|  |  |
| --- | --- |
| 文档编号 |  |
| 文档版本 |  |
| 文档管控 | 内部公开 |
| 存档日期 |  |

昆明湖项目MemBlock模块验证文档

|  |  |
| --- | --- |
| 编 写： |  |
| 校 对： |  |
| 审 核： |  |
| 批 准： |  |

昆明湖V1项目

2023年12月6日

文档修订记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **序号** | **版本编号** | **变化状态** | **变更说明** | **作者** | **日期** |
| 1 | V0.1 | C |  |  |  |
|  |  |  |  |  |  |
| 3 |  |  |  |  |  |
|  |  |  |  |  |  |

\*变化状态：C—创建，A—增加，M—修改，D—删除

文档审批信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **版本** | **审核** | **会签** | **批准** | **备注** |
| V1.0 |  |  |  |  |
|  |  |  |  |  |

目 录

[1.简介 1](#_Toc156241253)

[1.1文档介绍 1](#_Toc156241254)

[1.2参考文档 1](#_Toc156241255)

[1.3术语说明 1](#_Toc156241256)

[2.设计规格 2](#_Toc156241257)

[3.功能描述 2](#_Toc156241258)

[3.1功能概述 2](#_Toc156241259)

[3.2功能详述 3](#_Toc156241260)

[3.2.1接收Dispatch流水级的指令派遣 3](#_Toc156241261)

[3.2.2原子指令的执行 3](#_Toc156241262)

[3.2.3 MMIO空间的Load/Store指令执行 3](#_Toc156241263)

[3.2.4处理保留站发射的指令 4](#_Toc156241264)

[3.2.5 Load指令的重发 4](#_Toc156241265)

[3.2.6基于保留站的反馈重发 6](#_Toc156241266)

[3.2.7 Load指令的提交 7](#_Toc156241267)

[3.2.8 Store指令的提交 7](#_Toc156241268)

[3.2.9 Store Buffer提升Store的性能 7](#_Toc156241269)

[3.2.10分支预测错误的恢复 8](#_Toc156241270)

[3.2.11中断与异常的恢复 8](#_Toc156241271)

[3.2.12 Load to Load违例检测与恢复 8](#_Toc156241272)

[3.2.13 Store to Load Forwarding违例检测与恢复 9](#_Toc156241273)

[3.2.13访存违例预测器 10](#_Toc156241274)

[3.2.14基于取指令的重发 10](#_Toc156241275)

[3.2.14 LQ监听DCache重填 10](#_Toc156241276)

[3.2.15 Uncache写Outstanding机制 11](#_Toc156241277)

[4.总体设计 11](#_Toc156241278)

[4.1整体框图 11](#_Toc156241279)

[4.2接口列表 12](#_Toc156241280)

[4.3接口时序 12](#_Toc156241281)

[4.3.1接收派遣阶段指令时序示例 12](#_Toc156241282)

[4.3.2接收保留站发射指令示例 12](#_Toc156241283)

[4.3.3写回结果总线时序示例 13](#_Toc156241284)

[4.3.4反馈重发时序示例 13](#_Toc156241285)

[4.4寄存器配置 14](#_Toc156241286)

[4.5补充说明 15](#_Toc156241287)

[5. 模块设计 18](#_Toc156241288)

[5.1二级模块Uncache 18](#_Toc156241289)

[5.1.1功能 18](#_Toc156241290)

[5.1.2整体框图 18](#_Toc156241291)

[5.1.3接口列表 18](#_Toc156241292)

[5.1.4接口时序 18](#_Toc156241293)

[5.1.5关键电路 18](#_Toc156241294)

[5.1.6三级模块MMIOEntry 18](#_Toc156241295)

[5.2二级模块AtomicsUnit 18](#_Toc156241296)

[5.2.1功能 18](#_Toc156241297)

[5.2.1整体框图 18](#_Toc156241298)

[5.2.2接口列表 18](#_Toc156241299)

[5.2.3接口时序 18](#_Toc156241300)

[5.2.4关键电路 18](#_Toc156241301)

[5.3二级模块LsqWrapper 18](#_Toc156241302)

[5.3.1功能 18](#_Toc156241303)

[5.3.2整体框图 18](#_Toc156241304)

[5.3.3接口时序 18](#_Toc156241305)

[5.3.4关键电路 18](#_Toc156241306)

[5.3.5三级模块StoreQueue 18](#_Toc156241307)

[5.3.6三级模块LoadQueue 24](#_Toc156241308)

[5.4二级模块LoadUnit 38](#_Toc156241309)

[5.4.1功能 38](#_Toc156241310)

[5.4.2整体框图 38](#_Toc156241311)

[5.4.3接口列表 38](#_Toc156241312)

[5.4.4接口时序 38](#_Toc156241313)

[5.4.5关键电路 38](#_Toc156241314)

[5.5二级模块StoreUnit 38](#_Toc156241315)

[5.5.1功能 38](#_Toc156241316)

[5.5.2整体框图 40](#_Toc156241317)

[5.5.3接口列表 40](#_Toc156241318)

[5.5.4接口时序 40](#_Toc156241319)

[5.5.5关键电路 41](#_Toc156241320)

[5.5.6三级模块SQDataModule 41](#_Toc156241321)

[5.5.7三级模块SQAddrModule 41](#_Toc156241322)

[5.5.8三级模块DatamoduleResultBuffer 41](#_Toc156241323)

[5.6二级模块StdExeUnit 41](#_Toc156241324)

[5.6.1功能 41](#_Toc156241325)

[5.6.2整体框图 41](#_Toc156241326)

[5.6.3接口列表 41](#_Toc156241327)

[5.6.4接口时序 41](#_Toc156241328)

[5.6.5关键电路 41](#_Toc156241329)

[5.7二级模块SBuffer 41](#_Toc156241330)

[5.7.1功能 41](#_Toc156241331)

[5.7.2整体框图 41](#_Toc156241332)

[5.7.3接口列表 41](#_Toc156241333)

[5.7.4接口时序 41](#_Toc156241334)

[5.7.5关键电路 41](#_Toc156241335)

[5.7.6三级模块SbufferData 41](#_Toc156241336)

[5.8异常时间情况汇总 41](#_Toc156241337)

[5.8.1后端重定向 41](#_Toc156241338)

[5.8.2Load指令异常行为 41](#_Toc156241339)

[5.8.3Store指令异常行为 41](#_Toc156241340)

[5.8.4原子指令异常行为 41](#_Toc156241341)

[6. PPA优化设计 41](#_Toc156241342)

[7. 验证关注点 42](#_Toc156241343)

[8. Floorplan 建议 42](#_Toc156241344)

[9. 遗留问题 42](#_Toc156241345)

1.简介

1.1文档介绍

本文档描述MemBlock的功能与设计。

主要用于指导详细设计与验证。

注意：本文档不包含TLB、DCache的部分，这些部分位于

1.2参考文档

* PLRU替换算法
* TileLink手册
* 访存违例预测器相关论文

1.3术语说明

|  |  |  |
| --- | --- | --- |
| **缩写** | **全称** | **描述** |
| MemBlock | Memory Block | 访存单元 |
| LSQ | Load Store Queue | Load Store指令队列 |
| LQ | Load Queue | Load指令队列 |
| SQ | Store Queue | Store指令队列 |
| RS | Reservation Station | 保留站 |
| ROB | Reorder Buffer | 重排序缓存 |
| SBuffer | Store Buffer | Store缓存 |
| MMIO | Memory-Mapped Input/Output | 内存映射的输入输出 |
| PMP | Physical Memory Protection | 物理内存保护机制 |
| ReplayInst | Replay Instruction | 需要从去指令开始重新执行的指令 |
| DCache | Data Cache | 数据缓存 |
| TLB | Translation Lookaside Buffer | 地址转译后备缓存器 |
| MSHR | Miss-Status Handing Register | Miss请求处理寄存器 |
| STD | Store Data | Store的数据 |
| STA | Store Address | Store的地址 |
| AGU | Address Generate Unit | 地址生成单元 |
| AtomicsUnit | Atomics Unit | 原子指令执行单元 |
| Uncache | Uncache | 非Cache |
| WPU | Way Prediction Unit | 路预测器 |
| LQR | Load Queue Replay | Load指令重发缓存 |
| LQRAR | Load Queue RAR | RAR冲突检测缓存 |
| LQRAW | Load Queue RAW | RAW冲突检测缓存 |
| VirtualLQ | Virtual Load Queue | 虚拟Load指令缓存 |
| PC | Program Counter | 程序计数器 |

表1.1 术语说明

2.设计规格

* 支持正常地址空间的load/store指令的执行与写回。
* 支持MMIO空间的load/store指令的执行与写回
* 支持原子指令的执行与写回。
* 支持乱序调度load/store指令
* 支持原子指令在后端非推测执行
* 支持地址与数据分离的store发射
* 支持基于保留站和LQ的指令重发
* 支持基于取指令阶段的指令重发
* 支持SBuffer优化store指令性能
* 支持SBuffer和SQ的虚拟地址匹配数据前递
* 支持乱序违例检查与恢复
* 支持可配置的替换算法
* 支持分支预测取消
* 支持终端异常发生时的流水线冲刷
* 支持LQ提前提交指令
* 支持Uncache 写outstanding机制

3.功能描述

3.1功能概述

访存流水线负责访存类型指令的具体执行流程，包括正常的访存地址空间以及外设相关的MMIO地址空间，还负责原子指令执行。访存流水线接受保留站发射的指令，根据访存指令的类型，经过不同指令的流水线，得到指令执行结果，写回到ROB中，并通知前递旁路网络，唤醒后续相关指令以及数据前递。

3.2功能详述

3.2.1接收Dispatch流水级的指令派遣

Load与store指令有复杂的控制机制，例如定序、前递、违例等，因此需要有一个队列来保存load与store指令，保证先进先出的顺序，进行相关的控制，这个队列就是load queue和store queue。当指令在前端被取上来完成了译码和重命名等操作后，load store指令需要派遣到ROB、load queue、store queue中，之后load与 store指令会继续进入到对应的保留站，等待可以发射之后，发射到MemBlock的流水线中，此时在load queue和store queue中都能找到对应的项，方便后续的操作。

3.2.2原子指令的执行

原子访存指令与普通的load与store指令执行流程不同，它没有相应的队列，因为原子指令需要非推测执行，要保证整个后端流水线只有这一条指令在执行。当原子指令来到派遣阶段首先会先等待其前面的所有指令完成提交，然后开始执行这条原子指令，并阻塞后续所有指令的执行，直到原子指令执行完毕完成提交。原子指令会复用store的保留站，将地址从STA保留站发射出来，将数据从STD保留站发射出来，给到MemBlock的AtomicsUnit，AtomicsUnit中有一系列状态机控制原子指令一步一步去执行，例如先进行虚实地址转换，再访问DCache，最后写回。

原子指令的写回使用 load 的写回端口。（这样设计的原因是，原子指令会有和 store 类似的操作数，和load 类似的写回结果数）

3.2.3 MMIO空间的Load/Store指令执行

MMIO空间的load store指令是对外设的操作，这类指令不能推测执行，需要等待其成为ROB的头部才能真正去执行，即这条指令之前的指令已经全部完成提交，即将提交这条指令。对于load而言，必须已经完成了虚实地址转换，得到了物理地址；对于store而言必须完成了虚实地址转换并且要写的数据也已经就绪。待这些条件都满足时，会启动对应的状态机，一步一步控制MMIO指令的执行，例如load会由load queue中的状态机进行控制，先访问Uncache通道，得到结果后再由load queue将其写回到ROB与结果总线，ROB提交这条指令；store会由store queue中的状态机进行控制，步骤与load类似。

3.2.4处理保留站发射的指令

MemBlock有两条load流水线，两条store地址流水线，两条store数据流水线，每一条流水线都能独立接收对应的保留站发射出来的指令执行请求。不同流水线的各个阶段会进行不同的处理。

store数据流水线非常简单，分两级，std保留站发射的store在store queue中找到dispatch阶段分配到的相应项，写入store的数据，并标识数据已经就绪，流出流水线时通知ROB该条store指令的数据已经就绪。

store地址流水线分四级，会进行虚实地址转换，做store to load forwarding违例检查，在store queue的对应项中标识地址已经就绪，流出流水线时通知ROB该条store指令的地址已经就绪。

load流水线比较复杂，分四级，会进行虚实地址转换，做load-load违例检查，forward SBuffer与store queue的数据，访问DCache，将load的执行情况更新到load queue的对应项中，流出流水线时将load的结果通知到结果总线与ROB。

3.2.5 Load指令的重发

load流水线是非阻塞的，即不能因为某些特殊情况将指令卡在某一级流水线。例如load指令流水线访问TLB进行地址转换时，发现TLB miss，不能将这条指令一直停留在这一级流水线中等待TLB命中，需要让它从流水线中退出，让后续拥有执行条件的指令继续运行。对于TLB miss的load指令，它本身是没有完成的，所以需要让保留站重新发射这条指令。

load需要重发的情况如下：

* 在load流水线第二级判断
  + 访存违例预测
* 在load流水线第三级判断
  + 访存违例预测
  + TLB miss
  + store queue数据前递失败
  + Miss Queue拒绝
  + DCache miss
  + WPU预测失败
  + Bank conflict
  + LQRAR拒绝
  + LQRAW拒绝
  + 流水线访存违例

load指令从保留站发射出来后，保留站不会立马清空对应的指令，而是继续等待MemBlock发来反馈。MemBlock中的流水线会根据具体的执行情况判断是否需要让保留站重发指令。当load指令需要重发时，会优先选择基于LQR的重发机制，当LQR无法接收更多load重发请求时，则选择基于保留站的重发机制。

* 基于LQR重发
  + 入队：在load流水线第三级时，从FreeList中选择一个有效项的索引，在load流水线第四级判断load是否需要被重发，当load需要重发时使用该索引将相关信息写入LQR，同时更新Age Matrix以及相应标志位（blocking、scheduled和cause等）。
  + 出队：LQR选择逻辑有两种机制：基于LQ Index和基于Age Matrix。
    1. 基于LQ Index

根据最老的地址已经准备好的load的LQ Index，计算之后几条load的LQ Index，通过比较相应的LQ Index，如果LQ Index匹配，并且load指令已经可以被重发，则选择该load指令，进入重发阶段。

* + 1. 基于Age Matrix

根据load指令第一次进入LQR的时间选择最早进入LQR并且已经可以被重发的load指令，进入重发阶段

* + 重发优先级：
    1. 基于LQ Index的优先级高于基于Age Matrix
    2. DCache miss、数据前递失败和DCache重填唤醒的优先级高于其他重发情况。

load指令被重发后，LQR不会立即移除对应项项，而是将scheduled位置位true，如果load被重发经过流水线后，仍然需要重发，则仍将信息保存至上一次分配的位置，并将scheduled位置位false。当LQR有空间接受新的load重发请求时，可以将load指令从保留站中移除。

* 基于保留站重发

当LQR无法接收更多新的load的重发请求时，则需要通过保留站重发。

3.2.6基于保留站的反馈重发

store流水线是非阻塞的，即不能因为某些特殊情况将指令卡在某一级流水线。例如store指令流水线访问TLB进行地址转换时，发现TLB miss，不能将这条指令一直停留在这一级流水线中等待TLB命中，需要让它从流水线中退出，让后续拥有执行条件的指令继续运行。对于TLB miss的store指令，它本身是没有完成的，所以需要让保留站重新发射这条指令。

store指令从保留站发射出来后，保留站不会立马清空对应的指令，而是继续等待MemBlock发来反馈。MemBlock中的流水线会根据具体的执行情况判断是否需要让保留站重发指令。

store需要基于保留站重发的情况如下：store虚实地址转换失败（store流水线的第三级）。

* + 1. Load指令的提交

当load指令出现三种情况时，可以立即提交：1）不需要重发；2）出现异常；3）不是Uncache指令。当出现上述情况时，流水线会通知VirtualLQ可以立即提交该指令，释放对应的资源。

* + 1. Store指令的提交

store指令在ROB数据与地址就绪之后就能够提交，一旦提交，store queue中对应这一项需要做一个标记，表明已经被提交，这些项之后不能被取消，之后需要择时将已经提交的store写入到store buffer中。

* + 1. Store Buffer提升Store的性能

Sbuffer可以提高store的执行效率，但是也会带来实现上的许多问题，store queue中已经提交的store指令可以写入到Sbuffer中， Sbuffer再择时将数据写入到dcache中。因此已经提交的store可能会存在于store queue中，也可能存在于Sbuffer中。

Sbuffer每一项是一个cacheline，每个cacheline是 64 bytes，如果若干个store的地址都在同一个cacheline，可以对这些store进行写合并， 每个周期能同时写入2条store指令，每条store指令的数据宽度为16 bytes。Sbuffer一共有16项，当容量超过设定好的阈值时, Sbuffer会执行换出操作, 使用PLRU选出要写入数据缓存的cacheline, 并将其写入DCache。

Sbuffer支持被动刷新机制，在刷新信号为高时，Sbuffer中所有项都会被刷入到DCache中。例如Fence指令要求store queue中还未写入到Sbuffer的store指令全部写入到Sbuffer，并且将Sbuffer中所有未提交到一致性缓存层次中的数据进行提交，来保证多核一致性。

Sbuffer可以为load指令提供前递数据查询与数据供给，load在流水线的处理过程中会来查询Sbuffer，发现虚地址匹配，物理地址匹配就可以将Sbuffer的数据前递给load。

Sbuffer支持自动刷新机制，当前递过程中发现虚地址匹配，但物理地址不匹配或者物理地址匹配，但虚拟地址不匹配的情况都会将自身所有数据刷入到DCache中。

为了解决SBuffer项数有限导致store queue拥堵的情况，SBuffer支持强制快写的机制：store queue采用双阈值（上阈值和下阈值），当store queue提交项超过上阈值时，通知SBuffer快向DCache写入数据，并且提高SBuffer写DCache时的优先级，知道store queue的提交向少于下阈值时停止SBuffer快些。

Sbuffer 支持超时刷新机制。

3.2.10分支预测错误的恢复

处理器每时每刻都有可能发生分支预测错误，为了获得更高的性能，一旦检测到错误的发生，就需要进行状态恢复。已经进入到load queue与store queue的指令需要被取消，已经进入到流水线中执行的load与store指令也需要被取消，分支预测错误发生时刻正在进行派遣的load与store指令也需要被取消。SBuffer中保留的是已经被ROB提交的store结果，这些store指令顺序一定排在分支错误的分支指令之前，所以SBuffer不需要对分支预测错误进行处理。

3.2.11中断与异常的恢复

当ROB的head的指令提交时有异常，或者有中断产生，需要冲刷流水线并重定向pc，MemBlock中流水线内所有的指令需要被取消，load queue中所有指令都需要被取消，store queue中未提交的所有指令需要被取消，SBuffer不受影响。

3.2.12 Load to Load违例检测与恢复

多核环境下会出现load to load违例：单核环境下相同地址的load乱序执行本来是不关心的，但是如果两个load之间有另外一个核做了相同地址的store，并且本身这个核的两个load做了乱序调度，就有可能导致新的load没有看到store更新的结果，但是旧的load看到了，出现了顺序错误。

这种情况有一个特征，当前DCache一定会收到L2 cache发来的Probe请求，使得DCache主动释放掉这个数据副本，这时DCache会通知load queue，将相同地址的load queue中已经完成访存的项做一个release标记。后续发往流水线的load指令会查询load queue中在它之后相同地址的load指令，如果存在release标记，就发生了load-load违例。

LQRAR用于保存已经完成的load指令的用于load to load违例检测的信息。当load指令处于load流水线第三级（即loadUnit的s2）时，查询并分配空闲项将信息保存入LQRAR，在流水线第四级时得到load to load违例检查的结果，如果出现违例则需要刷新流水线，给RedirectGenerator部件发送重定向请求, 冲刷违例的load之后的所有指令。

* + 1. Store to Load Forwarding违例检测与恢复

由于load和store在流水线中都是乱序执行，会经常出现load越过了更老的相同地址的store，即这条load本应该从store 通过前递得到数据，但是由于store地址或者数据没有准备好，导致这条load没有前递到store的数据就已经提交，后续使用这条load结果的指令也都发生了错误，于是产生store to load forwarding违例。

当store address通过STA保留站发射出来进入store流水线时，会去查询LQRAW中在这条store后面的所有已经完成访存的相同地址的load，以及load流水线中正在进行的在该条store之后的相同地址的load，一旦发现有，就发生了store to load forwarding违例，可能有多个load发生了违例，需要找到离store最近的load，也就是最老的违例的load。

store to load forwarding违例较为频繁，所以一旦检测到store to load forwarding违例就需要立刻恢复。发生store to load forwarding违例后，找到最老的违例的load，给RedirectGenerator部件发送重定向请求，冲刷违例的load及之后的所有指令，将PC重定向到这条违例的load，这个冲刷过程类似于分支预测失败的冲刷过程，MemBlock中需要取消相关的流水线中的load与 store指令，取消相关的在load queue和store queue中的指令。

3.2.13访存违例预测器

每发生一次store to load forwarding违例，产生的代价很大，所以访存违例预测器设计的主要目的是降低store to load forwarding违例发生的次数。昆明湖架构的访存违例预测器采用Store Set算法：Store Set算法采用两张查找表Store Set ID Table (SSID) 和 Last Fetched Store Table (LFST)，分别记录LFST的索引和load指令依赖的store指令的ROB index。在发生store to load forwarding违例时, 触发违例的指令的PC会被传递到访存违例预测器以进行更新。每过一个刷新间隔, 违例预测器中的信息会被无效化。刷新间隔可以使用slvpredctl CSR寄存器配置。

在昆明湖架构中，不采用LFST，load指令的访存违例的结果，是通过流水线级时，通过利用SSID查询store queue，在load之前的store是否有匹配的项的结果来判断是否需要等待。

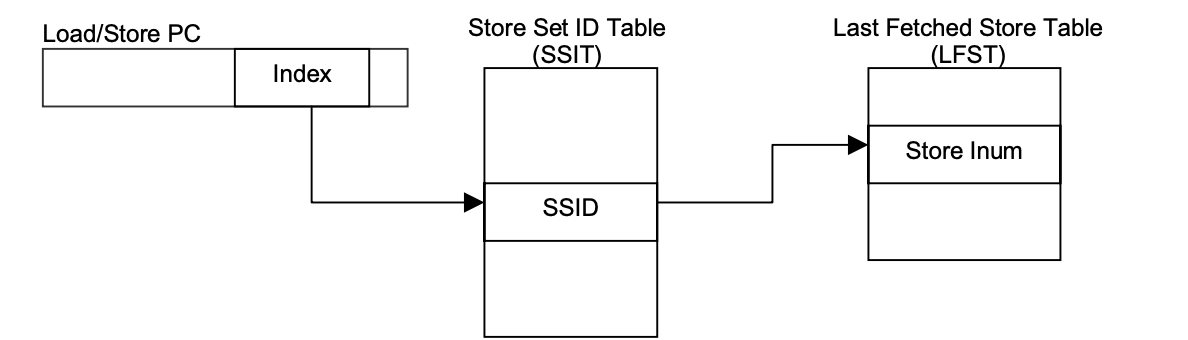


图3.1 Store Set算法框图

3.2.14基于取指令的重发

当load指令在load流水线中检测到从SBuffer或store queue前递数据时，发生了虚实地址不匹配的问题，就需要将这条指令的控制信号中标识需要从取指重发的信号置为高，等待其成为ROB的head提交时，冲刷流水线，并重定向PC，重新开始执行这条指令以及后续的所有指令。

* + 1. LQ监听DCache重填

load由load保留站发射出来后，经过load流水线进行处理，完成了物理地址转换，在访问DCache时出现了DCache miss，这条load在流出流水线时不会写回到ROB和更新结果总线，而是将这个情况报告给LQR，让LQR负责处理这条DCache miss的load。

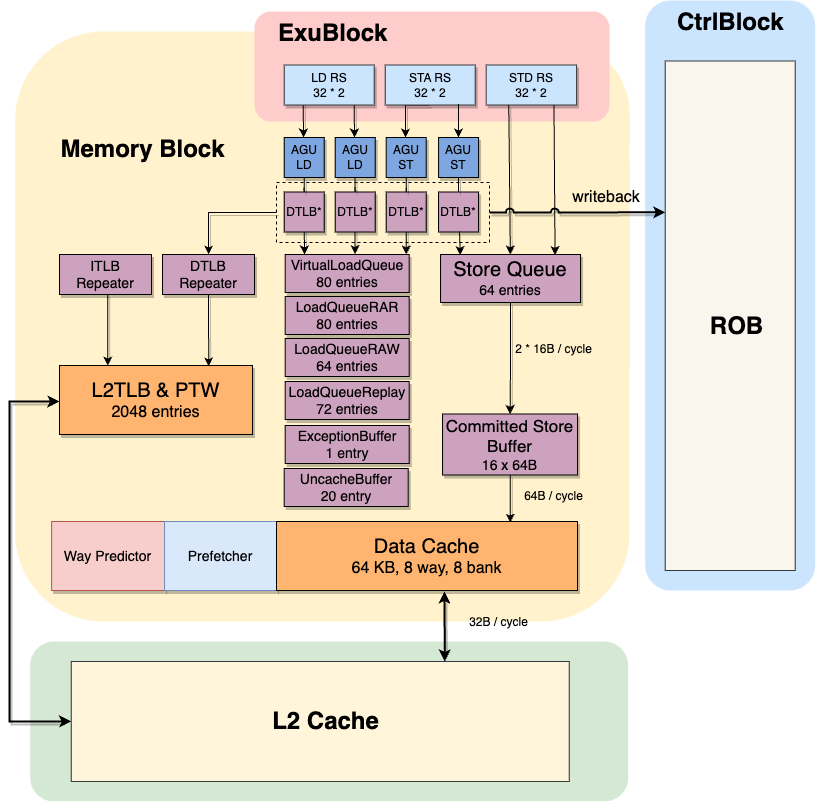
DCache每一次从L2 cache得到一个cacheline，首先会先于数据到达之前3拍推测发送Hint信号，以提前告知LQR唤醒指令，DCache会将其Hint请求发送给LQR，LQR中MSHR ID与这个cacheline相同，相当于完成了DCache miss的load的重填，更新对应的load，标记其不再miss，可以进入重发选择阶段。

* + 1. Uncache写Outstanding机制

MemBlock支持Uncache的顺序读操作和Outstanding的写操作，用户可以通过修改PMP的atomic标志位打开或者关闭相应的地址空间的Uncache的outstanding写功能， 当outstanding功能关闭是，uncache的写操作需要等待store指令变为ROB中最老的指令时，才能进行Uncache写操作， 等待Uncache写操作完成之后ROB才能。

4.总体设计

4.1整体框图



图片4.1 Memblock模块整体框图

4.2接口列表

详见IO卷

4.3接口时序

4.3.1接收派遣阶段指令时序示例

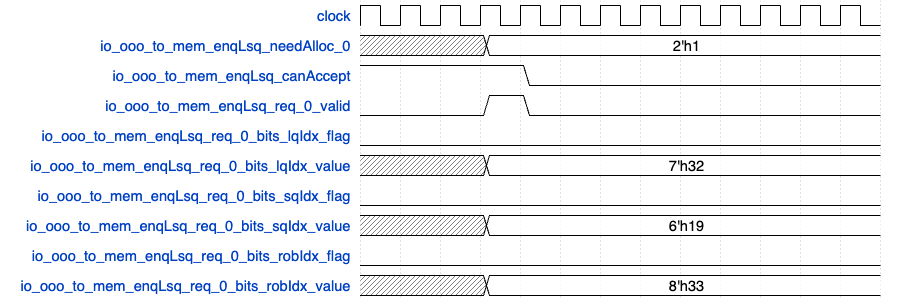


图 4.2 接收派遣阶段指令时序示例

needAlloc信号有效时，表明dispatch queue需要向load store或者store queue派遣指令，会检查当前队列中的容量，如果load queue和store queue都能接收派遣，即可把enqLsq\_req\_\*\_valid置位，指令在下一拍分配到队列中。

4.3.2接收保留站发射指令示例

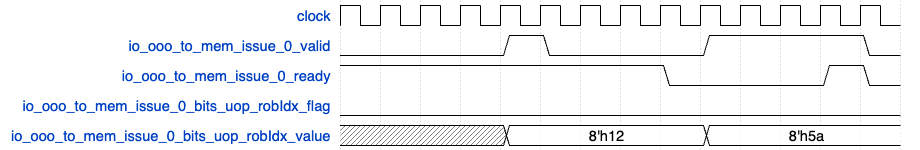


图 4.3 接收保留站发射指令示例

当issue\_\*\_ready和issue\_\*\_valid同时为高时，即完成了保留站发射指令的握手，保留站将load或者store指令发送到对应的流水线上，issue\_\*\_bits包含了处理这条指令所需要的全部信息。

4.3.3写回结果总线时序示例

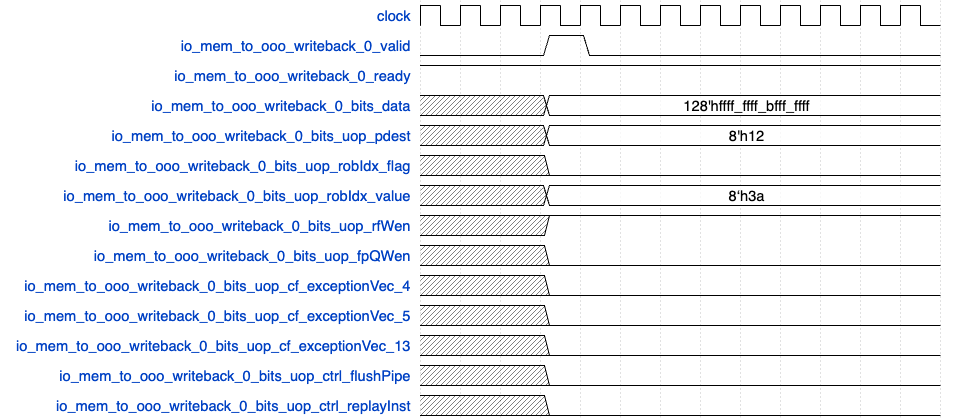


图4.4 写回结果总线时序示例

当writeback\_\*\_ready和writeback\_\*\_valid同时为高时，即完成了指令经过流水线后写回到ROB和结果总线的握手，writeback\_\*\_bits的结构与issue\_\*\_bits基本相同，指令从保留站发射出来后经过流水线各个阶段，各个阶段可能会修改bits的内容，流出最后一级变成writeback\_\*\_bits，包含指令的结果，控制信息，异常信息等，可以用于更新ROB，唤醒后续的指令。

4.3.4反馈重发时序示例

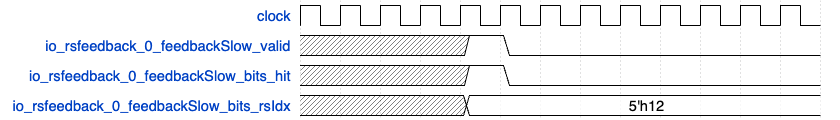


图 4.5 反馈重发时序示例

rsfeedback\_\*\_feedbackSlow\_valid为高，且rsfeedback\_\*\_feedbackSlow\_bits\_hit为高，就说明这条load或者store不需要让保留站进行重发，指令可以成功执行，保留站在接受到这个信息后可以将指令对应的项出队，不再占用保留站的空间，上图实例的是load不需要重发的情况。

若rsfeedback\_\*\_feedbackSlow\_valid为高，且rsfeedback\_\*\_feedbackSlow\_bits\_hit为低，就说明这条load或者store需要让保留站进行重发，同时还会反馈重发信息，rsIdx指明指令所在保留站的位置，sourceType指明重发的原因，如果是dataInvalid类型的重发，还会指明load指令依赖的store所在store queue的位置。

4.4寄存器配置

表4.2 csr配置寄存器说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器** | **地址** | **复位值** | **属性** | **描述** |
| smblockctl | 0x5C3 | 64’h77 | RW | bit0-3: sbuffer的阈值  bit4: 是否打开load load违例检查  bit5: 是否打开软件预取  bit6: 是否开启cache ecc error检测  bit7:是否打开uncache写outstanding功能  bit8-63: 保留 |
| slvpredctl | 0x5C2 | 64’h60 | RW | bit0: 是否关闭打开load违例预测  bit1: load违例预测相关  bit2: store set违例预测器配置参数  bit3： store set违例预测器配置参数  bit4-8： 违例预测器刷新时间  bit9-63: 保留 |
| CACHE\_OP | 0x5C5 | 64’h0 | RW | Cache操作的opcode，取值如下：  0：读tag ecc  1：读data ecc  2：读tag  3：读data  4：写tag ecc  5：写data ecc  6：写tag  7：写data  8：flush block |
| OP\_FINISH | 0x5C6 | 64’h0 | RW | Cache操作的结束信号 |
| CACHE\_LEVEL | 0x5C7 | 64’h0 | RW | 操作cache的id |
| CACHE\_WAY | 0x5C8 | 64’h0 | RW | 操作cache的路 |
| CACHE\_IDX | 0x5C9 | 64’h0 | RW | 操作cache的组 |
| CACHE\_BANK\_NUM | 0x5CA | 64’h0 | RW | 操作cache的bank |
| CACHE\_TAG\_ECC | 0x5CB | 64’h0 | RW | 操作cache写入的tag ecc |
| CACHE\_TAG\_BITS | 0x5CC | 64’h0 | RW | 保留 |
| CACHE\_TAG\_LOW | 0x5CD | 64’h0 | RW | 操作cache写入的tag |
| CACHE\_TAG\_HIGH | 0x5CE | 64’h0 | RW | 保留 |
| CACHE\_ECC\_WIDTH | 0x5CF | 64’h0 | RW | 保留 |
| CACHE\_DATA\_ECC | 0x5D0 | 64’h0 | RW | 操作cache写入的data ecc |
| CACHE\_DATA\_0 | 0x5D1 | 64’h0 | RW | 操作cache写入的data word0 |
| CACHE\_DATA\_1 | 0x5D2 | 64’h0 | RW | 操作cache写入的data word1 |
| CACHE\_DATA\_2 | 0x5D3 | 64’h0 | RW | 操作cache写入的data word2 |
| CACHE\_DATA\_3 | 0x5D4 | 64’h0 | RW | 操作cache写入的data word3 |
| CACHE\_DATA\_4 | 0x5D5 | 64’h0 | RW | 操作cache写入的data word4 |
| CACHE\_DATA\_5 | 0x5D6 | 64’h0 | RW | 操作cache写入的data word5 |
| CACHE\_DATA\_6 | 0x5D7 | 64’h0 | RW | 操作cache写入的data word6 |
| CACHE\_DATA\_7 | 0x5D8 | 64’h0 | RW | 操作cache写入的data word7 |
| CACHE\_ERROR | 0x5D9 | 64’h0 | RW | Cache error时的地址 |

4.5补充说明

|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **描述** | **位宽** | **编码** |
| \*\_ fuType | 功能部件类型 | 4 | 4’b0000: jmp  4’b0001: i2f  4’b0010: csr  4’b0110: alu  4’b0100: mul  4’b0101: div  4’b0011: fence  4’b0111: bku  4’b1000: fmac  4’b1011: fmisc  4’b1010: fDivSqrt  4’b1100: ldu  4’b1101: stu  4’b1111: mou |
| \*\_\_fuOpType | 功能部件操作码 | 7 | 7’b0000000: lb or sb  7’b0000001: lh or sh  7’b0000010: lw or sw or lr\_w  7’b0000011: ld or sd or lr\_d  7’b0000100: lbu  7’b0000101: lhu  7’b0000110: lwu or sc\_w  7’b0001000: prefetch\_i  7’b0001001: prefetch\_r  7’b0001010: prefetch\_w or amoswap\_w  7’b0000111: cbo\_zero or sc\_d  7’b0001100: cbo\_clean  7’b0001101: cbo\_flush  7’b0001110: cbo\_inval or amoadd\_w  7’b0001110: lr\_w  7’b0010010: amoxor\_w  7’b0010110: amoand\_w  7’b0011010: amoor\_w  7’b0011110: amomin\_w  7’b0100010: amomax\_w  7’b0100110: amominu\_w  7’b0101010: amomaxu\_w  7’b0001011: amoswap\_d  7’b0001111: amoadd\_d  7’b0010011: amoxor\_d  7’b0010111: amoand\_d  7’b0011011: amoor\_d  7’b0011111: amomin\_d  7’b0100011: amomax\_d  7’b0100111: amominu\_d  7’b0101011: amomaxu\_d |

# 5. 模块设计

5.1二级模块Uncache

5.1.1功能

5.1.2整体框图

5.1.3接口列表

5.1.4接口时序

5.1.5关键电路

5.1.6三级模块MMIOEntry

5.2二级模块AtomicsUnit

5.2.1功能

5.2.1整体框图

5.2.2接口列表

5.2.3接口时序

5.2.4关键电路

5.3二级模块LsqWrapper

5.3.1功能

5.3.2整体框图

5.3.3接口时序

5.3.4关键电路

5.3.5三级模块StoreQueue

5.3.5.1功能

store queue 是一个队列，用来装所有的 store 指令，功能如下：

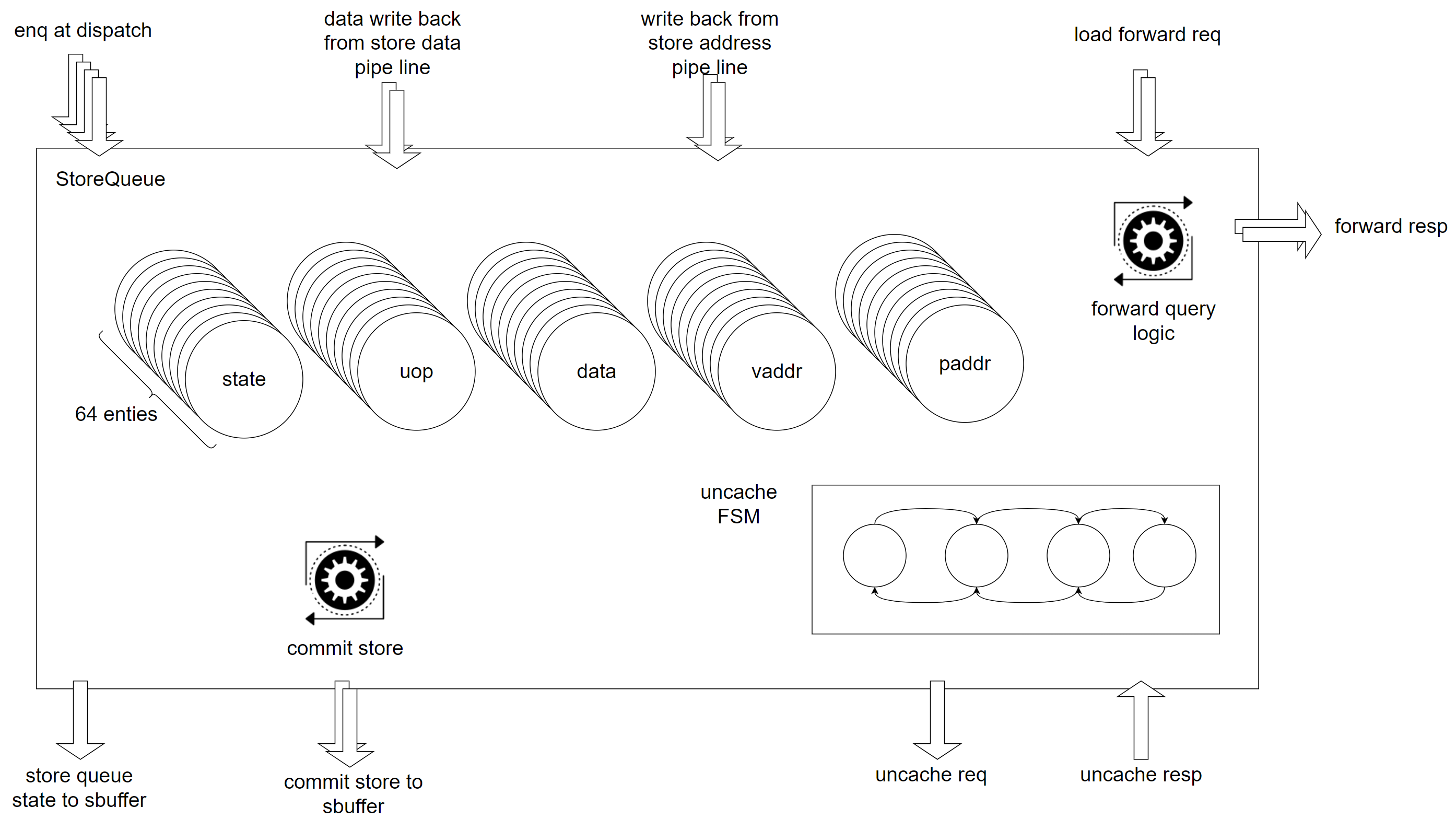
* 在跟踪 store 指令的执行状态
* 存储 store 的数据，跟踪数据的状态（是否到达）
* 为 load 提供查询接口，让 load 可以 forward 相同地址的 store
* 负责 MMIO store 的执行
* 将被 ROB 提交的 store 写到 sbuffer 中

store 进行了地址与数据分离发射的优化，即 StoreUnit 是 store 的地址发射出来走的流水线，StdExeUnit 是 store 的数据发射出来走的流水线，是两个不同的保留站，store 的数据就绪了就可以发射到 StdExeUnit，store 的地址就绪了就可以发射到 StoreUnit

**store queue 中每一项都有若干状态位来表示这个 store 处于什么样的状态**

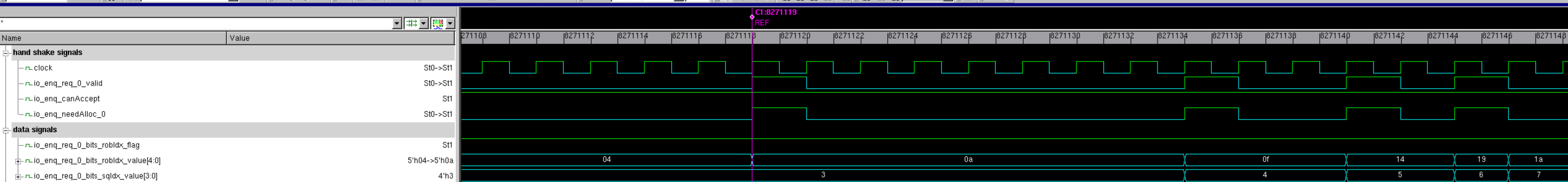
* allocated：RS在storeQueue队列有空闲时，会设置这个entry的allocated状态，开始记录这条store 的生命周期。同时发射到StoreUnit/ StdExeUnit 2条流水。当这条store指令被提交到Sbuffer时，allocated状态被清除。
* addrvalid：在StoreUnit的S1更新，表示是否已经经过了地址转换得到物理地址，用于 load forward 检查时的 cam 比较。
* datavalid：在StdExeUnit 的S1更新，表示store 的数据是否已经被发射出来，是否已经可用
* committed：在store 是否已经被 ROB commit 了
* pending：在StoreUnit的S2更新，在这条 store 是否是 MMIO 空间的 store，主要是用于控制 MMIO 的状态机
* mmio：在StoreUnit的S2更新，这条 store 是否是 MMIO 空间的 store，主要是用于控制对 sbuffer 的写

5.3.5.2整体框图



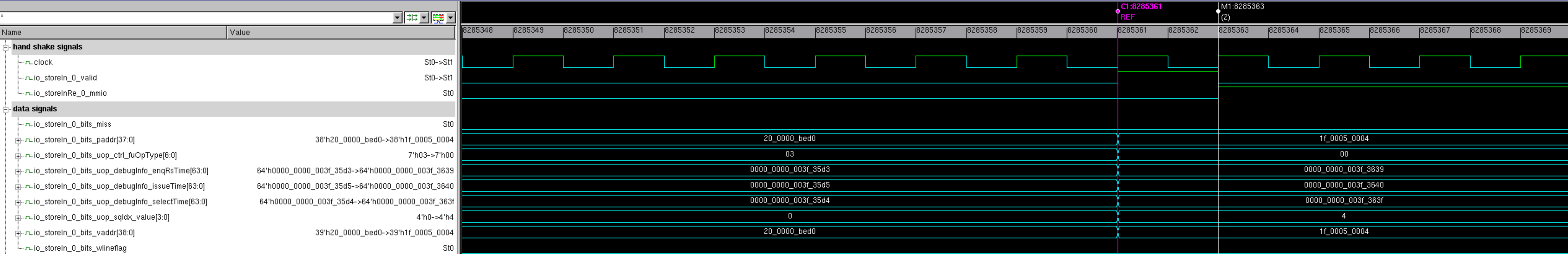
5.3.5.3接口时序

* 接收指令派遣时序示例



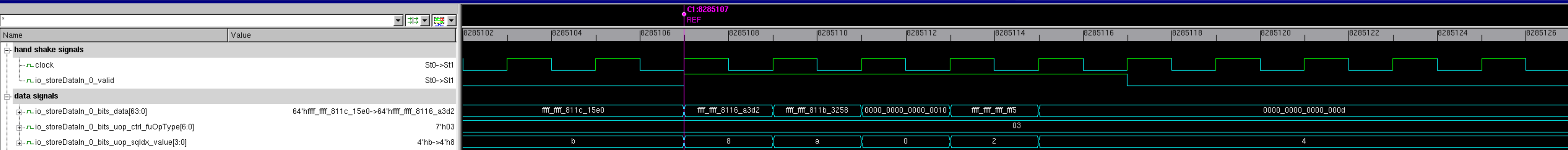
Sq接收派遣时序与lq类似，可以参考lq。

* 地址流水线写回store指令时序示例



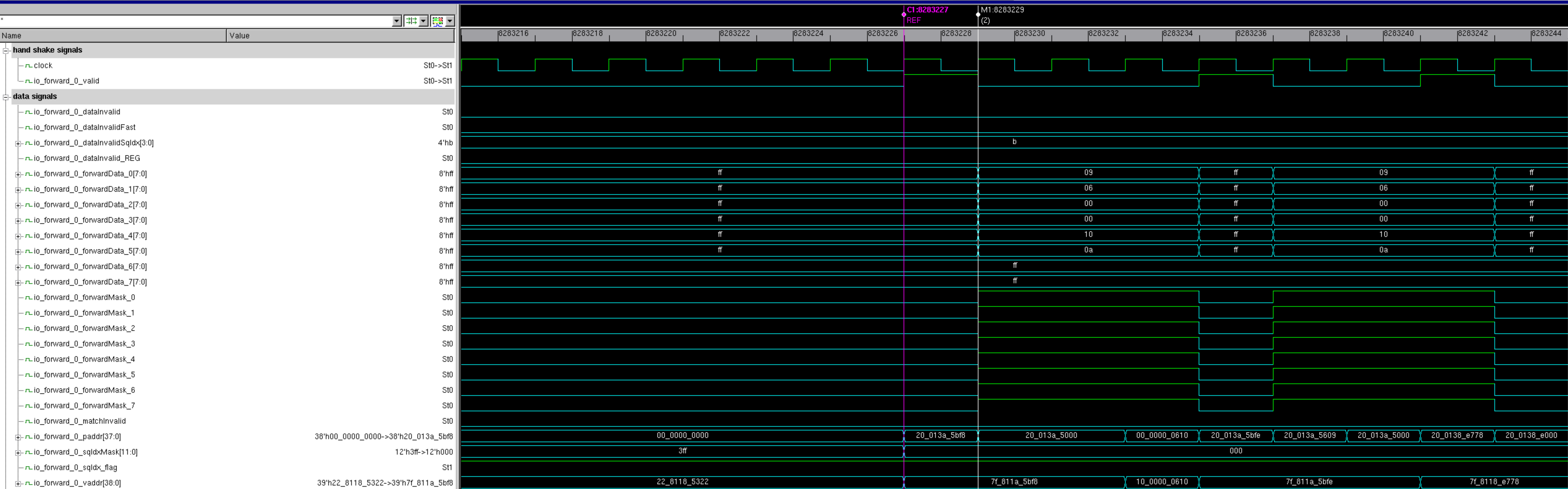
当io\_storeIn\_\*\_valid为高时，表示store地址流水线s1将指令写回到sq，在下一拍，即地址流水线的s2，如果store为mmio，会将io\_storeRe\_\*\_mmio置为高。

* 数据流水线写回store数据时序示例



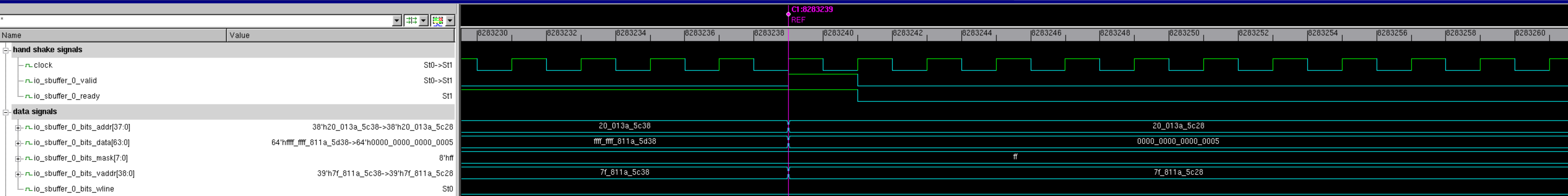
当io\_storeDataIn\_\*\_valid为高时，表示数据流水线将store的数据写给sq，数据为io\_storeDataIn\_\*\_bits\_data。

* 前递请求时序示例



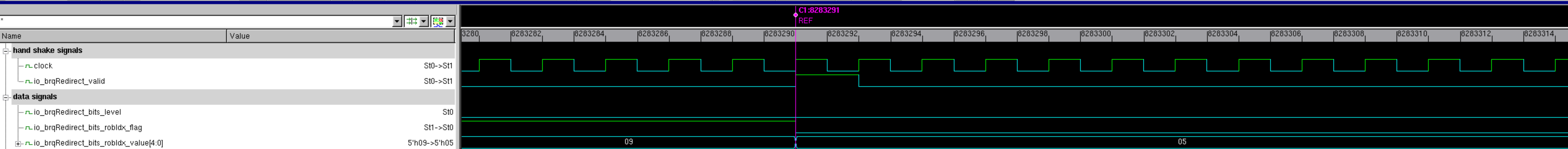
当io\_forward\_0\_valid为高时，表示load流水线给sq发来了前递查询请求，在请求到来的一拍之后得到响应，响应的数据为io\_forward\_0\_bits\_\*。

* 给sbuffer发送请求时序示例



当io\_sbuffer\_0\_valid和io\_sbuffer\_0\_ready为高时，表示握手成功，将sq的写请求发送给了sbuffer，具体请求的数据为io\_sbuffer\_0\_bits\_\*。

* 后端重定向时序示例



当io\_brqRedirect\_valid为高时，表示后端发生了重定向，sq需要取消指令和更新指针，重定向的具体信息在io\_brqRedirect\_bits\_\*。中

5.3.5.4关键电路

**入队与出队**

* STQ每次最多会有2个entry入队，通过入队指针enqPtrExt控制。在dispatch阶段最多可以分配2个entry，指针每次右移1位或2位。通过比较入队指针enqPtrExt和出队指针deqPtrExt得出已经在队列中有效entry。只有空闲的entry大于需要请求入队的指令时才会分配entry入队。入队时设置entry的状态位 allocated 为 true，其他状态位都为 false。
* STQ每次最多会有2个entry出队释放，通过输出指针deqPtrExt控制，每次指针右移一位或2位。STQ出队的触发信号是io.sbuffer(i).fire延后一拍的信号，因为sbuffer的写动作要用2拍完成，在sbuffer写完成之前entry不释放继续可以forward数据。

**从Store 的地址流水线写回结果**

* store 的地址从保留站发出来后会经过 [StoreUnit 流水线](#StoreAddrPipe)，通过lsq/lsq\_replenish总线接口在S1/S2把地址信息更新到store queue 中：

S1：DTLB(hit/miss), 虚拟地址VA 和物理地址PA

S2：mmio/pmp信息，是否是mmio地址空间操作等

**接收 store 的数据到STQ 的Datamodule**

* store 的数据是从与地址不同的保留站发出来的后经过StdExeUnit流水线，通过storeDataIn接口在S0/S1把数据写到对应的entry的datamodule里:

S0：给datamodule 发写请求

S1：写入数据到datamodule同时更新entry的 datavalid属性为True

接收 store 的mask到STQ 的Datamodule

store 的地址从保留站发出来之后会经过 StoreUnit 流水线，s0\_mask\_out在S0把地址中的mask信息更新到对应entry的datamodule里

**为 load 提供 forward 查询**

* load 需要查询 store queue 来找到在它之前的相同地址的与它最近的那个 store 的数据。
  + 查询总线(io.forwrd.sqIdx) 和StoreQueue的出栈指针比较，找出所有比load指令老的storeQueue中的entry。以flag相同或不同分为2种情况

(1)same flag-> older Store范围是 (tail, sqIdx)

(2)different flags-> older Store范围是(tail, VirtualLoadQueueSize) +(0, sqIdx)

* + 查询总线用va 和pa同时查询，如果发现物理地址匹配但是虚拟地址不匹配；或者虚拟地址匹配但是物理地址不匹配的情况就需要将那条 load 设置为 replayInst，等 load 到 ROB head 后replay。
  + 如果只发现一笔entry匹配且数据准备好，则直接forward
  + 如果只发现一笔entry匹配且数据没有准备好，就需要让保留站负责重发
  + 如果发现多笔匹配，则选择最老的一笔store forward
* 只有这样的 store 才能被 load forward：
  + allocated：这条 store 还在 store queue 内，还没有写到 sbuffer
  + datavalid：这条 store 的数据已经就绪
  + addrvalid：这条 store 已经完成了虚实地址转换，得到了物理地址
  + SSID (Store-Set-ID) 标记了之前load预测执行失败历史信息，如果当前load命中之前历史中的SSID，会等之前所有older的store都执行完；如果没有命中就只会等pa相同的older Store执行完成。

**MMIO store 的执行**

* MMIO 空间的 store 也只能等它到达 ROB 的 head 时才能执行，但是跟 load 稍微有些不同，store 到达 ROB 的 head 时，它不一定位于 store queue 的尾部，有可能有的 store 已经提交，但是还在 store queue 中没有写入到 sbuffer，需要等待这些 store 写到 sbuffer 之后，才能让这条 MMIO 的 store 去执行
* 利用一个状态机去控制 MMIO 的 store 执行
  + s\_idle：空闲状态，接收到 MMIO 的 store 请求后进入到 s\_req 状态
  + s\_req：给 uncache 通道发请求，请求被 uncache 通道接受后进入 s\_resp 状态，如果支持outstanding，则直接进入s\_wb
  + s\_resp：uncache 通道返回响应，接收后进入到 s\_wb 状态
  + s\_wb：将结果转化为内部信号，写回给 ROB，成功后进入到 s\_wait 状态
  + s\_wait：等待 ROB 将这条 store 指令提交，提交后重新回到 s\_idle 状态

**将被 ROB 提交后的 store 写给 sbuffer**

* 当指令被 ROB 提交之后，可以写到 sbuffer
* 先将这些 store 写到 dataBuffer 中，dataBuffer 是一个两项的缓冲区，用来处理从大项数 store queue 中的读出延迟
* 等待 sbuffer 可以写入再将 dataBuffer 中的 store 写入
* sbuffer 的写入需要两拍，还没有写入到 sbuffer 时，sbuffer 不能提供 forward 查询，所以得等完全写入到了 sbuffer 才去将 allocated 置为 false，一条 store 到这里才结束它的生命周期

**分支预测错误时**

* 当分支预测错误时，已经入队的 store 或者正在入队的 store 需要取消
* 分支预测错误信号到来时
* 第一拍的入队正常进行，那些被取消的正在入队的 store 不设置 allocated 为 true，但是正常分配了空间，同时需要产生取消的 mask；这一拍将已经在 store queue 中的需要取消的 store 的 allocated 置为 false
* 下一拍更新入队的指针，如果上一拍有正在入队的 store 被取消了，就把分配了的空间给释放，已经在 store queue 中的 store 被取消了也释放其空间，这些都是通过更新入队指针来完成的

5.3.6三级模块LoadQueue

5.3.6.1四级模块VirtualLoadQueue

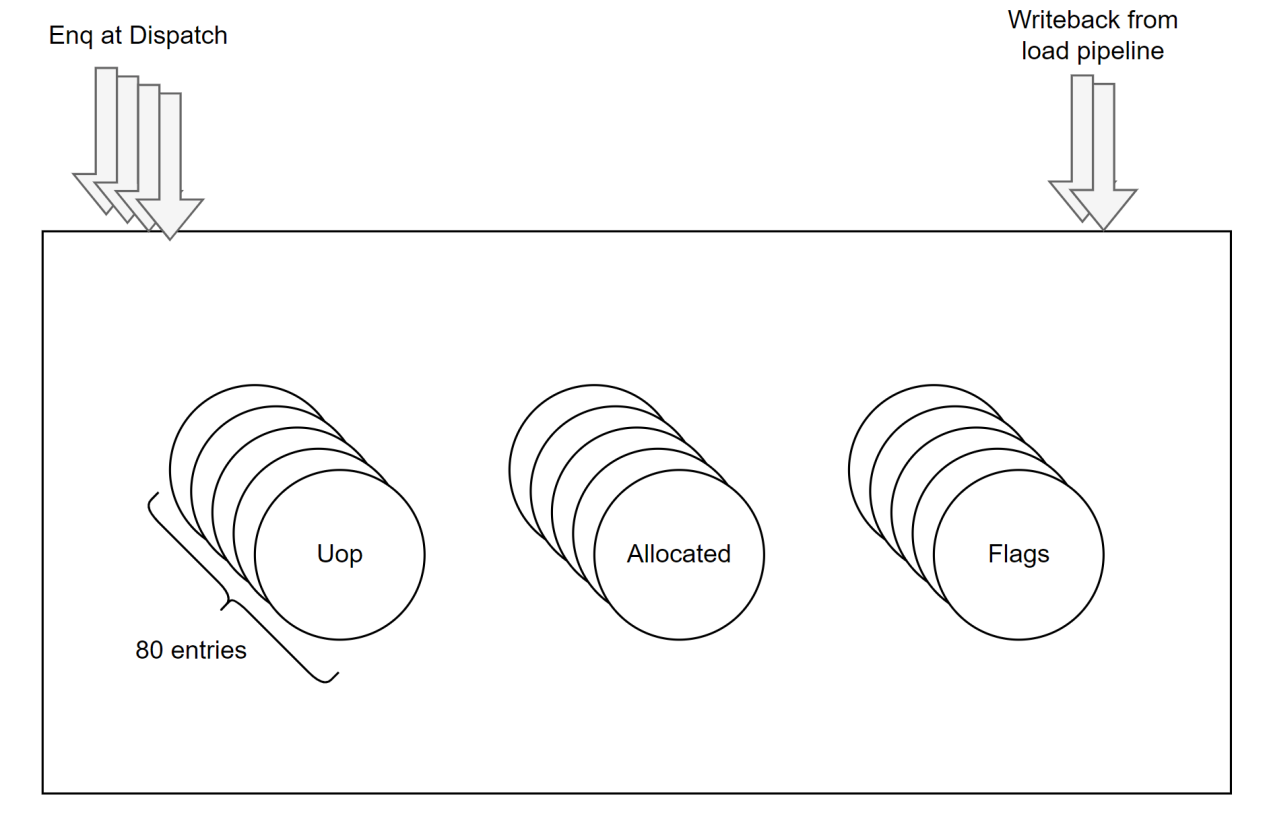
5.3.6.1.1功能

Virtualloadqueue是一个队列，用于存储所有load指令的MicroOp，维护load指令之间的顺序，类似于load指令的ROB，其主要功能为跟踪Load指令执行状态。

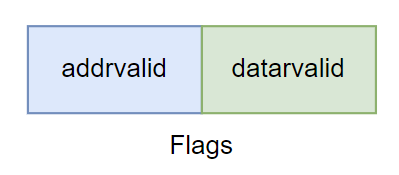
Virtualloadqueue对于每一个 entry 中的 load 指令都有若干状态位来标识这个 load 处于什么状态：

* allocated：该项是否分配了 load，用于确定 load 指令的生命周期。
* datavalid：load 的数据是否已经到达，可以用于 load 提交或者 store - load 违例检查。
* addrvalid：load的地址是否已经准备就绪，可以用于load相关的异常检查与tlb miss检查。

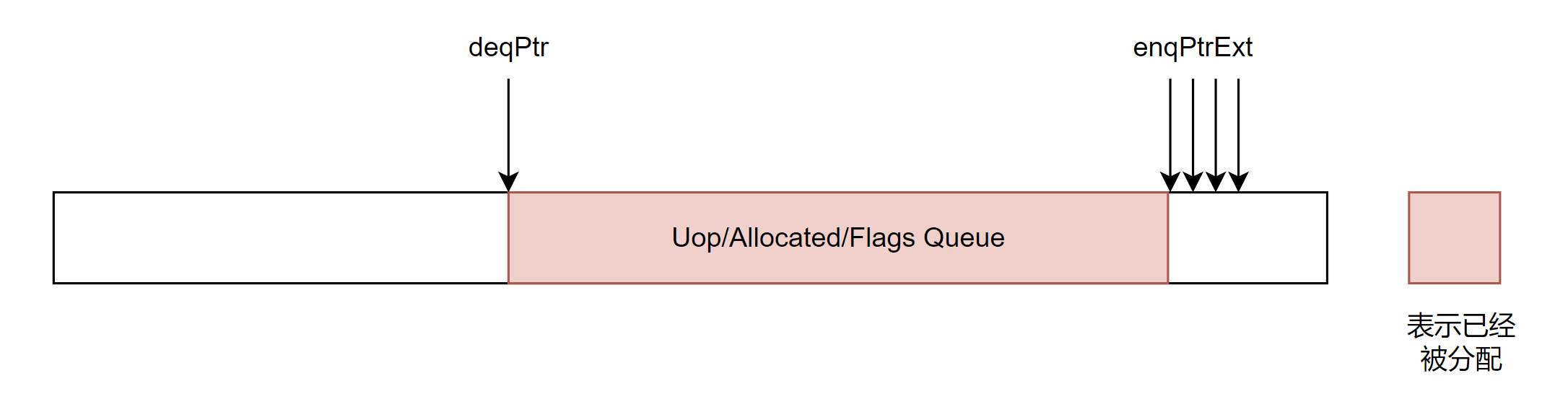
5.3.6.1.2整体框图



Flags组织方式如下



队列与队列指针的对应框图如下：

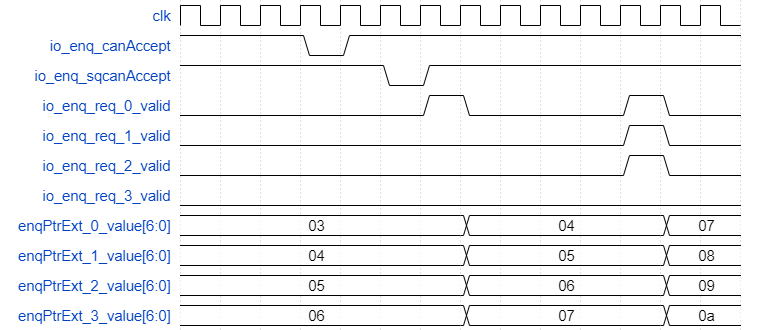


队列示意图

5.3.6.1.3接口时序

* 接收入队请求时序示例

当io\_enq\_canAccept与io\_enq\_sqcanAccept为高时，表示可以接收派遣指令。当io\_enq\_req\_\*\_valid为高时表示真实派遣指令到Virtualloadqueue，派遣指令的信息为rob的位置和Virtualloadqueue的位置。



* 流水线写回到Virtualloadqueue时序示例

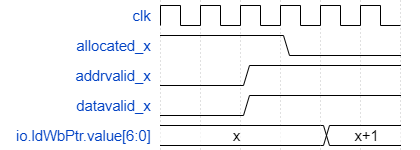
当io\_ldin\_\*\_valid为高时表示load流水线的s3写回lq，具体内容为io\_ldin\_\*\_bits\_\*。



其中allocated\_x, addrvalid\_x, datavalid\_x分别表示lq的第x项是否分配，地址是否准备好以及数据是否准备好。

* 出队时序示例

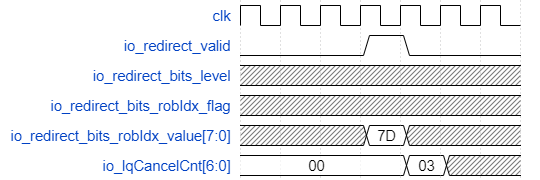
当队列中位于队头的表项datavalid与addrvalid都为高时，表项可以出队。



其中x表示出队的lq表项在环形队列中的位置，每写回一个表项队尾指针就+1。

* 重定向时序示例

当io.redirect.valid为高时，后端产生重定向，Virtual Load Queue需要取消相应指令，恢复指针，io\_lqCancelCnt表示取消的表项个数。



5.3.6.1.4关键电路

* 入队：
  + 入队时机：在指令的 dispatch 阶段，会将 load 指令从 dispatch queue 发送到 load queue，Virtual Load Queue用于保存指令的信息。
  + 流水线写回时机：load 从 iq 发出后，经过 load 流水线，到达流水线的 s3 时，将这条 load 的执行信息反馈给 load queue。
  + 流水线写回的信息：包括dcache 是否命中，load 是否正常拿到了数据（包括 dcache miss 但是可以从 sbuffer 和 store queue forward 完整数据的情况），tlb是否miss，是否需要重发load。load 是否发生了异常，load 是否是 MMIO 空间的，是否产生写后读违例、读后读违例，是否出现dcache的bank冲突。
  + Virtual Load Queue中对MMIO的处理：对于 MMIO 的 load，在接收完流水线中的信息后，它的状态为：allocated，addrvalid，datavalid，UncachedBuffer模块会进行MMIO的读操作。
* 出队：
  + 出队时机：当被分配的entries（allocated为高）到达队头，同时datavalid与addrvalid都为1时，表示可以出队。

#### 5.3.6.2四级模块LoadQueueRAR

5.3.6.2.1功能

多核环境下会出现load to load违例：单核环境下相同地址的load乱序执行本来是不关心的，但是如果两个load之间有另外一个核做了相同地址的store，并且本身这个核的两个load做了乱序调度，就有可能导致新的load没有看到store更新的结果，但是旧的load看到了，出现了顺序错误。

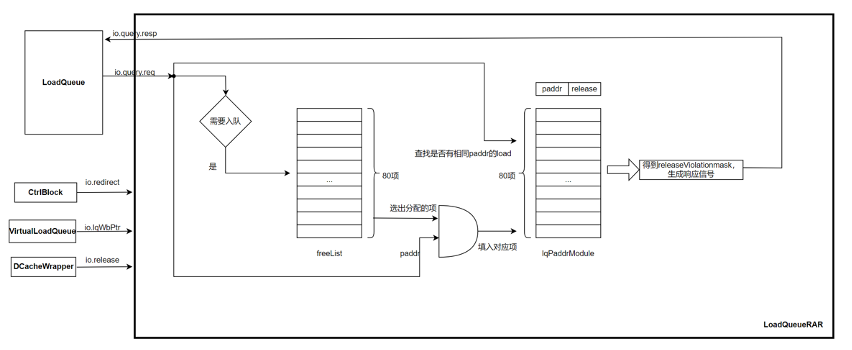
多核环境下的load-load违例有一个特征，当前DCache一定会收到L2 cache发来的Probe请求，使得DCache主动释放掉这个数据副本，这时DCache会通知load queue，将相同地址的load queue中已经完成访存的项做一个release标记。后续发往流水线的load指令会查询load queue中在它之后相同地址的load指令，如果存在release标记，就发生了load-load违例。

LQRAR用于保存已经完成的load指令的用于load to load违例检测的信息。当load指令处于load流水线第三级时，查询并分配空闲项将信息保存入LQRAR，在流水线第四级时得到load to load违例检查的结果 ，如果出现违例则需要刷新流水线，给RedirectGenerator部件发送重定向请求, 冲刷违例的load之后的所有指令。

LoadQueueRAR中需要标记以下信息：

* Allocated：表示entry是否有效。
* Uop：MicroOp相关信息。
* Paddr：进入LoadQueueRAR指令的物理地址。
* Released：表示该指令所访问的cacheline是否被release，多核环境下 L1 cache会接收到L2cache的probe请求。

5.3.6.2.2整体框图

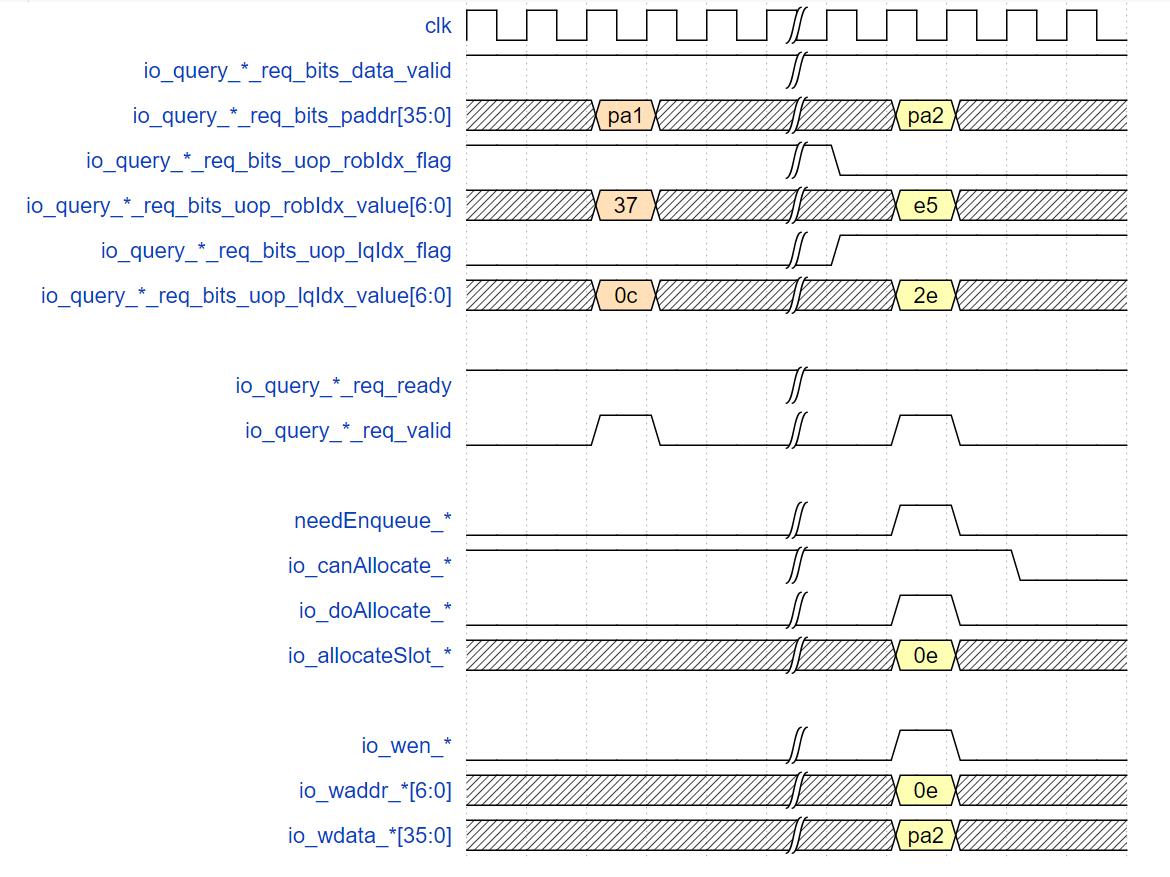


5.3.6.2.3接口列表

详见IO卷

5.3.6.2.4接口时序

* Query enqueue时序示例：



当io\_query\_\*\_req\_valid和io\_query\_\*\_req\_ready都为高时，表示握手成功，needEnqueue和io\_canAllocate\_\*都为高时，将io\_doAllocate\_\*置为高，表示query需要需要入队且FreeList可以分配， io\_allocateSlot\_\*为接收query入队的entry， 写入entry的信息为io\_w\*。

* load-load违例检查时序示例：

A screenshot of a computer screen

Description automatically generated

当io\_query\_\*\_req\_valid和io\_query\_\*\_req\_ready都为高时，表示握手成功，LoadQueueRAR接收ld-ld违例查询请求，当拍得到mask结果，在下一拍将io\_query\_\*\_resp\_valid置为高，给出响应。

上图中第3拍接收到第一个违例查询请求，在第4拍得到违例查询请求的响应。请求的信息为io\_query\_\*\_req\_bits\_\*，响应的信息为io\_query\_\*\_resp\_bits\_\*。当io\_query\_\*\_resp\_valid和io\_query\_\*\_resp\_bits\_rep\_frm\_fetch都为高时，表示发生ld-ld违例，query需要从取指重发。

5.3.6.2.5关键电路

* 入队:

当query到达load流水线的s2时，判断是否满足入队条件，如果在当前load指令之前有未完成的load指令,且当前指令没有被flush时，当前load可以入队。

在freelist中得到可以分配的entry以及index。

在PaddrModule中保存入队信息，包含query的物理地址，分配entry的index。

* load-load违例检查：

当 load 到达流水线的 s2 时，会检查RAR队列中是否存在与当前load指令物理地址相同且比当前指令年轻的load指令，如果这些 load 已经拿到了数据，并且被标记了release，说明发生 load - load 违例，被标记release的指令需要从取指重发。

一共分两拍：

* + 第一拍进行条件匹配，得到mask。
  + 第二拍生成是否发生违例的响应信号。

这种违例出现的频率比较低，所以是等待指令到达 ROB 的 head 才处理，类似于异常处理。

* Release条件：

LoadQueueRAR中的load指令被标记为release有两种情况：

* + missQueue模块的replace\_req在mainpipe流水线的s3栈发起release释放dcache块，release信号在下一拍进入loadqueue。
  + probeQueue模块的probe\_req在mainpipe流水线的s3栈发起release释放dcache块，release信号在下一拍进入loadqueue。

5.3.6.2.5五级模块LqPAddrModule

LqPAddrModule用于保存进入RAR队列的query的信息，包含物理地址，release标志位。

* + - * 1. FreeList

FreeList作为一个队列，可以实时监测RAR队列状态，判断是否可以为query分配entry，以及分配哪一项。

5.3.6.3四级模块**LoadQueueRAW**

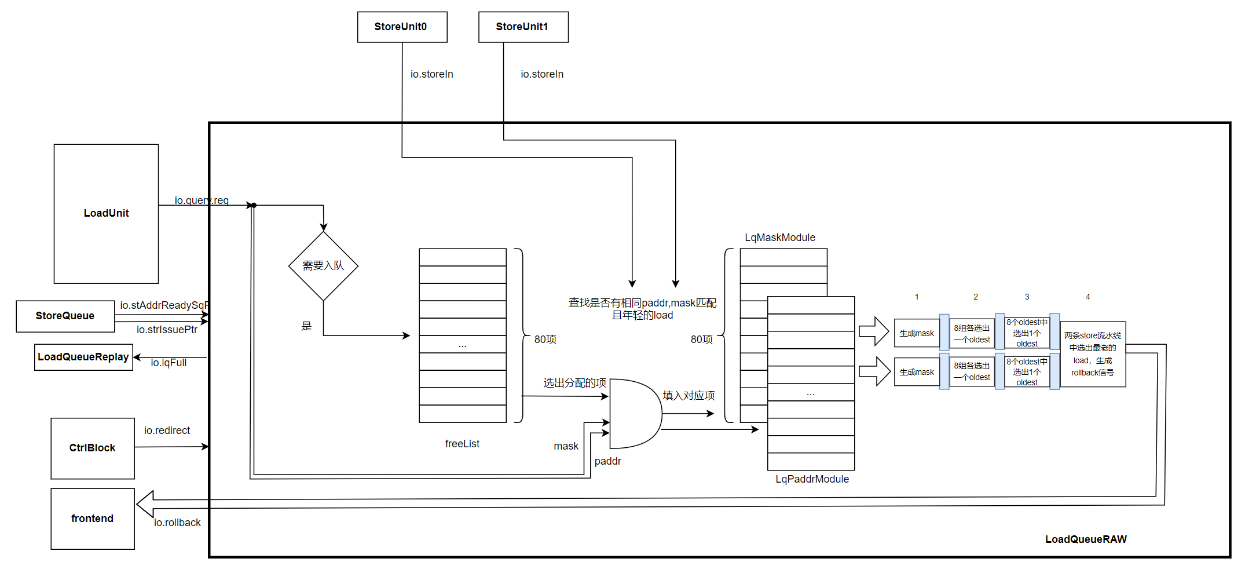
5.3.6.3.1功能

LoadQueueRAW是用于处理store to load违例的。由于load和store在流水线中都是乱序执行，会经常出现load越过了更老的相同地址的store，即这条load本应该从store 通过前递得到数据，但是由于store地址或者数据没有准备好，导致这条load没有前递到store的数据就已经提交，后续使用这条load结果的指令也都发生了错误，于是产生store to load forwarding违例。

当store address通过STA保留站发射出来进入store流水线时，会去查询LQRAW中在这条store后面的所有已经完成访存的相同地址的load，以及load流水线中正在进行的在该条store之后的相同地址的load，一旦发现有，就发生了store to load forwarding违例，可能有多个load发生了违例，需要找到离store最近的load，也就是最老的违例的load。

store to load forwarding违例较为频繁，所以一旦检测到store to load forwarding违例就需要立刻恢复。发生store to load forwarding违例后，找到最老的违例的load，给RedirectGenerator部件发送重定向请求，冲刷违例的load及之后的所有指令，将PC重定向到这条违例的load，这个冲刷过程类似于分支预测失败的冲刷过程，MemBlock中需要取消相关的流水线中的load与 store指令，取消相关的在load queue和store queue中的指令。

5.3.6.3.2整体框图

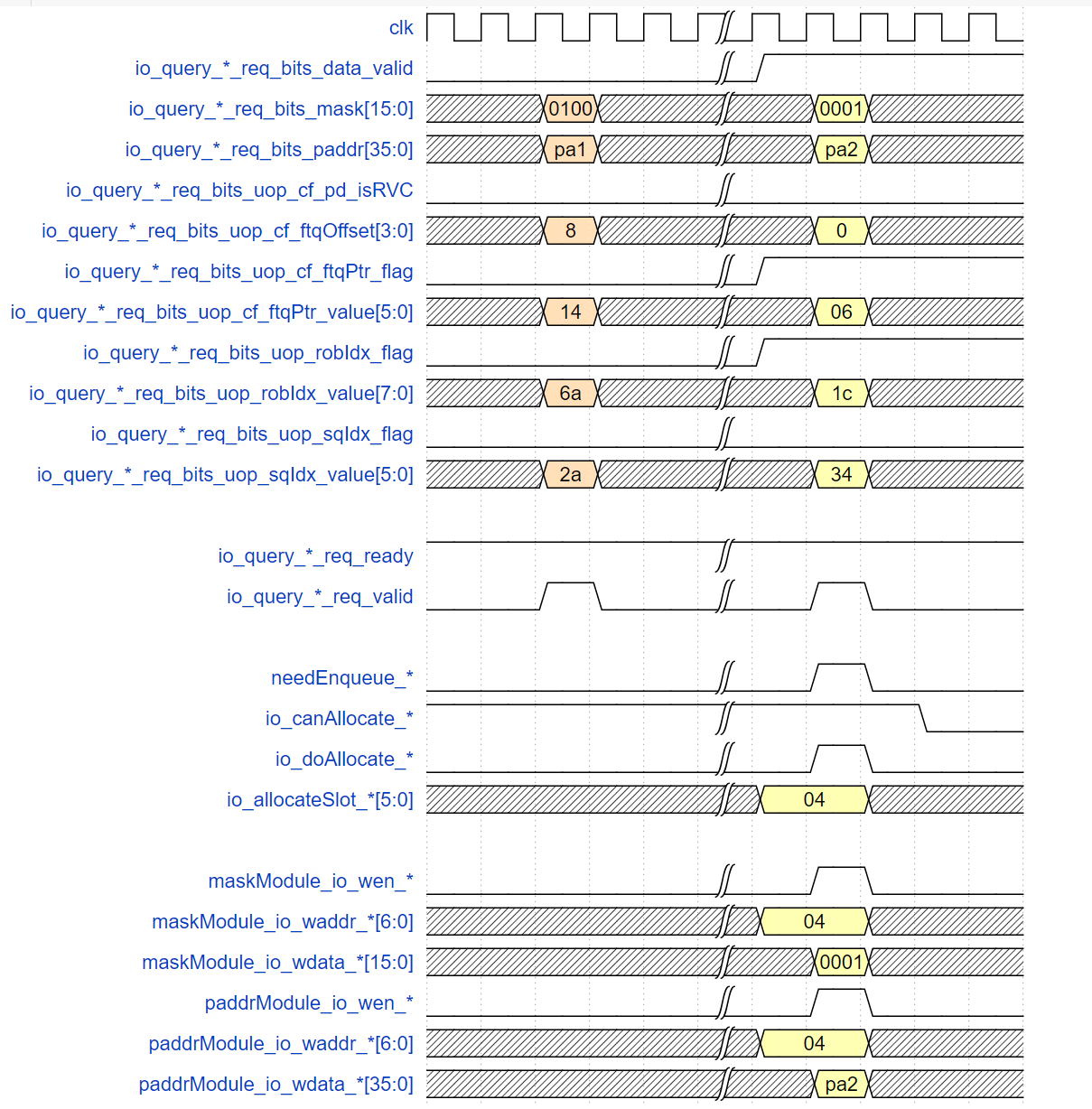


5.3.6.3.3借口列表

详见IO卷

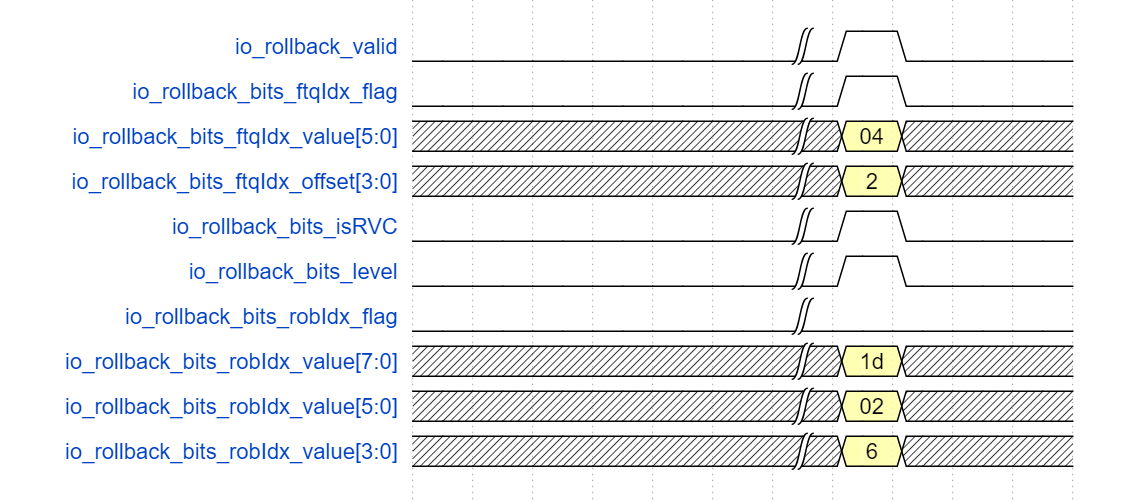
5.3.6.3.4接口时序

* Query enqueue时序示例：



当io\_query\_\*\_req\_valid和io\_query\_\*\_req\_ready都为高时，表示握手成功，needEnqueue和io\_canAllocate\_\*都为高时，将io\_doAllocate\_\*置为高，表示query需要入队且FreeList可以分配， io\_allocateSlot\_\*为接收query入队的entry， maskModule\_io\_w\*表示写入LqMaskModule的信息，paddrModule\_io\_w\*表示写入LqPAddrModule的信息。

* store-load违例检查时序示例**：**



当io\_rollback\_valid为高时，表示发生了store-load违例，违例的信息为io\_rollback\_bits\_\*。

5.3.6.3.5关键电路

* Query入队：

当query到达load流水线的s2时，判断是否满足入队条件，如果在当前load指令之前有地址未准备好的store指令，且当前指令没有被flush时，当前load可以入队。

在freelist中得到可以分配的entry以及index。

在PaddrModule中将入队query的物理地址保存到对应的entry。

在maskModule中将入队query的mask保存到对应的entry。

* store - load 违例检查：

store 到达 store 流水线的 s1 时会进行 store - load 检查，此时 store 需要与 load queue 中已经完成访存的 load，以及 load 流水线中s1和 s2 阶段正在访存的 load 作比较，这些 load 可能没有 forward 到 store 的数据。如果检查时发现 load 和 store 访问的物理地址有重叠的地方，且load比store年轻，就发生了违例，需要找到最老的 load，redo 这条 load 以及之后的所有指令（重新取指执行），在load流水线的s3阶段得到store-load违例检查的结果。

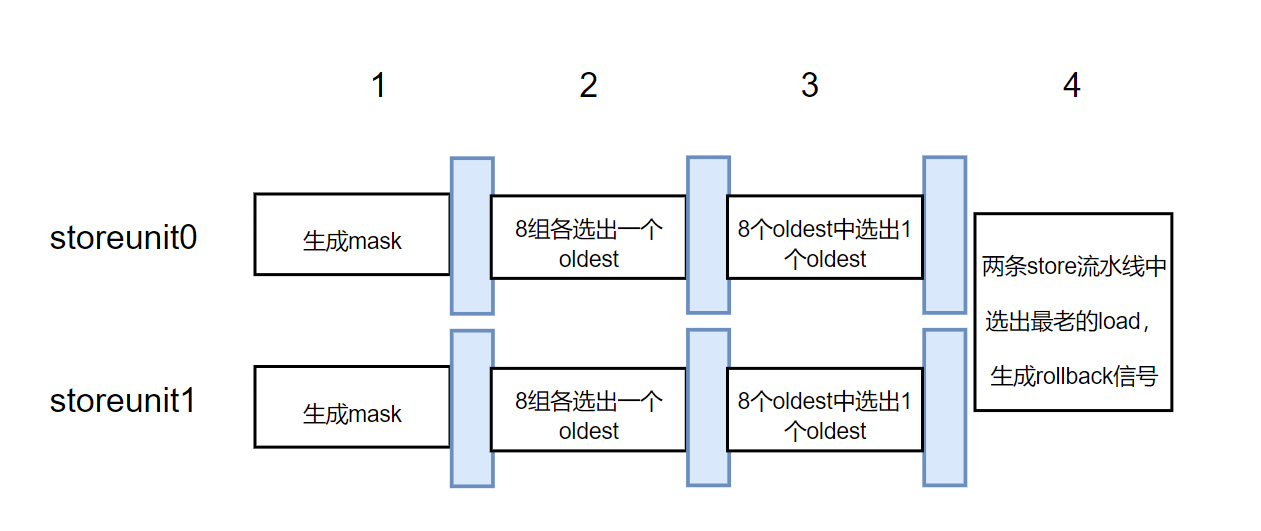
一共分四拍

第一拍进行物理地址匹配，条件匹配，得到 mask，匹配的是那些在这条 store 之后的新的 load，如果它们已经拿到了数据（datavalid）或者 dcache miss 了，正在等待 refill（miss），就一定没有 forward 到这个 store 的数据

第二拍store流水线中的store根据mask在loadqueueRAW里面找到所有匹配的load，loadqueueRAW一共有64项，将这64项平分为八组，从每组的8项里面各选出一个oldest，最多可能得到8个oldest。

第三拍从8个oldest里面选出一个最老的oldest。

第四拍如果两条store流水线中的store都发生了store-to-load违例，从两条store流水线各自在load queue匹配的oldest load中选出一个更老的oldest，产生回滚请求发给redirect。



这种违例的出现频率较高，所以是一旦出现就立刻回滚，类似于分支预测错误

5.3.6.3.6LqPAddrModule

5.3.6.3.7LqMaskModule

5.3.6.3.8FreeList

5.3.6.4四级模块LoadQueueReplay

5.3.6.4.1AgeDetector

5.3.6.4.2LqVAddrModule

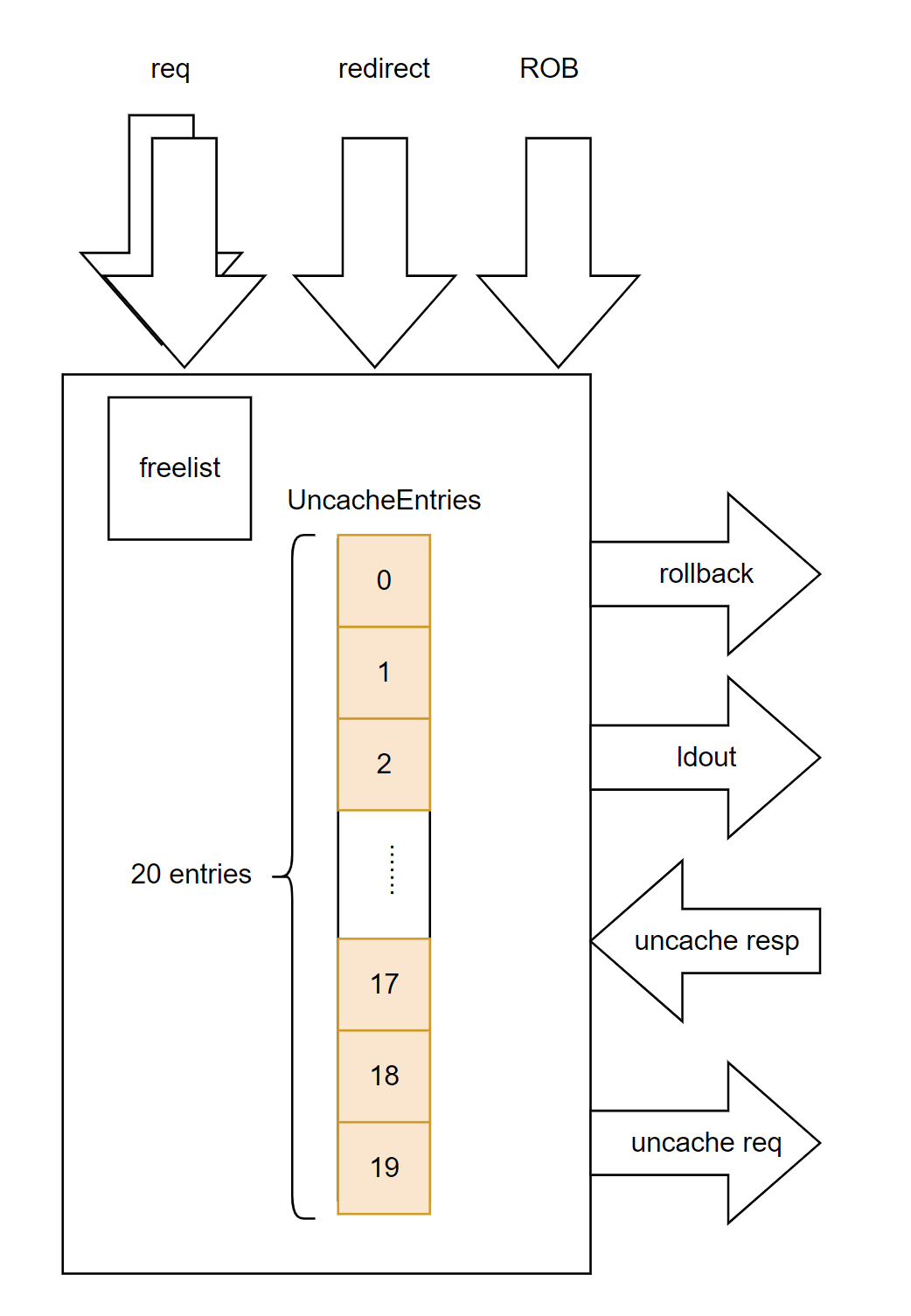
5.3.6.4.3FreeList

5.3.6.5四级模块UncacheBuffer

5.3.6.5.1功能

用于控制MMIO load的执行，本质上是一个基于freelist的队列，用于存储所有uncache的load请求，当缓存的uncache请求到达rob队头时，由内部状态机发起mmio读操作，取到uncache数据之后通过LDU0的写回端口写回数据。

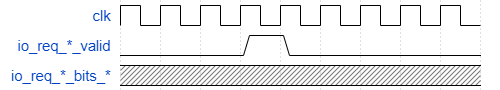
5.3.6.5.2整体框图



5.3.6.5.3接口时序

* 入队时序示例

当io\_req\_\*\_valid为高时，表示有一个入队请求，具体的入队信息为io\_req\_\*\_bits\_\*。



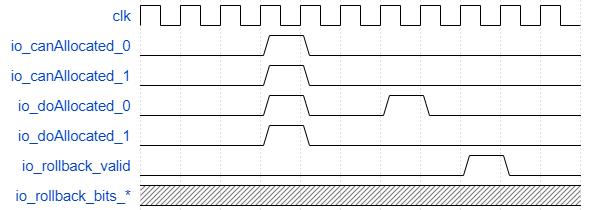
* 执行uncache请求时序示例

io\_uncache\_req\_valid为高时，表示uncachebuffer发起了一个uncache读请求，读请求的信息为io\_uncache\_req\_bits\_\*。io\_uncache\_resp\_valid为高时表示有一个uncache读请求的响应信息，响应的具体信息为io\_uncache\_resp\_bits\_\*。



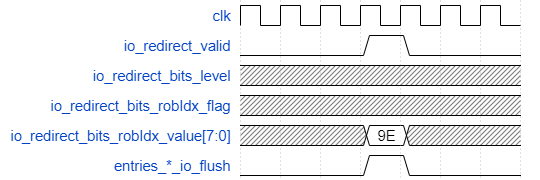
* rollback时序示例

当freelist分配失败时需要进行rollback，io\_rollback\_valid为高时表示需要进行rollback，rollback的具体信息为io\_rollback\_bits\_\*。



* 重定向时序示例

当io\_redirect\_valid为高时表示发生了一个重定向，UncacheBuffer需要取消相应的表项，释放相应的freelist。



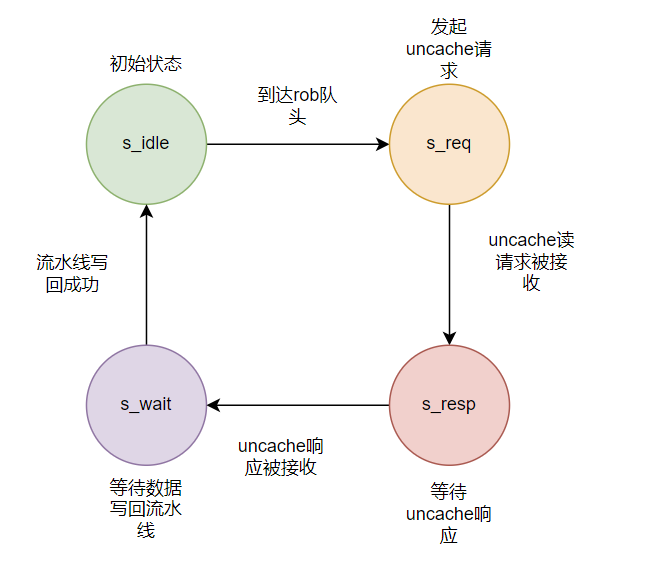
其中entries\_\*\_io\_flush为需要进行冲刷的表项发出的表项被冲刷信号。

5.3.6.5.4关键电路

* 入队时机：当Load请求到达LDU的s3阶段并且是一个MMIO请求时，会进入UncacheBuffer。
* 发起uncache的时机：当UncacheBufferEntry内存储的Load请求到达ROB队头时，接收到ROB发出的信号之后，开始MMIO的执行。
* rollback的时机：freelist没有空闲表现导致MMIO Load进入UncacheBuffer失败时需要进行rollback，此时需要根据robidx选择不能入队的MMIO中最老的指令进行rollback。
* redirect的响应：发生redirect时，需要对相应的表项进行冲刷，释放相应的freelist占用。
* 出队时机：UncacheEntry内部的MMIO请求取得的数据写回了流水线。
* uncache请求的执行过程：当开始执行MMIO Load时，UncacheBuffer会通过io\_uncache\_req\_\*端口向uncache模块发出uncache请求，uncache模块通过总线取得MMIO数据之后通过io\_uncache\_resp\_\*向UncacheBuffer返回数据，之后UncacheBuffer通过io\_ldout将读数据通过LDU\_0的写回端口写回后端。

5.3.6.5.4.1 UncacheBufferEntry

管理每个Uncahce请求，负责发起Uncache，写回Uncache数据。每个Entry内维护一个用于发起Uncache请求的状态机，状态机的状态转换图如下：

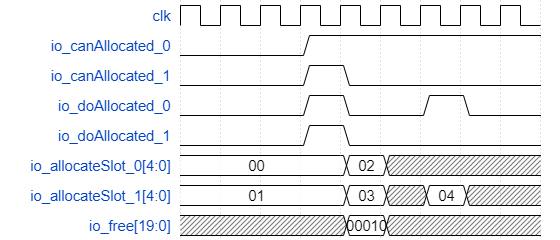


* s\_idl:该项还未发起一个MMIO请求。
* s\_req:向uncache模块发起MMIO请求，等待请求被接收。
* s\_resp:等待uncache模块的MMIO响应。
* s\_wait:等待将MMIO结果写回流水线。

FreeList

管理每个UncacheBufferEntry是否空闲。

freelist 分配与释放时序示例



其中io\_free[19:0]的每一个bit表示对应index的entries是否需要被释放。

5.3.6.5.5 四级模块LqExceptionBuffer

5.3.6.5.5.1功能

用于跟踪load指令产生的异常情况，根据robIdx选择离开LDU的指令中最老的发生异常的指令的虚地址输出。内部有两级流水，第一级流水缓存LDU\_\*的s3阶段输出的信息，第二个周期根据robIdx选取最老的发生异常的指令，输出其虚地址。

重定向时根据buffer内缓存的指令robIdx进行是否需要刷掉的判断。

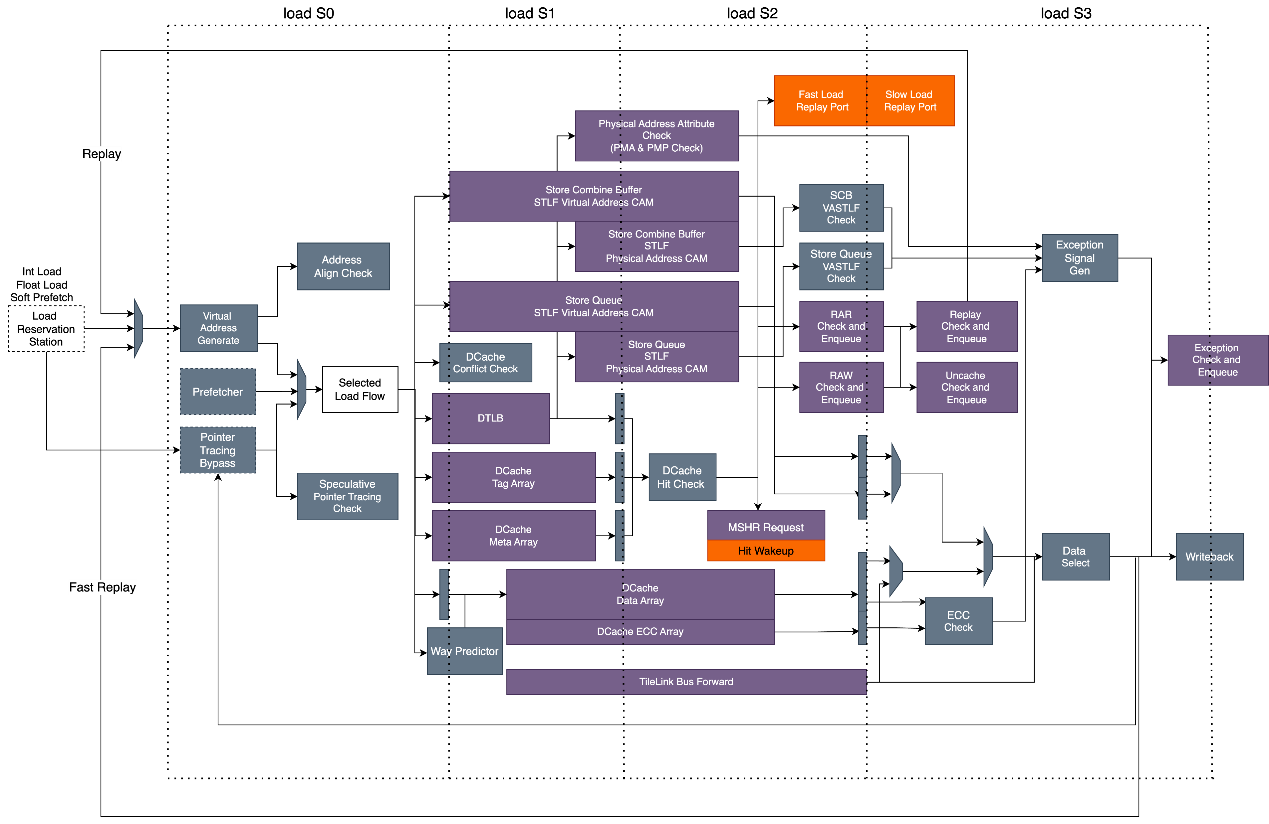
5.4二级模块LoadUnit

5.4.1功能

load指令流水线，接受load发射队列发送的load指令，在流水线中处理完成后将结果写回loadqueue和ROB，用于指令提交以及唤醒后续依赖本条指令的其他指令。同时，loadunit需要给发射队列、load/store queue反馈一些必要的信息。这些信息绝大多数直接体现在接口列表中，根据输入和输出，包括：

* 输入：
  + 接受来自ctrlblock的redirect信号，根据重定向信号刷新load流水线。
  + Load\_s0阶段接受来自exublock（发射队列）发射的load指令信息，根据指令的类型、是否命中访存违例预测器需要阻塞、指令的源操作数和立即数等信息对load指令进行处理。根据指令的类型（例如ld/lw/lb）等，可以判断需要加载的size，并根据size向dcache发送查询请求，并拼接数据等；根据是否命中访存违例预测器，判断是否需要等待storequeue forward的地址计算好，确认不会出现访存违例再提交；根据指令的源操作数和立即数，可以计算指令的虚拟地址，并根据虚拟地址向tlb查询物理地址、向dcache索引index。
  + Load\_s1阶段接受来自dtlb的回复，包括根据load指令虚拟地址查询得到的物理地址、tlb是否缺失、以及该地址是否发生page fault、access fault等信息。当TLB发生缺失时，需要交付给loadqueue replay等待唤醒重发，因此需要传递TLB返回的MSHRid到replay queue，当这个MSHRid从L2 TLB被refill时，可以根据refill的MSHRid精确唤醒。同时，对于TLB mshr已满或虽然访问TLB缺失，但正在被回填的情况，loadunit需要将TLB反馈的立即重发信号传递给loadqueue replay，replay queue无需等待即可调度重发。
  + Load\_s2阶段接受来自pmp检查的回复，包括这条load指令的物理地址是否发生access fault，或者属于MMIO空间。如果物理地址发生access fault，需要上报异常；如果属于MMIO空间，则将这条指令提交给loadqueue中的uncache buffer进行处理。
  + Load\_s2、load\_s3接受来自dcache的回复，包括dcache的返回数据、dcache是否缺失、是否发生ecc error、缺失时的MSHR id、是否发生bank conflict、是否缺失且MSHR已满等信息。同时，对于dcache miss，但正在由L2 Cache回填的情况，需要接受l2 cache forward的信息。
  + 对于dcache命中的情况，会在load\_s3将dcache查询得到的数据返回loadunit，在loadunit中可以根据指令的不同类型对数据进行拼接。对于dcache未命中的情况，会根据未命中的细分原因（例如发生bank conflict、MSHR已满等），将这些原因同时传递进load queue replay中，对其进行重发。如果dcache虽然缺失，但MSHR未满，dcache会将分配的MSHRid告知loadunit。Loadunit需要把MSHRid写入replay queue，等待精确唤醒。另外，如果dcache发生ecc error，loadunit需要将这条指令写回，并上报异常。
  + Load\_s2、load\_s3接受来自sbuffer和storequeue关于前递的回复。当loadunit中正在处理的指令在sbuffer或storequeue中与某项匹配成功时，可以前递sbuffer或storequeue中的数据作为load的结果，数据和匹配情况会在load\_s2返回。如果storequeue中出现前递的数据，或前递需要等待的某个地址没有准备好，会把这些信息同时写回loadqueue replay进行处理。当sbuffer和storequeue前递的数据均有效时，采用storequeue前递的数据。另外，在load\_s3会返回虚实地址匹配失败的信号，这种情况表示虽然load和store的虚拟地址相同，但物理地址不同，需要反馈从取指重发。
  + Load\_s3接受loadqueue中uncache buffer的请求，将uncache的数据根据地址低位进行拼接，并写回。
  + Load\_s2和load\_s3需要接受loadqueueRAW和loadqueueRAR的请求。在load\_s2会返回RAW和RAR queue是否已满的信息，需要根据这些信息交由replay queue重发。在load\_s3会返回RAR的检查结果，如果出现load-load违例，需要从取指重发。
  + Load\_s0阶段需要接受预取器的预取请求，和同样在load\_s0的其他请求根据优先级进行仲裁。预取器包括SMS与其妻，以及stream&stride的集成预取器。
  + Load\_s0可能接受load to load forward请求。目前不开启。
* 输出：
* 在loadunit中，根据流水线阶段进行描述
* Load\_s0：

5.4.2整体框图



5.4.3接口列表

详见IO卷

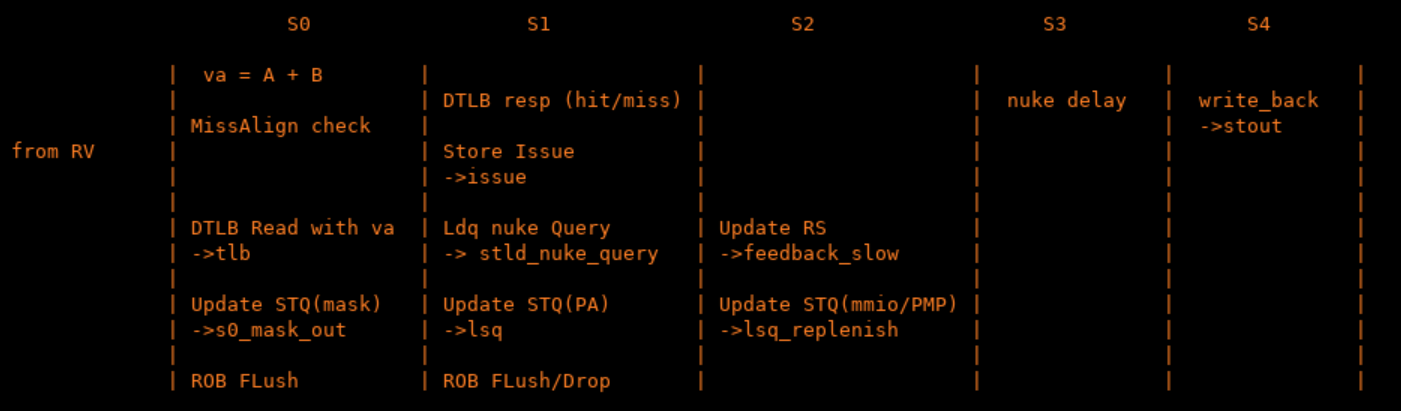
5.4.4接口时序

5.4.5关键电路

5.5二级模块StoreUnit

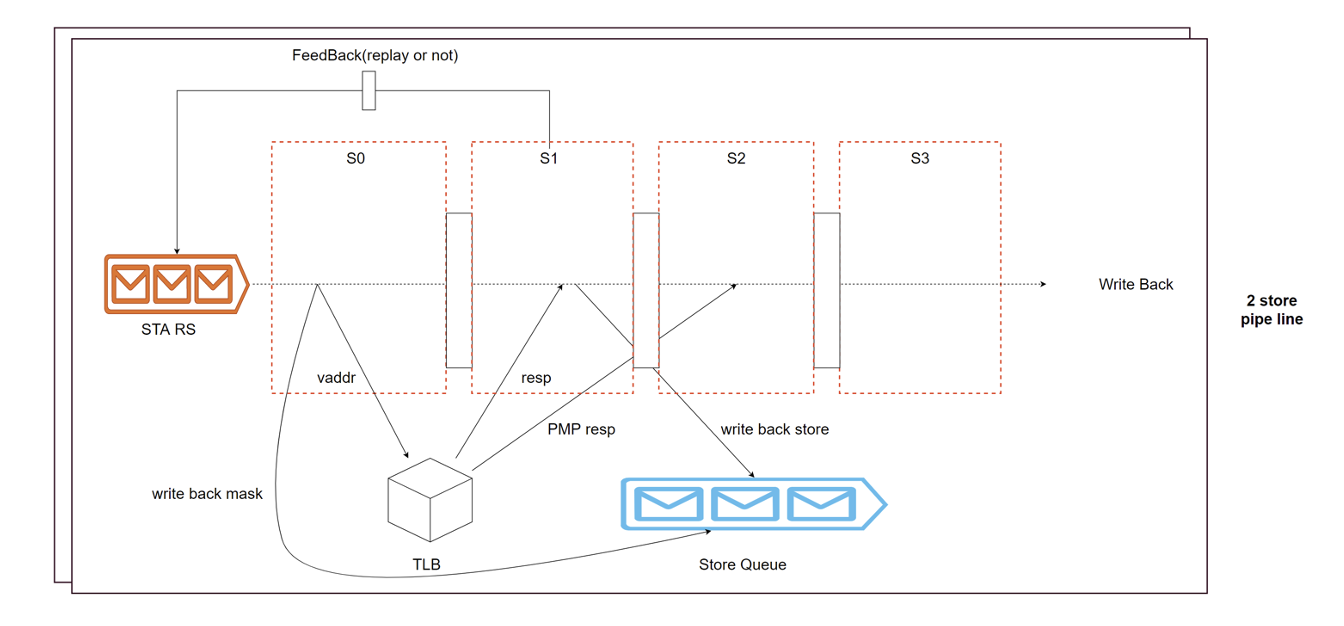
5.5.1功能

Store 指令地址流水线分为S0/S1/S2/S3/S4 五级, 如下图StoreAddrPipe所示。接受 store 地址保留站发来的请求，处理完成之后需要给结果总线发响应，处理过程中需要给保留站反馈信息，给StoreQueue反馈信息，最后写回, 如果中间出现异常则从保留站重新发射。



* S0:
* 计算VA地址
* 地址Miss-Align 检查更新到uop.cf.exceptionVec(storeAddrMisaligned)
* 发出DTLB读请求到总线tlb
* 更新指令的Mask信息到总线s0\_mask\_out -> StoreQueue
* 预测执行的Store有可能被ROB flush
* S1:
  + 更新DTLB查询结果到总线lsq -> storeQueue
  + 向LoadQueue发出store-load违例检查请求到总线stld\_nuke\_query
  + 如果DTLB hit，广播Store issue信息到总线issue
  + ROB flush或停止流水(TLB miss)
* S2:
  + mmio/PMP检查并更新总线lsq\_replenish -> storeQueue
  + 更新DTLB结果到总线feedback\_slow->RV
* S3: 违例检查延时
* S4：发起Write-Back请求到总线stout

5.5.2整体框图



5.5.3接口列表

详见IO卷

5.5.4接口时序

A screen shot of a computer

Description automatically generated

5.5.5关键电路

5.5.6三级模块SQDataModule

5.5.7三级模块SQAddrModule

5.5.8三级模块DatamoduleResultBuffer

5.6二级模块StdExeUnit

5.6.1功能

5.6.2整体框图

5.6.3接口列表

5.6.4接口时序

5.6.5关键电路

5.7二级模块SBuffer

5.7.1功能

5.7.2整体框图

5.7.3接口列表

5.7.4接口时序

5.7.5关键电路

5.7.6三级模块SbufferData

5.8异常时间情况汇总

5.8.1后端重定向

5.8.2Load指令异常行为

5.8.3Store指令异常行为

5.8.4原子指令异常行为

6. PPA优化设计

*内容包含:*

1. *Power*

*描述功耗设计目标*

*详细描述设计的功耗预估情况*

1. *Performance*

*详细描述设计的性能目标数据*

*详细描述性能的预估情况*

1. *Area*

*详细描述设计的面积目标数据*

*详细描述面积的预估情况*

*4.为优化PPA做的一些关键设计点（例如为了时序收敛做的一些面积/功耗/性能上的折中）*

7. 验证关注点

*从设计角度列举需要验证人员特别关注的测试点。*

*不涉及填“NA”*

8. Floorplan 建议

*芯片的floorplan考虑，依据数据流向，IO排布，模块大小进行芯片布局摆放设计*

*不涉及填“NA”*

9. 遗留问题

*需要跟踪的遗留问题*

*不涉及填“NA”*