目录

[一. 软件整体架构设计 2](#_Toc24812)

[1.1RDMA通用架构和支持的功能点 2](#_Toc14333)

[1.2 RDMA驱动架构 4](#_Toc12176)

[1.3模块划分说明 5](#_Toc6575)

[二. RDMA设备对象模型 7](#_Toc21309)

[2.1 对象模型与全局结构 7](#_Toc16265)

[2.2 初始化与卸载顺序 11](#_Toc28079)

[三. PCIE驱动层设计 12](#_Toc4171)

[3.1 Pcie设备探测与channel探测 12](#_Toc20159)

[3.2 网卡发现 12](#_Toc22582)

[3.3 添加与删除rdma设备 12](#_Toc1614)

[3.4 VF划分 13](#_Toc4546)

[3.5 bonding 13](#_Toc3740)

[四. 内存管理 16](#_Toc22807)

[4.1 硬件内存管理方案 16](#_Toc6782)

[4.2 软件内存管理方案 16](#_Toc2036)

[4.3 页表生成策略 17](#_Toc11780)

[4.3 队列管理模型 19](#_Toc9774)

[4.4 SRQ的特殊处理 20](#_Toc7568)

[4.5 sq/rq sw\_cmd 管理 22](#_Toc31483)

[五. RDMA资源管理 23](#_Toc11890)

[5.1通用资源管理层 23](#_Toc11671)

[5.2 软件资源管理 24](#_Toc11643)

[5.2.2 ucontext & pd管理 25](#_Toc6149)

[5.2.3 ah管理 25](#_Toc13105)

[5.2.4 CQ管理 27](#_Toc17667)

[5.2.4 SQ/RQ管理 27](#_Toc19071)

[5.2.5 MR & MW管理 28](#_Toc6755)

[5.2.6 QP管理 30](#_Toc5662)

[六. IO路径管理 31](#_Toc31592)

[6.1 SQ 31](#_Toc12808)

[6.2 RQ 31](#_Toc19165)

[6.3 SRQ 32](#_Toc14767)

[6.4 CQ 32](#_Toc13047)

[七. 中断管理 34](#_Toc24789)

[八. 用户态与内核态接口 36](#_Toc2377)

[九. 统计与Debug调试接口 39](#_Toc15412)

[9.1 标准统计 39](#_Toc5967)

[9.2 软件debug 41](#_Toc89)

[十. 硬件抽象接口层 42](#_Toc5212)

[10.1 Host 抽象操作 42](#_Toc28765)

[10.2 SF抽象操作 42](#_Toc17392)

[10.3 IO操作 44](#_Toc17250)

[十一. 用户态provider 架构 45](#_Toc959)

[十二. 2100r 固件/硬件接口层设计 46](#_Toc16125)

[十三. 2100r硬件资源管理 47](#_Toc12018)

[十四. 2100r寄存器操作流程 50](#_Toc30200)

[13.1 初始化 50](#_Toc10341)

[13.2 DB 50](#_Toc22029)

[13.3 AEQ处理流程 50](#_Toc11255)

[13.4 Debug 51](#_Toc17544)

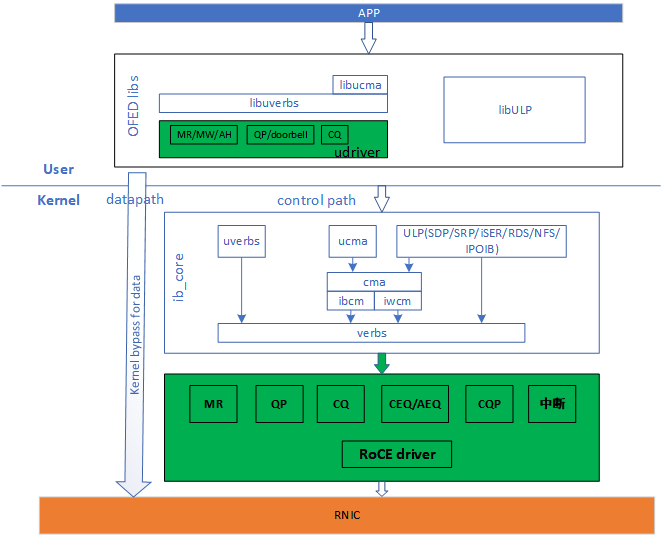
[十五. 用户态工具 52](#_Toc18205)

[15.1 HQOS 52](#_Toc18676)

## 软件整体架构设计

## 1.1RDMA通用架构和支持的功能点

通用Rdma驱动整体架构

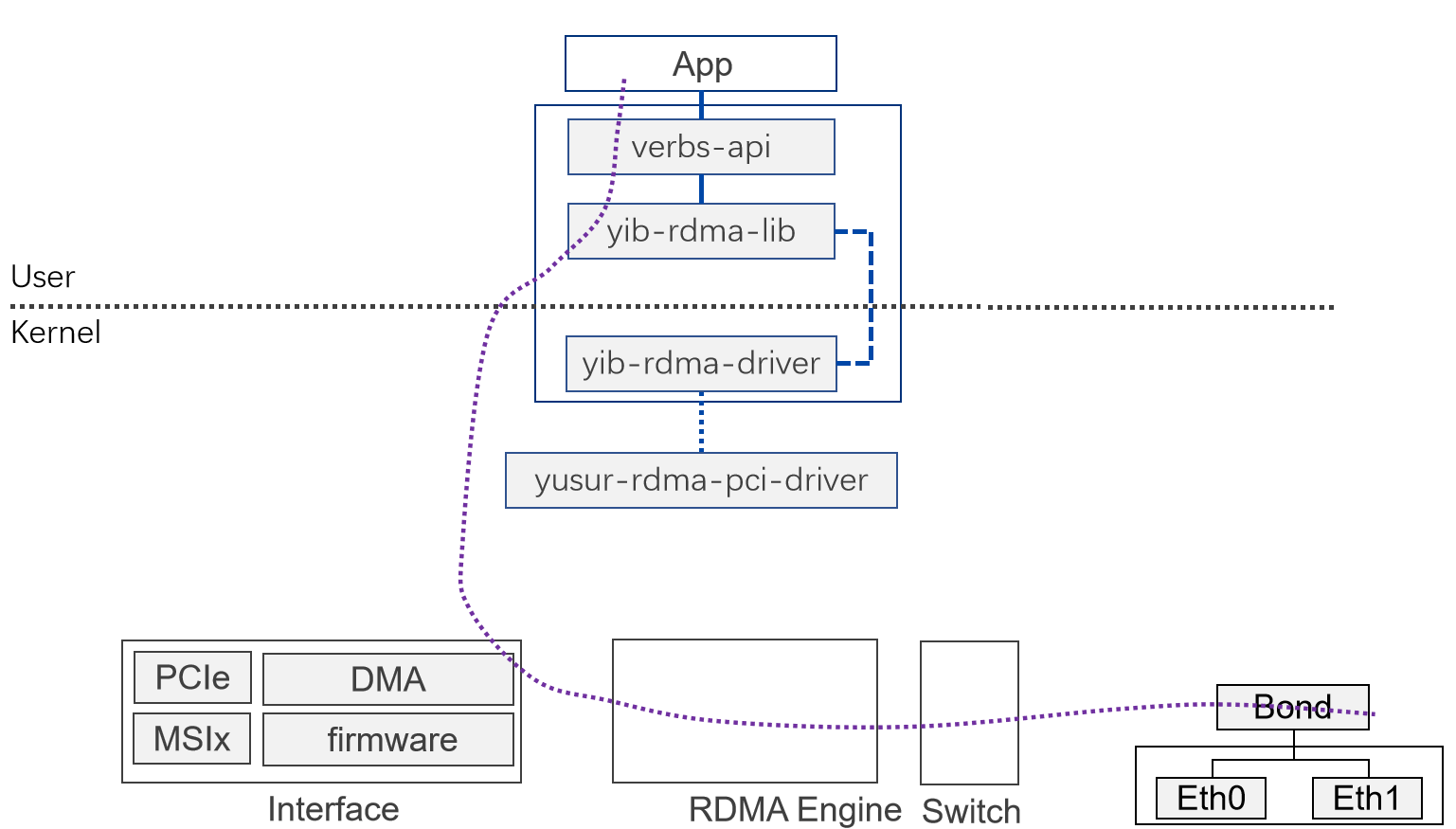


Rdma驱动包含用户态驱动和内核态驱动2个部分。

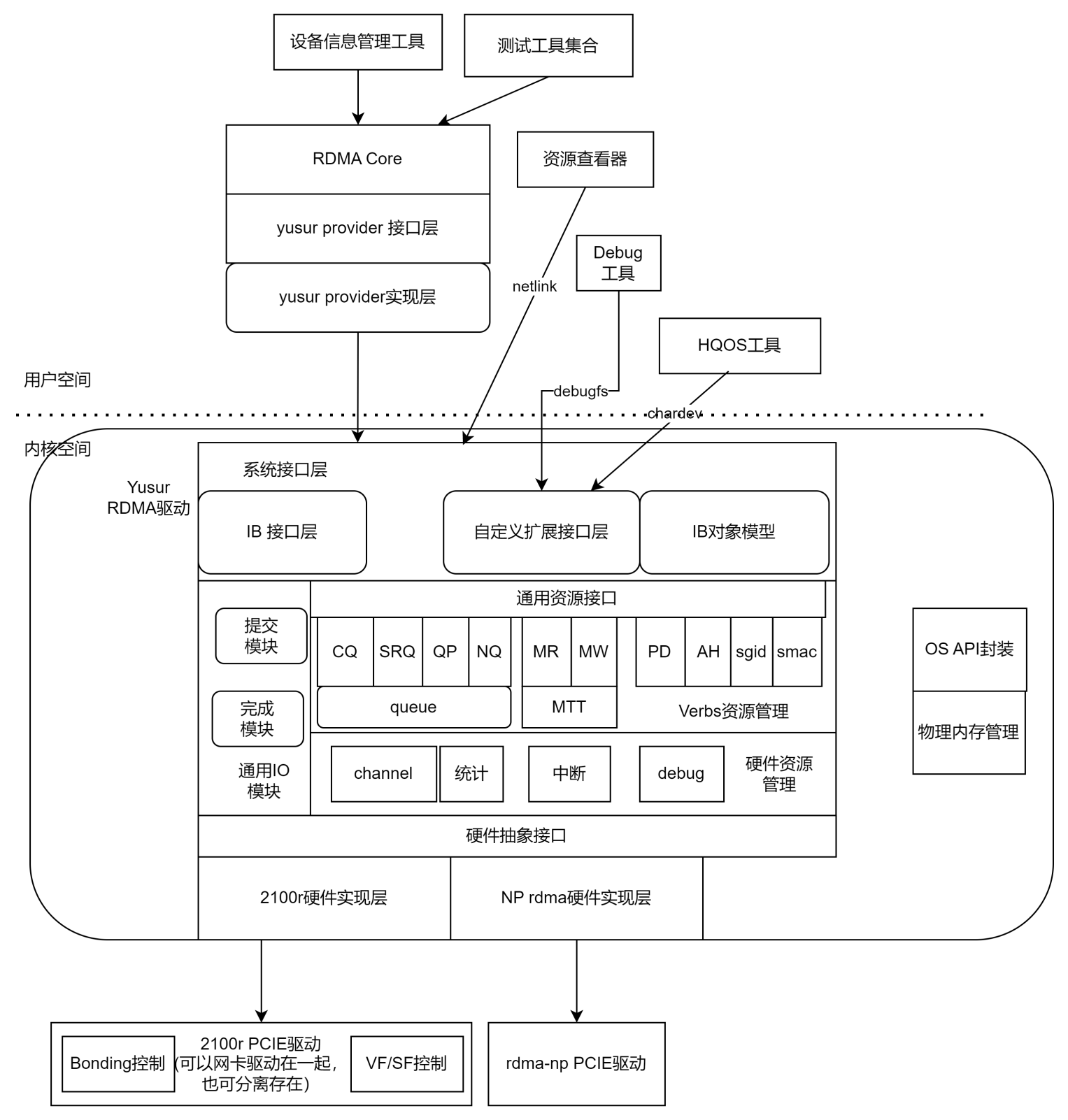
基本功能说明

|  |  |  |
| --- | --- | --- |
| **类别** | **功能与特性** | **测试说明** |
| 基本功能 | RoCEv2：RC与UD | 单QP/多QP下，RC与UD连接测试 |
| PMTU，Inline，Fence，Go\_back\_N，SE | RC模式下，单QP/多QP下，基础机制测试 |
| Send only/first/middle/last, Send with immediate, Send with Invalidate | RC模式下，单QP/多QP下，Send相关功能 |
| Write only/first/middle/last, Write with immediate | RC模式下，单QP/多QP下，Write相关功能 |
| Read request, response only/fisrt/middle/last | RC模式下，单QP/多QP下，Read相关功能 |
| Atomic FetchAdd/CmpSwap | 暂不支持，最终支持 |
| 功能符合Rocev2标准 | 可和第三方网卡互通 |
| Local\_invalidate, Fast-Register\_MR | 支持 |
| Bind\_MW | 支持 |
| SRQ | 支持（软件保证队列完成一致性） |
| 规格 | 63个PF/VF RDMA功能 | 可以支持63个PF/VF跑RDMA特性 |
| 最大X个 QP | 可以创建满规格QP跑业务 |
| 流量控制与调度 | PFC | PFC基本功能：反压与停流 |
| ETS&Shaping | PF/VF粒度shaping限速 |
| 队列优先级 | Qp支持 DSCP粒度优先级调度 |
| 拥塞控制算法 | DCQCN | 8打1流量模型组网；8台组网all to all流量测试 |
| 用户自定义算法 | 支持用户可编程自定义算法 |

## 1.2 RDMA驱动架构



软硬件数据流图



软件模块划分

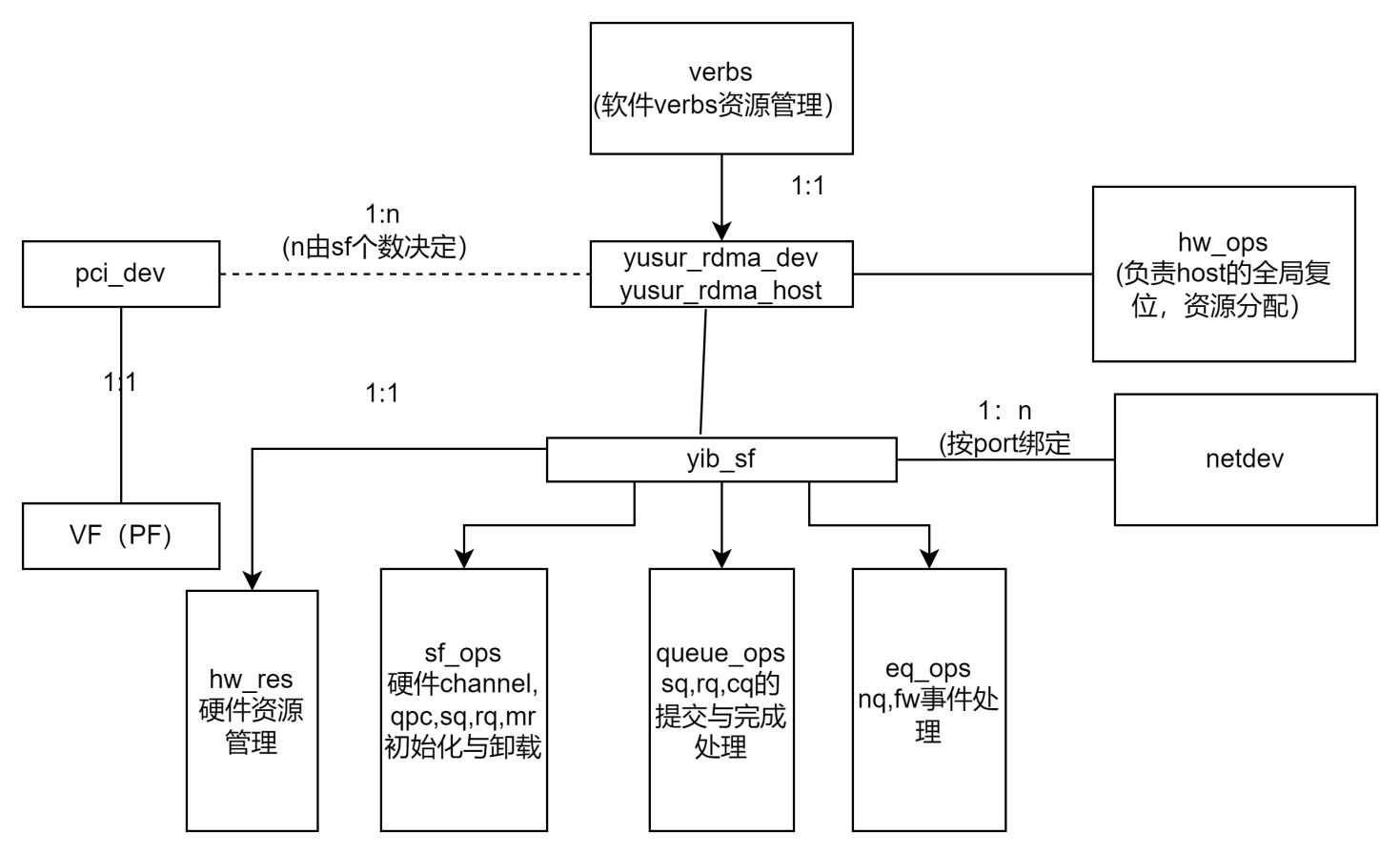
## 1.3模块划分说明

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 层级 | 模块或应用 | 功能职责说明 |
| 1 | 应用层 | 设备信息管理工具 | 1. 提供yib\_ibdev2net 负责获取网卡和rdma设备的对应信息 2. Show\_gids 列举所有设备信息 |
| 2 | 应用层 | 资源查看器 | Qp,cq,mr,等资源信息的查看， 基于netlink接口实现。 支持用户模式和debug模式两种。debug模式下，能看下能看到部分硬件内部状态。 |
| 3 | 应用层 | debug工具 | 1. 驱动调试模式配置 2. 选择Qp进入跟踪模式 |
| 4 | 应用层 | HQos工具 | （1） yib\_qos运行在虚拟机环境下配置cos. 这时能知道vm,host; 宿主机上只能配置pf的。 -d 指定设配 -p指定port(通知支持dscp->cos的映射配置）  （2） yib\_vm\_qos工具，运行在宿主机上，配置vm,host,port的值。  -type vm/host -d 设备 -p port |
| 5 | 应用库 | Rdma-core yib provider | rdma用户态驱动 |
| 6 | 内核态pcie驱动 | Yusur rdma pcie driver | 提供两种模式的pcie驱动：   1. RDMA独立模式 --- 用于前期调试和模拟器下使用 2. RDMA pcie做为网卡驱动的一部分存在，用于正式的发布产品 3. bonding模式的管理放在该层实现 |
| 7 | 内核态rdma驱动 | 系统接口层 | 1. 对象模型层是按ib框架的要求，进行系统初始化，能力查询，添加ib对象。 2. IB接口层，负责实现各种ib指令，如create\_qp, reg\_mr等操作的接口，是整个操作的入口层。 3. 自定义扩展接口包含 4. chardev接口，用户qos和其他正式功能的自定义命令 5. debugfs用于内部调试， 存在于debug版本中，部分功能对用户关闭。 |

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | 内核态rdma驱动 | 通用资源管理层 | 所有资源分配的基础：  软件资源对象管理接口，用于维护软件资源   1. QP 2. MR 3. CQ 4. NQ 5. SRQ 6. PD 7. SGID\_tbl 8. Smac\_tbl 9. AH   包含：添加，删除，查询，配置接口 |
| 9 | 内核态rdma驱动 | 内存管理 | 1.硬件资源对应的物理内存管理  2. 页表内存管理  3.队列内存管理 |
| 10 | 内核态rdma驱动 | 统计与debug | 实现统计与debug功能 |
| 13 | 内核态rdma驱动 | 通用IO路径 | 1.IO提交（维护软件统计信息）  2.IO完成（维护软件统计信息）  3.中断管理 |
| 14 | 内核态rdma驱动 | 硬件抽象接口 | 完成软件与具体硬件代码的分离 |
| 12 | 内核态rdma驱动 | 2100r硬件实现层 | 分为固件接口，硬件资源管理， io路径3个子模块 |
| 13 | 内核态rdma驱动 | Rdma-np硬件实现层 | doe资源管理， io路径， doe调度管理3个子模块 |

## RDMA设备对象模型

### 2.1 对象模型与全局结构

%3CmxGraphModel%3E%3Croot%3E%3CmxCell%20id%3D%220%22%2F%3E%3CmxCell%20id%3D%221%22%20parent%3D%220%22%2F%3E%3CmxCell%20id%3D%222%22%20value%3D%22VF%EF%BC%88PF)%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22120%22%20y%3D%22210%22%20width%3D%2290%22%20height%3D%2230%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%223%22%20value%3D%221%3A1%22%20style%3D%22text%3Bhtml%3D1%3BstrokeColor%3Dnone%3BfillColor%3Dnone%3Balign%3Dcenter%3BverticalAlign%3Dmiddle%3BwhiteSpace%3Dwrap%3Brounded%3D0%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22135%22%20y%3D%22140%22%20width%3D%2260%22%20height%3D%2230%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%224%22%20value%3D%22%22%20style%3D%22endArrow%3Dnone%3Bhtml%3D1%3Brounded%3D0%3BexitX%3D0.5%3BexitY%3D0%3BexitDx%3D0%3BexitDy%3D0%3BentryX%3D0.5%3BentryY%3D1%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%222%22%20target%3D%225%22%20parent%3D%221%22%3E%3CmxGeometry%20width%3D%2250%22%20height%3D%2250%22%20relative%3D%221%22%20as%3D%22geometry%22%3E%3CmxPoint%20x%3D%22380%22%20y%3D%22290%22%20as%3D%22sourcePoint%22%2F%3E%3CmxPoint%20x%3D%22200%22%20y%3D%22160%22%20as%3D%22targetPoint%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%225%22%20value%3D%22pci\_dev%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22120%22%20y%3D%2290%22%20width%3D%2290%22%20height%3D%2230%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%226%22%20value%3D%22%22%20style%3D%22endArrow%3Dnone%3Bdashed%3D1%3Bhtml%3D1%3Brounded%3D0%3BexitX%3D1%3BexitY%3D0.5%3BexitDx%3D0%3BexitDy%3D0%3BentryX%3D0%3BentryY%3D0.5%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%225%22%20target%3D%228%22%20parent%3D%221%22%3E%3CmxGeometry%20width%3D%2250%22%20height%3D%2250%22%20relative%3D%221%22%20as%3D%22geometry%22%3E%3CmxPoint%20x%3D%22380%22%20y%3D%22290%22%20as%3D%22sourcePoint%22%2F%3E%3CmxPoint%20x%3D%22420%22%20y%3D%22105%22%20as%3D%22targetPoint%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%227%22%20value%3D%221%3An%26lt%3Bbr%26gt%3B(n%E7%94%B1sf%E4%B8%AA%E6%95%B0%E5%86%B3%E5%AE%9A%EF%BC%89%22%20style%3D%22text%3Bhtml%3D1%3BstrokeColor%3Dnone%3BfillColor%3Dnone%3Balign%3Dcenter%3BverticalAlign%3Dmiddle%3BwhiteSpace%3Dwrap%3Brounded%3D0%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22230%22%20y%3D%2270%22%20width%3D%22110%22%20height%3D%2230%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%228%22%20value%3D%22yusur\_rdma\_dev%26lt%3Bbr%26gt%3Byusur\_rdma\_host%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22380%22%20y%3D%2290%22%20width%3D%22120%22%20height%3D%2230%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%229%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BexitX%3D0.25%3BexitY%3D1%3BexitDx%3D0%3BexitDy%3D0%3B%22%20edge%3D%221%22%20source%3D%2211%22%20target%3D%2215%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2210%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%2211%22%20target%3D%2216%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2211%22%20value%3D%22yib\_sf%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22360%22%20y%3D%22180%22%20width%3D%22160%22%20height%3D%2220%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2212%22%20value%3D%22%22%20style%3D%22endArrow%3Dnone%3Bhtml%3D1%3Brounded%3D0%3BentryX%3D0.439%3BentryY%3D1.033%3BentryDx%3D0%3BentryDy%3D0%3BentryPerimeter%3D0%3BexitX%3D0.439%3BexitY%3D-0.083%3BexitDx%3D0%3BexitDy%3D0%3BexitPerimeter%3D0%3B%22%20edge%3D%221%22%20source%3D%2211%22%20target%3D%228%22%20parent%3D%221%22%3E%3CmxGeometry%20width%3D%2250%22%20height%3D%2250%22%20relative%3D%221%22%20as%3D%22geometry%22%3E%3CmxPoint%20x%3D%22380%22%20y%3D%22170%22%20as%3D%22sourcePoint%22%2F%3E%3CmxPoint%20x%3D%22430%22%20y%3D%22120%22%20as%3D%22targetPoint%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2213%22%20value%3D%22hw\_ops%26lt%3Bbr%26gt%3B(%E8%B4%9F%E8%B4%A3host%E7%9A%84%E5%85%A8%E5%B1%80%E5%A4%8D%E4%BD%8D%EF%BC%8C%E8%B5%84%E6%BA%90%E5%88%86%E9%85%8D%EF%BC%89%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22610%22%20y%3D%2265%22%20width%3D%22120%22%20height%3D%2280%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2214%22%20value%3D%22%22%20style%3D%22endArrow%3Dnone%3Bhtml%3D1%3Brounded%3D0%3BexitX%3D0%3BexitY%3D0.5%3BexitDx%3D0%3BexitDy%3D0%3BentryX%3D1%3BentryY%3D0.5%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%2213%22%20target%3D%228%22%20parent%3D%221%22%3E%3CmxGeometry%20width%3D%2250%22%20height%3D%2250%22%20relative%3D%221%22%20as%3D%22geometry%22%3E%3CmxPoint%20x%3D%22460%22%20y%3D%22140%22%20as%3D%22sourcePoint%22%2F%3E%3CmxPoint%20x%3D%22480%22%20y%3D%22105%22%20as%3D%22targetPoint%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2215%22%20value%3D%22sf\_ops%26lt%3Bbr%26gt%3B%E7%A1%AC%E4%BB%B6channel%2C%20qpc%2Csq%2Crq%2Cmr%E5%88%9D%E5%A7%8B%E5%8C%96%E4%B8%8E%E5%8D%B8%E8%BD%BD%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22284%22%20y%3D%22240%22%20width%3D%2276%22%20height%3D%22120%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2216%22%20value%3D%22queue\_ops%26lt%3Bbr%26gt%3Bsq%2Crq%2Ccq%E7%9A%84%E6%8F%90%E4%BA%A4%E4%B8%8E%E5%AE%8C%E6%88%90%E5%A4%84%E7%90%86%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22407%22%20y%3D%22240%22%20width%3D%2266%22%20height%3D%22120%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2217%22%20value%3D%22eq\_ops%26lt%3Bbr%26gt%3Bnq%2Cfw%E4%BA%8B%E4%BB%B6%E5%A4%84%E7%90%86%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22500%22%20y%3D%22240%22%20width%3D%2270%22%20height%3D%22120%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2218%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BexitX%3D0.75%3BexitY%3D1%3BexitDx%3D0%3BexitDy%3D0%3BentryX%3D0.5%3BentryY%3D0%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%2211%22%20target%3D%2217%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%3E%3CmxPoint%20x%3D%22410%22%20y%3D%22210%22%20as%3D%22sourcePoint%22%2F%3E%3CmxPoint%20x%3D%22332%22%20y%3D%22250%22%20as%3D%22targetPoint%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2219%22%20value%3D%22%22%20style%3D%22endArrow%3Dnone%3Bhtml%3D1%3Brounded%3D0%3BexitX%3D1%3BexitY%3D0.5%3BexitDx%3D0%3BexitDy%3D0%3B%22%20edge%3D%221%22%20source%3D%2211%22%20parent%3D%221%22%3E%3CmxGeometry%20width%3D%2250%22%20height%3D%2250%22%20relative%3D%221%22%20as%3D%22geometry%22%3E%3CmxPoint%20x%3D%22380%22%20y%3D%22190%22%20as%3D%22sourcePoint%22%2F%3E%3CmxPoint%20x%3D%22630%22%20y%3D%22190%22%20as%3D%22targetPoint%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2220%22%20value%3D%221%EF%BC%9An%26lt%3Bbr%26gt%3B(%E6%8C%89port%E7%BB%91%E5%AE%9A%22%20style%3D%22text%3Bhtml%3D1%3BstrokeColor%3Dnone%3BfillColor%3Dnone%3Balign%3Dcenter%3BverticalAlign%3Dmiddle%3BwhiteSpace%3Dwrap%3Brounded%3D0%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22530%22%20y%3D%22160%22%20width%3D%2280%22%20height%3D%2230%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2221%22%20value%3D%22netdev%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22620%22%20y%3D%22160%22%20width%3D%22120%22%20height%3D%2260%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2222%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%2223%22%20target%3D%228%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2223%22%20value%3D%22verbs%26lt%3Bbr%26gt%3B(%E8%BD%AF%E4%BB%B6verbs%E8%B5%84%E6%BA%90%E7%AE%A1%E7%90%86%EF%BC%89%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22380%22%20y%3D%22-10%22%20width%3D%22120%22%20height%3D%2260%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2224%22%20value%3D%22hw\_res%26lt%3Bbr%26gt%3B%E7%A1%AC%E4%BB%B6%E8%B5%84%E6%BA%90%E7%AE%A1%E7%90%86%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22195%22%20y%3D%22250%22%20width%3D%2260%22%20height%3D%22110%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2225%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BexitX%3D0%3BexitY%3D0.25%3BexitDx%3D0%3BexitDy%3D0%3BentryX%3D0.5%3BentryY%3D0%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%2211%22%20target%3D%2224%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%3E%3CmxPoint%20x%3D%22410%22%20y%3D%22210%22%20as%3D%22sourcePoint%22%2F%3E%3CmxPoint%20x%3D%22332%22%20y%3D%22250%22%20as%3D%22targetPoint%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2226%22%20value%3D%221%3A1%22%20style%3D%22text%3Bhtml%3D1%3BstrokeColor%3Dnone%3BfillColor%3Dnone%3Balign%3Dcenter%3BverticalAlign%3Dmiddle%3BwhiteSpace%3Dwrap%3Brounded%3D0%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22255%22%20y%3D%22145%22%20width%3D%2260%22%20height%3D%2230%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2227%22%20value%3D%221%3A1%22%20style%3D%22text%3Bhtml%3D1%3BstrokeColor%3Dnone%3BfillColor%3Dnone%3Balign%3Dcenter%3BverticalAlign%3Dmiddle%3BwhiteSpace%3Dwrap%3Brounded%3D0%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22450%22%20y%3D%2250%22%20width%3D%2260%22%20height%3D%2230%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3C%2Froot%3E%3C%2FmxGraphModel%3E

struct yusur\_rdma\_dev {

struct yrdma\_device ydev;

struct pci\_dev \*pdev;

struct net\_device \*ndev[YIB\_MAX\_DEV\_PORTS];

int max\_ports;//设备支持的最大port数量

void \*global\_bar\_virtual\_addr[BAR\_SPACES\_MAX]; //rdma全局bar虚拟地址

u32 bar\_size[BAR\_SPACES\_MAX];//bar空间的大小

enum yrdma\_host\_type host\_type;

u32 chip\_subtype;//扩展使用目前填0

bool bonding\_mode;

u32 irq\_vector; //给rdma设备的起始中断向量,要求连续分配

int irq\_cnt; //中断向量的个数

void \*priv\_data;

func\_get\_netdev\_t get\_from\_bonding;//pcie驱动层赋值， 给底层的回调函数，用于查询再bonding模式下当前使用的netdev

func\_get\_left\_sfs get\_left\_sfs;//由底层赋值，pcie驱动层再sriov切割时查询sf的数量

struct mutex \*glb\_mutex;//全局锁，每pcie只有一个

};

pcie层驱动负责初始化上述结构（get\_left\_sfs除外）。

struct yusur\_ib\_dev {

os\_ib\_device ib\_dev;

os\_device \*dev;

bool bonding;

os\_spinlock\_t lock;

os\_net\_notifier netdev\_notifier;

struct yib\_hw\_host host;

void \*yrdev;

struct yib\_debugfs \* debugfs;//调试管理接口

}

struct yib\_ndev\_info {

os\_net\_device \*netdev; //网卡设备

int link\_down;//link状态，从网卡同步

u32 cur\_max\_mtu;//维护每个netdev对应的状态，如pmtu

int port\_id;

};

struct yib\_hw\_host {

os\_device \*dev;

//临时缓存和yib\_ibdev里一样，便于代码形式统一

struct net\_device \*net\_dev[YIB\_MAX\_DEV\_PORTS];

void \_\_iomem \*reg\_base[BAR\_SPACES\_MAX];

u32 bar\_size[BAR\_SPACES\_MAX];

u32 irq\_vector;

int irq\_cnt;

int chip\_subtype;//子类别

host\_verbs\_t verbs;//软件verbs资源管理

struct yib\_db\_frag db\_frag;//队列指针与统计信息，碎片内存管理

struct yib\_ndev\_info ndev\_info[YIB\_MAX\_DEV\_PORTS];

struct yib\_sf sf;

int sf\_per\_port;//一个sf对应多少个port

struct yib\_mac\_tbl mac\_tbl;//软件维护的mac表

struct yib\_hw\_ops hw\_ops;//host抽象层的操作

struct yib\_roce\_caps caps; //设备规模（qp,cq,rq)

struct yib\_dev\_funcs funcs;//设备能力(ud,srq使能等）

void \*hw\_priv; //具体由硬件实现层维护

void \*yib;

//设备的pcie信息

u16 vf\_id;

u8 pf\_id;

u8 host\_id;

};

设计注意事项：

1. pci驱动需要分配全局锁给sf和host（一个锁）

1. Yib\_sf用于管理每sf的硬件资源
2. host用于管理pcie全局资源，

struct yib\_sf {

struct yib\_hw\_host \*hw;

struct mutex \*host\_mutex;//pcie锁

bool bonding; //bonding状态

bool started; //硬件是否已启动

bool initialzied; //软件init是否调用完成

void \_\_iomem \*reg\_base[BAR\_SPACES\_MAX];//相对于该sf的寄存器

u32 bar\_size[BAR\_SPACES\_MAX];//sf的bar大小

struct pci\_dev \*pdev;

int num\_node;

int udp\_port;

u16 vf\_id;

u8 pf\_id;

u8 host\_id;

u32 irq\_vector;//sf的起始中断向量

int irq\_cnt;//sf中断个数

struct yib\_sf\_ops \*sf\_ops; //

struct yib\_queue\_ops \*queue\_ops; //队列操作注册函数指针

void \*sf\_priv;

struct yib\_qp \*gsi; //全局gsi qp的指针，一个sfzh

struct yib\_eq\_ops \*eq\_ops; //event队列操作

struct yib\_hw\_events cq\_cmpl\_evts;//cq完成事件

struct yib\_hw\_events cq\_err\_evts; //cq err事件

struct yib\_hw\_events qp\_fatal\_evts;//qp硬件进入 err state事件

struct yib\_hw\_events srq\_err\_evts; //sqr错误事件

struct yib\_hw\_events srq\_lastwqe\_evts;

struct yib\_hw\_events fw\_cmd\_evts; //fw通知事件

struct yib\_eq \*event\_queue[YIB\_MAX\_EQ\_NUM];//eq0: aeq eq1-n: nq

};

typedef struct

{

struct yib\_pool qp\_pool;

struct yib\_pool mrw\_pool;

struct yib\_pool rq\_pool;

struct yib\_pool cq\_pool;

struct yib\_pool eq\_pool;

struct yib\_pool pd\_pool;

struct yib\_pool ah\_pool;

bool initialzied;

} host\_verbs\_t;

### 2.2 初始化与卸载顺序

#### （1）初始化顺序

1. yusur\_hw\_ops\_verify: 各种ops的检查
2. yusur\_ib\_host\_init (从yusur\_rdma\_dev 同步信息到yusur\_ib\_dev上
3. yusur\_core\_ibdev\_init
4. yusur\_hw\_host\_find 根据设备型号匹配 填充各种ops.
5. yusur\_hw\_host\_pre\_init: 分配host->priv, 执行hw\_ops->global\_reset
6. hw\_ops->start\_host
7. sf初始话
8. Sf\_ops->sf\_pre\_init (对r2100则会向固件申请channel)
9. hw\_ops.init\_caps （获取host的能力信息）
10. hw->sf.initialzied = true
11. Yusur\_hw\_cap\_verify: host能力检测
12. yusur\_verbs\_init

根据host能力，初始化好软件verbs资源池yib\_pool\_init（cq,qp,mr,srq,pd,eq,ah)

F.yusur\_sf\_start 启动SF

1. 分配eq, sf->event\_queue[i] = yib\_eq\_alloc
2. sf->sf\_ops->start\_sf 启动sf, sf->started = true;
3. yusur\_init\_db\_frag准备碎片能存
4. 建立软件初始mac表
5. ib\_register\_device 注册ib设备流程
6. 低版本内核需要，单独注册sysfs yib\_create\_sysfs\_entries
7. yusur\_ib\_register\_net\_notifier 注册netdev mac变更获取回调handle\_netdev\_notifier
8. yusur\_debugfs\_add加载debugfs

#### （2） 卸载顺序

(1)yusur\_debugfs\_remove移除debugfs

(2) yusur\_ib\_unregister\_net\_notifier 卸载netdev mac变更回调

(3) sysfs\_remove\_group低版本内核，移除单独注册的sysfs

(4) ib\_unregister\_device 注销系统ibdev

(5) yusur\_core\_ibdev\_free

A. hw->sf.sf\_ops->stop\_sf(&hw->sf); 释放eq资源

B. host->hw\_ops.stop\_host(host);

C. yusur\_free\_db\_frag 碎片内存回收

D. yusur\_verbs\_uninit 软件资源池释放

E. yusur\_sf\_uninit sf资源回收

G.ib\_dealloc\_device ib设备资源释放

## PCIE驱动层设计

### Pcie设备探测与channel探测

1. 硬件必须提供一种方案让软件能计算出pf上能用的channel数量。

方案一： 根据bar\_size探测， channel\_num = (total\_bar\_size - reserved\_size)/channel\_rom\_size;

方案二： 硬件通过寄存器告知驱动channel\_num数量。

1. 软件接口用于在pcie层添加yusur rdma\_dev设备

说明： 一个pcie可以对应多个yusur\_rdma\_dev 用start\_sf\_id, 控制每个对应sf bar空间的顺序。(sf\_total可以是一个或多个）。pcie层不管理channel和port的对应关系。

硬件的PF的SF数量配置： 可以配置到flash中，pcie固件冷启动时识别； 或者FPGA指定。

### 网卡发现

1. 方案一：网卡和rdma在同一个pcie内， 按sf顺序对应。 网卡驱动层将netdev提交到yusur\_rdma\_dev中。 这时rdma pcie驱动和网卡pcie驱动是一个模块。
2. 方案二： 网卡和rdma在不同pcie内， 测试驱动可以指定网卡netdev,或用同一dev下不同的func来自动获取netdev。

### 添加与删除rdma设备

struct yusur\_rdma\_dev \*yusur\_rdma\_add\_dev(struct yusur\_rdma\_dev \*yrdev)；

void yusur\_rdma\_remove\_dev(struct yusur\_rdma\_dev \*yrdma\_dev);

### VF划分

typedef int (\*func\_get\_left\_sf\_cnt)(struct yusur\_rdma\_dev\*);

struct yusur\_rdma\_dev

{

......

func\_get\_left\_sfs get\_left\_sf; //这个要在rdma驱动中初始化

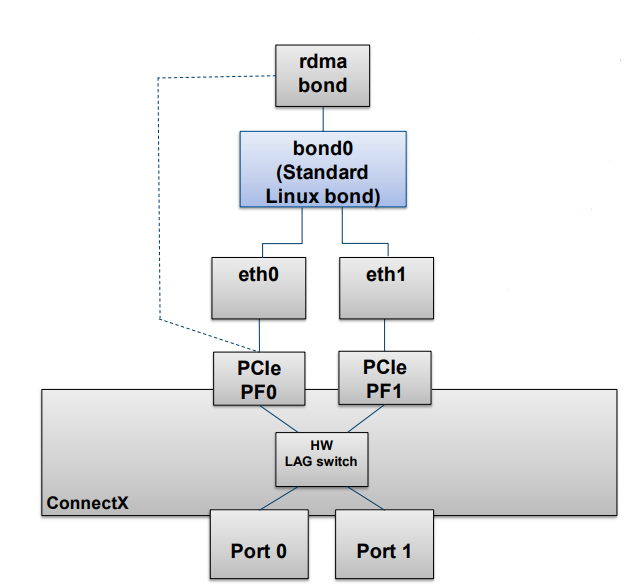
......

}

实现：sriov\_configure(n)

1. 调用固件接口查询，可以划分的剩余channel=yrdev->get\_left\_sf(yrdev)数量
2. 当channel数量少于n时，报错。
3. 进行sriov 切割。

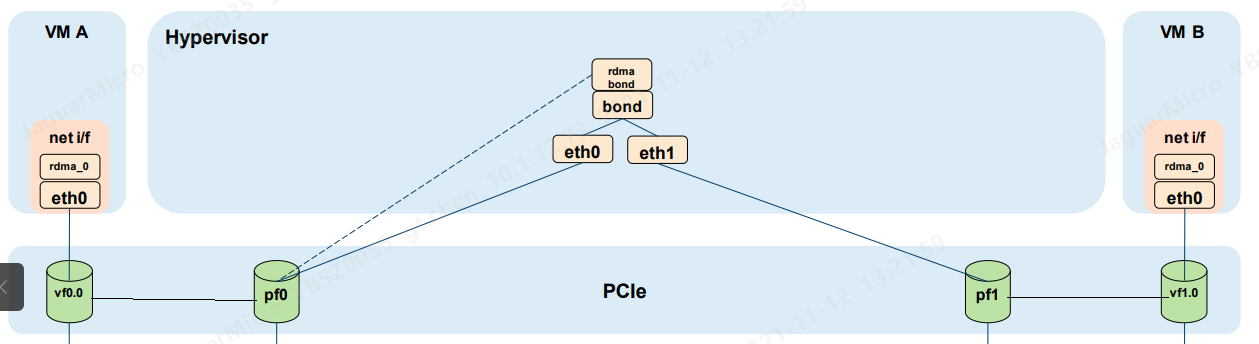
### bonding



设计要点：

（1）满足要聚合的pf在同一个pcie设备上。

（2）rdma vf的bond要求先聚合对应的pf, vf不直接支持聚合



Rdma的实现策略如下：

1.将yusur\_xx建立在pf0对应的pdev上。（yusur\_rdma\_dev(pf0,bonding\_mode))

2.将bonding模式传递给硬件

3.软件感知bonding的变化，对yusur\_rdma\_dev做如下操作：

A.将绑定取消时

（yusur\_rdma\_dev(pf0,bonding\_mode, bondidng\_ndev)) ---->

yusur\_rdma\_dev(pf0, single,ndev0)

Yusur\_rdma\_dev(pf1,single,ndev 1)

1. 建立bonding时：

(1)移除Yusur\_rdma\_dev(pf1,single,ndev 1)

(2)移除yusur\_rdma\_dev(pf0, single,ndev0)

(3)建立yusur\_rdma\_dev(pf0,bonding\_mode, bondidng\_ndev)

bonding管理的软件结构：

struct yusur\_rdma\_tracker {

enum netdev\_lag\_tx\_type tx\_type;

bool has\_inactive;

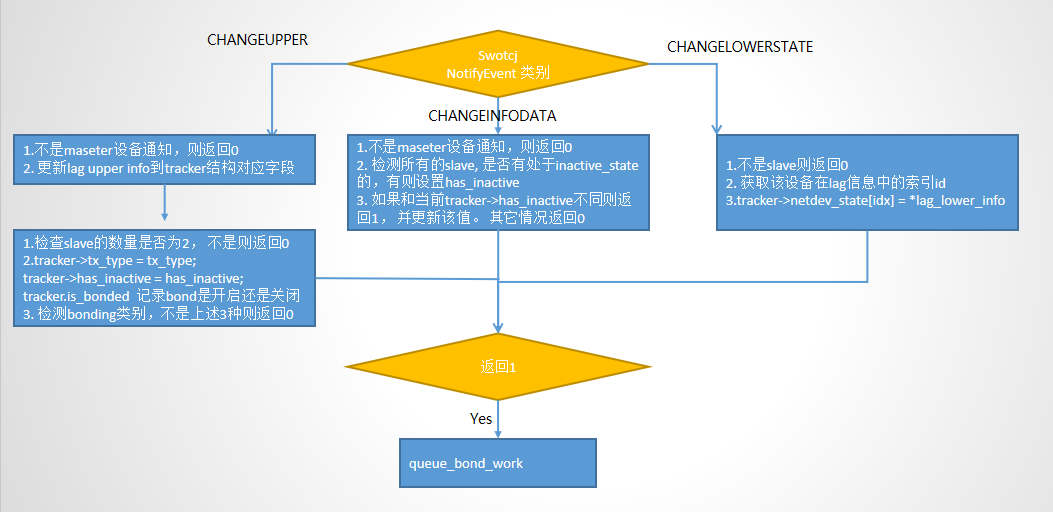
bool is\_bonded;

struct netdev\_lag\_lower\_state\_info info[2];

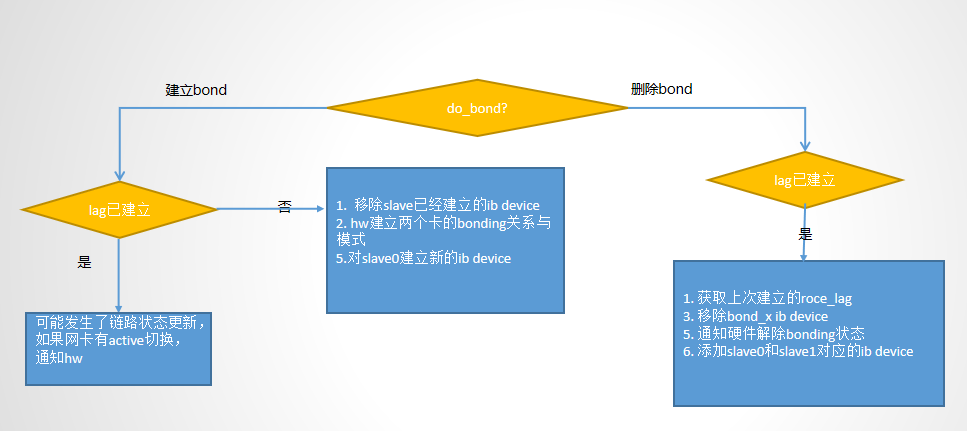
};

bonding信息获取方案：

1. 在pcie驱动上注册register\_netdevice\_notifier 注册
2. 分类事件处理：



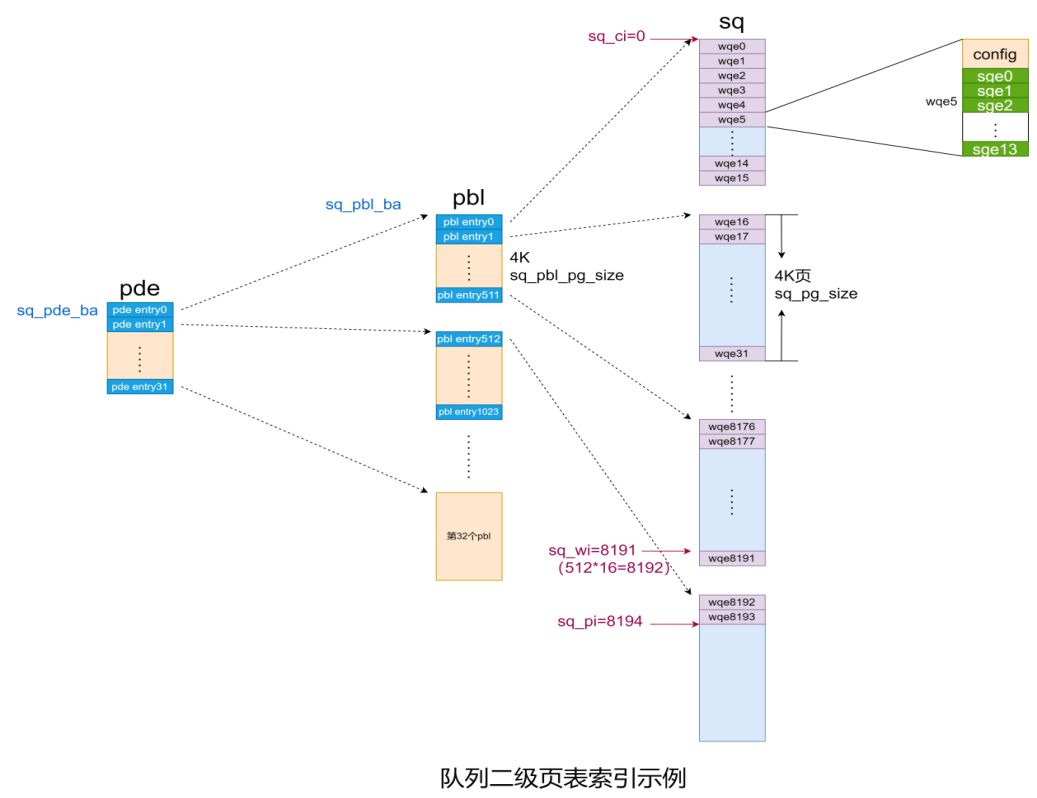
1. Queue\_bond\_work在单独的线程中运行：



## 内存管理

### 4.1 硬件内存管理方案

对于， qpc，mr,wqe等资源，硬件采用了2级页表的结构。 但具体内存如何分配由驱动根据实际情况来决定。 硬件的页表结构如下图所示：



u32 pbl\_pg\_size; //页表框的大小，4k的整数倍

u32 pg\_size; //页表大小， 4k的整数倍

u32 levels; //0级页表，入口地址直接就是数据内容，

//1 级页表， 入口为最终页框的地址

//2级页表，入口为根页框的地址

u64 root\_pa; //入口pa地址

### 4.2 软件内存管理方案

操作系统连续的物理内存不能超过2^MAX\_ORDER \* page\_size（一般os为2MB/4MB)，当内存超过这个范围，就无法分配出连续的内存了。为此引入了fragment结构，来管理物理内存。

struct yib\_buf\_list {

os\_dma\_addr\_t dma\_addr;//每一个分片的物理地址

void \*vaddr;//每一个分片的虚拟地址

};

struct yib\_frag\_buf {

struct yib\_buf\_list \*frags;//分片

int npages;//每个分片的页数

u64 size;//按每个分片相等算出来的总长度

u64 real\_size;//考虑最后一个分片不同后的总长度

};

Real\_size <= size, size/npages是能整除的。

void yib\_frag\_free\_node(struct yib\_sf \*sf, struct yib\_frag\_buf \*buf);//释放frag

void \*yib\_frag\_get\_vaddr(struct yib\_frag\_buf \*buf, int item\_size, int index, u32 start\_off = 0);//已知道元素大小和第几个元素，获取在frag中的虚拟地址

u64 yib\_frag\_get\_paddr(struct yib\_frag\_buf \*buf, int item\_size, int index, u32 start\_off = 0);//已知道元素大小和第几个元素，获取在frag中的物理地址

struct yib\_frag\_buf \* yib\_frag\_buf\_alloc\_node(struct yib\_sf \*sf, int size, bool init\_zero， bool must1);//当must1设置时，只能分一个页

void yib\_frag\_zero\_buf(struct yib\_frag\_buf \*buf);//数据区清0

u32 yib\_frag\_get\_pagesz(struct yib\_frag\_buf \*buf);//

u32 yib\_frag\_get\_lastsz(struct yib\_frag\_buf \*buf);//最后一个分片的大小

Fragment分配策略如下：

1. frag大小从 frag\_len = align\_up(4096, min(max\_order\_page, total));开始尝试分配， frag\_cnt= (total+max\_order\_page-1)/max\_order\_page
2. 分配失败，则max\_order\_page = max\_order\_page/2,重新开始分配。

### 4.3 页表生成策略

(1) struct yib\_page\_tbl \* {

int pbl\_pg\_size; //页表框的大小，4k的2^n次方

int pg\_size; //页表大小， 4k的2^n次方

int levels; //0级页表，入口地址直接就是数据内容，

//1 级页表， 入口为最终页框的地址

//2级页表，入口为根页框的地址

u64 root\_pa; //入口pa地址

struct yib\_frag\_buf \* tbl\_frag;//当tbl\_frag为1时，就不用rootpage了

struct yib\_frag\_buf \* root\_frag;//当level为2时，指向根页表，必须frag必须为1

};

U64 yib\_mem\_get\_pa(struct yib\_page\_tbl \* int index); 取第index个的物理地址8字节。 （MR的pa0不能用这个函数，必须上层指定）。

**（1） Fast MR场景**

由于一个Fast MR的大小（页数）max\_fast\_reg\_page\_list\_len可以提前提交给操作系统，所以实际分配时不会超过这个大小。因此采用策略如下：

Case1: 当pages中只有一个页时， 为0级页表， root\_pa 指向pages[0].paddr, page\_size为最终数据大小。

Case2: 当pages中有多个页时， root\_pa指向pages数组的物理地址:永远采用1级页表。 page\_size=操作系统PAGE\_SIZE, page\_table\_size = PAGE\_SIZE;

int yib\_create\_page\_table\_by\_pages(struct yib\_sf \*sf, struct yib\_page\_tbl \*tbl, u64 pages[], int npages);

1. **内核态分配场景**

Int yib\_create\_page\_table\_by\_frag(struct yib\_sf \*sf, struct yib\_page\_tbl \* tbl, struct yib\_frag\_buf \* dat\_frag);

适用于内核态MR，内核态WRE/CQE/RQE，内核态QPC资源等。

Case1: 数据frag\_dat只有一个分片， 这时用0级页表， root\_pa指向frag\_dat.frag[0]的物理地址。 page\_size=frag\_dat的real\_size。

Case 2: dat\_frag(最后一个可以不等）： page\_size= dat\_frag->size/dat\_frag->npages大小, 计算 page\_table\_item\_cnt = total数据大小/page\_size获取页数, 具体采用1级还是2级由yib\_create\_sub\_page\_table决定。(最有一个页表可以不填满）

1. **用户态场景**

int yib\_create\_page\_table\_by\_frag(struct yib\_sf \*sf, struct yib\_page\_tbl \* tbl, struct os\_sg\_list \*sglist, int npages, u32 page\_size);

(page\_size （4K/2MB/1G）通过内核辅助函数yib\_umem\_find\_best\_pgsz 获取。

适用于内核态MR，内核态WRE/CQE/RQE

Case 0: nsg = 1, 采用0级页表

Others:

step1: page\_table\_item\_cnt = total数据大小/page\_size获取页数, 具体采用1级还是2级由yib\_create\_sub\_page\_table决定。

**Step2**: for each sg; {

For each page in sg

填入tbl\_frag

}

1. 页表的生成策略

yib\_create\_sub\_page\_table（struct yib\_sf \*sf, struct yib\_page\_tbl \* tbl， int hw\_pages);

**Case1**: tbl\_frag只有一个散列，level=1级页表， page\_table\_size就是向上4k取整tbl\_frag.size/tbl\_frag.npages. Root\_pa指向：tbl\_frag[0].paddr

**Case2**: tbl\_frag有多个散列， page\_table\_size=tbl\_frag.size/npages; 这时root\_pa指向root\_tbl（需分配）

对于2级的case要判断 root\_tbl是否能放下，如果放不下就报错。

### 4.3 队列管理模型

struct yib\_queue\_info//sq rc,cq

{

os\_atomic pi;

os\_atomic ci;

u64 io\_count; // io提交数量

u64 err\_count; //完成错误个数, 【rq,sq 为队列满的次数】 cq为wc->err的个数 u32 pi\_toggle; //反转位

u32 ci\_toggle;//反转位

};

struct yib\_queue\_mem

{

union {

void \*umem; //用户态用这个

struct yib\_frag\_buf \*kmem;//内核态用这个

} mem;

int depth;

u16 item\_size;

bool is\_user;//判断是内核态还是用户态

struct yib\_queue\_info \*info;//队列信息

struct yib\_page\_tbl tbl;//页表

};

struct yib\_queue\_mem \* yib\_create\_queue(struct yib\_sf \* sf, int depth, int item\_size, void \* umem);//当umem为NULL时为内核态。

void yib\_destroy\_queue(struct yib\_sf \* sf, struct yib\_queue\_mem);

不同队列的PI,CI维护说明：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 队列 | PI | CI | 空判断 | 满判断 |
| SQ | 软件增加，DB通知硬件 | 硬件增加；软件需要从cqe完成后软件自己维护（问题：不是所有命令都产生cqe;可能导致wqe的延迟回收）  Ci = wqe\_index+1 | 软件无需判断 | Pi,ci和toggle来判断  （Pi+1）%depth = = ci 表示满了 |
| RQ | 软件增加，DB通知硬件 | 硬件增加；软件需要从cqe完成后软件自己维护:  提交时--，完成时++ |  | rq用两套机制,free\_count不为0. |
| NQ,CQ | 硬件维护，软件不感知 | 软件维护++ | Cqe/nqe维护togglebit,当swci.toogle\_bit 和cqe中toogle\_bit相反时，表示cqe可用， swci++（swpi.toogle不维护） | 无需判断 |
| SRQ | 就是最后一个按db的位置  DB通知硬件 | 不维护 |  | 根据posted\_bitmap设置 |

### 4.4 SRQ的特殊处理

SRQ对于不同qp间，顺序是可以乱序完成的，但硬件RQ和SRQ处理提交的逻辑是一样的，不能感知乱序的提交，因此引入了如下3个结构，

1：posted\_bitmap 软件每提交一个元素就设置对应position的bit位。

当完成时就清理该bit.

2: db\_bitmap,当软件能下发db时就设置对应position的bit位，完成是清理该bit.

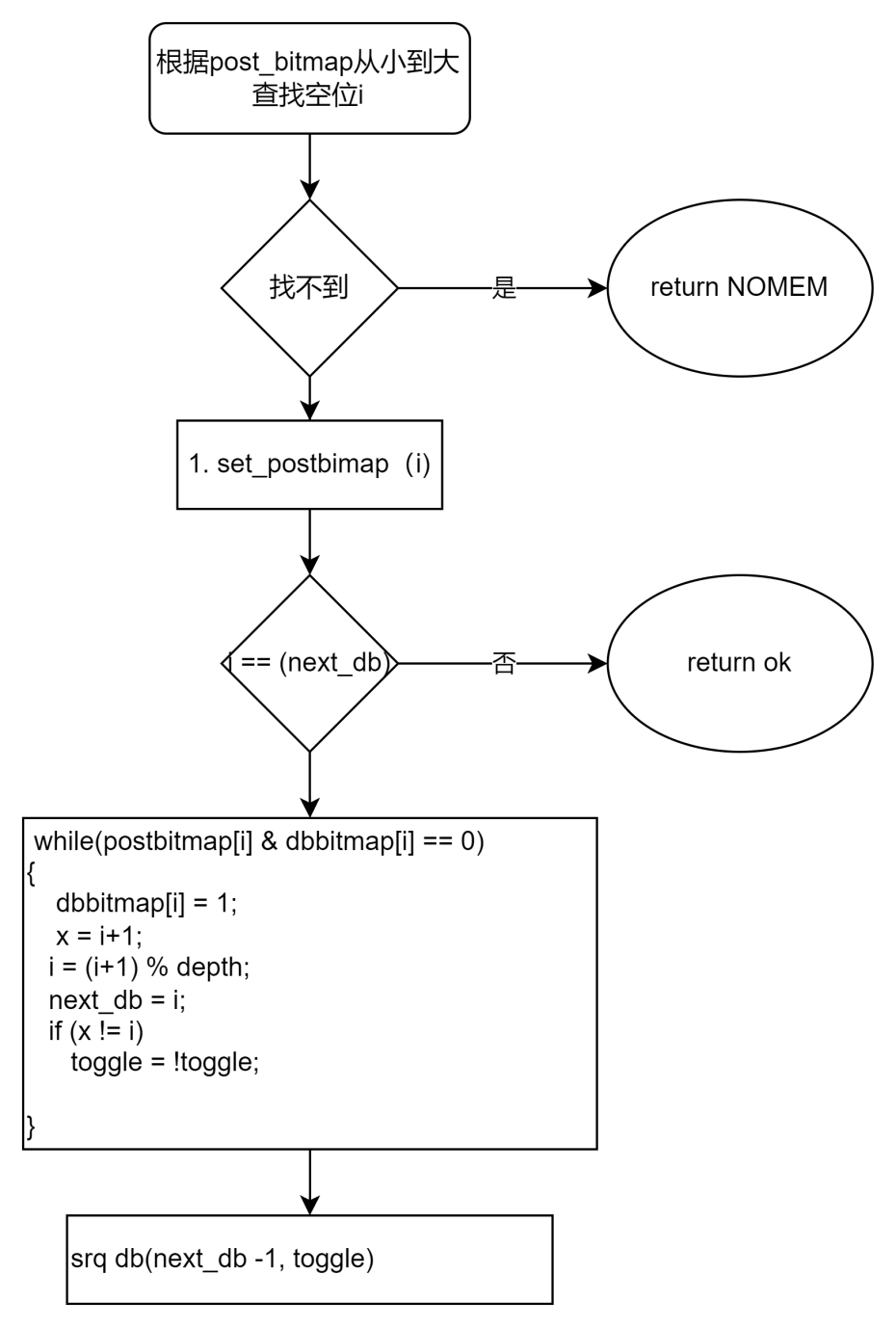
3: Nextdb\_pos: 最后按门铃的下一个元素。

完成流程如下：

Posted\_bimap(i) = 0;

db\_bitmap(i) = 0;

提交流程如下所示：

%3CmxGraphModel%3E%3Croot%3E%3CmxCell%20id%3D%220%22%2F%3E%3CmxCell%20id%3D%221%22%20parent%3D%220%22%2F%3E%3CmxCell%20id%3D%222%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%223%22%20target%3D%226%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%223%22%20value%3D%22%E6%A0%B9%E6%8D%AEpost\_bitmap%E4%BB%8E%E5%B0%8F%E5%88%B0%E5%A4%A7%E6%9F%A5%E6%89%BE%E7%A9%BA%E4%BD%8Di%22%20style%3D%22rounded%3D1%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-607.68%22%20y%3D%222060%22%20width%3D%22145.12%22%20height%3D%2250%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%224%22%20value%3D%22%E6%98%AF%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%226%22%20target%3D%227%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%225%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%226%22%20target%3D%229%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%226%22%20value%3D%22%E6%89%BE%E4%B8%8D%E5%88%B0%22%20style%3D%22rhombus%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-575.12%22%20y%3D%222140%22%20width%3D%2280%22%20height%3D%2280%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%227%22%20value%3D%22return%20NOMEM%22%20style%3D%22ellipse%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-400.12%22%20y%3D%222140%22%20width%3D%22120%22%20height%3D%2280%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%228%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%229%22%20target%3D%2212%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%229%22%20value%3D%221.%20set\_postbimap%EF%BC%88i)%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-595.12%22%20y%3D%222240%22%20width%3D%22120%22%20height%3D%2240%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2210%22%20value%3D%22%E5%90%A6%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%2212%22%20target%3D%2213%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2211%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BentryX%3D0.5%3BentryY%3D0%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%2212%22%20target%3D%2215%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%3E%3CmxPoint%20x%3D%22-540%22%20y%3D%222440%22%20as%3D%22targetPoint%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2212%22%20value%3D%22i%20%3D%3D%20(db\_pos%2B1)%25depth%22%20style%3D%22rhombus%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-575.12%22%20y%3D%222310%22%20width%3D%2280%22%20height%3D%2280%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2213%22%20value%3D%22return%20ok%22%20style%3D%22ellipse%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-400.12%22%20y%3D%222310%22%20width%3D%22120%22%20height%3D%2280%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2214%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%2215%22%20target%3D%2216%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2215%22%20value%3D%22%26amp%3Bnbsp%3Bwhile(postbitmap%5Bi%5D%20%26amp%3Bamp%3B%20dbbitmap%5Bi%5D%20%3D%3D%200)%26amp%3Bnbsp%3B%26lt%3Bbr%26gt%3B%7B%20%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3B%20dbbitmap%5Bi%5D%20%3D%201%3B%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3B%20if%20(db\_pos%20%26amp%3Bgt%3B%20i)%20toggle%3Dtoggle%3F0%3A1%3B%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3B%20db\_pos%20%3D%20i%3B%26amp%3Bnbsp%3B%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3Bi%20%3D%20(i%2B1)%20%25%20depth%3B%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3B%26lt%3Bbr%26gt%3B%7D%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3Balign%3Dleft%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-665%22%20y%3D%222420%22%20width%3D%22260%22%20height%3D%22150%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2216%22%20value%3D%22srq%20db(db\_pos%2C%20toggle)%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3Balign%3Dleft%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-645.12%22%20y%3D%222600%22%20width%3D%22220%22%20height%3D%2240%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3C%2Froot%3E%3C%2FmxGraphModel%3E%3CmxGraphModel%3E%3Croot%3E%3CmxCell%20id%3D%220%22%2F%3E%3CmxCell%20id%3D%221%22%20parent%3D%220%22%2F%3E%3CmxCell%20id%3D%222%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%223%22%20target%3D%226%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%223%22%20value%3D%22%E6%A0%B9%E6%8D%AEpost\_bitmap%E4%BB%8E%E5%B0%8F%E5%88%B0%E5%A4%A7%E6%9F%A5%E6%89%BE%E7%A9%BA%E4%BD%8Di%22%20style%3D%22rounded%3D1%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-607.68%22%20y%3D%222060%22%20width%3D%22145.12%22%20height%3D%2250%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%224%22%20value%3D%22%E6%98%AF%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%226%22%20target%3D%227%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%225%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%226%22%20target%3D%229%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%226%22%20value%3D%22%E6%89%BE%E4%B8%8D%E5%88%B0%22%20style%3D%22rhombus%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-575.12%22%20y%3D%222140%22%20width%3D%2280%22%20height%3D%2280%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%227%22%20value%3D%22return%20NOMEM%22%20style%3D%22ellipse%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-400.12%22%20y%3D%222140%22%20width%3D%22120%22%20height%3D%2280%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%228%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%229%22%20target%3D%2212%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%229%22%20value%3D%221.%20set\_postbimap%EF%BC%88i)%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-595.12%22%20y%3D%222240%22%20width%3D%22120%22%20height%3D%2240%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2210%22%20value%3D%22%E5%90%A6%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%2212%22%20target%3D%2213%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2211%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BentryX%3D0.5%3BentryY%3D0%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%2212%22%20target%3D%2215%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%3E%3CmxPoint%20x%3D%22-540%22%20y%3D%222440%22%20as%3D%22targetPoint%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2212%22%20value%3D%22i%20%3D%3D%20(next\_db)%22%20style%3D%22rhombus%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-575.12%22%20y%3D%222310%22%20width%3D%2280%22%20height%3D%2280%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2213%22%20value%3D%22return%20ok%22%20style%3D%22ellipse%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-400.12%22%20y%3D%222310%22%20width%3D%22120%22%20height%3D%2280%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2214%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%2215%22%20target%3D%2216%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2215%22%20value%3D%22%26amp%3Bnbsp%3Bwhile(postbitmap%5Bi%5D%20%26amp%3Bamp%3B%20dbbitmap%5Bi%5D%20%3D%3D%200)%26amp%3Bnbsp%3B%26lt%3Bbr%26gt%3B%7B%20%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3B%20dbbitmap%5Bi%5D%20%3D%201%3B%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3B%20x%20%3D%20i%2B1%3B%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3Bi%20%3D%20(i%2B1)%20%25%20depth%3B%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3Bnext\_db%20%3D%20i%3B%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3Bif%20(x%20!%3D%20i)%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3B%20toggle%20%3D%20!toggle%3B%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3B%26lt%3Bbr%26gt%3B%7D%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3Balign%3Dleft%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-665%22%20y%3D%222420%22%20width%3D%22260%22%20height%3D%22150%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2216%22%20value%3D%22srq%20db(next\_db%20-1%2C%20toggle)%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3Balign%3Dleft%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-645.12%22%20y%3D%222600%22%20width%3D%22220%22%20height%3D%2240%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3C%2Froot%3E%3C%2FmxGraphModel%3E%3CmxGraphModel%3E%3Croot%3E%3CmxCell%20id%3D%220%22%2F%3E%3CmxCell%20id%3D%221%22%20parent%3D%220%22%2F%3E%3CmxCell%20id%3D%222%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%223%22%20target%3D%226%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%223%22%20value%3D%22%E6%A0%B9%E6%8D%AEpost\_bitmap%E4%BB%8E%E5%B0%8F%E5%88%B0%E5%A4%A7%E6%9F%A5%E6%89%BE%E7%A9%BA%E4%BD%8Di%22%20style%3D%22rounded%3D1%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-607.68%22%20y%3D%222060%22%20width%3D%22145.12%22%20height%3D%2250%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%224%22%20value%3D%22%E6%98%AF%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%226%22%20target%3D%227%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%225%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%226%22%20target%3D%229%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%226%22%20value%3D%22%E6%89%BE%E4%B8%8D%E5%88%B0%22%20style%3D%22rhombus%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-575.12%22%20y%3D%222140%22%20width%3D%2280%22%20height%3D%2280%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%227%22%20value%3D%22return%20NOMEM%22%20style%3D%22ellipse%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-400.12%22%20y%3D%222140%22%20width%3D%22120%22%20height%3D%2280%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%228%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%229%22%20target%3D%2212%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%229%22%20value%3D%221.%20set\_postbimap%EF%BC%88i)%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-595.12%22%20y%3D%222240%22%20width%3D%22120%22%20height%3D%2240%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2210%22%20value%3D%22%E5%90%A6%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%2212%22%20target%3D%2213%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2211%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3BentryX%3D0.5%3BentryY%3D0%3BentryDx%3D0%3BentryDy%3D0%3B%22%20edge%3D%221%22%20source%3D%2212%22%20target%3D%2215%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%3E%3CmxPoint%20x%3D%22-540%22%20y%3D%222440%22%20as%3D%22targetPoint%22%2F%3E%3C%2FmxGeometry%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2212%22%20value%3D%22i%20%3D%3D%20(next\_db)%22%20style%3D%22rhombus%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-575.12%22%20y%3D%222310%22%20width%3D%2280%22%20height%3D%2280%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2213%22%20value%3D%22return%20ok%22%20style%3D%22ellipse%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-400.12%22%20y%3D%222310%22%20width%3D%22120%22%20height%3D%2280%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2214%22%20value%3D%22%22%20style%3D%22edgeStyle%3DorthogonalEdgeStyle%3Brounded%3D0%3BorthogonalLoop%3D1%3BjettySize%3Dauto%3Bhtml%3D1%3B%22%20edge%3D%221%22%20source%3D%2215%22%20target%3D%2216%22%20parent%3D%221%22%3E%3CmxGeometry%20relative%3D%221%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2215%22%20value%3D%22%26amp%3Bnbsp%3Bwhile(postbitmap%5Bi%5D%20%26amp%3Bamp%3B%20dbbitmap%5Bi%5D%20%3D%3D%200)%26amp%3Bnbsp%3B%26lt%3Bbr%26gt%3B%7B%20%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3B%20dbbitmap%5Bi%5D%20%3D%201%3B%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3B%20x%20%3D%20i%2B1%3B%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3Bi%20%3D%20(i%2B1)%20%25%20depth%3B%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3Bnext\_db%20%3D%20i%3B%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3Bif%20(x%20!%3D%20i)%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3B%20toggle%20%3D%20!toggle%3B%26lt%3Bbr%26gt%3B%26amp%3Bnbsp%3B%20%26amp%3Bnbsp%3B%26lt%3Bbr%26gt%3B%7D%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3Balign%3Dleft%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-665%22%20y%3D%222420%22%20width%3D%22260%22%20height%3D%22150%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3CmxCell%20id%3D%2216%22%20value%3D%22srq%20db(next\_db%20-1%2C%20toggle)%22%20style%3D%22rounded%3D0%3BwhiteSpace%3Dwrap%3Bhtml%3D1%3Balign%3Dleft%3B%22%20vertex%3D%221%22%20parent%3D%221%22%3E%3CmxGeometry%20x%3D%22-645.12%22%20y%3D%222600%22%20width%3D%22220%22%20height%3D%2240%22%20as%3D%22geometry%22%2F%3E%3C%2FmxCell%3E%3C%2Froot%3E%3C%2FmxGraphModel%3E

算法执行过程示例如下：

queue posted dbed next\_db

[x,x,x,x] [0,0,0,0] [0,0,0,0] 0

[0,x,x,x] [1,0,0,0] [1,0,0,0] 1

[0,1,x,x] [1,1,0,0] [1,1,0,0] 2

[0,1,2,x] [1,1,1,0] [1,1,1,0] 3

[0 1 2 3] [1,1,1,1] [1,1,1,1] 0

完成【0, x, 2, x] [1,0,1,0] [1,0,1,0] 0

[0,1a, 2,x] [1,1,1,0] [1,0, 1,0] 0

[0, 1a, 2, 3a] [1,1,1,1] [1,0,1,0] 0

完成[x, 1a, 2, 3a] [0,1,1,1] [0,0,1,0] 0

【0a, 1a, 2, 3a] [1,1,1,1] [1,1,1,0] 2

该流程对应的函数为：bool **yib\_srq\_db\_helper**(struct yib\_srq \*ysrq, int pos)

### 4.5 sq/rq sw\_cmd 管理

struct yib\_sw\_cmd

{

u8 bsignal:1; //正常完成是否会产生cqe，rq固定为1

u8 at\_err:1; //是否处于err state提交时 或 err完成时设置

U8 posted:1;

u8 sw\_posted;//允许一个位置反复提交多次

ib\_wc\_opcode opcode; //对应的wc的opcode

u64 wrid; //存储wr对应的wrid

};

sq/rq 每提交一个命令维护该数组, 完成时更新。该结构能判断硬件是否发生了错误的完成顺序。

yib\_sq\_swcmd\_done： SQ完成流程

1. 完成当前的位置，没posted, 错误打印
2. 前移一个位置

While(true) {

A.如果这个位置已完成(posted ==0)，则结束循环

B.如果，设置了CE 或err, 则报错(前一个没完成，不应该完成当前的位置）。

C 清0， sq\_cmd[i]

D.前移一个位置 i = queue\_index\_dec(i, ysq->queue->depth);

E. 完成数量++

}

Yib\_rq\_swcmd\_done: 只检测当前位置（因为每个rq必须按顺序完成）

yib\_srq\_swcmd\_done： 只检测当前位置（因为每个srq必须完成）,清当前位置的post\_bitmap,和db\_bitmap

int **yib\_sw\_cqe\_generate**(struct yusur\_ib\_dev \*yib, struct yib\_cq \*ycq, struct yib\_sw\_cqe \*input\_sw\_cqe) 该api用于提交软件注入的cqe

struct yib\_sw\_cqe {

u8 type;//sq:0 rq:1 srq:2

u8 status;

U8 bsw; //进case4 设置位1 （case参考 6.4 CQ)

u32 start\_pos;

u32 end\_pos; // [start\_pos, end\_pos)

u32 depth;

u64 handler; //sq:填写yqp地址，rq、srq:填写yrq地址

struct llist\_node node;//该值用户不需要填写

};

Poll\_cq框架会处理软件注入的cq.

## RDMA资源管理

## 5.1通用资源管理层

**struct yib\_pool** {

os\_rwlock pool\_lock; /\* protects pool add/del/search \*/

size\_t elem\_size;

os\_ref ref\_cnt;

void (\*cleanup)(struct yib\_pool\_entry \*obj);

enum yib\_pool\_state state;

u32 flags;//控制资源池的属性

enum yib\_elem\_type type;

unsigned int max\_elem;

unsigned int max\_real\_elem; //max\_real\_elem must <= max\_elem， 用于控制实际可分配的数量

os\_atomic num\_elem;

struct yib\_pool\_entry \*\*array;//是否支持数组检索（可不用）

unsigned long \*table; //管理资源分配的bitmap

size\_t table\_size;

u32 max\_index;

u32 min\_index;

u32 last; //上次分配到的位置

}; 该资源池用于管理软件verbs资源

**enum yib\_pool\_flags** {

YIB\_POOL\_ATOMIC = BIT(0),//If set use GFP\_ATOMIC for kzalloc

YIB\_POOL\_INDEX = BIT(1), //支持bitmap管理

YIB\_POOL\_RBTREE\_IDX = BIT(2), //后续扩展到rb 树检索

YIB\_POOL\_KEY = BIT(3), //rb数时使用

YIB\_POOL\_NO\_ALLOC = BIT(4),//由于verbs的接口有的资源是ib层分配，有的是驱动层分配，当由ib层分配时，设置该标志。

YIB\_POOL\_ARRAY\_IDX = BIT(5),//支持数组索引

};

int yib\_pool\_init(struct yib\_pool \*pool, enum yib\_elem\_type type, unsigned int max\_elem, unsigned int max\_real\_cnt);

void yib\_pool\_cleanup(struct yib\_pool \*pool);

当资源YIB\_POOL\_NO\_ALLOC不设置时：

void \***yib\_pool\_alloc**(struct yib\_pool \*pool);

void \***yib\_pool\_alloc\_specify\_index**(struct yib\_pool \*pool, int index); //用于gsi only

当资源YIB\_POOL\_NO\_ALLOC设置时：

int **yib\_add\_to\_pool**(struct yib\_pool \*pool, struct yib\_pool\_entry \*elem);

int **yib\_add\_to\_pool\_special**(struct yib\_pool \*pool, struct yib\_pool\_entry \*elem, bool specify, int sindex);//用于gsi only

#define **yib\_elem\_add\_ref**(elem) os\_ref\_get(&(elem)->ref\_cnt) 初始化后自用引用计数为1，后面用这个增加

删除时减少引用计数

#define **yib\_elem\_drop\_ref**(elem) os\_ref\_put(&(elem)->ref\_cnt, yib\_pool\_elem\_release)

## 5.2 软件资源管理

#### 5.2.1 MAC与GID管理

在yib\_host中有

struct yib\_mac\_tbl\_item

{

U8 mac[6];

U16 ref\_cnt;

U8 port\_num;

U8 is\_init; //init的不用引用计数

};

Struct yib\_mac\_tbl

{

Int max\_items;

struct yib\_mac\_tbl\_item[YIB\_MAX\_MAC\_TBL\_LEN];

};

当初始化时复制host.mac\_tbl.

当gid\_add时， 查找mac表，找到： 如果init的，就立即返回。如果其它就增加引用计数。

未找到： 增加，后续要更新mac表。

当gid\_del时， 查找mac表，找到：减少引用计数，为0则删除。后续要更新mac表

未找到：告警，结束。

### 5.2.2 ucontext & pd管理

Ucontext:yib->host.hw\_ops.get\_queue\_user\_info（YIB\_TYPE\_PD 底层驱动负责返回ucontext的全局信息。

Pd 资源只维护pdn无其它信息。

### 5.2.3 ah管理

软件维护完整的ah信息。

struct yib\_global\_route {

union ib\_gid dgid;

\_\_u32 flow\_label;

\_\_u8 sgid\_index;

\_\_u8 hop\_limit;

\_\_u8 traffic\_class;

};

struct yib\_av {

\_\_u8 port\_num;

\_\_u8 network\_type;

\_\_u8 dmac[6];

\_\_u8 smac\_index;

\_\_u8 smac[6];

\_\_u16 vlan\_id;//vlan\_id =0xFFFF表示不是vlan

\_\_u8 vlan\_pcp;//vlan的优先级

struct yib\_global\_route grh;

union ib\_gid sgid;

union {

struct sockaddr\_in \_sockaddr\_in;

struct sockaddr\_in6 \_sockaddr\_in6;

} sgid\_addr, dgid\_addr;

};

struct yib\_ah {

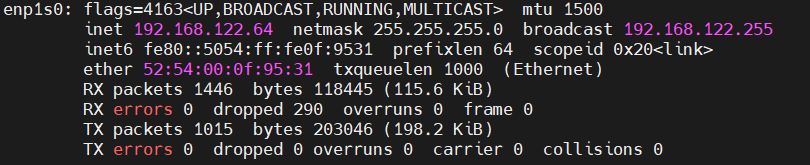
struct ibv\_ah ibv\_ah;

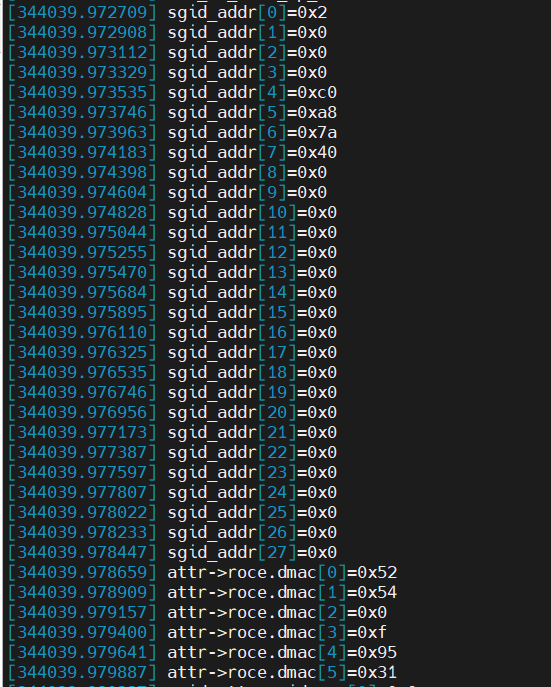
struct yib\_av av;

int ah\_num; //ah的索引

};

Mac/ip字节序说明：





### 5.2.4 CQ管理

struct yib\_cq {

#if IB\_LAYER\_ALLOC\_CQ

os\_ib\_cq ib\_cq;

struct yib\_pool\_entry entry;

#else

struct yib\_pool\_entry entry;

os\_ib\_cq ib\_cq;

#endif

os\_spinlock cq\_lock; //cq lock

os\_list\_head evt\_comp\_node; //完成异步通知

os\_list\_head evt\_err\_node; //错误异步通知

bool is\_user;

bool nvme\_off;

u32 comp\_vector;//完成向量

u32 cqid; //cqn

struct llist\_head sw\_done\_list; //软件提交的完成队列

struct yib\_sw\_cqe \*cur\_sw\_cqe;//当前处理到的完成队列

struct yib\_queue\_mem \*queue; //队列结构

bool is\_dying;

u64 int\_dup; //重复中断统计

u64 int\_occur; //中断次数统计

u64 int\_handle; //中断处理次数统计（可合并处理）

};

1. yib\_cq\_alloc\_new建立cq
2. yib\_cq\_info\_init :sf\_ops->cq\_info\_init 调用硬件建立cq
3. 用户态：hw\_ops.get\_queue\_user\_info(&yib->host, YIB\_TYPE\_CQ

### 5.2.4 SQ/RQ管理

struct yib\_sq

{

struct yib\_queue\_mem \*queue;

os\_spinlock sq\_lock;

struct yib\_sw\_cmd \*sw\_cmds;

};

struct yib\_rq

{

struct yib\_pool\_entry entry;

struct yib\_queue\_mem \*queue;

os\_spinlock rq\_lock;

bool bsrq:1; //是srq 还是rq

bool bxrc:1;

bool is\_user:1; //用户态还是内核态

void \*yib;

u32 xrcd\_val;

void\* parent;//当是rq时，指向yib\_qp; 当是srq时指向yib\_srq

u32 max\_recv\_sge;

struct yib\_sw\_cmd \*sw\_cmds;

void \*privdata; //由具体硬件实现层来定义

};

struct yib\_srq {

os\_ib\_srq ib\_srq;

struct yib\_rq \*yrq;

struct yib\_cq \*yrq\_cq; //use for xrc mode only

os\_srq\_attr attr;

unsigned long \*post\_bitmap;

unsigned long \*db\_bitmap;

bool toggle;

u32 next\_db;

volatile u32 limit;

os\_list\_head evt\_limit\_node;

os\_list\_head evt\_err\_node;

os\_list\_head evt\_lastwqe\_node;

};

### 5.2.5 MR & MW管理

struct yib\_mw {

os\_ib\_mw ib\_mw;

struct yib\_mr \*pmr; //mw的父mr

};////其他信息bind时从上层传下来

struct yib\_fmr {

void \*descs; //对齐后的虚拟地址

void \*descs\_alloc;//真实的分配虚拟地址

dma\_addr\_t desc\_map;//对齐后的物理地址

int max\_descs;

int ndescs; //实际的页数

int desc\_size;

u64 pa;//起始物理地址

};

struct yib\_mr {

struct yib\_pool\_entry entry;

os\_ib\_mr ib\_mr;////mw 不能用该字段

struct yib\_mr\_type type;

enum yib\_mr\_state state;

u8 access;

os\_ib\_umem \*umem;

u32 pd\_num;

union {

void \*page\_tbl; //normal mr时有效， dma mr不用这个

struct yib\_mw \*pmw; //mw时有效

struct yib\_fmr \*fmr; //fast mr时有效

} priv;

};

1. Normal Mr

A.建立mr yib\_mr\_alloc\_new(yib, ypd, true, false, false);

B.获取用户内核态地址信息（支持GDR) umem = yib\_umem\_get\_mem\_peer

C.sf\_ops->mr\_mtt\_init 更新mtt

C.sf\_ops->mr\_mpt\_update更新mpt

(2) Fast MR (alloc mr) 只到free状态 （fmr bind在io路径）

Alloc\_mr:

A.yib\_mr\_alloc\_new(yib, ypd, false, false, true);

B. yib\_fmr\_init(yib, mr); 准备一个固定的内存区用于存储fmr要用的地址（物理/虚拟）

map\_mr\_sg：

ib\_sg\_to\_pages(ibmr, sglist, sg\_nents, sg\_offset, yib\_set\_page); //填充mtt

1. DMA

Case1: local\_dma\_lkey

在ib register dev中建立一个全局的global key(释放时要释放该key)

Case2: get\_dma\_key

yib\_mr\_alloc\_new(yib, ypd, false, true, false);

### 5.2.6 QP管理

struct yib\_qp {

#if IB\_LAYER\_ALLOC\_QP

os\_ib\_qp ib\_qp;

struct yib\_pool\_entry entry;

#else

struct yib\_pool\_entry entry;

os\_ib\_qp ib\_qp;

#endif

enum ib\_qp\_type qp\_type;

struct yib\_sq ysq;

/\*

qp有四种rq使用方式（写寄存器的人要了解）:

a.对xrc\_init使用xrcd\_val,没有rq/srq

b.xrc\_tgt没有rq,使用srq xrcd\_val

c.rq使用srq

d.rq使用rq

\*/

union {

struct yib\_rq \*yrq;

struct yib\_srq \*ysrq; //有srq时使用

u32 xrcd\_val; //XRC\_INIT类别使用

} type;

struct yib\_pd \*ypd;

bool is\_user;

bool nvme\_off;

bool valid;

bool use\_srq;

os\_atomic resetting;

os\_qp\_attr attr;

os\_qp\_cap cap;

struct yib\_av pri\_av;

struct yib\_av alt\_av;

int mtu;

enum ib\_sig\_type sq\_sig\_type;

u32 internal\_state;

struct yib\_cq \*ysq\_cq;

struct yib\_cq \*yrq\_cq;

u32 sq\_psn;

os\_list\_head evt\_fatal\_node;

};

Qp：建立：

A.yib\_create\_qp\_chk 检查创建属性

B.yib\_qp\_alloc 分配软件资源

C.yib\_qp\_attach 分配队列

D.yib\_qp\_attach\_cq

D1:yib->host.sf.sf\_ops->rq\_info\_init 非srq场景,初始化rq

D2: sf\_ops->qp\_info\_init 初始化qp

QP修改：yib\_qp\_from\_attr-->sf\_ops->qp\_info\_update

QP:卸载： 减少rq引用计数（触发sf\_ops->rq\_info\_init ）

sf\_ops->qp\_info\_init

## IO路径管理

### 6.1 SQ

A.检查qp状态，< RTS,直接报错 -EINVAL

加锁

B. = IB\_QPS\_ERR， 进入软件提交模式

C. while(wr)

(1) 根据qp\_type与opcode做检测， 当发送的opcode不满足qp类别，则--EINVAL

(2) 根据sg 计算数据长度

(3) 直接错误检测（inline, sge数量检测）

(4) yib->host.sf.queue\_ops->check\_sq\_full(&yib->host, ysq) 空满判断

(5) yib->host.sf.queue\_ops->fill\_wqe 填充wqe

(6) 软件sw\_cmd填充，软件pi++

(3) wr = wr->next;

D. yib->host.sf.queue\_ops->sq\_pi\_db\_update(&yib->host, &yqp->ysq, io\_cnt); 批量提交

解锁

### 6.2 RQ

A.检查qp状态，< INIT,直接报错 -EINVAL

加锁

B. = IB\_QPS\_ERR， 进入软件提交模式

C. while(wr)

(1) 根据sg 计算数据长度

(2) 直接错误检测（sge数量检测）

(4) yib->host.sf.queue\_ops->check\_rq\_full(&yib->host, ysq) 空满判断

(5) yib->host.sf.queue\_ops->fill\_rqe 填充rqe

(6) 软件sw\_cmd填充，软件pi++

(3) wr = wr->next;

1. yib->host.sf.queue\_ops->rq\_pi\_db\_update(&yib->host, yrq, io\_cnt)；批量提交

解锁

if (yib->host.sf.queue\_ops->rq\_post\_db) {

yib->host.sf.queue\_ops->rq\_post\_db(&yib->host, yrq, io\_cnt); //适用于rdma-np doe模式

}

### 6.3 SRQ

A.检查srq状态，err状态 -EINVAL

B. while(wr)

(1) 根据sg 计算数据长度

(2) 直接错误检测（sge数量检测）

(3) yib->host.sf.queue\_ops->check\_srq\_full(&yib->host, ysq) 空满判断

(4) yib->host.sf.queue\_ops->fill\_rqe 填充rqe

(5) 软件sw\_cmd填充，软件pi++， post\_bitmap设置

(6)yib->host.sf.queue\_ops->srq\_pi\_db\_update(&yib->host, ysrq, pos);//不支持批量提交

(6) wr = wr->next;

### 6.4 CQ

**6.4.1 CQ POLL**

加锁

While（num-- >0)

1. 检查当前是否存在软件完成的cqe,不为空则

yib\_cur\_sw\_cqe\_process，处理成功则 continue

1. 硬件cqe判断： yib->host.hw\_ops.queue\_ops->check\_cq\_empty(&yib->host, ycq, cqe))
2. 硬件cqe ready没有 则从软件cqe中获取yib\_sw\_poll\_cq； continue
3. 有硬件cqeyib->host.hw\_ops.queue\_ops->fill\_cqe
4. Ci++, 统计计数更新

循环外：

更新硬件yib->host.hw\_ops.queue\_ops->cq\_ci\_db\_update(&yib->host, ycq, poll\_cnt);

Sw cq poll:

static bool **yib\_sw\_poll\_cq**(struct yusur\_ib\_dev \*yib, struct yib\_cq \*ycq, os\_ib\_wc \*wc\_out);

1. ycq->cur\_sw\_cqe 不为空时，先调用yib\_cur\_sw\_cqe\_process从ycq->cur\_sw\_cqe 获取cqe,
2. 当ycq->cur\_sw\_cqe为空时，从ycq->sw\_done\_list获取，并设定ycq->cur\_sw\_cqe.

加入sw cq后，导致逻辑较复杂，sw cq提交总体场景分为四种：

**6.4.2 使用软件cqe的场景**

**场景1**： err state后， sq/rq post走 sw\_cqe提交： （这时io不提交到硬件队列中）(维护指针）

1. 在 io提交函数中，判断是否为err state. 如果是则主动yib\_sw\_cqe\_generate(. ------- sw\_cmd.at\_err = 1,提交了
2. 当poll\_cq 对应的fill\_cqe中，处理时需要判断如果是 这种case,则不要调用sq/rq （yib\_sq\_swcmd\_done yib\_rq\_swcmd\_done ， 维护指针)

（子场景1： 软件主动进入err state

子场景2： 硬件进入err state后，软件感知后的行为）

**场景2**： err state时， 固件发起aeq聚合cqe完成 （这时io已经在硬件队列中了）

1. 在aeq中，
2. 先获取sq/rq lock

(2)根据sq/rq->pi sq/rq->ci 来generate cqe (同时判断sw\_cmd.at\_err位 == 0） yib\_sw\_cqe\_generate,

(3) 修改软件state to err state

(4) 释放sq/rq lock

1. 当poll\_cq， 维护指针

**场景3**： 提交一个io到sq/rq后一直没完成，需要通过软件注入让 io完成。（已提交到硬件队列中）

1. 通过debug命令注入yib\_sw\_cqe\_generate
2. Poll\_cq，维护指针

**场景4**： 提交一个io(但不提交到硬件队列中），完成时需要保序 （这时io不提交到硬件队列中）（不维护指针）（只有sq有这个场景）

1. 软件通过fill\_wqe提交特殊的命令（如纯软件实现fmr). 由于后面一个命令提交到硬件队列时，必须在当前位置。 所以这里一定不能pi++.

增加一个Sw\_cmd.sw\_posted++； sw\_cqe.bsw = 1

1. Poll\_cq完成时不维护指针 (bsw == 1则 sw\_cmd.sw\_posted --)

relase版本没有3,4两种。

**6.4.3 cq相关注册函数指针实现指南**

1. check\_cq\_empty
2. 判断硬件cqe 是否存在，不存在则返回true
3. 硬件cqe存在， 检查cqe对应的位置是否正确， 不正确返回true
4. 检查对应位置是否存在sw\_cmd.sw\_posted是否大于0（只有sq做这个判断）， 大于0，则返回true
5. 返回false
6. Fill\_cqe (主干流程）

走到这里，一定是硬件cqe, 直接填充，并调用yib\_sq/rq/srq\_swcmd\_done(sw\_done = 0)

1. sw\_fill\_cqe
2. 检查sw\_cqe位置是否正确，不正确返回-1
3. 根据类别判断：处理完成返回0.

Case4: Sw\_cqe.bsw如果设置了， 则sw\_cmd.sw\_posted --， 完成时不能递增sq/rq ci

Case3: 完成时维护指针

Case2:完成时维护指针

Case1: 完成时维护指针

## 中断管理

struct yib\_eq \*yib\_eq\_alloc(struct yib\_sf \*sf, host\_verbs\_t \*verbs, int depth, bool use\_thread, int int\_vector)

1. yib\_pool\_alloc(&verbs->eq\_pool);分配一个eq

B. 根据全局配置，决定是否使用中断模式

C. eq\_item\_size = sf->eq\_ops->get\_eq\_isize(sf, yeq);获取eq大小

D. 建立eq队列的物理地址yeq->queue = yib\_create\_queue(sf, depth, eq\_item\_size, NULL);

E.EQ队列的硬件初始化：sf->sf\_ops->eq\_info\_init(sf, yeq, false);

F.启动线程或中断模式

（有两种类别的eq, 其中aeq用于异常事件）。

Void yib\_eq\_free(struct yib\_sf \*sf, int depth，int index);

struct yib\_eq

{

struct yib\_pool\_entry entry;

struct yib\_queue\_mem \*queue;

struct yib\_sf \*ysf;

int int\_vector;

bool use\_thread;

bool baeq;

os\_atomic pi;

os\_atomic ci;

int assign\_irq;//是否已分配好irq

struct os\_thread\_t \*thread;

u64 int\_cnt; //产生中断的总次数

u64 err\_evt\_cnt;//evt code 无效的总次数

u64 evt\_valid\_cnt[YIB\_QP\_EVENT\_END]; //evt->opcode 内容有效次数

u64 evt\_invalid\_cnt[YIB\_QP\_EVENT\_END];//evt->opcode 内容无效次数

u64 evt\_dup\_cnt[YIB\_QP\_EVENT\_END]; ////evt->opcode 软件可合并总次数

};

struct yib\_hw\_events {

void (\*func)(void \*arg);

void \*arg;

os\_task tasklet;

char name[16];

struct list\_head list;

struct list\_head process\_list;

struct tasklet\_struct task;

spinlock\_t lock; /\* lock completion tasklet list \*/

bool destroyed;

int occur;

};

eq处理流程：

sf.xxx\_evts.occur = 0;

while(sf->eq\_ops->check\_eq\_empty(sf, yeq) == false) {

buf = yib\_queue\_get\_ci\_vaddr(yeq->queue);

sf->eq\_ops->eq\_handler(sf, yeq, buf);

yeq->ci++;

}

sf->eq\_ops->eq\_ci\_db\_update(sf, yeq, io\_cnt);

根据sf.xxx\_evts.occur !=0 就执行对应方法的sf->xxx\_evts 的yib\_hw\_events\_run

异步事件提交api:

int yib\_hw\_events\_init(struct yib\_hw\_events \* evts, void (\*func)(void \*), char \*name);//定义异步处理的tasklet

bool yib\_hw\_event\_add(struct yib\_hw\_events \* evts, os\_list\_head \*node);提交异步事件到列表中

void yib\_hw\_events\_run(struct yib\_hw\_events \* evts)； 启动异步事件处理的tasklet

void yib\_hw\_events\_cleanup(struct yib\_hw\_events \* evts);

上述api用户处理，中断中耗时长的处理流程，如cq\_handler的调用， event\_handler的调用。

## 用户态与内核态接口

#### 8.1 基本接口

Rdma-core与内核提供的标准接口：

ibv\_cmd\_xxx 方法，底层通过调用execute\_ioctl\_fallback 来实现。

部分资源数据需要定义自己的扩展字段，扩展信息如下：

struct yib\_ib\_alloc\_pd\_resp {

\_\_u32 pdn; //pd建立完成后内核返回pdn值

};

struct yib\_create\_ah\_resp {

\_\_u32 ah\_num;

\_\_u16 vlan\_id;

\_\_u8 vlan\_pcp;

\_\_u8 smac\_idx;

\_\_u8 smac[8];

};

struct yib\_ib\_create\_cq {

\_\_aligned\_u64 cq\_va; //建立cq时用户态指定cq的va地址

\_\_aligned\_u64 cq\_handle;

\_\_u32 nvme\_off; //该cq是否使用nvme\_off

};

struct yib\_ib\_create\_srq {

\_\_aligned\_u64 srq\_va; //建立srq时用户态指定srq的va地址

\_\_aligned\_u64 srq\_handle;

\_\_u32 nvme\_off; //该srq是否使用nvme\_off

};

struct yib\_ib\_create\_qp {

\_\_aligned\_u64 sq\_va; //sq基地址

\_\_aligned\_u64 qp\_handle;

\_\_aligned\_u64 rq\_handle;

\_\_aligned\_u64 rq\_va; //rq基地址

\_\_u32 nvme\_off;

\_\_u32 reserved;

};

struct yib\_ib\_modify\_qp {

\_\_u32 max\_burst\_sz; //突发长度

\_\_u16 typical\_pkt\_sz;

\_\_u16 reserved;

};

struct yib\_ib\_create\_cq\_resp {

\_\_u32 cqn; //内核返回cqn

\_\_u32 max\_cqe; //内核返回实际cqe队列深度

\_\_u32 reserved;

};

struct yib\_ib\_create\_qp\_resp {

\_\_aligned\_u64 capture\_pa; //预留，暂不使用

\_\_aligned\_u64 nvme\_rq\_rqe\_pa;

\_\_u32 qpn;

\_\_u32 rqn;

\_\_u32 max\_send\_wr; //sq队列深度

\_\_u32 max\_recv\_wr; //rq队列深度

\_\_u32 max\_send\_sge;

\_\_u32 max\_recv\_sge;

\_\_u32 max\_inline\_data;

\_\_u32 send\_isize;

\_\_u32 recv\_isize;

};

Struct yib\_ib\_create\_mr\_req {

\_\_aligned\_u64 mr\_handle;

};

struct yib\_ib\_create\_srq\_resp {

\_\_u32 rqn;

\_\_u32 max\_wr;//srq队列深度

\_\_u32 max\_sge;

\_\_u32 isize;

};

struct yib\_ib\_alloc\_uctx\_resp {

\_\_u32 bar\_offset; //寄存器操作物理地址

\_\_u32 bar\_map\_len; //寄存器区域长度

\_\_u32 chip\_type;

\_\_u32 chip\_subtype;

\_\_u32 reserved;

};

#### 8.2 mmap接口

#### void\* mmap(void\* start,[size\_t](https://baike.baidu.com/item/size_t/8101179?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/mmap/_blank) length,int prot,int flags,int fd,off\_t offset);

Ib device的mmap 第三个参数进行了扩展，

Bit[31:28] 为映射方式：

typedef enum

{

YIB\_MMAP\_TYPE\_REG = 0,

YIB\_MMAP\_TYPE\_CQ,

YIB\_MMAP\_TYPE\_SQ,

YIB\_MMAP\_TYPE\_RQ,

YIB\_MMAP\_TYPE\_PRIO = 4, （0：dscp: 1 pcp)

YIB\_MMAP\_TYPE\_RSV2,

YIB\_MMAP\_TYPE\_RSV3 = 6,

YIB\_MMAP\_TYPE\_RSV4,

YIB\_MMAP\_TYPE\_HW1 = 8,

YIB\_MMAP\_TYPE\_HW2,

YIB\_MMAP\_TYPE\_HW3,

YIB\_MMAP\_TYPE\_HW4,

YIB\_MMAP\_TYPE\_HW5,

YIB\_MMAP\_TYPE\_HW6,

YIB\_MMAP\_TYPE\_HW7,

YIB\_MMAP\_TYPE\_HW8,

}yib\_mmap\_t;

YIB\_MMAP\_TYPE\_HW1(8)--- HW8(15) 是不同硬件的自定义映射方式。

硬件实现者需要实现：yib->host.hw\_ops.host\_def\_mmap

YIB\_MMAP\_TYPE\_REG: 映射寄存器, offset bit[27:0] 为 (相对与bar的起始地址）

YIB\_MMAP\_TYPE\_CQ ---- RQ: 映射CQ(SQ,RQ)管理信息， 这里填写队列软件id.

映射长度计算： sizeof(struct yib\_queue\_inf)\* 总队列个数。（用于调试）

struct yib\_queue\_info//sq rc,cq

{

os\_atomic pi;

os\_atomic ci;

u64 io\_count; // 完成或提交次数

u64 err\_count; //完成错误个数, 【rq,sq 为队列满的次数】 cq为wc->err的个数

u32 direct\_cnt; //[sq,rq]为直接返回失败的次数， cq为软件完成次数

u16 pi\_toggle; //翻转位

u16 ci\_toggle;

}; (调试模式下，该结构从内核态mmap, 发布模式用户态直接malloc出来）。

#### 8.3 扩展接口

struct yib\_ext\_in\_cmd\_hdr {

U32 cmd; //1:debug 2:nvmne 3:todo

U32 input\_data[7];

};

struct yib\_ext\_out\_cmd\_hdr {

u32 status;

u32 output\_data[7]; //根据自定命令的数据

};

该功能依赖 if (IS\_ENABLED(CONFIG\_INFINIBAND\_USER\_ACCESS)) 操作系统的版本。

## 统计与Debug调试接口

### 9.1 标准统计

| Counter | ****Description**** |
| --- | --- |
| duplicate\_request | Number of received packets. A duplicate request is a request that had been previously executed. |
| implied\_nak\_seq\_err | Number of time the requested decided an ACK. with a PSN larger than the expected PSN for an RDMA read or response. |
| local\_ack\_timeout\_err | The number of times QP's ack timer expired for RC QPs at the sender side.  The QP retry limit was not exceed, therefore it is still recoverable error. |
| np\_cnp\_sent | The number of CNP packets sent by the Notification Point when it noticed congestion experienced in the RoCEv2 IP header (ECN bits). |
| np\_ecn\_marked\_roce\_packets | The number of RoCEv2 packets received by the notification point which were marked for experiencing the congestion (ECN bits where '11' on the ingress RoCE traffic) . |
| out\_of\_buffer | The number of drops occurred due to lack of WQE for the associated QPs. |
| out\_of\_sequence | The number of out of sequence packets received. |
| packet\_seq\_err | The number of received NAK sequence error packets. The QP retry limit was not exceeded. |
| req\_cqe\_error | The number of times requester detected CQEs completed with errors. |
| req\_cqe\_flush\_error | The number of times requester detected CQEs completed with flushed errors. |
| req\_remote\_access\_errors | The number of times requester detected remote access errors. |
| req\_remote\_invalid\_request | The number of times requester detected remote invalid request errors. |
| resp\_cqe\_error | The number of times responder detected CQEs completed with errors. |
| resp\_cqe\_flush\_error | The number of times responder detected CQEs completed with flushed errors. |
| resp\_local\_length\_error | The number of times responder detected local length errors. |
| resp\_remote\_access\_errors | The number of times responder detected remote access errors. |
| rnr\_nak\_retry\_err | The number of received RNR NAK packets. The QP retry limit was not exceeded. |
| rp\_cnp\_handled | The number of CNP packets handled by the Reaction Point HCA to throttle the transmission rate. |
| rp\_cnp\_ignored | The number of CNP packets received and ignored by the Reaction Point HCA. This counter should not raise if RoCE Congestion Control was enabled in the network. If this counter raise, verify that ECN was enabled on the adapter.. |
| rx\_atomic\_requests | The number of received ATOMIC request for the associated QPs. |
| rx\_read\_requests | The number of received READ requests for the associated QPs. |
| rx\_write\_requests | The number of received WRITE requests for the associated QPs. |
| rx\_icrc\_encapsulated | The number of RoCE packets with ICRC errors. |
| roce\_slow\_restart | Counts the number of times RoCE slow restart was used |
| roce\_slow\_restart\_cnps | Counts the number of times RoCE slow restart generated CNP packets |
| roce\_slow\_restart\_trans | Counts the number of times RoCE slow restart changed state to slow restart |
| roce\_adp\_retrans\_to | Counts the number of adaptive retransmissions for RoCE traffic |
| roce\_slow\_restart | Counts the number of times RoCE traffic reached timeout due to adaptive retransmission |

### 9.2 软件debug

（1） log调试等级与模块配置

（2） 资源信息跟踪等级

Normal: 只显示软件基本信息

Info: 显示软件完整信息

Dbg: 软件完整信息+同步获取硬件资源信息

资源信息netlink实现：

int (\*fill\_res\_mr\_entry)(struct sk\_buff \*msg, struct ib\_mr \*ibmr);

int (\*fill\_res\_cq\_entry)(struct sk\_buff \*msg, struct ib\_cq \*ibcq);

int (\*fill\_res\_qp\_entry)(struct sk\_buff \*msg, struct ib\_qp \*ibqp);

Srq,Nq, smac,sgid通过debugfs接口实现,用户输入index,打印调试信息。

对于不支持上述netlink的qp,mr, cq也用debugfs来实现。

host需要实现： qp\_debug/mr\_debug/cq\_debug 来实现调试信息netlink。

1. Debugfs调试接口

Host 需要实现： srq\_debugfs/sf\_debugfs/eq\_debugfs/smac\_debugfs/sgid\_debugfs 用于调试

1. 硬件调试接口

A.硬件特殊debug寄存器显示

B.QP追踪统计

C.硬件错误注入接口

D.物理内存读写调试接口

E.中断注入接口

## 硬件抽象接口层

### 10.1 Host 抽象操作

struct yib\_hw\_ops {

enum yrdma\_host\_type host\_type; //host类别

const int priv\_size; //硬件私有大小

int hw\_counter\_num; // 统计计数器的个数

char host\_name[YIB\_HOSTNAME\_MAXLEN];

void (\*global\_reset)(struct yib\_hw\_host \*hw); //第1个被调用硬件全局复位，整个host/pf/vf级别只能做一次

int (\*init\_caps)(struct yib\_hw\_host \*hw, struct yib\_sf \*sf); //第2个被调用的方法，该方法仅返回硬件的能力

int (\*start\_host)(struct yib\_hw\_host \*hw); //全局初始化

void (\*stop\_host)(struct yib\_hw\_host \*hw); //全局卸载

void (\*set\_pf\_speed\_bw)(struct yib\_hw\_host \*hw, uint32\_t rate\_limit, uint32\_t max\_burst\_sz, int host\_id, int pf\_id); //host/pf限速接口

void (\*get\_hw\_counter)(struct yib\_hw\_host \*hw, u64 \*stats, u32 \*types); //填充counter信息

int (\*host\_reg\_mmap)(struct yib\_hw\_host \*hw, struct vm\_area\_struct \*vma, ssize\_t length, u64 offset); //映射寄存器到用户空间

int (\*host\_def\_mmap)(struct yib\_hw\_host \*hw, struct vm\_area\_struct \*vma, ssize\_t length, u64 offset, int type); //host自定义用户空间映射器

int (\*get\_usable\_channels)(struct yib\_hw\_host \*hw);//用于pcie sriov en前检测vf数量是否满足。

void (\*get\_queue\_user\_info)(struct yib\_hw\_host \*hw, enum yib\_elem\_type type, void \*verbs, u32 hw\_id, u8 \*res, int \*len);//填充verbs资源建立后的返回值

void (\*host\_debugfs\_reg)(struct yib\_hw\_host \*hw); //读取寄存器内存

void (\*host\_debugfs\_mem)(struct yib\_hw\_host \*hw, u32 len, u64 addr); //读取主机物理内存

void (\*smac\_debugfs)(struct yib\_hw\_host \*hw);//打印整个smac表

void (\*sgid\_debugfs)(struct yib\_hw\_host \*hw);//打印整个sgid表

};

### 10.2 SF抽象操作

struct yib\_sf\_ops { //这个方法内只能使用sf信息,不能使用host的信息

enum yrdma\_host\_type host\_type;

const int priv\_size; //私有sf结构大小

int (\*sf\_pre\_init)(struct yib\_sf \*sf); //sf静态初始信息赋值

int (\*start\_sf)(struct yib\_sf \*sf); //sf初始化

void (\*stop\_sf)(struct yib\_sf \*sf); //sf卸载

void (\*add\_mac)(struct yib\_sf \*sf, u8 \*mac, int index, u8 port\_num);// 配置网卡mac

void (\*modify\_mac)(struct yib\_sf \*sf, u8 \*mac, int index, u8 port\_num, bool bdel);//网卡mac修改

void (\*set\_gid)(struct yib\_sf \*sf, bool brocev2, bool bipv4, int index, u8 \*raw, bool bdel); //配置gid

int (\*mrw\_alloc)(struct yib\_sf \*sf, struct yib\_mr \*mr, u32 \*lkey, u32 \*rkey); //分配mr,mw

void (\*mrw\_destroy)(struct yib\_sf \*sf, struct yib\_mr \*mr); //释放mr, mw

int (\*mr\_mtt\_init)(struct yib\_sf \*sf, struct yib\_mr \*mr, struct scatterlist \*sg, int npages, u32 page\_size, u64 pa0); //仅noamal\_mr使用

void (\*mr\_mpt\_update)(struct yib\_sf \*sf, struct yib\_mr \*mr); //操作mpt表

int (\*mr\_debug)(struct yib\_sf \*sf, struct yib\_mr \*mr, void\*msg);

//cq

void (\*cq\_info\_init)(struct yib\_sf \*sf, struct yib\_cq \*cq, bool b\_del); //CQ建立删除

void (\*cq\_notify\_update)(struct yib\_sf \*sf, struct yib\_cq \*cq, u32 flag);//cq notify更新

int (\*cq\_debug)(struct yib\_sf \*sf, struct yib\_cq \*cq, void \*msg);

//qp

void (\*qp\_info\_init)(struct yib\_sf \*sf, struct yib\_qp \*yqp, bool enable, bool use\_srq); //qp初始化

void (\*qp\_info\_update)(struct yib\_sf \*sf, struct yib\_qp \*yqp, u32 mask, bool state\_chg);//qp modify

void (\*qp\_query)(struct yib\_sf \*sf, struct yib\_qp \*qp, os\_qp\_attr \*attr, int mask, bool bdebug); //qp信息查询

int (\*qp\_debug)(struct yib\_sf \*sf, struct yib\_qp \*qp, void\*msg); //qp debug

//srq & rq

void (\*rq\_info\_init)(struct yib\_sf \*sf, struct yib\_rq \*yrq, bool enable, bool bsrq);//srq和rq都会调用这个步骤，rq时需要在qp\_info\_init之前

int (\*srq\_debugfs)(struct yib\_sf \*sf, struct yib\_srq \*ysrq);

//vf级别的限速

void (\*set\_speed\_bw)(struct yib\_sf \*sf, u32 rate\_limit, u32 max\_burst\_sz, u16 typical\_pkt\_sz);

void (\*sf\_debugfs)(struct yib\_sf \*sf);

};

### 10.3 IO操作

struct yib\_eq\_ops {

bool intr\_enable;

int (\*get\_eq\_inrt\_num)(struct yib\_sf \*sf);

int (\*get\_eq\_isize)(struct yib\_sf \*sf, struct yib\_eq \*eq);

bool (\*check\_eq\_empty)(struct yib\_sf \*sf, struct yib\_eq \*eq);

bool (\*eq\_handler)(struct yib\_sf \*sf, struct yib\_eq \*eq, u8\* buf);//中断中执行,返回是否提交到了event 队列中

void (\*eq\_ci\_db\_update)(struct yib\_sf \*sf, struct yib\_eq \*eq, int io\_cnt);

void (\*eq\_info\_init)(struct yib\_sf \*sf, struct yib\_eq \*eq, bool b\_del); //eq初始化与卸载

int (\*eq\_debugfs)(struct yib\_sf \*sf, struct yib\_eq \*yeq);

};

struct yib\_queue\_ops {

int cqe\_isize;

处于spinlock irqoff环境下执行

int (\*fill\_rqe)(struct yib\_hw\_host \*hw, struct yib\_rq \*rq, const void \*os\_wq, u8 \*buf, u32 length); //填充rqe/srqe

int (\*fill\_wqe)(struct yib\_hw\_host \*hw, struct yib\_qp \*qp, const void \*os\_wq, u8 \*buf, u32 length, u32 mask);//填充wqe

//qp\_cache是否使用，由底层的情况决定，当底层硬件有能力在cq中返回qp地址信息时不使用， 否则底层硬件可返回qpn(这时需要使用qp\_cache)

void (\*fill\_cqe)(struct yib\_hw\_host \*hw, struct yib\_cq \*cq, struct yib\_qp \*\*qp\_cache, void \*os\_cq, u8 \*buf);//填充wc

int(\*sw\_fill\_cqe)(struct yib\_hw\_host \*hw, struct yib\_cq \*cq, void \*os\_cq);//返回0表示处理成功，小于0表示不保序

bool (\*check\_sq\_full)(struct yib\_hw\_host \*hw, struct yib\_sq \*sq);

bool (\*check\_rq\_full)(struct yib\_hw\_host \*hw, struct yib\_rq \*rq);

bool (\*check\_srq\_full)(struct yib\_hw\_host \*hw, struct yib\_srq \*srq, int \*pos);

bool (\*check\_cq\_empty)(struct yib\_hw\_host \*hw, struct yib\_cq \*cq);

//bool (\*is\_resize\_cq)(u8 \*buf);

int (\*get\_sq\_item\_size)(int \*inline\_len, int \*max\_sg, bool is\_ud);//inline\_len和max\_sg为输入输出参数

int (\*get\_rq\_item\_size)(int \*max\_sg);

//db update io\_cnt为要提交的Io数量，处于spinlock irqoff环境下执行

void (\*sq\_pi\_db\_update)(struct yib\_hw\_host \*hw, struct yib\_sq \*sq, int io\_cnt); /的io数

void (\*rq\_pi\_db\_update)(struct yib\_hw\_host \*hw, struct yib\_rq \*rq, int io\_cnt);

void (\*srq\_pi\_db\_update)(struct yib\_hw\_host \*hw, struct yib\_srq \*srq, int pos);

void (\*cq\_ci\_db\_update)(struct yib\_hw\_host \*hw, struct yib\_cq \*cq, int poll\_cnt);

//该函数在spin lock外执行， 用于在io路径上 需要调度的场景(如doe下命令填充rqe)（不推荐使用)

void (\*rq\_post\_db\_ext)(struct yib\_hw\_host \*hw, struct yib\_rq \*rq, int io\_cnt);//ext结尾的函数指针可以为空

};

## 用户态provider 架构

**设计原则**：

1. 除了io之外的流程（管理路径）， 用户态不感知硬件的行为，所有硬件行为在内核态实现（但不同类别的软件差异，用户态需要感知）。
2. 用户态队列的内存由用户态分配， 内核态队列内存由内核态分配
3. 用户态能感知到硬件大类别的差异， bar空间管理由内核提供mmap给用户空间分配。

**抽象结构**：

struct yib\_hw\_ctx\_ops //硬件无关的抽象操作,所i有硬件都对应此结构

{

//硬件相关实现

int (\*hw\_context\_alloc)(void \*ptr) ;

// swtest\_hw\_context\_alloc(){ return &yib\_2100r\_hw\_ctx;}

int (\*hw\_context\_dealloc)(void \*ptr);

int (\*hw\_global\_map\_reg)(struct yib\_context \*ctx); //映射regi寄存器

int (\*hw\_global\_unmap\_reg)(struct yib\_context \*ctx);

//硬件相关的初始化函数

int (\*hw\_cq\_init)(struct yib\_context \* ctx, struct yib\_cq\* );

int (\*hw\_qp\_init)(struct yib\_context \* ctx, struct yib\_qp\* );

// int (\*hw\_rq\_init)(struct yib\_context \* ctx, struct yib\_rq\* );

int (\*hw\_rq\_init)(struct yib\_context \* ctx, struct yib\_qp\* );

int (\*hw\_cq\_uninit)(struct yib\_context \* ctx, struct yib\_cq\* );

int (\*hw\_qp\_uninit)(struct yib\_context \* ctx, struct yib\_qp\* );

int (\*hw\_rq\_uninit)(struct yib\_context \* ctx, struct yib\_qp\* );//srq,rq

int (\*hw\_mr\_init)(struct yib\_context \* ctx , struct yib\_mr\*);

int (\*hw\_mr\_uninit)(struct yib\_context \* ctx , struct yib\_mr\*);

//同步内核态 queue\_ops

int cqe\_isize;

int (\*fill\_rqe)(struct yib\_context \* ctx, struct yib\_rq \*rq, const void \*os\_wq, u8 \*buf, u32 length);

int (\*fill\_wqe)(struct yib\_context \* ctx, struct yib\_qp \*qp, const void \*os\_wq, u8 \*buf, u32 length, u32 mask);

//qp\_cache是否使用，由底层的情况决定，当底层硬件有能力在cq中返回qp地址信息时不使用， 否则底层硬件可返回qpn(这时需要使用qp\_cache)

void (\*fill\_cqe)(struct yib\_context \* ctx, struct yib\_cq \*cq, struct yib\_qp \*\*qp\_cache, void \*os\_cq, u8 \*buf);

int (\*sw\_fill\_cqe)(struct yib\_context \* ctx, struct yib\_cq \*cq, void \*os\_cq);//返回0表示处理成功，小于0表示不保序

bool (\*check\_sq\_full)(struct yib\_context \*, struct yib\_sq \*sq);

bool (\*check\_rq\_full)(struct yib\_context \*, struct yib\_rq \*rq);

bool (\*check\_srq\_full)(struct yib\_context \*, struct yib\_srq \*srq, int \*pos);

bool (\*check\_cq\_empty)(struct yib\_context \*, struct yib\_cq \*cq);

//bool (\*is\_resize\_cq)(u8 \*buf);

int (\*get\_sq\_item\_size)(enum ibv\_qp\_type qp\_type , int \*inline\_size , int \*max\_sge);//inline\_len和max\_sg为输入输出参数

int (\*get\_rq\_item\_size)(int \*max\_sge);

//db update

void (\*sq\_pi\_db\_update)(struct yib\_context \*, struct yib\_sq \*sq, int io\_cnt);

void (\*rq\_pi\_db\_update)(struct yib\_context \*, struct yib\_rq \*rq, int io\_cnt);

void (\*srq\_pi\_db\_update)(struct yib\_context \*, struct yib\_srq \*srq, int pos);

void (\*cq\_ci\_db\_update)(struct yib\_context \*, struct yib\_cq \*cq, int poll\_cnt);

void (\*rq\_post\_db\_ext)(struct yib\_context \*, struct yib\_rq \*rq, int io\_cnt);//ext结尾的函数指针可以为空

int (\*set\_capture)(struct yib\_qp \*qp , int enable);

struct yib\_queue\_ops queue\_ops;

};

## 2100r 固件/硬件接口层设计

硬件接口参考文件：<<驱动固件硬件接口》》

软件固件访问（fw.c)

struct yib\_fw\_req {

\_\_le32 pci\_info;

\_\_le32 byte4;

.....

\_\_le32 byte60;

os\_cdma\_t dma\_buf; //用于内部命令的dma缓冲区

};

//返回值表示成功或失败,0表示成功

int yib\_fw\_interface\_init(struct yib\_sf \*sf);

int yib\_fw\_send\_cmd(struct yib\_sf \*sf, struct yib\_fw\_req \*req);

int yib\_fw\_wait\_cmd(struct yib\_sf \*sf);

**固件命令总表**：

|  |  |  |
| --- | --- | --- |
| 序号 | 命令名(命令码) | 含义 |
| 1 | alloc\_channel(1) | 分配channel |
| 2 | query\_left\_channels(2) | 查询剩余能分配的channel数 |
| 3 | start\_channel(3) | 使能channel |
| 4 | destroy\_channel(4) | 销毁channel |
| 5 | query\_device(5) | 查询设备的能力 |
| 6 | create\_cq(6) | 创建cq |
| 7 | query\_cq(7) | 查询cq |
| 8 | destroy\_cq(8) | 销毁cq |
| 9 | create\_srq(9) | 创建srq |
| 10 | query\_srq(10) | 查询srq |
| 11 | destroy\_srq(11) | 销毁srq |
| 12 | query\_nq(12) | 查询nq |
| 13 | create\_qp(13) | 创建qp |
| 14 | modify\_qp(14) | 修改qp |
| 15 | query\_qp(15) | 查询qp |
| 16 | destroy\_qp(16) | 销毁qp |
| 17 | reg\_mr(17) | 注册mr |
| 18 | dereg\_mr(18) | 注销mr |
| 19 | add\_sgid(19) | 添加sgid |
| 20 | del\_sgid(20) | 删除sgid |
| 21 | set\_smac(21) | 设置smac |
| 22 | chg\_active\_port(22) | bonding主从切换 |
| 23 | set\_qos(23) | 设置调度的参数 |

## 2100r硬件资源管理

具体字段参看硬件datastructure。

软件对硬件的详细字段定义请参考： <http://gitlab.yusur.tech/PRD/DSPG/RAPT/rdma-header>

该项目定义了软件要使用的所有字段。

struct yib\_2100r\_resource {

struct yib\_page\_tbl hwqueues;

Struct yib\_frag\_buf \*hwqueues\_dat

struct yib\_mem\_tbl smacs;

struct yib\_frag\_buf \*smacs\_dat;

struct yib\_mem\_tbl sgids;

struct yib\_frag\_buf \*sgids\_dat;

struct yib\_mem\_tbl mpts;

struct yib\_frag\_buf \*mpts\_dat;

int qp\_cnt;

U32 qp\_server\_offset;

U32 qp\_common\_offset

U32 sq\_offset;

U32 rq\_offset

U32 cq\_offset;

U32 nq\_offset;

Int rq\_cnt;

Int cq\_cnt;

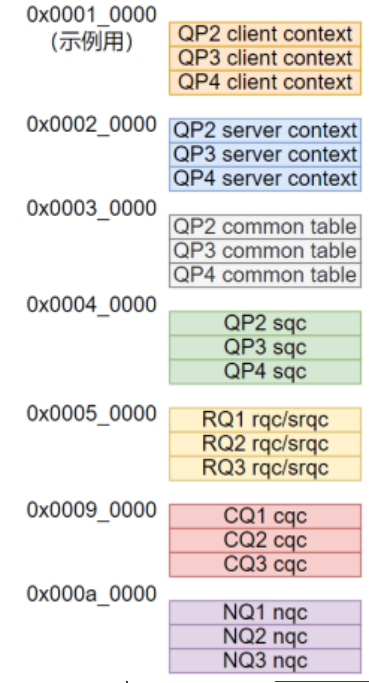
Int nq\_cnt;

int mpt\_cnt;

Int sgid\_len;

};

驱动初始化的时候填充该值。Hw\_queue的摆放顺序如下（注意一种资源必须4k对齐，不足的可以空着， 一个资源的size能被4k整除）



**资源类别：**

MPT, qpc, qp\_client, qp\_server, cq, n1, smac，sgid

硬件资源定义参考http://gitlab.yusur.tech/PRD/DSPG/RAPT/rdma-header

**资源bit访问定义接口：**

**如MPT中pd是[8,31]bit定义如下**

#define YIB\_MPT\_FIELD\_LOC(h, l) YIB\_FIELD\_LOC(struct yib\_hw\_mpt\_entry, h, l)

#define YIB\_MPT\_PD YIB\_MPT\_FIELD\_LOC(31, 8)

(MPT可替换成具体的资源名称）。

使用：

**yib\_hwres\_write**(结构体虚拟基地址, 字段名, 要写入的值);

Val = **yib\_hwres\_read**(结构体虚拟基地址, 字段名）；

例子： yib\_hwres\_read(mpt, YIB\_MPT\_PD); //获取pd域的值

获取资源虚拟地址的方法：

typdef enum

{

2100R\_TYPE\_CLIENT\_FLOW,

2100R\_TYPE\_SERVER\_FLOW,

2100R\_TYPE\_QPC\_COMMON,

2100R\_TYPE\_SQC,

2100R\_TYPE\_RQC,

2100R\_TYPE\_NQC,

2100R\_TYPE\_SGID\_TBL,

2100R\_TYPE\_SMAC\_TBL,

2100R\_TYPE\_MPT,

} r2100\_hwres\_type;

void \* r2100\_get\_hwres\_va(struct yib\_2100r\_resource\* hwres, int index, r2100\_hwres\_type);

u64 r2100\_get\_hwqpclient\_pa(struct yib\_2100r\_resource\* hwres, int index, r2100\_hwres\_type);

## 2100r寄存器操作流程

### 13.1 初始化

### 13.2 DB

### 13.3 AEQ处理流程

根据固件驱动接口文档中的返回类别，按如下分类处理

#### FW 完成处理

1. 填充命令完成信息到sf\_priv.fw.resp中
2. 唤醒fw命令等待完成进程。

#### QP Fatal

A. 固件不支持完整处理流程时（驱动来处理）：获取qp sq位置， 将所有软件已发送，而未完成的命令做flush处理； 获取qp rq位置，对所有软件已提交而未完成的命令做flush处理 （ei位置的错误，需要返回实际的err码（从pipe\_err, pipe\_num获取近似值）。

固件支持完整处理流程时： 仅上报qp\_fatal err, 同时将qp配置为error state.

1. 记录当前pi, 后续在提交的命令都flush处理。

（CQ有一个flush 队列，按该队列输出wc).

#### CQ ERR (CQ full)

ycq->ib\_cq.event\_handler(&ev, ycq->ib\_cq.cq\_context);

#### Device Error

Todo

#### (5) SRQ error

Todo

### 13.4 Debug

## 用户态工具

### 15.1 HQOS

yib\_hqos 工具：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 配置参数 | 说明 | 备注 |
| 1 | -d | 指定IB设备 |  |
| 2 | -t | host: host ratelimit配置 （需在pf上执行，驱动判断是否为pf)  vf: vf ratelimti配置  port: port ratelimit配置  cos: 配置硬件优先级的ratelimig | (CIR,CBS) C=0表示bypass (用户输入单位为kbps |
| 3 | -e | EBS，单位 kbps | -q -e -c (port/host/vf)一起使用 |
| 4 | -c | CBS， 单位 kbps |
| 6 | -m --pcp=  -m --dscp= | dscpmap: 配置dscp优先级映射表 (x,y) (将dspc优先级x,设成硬件优先级y)  pcpmap: (同上) |  |
| 7 | -r | Recvbuf: (a,b,c,d,e,f,g,h) 配置8个优先级的大小，单位KByte, 全0表示自动分配 |  |
| 8 | -p | pfc使能 |  |

-d，--device 指定设备, -d yusur\_0  
  
-r, --ratelimit 限速 -r cos,1,100,200

channel,00,200 （host,port只有pf能配置，框架层负责判断）

（注意cos需要指定优先级， 其它只指定cir,cbs)

-m, --cp2prio 映射表 -m pcp 1-20=0 21,23,25=1 22,19=3 dscp 33,63=4 (逐个解析，pcp的表项1到20映射优先级为0,21,23,25映射为优先级1)  
  
-M，--get\_cp 获取映射表，全部输出,输出格式：  
prio0: 01 02 03 04 05 06 07 08  
11 13 15  
prio1: 18 20 22  
prio2: 25  
  
  
  
-b, --set\_buf设置buff -b 12,23,23,,23,23,123,123 (没配的表示不修改) --- 只有pf让配置 （VF不让配置）  
  
  
-B, --get\_buf查询buf，输出  
  
priority buff:  
prio 0 1 2 3 4 5 6 7  
buff 12 0 13 15 13 13 13 13  
total buff:  
12345  
  
-p， --pfc设置pfc -p 1,0,1,1,0,0,0,0  
-p 1,0,,1,0,0,0,0 (,,表示不修改)  
-p 1,2,4,5=1 3,7,9=0 (1,2,4,5开，3,7,9 关，没配置的不修改)  
  
-P， --get\_pfc 查询pfc，输出：  
PFC：  
prio 0 1 2 3 4 5 6 7  
enable 0 0 1 0 1 1 0 0  
shreshole 12 222 333 3 9 3 3 3

用户态通过mmapYIB\_MMAP\_TYPE\_PRIO接口，将dscp/pcp\_array映射到 用户空间。

带宽<=100Mbps时，桶深CBS(Bytes)=带宽(kbps)\*1500(ms)/8，

带宽>100Mbps时，桶深CBS(Bytes)=100,000(kbps)\*1500(ms)/8。

mmap映射空间表：

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 起始地址 | 含义 | 大小 |
| 1 | 0 | DSCP映射表 | 64item\*4byte |
| 2 | 256 | Pcp映射表 | 16item\*4byte |
| 3 | 320 | RX buf 大小表 | 8\*4byte |
| 4 | 352 | PFC使能表 | 8 \*1byte,用bit控制 |
| 5 | 360 | PFC低水线 | 8\*4byte |
| 6 | 392 | PFC高水线 | 8\*4byte |

sysfs接口：

1. Rx\_buf mode: 0 全局， 1每优先级一个
2. rdma\_map: struct {

Int type; // 0 dscp, 1 pcp, 2: rx\_buf, 3 pfc enable 4: pfc low 5: pfc\_high

};

(3)

struct yib\_rdma\_qos\_info {

uint32\_t cir; //用户输入

uint32\_t cbs; //用户输入

uint32\_t level; //第几级的class 0:qp 1:cos 2:channel 3:host 4:port

uint32\_t obj\_index; //qpnx,cosx,hostx

uint32\_t is\_multi\_mode;

uint32\_t inherit\_mode; //继承模式，仅用于multi\_mode

uint32\_t bypass; //为1表示bypass

};