从零开始的RISC-V模拟器开发

第12讲 QEMU篇之中断虚拟化2

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

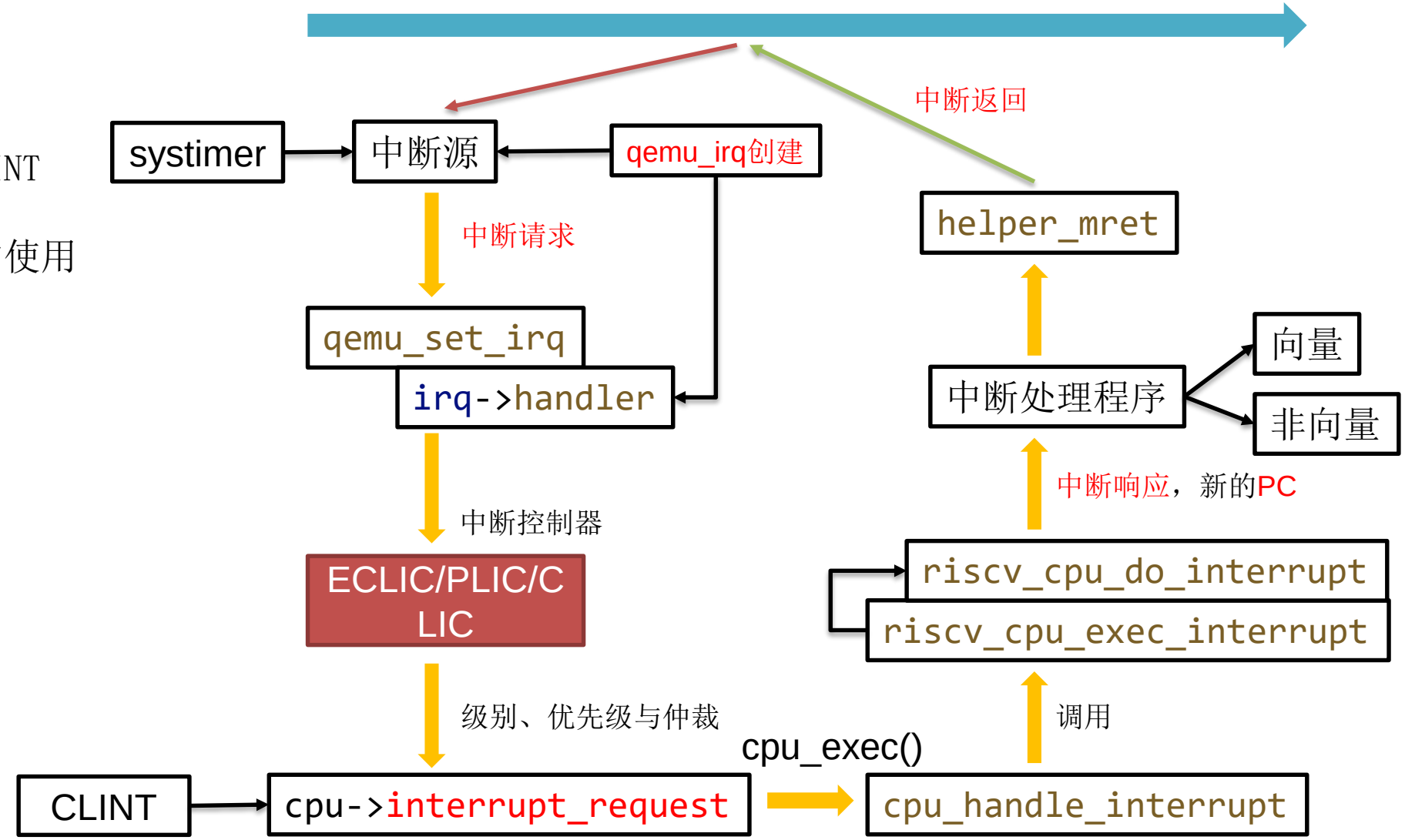
中断虚拟化

- ECLIC 与 CLIC
- SYSTIMER 与 CLINT
- PLIC介绍
- ECLIC在UART中的使用

本课内容

中断虚拟化

- ECLIC 与 CLIC
- SYSTIMER 与 CLINT
- PLIC介绍
- ECLIC在UART中的使用



中断虚拟化

ECLIC(Enhanced Core Local Interrupt Controller)

参考：https://doc.nucleisys.com/nuclei_spec/isa/eclic.html

CLIC(Core-Local Interrupt Controller)

参考: https://github.com/riscv/riscv-fast-interrupt

SYSTIMER(Timer Unit)

参考: https://doc.nucleisys.com/nuclei_spec/isa/timer.html

CLINT(Core Local Interrupt Controller)

参考: https://www.sifive.com/documentation

PLIC(Platform-Level Interrupt Controller)

参考: https://github.com/riscv/riscv-plic-spec

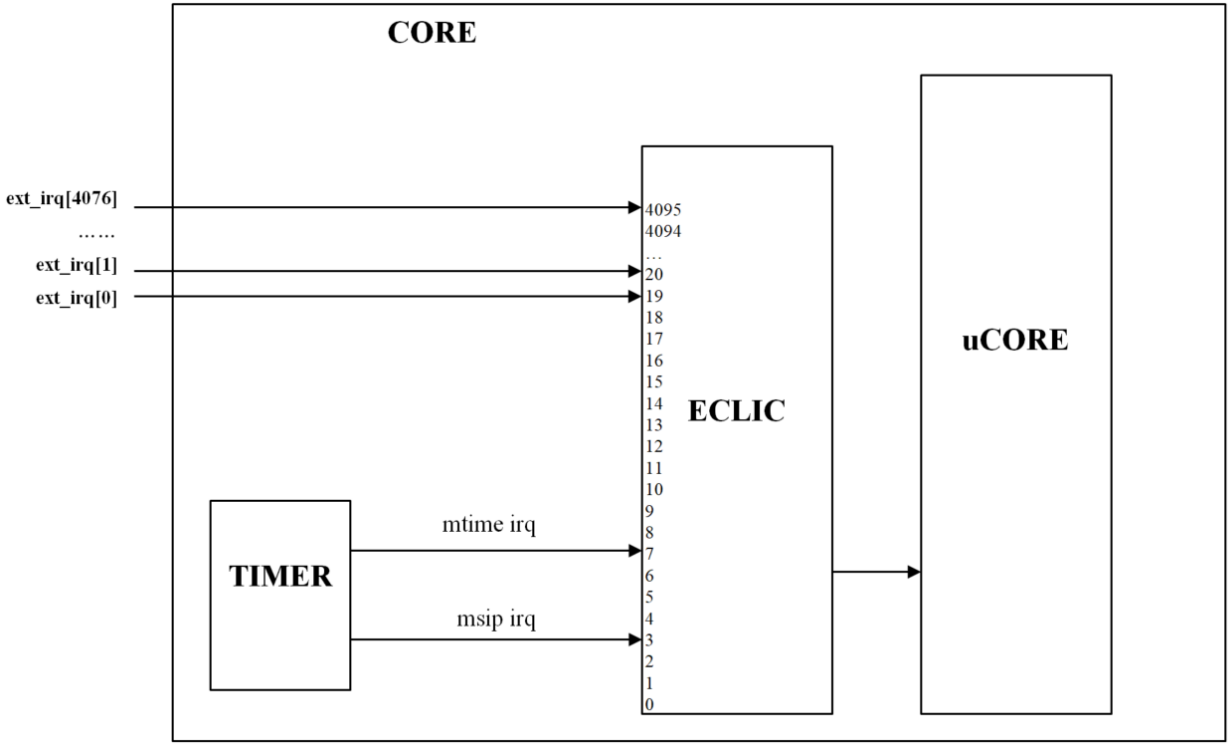
nuclei N600
nuclei NX600

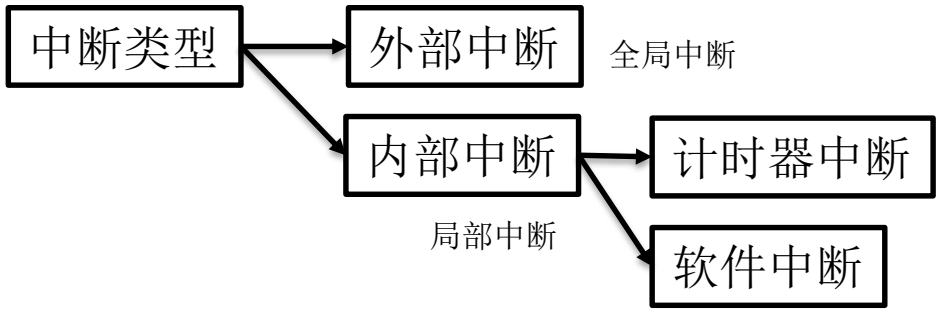
nuclei UX600
nuclei UX900

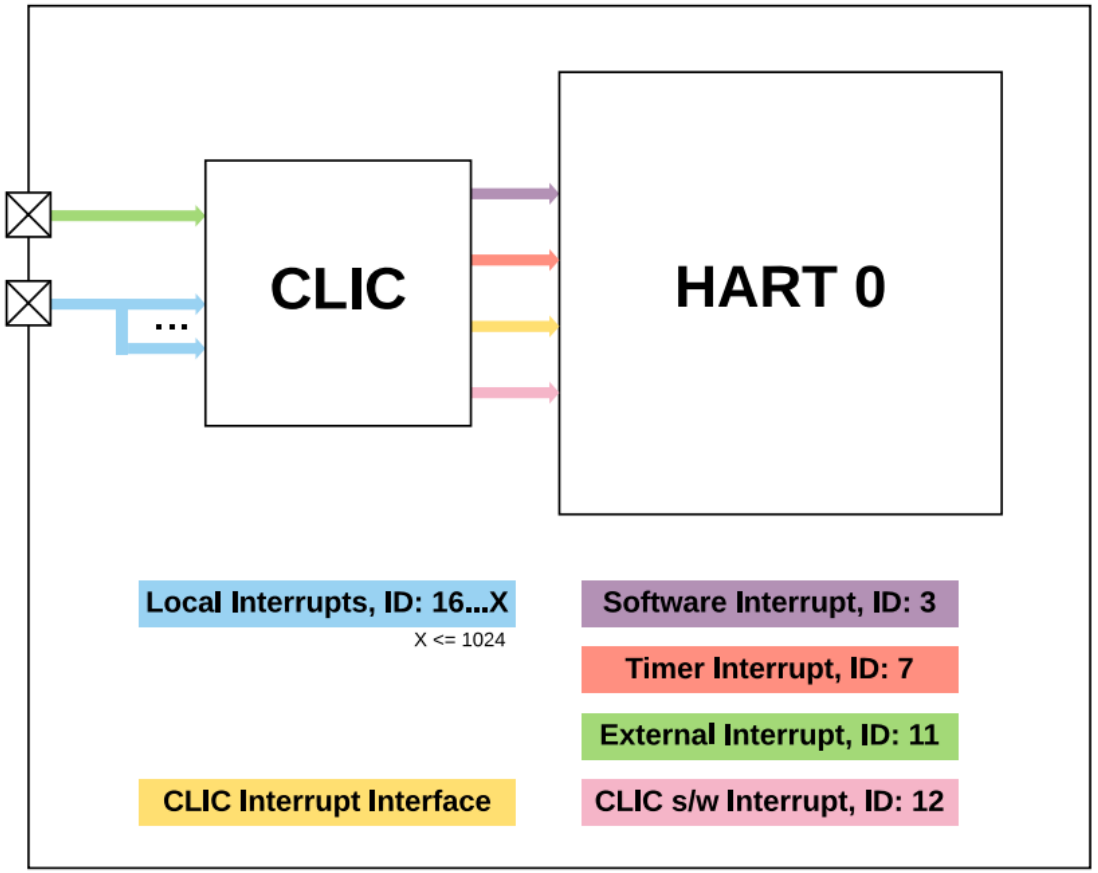
sifive E31
sifive U54
t-head C906/C910

sifive E20/E21
t-head E906/E907

CLIC 与 ECLIC







来自:sifive E20 Core Complex Manual

只服务于一个处理器内核Hart
支持 4096

CLIC 与 ECLIC

```
M-mode CLIC memory map
  Offset
  ###   0x0008-0x003F          reserved    ###
  ###   0x00C0-0x07FF          reserved    ###
  ###   0x0800-0x0FFF          custom      ###

  0x0000      1B      RW      cliccfg
  0x0004      4B      R       clicinfo


  0x0040      4B      RW      clicinttrig[0]
  0x0044      4B      RW      clicinttrig[1]
  0x0048      4B      RW      clicinttrig[2]
  ...
  0x00B4      4B      RW      clicinttrig[29]
  0x00B8      4B      RW      clicinttrig[30]
  0x00BC      4B      RW      clicinttrig[31]


  0x1000+4*i    1B/input    R or RW    clicintip[i]
  0x1001+4*i    1B/input    RW         clicintie[i]
  0x1002+4*i    1B/input    RW         clicintattr[i]
  0x1003+4*i    1B/input    RW         clicintctl[i]
  ...
  0x4FFC        1B/input    R or RW    clicintip[4095]
  0x4FFD        1B/input    RW         clicintie[4095]
  0x4FFE        1B/input    RW         clicintattr[4095]
  0x4FFF        1B/input    RW         clicintctl[4095]
```

ECLIC Registers:

Offset	Permission	Register	Width
0x0000	RW	cliccfg	8-bit
0x0004	R	clicinfo	32-bit
0x000b	RW	mth	8-bit
0x1000+4*i	RW	clicintip[i]	8-bit
0x1001+4*i	RW	clicintie[i]	8-bit
0x1002+4*i	RW	clicintattr[i]	8-bit
0x1003+4*i	RW	clicintctl[i]	8-bit

来自:Nuclei_N级别指令架构手册.pdf

Optional interrupt triggers (clicinttrig[i]) are used to generate a breakpoint exception, entry into Debug Mode, or a trace action.

来自：https://github.com/riscv/riscv-fast-interrupt/blob/master/clic.adoc#clic-interrupt-trigger-clicinttrig

CLIC 与 ECLIC

Number	Name	Description
0xm00	xstatus	Status register
0xm02	xedeleg	Exception delegation register
0xm03	~~xideleg~~	Interrupt delegation register (INACTIVE IN CLIC MODE)
0xm04	~~xie~~	Interrupt-enable register (INACTIVE IN CLIC MODE)
0xm05	xtvec	Trap-handler base address / interrupt mode
(NEW) 0xm07	xtvt	Trap-handler vector table base address
0xm40	xscratch	Scratch register for trap handlers
0xm41	xepc	Exception program counter
0xm42	xcause	Cause of trap
0xm43	xtval	Bad address or instruction
0xm44	~~xip~~	Interrupt-pending register (INACTIVE IN CLIC MODE)
(NEW) 0xm45	xnxti	Interrupt handler address and enable modifier
(NEW) 0xm46	xintstatus	Current interrupt levels
(NEW) 0xm47	xintthresh	Interrupt-level threshold
(NEW) 0xm48	xscratchcsw	Conditional scratch swap on priv mode change
(NEW) 0xm49	xscratchcswl	Conditional scratch swap on level change
(NEW) 0x3??	mclicbase	Base address for CLIC memory mapped registers

m is the nibble encoding the privilege mode (M=0x3, S=0x1, U=0x0)

具体参考：https://github.com/riscv/riscv-fast-interrupt/blob/master/clic.adoc#clic-csrs

中断相关差异CSR:

pushmsubm

pushmcause

pushmepc

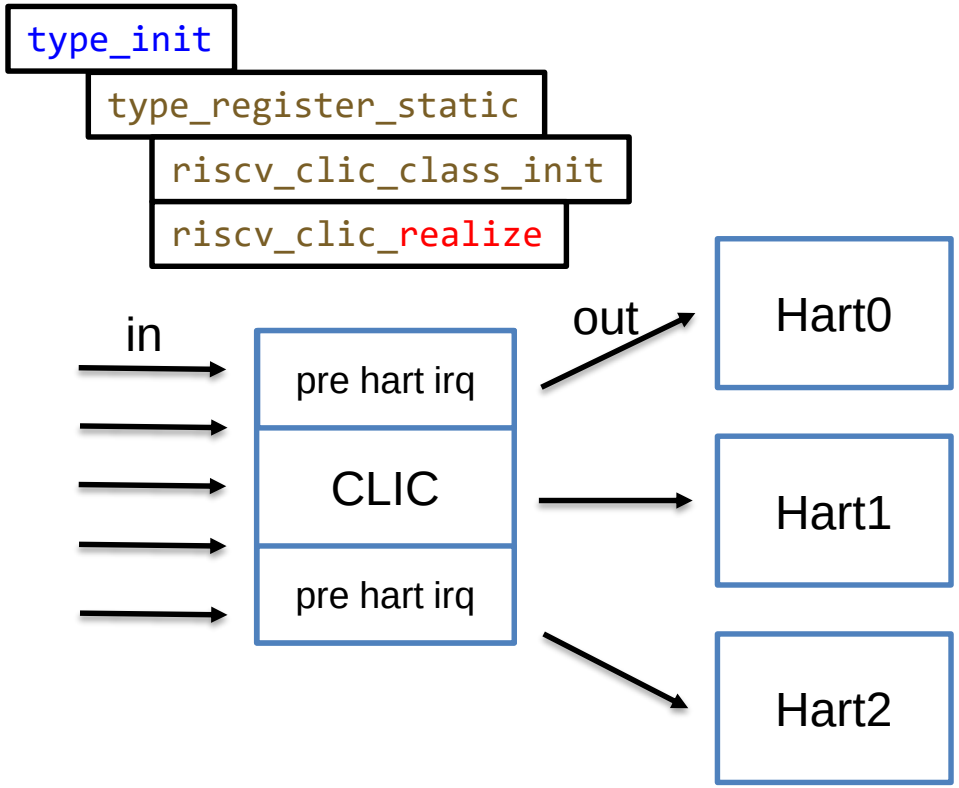
mtvt2

jalmnxti

CLIC实现

参考：https://github.com/romanheros/qemu/tree/riscv-clic-upstream-rfc

静态注册(QOM)




```
static const MemoryRegionOps riscv_clic_ops = {
    .read = riscv_clic_read,
    .write = riscv_clic_write,
};
```

hw\intc\riscv_clic.c

```
static void riscv_clic_cpu_irq_handler(void *opaque, int irq, int level)
{
    ……
    env->exccode = clic->exccode[cpu->cpu_index];
    cpu_interrupt(env_cpu(env), CPU_INTERRUPT_CLIC);
}
```

```
static void riscv_clic_realize(DeviceState *dev, Error **errp)
{
    ……
    memory_region_init_io(&clic->mmio, OBJECT(dev), &riscv_clic_ops, clic,
                          TYPE_RISCV_CLIC, clic->clic_size);

    clic->clicintip = g_new0(uint8_t, irqs);
    ……
    ……
    clic->cpu_irqs = g_new0(qemu_irq, clic->num_harts);
    sysbus_init_mmio(SYS_BUS_DEVICE(dev), &clic->mmio);

    qdev_init_gpio_in(dev, riscv_clic_set_irq, irqs);
    qdev_init_gpio_out(dev, clic->cpu_irqs, clic->num_harts);

    for (i = 0; i < clic->num_harts; i++) {
        RISCVCPU *cpu = RISCV_CPU(qemu_get_cpu(i));
        qemu_irq irq = qemu_allocate_irq(riscv_clic_cpu_irq_handler,
                                         &cpu->env, 1);
        qdev_connect_gpio_out(dev, i, irq);
        cpu->env.clic = clic;
        cpu->env.mclicbase = clic->mclicbase;
    }
}
```

CLIC实现

参考：https://github.com/romanheros/qemu/tree/riscv-clic-upstream-rfc

```
riscv_clic_set_irq
```

```
hartid = riscv_clic_get_hartid(clic, addr);
mode = riscv_clic_get_mode(clic, addr);
irq = riscv_clic_get_irq(clic, addr);
type = riscv_clic_get_trigger_type(clic, id);
```

```
riscv_clic_update_intip
```

```
riscv_clic_next_interrupt
```

```
qemu_set_irq(clic->cpu_irqs[hartid], 1);
```

```
qemu_irq irq = qemu_allocate_irq(riscv_clic_cpu_irq_handler,
                                 &cpu->env, 1);
```

```
cpu_interrupt(env_cpu(env),
    CPU_INTERRUPT_CLIC);
```

中断源 → clic → CPU处理 → CSR功能

helper_mret

```
bool riscv_cpu_exec_interrupt(CPUState *cs, int interrupt_request)
{
……
    if (interrupt_request & CPU_INTERRUPT_CLIC) {
        RISCVCPU *cpu = RISCV_CPU(cs);
        CPURISCVState *env = &cpu->env;
        int mode = (env->exccode >> 12) & 0b11;
        int enabled = riscv_cpu_local_irq_mode_enabled(env, mode);
        if (enabled) {
            cs->exception_index = RISCV_EXCP_INT_CLIC | env->exccode;
            cs->interrupt_request = cs->interrupt_request & ~CPU_INTERRUPT_CLIC;
            riscv_cpu_do_interrupt(cs);
            return true;
        }
    }
……
}
```

```
    if (clic) {
        mode = (cause >> 12) & 3;
        level = (cause >> 14) & 0xff;
        cause &= 0xfff;
        cause |= get_field(env->mstatus, MSTATUS_MPP) << 28;
        switch (mode) {
        case PRV_M:
            cause |= get_field(env->mintstatus, MINTSTATUS_MIL) << 16;
            cause |= get_field(env->mstatus, MSTATUS_MIE) << 27;
            env->mintstatus = set_field(env->mintstatus, MINTSTATUS_MIL, level);
            break;
        case PRV_S:
            ……
            break;
        }
    }
```

CLIC实现

mnxti

jalmnxti

clic.adoc#interrupt-handling-software

```
irq_enter:
   addi sp, sp, -FRAMESIZE # Allocate space on stack. (2)
   sw a1, OFFSET(sp)       # Save a1.
   csrr a1, mcause         # Get mcause of interrupted context.
   sw a0, OFFSET(sp)       # Save a0.
   csrr a0, mepc           # Get mepc of interrupt context.
   bgez a1, handle_exc     # Handle synchronous exception. (3)
   sw a0, OFFSET(sp)       # Save mepc.
   sw a1, OFFSET(sp)       # Save mcause of interrupted context.
   sw a2-a7, OFFSET(sp)    # Save other argument registers.
   sw t0-t6, OFFSET(sp)    # Save temporaries.
   sw ra, OFFSET(sp)       # 1 return address (5)
   csrrsi a0, mnxti, MIE   # Get highest current interrupt and enable
interrupts.
#----Interrupts enabled ---------------------- (7)
   beqz a0, exit           # Check if original interrupt vanished. (8)

 service_loop:       # 5 instructions in pending-interrupt service loop.
   lw a1, (a0)
   csrrsi x0, mstatus, MIE # Ensure interrupts enabled. (10)

   jalr a1         # Call C ABI Routine, a0 has interrupt ID encoded. (11)
                   # Routine must clear down interrupt in CLIC.
   csrrsi a0, mnxti, MIE
  bnez a0, service_loop  # Loop to service any interrupt. (13)

 #--- Restore ABI registers with interrupts enabled --- (14)
```

```c
static int rmw_mnxti(CPURISCVState *env, int csrno, target_ulong *ret_value,
                     target_ulong new_value, target_ulong write_mask)
{
    int clic_priv, clic_il, clic_irq;
    bool ready;
    CPUState *cs = env_cpu(env);
    if (write_mask)
        env->mstatus |= new_value & (write_mask & 0b11111);
    qemu_mutex_lock_iothread();
    ready = get_xnxti_status(env);
    if (ready) {
        riscv_clic_decode_exccode(env->exccode, &clic_priv, &clic_il,
                                  &clic_irq);
        if (write_mask) {
            bool edge = riscv_clic_edge_triggered(env->clic, clic_priv,
                                                  cs->cpu_index, clic_irq);
            if (edge) {
                riscv_clic_clean_pending(env->clic, clic_priv,
                                         cs->cpu_index, clic_irq);
            }
            env->mintstatus = set_field(env->mintstatus,
                                        MINTSTATUS_MIL, clic_il);
            env->mcause = set_field(env->mcause, MCAUSE_EXCCODE, clic_irq);
        }
        if (ret_value) {
            *ret_value = (env->mtvt & ~0x3f) + sizeof(target_ulong) * clic_irq;
        }
    } else {
        if (ret_value) {
            *ret_value = 0;
        }
    }
    qemu_mutex_unlock_iothread();
    return 0;
}
```

参考手册：7.4.24

参考：https://github.com/romanheros/qemu/tree/riscv-clic-upstream-rfc

SYSTIMER 与 CLINT

表 6-1 TIMER 寄存器的存储器映射地址

模块内偏移地址	读写属性	寄存器名称	复位默认值	功能描述
0x0	可读可写	mtime_lo	0x00000000	反映计时器 mtime 的低 32 位值，参见第 6.1.3 节了解其详细介绍。
0x4	可读可写	mtime_hi	0x00000000	反映计时器 mtime 的高 32 位值，参见第 6.1.3 节了解其详细介绍。
0x8	可读可写	mtimecmp_lo	0xFFFFFFFF	配置计时器的比较值 mtimecmp 低 32 位，参见第 6.1.5 节了解其详细介绍。
0xC	可读可写	mtimecmp_hi	0xFFFFFFFF	配置计时器的比较值 mtimecmp 高 32 位，参见第 6.1.5 节了解其详细介绍。
0xFF0	可读可写	msftrst	0x00000000	生成软件复位请求，参见第 6.1.7 节了解其详细介绍。
0xFF8	可读可写	mtimectl	0x00000000	控制计时器i介绍。
0xFFC	可读可写	msip	0x00000000	生成软件中断绍。

注意：
- TIMER 的寄存器只支持操作尺寸（Size）为 word 的对齐读写访问。
- TIMER 的寄存器区间为 0x0 ~ 0xFFF，除了上表中列出的寄存器之外的

来自:sifive E31 Core Complex Manual

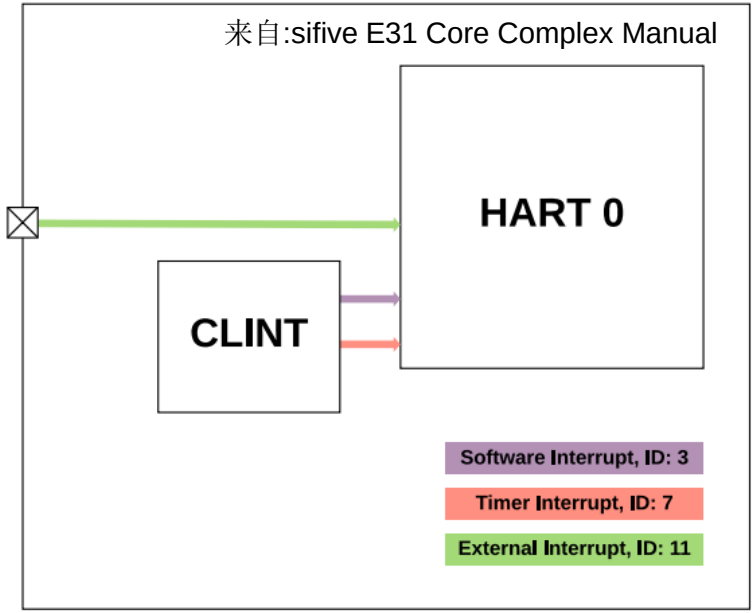


Figure 92: CLINT Block Diagram

Address	Width	Attr.	Description	Notes
0x0200_0000	4B	RW	msip for hart 0	MSIP Register (1-bit wide)
0x0200_0004			Reserved	
...				
0x0200_3FFF				
0x0200_4000	8B	RW	mtimecmp for hart 0	MTIMECMP Register
0x0200_4008			Reserved	
...				
0x0200_BFF7				
0x0200_BFF8	8B	RW	mtime	Timer Register
0x0200_C000			Reserved	

Table 104: CLINT Memory Map

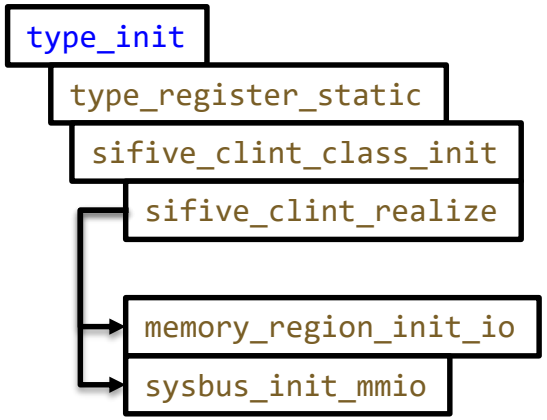
CLINT实现

hw\intc\sifive_clint.h

```
typedef struct SiFiveCLINTState {
    /*< private >*/
    SysBusDevice parent_obj;

    /*< public >*/
    MemoryRegion mmio;
    uint32_t hartid_base;
    uint32_t num_harts;
    uint32_t sip_base;
    uint32_t timecmp_base;
    uint32_t time_base;
    uint32_t aperture_size;
    uint32_t timebase_freq;
} SiFiveCLINTState;
```

静态注册(QOM)



hw\intc\sifive_clint.c

```
DeviceState *sifive_clint_create(hwaddr addr, hwaddr size,
    uint32_t hartid_base, uint32_t num_harts, uint32_t sip_base,
    uint32_t timecmp_base, uint32_t time_base, uint32_t timebase_freq,
    bool provide_rdtime)
{
    int i;
    for (i = 0; i < num_harts; i++) {
        CPUState *cpu = qemu_get_cpu(hartid_base + i);
        CPURISCVState *env = cpu ? cpu->env_ptr : NULL;
        if (!env) {
            continue;
        }
        if (provide_rdtime) {
            riscv_cpu_set_rdtime_fn(env, cpu_riscv_read_rtc, timebase_
freq);
        }
        env->timer = timer_new_ns(QEMU_CLOCK_VIRTUAL,
                                  &sifive_clint_timer_cb, cpu);
        env->timecmp = 0;
    }
    DeviceState *dev = qdev_new(TYPE_SIFIVE_CLINT);
    qdev_prop_set_uint32(dev, "num-harts", num_harts);
    ……
    sysbus_realize_and_unref(SYS_BUS_DEVICE(dev), &error_fatal);
    sysbus_mmio_map(SYS_BUS_DEVICE(dev), 0, addr);
    return dev;
}
```

MIP:

MSIP – machine soft ip

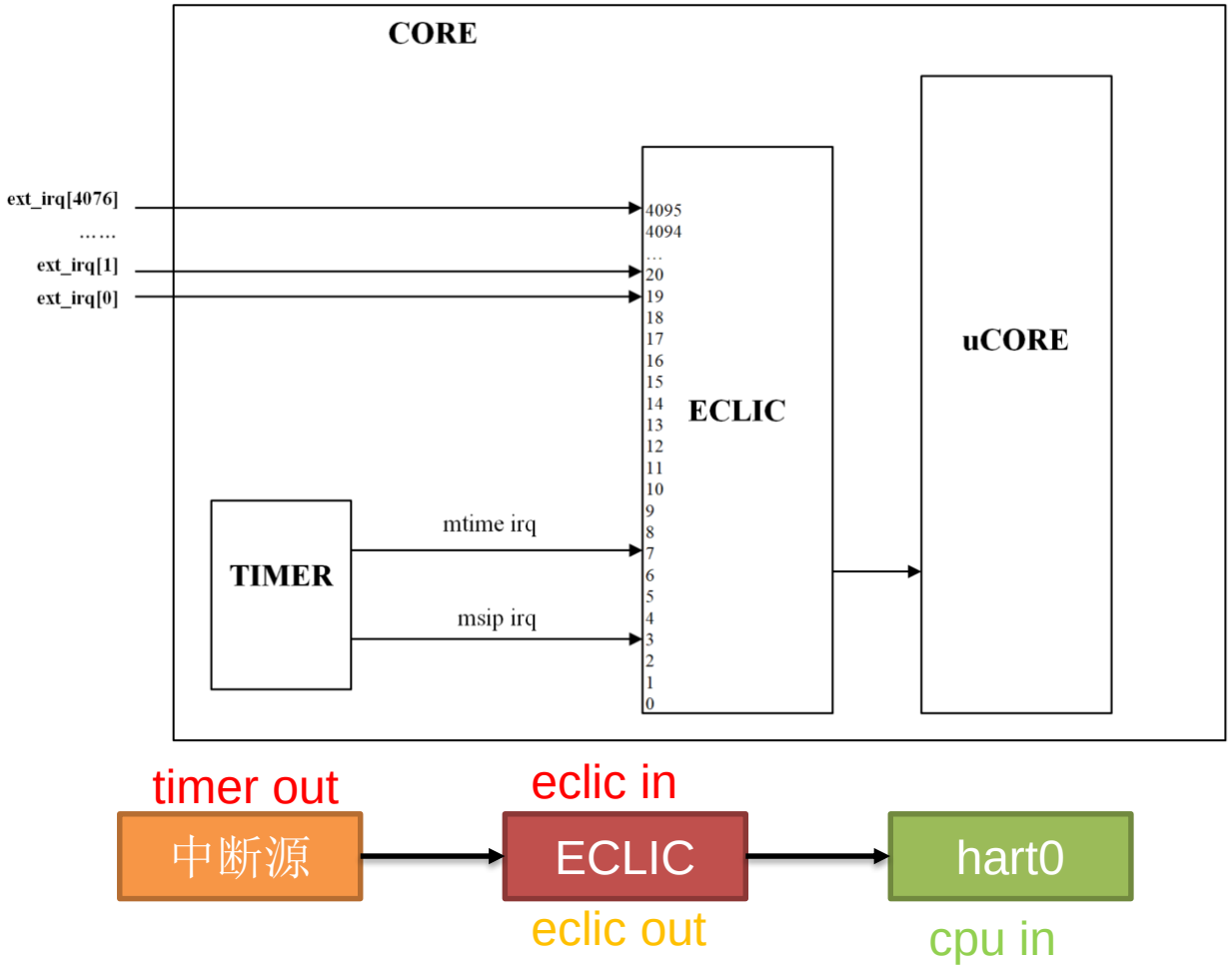
MTIP – machine timer ip

```
riscv_cpu_update_mip(RISCV_CPU(cpu),
    MIP_MSIP, BOOL_TO_MASK(value));
```

```
static void sifive_clint_timer_cb(void *opaque)
{
  riscv_cpu_update_mip(cpu, MIP_MTIP, BOOL_TO_MASK(1));
}
```

hw\core\qdev.c

SYSTIMER中断修改(ECLIC)




```
struct IRQState {
    Object parent_obj;

    qemu_irq_handler handler;
    void *opaque;
    int n;
};
```

```
struct NamedGPIOList {
    char *name;
    qemu_irq *in;
    int num_in;
    int num_out;
    QLIST_ENTRY(NamedGPIOList) node;
};
```

in创建与初始化

```
void qdev_init_gpio_in(DeviceState *dev,
qemu_irq_handler handler, int n)
```

out创建与初始化

```
void qdev_init_gpio_out(DeviceState *dev,
 qemu_irq *pins, int n);
```

in获取

```
qemu_irq qdev_get_gpio_in(DeviceState *dev,
 int n);
```

out获取

```
void qdev_connect_gpio_out(DeviceState *dev,
 int n, qemu_irq pin);
```

```
qemu_irq qemu_allocate_irq(qemu_irq_handler handler, void *opaque, int n)
```

hw\core\qdev.c

SYSTIMER中断修改(ECLIC)

```
(qemu) info qom-tree
/machine (mcu_200t-machine)
  /peripheral (container)
  /peripheral-anon (container)
  /soc (riscv.nuclei.n.soc)
    /cpus (riscv.hart_array)
      /harts[0] (nuclei-n600-riscv-cpu)
    /eclic (riscv.nuclei.eclic)
      /riscv.nuclei.eclic[0] (memory-region)
      /unnamed-gpio-in[0] (irq)
      /unnamed-gpio-in[10] (irq)
      /unnamed-gpio-in[11] (irq)
      /unnamed-gpio-in[12] (irq)
      /unnamed-gpio-in[13] (irq)
      /unnamed-gpio-in[14] (irq)
      /unnamed-gpio-in[15] (irq)
      /unnamed-gpio-in[16] (irq)
      /unnamed-gpio-in[17] (irq)
      /unnamed-gpio-in[18] (irq)
      /unnamed-gpio-in[19] (irq)
      /unnamed-gpio-in[1] (irq)
      /unnamed-gpio-in[20] (irq)
      /unnamed-gpio-in[21] (irq)
      /unnamed-gpio-in[22] (irq)
      /unnamed-gpio-in[23] (irq)
      /unnamed-gpio-in[24] (irq)
      /unnamed-gpio-in[25] (irq)
      /unnamed-gpio-in[26] (irq)
      /unnamed-gpio-in[27] (irq)
      /unnamed-gpio-in[28] (irq)
      /unnamed-gpio-in[29] (irq)
      /unnamed-gpio-in[2] (irq)
      /unnamed-gpio-in[30] (irq)
```

```c
struct IRQState {
    Object parent_obj;

    qemu_irq_handler handler;
    void *opaque;
    int n;
};
```

```c
struct NamedGPIOList {
    char *name;
    qemu_irq *in;
    int num_in;
    int num_out;
    QLIST_ENTRY(NamedGPIOList) node;
};
```

```c
/* Init ECLIC IRQ */
    eclic->irqs[Internal_SysTimerSW_IRQn] =
        qemu_allocate_irq(nuclei_eclic_irq_request,
                          eclic, Internal_SysTimerSW_IRQn);
    eclic->irqs[Internal_SysTimer_IRQn] =
        qemu_allocate_irq(nuclei_eclic_irq_request,
                          eclic, Internal_SysTimer_IRQn);

    for (id = Internal_Reserved_Max_IRQn; id < eclic->num_sources; id++) {
        eclic->irqs[id] = qemu_allocate_irq(nuclei_eclic_irq_request,
                                            eclic, id);
    }
```

```c
 /* Allocate irq by qdev_init_gpio */
qdev_init_gpio_in(dev, nuclei_eclic_irq_request, eclic->num_sources);
```

```c
qemu_irq irq;
irq = qemu_allocate_irq(set_cpu_irq, dev, 0);
qdev_connect_gpio_out(dev, 0, cpu_irq);
```

```c
 /* Allocate irq by qdev_init_gpio */
qdev_init_gpio_in(dev, nuclei_eclic_set_ip, eclic->num_sources);
qdev_init_gpio_out(dev, irq, 0);
```

SYSTIMER中断修改(Systimer)

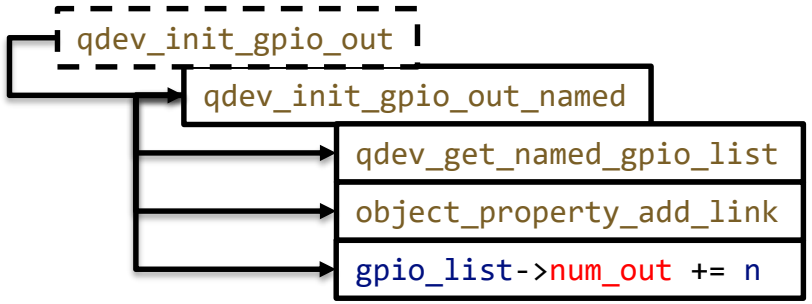
hw\intc\nuclei_systimer.c

```
static void nuclei_timer_realize(DeviceState *dev, Error **errp)
{
    ……
    NucLeiSYSTIMERState *s = NUCLEI_SYSTIMER(dev);

    sysbus_init_irq(SYS_BUS_DEVICE(dev), &s->soft_irq);
    sysbus_init_irq(SYS_BUS_DEVICE(dev), &s->timer_irq);
    ……
}
```

创建

```
void sysbus_init_irq(SysBusDevice *dev, qemu_irq *p)
{
    qdev_init_gpio_out_named(DEVICE(dev), p, SYSBUS_DEVICE_GPIO
_IRQ, 1);
}
```



hw\riscv\nuclei_n.c

```
s->timer.soft_irq =&(NUCLEI_ECLIC(s->eclic)-
>irqs[Internal_SysTimerSW_IRQn]);
s->timer.timer_irq = &(NUCLEI_ECLIC(s->eclic)-
>irqs[Internal_SysTimer_IRQn]);
```

```
sysbus_connect_irq(SYS_BUS_DEVICE(&s->timer), 0,
                   qdev_get_gpio_in(DEVICE(&s->eclic),
                                    Internal_SysTimerSW_IRQn));

sysbus_connect_irq(SYS_BUS_DEVICE(&s->timer), 1,
                   qdev_get_gpio_in(DEVICE(&s->eclic),
                                    Internal_SysTimer_IRQn));
```

```
void sysbus_connect_irq(SysBusDevice *dev, int n, qemu_irq irq)
{
    SysBusDeviceClass *sbd = SYS_BUS_DEVICE_GET_CLASS(dev);
    qdev_connect_gpio_out_named(DEVICE(dev), SYSBUS_DEVICE_GPIO_IRQ, n
, irq);
    if (sbd->connect_irq_notifier) {
        sbd->connect_irq_notifier(dev, irq);
    }
}
```

PLIC

来自:sifive U54-MC Core Complex Manual

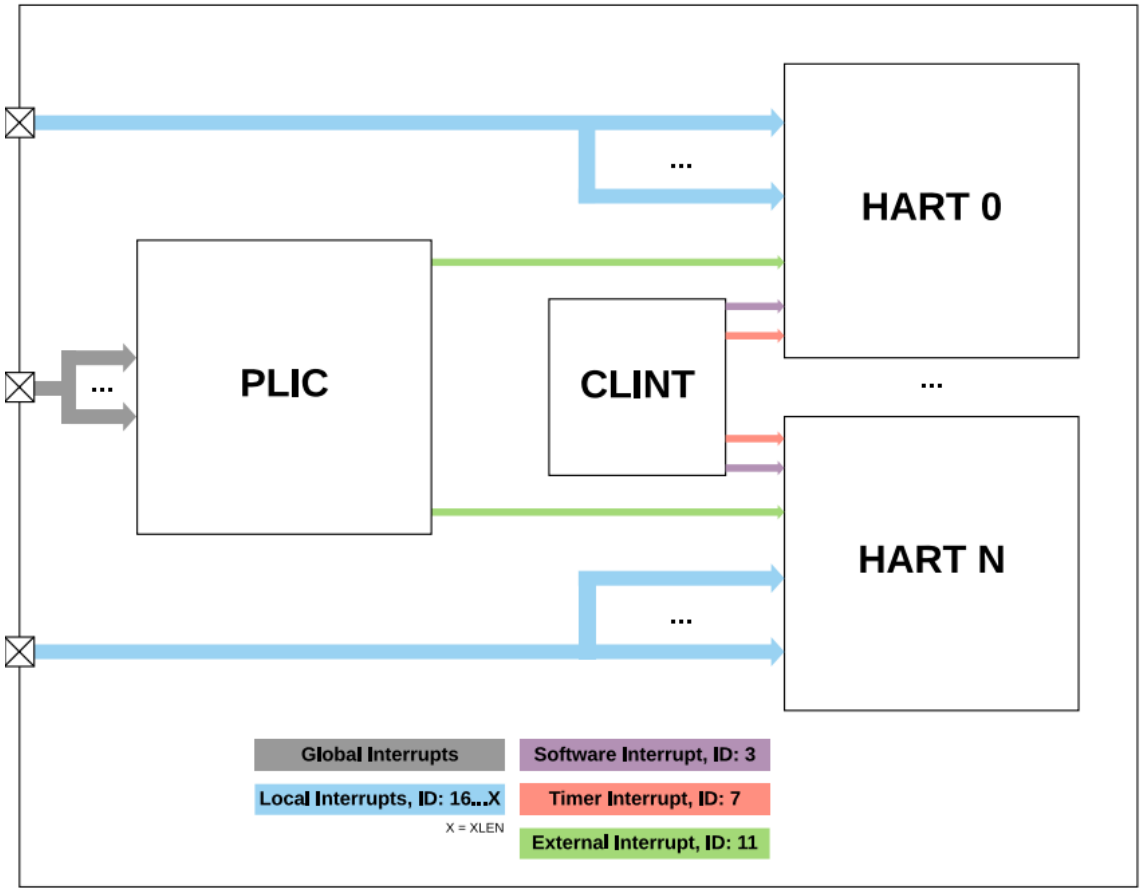


Figure 103: PLIC Multi-Core Block Diagram

1023 interrupts (0 is reserved)

15872 contexts

PLIC Memory Map

base + 0x000000: Reserved (interrupt source 0 does not exist)
base + 0x000004: Interrupt source 1 priority
base + 0x000008: Interrupt source 2 priority
...
base + 0x000FFC: Interrupt source 1023 priority
base + 0x001000: Interrupt Pending bit 0-31
base + 0x00107C: Interrupt Pending bit 992-1023
...
base + 0x002000: Enable bits for sources 0-31 on context 0
base + 0x002004: Enable bits for sources 32-63 on context 0
...
base + 0x00207F: Enable bits for sources 992-1023 on context 0
base + 0x002080: Enable bits for sources 0-31 on context 1
base + 0x002084: Enable bits for sources 32-63 on context 1
...
base + 0x0020FF: Enable bits for sources 992-1023 on context 1
base + 0x002100: Enable bits for sources 0-31 on context 2
base + 0x002104: Enable bits for sources 32-63 on context 2
...
base + 0x00217F: Enable bits for sources 992-1023 on context 2
...
base + 0x1F1F80: Enable bits for sources 0-31 on context 15871
base + 0x1F1F84: Enable bits for sources 32-63 on context 15871
base + 0x1F1FFF: Enable bits for sources 992-1023 on context 15871
...
base + 0x200000: Priority threshold for context 0
base + 0x200004: Claim/complete for context 0
...
base + 0x201000: Priority threshold for context 1
base + 0x201004: Claim/complete for context 1
...
base + 0x3FFE000: Priority threshold for context 15871
base + 0x3FFE004: Claim/complete for context 15871

PLIC

来自:sifive U54-MC Core Complex Manual

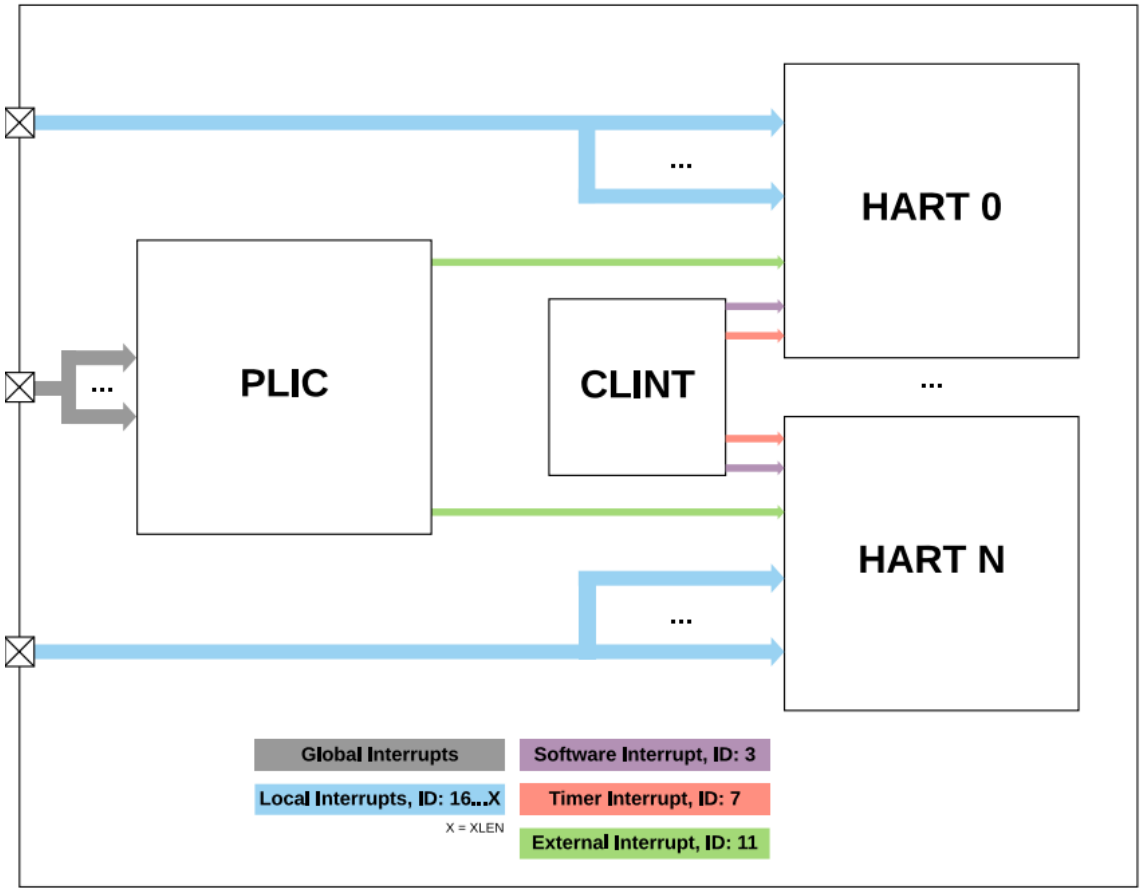


Figure 103: PLIC Multi-Core Block Diagram

1023 interrupts (0 is reserved) 15872 contexts

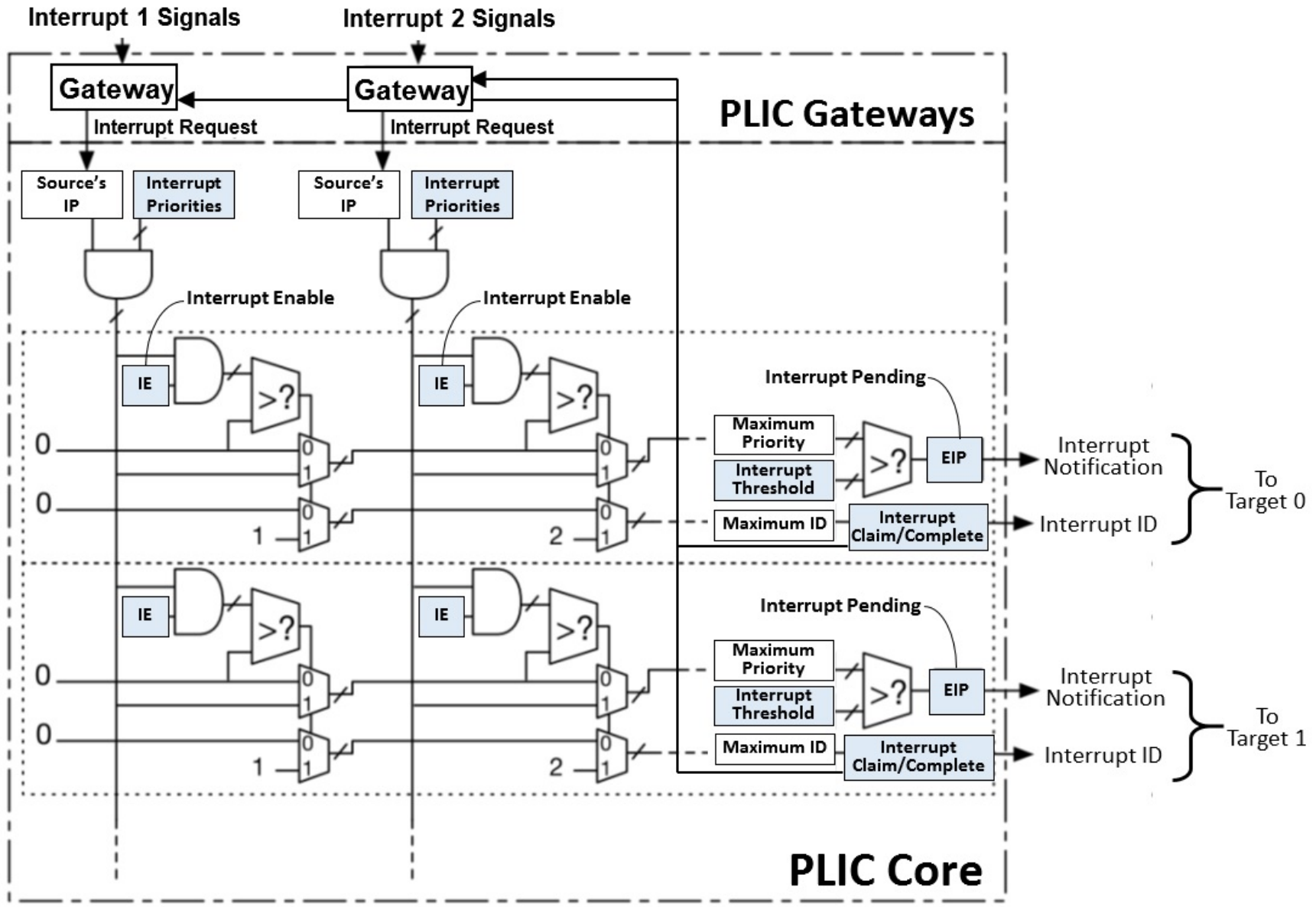
Address	Width	Attr.	Description	Notes
0x0C00_0000			Reserved	
0x0C00_0004	4B	RW	Source 1 priority	See Section 9.3 for more information
...				
0x0C00_0210	4B	RW	Source 132 priority	
0x0C00_0214			Reserved	
...				
0x0C00_1000	4B	RO	Start of pending array	See Section 9.4 for more information
...				
0x0C00_1010	4B	RO	Last word of pending array	
0x0C00_1014			Reserved	
...				
0x0C00_2000	4B	RW	Start Hart 0 M-Mode interrupt enables	See Section 9.5 for more information
...				
0x0C00_2010	4B	RW	End Hart 0 M-Mode interrupt enables	
0x0C00_2014			Reserved	
...				
0x0C00_2080	4B	RW	Start Hart 0 S-Mode interrupt enables	See Section 9.5 for more information
...				
0x0C00_2090	4B	RW	End Hart 0 S-Mode interrupt enables	
0x0C00_2094			Reserved	
...				
0x0C1F_F000	1B	RW	PLIC global clock gating disable feature	See Section 9.6 for more information
0x0C1F_F001			Reserved	
...				
0x0C20_0000	4B	RW	Hart 0 M-Mode priority threshold	See Section 9.7 for more information
0x0C20_0004	4B	RW	Hart 0 M-Mode claim/ complete	See Section 9.8 for more information
0x0C20_0008			Reserved	
...				
0x0C20_1000	4B	RW	Hart 0 S-Mode priority threshold	See Section 9.7 for more information
0x0C20_1004	4B	RW	Hart 0 S-Mode claim/complete	See Section 9.8 for more information
0x0C20_1008			Reserved	
...				
0x1000_0000			End of PLIC Memory Map	

来自:sifive U54-MC Core Complex Manual

Table 105: PLIC Memory Map

PLIC

- Interrupt Priorities registers
- Interrupt Pending Bits registers
- Interrupt Enables registers
- Priority Thresholds registers
- Interrupt Claim registers
- Interrupt Completion registers



来自: https://github.com/riscv/riscv-plic-spec

PLIC实现

hw\intc\sifive_plic.c

静态注册

- type_init
  - type_register_static
  - sifive_plic_class_init
  - sifive_plic_realize

读写接口

```
static const MemoryRegionOps sifive_plic_ops =
{
    .read = sifive_plic_read,
    .write = sifive_plic_write,
    ……
};
```

功能接口

- sifive_plic_set_pending
- sifive_plic_set_claimed
- sifive_plic_irqs_pending
- sifive_plic_claim
- sifive_plic_update

XLEN-1	12	11	10	9	8	7	6	5	4	3	2	1	0
WIRI		MEIP	HEIP	SEIP	UEIP	MTIP	HTIP	STIP	UTIP	MSIP	HSIP	SSIP	USIP
XLEN-12		1	1	1	1	1	1	1	1	1	1	1	1

Figure 3.10: Machine interrupt-pending register (mip).

MEIP, SEIP 发生未处理的外部中断位

实例化

```
static void sifive_plic_realize(DeviceState *dev, Error **errp)
{
    ……
    memory_region_init_io(&plic-
>mmio, OBJECT(dev), &sifive_plic_ops, plic,
                          TYPE_SIFIVE_PLIC, plic->aperture_size);
    ……
    qdev_init_gpio_in(dev, sifive_plic_irq_request, plic->num_sources);
    ……
}
```

- sifive_plic_irq_request
  - sifive_plic_set_pending
  - sifive_plic_update

```
cpu->interrupt_request |= CPU_INTERRUPT_HARD;
```

```
static void sifive_plic_update(SiFivePLICState *plic)
{
    ……
    int level = sifive_plic_irqs_pending(plic, addrid);
    switch (mode) {
    case PLICMode_M:
        riscv_cpu_update_mip(RISCV_CPU(cpu),
                MIP_MEIP, BOOL_TO_MASK(level));
        break;
    case PLICMode_S:
        riscv_cpu_update_mip(RISCV_CPU(cpu),
                MIP_SEIP, BOOL_TO_MASK(level));
        break;
    default:
        break;
    }
    ……
}
```

发出中断

PLIC实现

hw\intc\sifive_plic.c

```
DeviceState *sifive_plic_create(hwaddr addr, char *hart_config,
    uint32_t hartid_base, uint32_t num_sources,
    uint32_t num_priorities, uint32_t priority_base,
    uint32_t pending_base, uint32_t enable_base,
    uint32_t enable_stride, uint32_t context_base,
    uint32_t context_stride, uint32_t aperture_size)
{
    DeviceState *dev = qdev_new(TYPE_SIFIVE_PLIC);
    assert(enable_stride == (enable_stride & -enable_stride));
    assert(context_stride == (context_stride & -context_stride));
    qdev_prop_set_string(dev, "hart-config", hart_config);
    qdev_prop_set_uint32(dev, "hartid-base", hartid_base);
    qdev_prop_set_uint32(dev, "num-sources", num_sources);
    qdev_prop_set_uint32(dev, "num-priorities", num_priorities);
    qdev_prop_set_uint32(dev, "priority-base", priority_base);
    qdev_prop_set_uint32(dev, "pending-base", pending_base);
    qdev_prop_set_uint32(dev, "enable-base", enable_base);
    qdev_prop_set_uint32(dev, "enable-stride", enable_stride);
    qdev_prop_set_uint32(dev, "context-base", context_base);
    qdev_prop_set_uint32(dev, "context-stride", context_stride);
    qdev_prop_set_uint32(dev, "aperture-size", aperture_size);
    sysbus_realize_and_unref(SYS_BUS_DEVICE(dev), &error_fatal);
    sysbus_mmio_map(SYS_BUS_DEVICE(dev), 0, addr);
    return dev;
}
```

获取IRQ　　hw\riscv\sifive_e.c

```
qdev_get_gpio_in(DEVICE(s->plic),SIFIVE_E_X_IRQNUM)
```

设置

```
sysbus_connect_irq
qemu_irq irq= qdev_get_gpio_in()
……
```

Interrupt source 1 priority

Interrupt Pending bit 0-31

Enable bits for sources 0-31 on context 0

Priority threshold for context 0

UART中断使用

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.

include\hw\char\nuclei_uart.h

```
typedef struct NucLeiUARTState
{
    /*< private >*/
    SysBusDevice parent_obj;

    /*< public >*/
    qemu_irq irq;

    MemoryRegion mmio;
    CharBackend chr;

    uint8_t rx_fifo[8];
    unsigned int rx_fifo_len;

    uint32_t txdata;
    uint32_t rxdata;
    uint32_t txctrl;
    uint32_t rxctrl;
    uint32_t ie;
    uint32_t ip;
    uint32_t div;

} NucLeiUARTState;
```

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

UART中断使用

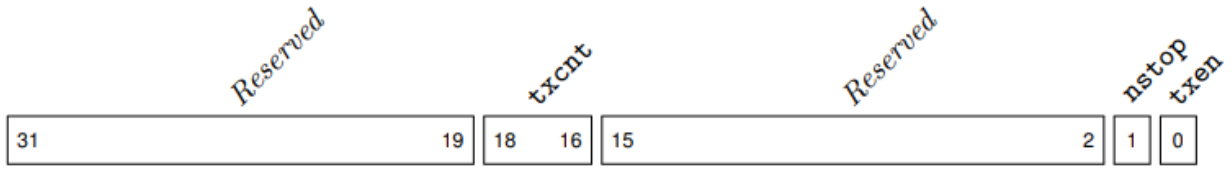


Figure 12.3: Format of `txctrl` register.

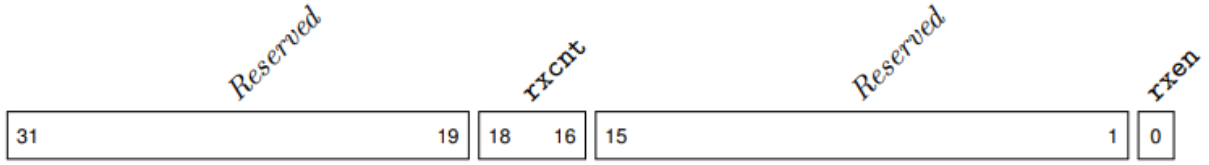


Figure 12.4: Format of `rxctrl` register.

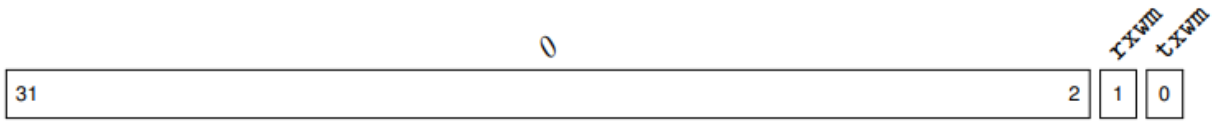


Figure 12.5: Format of `ie` and `ip` registers.

- txcnt:
  表示TX-FIFO产生中断的阈值
- nstop:
  0-表示发送1字节后插入1个停止位8-N-1
  1-表示发送1字节后插入2个停止位8-N-2
- txen:
  1-表示使能UART发送行为
  0-表示不使能UART发送行为
- rxcnt:
  表示RX-FIFO产生中断的阈值
- rxen:
  1-表示使能UART接收行为
  0-表示不使能UART接收行为
- rxwm:
  1-表示使能/产生UART接收中断
  0-表示不使能/不产生UART接收中断
- txwm:
  1-表示使能/产生UART发送中断
  0-表示不使能/不产生UART发送中断

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

UART中断使用

hw\char\nuclei_uart.c

```
static void nuclei_uart_realize(DeviceState *dev, Error **errp)
{
    NucLeiUARTState *s = NUCLEI_UART(dev);

    sysbus_init_irq(SYS_BUS_DEVICE(dev), &s->irq);

    ……
}
```

初始化

hw\riscv\nuclei_n.c

```
sysbus_connect_irq(SYS_BUS_DEVICE(&s->uart0), 0,
                   qdev_get_gpio_in(DEVICE(&s->eclic),22));
```

UART中断源 → ECLIC 22中断

qemu handler `nuclei_eclic_irq_request`

SDK handler 根据IP判断是否有数据

相关寄存器实现

hw\char\nuclei_uart.c

```
static uint64_t
uart_read(void *opaque, hwaddr offset, unsigned int size)
{
    ……
    case NUCLEI_UART_REG_IP:
        value = uart_ip(s);
        break;
    ……
}
```

```
static uint64_t uart_ip(NucLeiUARTState *s)
{
    uint64_t ret = 0;

    uint64_t txcnt = NUCLEI_UART_GET_TXCNT(s->txctrl);
    uint64_t rxcnt = NUCLEI_UART_GET_RXCNT(s->rxctrl);

    if (txcnt != 0) {
        ret |= NUCLEI_UART_IP_TXWM;
    }
    if (s->rx_fifo_len > rxcnt){
        ret |= NUCLEI_UART_IP_RXWM;
    }

    return ret;
}
```

UART中断使用

```
static void uart_write(void *opaque, hwaddr offset,
            uint64_t value, unsigned int size)
{
    ……
    unsigned char ch = value;
    ……
    case NUCLEI_UART_REG_TXDATA:
        qemu_chr_fe_write(&s->chr, &ch, 1);
        update_irq(s);
        break;
    ……
}
```

写数据

```
static uint64_t uart_read(void *opaque, hwaddr offset, unsigned int size)
{
    ……
    case NUCLEI_UART_REG_RXDATA:
        if (s->rx_fifo_len)
        {
            fifo_val = s->rx_fifo[0];
            memmove(s->rx_fifo, s->rx_fifo + 1, s->rx_fifo_len - 1);
            s->rx_fifo_len--;
            qemu_chr_fe_accept_input(&s->chr);
            update_irq(s);
            return fifo_val;
        } 0x80000000
    ……
}
```

读数据

hw\char\nuclei_uart.c

```
static void update_irq(NucLeiUARTState *s)
{
    int cond = 0;
    s->txctrl |= 0x1;
    if (s->rx_fifo_len)
        s->rxctrl &= ~0x1;
    else
        s->rxctrl |= 0x1;

    if ((s->ie & NUCLEI_UART_IE_TXWM) ||
        ((s->ie & NUCLEI_UART_IE_RXWM) && s->rx_fifo_len)){
        cond = 1;
    }

    if (cond)
        qemu_irq_raise(s->irq);
    else
        qemu_irq_lower(s->irq);
}
```

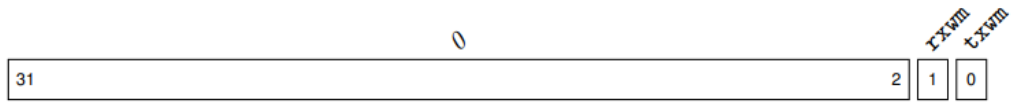


Figure 12.5: Format of ie and ip registers.

IE对下控制 IP对上判断

UART中断使用

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

```
initialize rti_board_start:0 done

 \ | /
- RT -     Thread Operating System
 / | \     4.0.3 build Sep  9 2020
 2006 - 2020 Copyright by rt-thread team
do components initialization.
initialize rti_board_end:0 done
initialize dfs_init:0 done
initialize libc_system_init:0 done
initialize finsh_system_init:0 done
msh />ps
thread   pri  status      sp     stack size max used left tick  error
-------- ---  ------- ---------- ----------  ------  ---------- ---
serrxsim   5  suspend 0x000000a8 0x0000018c    50%   0x00000002 000
tshell    20  running 0x000001b8 0x00001000    14%   0x0000000a 000
tidle0    31  ready   0x000000b8 0x0000018c    54%   0x00000011 000
timer      4  suspend 0x00000098 0x00000200    29%   0x00000003 000
msh />
```

参考：https://github.com/RT-Thread/rt-thread/tree/master/bsp/nuclei

下节课内容：

总线虚拟化

- QEMU 总线是什么
- IIC总线介绍
- SPI总线介绍

谢谢

wangjunqiang@iscas.ac.cn