opensbi source code analysis

Xiang W <wxjstz@126.com / wangxiang@nj.iscas.ac.cn>

Outline

- **Introduction**
- **History**
- **Basic Framework**
- **Extension**
  - sbi_ecall
  - sbi_console
  - sbi_timer
  - sbi_ipi
  - sbi_tlb
  - sbi_hsm
  - sbi_domain
  - sbi_pmu

Introduction

- RISC-V has three privilege levels (M-Mode/S-Mode/U-Mode), and the operating system running in S-Mode. In M-Mode, some basic firmware running here, these firmware need to provide some interfaces to the operating system, these interfaces are called SBI (Supervisor Binary Interface)
- opensbi is an implementation of SBI
- SBI definition can refer to https://github.com/riscv/riscv-sbi-doc

History

- There are currently three versions of SBI: v0.1.0, v0.2.0, v0.30
- Among them, v0.3.0 just adds some extensions on the basic of v0.2.0
- There is a big difference between v0.1.0 and v0.2.0

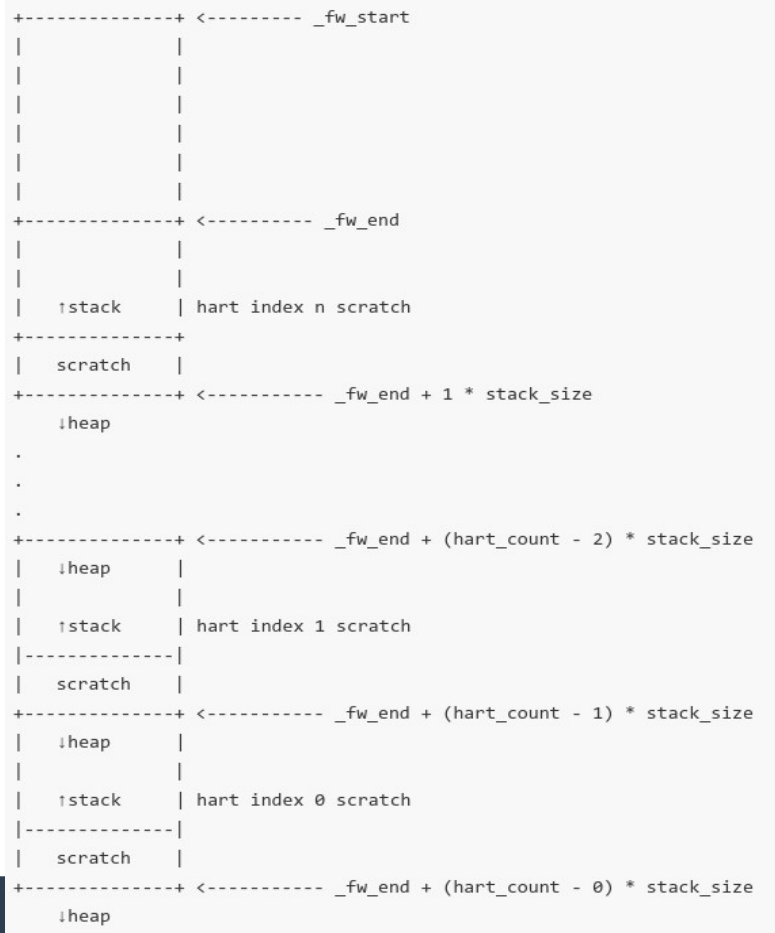
Description	V0.1.0	V0.2.0/V0.3.0
Extension ID	-	a7
Function ID	a7	a6
Arguments	a0 - a6	a0 - a5
Return Value	a0	a0, a1

History

- **New extension improvements**
  - Expansibility
  - Added a clear return result, there will be no response as before
  - Detectable, the operating system can query SBI information and detect whether the extension is available (implemented by adding a basic extension with 0x10 extension id)
  - Backward compatibility, by using the function ID 0-8 called by the old SBI as the extension ID of the new extension(ecall_legacy)

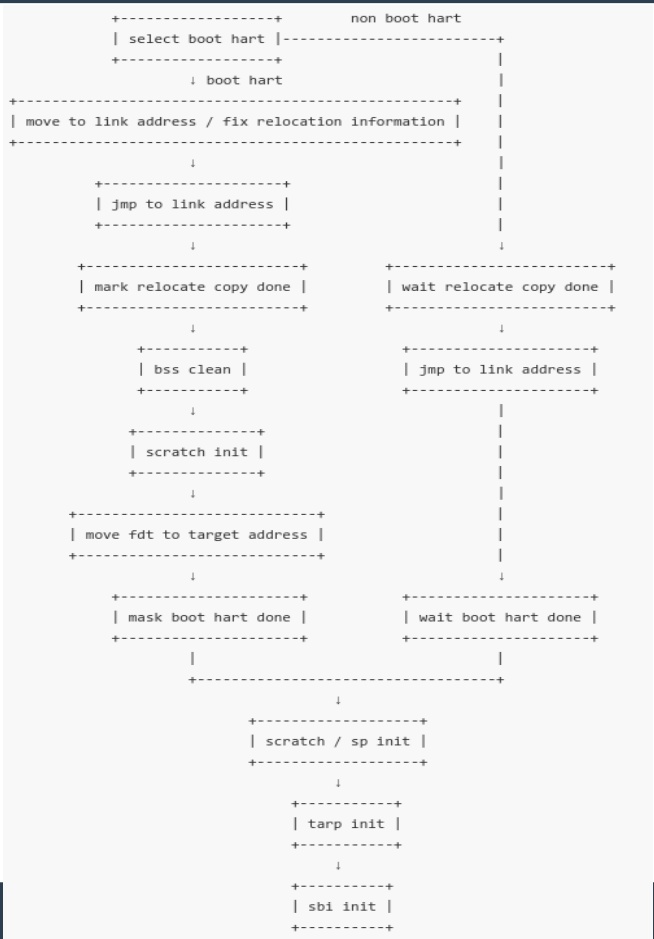
Basic Framework (Memory Layout)

- The memory layout is mainly realized by the code in the firmware directory
- The heap marked in the figure is not a real heap (cannot release memory), only used to expand scratch
- Point to sbi_scratch through the MSCRATCH register, so that sbi_scratch can be easily accessed, and the stack can be reset when interrupted



Basic Framework (Initialization Process)

- **These codes are mainly located in:**
  - firmware/base.S
- **There are currently two linking methods, one is to generate position-independent code, and the other is to move the code to the target location before running**



Basic Framework (Firmware Type)

- **Currently, three types of firmware can be generated: payload jump and dynamic. By implementing the hook in firmware/base.S.**
  - payload: The next level of firmware is linked to opensbi, and the next level of firmware running in S-Mode
  - jump: The firmware is stored in a fixed location in the memory. After the opensbi initialization is completed, it jumps to this address, and the next level of firmware running in S-Mode
  - dynamic: the firmware information is passed in through a register(this register point to struct fw_dynamic_info)

Basic Framework (Portability)

- For portability, opensbi defines two structures sbi_platform/sbi_platform_operations. sbi_platform mainly saves some platform-related description information. sbi_platform_operations stores some hook functions, which are called in some extensions.
- The main work of transplantation is to realize these two structures

Basic Framework (misc)

- **RISC-V does not stipulate that hart IDs need to be continuous. When allocating some resources (such as stacks), it is necessary to convert hart IDs into continuous indexes. In order to achieve the conversion between hart id and index, the platform needs to implement an array hart_index2id**

```
struct sbi_platform {
    //...
    const u32 *hart_index2id;
};
```

Extension (sbi_ecall)

- **In order to facilitate the expansion of the SBI interface, the system defines a structure to maintain the SBI extension. To implement an SBI extension, you only need to implement this structure.**

```
struct sbi_ecall_extension {
    struct sbi_dlist head;
    unsigned long extid_start;
    unsigned long extid_end;
    int (* probe)(unsigned long extid, unsigned long *out_val);
    int (* handle)(unsigned long extid, unsigned long funcid,
    const struct sbi_trap_regs *regs,
    unsigned long *out_val,
    struct sbi_trap_info *out_trap);
};
```

- **A set of extended extension ids can have gaps, and the probe method is needed to determine whether the specific extension ids are available**
- **opensbi maintains extension through linked list**

Extension (sbi_console)

- **This module mainly implements a function similar to printf, which is used to print some debugging information. And exported sbi_getc and sbi_putc for use in SBI calls.**
- **The specific platform needs to implement the following interfaces:**

```
struct sbi_platform_operations {
    // ...
    /** Write a character to the platform console output */
    void (*console_putc)(char ch);
    /** Read a character from the platform console input */
    int (*console_getc)(void);
    /** Initialize the platform console */
    int (*console_init)(void);
    // ⋯
}
```

Extension (sbi_timer)

- **This module mainly implements timing interrupts, and forwards the interrupts to S-Mode. And exports some interfaces to SBI to call.**
- **The specific platform needs to implement the following interfaces**

```
struct sbi_platform_operations {
  // ...
  /** Get platform timer value */
  u64 (*timer_value)(void);
  /** Start platform timer event for current HART */
  void (*timer_event_start)(u64 next_event);
  /** Stop platform timer event for current HART */
  void (*timer_event_stop)(void);
  /** Initialize platform timer for current HART */
  int (*timer_init)(bool cold_boot);
  /** Exit platform timer for current HART */
  void (*timer_exit)(void);
  // ...
}
```

Extension (sbi_ipi)

- **IPI (inter-processor interrupt) is a method for multi-core communication. Usually, the hardware provides only one interrupt signal. In order to transmit multiple types of messages through ipi, a variable ipi_data is created on each sbi_scratch, and the type of interrupt to be sent is identified by setting the bit in ipi_data before sending the ipi interrupt.**
- **The system describes how to handle ipi interrupts through a structure**

```
struct sbi_ipi_event_ops {
    char name[32];
    // Pass parameters to hart receiving ipi interrupt
    int (* update)(struct sbi_scratch *scratch,struct sbi_scratch *remote_scratch,u32 remote_hartid, void *data);
    // Perform synchronization operations
    void (* sync)(struct sbi_scratch *scratch);
    // The hart that received the ipi interrupt executes this function
    void (* process)(struct sbi_scratch *scratch);
};
```

- **The system has an array to record how to deal with various types of ipi**

Extension (sbi_ipi)

- **sbi_ipi provides a function for other modules to register ipi interrupts**

  int sbi_ipi_event_create(const struct sbi_ipi_event_ops *ops)

- **The specific platform needs to implement the following interfaces**

```
struct sbi_platform_operations {
  // ...
  /** Send IPI to a target HART */
  void (*ipi_send)(u32 target_hart);
  /** Clear IPI for a target HART */
  void (*ipi_clear)(u32 target_hart);
  /** Initialize IPI for current HART */
  int (*ipi_init)(bool cold_boot);
  /** Exit IPI for current HART */
  void (*ipi_exit)(void);
  // ...
}
```

Extension (sbi_tlb)

- **sbi_tlb is mainly used to handle cache coherency between multiple hart**
- **multi-core communication through ipi message**
- **In order to receive messages from multiple harts at the same time, a fifo queue is created on each sbi_scratch to prevent message loss**
- **The specific platform needs to implement the following interfaces:**

```
struct sbi_platform_operations {
  /** Get tlb flush limit value **/
  u64 (*get_tlbr_flush_limit)(void);
}
```

Extension (sbi_hsm)

- hsm (hart status management), this extension is used to manage hart, can be used to manage hart hibernation, or reload the operating system.

Extension (sbi_hsm)

- **In order to record the state of each hart, a structure is created on sbi_scratch to record the current processor state**
- **This structure is also used to record the interrupt (mie/mip) status when suspused**
- **The specific platform needs to implement the following interfaces**

```
struct sbi_platform_operations {
  // ...
  int (*hart_start)(u32 hartid, ulong saddr);
  int (*hart_stop)(void);
  int (*hart_suspend)(u32 suspend_type, ulong raddr);
  // ...
}
```

Extension (sbi_domain)

- **sbi_domain is used to manage a group of hart, they have the same memory access permissions (physical memory access permissions, PMP). The domain is described by the following structure**

```
struct sbi_domain {
    u32 index;
    struct sbi_hartmask assigned_harts;
    char name[64];
    const struct sbi_hartmask *possible_harts;
    struct sbi_domain_memregion *regions;
    u32 boot_hartid;
    unsigned long next_arg1;
    unsigned long next_addr;
    unsigned long next_mode;
    bool system_reset_allowed;
};
```

- **The current system supports up to 32 domains. Each hart is bound to a domain.**

Extension (sbi_domain)

- **Currently opensbi has created a domain (root) mainly used to record the resources used by opensbi and prevent low-privilege access.**
- **The specific platform needs to implement the following interfaces**

```
struct sbi_platform_operations {
  // ...
  /** Get platform specific root domain memory regions */
  struct sbi_domain_memregion *(*domains_root_regions)(void);
  /** Initialize (or populate) domains for the platform */
  int (*domains_init)(void);
  // ...
}
```

Extension (sbi_pmu)

- PMU is short for Performance Monitoring Unit, PMU is used to manage some counters related to performance.
- opensbi manages counters provided by hardware and also manages some events defined by opensbi
- Because the RISC-V standard only defines the specific functions of two counters, the other counter functions are defined by the specific platform. So the software operates these timers through abstract events. These events are used to describe a timer (type and specific information).
- The specific platform will transmit some information to describe these timer functions

Extension (sbi_pmu)

- **The software-defined counter is described by the following structure**

```
struct sbi_pmu_fw_event {
    unsigned long event_idx;
    unsigned long curr_count;
    bool bStarted;
};
```

- **The system organizes these counters through an array, and defines these counters for each hart**

```
static struct sbi_pmu_fw_event fw_event_map[SBI_HARTMASK_MAX_BITS][SBI_PMU_FW_EVENT_MAX] = {0};
```

- **The system identifies a counter through a logical id, which facilitates the management of the counter, especially the software-defined counter. In order to record the mapping relationship between logical id and counter, it is described by the following array**

```
static uint32_t active_events[SBI_HARTMASK_MAX_BITS][SBI_PMU_HW_CTR_MAX + SBI_PMU_FW_CTR_MAX];
```

Extension (sbi_pmu)

- **Main API**
  - match:pass in an event, opensbi helps to find the specified counter, execute the corresponding operation, and return the logical id
  - Read write or control these counters through logical id

Reference

- https://github.com/riscv/opensbi
- https://github.com/riscv/riscv-sbi-doc


Think You !