The future of RISC-V Supervisor Binary Interface (SBI)

Atish Patra
Principal R&D Engineer
Outline

- RISC-V Supervisor Binary Interface (SBI)
- Current status
- Limitations
- Extension proposal
- OpenSBI project
- Boot Flow
- Demo
RISC-V Supervisor Binary Interface (SBI)

- Provides an interface to access machine mode only registers
- Clear separation between Supervisor & Supervisor Execution Environment (SEE)
- Helps to run single OS image across different SEE
- Currently implemented by the Berkeley Boot Loader (BBL)

Calling convention
- S mode traps into M mode using `ecall` instructions
- Arguments are passed via registers a0-a2
- SBI call type is specified via register a7
- Unsupported SBI returns -38 (ENOSYS in Linux)
- a0 is clobbered register

Documentation available at
- https://github.com/riscv/riscv-sbi-doc
## Current Interface

<table>
<thead>
<tr>
<th>Type</th>
<th>Function</th>
<th>Function ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer</td>
<td>sbi_set_timer</td>
<td>0</td>
</tr>
<tr>
<td>Console</td>
<td>sbi_console_putchar</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>sbi_console_getchar</td>
<td>2</td>
</tr>
<tr>
<td>IPI</td>
<td>sbi_clear_ipi</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>sbi_send_ipi</td>
<td>4</td>
</tr>
<tr>
<td>Memory Model</td>
<td>sbi_remote_fence_i</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>sbi_remote_sfence_vma</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>sbi_remote_sfence_vma_asid</td>
<td>7</td>
</tr>
<tr>
<td>Shutdown</td>
<td>sbi_shutdown</td>
<td>8</td>
</tr>
</tbody>
</table>
Limitations

• Fixed, Not extendable
• No way to modify existing function signatures
• Changes cannot easily maintain backward compatibility
• No clean way to add new SBI function calls
  – Power management
  – Hart hotplug
  – Vendor specific extensions
SBI Scope

• Shouldn’t be treated as a kitchen sink
• New functionality only if absolutely necessary
• Backward compatibility
• No mandated usage of DT or ACPI
• Any functionality can be replaced by S-mode in future
• Anything else??
SBI proposal working model

• SBI specification to be part of the RISC-V Unix class platform specification
• Need to be approved by RISC-V Unix class platform specification working group
• Streamline the proposal discussion and quick turn around time
  ▪ A mandatory base SBI spec
  ▪ Existing SBI spec will be considered as legacy extension
  ▪ Every other SBI feature set will be a separate extension based on the base spec
  ▪ Every extension will be a sub-specification
  ▪ Can be discussed in parallel once the base SBI specification is finalized
• Need to have an implementation before freeze
SBI Base specification - I

- Calling Convention
  - Follows existing calling convention except return type
  - May return a structure
    
    ```c
    struct sbi_ret {
      long value;
      long error;
    }
    ```
    
    - Value in a0 as return value or error from SBI function
    - Error in a1 as any error SBI library wants to return

- A versioning scheme
  - 32bit unsigned integer
  - `<major.minor>` scheme
    
    - The existing legacy SBI version will be 0.1
    - The proposed base SBI version will be 0.2
SBI Base specification - II

- A SBI function ID scheme
  - A combination of function set number and function type number.
- Statically defined function set numbers.
- Both Hart/System power management functions will be a sub-specification.
- Use Reserved set in case of function set is not clear and may be standardized in future.
- Vendors specific functionalities should use Vendor function set.

<table>
<thead>
<tr>
<th>Function Set</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legacy</td>
<td>0x000</td>
<td>Existing SBI Functions. Not mandatory.</td>
</tr>
<tr>
<td>Base</td>
<td>0x001</td>
<td>Base Functions mandatory for any SBI version.</td>
</tr>
<tr>
<td>Hart PM</td>
<td>0x002</td>
<td>Hart power management.</td>
</tr>
<tr>
<td>System PM</td>
<td>0x003</td>
<td>System-level power management</td>
</tr>
<tr>
<td>Reserved</td>
<td>0x010-0x7ff</td>
<td>Reserved for experimental extensions</td>
</tr>
<tr>
<td>Vendor</td>
<td>0x800-0xffff</td>
<td>Vendor specific Functions</td>
</tr>
</tbody>
</table>
## SBI Base version Functions

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sbi_get_version(void)</code></td>
<td>Returns SBI specification version</td>
</tr>
<tr>
<td><code>sbi_set_sbiimp_version(void)</code></td>
<td>Returns SBI implementation version</td>
</tr>
<tr>
<td><code>sbi_is_function_set(u32 fset)</code></td>
<td>Check if given function set is valid or not</td>
</tr>
<tr>
<td><code>sbiret sbi_is_function_type(u32 ftype, u32 fset)</code></td>
<td>Check if a function type is implemented for a give function set.</td>
</tr>
<tr>
<td><code>sbiret sbi_get_vendor_id(void)</code></td>
<td>Returns the vendor ID</td>
</tr>
<tr>
<td><code>sbiret sbi_get_mimp_id(void)</code></td>
<td>Returns the machine implementation ID</td>
</tr>
<tr>
<td><code>sbi_get_sbiimp_id</code></td>
<td>Returns the SBI implementation ID</td>
</tr>
</tbody>
</table>
OpenSBI project – Why?

- BBL/Coreboot provides separate SBI implementation
- More fragmentations expected going forward considering vendor specific usage
- Difficult to maintain & track the SBI changes as it evolves
- Need a BBL replacement.
- Need easy plugin model for different platform/soc vendors
- OpenSBI to the rescue!!
OpenSBI project – What?

- An Open Source SBI implementation project
- Driven by community
- Licensed under BSD-2 clause
- Builds a static library that any M-mode boot loader can link
- Provides a reference implementation of platform code
- Provides a reference implementation of firmware code as well
- Protects firmware using PMP support
- Source level documentation using Doxygen
- Available at
  - https://github.com/riscv(opensbi)
OpenSBI project – How?

- **libsbi.a**
  - A static library that provides SBI implementation
  - Other M-mode boot loader may just link this for SBI functionality
  - Every future proposed SBI extension will be implemented

- **libplatsbi.a**
  - A static library that provides reference platform implementation
  - Contains minimal platform drivers required for bringup
  - Links libsbi.a for sbi implementation
  - Platform vendors are welcome to add their platform support

- **Currently supported platforms**
  - QEMU Virt
  - QEMU SiFive_u
  - SiFive Fu540 (HiFive Unleashed)
  - Kendryte K210 SoC
OpenSBI project – How?

• Builds several firmware binary
• A reference implementation
• Platform specific bootable firmware binary
• Firmware with Payload
  ▪ Any higher stage boot loader i.e. U-Boot binary as payload
  ▪ Supervisor OS i.e. Linux as a direct payload
  ▪ Allows separate device tree linking
• Firmware with Jump Address
  ▪ Jumps to address of next booting stage entry
  ▪ No need to provide payload binary for next stage
  ▪ Booting stage prior to OpenSBI should be capable of loading next stage module
• Vendors may choose to use one of the firmware as is or build their own
Current Boot flow – BBL + Linux

- Difficult add support for other platforms
- No way to separate DT from kernel image
- No network booting
- Kernel image has to be embedded bbl image

ZSBL – Zero Stage boot loader
FSBL – First Stage boot loader
BBL  – Berkeley boot loader
Boot flow – OpenSBI + Linux

- Full SMP support
- Follows existing RISC-V boot flow model
- Takes Linux Image file as a payload
- Can accept separate DT file as well
Boot flow – OpenSBI + U-Boot + Linux

- Follows a standard boot flow
- Uses U-Boot as the last stage boot loader
- U-Boot binary as the payload to OpenSBI
- Linux image loaded via tftp
- Linux SMP (all cores) will be usable once SBI hart power management extension is available in OpenSBI
Booting – QEMU

• Instructions available at docs/platform/qemu_virt.md

• Linux Image as a direct payload
  – Building
  ```
  make PLATFORM=qemu/virt FW_PAYLOAD_PATH=<linux_build_directory>/arch/riscv/boot/Image
  ```
  – Running
  ```
  qemu-system-riscv64 -M virt -m 256M -display none -serial stdio -kernel build/platform/qemu/virt/firmware/fw_payload.elf \\
  -drive file=<path_to_linux_rootfs>,format=raw,id=hd0 \\
  -device virtio-blk-device,drive=hd0 \\
  -append "root=/dev/vda rw console=ttys0"
  ```

• U-Boot image as a payload
  – Building
  ```
  make PLATFORM=qemu/virt FW_PAYLOAD_PATH=<uboot_build_directory>/u-boot.bin
  ```
  – Running
  ```
  qemu-system-riscv64 -M virt -m 256M -display none -serial stdio -kernel build/platform/qemu/virt/firmware/fw_payload.elf
  ```
Booting – HiFive Unleashed

- Instructions also available at docs/platform/sifive_fu540.md
- Linux Image as a direct payload
  
  ```
  make PLATFORM=sifive/fu540 \
  FW_PAYLOAD_PATH=<linux_build_directory>/arch/riscv/boot/Image
  ```

- U-Boot binary as payload
  
  ```
  make PLAT=sifive/hifive_u540 \
  FW_PAYLOAD_PATH=~/workspace/u-boot-riscv/u-boot.bin
  ```
Booting – HiFive Unleashed
Demo – OpenSBI + Linux
Demo – OpenSBI + U-Boot + Linux
OpenSBI future work

- 32-bit support
- Yocto recipe for OpenSBI in meta-riscv
- SBI v0.2 support (after SBI extension spec frozen)
- SBI Hart power management support
- Link libsbi in U-Boot M mode
- Link libsbi in Coreboot (Volunteer ??)
- CLIC use in OpenSBI (???)
- More platforms ...
Thank you!!

- Q&A?
# Function type list

## SBI Function List in both SBI v0.2 and v0.1

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Function Set</th>
<th>ID(v0.2)</th>
<th>ID (v0.1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbi_set_timer</td>
<td>Legacy</td>
<td>0x0000 0000</td>
<td>0</td>
</tr>
<tr>
<td>sbi_console_putchar</td>
<td>Legacy</td>
<td>0x0000 0001</td>
<td>1</td>
</tr>
<tr>
<td>sbi_console_getchar</td>
<td>Legacy</td>
<td>0x0000 0002</td>
<td>2</td>
</tr>
<tr>
<td>sbi_clear_ipl</td>
<td>Legacy</td>
<td>0x0000 0003</td>
<td>3</td>
</tr>
<tr>
<td>sbi_send_ipl</td>
<td>Legacy</td>
<td>0x0000 0004</td>
<td>4</td>
</tr>
<tr>
<td>sbi_remote_fence_i</td>
<td>Legacy</td>
<td>0x0000 0005</td>
<td>5</td>
</tr>
<tr>
<td>sbi_remote_fence_vma</td>
<td>Legacy</td>
<td>0x0000 0006</td>
<td>6</td>
</tr>
<tr>
<td>sbi_remote_fence_vma_asid</td>
<td>Legacy</td>
<td>0x0000 0007</td>
<td>7</td>
</tr>
<tr>
<td>sbi_shutdown</td>
<td>Legacy</td>
<td>0x0000 0008</td>
<td>8</td>
</tr>
<tr>
<td>sbi_get_spec_version</td>
<td>Base</td>
<td>0x0010 0001</td>
<td>-</td>
</tr>
<tr>
<td>sbi_set_sbiimp_version</td>
<td>Base</td>
<td>0x0010 0002</td>
<td>-</td>
</tr>
<tr>
<td>sbi_is_function_set</td>
<td>Base</td>
<td>0x0010 0003</td>
<td>-</td>
</tr>
<tr>
<td>sbi_is_function_type</td>
<td>Base</td>
<td>0x0010 0003</td>
<td>-</td>
</tr>
<tr>
<td>sbi_get_vendor_id</td>
<td>Base</td>
<td>0x0010 0004</td>
<td>-</td>
</tr>
<tr>
<td>sbi_get_mlimp_id</td>
<td>Base</td>
<td>0x0010 0005</td>
<td>-</td>
</tr>
<tr>
<td>sbi_get_sbiimp_id</td>
<td>Base</td>
<td>0x0010 0006</td>
<td>-</td>
</tr>
<tr>
<td>sbi_set_timer</td>
<td>Exp-1</td>
<td>0x0100 0000</td>
<td>-</td>
</tr>
<tr>
<td>sbi_console_putchar</td>
<td>Exp-1</td>
<td>0x0100 0001</td>
<td>-</td>
</tr>
<tr>
<td>sbi_console_getchar</td>
<td>Exp-1</td>
<td>0x0100 0002</td>
<td>-</td>
</tr>
<tr>
<td>sbi_clear_ipl</td>
<td>Exp-1</td>
<td>0x0100 0003</td>
<td>-</td>
</tr>
<tr>
<td>sbi_send_ipl</td>
<td>Exp-1</td>
<td>0x0100 0004</td>
<td>-</td>
</tr>
<tr>
<td>sbi_remote_fence_i</td>
<td>Exp-1</td>
<td>0x0100 0005</td>
<td>-</td>
</tr>
<tr>
<td>sbi_remote_fence_vma</td>
<td>Exp-1</td>
<td>0x0100 0006</td>
<td>-</td>
</tr>
<tr>
<td>sbi_remote_fence_vma_asid</td>
<td>Exp-1</td>
<td>0x0100 0007</td>
<td>-</td>
</tr>
</tbody>
</table>
## Error code table

<table>
<thead>
<tr>
<th>Error Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBI_SUCCESS</td>
<td>0</td>
</tr>
<tr>
<td>SBI_ERR_FAILURE</td>
<td>-1</td>
</tr>
<tr>
<td>SBI_ERR_NOT_SUPPORTED</td>
<td>-2</td>
</tr>
<tr>
<td>SBI_ERR_INVALID_PARAM</td>
<td>-3</td>
</tr>
<tr>
<td>SBI_ERR_DENIED</td>
<td>-4</td>
</tr>
<tr>
<td>SBI_ERR_INVALID_ADDRESS</td>
<td>-5</td>
</tr>
</tbody>
</table>
OpenSBI usage constraints

- With libsbi.a, firmware has to provide the platform specific hooks
- RISC-V MSCRATCH CSR must point to a valid OpenSBI scratch space
- RISC-V SP register (i.e. stack pointer) must be set per-HART pointing to distinct non-overlapping stacks
- Only calls two functions
  - sbi_init – gets called when hart boots up
  - sbi_trap_handler – Forward all traps and interrupts or at least for the following
    - M-mode timer interrupt
    - M-mode software interrupt
    - Illegal instruction trap
    - Misaligned load trap
    - Misaligned store trap
    - Supervisor ecall trap
    - Hypervisor ecall trap