从零开始的RISC-V模拟器开发

第14讲 QEMU篇之系统集成

中国科学院软件研究所

PLCT实验室

王俊强 wangjunqiang@iscas.ac.cn

李威威 liweiwei@iscas.ac.cn

吴伟 wuwei2016@iscas.ac.cn

本课内容

Machine(Linux)集成

- 支持Linux Machine构建
- 整体流程

Linux 与 NUCLEI CPU

获取地址:
https://github.com/Nuclei-Software/nuclei-linux-sdk.git

- .github/workflows
- buildroot @ 2665fb0 → For rootfs
- conf
- freeloader @ 39c8eb9
- linux @ 70e1148 → For Linxu kernel
- opensbi @ ad8651f → For opensbi
- u-boot @ ccba5cf → For u-boot
- .gitignore
- .gitlab-ci.yml
- .gitmodules
- LICENSE
- Makefile
- Makefile.core
- README.md

编译方式:
make xxx
参考:make help

Test For UX600FD

XI_spike/QEMU	QEMU/FPGA
opensbi	opensbi
Linux	Uboot
(initrd)rootfs	Linux
	rootfs

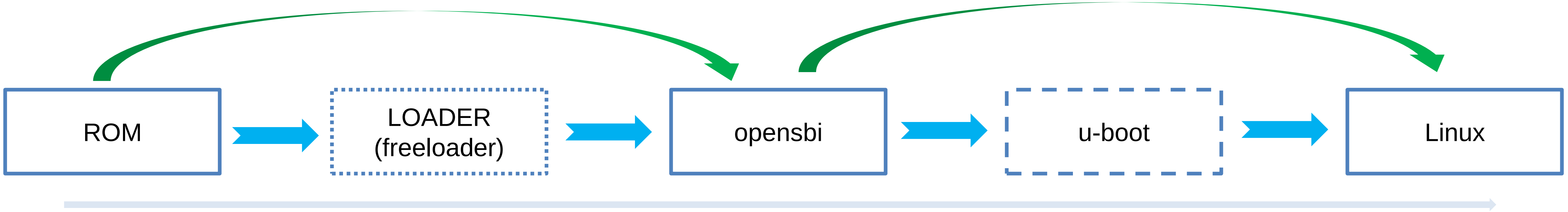
参考: https://github.com/Nuclei-Software/nuclei-linux-sdk/blob/dev_nuclei_next/README.md

opensbi模式:
https://github.com/riscv/opensbi/blob/master/docs/platform/qemu_virt.md

Linux 与 NUCLEI CPU

包含组件	主要用途
freeloader	opensbi u-boot搬运
opensbi	初始化设置
u-boot	初始化设置/引导OS
linux	OS初始化并挂载文件系统
buildroot	生成文件系统

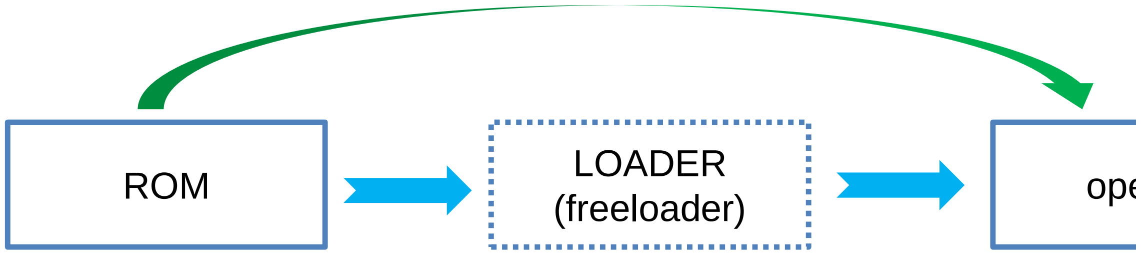
目前支持外设：
Nuclei UART/USART(兼容sifive)
Nuclei SPI(兼容sifive)
Nuclei GPIO(兼容sifive)
目前支持CPU：
ux600 and ux900
ux600fd and ux900fd



opensbi模式:https://github.com/riscv/opensbi/blob/master/docs/platform/qemu_virt.md 源码: https://github.com/Nuclei-Software/nuclei-linux-sdk.git

Linux 与 NUCLEI CPU

包含组件	主要用途
freeloader	opensbi u-boot搬运
opensbi	初始化设置
u-boot	初始化设置/引导OS
linux	OS初始化并挂载文件系统
buildroot	生成文件系统



	Component	Address Spaces	Description
Core Private Peripherals	TIMER	0x0200_0000 ~ 0x0200_0FFF	TIMER Unit address space.
	ECLIC	0x0C00_0000 ~ 0x0C00_FFFF	ECLIC Unit address space.
	DEBUG	0x0000_0000 ~ 0x0000_0FFF	DEBUG Unit address space.
Memory Resource	ILM	0x8000_0000 ~	ILM address space.
	DLM	0x9000_0000 ~	DLM address space.
	ROM	0x0000_1000 ~ 0x0000_1FFF	Internal ROM.
	Off-Chip QSPI0 Flash Read	0x2000_0000 ~ 0x3FFF_FFFF	QSPI0 with XiP mode read-only address space.
Peripherals	GPIO	0x1001_2000 ~ 0x1001_2FFF	GPIO Unit address space.
	UART0	0x1001_3000 ~ 0x1001_3FFF	First UART address space.
	QSPI0	0x1001_4000 ~ 0x1001_4FFF	First QSPI address space.
	PWM0	0x1001_5000 ~ 0x1001_5FFF	First PWM address space.
	UART1	0x1002_3000 ~ 0x1002_3FFF	Second UART address space.
	QSPI1	0x1002_4000 ~ 0x1002_4FFF	Second QSPI address space.
	PWM1	0x1002_5000 ~ 0x1002_5FFF	Second PWM address space.
	QSPI2	0x1003_4000 ~ 0x1003_4FFF	Third QSPI address space.
	PWM2	0x1003_5000 ~ 0x1003_5FFF	Third PWM address space.
	I2C Master	0x1004_2000 ~ 0x1004_2FFF	I2C Master address space.
Default slave	The other space is write-ignored and read-as zero.		

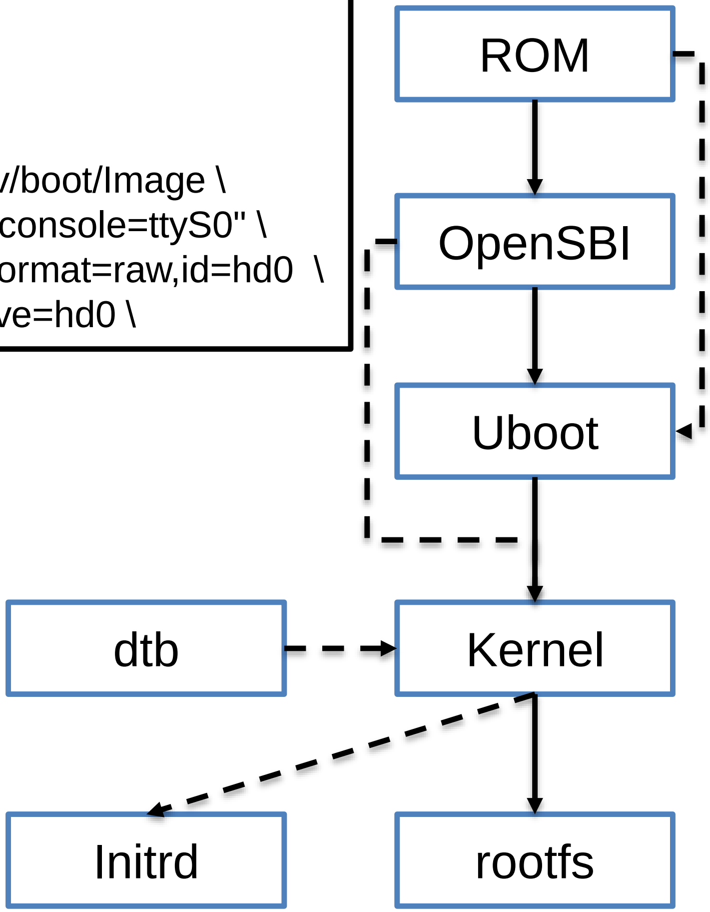
DDR:[NUCLEI_U_DRAM] = {0xa0000000, 0x0},

opensbi模式:https://github.com/riscv/opensbi/blob/master/docs/platform/qemu_virt.md 源码: https://github.com/Nuclei-Software/nuclei-linux-sdk.git

参考: https://qemu.readthedocs.io/en/latest/system/riscv/sifive_u.html

Linux 与 NUCLEI CPU

```
$qemu-system-riscv64 \
-bios none \
-M virt -nographic \
-m 2G \
-kernel linux/linux/arch/riscv/boot/Image \
-append "root=/dev/vda rw console=ttyS0" \
-drive file=linux/rootfs.img,format=raw,id=hd0  \
-device virtio-blk-device,drive=hd0 \
```



启动示例

```
$qemu-system-riscv64 -M sifive_u -smp 5 -m 2G \
    -display none -serial stdio \
    -kernel arch/riscv/boot/Image \
    -initrd /path/to/rootfs.ext4 \
    -append "root=/dev/ram"
```

```
$qemu-system-riscv64 -M sifive_u -smp 5 -m 8G \
    -display none -serial stdio \
    -kernel arch/riscv/boot/Image \
    -dtb arch/riscv/boot/dts/sifive/hifive-unleashed-a00.dtb \
    -initrd /path/to/rootfs.ext4 \
    -append "root=/dev/ram"
```

```
$qemu-system-riscv64 -M sifive_u,msel=11 -smp 5 -m 8G \
    -display none -serial stdio \
    -bios /path/to/u-boot-spl.bin \
    -drive file=/path/to/sdcard.img,if=sd
```

```
$qemu-system-riscv64 -M sifive_u,msel=6 -smp 5 -m 8G \
    -display none -serial stdio \
    -bios /path/to/u-boot-spl.bin \
    -drive file=/path/to/spi-nor.img,if=mtd
```

```
$qemu-system-riscv64 -M virt -m 256M -smp 8 \
-nographic \
-bios fw_jump.elf \
-kernel u-boot/u-boot.bin -device
loader,file=linux/arch/riscv/boot/Image,addr=0x84000000 \
-drive file=rootfs.img,format=raw,id=hd0 \
-device virtio-blk-device,drive=hd0
```

Linux 与 NUCLEI CPU

```
$qemu-system-riscv64  \
-M ddr_200t \
-nographic \
-m 256M \
-bios none \
-append "console=ttyS0 earlycon=sbi" \
-kernel fw_payload.elf \
-initrd ./rootfs_riscv64.img
```

SIM模式

```
$qemu-system-riscv64  \
-M ddr_200t \
-nographic \
-m 256M \
-bios freeloader.elf \
-drive file=/path/to/sdcard.img,if=sd
```

SD模式

OpenSBI

Kernel

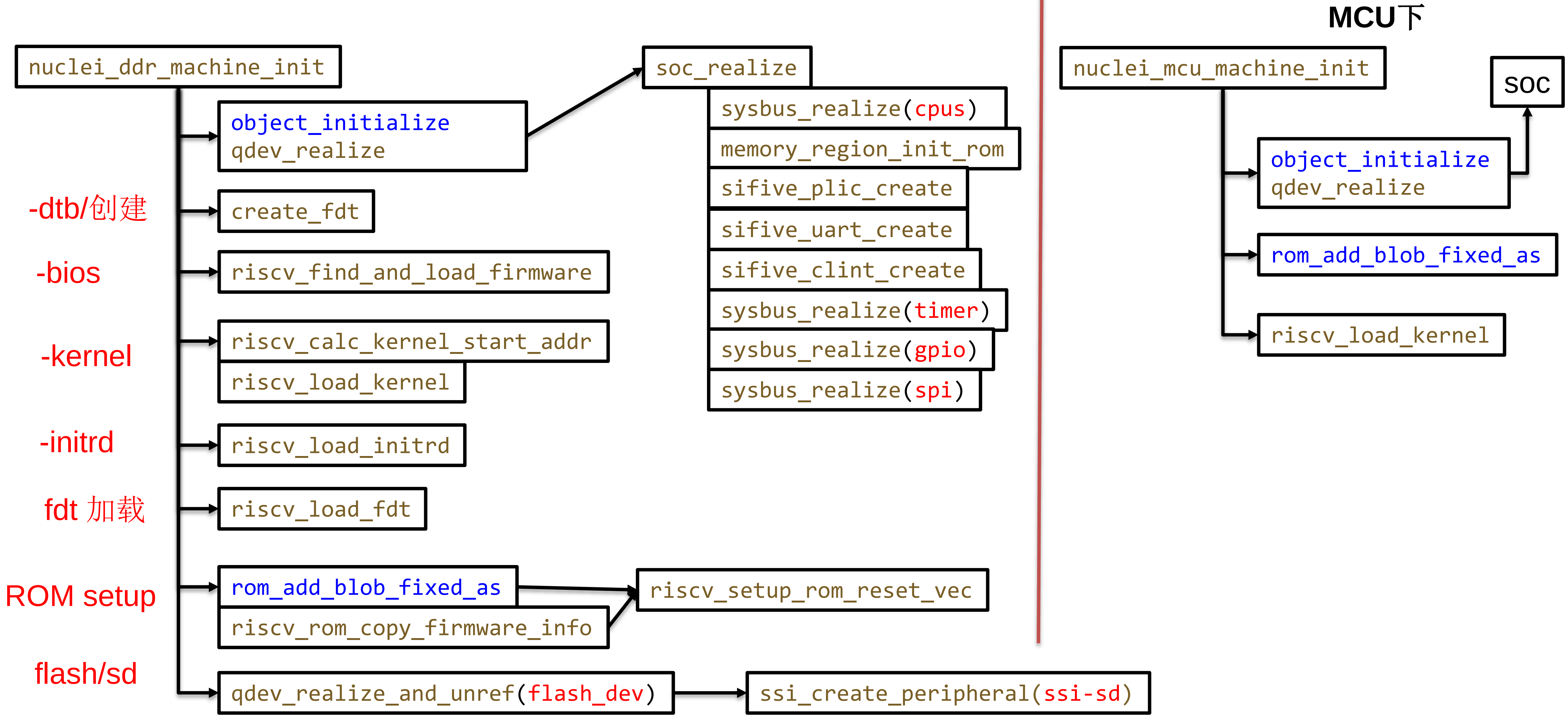
```
OpenSBI v0.9

Platform Name             : Nuclei Demo SoC
Platform Features         : timer,mfdeleg
Platform HART Count       : 1
Firmware Base             : 0xa0000000
Firmware Size             : 84 KB
Runtime SBI Version       : 0.2

Domain0 Name              : root
Domain0 Boot HART         : 0
Domain0 HARTs             : 0*
Domain0 Region00          : 0x00000000a0000000-0x00000000a001ffff ()
Domain0 Region01          : 0x0000000000000000-0xffffffffffffffff (R,W,X)
Domain0 Next Address      : 0x00000000a0200000
Domain0 Next Arg1         : 0x00000000a8000000
Domain0 Next Mode         : S-mode
Domain0 SysReset          : yes

Boot HART ID              : 0
Boot HART Domain          : root
Boot HART ISA             : rv64imafdcsu
Boot HART Features        : scounteren,mcounteren
Boot HART PMP Count       : 16
Boot HART PMP Granularity : 4
Boot HART PMP Address Bits: 54
Boot HART MHPM Count      : 0
Boot HART MHPM Count      : 0
Boot HART MIDELEG         : 0x0000000000000222
Boot HART MEDELEG         : 0x000000000000b109
[    0.000000] Linux version 5.10.0+ (wang@lelouch) (riscv-nuclei-linux-gnu-gcc (GCC) 9.2.0, GNU ld
[    0.000000] OF: fdt: Ignoring memory range 0xa0000000 - 0xa0200000
[    0.000000] earlycon: sbi0 at I/O port 0x0 (options '')
[    0.000000] printk: bootconsole [sbi0] enabled
[    0.000000] efi: UEFI not found.
[    0.000000] Zone ranges:
[    0.000000]   DMA32    [mem 0x00000000a0200000-0x00000000afffffff]
[    0.000000]   Normal   empty
```

Linux 与 NUCLEI CPU 之Machine主流程



Linux 与 NUCLEI CPU 之FDT

Device Tree

DTS (Device Tree Source)

DTC (Device Tree Compiler)

DTB (Device Tree Blob)

nuclei_rv64imafdc.dts

```
uart0: serial@10013000 {
   compatible = "nuclei,uart0";
   reg = <0x0 0x10013000 0x0 0x1000>;
   interrupt-parent = <&plic0>;
   interrupts = <33>;
   clocks = <&hfclk>;
   status = "okay";
};
```

DTS => DTB

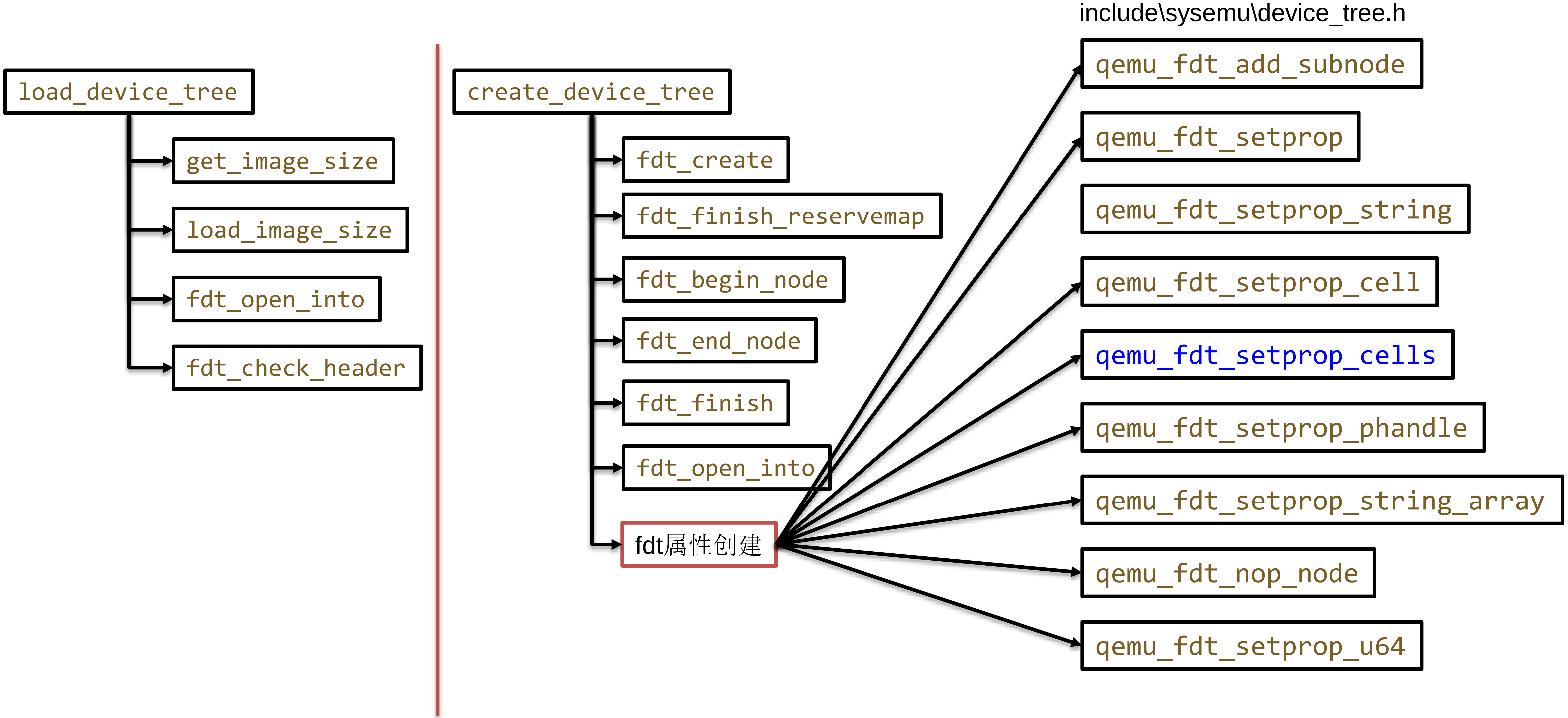
hw\riscv\nuclei_u.c

```
static void create_fdt(NucLeiUState *s, const struct MemMapEntry *memmap,
                       uint64_t mem_size, const char *cmdline)
{
    MachineState *ms = MACHINE(qdev_get_machine());
    void *fdt;
    ……
    if (ms->dtb)
    {
        fdt = s->fdt = load_device_tree(ms->dtb, &s->fdt_size);
        if (!fdt)
        {
            ……
        }
        goto update_bootargs;
    }
    else
    {
        fdt = s->fdt = create_device_tree(&s->fdt_size);
        if (!fdt)
        {
            error_report("create_device_tree() failed");
            exit(1);
        }
    }
  qemu_fdt_setprop_string(fdt, "/", "model", "nuclei,ux600");

}
```

b站: https://www.bilibili.com/video/BV1Ti4y1g7oH?from=search&seid=12813802945026575443

Linux 与 NUCLEI CPU 之FDT



Linux 与 NUCLEI CPU 之FDT

```
#address-cells = <2>;
 #size-cells = <2>;
 compatible = "nuclei,demo-soc";
 model = "nuclei,demo-soc";
```

hw\riscv\nuclei_u.c

```
    qemu_fdt_setprop_string(fdt, "/", "model", "nuclei, demo-soc");
    qemu_fdt_setprop_string(fdt, "/", "compatible",
                            "nuclei, demo-soc");
    qemu_fdt_setprop_cell(fdt, "/", "#size-cells", 0x2);
    qemu_fdt_setprop_cell(fdt, "/", "#address-cells", 0x2);
```

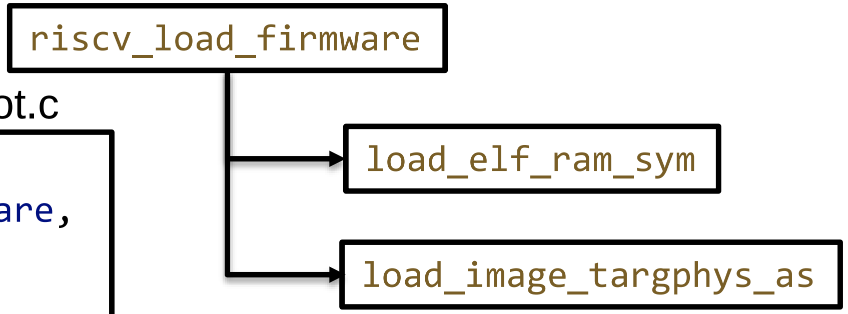
```
uart0: serial@10013000 {
   compatible = "nuclei,uart0";
   reg = <0x0 0x10013000 0x0 0x1000>;
   interrupt-parent = <&plic0>;
   interrupts = <33>;
   clocks = <&hfclk>;
   status = "okay";
 };
```

hw\riscv\nuclei_u.c

```
    uart_phandle = phandle++;
    nodename = g_strdup_printf("/serial@%lx",
                               (long)memmap[NUCLEI_U_UART0].base);
    qemu_fdt_add_subnode(fdt, nodename);
    qemu_fdt_setprop_string(fdt, nodename, "compatible", “nuclei,uart0");
    qemu_fdt_setprop_cells(fdt, nodename, "reg",
                           0x0, memmap[NUCLEI_U_UART0].base,
                           0x0, memmap[NUCLEI_U_UART0].size);
    qemu_fdt_setprop_cell(fdt, nodename, "clocks", hfclk_phandle);
    qemu_fdt_setprop_cell(fdt, nodename, "interrupt-parent", plic_phandle);
    qemu_fdt_setprop_cell(fdt, nodename, "interrupts", NUCLEI_U_UART0_IRQ);
    qemu_fdt_setprop_cell(fdt, nodename, "phandle", uart_phandle);
    qemu_fdt_setprop_string(fdt, nodename, “status”, "okay");
    g_free(nodename);
```

来自: https://github.com/Nuclei-Software/nuclei-linux-sdk/blob/dev_nuclei_next/conf/nuclei_rv64imafdc.dts

Linux 与 NUCLEI CPU 之BIOS



hw\riscv\boot.c

```c
target_ulong riscv_find_and_load_firmware(MachineState *machine,
                                          const char *default_machine_firmware,
                                          hwaddr firmware_load_addr,
                                          symbol_fn_t sym_cb)
{
    char *firmware_filename = NULL;
    target_ulong firmware_end_addr = firmware_load_addr;

    if ((!machine->firmware) || (!strcmp(machine->firmware, "default"))) {
        ……
        firmware_filename = riscv_find_firmware(default_machine_firmware);
    } else if (strcmp(machine->firmware, "none")) {
        firmware_filename = riscv_find_firmware(machine->firmware);
    }

    if (firmware_filename) {
        /* If not "none" load the firmware */
        firmware_end_addr = riscv_load_firmware(firmware_filename,
                                                firmware_load_addr, sym_cb);
        g_free(firmware_filename);
    }
    return firmware_end_addr;
}
```

opensbi

or

uboot

Linux 与 NUCLEI CPU 之BIOS

OpenSBI Platform Firmwares

- Firmware with Dynamic Information (FW_DYNAMIC)
- Firmware with Jump Address (FW_JUMP)
- Firmware with Payload (FW_PAYLOAD)

make PLATFORM=generic(nuclei/ux600)
make PLATFORM=generic(nuclei/ux600) FW_PAYLOAD_PATH=<uboot_build_directory>/u-boot.bin
make PLATFORM=generic(nuclei/ux600) FW_PAYLOAD_PATH=<linux_build_directory>/arch/riscv/boot/Image

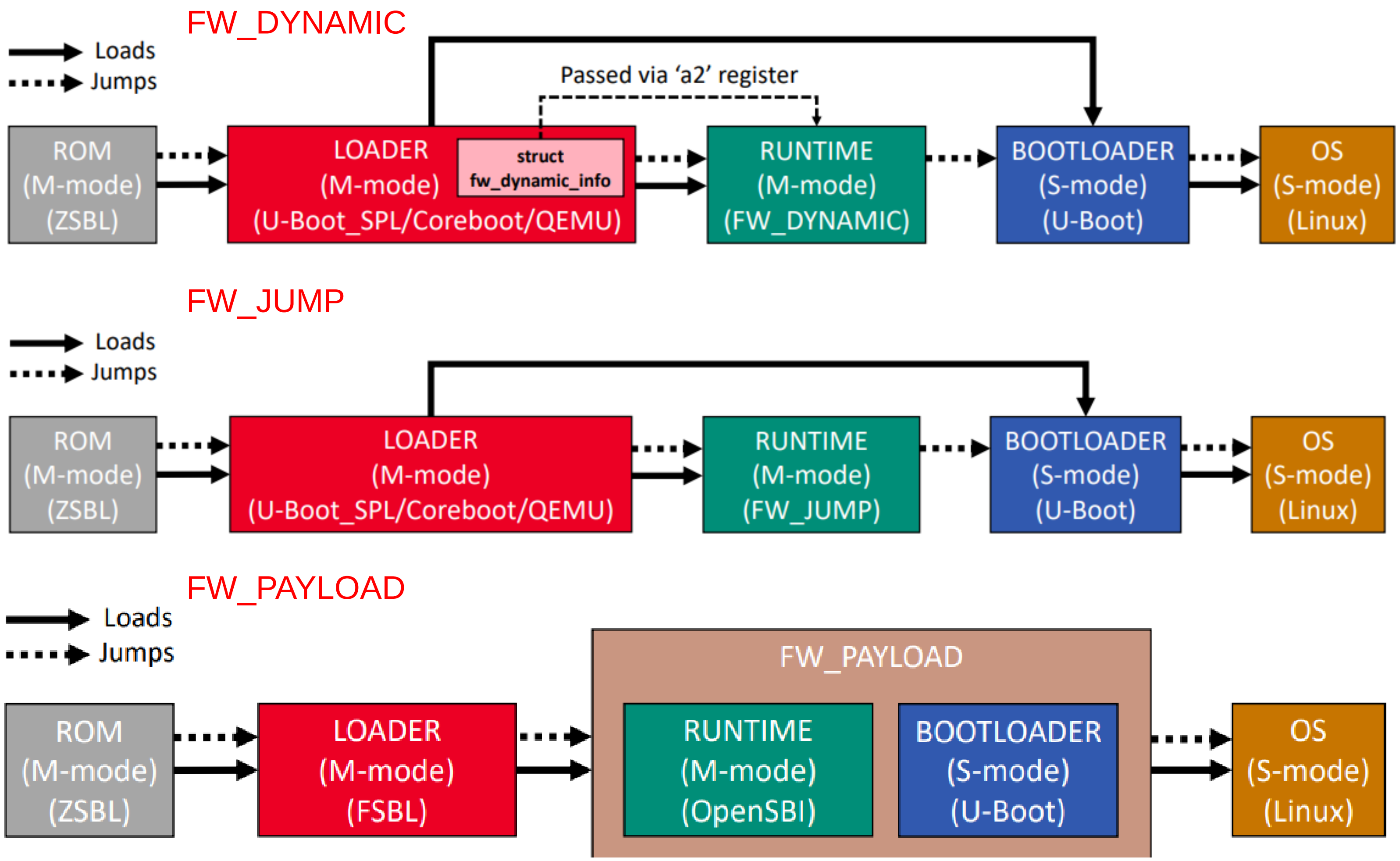
platform/generic/config.mk

```
# Blobs to build
FW_TEXT_START=0x80000000
FW_DYNAMIC=y
FW_JUMP=y
ifeq ($(PLATFORM_RISCV_XLEN), 32)
  # This needs to be 4MB aligned for 32-bit system
  FW_JUMP_ADDR=$(shell printf "0x%X" $$(($(FW_TEXT_START) + 0x400000)))
else
  # This needs to be 2MB aligned for 64-bit system
  FW_JUMP_ADDR=$(shell printf "0x%X" $$(($(FW_TEXT_START) + 0x200000)))
endif
FW_JUMP_FDT_ADDR=$(shell printf "0x%X" $$(($(FW_TEXT_START) + 0x2200000)))
FW_PAYLOAD=y
ifeq ($(PLATFORM_RISCV_XLEN), 32)
  # This needs to be 4MB aligned for 32-bit system
  FW_PAYLOAD_OFFSET=0x400000
else
  # This needs to be 2MB aligned for 64-bit system
  FW_PAYLOAD_OFFSET=0x200000
endif
FW_PAYLOAD_FDT_ADDR=$(FW_JUMP_FDT_ADDR)
```

https://github.com/riscv/riscv-sbi-doc/blob/master/riscv-sbi.adoc

https://github.com/riscv/opensbi/blob/master/docs/firmware/fw.md

Linux 与 NUCLEI CPU 之BIOS



nuclei/demosoc/config.mk

```
FW_TEXT_START ?= 0xA0000000
FW_DYNAMIC=y
FW_JUMP=y

# This needs to be 2MB aligned for 64-bit system
FW_JUMP_ADDR=
    $(……$(FW_TEXT_START) + 0x200000)))
FW_JUMP_FDT_ADDR=
    $(……$(FW_TEXT_START) + 0x8000000)))
FW_PAYLOAD=y

# This needs to be 2MB aligned for 64-bit system
FW_PAYLOAD_OFFSET=0x200000
FW_PAYLOAD_FDT_ADDR=
    $(FW_JUMP_FDT_ADDR)
```

FW_JUMP_ADDR=0xA0200000

FW_*_FDT_ADDR=0xA8000000

来自: https://riscv.org/wp-content/uploads/2019/06/13.30-RISCV_OpenSBI_Deep_Dive_v5.pdf

Linux 与 NUCLEI CPU 之Kernel

```c
kernel_entry = riscv_load_kernel(machine->kernel_filename,
                                 kernel_start_addr, NULL);
```

- load_elf_ram_sym
- load_uimage_as
- load_image_targphys_as

```c
#define QEMU_ALIGN_DOWN(n, m) ((n) / (m) * (m))
#define QEMU_ALIGN_UP(n, m) QEMU_ALIGN_DOWN((n) + (m) - 1, (m))
target_ulong riscv_calc_kernel_start_addr(RISCVHartArrayState *harts,
                                          target_ulong firmware_end_addr)

{
    if (riscv_is_32bit(harts)) {
        return QEMU_ALIGN_UP(firmware_end_addr, 4 * MiB);
    } else {
        return QEMU_ALIGN_UP(firmware_end_addr, 2 * MiB);
    }
}
```

hw\riscv\boot.c

start_addr	firmware_end_addr	kernel_start_addr	kernel_entry
Flash: 0x20000000 SD:0xa0000000 DDR:0xa0000000	由固件决定	+0x400000 +0x200000	由kernel决定

https://github.com/riscv/opensbi/blob/master/docs/firmware/fw.md

Linux 与 NUCLEI CPU 之Initrd

```
fdt_load_addr = riscv_load_fdt(memmap[NUCLEI_U_DRAM].base,
                               machine->ram_size, s->fdt);
```

```
hwaddr riscv_load_initrd(const char *filename, uint64_t mem_size,
                         uint64_t kernel_entry, hwaddr *start)
{
    int size;

    *start = kernel_entry + MIN(mem_size / 2, 128 * MiB);

    size = load_ramdisk(filename, *start, mem_size - *start);
    if (size == -1) {
        size = load_image_targphys(filename, *start, mem_size - *start);
        if (size == -1) {
            error_report("could not load ramdisk '%s'", filename);
            exit(1);
        }
    }

    return *start + size;
}
```

hw\riscv\boot.c

```
qemu_fdt_setprop_cell(s->fdt, "/chosen",
                      "linux,initrd-start", start);
qemu_fdt_setprop_cell(s->fdt, "/chosen", "linux,initrd-end",
                      end);
```

https://github.com/riscv/opensbi/blob/master/docs/firmware/fw.md

MCU ROM 加载(参加内存章节)

include/hw/loader.h

```
#define rom_add_blob_fixed_as(_f, _b, _l, _a, _as)      \
    rom_add_blob(_f, _b, _l, _l, _a, NULL, NULL, NULL, _as, true)
```

```
/* Mask ROM reset vector */
   uint32_t reset_vec[4];

   if (s->revb) {
       reset_vec[1] = 0x200102b7;  /* 0x1004: lui     t0,0x20010 */
   } else {
       reset_vec[1] = 0x204002b7;  /* 0x1004: lui     t0,0x20400 */
   }
   reset_vec[2] = 0x00028067;      /* 0x1008: jr      t0 */

   reset_vec[0] = reset_vec[3] = 0;

   /* copy in the reset vector in little_endian byte order */
   for (i = 0; i < sizeof(reset_vec) >> 2; i++) {
       reset_vec[i] = cpu_to_le32(reset_vec[i]);
   }
   rom_add_blob_fixed_as("mrom.reset", reset_vec, sizeof(reset_vec),
                          memmap[SIFIVE_E_DEV_MROM].base, &address_space_memory);

   if (machine->kernel_filename) {
       riscv_load_kernel(machine->kernel_filename,
                          memmap[SIFIVE_E_DEV_DTIM].base, NULL);
   }
```

```
/* Default Reset Vector adress */
#define DEFAULT_RSTVEC      0x1000

static void rv32_nuclei_n_cpu_init(Object *obj)
{
    ……
    set_resetvec(env, DEFAULT_RSTVEC);
    ……
}
```

```
$qemu-system-riscv32  \
-nographic -machine mcu_200t \
-kernel ./hbird/demo_eclic.elf \
-nodefaults -serial stdio
```

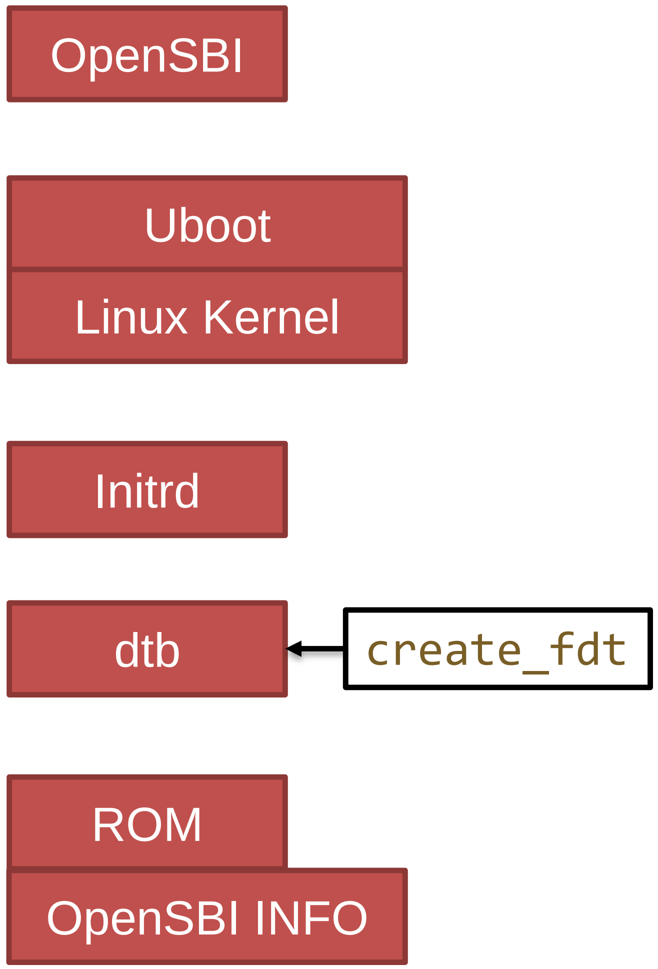
freedom-e-sdk/bsp/qemu-sifive-e31/metal.default.lds

```
MEMORY
{
   ram (airwx) : ORIGIN = 0x80000000, LENGTH = 0x400000
   rom (irx!wa) : ORIGIN = 0x20400000, LENGTH = 0x1fc00000
}
```

Linux 与 NUCLEI CPU 之ROM

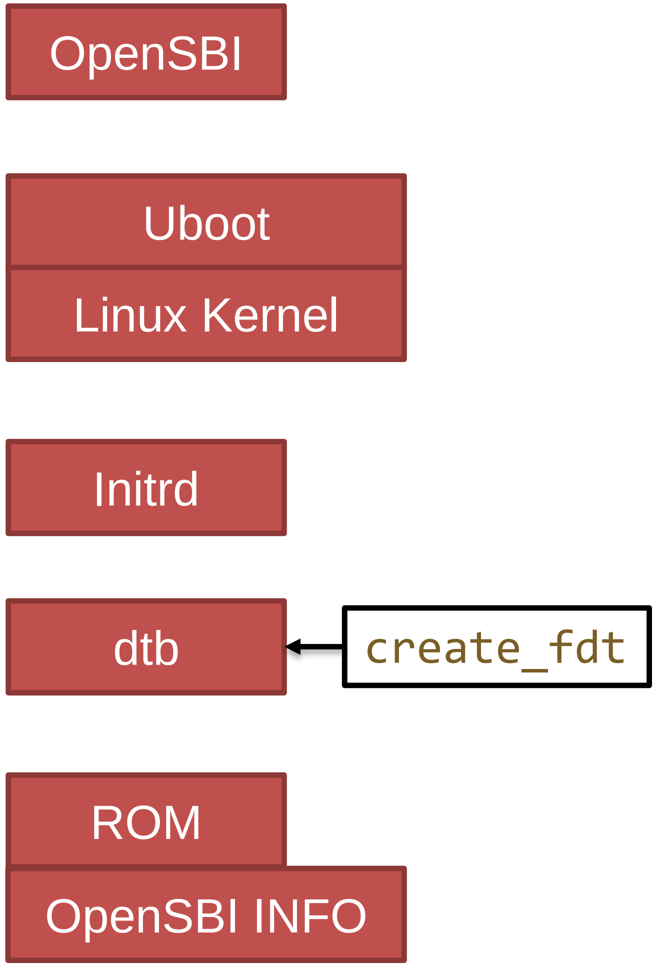
hw/riscv/nuclei_u.c：类似的ROM RAM DDR创建，不一样的加载

```
/* load the reset vector */
riscv_setup_rom_reset_vec(machine, &s->soc.cpus, start_addr,
                          memmap[NUCLEI_U_MROM].base,
                          memmap[NUCLEI_U_MROM].size, kernel_entry,
                          fdt_load_addr, s->fdt);
```



Linux 与 NUCLEI CPU 之ROM

```
    uint32_t reset_vec[10] = {
        0x00000297,                 /* 1:  auipc  t0, %pcrel_hi(fw_dyn) */
        0x02828613,                 /*     addi   a2, t0, %pcrel_lo(1b) */
        0xf1402573,                 /*     csrr   a0, mhartid  */
        0,
        0,
        0x00028067,                 /*     jr     t0 */
        start_addr,                 /* start: .dword */
        start_addr_hi32,
        fdt_load_addr,              /* fdt_laddr: .dword */
        0x00000000,
                                    /* fw_dyn: */
    };
    if (riscv_is_32bit(harts)) {
        reset_vec[3] = 0x0202a583;  /*     lw     a1, 32(t0) */
        reset_vec[4] = 0x0182a283;  /*     lw     t0, 24(t0) */
    } else {
        reset_vec[3] = 0x0202b583;  /*     ld     a1, 32(t0) */
        reset_vec[4] = 0x0182b283;  /*     ld     t0, 24(t0) */
    }
    /* copy in the reset vector in little_endian byte order */
    for (i = 0; i < ARRAY_SIZE(reset_vec); i++) {
        reset_vec[i] = cpu_to_le32(reset_vec[i]);
    }
    rom_add_blob_fixed_as("mrom.reset", reset_vec, sizeof(reset_vec),
                          rom_base, &address_space_memory);
    riscv_rom_copy_firmware_info(machine, rom_base, rom_size, sizeof(reset_vec),
                                 kernel_entry);
```



Linux 与 NUCLEI CPU 之设备(UART)

UART Overview

Address	Name	Description
0x000	txdata	Transmit data register
0x004	rxdata	Receive data register
0x008	txctrl	Transmit control register
0x00C	rxctrl	Receive control register
0x010	ie	UART interrupt enable
0x014	ip	UART Interrupt pending
0x018	div	Baud rate divisor

Table 12.1: Register offsets within UART memory map.

来自：https://static.dev.sifive.com/SiFive-E300-platform-reference-manual-v1.0.1.pdf

实现见： hw\char\sifive_uart.c

```
nuclei_u_soc_realize
```

```
sifive_uart_create(system_memory, memmap[NUCLEI_U_UART0].base,serial_hd(0), qdev_get_gpio_in(DEVICE(s->plic), NUCLEI_U_UART0_IRQ));
```

Linux 与 NUCLEI CPU 之设备(PLIC+CLINT)

```
nuclei_u_soc_realize
```

```
s->plic = sifive_plic_create(memmap[NUCLEI_U_PLIC].base,
                             plic_hart_config, 0,
                             NUCLEI_U_PLIC_NUM_SOURCES,
                             NUCLEI_U_PLIC_NUM_PRIORITIES,
                             NUCLEI_U_PLIC_PRIORITY_BASE,
                             NUCLEI_U_PLIC_PENDING_BASE,
                             NUCLEI_U_PLIC_ENABLE_BASE,
                             NUCLEI_U_PLIC_ENABLE_STRIDE,
                             NUCLEI_U_PLIC_CONTEXT_BASE,
                             NUCLEI_U_PLIC_CONTEXT_STRIDE,
                             memmap[NUCLEI_U_PLIC].size);
```

实现见:hw\intc\sifive_plic.c

```
sifive_clint_create(memmap[NUCLEI_U_CLINT].base,
                    memmap[NUCLEI_U_CLINT].size, 0, 1,
                    SIFIVE_SIP_BASE, SIFIVE_TIMECMP_BASE, SIFIVE_TIME_BASE,
                    NUCLEI_CLINT_TIMEBASE_FREQ, false);
```

实现见: hw\intc\sifive_clint.c

```
object_property_set_int(OBJECT(&s->timer), "aperture-size",
                        memmap[NUCLEI_N_TIMER].size,   &error_abort);
sysbus_realize(SYS_BUS_DEVICE(&s->timer), errp))
sysbus_mmio_map(SYS_BUS_DEVICE(&s->timer), 0, memmap[NUCLEI_U_TIMER].base);
s->timer.timebase_freq = NUCLEI_U_TIMEBASE_FREQ;
```

实现见：hw\intc\nuclei_systimer.c

Linux 与 NUCLEI CPU 之设备(SPI)

Register Name	Offset Address	Description
SPI_SCKDIV	0x00	Serial clock divisor
SPI_SCKMODE	0x04	Serial clock mode
SPI_CSID	0x10	Chip select ID
SPI_CSDEF	0x14	Chip select default
SPI_CSMODE	0x18	Chip select mode
SPI_DELAY0	0x28	Delay control 0
SPI_DELAY1	0x2C	Delay control 1
SPI_FMT	0x40	Frame format
SPI_TXDATA	0x48	TX FIFO data
SPI_RXDATA	0x4C	RX FIFO data
SPI_TXMARK	0x50	TX FIFO watermark
SPI_RXMARK	0x54	RX FIFO watermark
SPI_FCTRL	0x60	SPI flash interface control
SPI_FFMT	0x64	SPI flash instruction format
SPI_IE	0x70	SPI interrupt enable
SPI_IP	0x74	SPI interrupt pending

实现见： include\hw\ssi\sifive_spi.h
hw\ssi\sifive_spi.c

```
nuclei_u_soc_instance_init
```

```
object_initialize_child(obj, "spi0",
                        &s->spi0, TYPE_SIFIVE_SPI);
```

```
object_initialize_child(obj, "spi2",
                        &s->spi2, TYPE_SIFIVE_SPI);
```

```
nuclei_u_soc_realize
```

```
sysbus_realize(SYS_BUS_DEVICE(&s->spi0), errp);
sysbus_mmio_map(SYS_BUS_DEVICE(&s->spi0), 0,
                memmap[NUCLEI_U_SPI0_IRQ].base);
sysbus_connect_irq(SYS_BUS_DEVICE(&s->spi0), 0,
                qdev_get_gpio_in(DEVICE(s->plic),
                NUCLEI_U_SPI0_IRQ));
```

参考:《RISC-V 架构与嵌入式开发快速入门》
参考: nuclei-sdk linux nuclei spi驱动
来自: https://gitee.com/riscv-mcu/e203_hbirdv2/blob/master/doc/source/soc_peripherals/ips.rst

Linux 与 NUCLEI CPU 之设备(SPI Slave)

nuclei_ddr_machine_init

```
/* Connect an SPI flash to SPI0 */
flash_dev = qdev_new("gd25q32");
dinfo = drive_get_next(IF_MTD);
if (dinfo)
{
   qdev_prop_set_drive_err(flash_dev, "drive",
                           blk_by_legacy_dinfo(dinfo),
                           &error_fatal);
}
qdev_realize_and_unref(flash_dev, BUS(s->soc.spi0.spi), &error_fatal);

flash_cs = qdev_get_gpio_in_named(flash_dev, SSI_GPIO_CS, 0);
sysbus_connect_irq(SYS_BUS_DEVICE(&s->soc.spi0), 1, flash_cs);

/* Connect an SD card to SPI2 */
sd_dev = ssi_create_peripheral(s->soc.spi2.spi, "ssi-sd");

sd_cs = qdev_get_gpio_in_named(sd_dev, SSI_GPIO_CS, 0);
sysbus_connect_irq(SYS_BUS_DEVICE(&s->soc.spi2), 1, sd_cs);
```

Flash实现： hw\block\m25p80.c

命令：

```
$qemu-system-riscv64  \
-M ddr_200t \
-nographic \
-m 256M \
-bios freeloader.elf
-drive file=sdcard.img,if=sd
```

SD实现： hw\sd\ssi-sd.c

命令：

```
$qemu-system-riscv64  \
-M ddr_200t \
-nographic \
-m 256M \
-bios freeloader.elf
-drive file=sdcard.img,if=sd
```

SD制作: dd + mcopy

genimage + genimage_sd.cfg

ISCAS 中国科学院软件研究所 Institute of Software Chinese Academy of Sciences | 智能软件研究中心 Intelligent Software Research Center

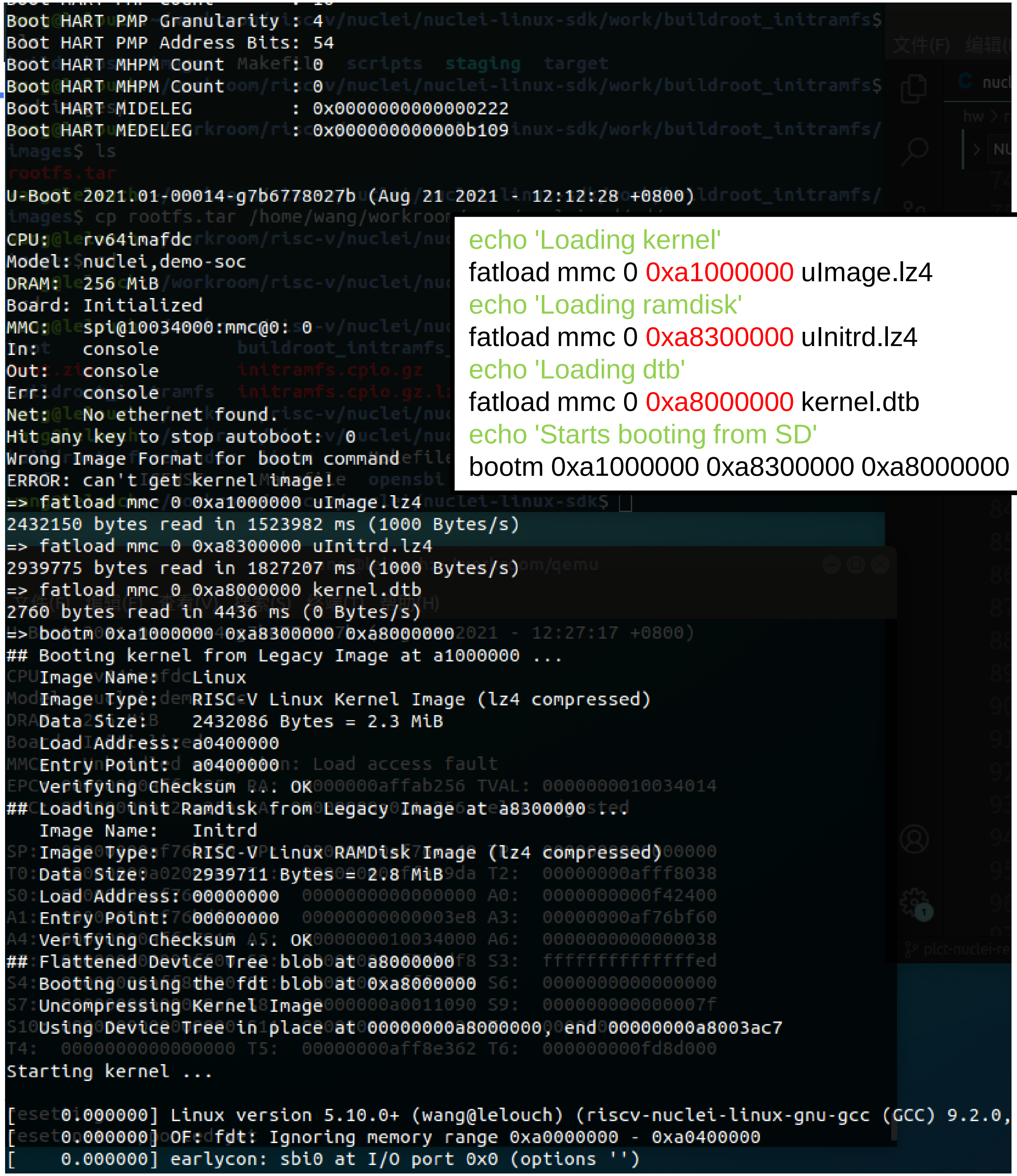
Linux 与 NUCLEI CPU

SIM模式

```
$qemu-system-riscv64  \
-M ddr_200t \
-nographic \
-m 256M \
-bios none \
-append "console=ttyS0 earlycon=sbi" \
-kernel fw_payload.elf \
-initrd ./rootfs_riscv64.img
```

SD模式

```
$qemu-system-riscv64  \
-M ddr_200t \
-nographic \
-m 256M \
-bios none \
-kernel freeloader.elf
-drive file=/path/to/sdcard.img,if=sd
```




```
echo 'Loading kernel'
fatload mmc 0 0xa1000000 uImage.lz4
echo 'Loading ramdisk'
fatload mmc 0 0xa8300000 uInitrd.lz4
echo 'Loading dtb'
fatload mmc 0 0xa8000000 kernel.dtb
echo 'Starts booting from SD'
bootm 0xa1000000 0xa8300000 0xa8000000
```

下节课内容：

外设虚拟化

- Nuclei GPIO设备实现
- Nuclei IIC设备实现
- Nuclei SPI设备实现
- Nuclei DMA设备实现与应用

谢谢

wangjunqiang@iscas.ac.cn