2.virt源码分析 | TimerのBlog

yanglianoo.github.io/2023/06/14/QEMU中自定义开发板2-virt源码分析

2023年6月14日

1. QEMU内置开发板riscv-virt硬件介绍

官方介绍文档：‘virt’ Generic Virtual Platform (virt) — QEMU documentation

riscv-virt是QEMU（Quick Emulator）中提供的一块虚拟开发板，用于模拟RISC-V架构的虚拟硬件环境。

virt提供的硬件信息如下：

- Up to 8 generic RV32GC/RV64GC cores, with optional extensions
- Core Local Interruptor (CLINT)
- Platform-Level Interrupt Controller (PLIC)
- CFI parallel NOR flash memory
- 1 NS16550 compatible UART
- 1 Google Goldfish RTC
- 1 SiFive Test device
- 8 virtio-mmio transport devices
- 1 generic PCIe host bridge
- The fw_cfg device that allows a guest to obtain data from QEMU

1. RV32GC/RV64GC处理器核心：
    - 最多支持8个通用的RV32GC或RV64GC处理器核心。
    - 可选择性添加扩展指令集。
2. Core Local Interruptor (CLINT)：
   模拟的Core Local Interruptor用于处理核心级的中断和计时器。
3. Platform-Level Interrupt Controller (PLIC)：
   模拟的Platform-Level Interrupt Controller用于处理平台级的中断，分发给各个处理器核心。
4. CFI并行NOR闪存存储器：
   模拟的CFI并行NOR闪存用于存储固件或其他数据。
5. 1个NS16550兼容串口（UART）：
   模拟的串口设备，用于与虚拟开发板进行输入输出和调试。
6. 1个Google Goldfish RTC：
   模拟的RTC（实时时钟）设备，用于提供计时和时间相关的功能。
7. 1个SiFive Test设备：
   模拟的SiFive Test设备，用于进行测试和验证。
8. 8个virtio-mmio传输设备：
   提供了8个virtio-mmio传输设备，用于与虚拟机中运行的客户操作系统进行通信。
9. 1个通用的PCIe主机桥：
   模拟的PCIe主机桥，用于支持PCIe设备的连接和通信。

10. fw_cfg设备：
    提供了fw_cfg设备，允许虚拟机从QEMU获取数据。这个设备可用于传递配置信息和数据给虚拟机。

2. virt源码分析

2.1 Kconfig

在Kconfig中关于virt的配置如下：

plaintext

```
config RISCV_VIRT
    bool
    imply PCI_DEVICES
    imply VIRTIO_VGA
    imply TEST_DEVICES
    imply TPM_TIS_SYSBUS
    select RISCV_NUMA
    select GOLDFISH_RTC
    select PCI
    select
PCI_EXPRESS_GENERIC_BRIDGE
    select PFLASH_CFI01
    select SERIAL
    select RISCV_ACLINT
    select RISCV_APLIC
    select RISCV_IMSIC
    select SIFIVE_PLIC
    select SIFIVE_TEST
    select VIRTIO_MMIO
    select FW_CFG_DMA
    select PLATFORM_BUS
    select ACPI
```

select：select关键字表示启用RISCV_VIRT选项时，会自动选择（即启用）其他相关选项。

- `select RISCV_NUMA`：选择RISCV_NUMA选项，表示虚拟机将支持RISC-V架构的NUMA（非一致性存储访问）特性。
- `select GOLDFISH_RTC`：选择GOLDFISH_RTC选项，表示虚拟机将支持Google Goldfish RTC设备，用于提供计时和时间相关的功能。
- `select PCI`：选择PCI选项，表示虚拟机将支持PCI（Peripheral Component Interconnect）总线。
- `select PCI_EXPRESS_GENERIC_BRIDGE`：选择PCI_EXPRESS_GENERIC_BRIDGE选项，表示虚拟机将支持通用PCI Express桥接器。
- `select PFLASH_CFI01`：选择PFLASH_CFI01选项，表示虚拟机将支持CFI（Common Flash Interface）规范的并行 NOR Flash 存储器。
- `select SERIAL`：选择SERIAL选项，表示虚拟机将支持串口设备，用于输入输出和调试。

- `select RISCV_ACLINT`：选择RISCV_ACLINT选项，表示虚拟机将支持ACLINT（Architectural Core Local Interruptor）设备，用于处理核心级的中断和计时器。
- `select RISCV_APLIC`：选择RISCV_APLIC选项，表示虚拟机将支持APLIC（Architectural Platform-Level Interrupt Controller）设备，用于处理平台级的中断。
- `select RISCV_IMSIC`：选择RISCV_IMSIC选项，表示虚拟机将支持IMSIC（Interrupt-Management Standardized Interface Controller）设备。
- `select SIFIVE_PLIC`：选择SIFIVE_PLIC选项，表示虚拟机将支持SiFive PLIC（Platform-Level Interrupt Controller）设备。
- `select SIFIVE_TEST`：选择SIFIVE_TEST选项，表示虚拟机将支持SiFive Test设备，用于测试和验证。
- `select VIRTIO_MMIO`：选择VIRTIO_MMIO选项，表示虚拟机将支持VirtIO MMIO传输设备，用于与客户操作系统进行通信。
- `select FW_CFG_DMA`：选择FW_CFG_DMA选项，表示虚拟机将支持DMA（Direct Memory Access）传输，用于从QEMU获取数据。
- `select PLATFORM_BUS`：选择PLATFORM_BUS选项，表示虚拟机将支持平台总线，用作硬件组件之间的通信接口。
- `select ACPI`：选择ACPI选项，表示虚拟机将支持ACPI（Advanced Configuration and Power Interface）标准，用于管理系统配置和电源管理。

2.2 virt.h

2.2.1 `DECLARE_INSTANCE_CHECKER`

c

```
#define TYPE_RISCV_VIRT_MACHINE MACHINE_TYPE_NAME("virt")
typedef struct RISCVVirtState RISCVVirtState;   //结构体前
向声明
DECLARE_INSTANCE_CHECKER(RISCVVirtState,
RISCV_VIRT_MACHINE,
                         TYPE_RISCV_VIRT_MACHINE)
```

首先调用`DECLARE_INSTANCE_CHECKER`这个宏，这个宏在qemu源码中的定义如下：

c

```
#define DECLARE_INSTANCE_CHECKER(InstanceType, OBJ_NAME,
TYPENAME) \
    static inline G_GNUC_UNUSED InstanceType * \
    OBJ_NAME(const void *obj) \
    { return OBJECT_CHECK(InstanceType, obj, TYPENAME); }
```

`DECLARE_INSTANCE_CHECKER`宏用于为QOM（QEMU Object Model）类型提供实例类型转换函数。它接受三个参数：`InstanceType`表示实例结构体的名称，`OBJ_NAME`表示以大写字母和下划线分隔的对象名称，`TYPENAME`表示类型名称。

因此这里就是用来将RISCVVirtState转换为一个QOM类型的结构体，后续需要去定义这个结构体，经过这一步可以认为声明了一个名交virt的riscv板子。

2.2.2 RISCVVirtState定义

c

```
typedef enum RISCVVirtAIAType {
    VIRT_AIA_TYPE_NONE = 0,
    VIRT_AIA_TYPE_APLIC,
    VIRT_AIA_TYPE_APLIC_IMSIC,
} RISCVVirtAIAType;

struct RISCVVirtState {
    /*< private >*/
    MachineState parent;    //继承
MachineState

    /*< public >*/
    Notifier machine_done;  //
    DeviceState *platform_bus_dev;
    RISCVHartArrayState
soc[VIRT_SOCKETS_MAX];
    DeviceState
*irqchip[VIRT_SOCKETS_MAX];
    PFlashCFI01 *flash[2];   //flash
    FWCfgState *fw_cfg;      //

    int fdt_size;
    bool have_aclint;
    RISCVVirtAIAType aia_type;
    int aia_guests;
    char *oem_id;
    char *oem_table_id;
    OnOffAuto acpi;
    const MemMapEntry *memmap; //内存映射
};
```

在RISCVVirtState这个结构体中，定义了很多其他设备和成员变量，在qemu中各个硬件都是虚拟的，可以将一种硬件看作是一个结构体类，virt这块板子有哪些硬件就要在RISCVVirtState这个本体结构体中去包含这些属性，和硬件相关的设备定义基本都在hw/目录下。

2.2.3 枚举变量

c

```
enum {
    VIRT_DEBUG,
    VIRT_MROM,
    VIRT_TEST,
    VIRT_RTC,
    VIRT_CLINT,
    VIRT_ACLINT_SSWI,
    VIRT_PLIC,
    VIRT_APLIC_M,
    VIRT_APLIC_S,
    VIRT_UART0,
    VIRT_VIRTIO,
    VIRT_FW_CFG,
    VIRT_IMSIC_M,
    VIRT_IMSIC_S,
    VIRT_FLASH,
    VIRT_DRAM,
    VIRT_PCIE_MMIO,
    VIRT_PCIE_PIO,
    VIRT_PLATFORM_BUS,
    VIRT_PCIE_ECAM
};

enum {
    UART0_IRQ = 10, //串口中断号
    RTC_IRQ = 11,   // RTC 中断号
    VIRTIO_IRQ = 1, /* 1 to 8 */
    VIRTIO_COUNT = 8,
    PCIE_IRQ = 0x20, /* 32 to 35 */
    VIRT_PLATFORM_BUS_IRQ = 64, /* 64 to
95 */
};
```

这里就是使用枚举定义了一些和硬件、中断号相关的变量。

2.2.4 宏定义

c

```
#define VIRT_PLATFORM_BUS_NUM_IRQS 32

#define VIRT_IRQCHIP_NUM_MSIS 255
#define VIRT_IRQCHIP_NUM_SOURCES 96
#define VIRT_IRQCHIP_NUM_PRIO_BITS 3
#define VIRT_IRQCHIP_MAX_GUESTS_BITS 3
#define VIRT_IRQCHIP_MAX_GUESTS ((1U << VIRT_IRQCHIP_MAX_GUESTS_BITS)
- 1U)

#define VIRT_PLIC_PRIORITY_BASE 0x00
#define VIRT_PLIC_PENDING_BASE 0x1000
#define VIRT_PLIC_ENABLE_BASE 0x2000
#define VIRT_PLIC_ENABLE_STRIDE 0x80
#define VIRT_PLIC_CONTEXT_BASE 0x200000
#define VIRT_PLIC_CONTEXT_STRIDE 0x1000
#define VIRT_PLIC_SIZE(__num_context) \
    (VIRT_PLIC_CONTEXT_BASE + (__num_context) *
VIRT_PLIC_CONTEXT_STRIDE)

#define FDT_PCI_ADDR_CELLS    3
#define FDT_PCI_INT_CELLS     1
#define FDT_PLIC_ADDR_CELLS   0
#define FDT_PLIC_INT_CELLS    1
#define FDT_APLIC_INT_CELLS   2
#define FDT_IMSIC_INT_CELLS   0
#define FDT_MAX_INT_CELLS     2
#define FDT_MAX_INT_MAP_WIDTH (FDT_PCI_ADDR_CELLS + FDT_PCI_INT_CELLS
+ \
                               1 + FDT_MAX_INT_CELLS)
#define FDT_PLIC_INT_MAP_WIDTH  (FDT_PCI_ADDR_CELLS +
FDT_PCI_INT_CELLS + \
                               1 + FDT_PLIC_INT_CELLS)
#define FDT_APLIC_INT_MAP_WIDTH (FDT_PCI_ADDR_CELLS +
FDT_PCI_INT_CELLS + \
                               1 + FDT_APLIC_INT_CELLS)
```

2.3 virt.c

2.3.1 硬件内存映射定义

plaintext

```
static const MemMapEntry virt_memmap[] = {
    [VIRT_DEBUG] =        {        0x0,         0x100 },
    [VIRT_MROM] =         {     0x1000,        0xf000 },
    [VIRT_TEST] =         {   0x100000,        0x1000 },
    [VIRT_RTC] =          {   0x101000,        0x1000 },
    [VIRT_CLINT] =        {  0x2000000,       0x10000 },
    [VIRT_ACLINT_SSWI] =  {  0x2F00000,        0x4000 },
    [VIRT_PCIE_PIO] =     {  0x3000000,       0x10000 },
    [VIRT_PLATFORM_BUS] = {  0x4000000,     0x2000000 },
    [VIRT_PLIC] =         {  0xc000000, VIRT_PLIC_SIZE(VIRT_CPUS_MAX *
2) },
    [VIRT_APLIC_M] =      {  0xc000000, APLIC_SIZE(VIRT_CPUS_MAX) },
    [VIRT_APLIC_S] =      {  0xd000000, APLIC_SIZE(VIRT_CPUS_MAX) },
    [VIRT_UART0] =        { 0x10000000,         0x100 },
    [VIRT_VIRTIO] =       { 0x10001000,        0x1000 },
    [VIRT_FW_CFG] =       { 0x10100000,          0x18 },
    [VIRT_FLASH] =        { 0x20000000,     0x4000000 },
    [VIRT_IMSIC_M] =      { 0x24000000, VIRT_IMSIC_MAX_SIZE },
    [VIRT_IMSIC_S] =      { 0x28000000, VIRT_IMSIC_MAX_SIZE },
    [VIRT_PCIE_ECAM] =    { 0x30000000,    0x10000000 },
    [VIRT_PCIE_MMIO] =    { 0x40000000,    0x40000000 },
    [VIRT_DRAM] =         { 0x80000000,           0x0 },
};
```

MemMapEntry 结构体定义如下

c

```
typedef struct
MemMapEntry {
    hwaddr base;
    hwaddr size;
} MemMapEntry;
```

可以看到上面内存映射的第一个参数为硬件映射的地址，第二个参数为映射的内存长度。

2.3.2 注册virt机器

在virt.c代码的最下方可以看到注册相关的代码，首先需要定义一个`virt_machine_typeinfo`，然后调用`type_register_static`函数，最后调用 type_init这个宏。

c

```
static const TypeInfo virt_machine_typeinfo = {
    .name       = MACHINE_TYPE_NAME("virt"),  //注册板卡，定义
名称
    .parent     = TYPE_MACHINE,                             //父
类名称
    .class_init = virt_machine_class_init,    //
    .instance_init = virt_machine_instance_init,
    .instance_size = sizeof(RISCVVirtState),
    .interfaces = (InterfaceInfo[]) {
         { TYPE_HOTPLUG_HANDLER },
         { }
    },
};

static void virt_machine_init_register_types(void)
{
    type_register_static(&virt_machine_typeinfo);
}

type_init(virt_machine_init_register_types)
```

这里有几个重要的成员变量：

- @class_init: 函数指针，该函数在所有父类初始化完成后调用，允许类设置其默认的虚拟方法指针。也可以使用该函数覆盖父类的虚拟方法。
- @instance_init: 函数指针，该函数用于初始化一个对象。父类已经完成初始化，因此该类型只需负责初始化自己的成员。
- @instance_size: 对象的大小（派生自 #Object）。如果 @instance_size 为 0，则对象的大小将为父对象的大小。
- @interfaces: 与该类型关联的接口列表。应指向以零填充的静态数组作为终止元素。

在qemu中想要定义自己的板卡是采用类似C++中继承的方式来实现的，在virt.h中可以认为`RISCVVirtState`为一个子类，继承了qemu中的统一的machine的父类，我们需要在子类中来创建属于自己的硬件。所以下一步就需要来定义`virt_machine_class_init`和`virt_machine_instance_init`这两个函数。

2.3.3 `virt_machine_class_init`函数定义

c

```
static void virt_machine_class_init(ObjectClass *oc, void *data)
{
    char str[128];
    MachineClass *mc = MACHINE_CLASS(oc);
    HotplugHandlerClass *hc = HOTPLUG_HANDLER_CLASS(oc);

    mc->desc = "RISC-V VirtIO board";
    mc->init = virt_machine_init;
    mc->max_cpus = VIRT_CPUS_MAX;
    mc->default_cpu_type = TYPE_RISCV_CPU_BASE;
    mc->pci_allow_0_address = true;
    mc->possible_cpu_arch_ids = riscv_numa_possible_cpu_arch_ids;
    mc->cpu_index_to_instance_props = riscv_numa_cpu_index_to_props;
    mc->get_default_cpu_node_id = riscv_numa_get_default_cpu_node_id;
    mc->numa_mem_supported = true;
    mc->default_ram_id = "riscv_virt_board.ram";
    assert(!mc->get_hotplug_handler);
    mc->get_hotplug_handler = virt_machine_get_hotplug_handler;

    hc->plug = virt_machine_device_plug_cb;

    machine_class_allow_dynamic_sysbus_dev(mc, TYPE_RAMFB_DEVICE);
#ifdef CONFIG_TPM
    machine_class_allow_dynamic_sysbus_dev(mc, TYPE_TPM_TIS_SYSBUS);
#endif

    object_class_property_add_bool(oc, "aclint", virt_get_aclint,
                                   virt_set_aclint);
    object_class_property_set_description(oc, "aclint",
                                          "Set on/off to enable/disable
"
                                          "emulating ACLINT devices");

    object_class_property_add_str(oc, "aia", virt_get_aia,
                                  virt_set_aia);
    object_class_property_set_description(oc, "aia",
                                          "Set type of AIA interrupt "
                                          "conttoller. Valid values are
"
                                          "none, aplic, and aplic-
imsic.");

    object_class_property_add_str(oc, "aia-guests",
                                  virt_get_aia_guests,
                                  virt_set_aia_guests);
    sprintf(str, "Set number of guest MMIO pages for AIA IMSIC. Valid
value "
                 "should be between 0 and %d.",
VIRT_IRQCHIP_MAX_GUESTS);
    object_class_property_set_description(oc, "aia-guests", str);
    object_class_property_add(oc, "acpi", "OnOffAuto",
                              virt_get_acpi, virt_set_acpi,
                              NULL, NULL);
    object_class_property_set_description(oc, "acpi",
                                          "Enable ACPI");
}
```

2.3.4 virt创建CPU

2.3.5 virt创建PLIC

2.3.6 virt创建ACLINT

相关推荐