p.64)	R	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.
RVS.15.72	15.7 (p.64) Table 69	R	The possible error codes returned in **sbiret.error** are shown in the table below. NACL Synchronize CSR Errors

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.

ID	REFERENCE	TYPE	DEFINITION
RVS.15.73	15.8 (p.64)	H	Function: Synchronize shared memory HFENCEs (FID #3)

RVS.15.74	15.8 (p.64)	R	`struct sbiret sbi_nacl_sync_hfence(`
			`                unsigned long entry_index)`

RVS.15.75	15.8 (p.64)	R	Synchronize HFENCEs in the nested acceleration shared memory.
RVS.15.76	15.8 (p.64)	R	This is an optional function which is only available if the SBI_NACL_FEAT_SYNC_HFENCE feature is available.
RVS.15.77	15.8 (p.64)	R	The parameter **entry_index** specifies the set of nested HFENCE entries to be synchronized.
RVS.15.78	15.8 (p.64, p.65)	R	If **entry_index** is all-ones bitwise then all nested HFENCE entries are synchronized as described in the Section 15.2.
RVS.15.79	15.8 (p.65)	R	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.
RVS.15.80	15.8 (p.65) Table 70	R	The possible error codes returned in **sbiret.error** are shown in the table below. 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.15.81	15.9 (p.65)	H	Function: Synchronize shared memory and emulate SRET (FID #4)
RVS.15.82	15.9 (p.65)	R	`struct sbiret sbi_nacl_sync_sret(void)`
RVS.15.83	15.9 (p.65)	R	Synchronize CSRs and HFENCEs in the nested acceleration shared memory and emulate the SRET instruction.
RVS.15.84	15.9 (p.65)	R	This is an optional function which is only available if the SBI_NACL_FEAT_SYNC_SRET feature is available.
RVS.15.85	15.9 (p.65)	R	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.

ID	REFERENCE	TYPE	DEFINITION
RVS.15.86	15.9 (p.65)	R	This function does not return upon success ...
RVS.15.87	15.9 (p.65) Table 71	R	… and the possible error codes returned in **sbiret.error** upon failure are shown in the tabe below. 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.15.88	15.10 (p.65)	H	Function Listing
RVS.15.89	15.10 (p.65)	R	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

CHAPTER 16 Steal-time Accounting Extension (EID #0x535441 "STA")

ID	REFERENCE	TYPE	DEFINITION
RVS.16.1	16.0 (p.67)	H	Steal-time Accounting Extension (EID #0x535441 "STA")
RVS.16.2	16.0 (p.67)	I	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.
RVS.16.3	16.0 (p.67)	I	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.
RVS.16.4	16.0 (p.67)	I	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.
RVS.16.5	16.0 (p.67)	I	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.
RVS.16.6	16.1 (p.67)	H	Function: Set Steal-time Shared Memory Address (FID #0)

RVS.16.7	16.1 (p.67)	R	`struct sbiret sbi_steal_time_set_shmem(`
			`      unsigned long shmem_phys_lo,`
			`      unsigned long shmem_phys_hi,`
			`      unsigned long flags)`

RVS.16.8	16.1 (p.67)	R	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.
RVS.16.9	16.1 (p.67)	R	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.
RVS.16.10	16.1 (p.67)	R	**shmem_phys_lo** MUST be 64-byte aligned.
RVS.16.11	16.1 (p.67)	R	The size of the shared memory must be at least 64 bytes.
RVS.16.12	16.1 (p.67)	R	The SBI implementation MUST zero the first 64 bytes of the shared memory before returning from the SBI call.
RVS.16.13	16.1 (p.67)	R	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.
RVS.16.14	16.1 (p.67)	R	The **flags** parameter is reserved for future use and MUST be zero.
RVS.16.15	16.1 (p.67)	R	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.
RVS.16.16	16.1 (p.67)	R	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.
RVS.16.17	16.1 (p.67)	R	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.

ID	REFERENCE	TYPE	DEFINITION
RVS.16.18	16.1 (p.67)	C	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.
RVS.16.19	16.1 (p.68) Table 73	R	The shared memory layout is defined in the table below. STA Shared Memory Structure

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.

ID	REFERENCE	TYPE	DEFINITION
RVS.16.20	16.1 (p.69)	R	**sbiret.value** is set to zero ...
RVS.16.21	16.1 (p.69) Table 74	R	… and the possible error codes returned in **sbiret.error** are shown in the table below. 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.

ID	REFERENCE	TYPE	DEFINITION
RVS.16.22	16.2 (p.69)	H	Function Listing

ID	REFERENCE	TYPE	DEFINITION
RVS.16.23	16.2 (p.69) Table 75	R	STA Function List

Function Name	SBI Version	FID	EID
sbi_steal_time_set_shmem	2.0	0	0x535441

CHAPTER 17 Experimental SBI Extension Space (EIDs #0x08000000 - #0x08FFFFFF)

ID	REFERENCE	TYPE	DEFINITION
RVS.17.1	17.0 (p.70)	H	Experimental SBI Extension Space (EIDs #0x08000000 - #0x08FFFFFF)
RVS.17.2	17.0 (p.70)	I	The SBI specification doesn’t define any rules for the EID management for experimental SBI extensions.

CHAPTER 18 Vendor Specific Extension Space (EIDs #0x09000000 - #0x09FFFFFF)

ID	REFERENCE	TYPE	DEFINITION
RVS.18.1	18.0 (p.71)	H	Vendor Specific Extension Space (EIDs #0x09000000 - #0x09FFFFFF)
RVS.18.2	18.0 (p.71)	R	The lower 24 bits of vendor specific EID must match the lower 24 bits of the **mvendorid** value.

CHAPTER 19 Firmware Specific Extension Space (EIDs #0x0A000000 - #0x0AFFFFFF)

ID	REFERENCE	TYPE	DEFINITION
RVS.19.1	19.0 (p.72)	H	Firmware Specific Extension Space (EIDs #0x0A000000 - #0x0AFFFFFF)
RVS.19.2	19.0 (p.72)	R	The lower 24 bits of the firmware EID must match the lower 24 bits of the SBI implementation ID.
RVS.19.3	19.0 (p.72)	R	The firmware specific SBI extensions space is reserved for SBI implementations.
RVS.19.4	19.0 (p.72)	I	It provides firmware specific SBI functions which are defined in the external firmware specification.