OpenSBI Deep Dive

Anup Patel <anup.patel@wdc.com>
Western Digital Research
Outline

• OpenSBI Introduction
  – Overview and features

• OpenSBI Platform Specific Support

• OpenSBI Usage
  – As a firmware: Reference Firmwares
  – As a library: API

• Conclusion
OpenSBI Introduction
What is SBI?

- SBI stands for RISC-V Supervisor Binary Interface
  - System call style calling convention between Supervisor (S-mode OS) and Supervisor Execution Environment (SEE)

- SEE can be:
  - A M-mode RUNTIME firmware for OS/Hypervisor running in HS-mode
  - A HS-mode Hypervisor for Guest OS running in VS-mode

- SBI calls help:
  - Reduce duplicate platform code across OSes (Linux, FreeBSD, etc)
  - Provide common drivers for an OS which can be shared by multiple platforms
  - Provide an interface for direct access to hardware resources (M-mode only resources)

- Specifications being drafted by the Unix Platform Specification Working group
  - Maintain and evolve the SBI specifications
  - Currently, SBI v0.1 in-use and SBI v0.2 in draft stage
What is OpenSBI?

• OpenSBI is an open-source implementation of the RISC-V Supervisor Binary Interface (SBI) specifications
  – Licensed under the terms of the BSD-2 clause license
  – Helps to avoid SBI implementation fragmentation

• Aimed at providing RUNTIME services in M-mode
  – Typically used in boot stage following ROM/LOADER

• Provides support for reference platforms
  – Generic simple drivers included for M-mode to operate
    ▪ PLIC, CLINT, UART 8250
  – Other platforms can reuse the common code and add needed drivers
Typical Boot Flow

authenticate & loads

jumps

- runs from on-chip sram
- ddr initialization
- loads runtime and bootloader

- runs from ddr
- typically open-source
- filesystem support
- network booting
- boot configuration
- lots of other features

- runs from on-chip rom
- uses on-chip sram
- soc power-up and clock setup

- runs from ddr
- soc security setup
- runtime services as-per specifications
Important Features

• Layered structure to accommodate various use cases
  – Generic SBI library with platform abstraction
    • Typically used with external firmware and bootloader
      – EDK2 (UEFI implementation), Secure boot working group
  – Platform specific library
    • Similar to core library but including platform specific drivers
  – Platform specific reference firmware
    • Three different types of RUNTIME firmware

• Wide range of hardware features supported
  – RV32 and RV64
  – Misaligned load/store handling
  – Missing CSR emulation
  – Protects firmware using PMP support

• Well documented using Doxygen
OpenSBI Platform Specific Support
Why Platform Specific Support?

• Any SBI implementation requires hardware dependent (platform-specific) methods
  – Print a character to console
  – Get an input character from console
  – Inject an IPI to any given HART subset
  – Get value of memory-mapped system timer
  – Start timer event for a given HART
  – … more to come …

• OpenSBI platform-specific support is implemented as a set of platform-specific hooks in the form of a struct sbi_platform data structure instance
  – Hooks are pointers to platform dependent functions

• Platform independent generic OpenSBI code is linked into a libsbi.a static library

• For every supported platform, we create a libplatsbi.a static library
  – libplatsbi.a = libsbi.a + struct sbi_platform instance
Supported Platforms

• Supported platforms are available under `/platform` directory in OpenSBI source code tree

• Currently:
  – qemu/virt: QEMU RISC-V generic virtual machine
    (Refer, docs/platform/qemu_virt.md)
  – qemu/sifive_u: QEMU SiFive Unleashed virtual machine
    (Refer, docs/platform/qemu_sifive_u.md)
  – sifive/fu540: SiFive FU540 SOC
    (Refer, docs/platform/sifive_fu540.md)
  – kendryte/k210: Kendryte K210 SOC

• More to come
Adding Support for New Platforms

• To add support for a new <xyz> platform
  1. Create directory named <xyz> under /platform directory
  2. Create platform configuration file <xyz>/config.mk
     ▪ config.mk will provide compiler flags, select common drivers, and select firmware options
     ▪ platform/template/config.mk can be used as reference for creating config.mk
  3. Create platform objects file <xyz>/objects.mk for listing platform-specific objects to be compiled
     ▪ platform/template/objects.mk can be used as reference for creating objects.mk
  4. Create platform source file <xyz>/platform.c providing “struct sbi_platform” instance
     ▪ platform/template/platform.c can be used as reference for creating platform.c

• The <xyz> platform support directory can also placed outside OpenSBI sources
Compilation Options for Platform Support

• `CROSS_COMPILE` environment variable need to be set for cross-compilation

• Build only generic OpenSBI (`libsbi.a`)  
  – `make`

• Build platform-specific OpenSBI (`libplatsbi.a`) for `platform/<xyz>` in OpenSBI sources  
  – `make PLATFORM=<xyz>`

• Build platform-specific OpenSBI (`libplatsbi.a`) for `<xyz>` not part of OpenSBI sources  
  – `make PLATFORM_DIR=<path_to_<xyz>_directory>`
Using OpenSBI As a Firmware
Reference Firmwares

• OpenSBI provides several types of reference firmware, all platform-specific
  – **FW_PAYLOAD**: Firmware with the next booting stage as a payload
  – **FW_JUMP**: Firmware with static jump address to the next booting stage
  – **FW_DYNAMIC**: Firmware with dynamic information on the next booting stage

• SOC Vendors may choose:
  – Use one of OpenSBI reference firmwares as their M-mode RUNTIME firmware
  – Build M-mode RUNTIME firmware from scratch with OpenSBI as library
  – Extend existing M-mode firmwares (U-Boot_M_mode/EDK2) with OpenSBI as library
• OpenSBI firmware with the next booting stage as a payload
  – Any S-mode BOOTLOADER/OS image as the payload to OpenSBI FW_PAYLOAD
  – Allows overriding device tree blob (i.e. DTB)
  – Very similar to BBL hence fits nicely in existing boot-flow of SiFive Unleashed board

• Down-side:
  – We have to re-create FW_PAYLOAD image whenever OpenSBI or the BOOTLOADER (U-Boot) changes
  – No mechanism to pass parameters from previous booting stage (i.e. LOADER) to FW_PAYLOAD
FW_JUMP

- OpenSBI firmware with a fixed jump address to the next booting stage
  - Next stage booting stage (i.e. BOOTLADER) and FW_JUMP are loaded by the previous booting stage (i.e. LOADER)
  - Very useful for QEMU because we can use pre-compiled FW_JUMP

- Down-side:
  - Previous booting stage (i.e. LOADER) has to load next booting stage (i.e. BOOTLADER) at a fixed location
  - No mechanism to pass parameters from previous booting stage (i.e. LOADER) to FW_JUMP
OpenSBI firmware with dynamic information about the next booting stage
- The next stage booting stage (i.e. BOOTLADER) and FW_DYNAMIC are loaded by the previous booting stage (i.e. LOADER)
- The previous booting stage (i.e. LOADER) passes the location of `struct fw_dynamic_info` to FW_DYNAMIC via ‘a2’ register

Down-side:
- Previous booting stage (i.e. LOADER) needs to be aware of `struct fw_dynamic_info`
Using OpenSBI As a Library
Typical use as Library

- External M-mode firmware linked to OpenSBI library
- Example: open-source EDK2 (UEFI implementation) OpenSBI integration
  - HPE leading this effort (Ongoing)
  - OpenSBI built with EDK2 build environment
Constraints on using OpenSBI Library

• Same GCC target options (i.e. `-march`, `-mabi`, and `-mcmodel`) need to be used for the external firmware and OpenSBI sources

• External firmware must create per-HART non-overlapping:
  1. Program Stack
  2. OpenSBI scratch space (i.e. `struct sbi_scratch` instance with extra space above)

• Two constraints in calling any OpenSBI functions from external firmware:
  1. `MSCRATCH` CSR of calling HART must be set to its own OpenSBI scratch space
  2. `SP` register (i.e. the stack pointer) of calling HART must be set to its own stack

• External firmware must also ensure that:
  – Interrupts are disabled in the `MSTATUS` and `MIE` CSRs when calling `sbi_init()`
  – `sbi_init()` is called for each HART that is powered-up at boot-time or in response to a CPU hotplug event
  – `sbi_trap_handler()` is called for M-mode interrupts and M-mode traps
Conclusion
Important Facts

• OpenSBI provides only RUNTIME firmware/library

• OpenSBI platform specific support makes OpenSBI easily extensible for new SOCs

• OpenSBI reference firmwares:
  – Are optional and SOC vendors can choose to implement their own
  – Don’t enforce any particular boot-flow
On-Going and Future Work

• SBI specifications
  – SBI v0.2 specification
  – SBI v0.2 HART power management extension
  – SBI v0.2 remote fences extension (fence.i, sfence.vma, hfence.gvma, and hfence bvma)

• OpenSBI
  – RISC-V hypervisor extension support (We have a demo here !!!)
  – SBI v0.2 support
  – SBI v0.2 HART power management support
  – SBI v0.2 remote fences support
  – Support other M-mode bootloaders such as U-Boot_SPL/Coreboot
  – Support RISC-V EDK2 integration
  – More platforms support
    • Need hardware !