4.为quard-star添加中断控制器 | TimerのBlog

yanglianoo.github.io/2023/06/16/QEMU自定义开发板4-为quard-star添加中断控制器

2023年6月16日

1.quard_star.h 修改

```
struct QuardStarState {
    /*< private >*/
    MachineState parent;

    /*< public >*/
    RISCVHartArrayState soc[QUARD_STAR_SOCKETS_MAX];
    PFlashCFI01 *flash;
    DeviceState *plic[QUARD_STAR_SOCKETS_MAX];
};

enum {
    QUARD_STAR_MROM,
    QUARD_STAR_SRAM,
    QUARD_STAR_CLINT,
    QUARD_STAR_PLIC,
    QUARD_STAR_UART0,
    QUARD_STAR_FLASH,
    QUARD_STAR_DRAM,
};

enum {
    QUARD_STAR_UART0_IRQ = 10,  //定义了串口中断号为10
};

#define QUARD_STAR_PLIC_NUM_SOURCES    127      //PLIC 支持的中断源的最大数量
#define QUARD_STAR_PLIC_NUM_PRIORITIES 7        //PLIC 支持的中断优先级的数量
#define QUARD_STAR_PLIC_PRIORITY_BASE  0x04     //PLIC 中断优先级寄存器的基址偏移值，用于访问中断优先级信息
#define QUARD_STAR_PLIC_PENDING_BASE   0x1000   //PLIC 中断挂起寄存器的基址偏移值，用于访问中断挂起状态
#define QUARD_STAR_PLIC_ENABLE_BASE    0x2000   //PLIC 中断使能寄存器的基址偏移值，用于控制中断使能状态
#define QUARD_STAR_PLIC_ENABLE_STRIDE  0x80     //PLIC 中断使能寄存器之间的地址间隔
#define QUARD_STAR_PLIC_CONTEXT_BASE   0x200000 //PLIC 上下文保存寄存器的基址偏移值，用于保存中断处理程序的上下文信息
#define QUARD_STAR_PLIC_CONTEXT_STRIDE 0x1000   //PLIC 上下文保存寄存器之间的地址间隔
```

- 在QuardStarState定义了一个plic设备，为一个数组，因为有8个CPU
- 定义了一些与riscv中plic相关的信息

2.quard_star.c 修改

修改内存映射数组，新增PLIC和CLINT

```
static const MemMapEntry quard_star_memmap[] = {
    [QUARD_STAR_MROM]  =  {         0x0,         0x8000 },
    [QUARD_STAR_SRAM]  =  {      0x8000,         0x8000 },
    [QUARD_STAR_CLINT] =  { 0x02000000,        0x10000 },
    [QUARD_STAR_PLIC]  =  { 0x0c000000,      0x4000000 },
    [QUARD_STAR_UART0] =  { 0x10000000,          0x100 },
    [QUARD_STAR_FLASH] =  { 0x20000000,      0x2000000 },
    [QUARD_STAR_DRAM]  =  { 0x80000000,           0x80 },
};
```

新建quard_star_plic_create函数，plic的创建和virt.c中保持一致，virt.c中创建plic的源码如下，调用了virt_create_plic函数，使用的硬件为SIFIVE_PLIC

```c
    /* Per-socket interrupt controller */
    if (s->aia_type == VIRT_AIA_TYPE_NONE) {
        s->irqchip[i] = virt_create_plic(memmap, i,
                                         base_hartid, hart_count);
    } else {
        s->irqchip[i] = virt_create_aia(s->aia_type, s->aia_guests,
                                        memmap, i, base_hartid,
                                        hart_count);
    }
```

c

```c
/* 创建plic */
static void quard_star_plic_create(MachineState *machine)
{
    int socket_count = riscv_socket_count(machine);
    QuardStarState *s = RISCV_VIRT_MACHINE(machine);
    int i,hart_count,base_hartid;
    for ( i = 0; i < socket_count; i++) {

        hart_count = riscv_socket_hart_count(machine, i);
        base_hartid = riscv_socket_first_hartid(machine, i);
        char *plic_hart_config;
        /* Per-socket PLIC hart topology configuration string */
        plic_hart_config = riscv_plic_hart_config_string(machine->smp.cpus);

        s->plic[i] = sifive_plic_create(
            quard_star_memmap[QUARD_STAR_PLIC].base + i
*quard_star_memmap[QUARD_STAR_PLIC].size ,
            plic_hart_config, hart_count , base_hartid,
            QUARD_STAR_PLIC_NUM_SOURCES,
            QUARD_STAR_PLIC_NUM_PRIORITIES,
            QUARD_STAR_PLIC_PRIORITY_BASE,
            QUARD_STAR_PLIC_PENDING_BASE,
            QUARD_STAR_PLIC_ENABLE_BASE,
            QUARD_STAR_PLIC_ENABLE_STRIDE,
            QUARD_STAR_PLIC_CONTEXT_BASE,
            QUARD_STAR_PLIC_CONTEXT_STRIDE,
            quard_star_memmap[QUARD_STAR_PLIC].size);
        g_free(plic_hart_config);
    }
}
```

新增`quard_star_aclint_create`函数，相比于virt简化了许多，使用的硬件是`SiFive CLINT`

c

```
static void quard_star_aclint_create(MachineState *machine)
{
    int i , hart_count,base_hartid;
    int socket_count = riscv_socket_count(machine);
    //每个CPU都需要创建 aclint
    for ( i = 0; i < socket_count; i++) {

        base_hartid = riscv_socket_first_hartid(machine, i);
        hart_count = riscv_socket_hart_count(machine, i);

        riscv_aclint_swi_create(
        quard_star_memmap[QUARD_STAR_CLINT].base + i
*quard_star_memmap[QUARD_STAR_CLINT].size,
        base_hartid, hart_count, false);
        riscv_aclint_mtimer_create(quard_star_memmap[QUARD_STAR_CLINT].base +
             + i *quard_star_memmap[QUARD_STAR_CLINT].size+ RISCV_ACLINT_SWI_SIZE,
            RISCV_ACLINT_DEFAULT_MTIMER_SIZE, base_hartid, hart_count,
            RISCV_ACLINT_DEFAULT_MTIMECMP, RISCV_ACLINT_DEFAULT_MTIME,
            RISCV_ACLINT_DEFAULT_TIMEBASE_FREQ, true);
    }
}
```

`machine_init`函数修改，新增两个硬件

```
/* quard-star 初始化各种硬件 */
static void quard_star_machine_init(MachineState *machine)
{
    //创建CPU
    quard_star_cpu_create(machine);
   // 创建主存
    quard_star_memory_create(machine);
   //创建flash
    quard_star_flash_create(machine);
    //创建PLIC
    quard_star_plic_create(machine);
    //创建RISCV_ACLINT
    quard_star_aclint_create(machine);
}
```

3. Kconfig修改

新增了ACLINT和PLIC设备

c

```
config QUARD_STAR
    bool
    select SERIAL
    select PFLASH_CFI01
//Pflash
    select RISCV_ACLINT
//ACLINT
    select RISCV_APLIC  //APLIC
    select SIFIVE_PLIC
//SIFIVE_PLIC
```

4. 测试

在进行测试时可将`quard_star_flash_create`先注释掉，一个个硬件依次测试。

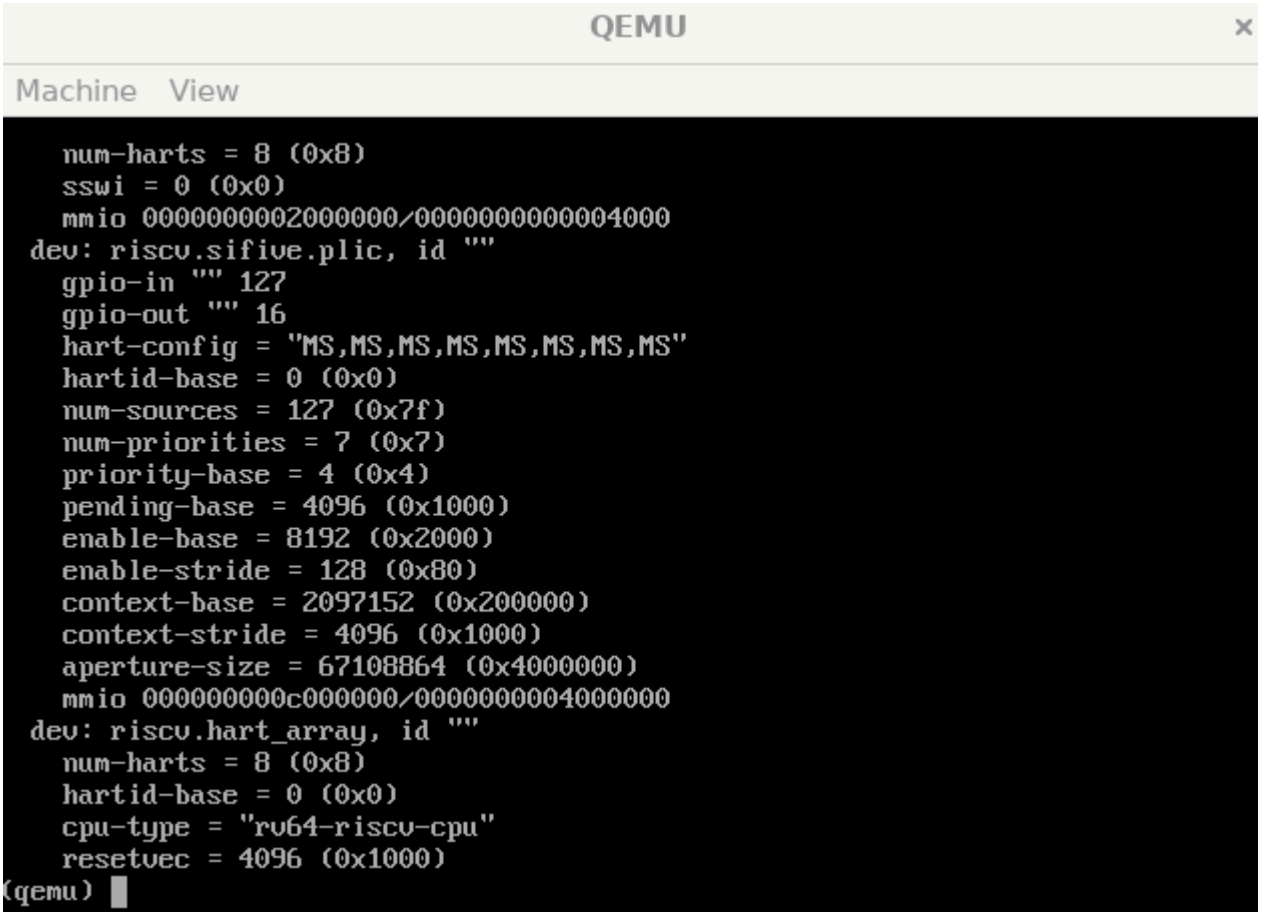
```
/* quard-star 初始化各种硬件 */
static void quard_star_machine_init(MachineState *machine)
{
    //创建CPU
    quard_star_cpu_create(machine);
   // 创建主存
    quard_star_memory_create(machine);
   //创建flash
   // quard_star_flash_create(machine);
    //创建PLIC
    quard_star_plic_create(machine);
    //创建RISCV_ACLINT
    quard_star_aclint_create(machine);
}
```

运行脚本：

plaintext

```
timer@DESKTOP-JI9EVEH:~/quard-star$
./build.sh
timer@DESKTOP-JI9EVEH:~/quard-star$
./run.sh
```

如下中断控制器添加成功：



> 代码地址：yanglianoo/quard-star: 从零基于qemu创建riscv嵌入式开发板，并移植操作系统 (github.com)
>
> 有问题请与我联系：wechat：13699648817