

gic cpu接口设置

一. 设置步骤

- 设置权限掩码，权限比该掩码高的中断会被接收
- 使能本核对应的cpu接口

二. 设置代码

```
void cpu0_gic_Int_init (void)
{
    u32    reg;
    u32    core_msk;
    u32    i;
    u16    irq;
    gic_irq_vect_t *pvect;
    u32 flags;
    struct bm_gic_chip_data *gic_data = (struct bm_gic_chip_data *) (AMPPHY_START +
AMP_GIC_DATA_OFFSET);
    void *base = (void *)GIC_INT_DIST_BASE; //中断分发器寄存器组基地址
    u32 gic_irqs, gic_max_irq;

    BITCLR32(GIC_INT_REG_ICDDCR, BIT0); //关闭GIC Distributor
    //uartprintf("*****###gic_irqs\r\n");
    gic_irqs = readl(base + 4) & 0x1f; //ICDICTR
    gic_irqs = (gic_irqs + 1) * 32; // (N+1)*32 = maxnum, 支持最大的中断数目256
    //uartprintf("++gic_irqs = %d\r\n", gic_irqs);
    if (gic_irqs > 1020)
        gic_irqs = 1020;
    gic_data->gic_irqs = gic_irqs; //gic支持的最大中断数，最大中断号是gic_irqs - 1
    gic_max_irq = gic_irqs - 1; //gic支持的最大中断号255

    // 可以单独控制禁止分发某个中断，这里禁止全部，i 代表组号，从第0组开始到最后一组
    // 一个寄存器设置32个中断
    for (i = 0u; i <= (gic_max_irq / 32u); i++) { // Disable all the GIC irq
sources.
        reg = (u32)&GIC_INT_REG_ICDICR;
        reg += i * 4u; //

        (*(u32 *)reg) = 0xFFFFFFFF; //禁止分发中断到cpu接口

        reg = (u32)&GIC_INT_REG_ICDICPR;
        reg += i * 4u;

        // (*(u32 *)reg) = 0xa0a0a0a0;
        (*(u32 *)reg) = 0xFFFFFFFF; //清中断的pending
    }
}
```

```

core_msk = (BIT0 | BIT8 | BIT16 | BIT24); // 该寄存器中的4个中断的目标cpu都是cpu0
//core_msk = (BIT1 | BIT9 | BIT17 | BIT25);

// 设置每个中断的目标cpu和优先级,从中断32开始,i有多少寄存器要设置
// 一个寄存器设置4个中断
for (i = 0u; i <= ((gic_max_irq - 32u) / 4u); i++) {
    reg = (u32)&GIC_INT_REG_ICDIPTR;    //target
    reg += (((32u / 4u) + i) * 4u);

    (* (u32 *)reg) = core_msk;          // 该寄存器中的4个中断的目标cpu都是cpu0

    reg = (u32)&GIC_INT_REG_ICDIPR;      //priority
    reg += (((32u / 4u) + i) * 4u);

    (* (u32 *)reg) = 0xa0a0a0a0;        // 该寄存器中的4个中断的权限值都是0xa0
}

gic_Int_cfg_target(GIC_DMA0);           //设置中断GIC_DMA0目标cpu是cpu1
gic_Int_cfg_target(GIC_PFGA0);          //设置中断GIC_PFGA0目标cpu是cpu1

gic_Int_cfg(GIC_PFGA0, 0x80, GIC_INT_POL_EDGE_RISING); //设置中断GIC_PFGA0的权限值为
0x80, 触发方式为GIC_INT_POL_EDGE_RISING

// Initialize the vector table.
for (irq = 0u; irq < MAX_IRQ_NR; irq++) {
    pvect = &gic_vect_tbl[irq];        //一个条目地址,该条目包含该irq对应的函数和参数

    flags = cpu_local_irq_save();       //屏蔽cpu irq,并在之前保存cpsr
    gic_Int_vect_clr(pvect);            // Initialize main vector table entry.
    cpu_local_irq_restore(flags);       //使能cpu irq,并恢复屏蔽之前保存的cpsr
}

BITCLR32(GIC_INT_REG_ICCPMR, 0xff);    //将寄存器前8位清0
BITSET32(GIC_INT_REG_ICCPMR, BIT4|BIT5|BIT6|BIT7); //设置权限掩码为0xf0,权限比它高,即值
比它小cpu接口会接收
BITSET32(GIC_INT_REG_ICCICR, BIT0|BIT1); // 使能 CPU interface.
BITCLR32(GIC_INT_REG_ICCICR, BIT3);
BITSET32(GIC_INT_REG_ICDDCR, BIT0);    //使能 GIC Distributor
}

```

```

void cpu1_gic_Int_init (void)
{
    u32    i;
    u16    irq;
    gic_irq_vect_t *pvect;
    u32    flags;

```

```

//初始化向量表(是一个数组, 不是那个向量表)为初始值
for (irq = 0u; irq < MAX_IRQ_NR; irq++) {
    pvect = &gic_vect_tbl[irq];           //一个条目地址, 该条目包含该irq对应的函数和参数

    flags = cpu_local_irq_save();          //屏蔽cpu irq, 并在之前保存cpsr
    gic_Int_vect_clr(pvect);               // Initialize main vector table entry.
    cpu_local_irq_restore(flags);          //使能cpu irq, 并恢复屏蔽之前保存的cpsr
}

BITCLR32(GIC_INT_REG_ICCPMR, 0xffff);    //设置本cpu的过滤器
BITSET32(GIC_INT_REG_ICCPMR, BIT4|BIT5|BIT6|BIT7); //如果中断的优先级高于该值则中断可发送
给cpu
BITSET32(GIC_INT_REG_ICCICR, BIT0|BIT1); // Enable CPU interface.
}

```