# 1. 当前 TLB 大小与组织

- **I-TLB / D-TLB 容量**
  - `iTlbSize = 4`、`dTlbSize = 4`。
  - `tlbWayCount = 2`，因此 `setCount = portTlbSize / tlbWayCount = 4 / 2 = 2`。
  - 每个端口的 TLB 结构：
    - **2 组（sets） × 2 路（ways） = 4 项**。
    - 每组可以存 2 条 TLB 条目。
- **分区后的 set 划分（每个端口）**
  - `tlbSecureSetCount = 1` → `partitionSecureSetCount = 1`。
  - `partitionNonSecureSetCount = setCount - secureSetCount = 1`。
  - 也就是：**1 组 non-secure + 1 组 secure**，两组大小完全对称。

---

# 2. 分区细节

根据 `MmuPortConfig` 的派生字段计算结果：

- `setsPerSecureDomain = 1`
- `secureSetCount = 1`
- `partitionDomainCapacity = secureSetCount / setsPerSecureDomain = 1`
- `maxSecureDomains = 1`（命令行里为 0 → 回落为 `partitionDomainCapacity`）
- `partitionSidWidth = log2Up(maxSecureDomains + 1) = 1` bit

因此，对当前 netlist 来说：

- **SID 取值空间**：`SID ∈ {0, 1}`
  - `SID = 0`：非安全域（NS）。
  - `SID = 1`：唯一的安全域。
- **set 物理布局（每端口）**：
  - `nonSecureSetCount = 1` → `secureBaseSet = 1`。
  - 可以理解为：
    - `set 0`：**NS 组**（只给 SID=0 用）。
    - `set 1`：**secure 组**（只给 SID=1，用于安全进程）。

---

# 3. 分配 / 释放 / Flush 策略（硬件侧）

这些逻辑都在 `MmuPlugin` 中实现完备：

- **Lookup / Refill 使用的 SID**
  - 查找路径用的是 `csr.partition.currentSid`（CSR `TLB_SID`）。
  - Refill 状态机在从 `IDLE → L1_CMD` 时把当时的 `currentSid` 锁存到 `refillSid`，后续写回只用 `refillSid`，避免上下文切换污染别的域。
- **alloc（`TLB_ALLOC_SID` + `TLB_CMD.bit0`）**

    - 条件：`allocSid ∈ [1, maxSecureDomains]`（当前就是 1）。
    - 行为：
        - 计算该 SID 对应的 secure set 区间（这里就是那唯一一组）。
        - 在 `partitionTable` 中把相应 entry 的 `sid` 设为 `allocSid`，`allocated` 置位。
        - 把该 secure set 中所有 TLB 行的 `valid` 清零。
        - 在端口 `id == 0` 上把 `TLB_STATUS.bit0` 置 1 表示"本次 alloc 命令已处理"。

- **free（`TLB_FREE_SID` + `TLB_CMD.bit1`）**
    - 条件：`freeSid ∈ [1, maxSecureDomains]`。
    - 行为：
        - 遍历所有 secure set，找到 `allocated && sid == freeSid` 的 entry。
        - 把这些 entry 的 `allocated` 清 0，`sid` 置回 0。
        - 同时把对应 secure set 的 TLB 行全部清 `valid`。
        - 在端口 `id == 0` 上把 `TLB_STATUS.bit1` 置 1。

- **按 SID flush（`TLB_FLUSH_SID` + `TLB_CMD.bit2`）**
    - 遍历所有 secure set，找到 `allocated && sid == flushSid` 的 entry，只清这些 set 的 TLB 行。
    - 在端口 `id == 0` 上把 `TLB_STATUS.bit2` 置 1。

- **全局 flush（`TLB_CMD.bit3`）**
    - 不看 SID，直接把所有端口、所有 set 的 `valid` 清零，相当于一个"硬件级 `SFENCE.VMA`"。

---

# 4. NS 复用 S Sets 的实现状态

`chooseLookupSet` 里控制复用行为的代码是一个**编译期条件**：

```
if(port.args.allowNonSecureOnFreeSecureSets) {
  when(sid === 0 && !partitionTable(...).allocated) {
    chosen := secureProbe   // NS 落到空闲的 secure set
  }
}
```

- `allowNonSecureOnFreeSecureSets = false`（当前配置 `--tlb-allow-ns-reuse=False`）

---

# 5. 进程管理预设流程

## 5.1 概念约定

- **SID=0**：非安全域（NS），所有普通进程默认使用，永远不需要 alloc/free。
- **SID ∈ [1, maxSecureDomains]**：安全域，由 OS 从一个 SID 池中分配，绑定到具体的地址空间（`mm_struct`）。
- **SID 的分配者**：OS 内核，具体由内核维护的一个小位图（`sid_used[1..maxSecureDomains]`）来管理哪些 SID 空闲。

- **SID 的存储位置**：存在每个 `mm_struct` 的 `tlb_sid` 字段中（per 地址空间，共享同一 mm 的线程天然共享同一 SID）。
- **SID 的传递路径**：`mm_struct.tlb_sid` → 调度器 `switch_to()` → `csrw TLB_SID, sid`（写 CSR）→ 硬件 `currentSid`。

---

## 5.2 NS 进程的完整生命周期

```
进程创建（普通进程）
  OS：mm->tlb_sid = 0    （默认，无需任何 TLB 操作）

进程被调度运行
  OS：csrw SATP, 进程页表    （切换页表，标准流程）
      csrw TLB_SID, 0          （写入 currentSid = 0）
  硬件：后续所有 TLB lookup/refill 只使用 set 0（NS 组）

进程退出
  OS：无需任何 TLB 域操作     （SID=0 不占用任何 partitionTable entry）
```

---

## 5.3 安全进程的完整生命周期

```
进程申请成为安全域（syscall / prctl）
  OS：从 SID 池中找一个空闲 sid ∈ [1, maxSecureDomains]
      若无空闲：返回 -EBUSY
      mm->tlb_sid = sid
      csrw TLB_ALLOC_SID, sid
      csrw TLB_CMD, 0b0001            （bit0=allocTrigger）
      （可选）读 TLB_STATUS.bit0 确认成功
  硬件：在 partitionTable 中标记该 SID 对应的 secure set 为 allocated
        把该 secure set 的所有 valid 清零（清除可能残留的旧数据）

进程被调度运行
  OS：csrw SATP, 进程页表
      csrw TLB_SID, mm->tlb_sid       （写入对应的安全 SID）
  硬件：后续所有 TLB lookup/refill 只使用该 SID 对应的 secure 组

进程退出或主动释放安全域
  OS：csrw TLB_FREE_SID, mm->tlb_sid
      csrw TLB_CMD, 0b0010            （bit1=freeTrigger）
      mm->tlb_sid = 0
      把 sid 归还给 SID 池
  硬件：把 partitionTable 中该 SID 的 entry 清零（allocated = False）
        把该 secure set 的所有 valid 清零（防止数据残留）
```

---

## 5.4 进程切换时的替换操作

下面列出所有场景，明确 OS 和硬件各自的动作。

**场景 A：NS 进程 → NS 进程**

```
OS 动作：
  1. csrw SATP, next->mm->pgd
  2. csrw TLB_SID, 0
  （可根据地址空间变化决定是否 SFENCE.VMA，标准流程不变）

硬件动作：
  - currentSid 保持为 0
  - lookup/refill 只在 set 0（NS 组）中进行
  - secure set 完全不受影响
```

**场景 B：NS 进程 → 安全进程**

```
OS 动作：
  1. csrw SATP, next->mm->pgd
  2. csrw TLB_SID, next->mm->tlb_sid     （例如写入 1）

硬件动作：
  - currentSid 切换为 1
  - lookup/refill 只在 SID=1 对应的 secure set 中进行
  - set 0（NS 组）不受影响，其中的旧条目安全隔离
```

**场景 C：安全进程 A → 安全进程 B（不同 SID，当前配置不支持）**

> 当前 `maxSecureDomains=1`，只有一个安全 SID，此场景需要将 `maxSecureDomains` 扩展为 ≥2 后才能实现。

```
前提：A 使用 SID=1，B 使用 SID=2，硬件已配置 maxSecureDomains≥2

OS 动作：
  1. csrw SATP, next->mm->pgd
  2. csrw TLB_SID, next->mm->tlb_sid     （写入 2）

硬件动作：
  - currentSid 切换为 2
  - lookup/refill 只在 SID=2 对应的 secure 组
  - SID=1 对应的 secure 组保留，A 下次被调度时仍有效
  - 两个 secure 组在 partitionTable 层面严格隔离，互不干扰
```

**场景 D：安全进程 A → 同 SID=1 的另一安全进程（当前配置下的复用）**

> 当前只有一个安全 SID，若两个"安全"进程都被分配到 SID=1，它们之间没有 TLB 空间隔离，OS 必须用强制清洗来保证安全。

```
OS 动作（切出 A、切入 B 时）：
  1. csrw TLB_FLUSH_SID, 1
     csrw TLB_CMD, 0b0100              （bit2=flushSidTrigger，清空 SID=1 的条目）
  2. csrw SATP, B->mm->pgd
  3. csrw TLB_SID, 1

硬件动作：
  - flushSid 把 secure set 中所有 valid 清零
  - B 进来时 secure set 是空的，无法观察 A 的 TLB 历史
  - B 的后续访问重新填入，不会看到 A 的任何痕迹
```

---

## 5.5 整体流程图

```
用户态 syscall                内核（OS）                    硬件（VexRiscv TLB）
──────────────────────────────────────────────────────────────────────────────

进程申请安全域
  │                   从 sid_pool 分配 sid
  │                   mm->tlb_sid = sid
  │                   csrw TLB_ALLOC_SID, sid ──────→ partitionTable[sid] 标记为
allocated
  │                   csrw TLB_CMD, 0x1       ──────→ secure set 全部 valid=0

调度器运行
  │                   csrw SATP, mm->pgd      ──────→ 切换页表
  │                   csrw TLB_SID, sid        ──────→ currentSid = sid
  │                         （后续 CPU 访问）   ──────→ lookup/refill 只在 secure
set[sid] 中

调度器切走
  │                   csrw TLB_SID, next_sid  ──────→ currentSid = next_sid
  │                   csrw SATP, next->mm->pgd ─────→ 切换页表

进程退出
  │                   csrw TLB_FREE_SID, sid  ──────→
partitionTable[sid].allocated = false
  │                   csrw TLB_CMD, 0x2       ──────→ secure set 全部 valid=0
  │                   sid 归还给 sid_pool
  │                   mm->tlb_sid = 0
```