Sail RISC-V PLIC 实现

min版 · 实现范围：64 sources/2 contexts

分享人：袁栋

PLIC介绍

外设中断源

UART　VIRTIO　GPIO

PLIC 汇聚分发

将多个外设中断源汇聚
分发至不同 hart 的特权级上下文

hart0

M-mode
Machine mode context

S-mode
Supervisor mode context

- PLIC 用于汇聚 SoC 上的外设中断源，并分别投递给不同 hart 的不同特权级上下文

三组核心状态寄存器

pending[source]

`pending[source]`
标记某个 source 是否"有中断等待处理"

功能说明：

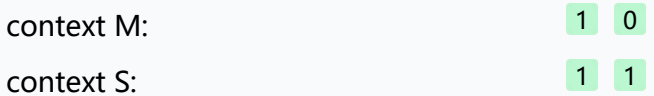
- 每一位对应一个中断源
- 中断到达时置位
- PLIC仲裁时检查此位

enable[context][source]

`enable[context][source]`
某个 context 是否允许接收该 source

功能说明：

- 二维位图控制中断使能
- 每个context维护独立位图
- 中断分发前检查此位

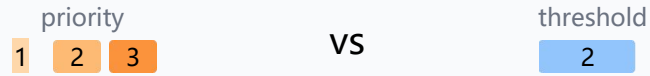


priority[source] + threshold[context]

`priority[source]`
`threshold[context]`
决定中断优先级筛选

功能说明：

- priority: 中断源优先级
- threshold: context阈值
- 仅接收 priority > threshold 的中断



claim/complete 与 CPU 交互

CPU

中断处理流程

- 进入 trap handler
- 读 claim/complete 寄存器
- 处理中断 ID
- 写 complete 寄存器

MMIO 地址

同一物理地址

读 = claim

写 = complete

PLIC

claim (读操作)

- 原子选择最高优先级中断
- 清pending bit
- 设置claimed bit
- 返回中断ID

complete (写操作)

- 写回处理完的中断ID
- 清claimed bit
- 更新中断优先级

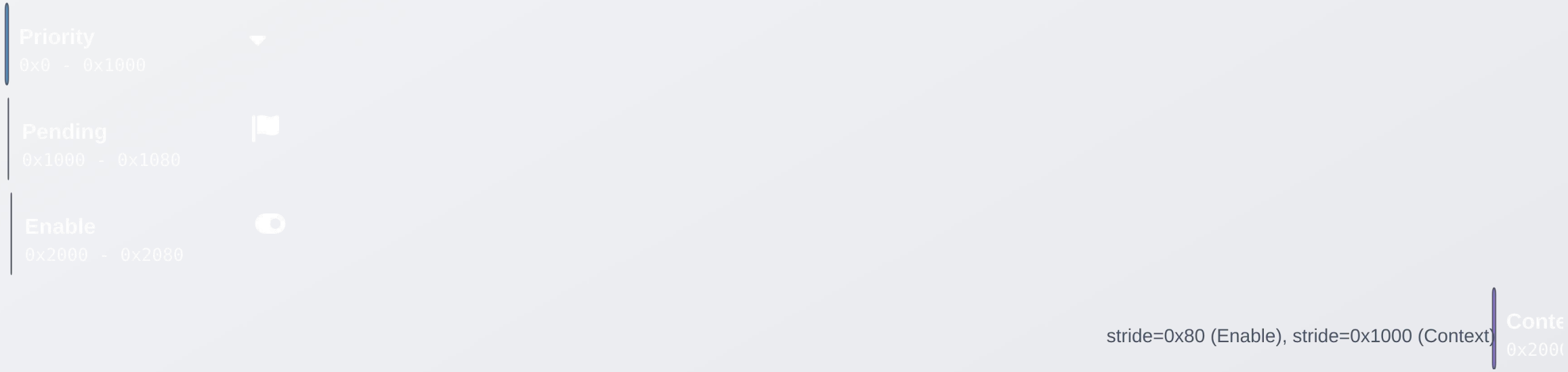
关键特性

- claim/complete 使用同一 MMIO 地址，实现双向交互
- claim 操作具有原子性，确保中断处理的正确性

PLIC 寄存器地址规范

所有寄存器都按 32-bit 访问（LW/SW 原子访问）

PLIC 寄存器内存映射 (base + offset)



关键地址定义

PLIC_OFF_PRIORITY_END
0x1000 (Priority寄存器结束地址)

PLIC_ENABLE_STRIDE
0x80 (Enable寄存器跨距)

PLIC_CTX_STRIDE
0x1000 (Context控制区跨距)

PLIC_MAX_SOURCE
1024 (最大支持中断源数量)

min实现假设与状态变量

简化假设

中断源数量

plic_nsrc = 64

ID 0..63，0 保留

上下文数量

plic_nctx = 2

ctx0 = hart0/M

ctx1 = hart0/S

pending 置位方式

允许对 pending 寄存器写入来置位 pending

非由外设网关硬件置位

状态变量

plic_priority : list(bits(32))

每个 source 的优先级

plic_pending : bits(64)

pending 位图

plic_enable_m / plic_enable_s : bits(64)

两个 context 的 enable 位图

plic_threshold_m / plic_threshold_s : bits(32)

两个 context 的 threshold

plic_claimed : bits(64)

in-service/claimed 标记

仲裁逻辑（plic_best_id_from）

function plic_best_id_from(ctx, i, best_id, best_pr)

从 i=1..63 扫描候选中断源，返回满足条件的最佳中断ID

使能条件
enable[ctx][i] == 1

挂起条件
pending[i] == 1

未claimed条件
claimed[i] == 0

优先级条件
priority[i] > threshold[ctx]

选择规则

优先级最高
选择 priority 最大的中断

ID最小规则
priority相同时选择ID最小的

算法流程

从 i=1 开始扫描

检查四个条件

比较优先级

更新最佳选择

继续扫描

update_mip 逻辑

plic_update_mip() 函数实现

```
def plic_update_mip():
    bid_m = plic_best_id(0)   # 获取M-context最佳中断ID
    bid_s = plic_best_id(1)   # 获取S-context最佳中断ID
    mip[MEI] = (bid_m != 0)          # 设置MEI标志位
    mip[SEI] = (bid_s != 0)          # 设置SEI标志位
```

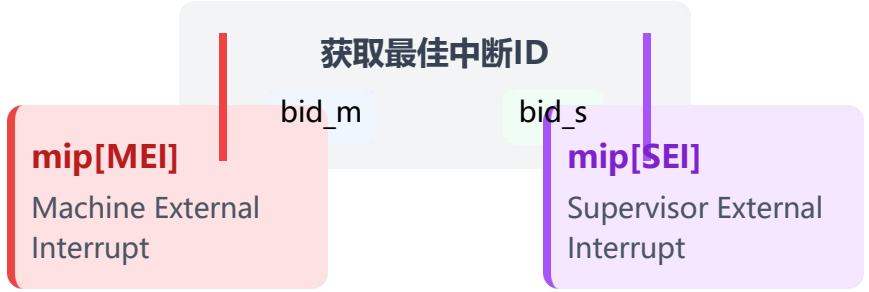
函数功能

根据每个context的最佳中断ID，更新对应的中断待处理标志位

update_mip 逻辑流程

M-context 仲裁
→ 调用 plic_best_id(0)

S-context 仲裁
→ 调用 plic_best_id(1)



逻辑说明

当bid_m!=0时，表示M-context有待处理中断，设置mip[MEI]=1；当bid_s!=0时，表示S-context有待处理中断，设置mip[SEI]=1

中断处理完整流程

软件 (OS/驱动)

1. 软件初始化

写入 priority、enable 和 threshold 寄存器

`plic_store()`

2. 外设触发 → pending置位

通过非标准注入设置 pending 位

`plic_store()`

7. CPU complete

写回中断 ID 完成处理

`plic_store()`

PLIC

3. PLIC 仲裁

选择最高优先级 pending 中断

`plic_best_id_from()`

4. 更新 mip 寄存器

设置 MEIP/SEIP 位通知 CPU

`plic_update_mip()`

6. 执行设备 ISR

根据中断 ID 分发到相应设备处理程序

CPU

5. CPU 进入 trap handler

读 claim/complete 寄存器获取中断 ID

`plic_load()`

5.1 claim 再次仲裁

状态可能变化，需要重新仲裁

`plic_best_id()`

5.2 清 pending 位

claim 时清除 pending 位并设置 claimed 位

软件步骤　PLIC 步骤　CPU 步骤

函数调用使用等宽字体显示，如 *plic_store()*

Min实现限制与优化方向

当前限制

sources 只有 64
OS可能需要更多；规范默认最大 1024

contexts 只有 2
仅覆盖 hart0 M/S；xv6/Linux 需要每 hart 的 S-context

pending 注入是非标准
通过软件写入而非网关硬件置位

priority 用 list
扩展到 1024 entries 时，应使用 vector 数据结构

优化方向

扩展 sources → 1024
支持标准 PLIC 规范的最大中断源数量

扩展 contexts → 多hart
支持多 hart 环境下的每个 hart 的 S-context（甚至 M-context）

替换 pending 注入 → 外设网关输入
使用外设网关硬件置位 pending 位，符合真实硬件行为

目标： 使 min 实现逐步接近标准 PLIC 规范，兼容 xv6/Linux 等OS需求

关键公式与语义备份

寄存器地址公式

Priority 寄存器

地址公式: `base + 0x4 + 4*id`

id 从 0 到 1023，每个 priority 占 4 字节

Pending 寄存器

地址公式: `base + 0x1000 + (id/32)*4`

1024 bits = 128 bytes = 0x80

Enable 寄存器

地址公式: `base + 0x2000 + ctx*0x80`

stride=0x80，1024 sources → 128 bytes

Context 控制区

地址公式: `base + 0x200000 + ctx*0x1000`

每个 context 一页 4KB，stride=0x1000

claim/complete 语义

PLIC 内部状态

pending[id]
待处理中断位图

claimed[id]
已声明中断位图

claim/complete 操作

→ 读操作 (claim)
原子地选择最高优先级 pending 中断
返回 ID 并清空对应 pending bit

← 写操作 (complete)
写回处理完的中断 ID
清空 claimed bit

原子操作语义

- claim 操作同时执行 "清pending" 和 "设置claimed" 两个动作
- 如果没有可用中断，claim 操作返回 0