|  |  |
| --- | --- |
| 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CoupledL2模块AS

|  |  |
| --- | --- |
| 编 写： | 陈熙、蔡洛姗、马月骁 |
| 校 对： |  |
| 审 核： |  |
| 批 准： |  |

昆明湖项目

2024年1月18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 | 新建 | 陈熙 | 2023.12.15 |
| 2 | V0.9 | A | 初稿 | 陈熙、蔡洛姗、马月骁 | 2024.1.18 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

\*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](#_Toc156507730)

[1.1 文档介绍 1](#_Toc156507731)

[1.2 参考文档 1](#_Toc156507732)

[1.3 术语说明 1](#_Toc156507733)

[1.4 技术背景 2](#_Toc156507734)

[2 设计规格 2](#_Toc156507735)

[3 功能描述 2](#_Toc156507736)

[3.1 功能概述 3](#_Toc156507737)

[3.2 分特性详细描述： 3](#_Toc156507738)

[***3.2.1*** ***特性1：接收和发送TileLink总线请求及响应*** 3](#_Toc156507739)

[***3.2.2*** ***特性2：采用TileLink一致性协议*** 3](#_Toc156507740)

[***3.2.3*** ***特性3：采用目录记录缓存行信息*** 4](#_Toc156507741)

[***3.2.4*** ***特性4：采用主流水线架构*** 4](#_Toc156507742)

[3.2.5 ***特性5：在收到重填数据时再进行替换路的选择和释放*** 7](#_Toc156507743)

[***3.2.6*** ***特性6：支持同Set请求的并行访问*** 7](#_Toc156507744)

[***3.2.7*** ***特性7：利用Hint信号进行load提前唤醒*** 7](#_Toc156507745)

[***3.2.8*** ***特性9：进行硬件预取*** 8](#_Toc156507746)

[***3.2.9*** ***特性10：处理Cache别名问题*** 8](#_Toc156507747)

[3.2.10 ***特性11：支持请求融合*** 9](#_Toc156507748)

[4 总体设计 10](#_Toc156507749)

[4.1 整体框图 10](#_Toc156507750)

[4.2 接口列表 11](#_Toc156507751)

[4.3 接口时序 11](#_Toc156507752)

[4.4 时钟复位 11](#_Toc156507753)

[4.5 寄存器配置 12](#_Toc156507754)

[4.6 补充说明 12](#_Toc156507755)

[5 模块设计 12](#_Toc156507756)

[5.1 Slice 13](#_Toc156507757)

[5.1.1 功能 13](#_Toc156507758)

[5.1.2 整体框图 13](#_Toc156507759)

[5.1.3 接口列表 14](#_Toc156507760)

[5.1.4 接口时序 14](#_Toc156507761)

[5.1.5 关键电路 14](#_Toc156507762)

[5.1.6 三级模块设计 14](#_Toc156507763)

[5.2 通道控制器 14](#_Toc156507764)

[5.2.1 SinkA 14](#_Toc156507765)

[5.2.2 SinkB 15](#_Toc156507766)

[5.2.3 SinkC 16](#_Toc156507767)

[5.2.4 RefillUnit 18](#_Toc156507768)

[5.2.5 AcquireUnit 19](#_Toc156507769)

[5.2.6 SourceB 19](#_Toc156507770)

[5.2.7 SourceC 20](#_Toc156507771)

[5.2.8 GrantBuffer 22](#_Toc156507772)

[5.3 RequestArb（ReqArb） 24](#_Toc156507773)

[5.3.1 功能 24](#_Toc156507774)

[5.3.2 整体框图 25](#_Toc156507775)

[5.3.3 接口列表 26](#_Toc156507776)

[5.3.4 接口时序 26](#_Toc156507777)

[5.3.5 关键电路 26](#_Toc156507778)

[5.3.6 三级模块设计 27](#_Toc156507779)

[5.4 Directory 27](#_Toc156507780)

[5.4.1 功能 27](#_Toc156507781)

[5.4.2 整体框图 27](#_Toc156507782)

[5.4.3 接口列表 28](#_Toc156507783)

[5.4.4 接口时序 28](#_Toc156507784)

[5.5 MainPipe 28](#_Toc156507785)

[5.5.1 功能 28](#_Toc156507786)

[5.5.2 整体框图 30](#_Toc156507787)

[5.5.3 接口列表 30](#_Toc156507788)

[5.5.4 接口时序 30](#_Toc156507789)

[5.5.5 关键电路 30](#_Toc156507790)

[5.5.6 三级模块设计 30](#_Toc156507791)

[5.6 MSHRCtl 30](#_Toc156507792)

[5.6.1 功能 30](#_Toc156507793)

[5.6.2 整体框图 31](#_Toc156507794)

[5.6.3 接口列表 31](#_Toc156507795)

[5.6.4 接口时序 31](#_Toc156507796)

[5.6.5 关键电路 31](#_Toc156507797)

[5.6.6 三级模块设计 32](#_Toc156507798)

[5.7 MSHR 32](#_Toc156507799)

[5.7.1 功能 32](#_Toc156507800)

[5.7.2 整体框图 33](#_Toc156507801)

[5.7.3 接口列表 33](#_Toc156507802)

[5.7.4 接口时序 33](#_Toc156507803)

[5.7.5 关键电路 33](#_Toc156507804)

[5.7.6 三级模块设计 34](#_Toc156507805)

[5.8 DataStorage 34](#_Toc156507806)

[5.8.1 功能 34](#_Toc156507807)

[5.8.2 整体框图 34](#_Toc156507808)

[5.8.3 接口列表 35](#_Toc156507809)

[5.8.4 接口时序 35](#_Toc156507810)

[5.8.5 关键电路 35](#_Toc156507811)

[5.9 RequestBuffer（ReqBuf） 35](#_Toc156507812)

[5.9.1 功能 35](#_Toc156507813)

[5.9.2 整体框图 35](#_Toc156507814)

[5.9.3 接口列表 36](#_Toc156507815)

[5.9.4 接口时序 36](#_Toc156507816)

[5.9.5 关键电路 37](#_Toc156507817)

[5.10 RefillBuffer 37](#_Toc156507818)

[5.10.1 功能 37](#_Toc156507819)

[5.10.2 整体框图 38](#_Toc156507820)

[5.10.3 接口列表 38](#_Toc156507821)

[5.10.4 接口时序 38](#_Toc156507822)

[5.10.5 关键电路 38](#_Toc156507823)

[5.11 ReleaseBuffer 39](#_Toc156507824)

[5.11.1 功能 39](#_Toc156507825)

[5.11.2 整体框图 39](#_Toc156507826)

[5.11.3 接口列表 39](#_Toc156507827)

[5.11.4 接口时序 39](#_Toc156507828)

[5.11.5 关键电路 40](#_Toc156507829)

[5.12 Prefetcher 40](#_Toc156507830)

[5.12.1 功能 40](#_Toc156507831)

[5.12.2 整体框图 42](#_Toc156507832)

[5.12.3 接口列表 42](#_Toc156507833)

[5.12.4 接口时序 45](#_Toc156507834)

[5.12.5 三级模块设计 46](#_Toc156507835)

[5.12.6 三级模块 PrefetchReceiver 47](#_Toc156507836)

[5.12.7 三级模块PrefetchQueue 50](#_Toc156507837)

[5.12.8 三级模块BestOffestPrefetch 52](#_Toc156507838)

[5.12.9 二级模块TemporalPrefetch 64](#_Toc156507839)

[6 PPA 72](#_Toc156507840)

[7 验证关注点 72](#_Toc156507841)

[8 Floorplan 建议 72](#_Toc156507842)

[9 遗留问题 73](#_Toc156507843)

1. 简介
   1. 文档介绍

*概述本文档的目的、用途、适用人群、在整体项目中的位置等*

本文档是CoupledL2（以下简称 CPL2）的AS文档，描述CPL2架构设计。本文档主要用于指导芯片模块的详细设计及验证。

* 1. 参考文档

*列出相关的参考文档。*

1. XiangShan官方文档（<https://xiangshan-doc.readthedocs.io/zh_CN/latest/huancun/overview/>）
2. TileLink总线协议（<https://www.starfivetech.com/uploads/tilelink_spec_1.8.1.pdf>）
3. Best Offset Prefetch：《Best-offset hardware prefetching》

（<https://ieeexplore.ieee.org/document/7446087> ）

1. Teporal Prefetch：《Temporal Prefetching Without the Off-Chip Metadata》（<https://dl.acm.org/doi/10.1145/3352460.3358300> ）
   1. 术语说明

*列出本文档的关键术语说明。*

表1.1 术语说明

|  |  |  |
| --- | --- | --- |
| **缩写** | **全称** | **描述** |
| BOP | Best Offset Prefetch | 最佳偏移量预取 |
| DCache | Data Cache | 数据缓存 |
| TP | Temporal Prefetch | 基于时间相关性的预取 |
| Meta2 | Meta data’s meta data | 元数据的元数据 |
| rrTable | Recent Request Table | 最近访问请求记录表 |

1. 设计规格

*如果是模块的AS需要列出本模块支持的规格，如果是总体AS可忽略*

*规格中包含功能、性能、PPA规格。*

CoupledL2：

* 1MB容量，8路组相连，分为4个Bank
* 缓存块Block大小为 64 Bytes，总线数据位宽 256 bit = 32 Bytes，所以每次传输包含64/32 = 2个beat
* 采用基于目录的TileLink缓存一致性协议
* 包含策略为全包含（Inclusive）
* 采用主流水线架构
* 16个MSHR，其中15个为AB请求通用，1个为B请求专用
* 支持同set请求的并行访问（去除set阻塞）
* 支持在收到重填数据时再进行替换路的选择和释放
* 支持访存请求和预取请求的融合
* 支持产生Hint信号提前唤醒load指令
* 支持BOP预取器、Temporal预取器、接收来自L1的预取（receiver）

1. 功能描述
   * 1. *进行功能概述。（从输入、处理、输出几个方面概述设计实现了什么功能）*
     2. *按特性，每条特性分步骤进行详细描述*

*注意:要描述清楚对于异常的输入的处理，比如是支持纠错（怎么纠正），还是上报异常或中断，还是不处理错误但保证不死机，还是直接忽略等。*

*例：*

* 1. 功能概述

CPL2模块通过通道控制模块接收TileLink总线上的请求，并将其转换为Cache内部请求。之后通过请求仲裁模块进入主流水线，读取目录获取缓存块的状态，根据缓存块状态和请求信息判断是否能够处理。若本层缓存可以直接处理该请求，则继续在主流水线中进行读数据、更新目录等操作，然后进入通道控制器模块，转化为 TileLink 总线请求回复。

若需要和其它缓存进行交互才能处理该请求，则为其配一个MSHR。MSHR根据需求向上下层Cache发送子请求，等待收到响应并满足释放条件后，再释放任务重新进入主流水线，进行读缓冲区、读写数据、更新目录等操作，然后进入通道控制器模块，转化为 TileLink 总线请求回复。

当一个请求所需的全部操作在MSHR中完成时，MSHR被释放，等待接收新的请求。

* 1. 分特性详细描述：
     1. 特性1：接收和发送TileLink总线请求及响应

CPL2包含和上下层总线相连的通道控制模块。在功能上，Sink 模块接收 TileLink 总线上的主动请求和被动响应，转化为MSHR内部请求。具体地：

对于主动请求，将该请求转换为 CPL2内部请求发送给ReqArb模块。

对于被动响应，则将响应回馈给正在处理对应请求的 MSHR。

通道控制器还负责接收 CPL2的内部请求，经过处理和包装发送到 TileLink 总线上。

具体各模块完成的功能将在模块设计部分详细介绍。

* + 1. **特性2：采用TileLink一致性协议**

XiangShan的缓存系统中每个地址都遵循TileLink一致性树的规则。每个块在缓存系统的各层都有N(None)、B(Branch)、T(Trunk)、TT(Trunk TIP)四个状态，分别对应着没有权限、只读、不读不写（内部状态）、可读可写。

一致性树按照内存、L3、L2、L1的顺序自下而上生长，内存作为根节点拥有可读可写的权限，子节点的权限都不能超过父节点的权限。其中TT代表拥有T权限的最上层子节点（也是T权限树的叶子节点），说明该节点上层只有N或B权限，相反T权限而不是TT权限的节点代表上层一定还有T/TT权限节点。详细规则请参考TileLink手册9.1章。

* + 1. **特性3：采用目录记录缓存行信息**

CPL2是基于目录结构的 Inclusive Cache（此处所指的“目录”是广义的，包含元数据和Tag）。元数据包含：【状态位state / 脏位dirty / 是否在上层缓存 clients / 在上层的别名位 alias / 是否是预取上来的 prefetch / 来自哪个预取器 prefetchSrc / 是否被访问过 accessed】

在流水线S1级 RequestArb会向目录发起读请求，读取Tag Array判断是否命中。如果命中则选择命中路，如果不命中则根据替换算法选择一个替换路，然后将选中路的元数据信息返回给S3级MainPipe。

* + 1. **特性4：采用主流水线架构**

CPL2采用主流水线架构，来自各通道的请求经过仲裁进入主流水线，进行都目录操作，然后根据请求信息和目录结果安排响应的操作（3.1节有介绍）。具体地：

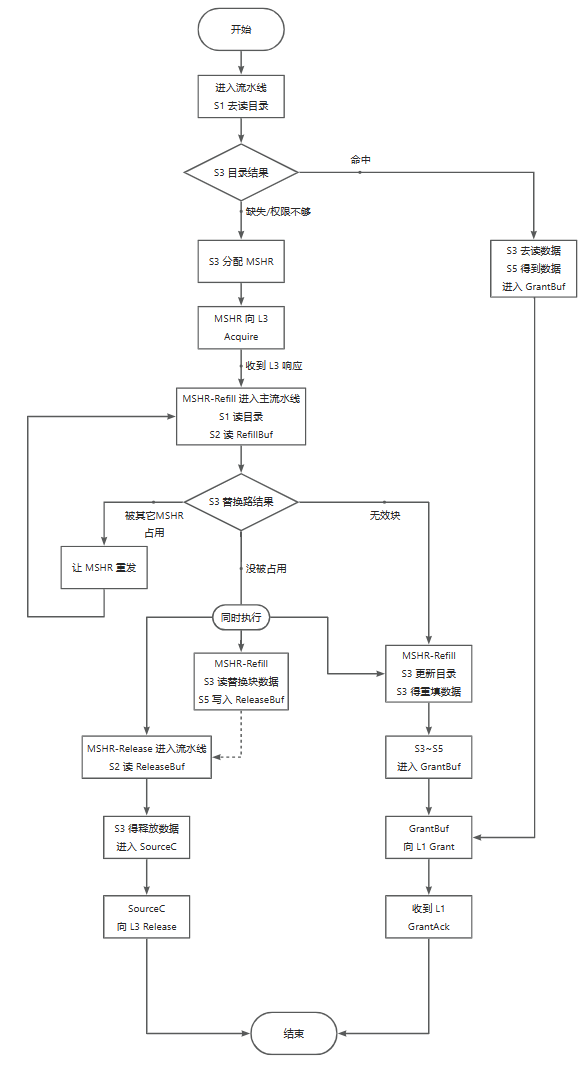
**一个 Acquire 请求的处理流程：**

见下页图

**一个 Probe 请求的处理流程：**

1. 从 SinkB 接收到总线上的 L3 Probe 请求，并转化为内部请求；
2. S1 进入流水线，并读目录查询该数据块状态；
3. S3 得到目录结果，如果不命中，则可以直接进入 SourceC，向 L3 响应数据块不存在；END
4. 如果命中且该数据块仅在 L2 中不在 L1 中，则 S3 去读数据，S5 得到读数据结果，然后进入 SourceC，向 L3 响应（如果是脏的则包含数据）；END
5. 如果命中且该数据块既在 L2 中又在 L1 中，则我们还需要去 Probe L1，此时为其分配一个 MSHR，记录这个请求的信息；同时继续 S3 去读数据，S5 得到读数据结果，并写入 ReleaseBuf（mshrId 作为索引）；
6. MSHR 通过 SourceB 向 L1 发送 Probe，等待；
7. L1 返回 ProbeAck，如果有数据则写入 ReleaseBuf（mshrId 作为索引）；并唤醒 MSHR，让其发出 MSHR\_ProbeAck 任务进入流水线；
8. MSHR\_ProbeAck 在 S2 读 ReleaseBuf，在 S3 获取到数据，可在S3~S5 进入 SourceC，向 L3 响应数据（如果是脏的则包含数据）； END

之所以 5 中，我们还要去读数据写入 ReleaseBuf，是考虑到如果 L1 的 ProbeAck 没有返回数据，但是 L2 本身是脏的所以需要向下 ProbeAck 数据。这种情况下我们就提前将数据读出来，加快 ProbeAck 流程。



**一个 Release 请求的处理流程：**

1. 从 SinkC 接收到总线上的L1 的 Release 请求，并转化为内部请求；
2. S1 进入流水线，并读目录查询该数据块状态；
3. S3 得到目录结果，一定是命中的（L2 Inclusive 特性决定的），则 S3 将新数据写入数据存储，然后可在 S3~S5 进入 SourceD，向 L1 响应 ReleaseAck；END
   * 1. **特性5：在收到重填数据时再进行替换路的选择和释放**

*缓存在收到新请求但是set已满的时候，按照常规逻辑，我们需要先选择一个旧的数据块进行替换，将它释放为新数据块腾出位置，然后等待新数据块从下级缓存重填上来，再将其写入。但是这种方式会存在一些问题：*

*一方面，因为从下级缓存重填往往需要较长的时间（几十拍到上百拍），在这段时间内，旧的数据块已经被释放掉，新的数据块尚未收到，所以这个位置实际上是没有 有效数据的，从而造成了缓存资源的空闲和浪费，相当于降低了缓存的有效容量。*

*另一方面，如果在这段时间内，上层缓存正好又要访问被替换的数据块，因为此时数据块已经被释放，所以它只能再次向下层缓存获取，从而使得访问延迟大大增加。*

CPL2将替换路的选择和替换数据的释放延后到收到重填数据之时。具体地，在请求进入缓存时，需要读取目录信息来判断是否命中。如果命中，则读取数据并返回（标准流程）。如果缺失，我们不会根据读目录结果来选择一个替换块并安排替换块的释放，我们只为其分配一项 MSHR，并向下层缓存发请求获取数据。

等待下层返回重填数据后，让 MSHR 任务再次读目录，此时再选出替换块，从数据存储单元读出替换块的数据，向下层缓存释放。最后再将新数据块写入存储单元。

由于我们与DataStorage交互只在MainPipe的S3级，且DataStorage的SRAM是单端口的，因此我们无法利用一个MSHR Task同时完成①将被替换数据块的内容读出并向下层缓存释放，②将新数据块写入。所以我们必须要分成 MSHR Refill和MSHR Release两个任务，Refill先于Release发出。

又因为两个观察：①读旧数据必须要早于写新数据；②必须尽快向L1返回数据，所以我们为两个MSHR Task分别安排如下任务：

MSHR Refill：①读RefillBuffer获取重填数据反馈给L1；②读DataStorage获取旧数据存入ReleaseBuf；③更新目录为新数据的元数据。

MSHR Release：①读ReleaseBuf并将数据向L3释放；②读 RefillBuffer将重填数据写入DataStorage。

* + 1. **特性6：支持同Set请求的并行访问**

对于Inclusive 的缓存，对同一地址的操作只会涉及到同一个路，因此只要确保对不同地址的操作占据不同的路即可。对于命中的请求，可以确定不同地址不是一路。对于缺失的请求，需要选择一个替换路，我们需要在选择替换路的时候，避开其它MSHR正在占用的路（占用的定义是【MSHR已经选择出了替换路，但此时尚未将替换数据释放下去】，对应特性5中）。 我们将所有MSHR的信息传给目录，来帮助进行决策。

* + 1. **特性7：利用Hint信号进行load提前唤醒**

CPL2对每一次重填回L1，都会在重填响应发出的前3拍，生成一个Hint信号送到MemBlock。MemBlock会提前唤醒对应的 load指令，让其提前发出进入loadpipe，从而在其需要数据的那一拍，正好能收到L2的重填数据，从而降低了 load 的延迟。L2根据流水线每一级以及 GrantBuffer的状态，来分别为每一个请求计算它是否应当在这一拍里发送 Hint 信号。

* + 1. **特性8：进行硬件预取**

CPL2包含BOP预取器、Temporal预取器，还可以接收来自L1的预期请求（receiver）。Prefetcher模块接收来自L2 Cache的预取训练数据以及对应的TP meta数据，并将训练数据分别发送给发送给BOP模块以及TP模块进行预取训练。同时，Prefetcher模块将来自L1 DCache的预取请求发送给PrefechReceiver模块，并进行筛选。

Prefetcher模块接收来自BOP模块、TP模块以及PrefechReceiver模块的预取请求，并将预取请求缓存至PrefechQueue中。当外部总线允许接收预取请求时，Prefetcher模块从PrefetchQueue中取出预取请求并发送。Prefetcher模块接收预取请求的响应，并发送至BOP模块以及TP模块。

Prefetcher模块接收TP模块完成训练后更新的TP meta数据，当外部总线允许时，发送更新的TP meta数据。

* + 1. **特性9：支持请求融合**

实验观察发现，L2缓存中存在占比较大的不及时预取，也就是Prefetch虽然预测到了未来需要的数据，但请求发送较晚，当Prefetch导致的Cache miss还在MSHR中等待L3数据返回时，对同地址的Acquire请求已经到达L2。为了不让此类Acquire请求在RequestBuffer入口被阻塞，导致L2入口被占满，后续请求无法进入，当前L2设计了一套合并不及时的Prefetch与后续同地址Acquire的请求融合机制。请求融合功能实现如下：

1. 在SinkA通道的入口RequestBuffer中判断来自L1的A请求是否满足合并条件，条件为：新请求为 Acquire，且在 MSHRs 中存在 miss 请求为 Prefetch，并与 Acquire 地址相同。

2. 若满足合并条件，则新请求不需要进入队列被阻塞，而是直接进入同地址Prefetch对应的MSHR项中，并对该项标记mergeA，新增加一系列请求状态信息，使其包含两个请求的内容。

3. 当目标数据从L3返回后，MSHR项被唤醒，发送任务到主流水线进行处理。在主流水线中选择替换路并回填新数据，而数据块的meta则更新为Acquire请求处理完成后的状态，同时该请求还会将信息传入预取器作为训练。

4. 在处理请求响应时，这个合并请求会从主流水线进入GrantBuffer，对于Prefetch请求，L2返回预取响应；而对于Acquire请求，L2通过grantQueue队列对发出Acquire的上游节点返回数据和响应。

* + 1. **特性10：处理Cache别名问题**

在昆明湖架构中，DCache采用 64KB 的 4 路组相联结构，Cache索引和块偏移所占 bit 数已经超过了页偏移 (4K 页的页偏移为 12 bits)，由此引入了 Cache别名问题：当两个虚页映射到同一个物理页时，两个虚页的别名位 (alias bit)，很有可能是不一样的，如果不做额外处理的话，通过VIPT索引后这两个虚页会位于 Cache不同的 set，导致同一个物理页在 Cache中缓存了两份，造成一些一致性错误。

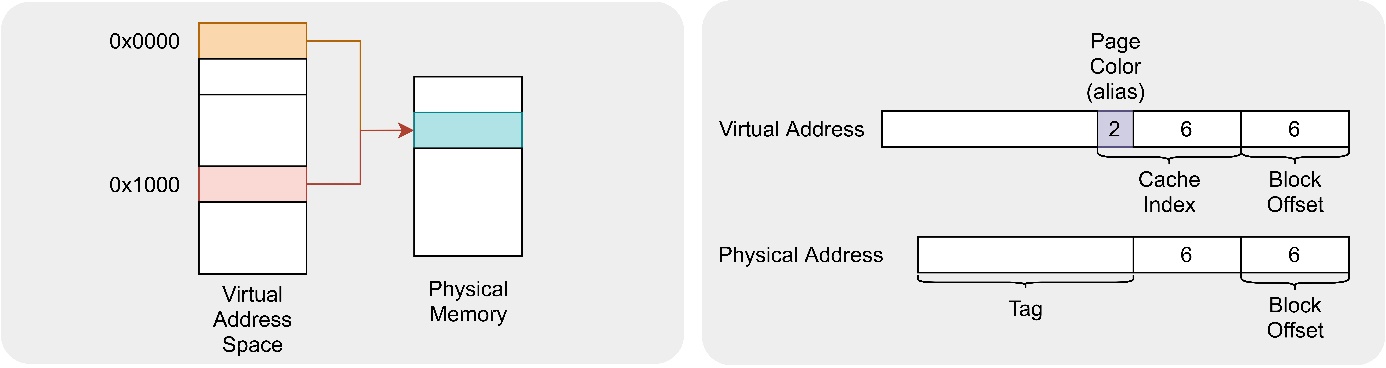


图 3-1 Cache别名问题

为了让 L1 Cache 继续沿用 VIPT，CoupledL2采用硬件方式解决 Cache别名问题。具体的解决方式是由 L2 Cache 记录上层数据的别名位，保证一个物理块在上层的一个 VIPT Cache中最多只有一种别名位。在上层向下Acquire时会带上别名位，如果命中但和记录的别名不一致，就会向上将之前记录的别名位的块Probe下来，并修改记录的别名。

下面举一个例子说明 L2 如何解决 Cache别名问题。如下图所示，DCache 中有一个虚地址为0x0000 的块，虚地址 0x0000 和 0x1000 映射到了同一个物理地址，且这两个地址的别名是不一样的；此时 DCache 向 L2 Acquire 了地址为 0x1000 的块，并在 Acquire 请求的 user 域中记录了别名 (0x1)，L2 在读目录后发现请求命中，但是 Acquire 的别名 (0x1) 和 L2 记录的 DCache 在该物理地址的别名 (0x0) 不同，于是 L2 会发起一个 Probe 子请求，并在 Probe 的 data 域中记录要 Probe 下来的别名 (0x0)；Probe 子请求完成后，L2 再将这个块返回给 DCache，并将 L2 目录中记录的别名改为 (0x1)。

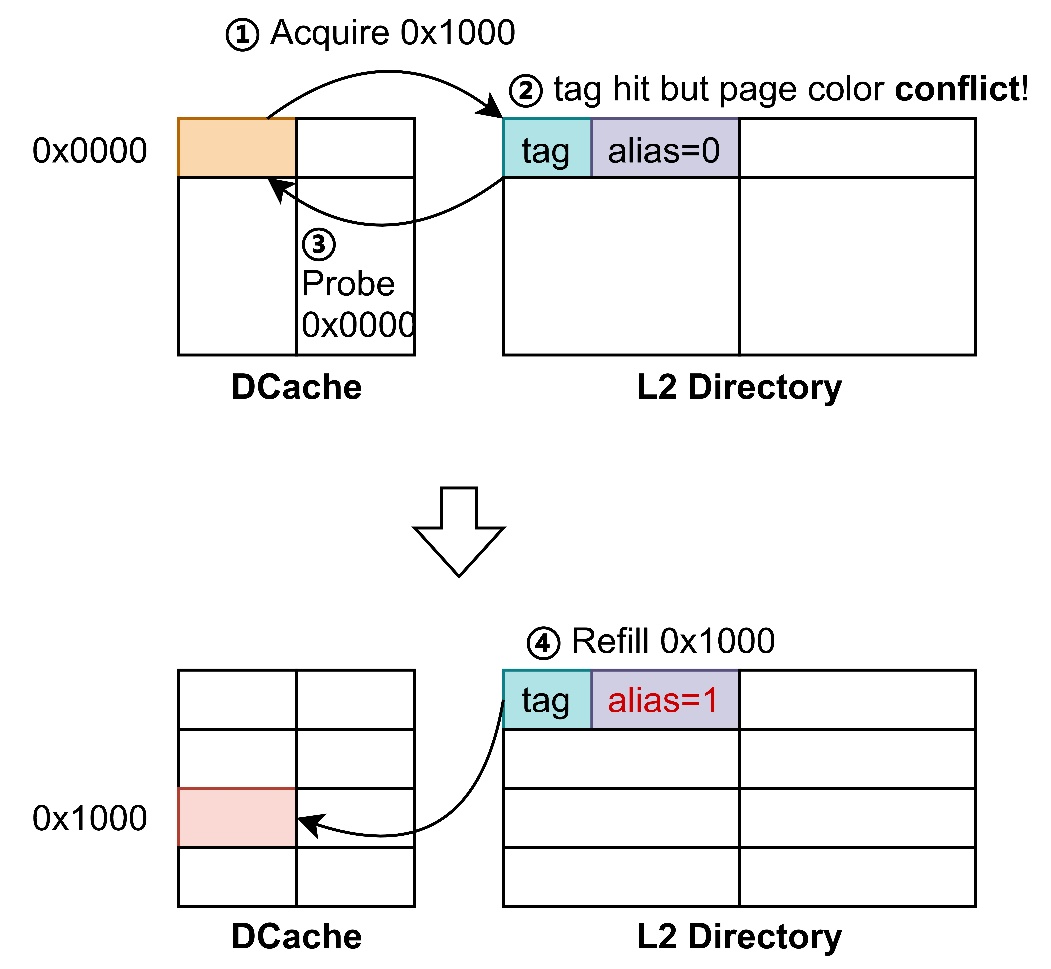


图 3-2 Cache别名问题处理流程

1. 总体设计

*总体设计的标准：对设计进行分解，完成子模块划分、顶层接口定义、接口时序、数据流、控制流的设计。*

*总体设计面向的对象：顶层集成人员、验证人员、软件人员、设计人员*

1. 1. 整体框图

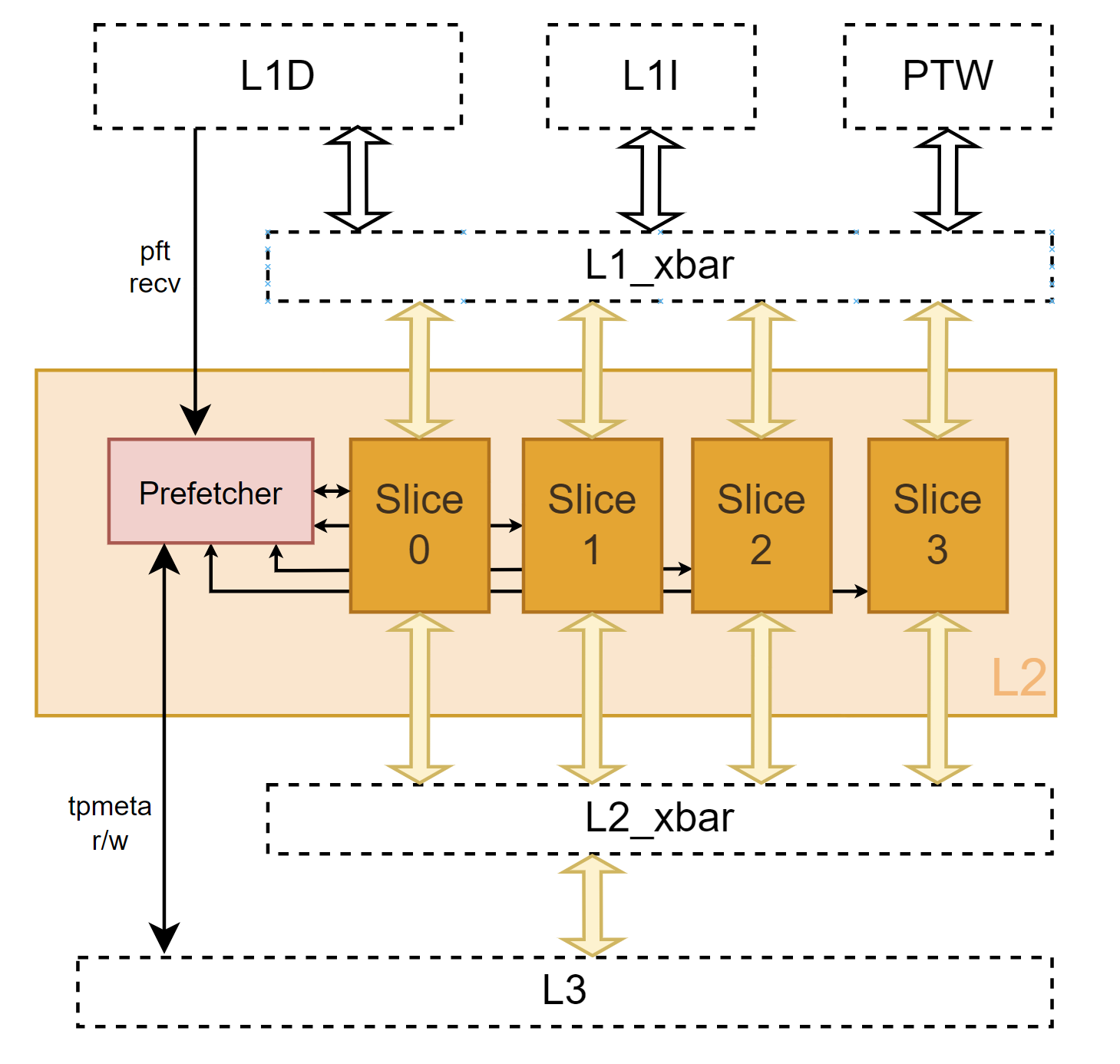
*此节画整体框图并配上文字说明。图可以不只1个，要体现模块划分、模块之间的关系、整体流水线设计等。*

*图中不同属性的信号要注意区分开，比如控制信号与数据信号、一般信号与时钟复位信号等。*

*控制流(代表控制关系的信号组流向)和数据流（设计各个环节处理的数据信号组流向）可用数字标明按照处理顺序上的先后关系。*

*简洁起见，图中可以使用代号标记信号组，然后在后文中详细说明每组信号组成，标准总线接口可不展开描述，比如使用axi\_m\_sig代表AXI master接口的所有信号。*

*文字描述要简洁清晰，重点是描述模块外部接口、内部各子模块功能、模块之间的关系。*



* 1. 接口列表

*如有专门的的接口列表文档，本章节可忽略，附上对应的接口列表名称路径即可*

*描述本模块所有的输入输出接口。接口列表是顶层集成的重要依据，需要准确清晰，并说明注意事项。*

*其中：*

1. *源/目的一栏描述源头模块和目的地模块*
2. *描述一栏需要列举信号每bit含义描述、取值范围、跟其他信号之间的约束（比如Xdata在Xvalid为高时有效、Xmode为2时signalA取值只能为1、2等等））*

见 CoupledL2.xlsx

* 1. 接口时序

*顶层接口时序图*

1. *非标准接口需要有关键信号的时序示意图*
2. *标准接口说明使用的接口协议即可（如AXI、APB等）*

*例:*

顶层接口遵循 TileLink总线协议。

* 1. 时钟复位

*描述时钟相关设计：时钟域划分、时钟信号来源、时钟频率、时钟门控等。*

*描述复位相关设计：复位信号来源、异步复位还是同步复位、复位过程、解复位过程。*

*描述时钟信号与复位信号的对应关系，例如：*

|  |  |  |
| --- | --- | --- |
| Module | Clock | Reset |
| Core | Clk\_core | rst\_core\_logic\_n  rst\_core\_cfg\_n |
|  | Clk\_iref (apb\_slave) | rst\_iref\_n |
| DDRC/P | Clk\_ddr\_c\_p | Rst\_ddrc\_n  Rst\_ddrp\_n |
|  | Clk\_ddr\_axi | Rst\_ddri\_n |

CPL2内采用统一的时钟域，同步复位。

* 1. 寄存器配置

*如果模块涉及寄存器配置（包括状态信息、统计信息通过寄存器上报软件）需要简要描述相关的寄存器，描述可使用表格，格式如下。*

*总体AS可忽略此节*

表4.2 XXXX寄存器说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器** | **地址** | **复位值** | **属性** | **描述** |
| cfg\_fetch\_en | 0X0 | 32’d0 | RW | bit31-1: 保留  bit0: fetch\_en寄存器配置信号 |
| cfg\_clk\_sel | 0X 4 | 32’d0 | RW | bit31-1: 保留  bit0: 时钟动态切换信号  0：选择晶振时钟  1：选择PLL时钟 |
| pll\_lock | 0X 8 | 32’d0 | RO | bit31-1: 保留  bit0: PLL锁定信号  0：PLL未锁定  1：PLL锁定 |
|  |  |  |  |  |

*注：RO——只读寄存器；RW——可读可写寄存器*

无

* 1. 补充说明

*可选项。按照模块特点，根据4.1整体框图的划分，补充部分核心模块、关键电路、关键信号信号的说明。*

无

1. 模块设计

*模块设计的标准：能用于指导RTL代码的编写。理想情况下，RTL代码是对设计方案的翻译。*

*模块设计面向的对象：模块设计人员、模块验证人员*

*本模块下面各级子模块的详细设计说明。包括模块功能概述、模块IO、模块的设计框图、关键设计（流水线、memory（ram、fifo、寄存器组等）、主控制电路（包含不限于状态机、仲裁、关键握手时序等））。*

*对于关键设计描述要求：*

1. *Memory（ram、寄存器组、fifo等）：宽度、深度、接口含义、读写时序、data的详细描述、data在memory中存放的格式等信息*
2. *流水：有流水线框图、每一级流水线描述*
3. *仲裁：仲裁策略、优先级处理等*
4. *状态机：有状态机设计图，需要有每个状态描述、状态之间的跳转条件、复位状态等。*

*例如：*

* 1. Slice

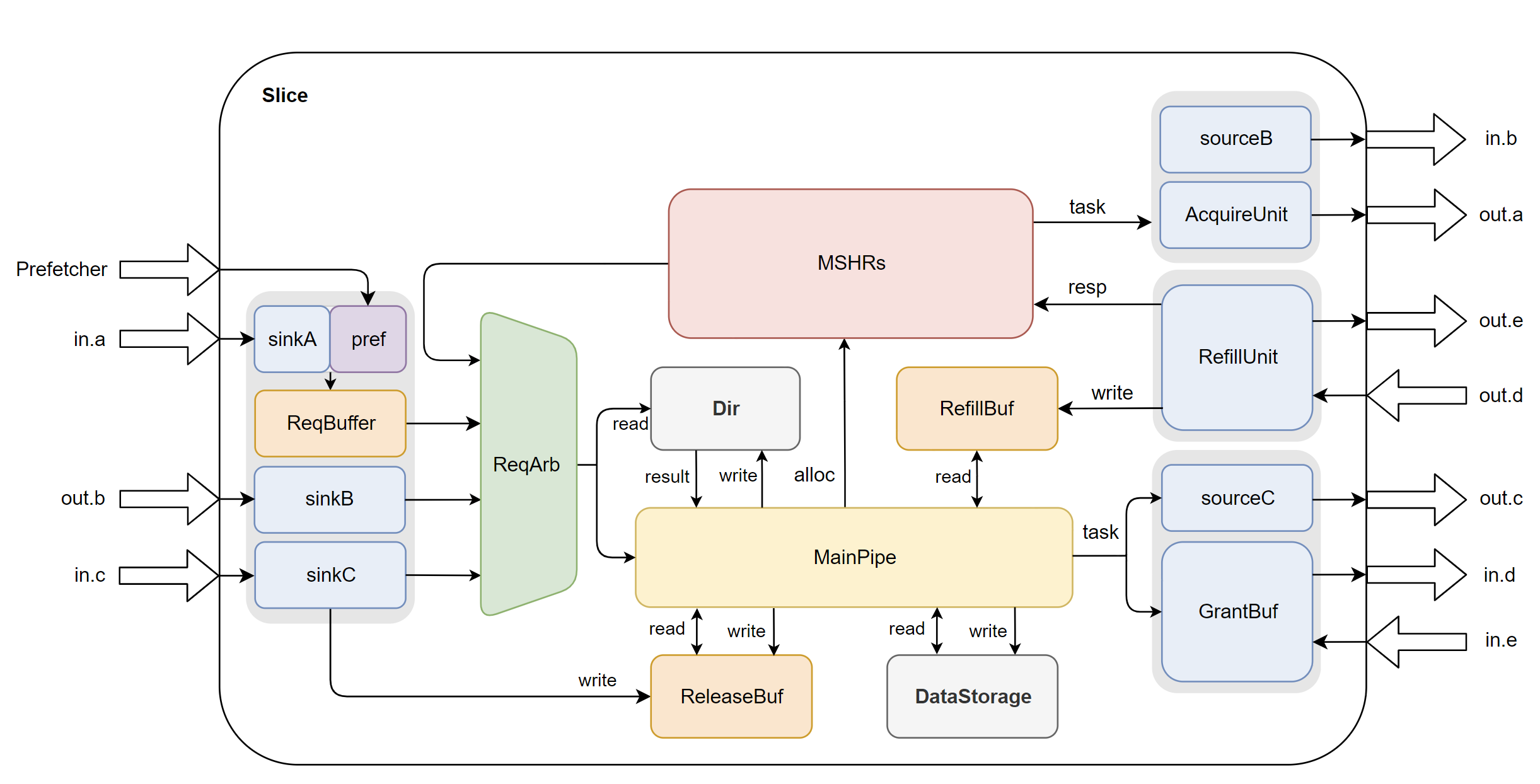
1. * 1. 功能

*简要描述该模块实现的功能*

CoupledL2将地址空间按照block索引的低位（第7，6位）划分为4个Slice。每个Slice独立进行处理，其功能即第三章功能描述部分。

* + 1. 整体框图

*要求如4.1*



* + 1. 接口列表

*只需要写IO信号*

* + 1. 接口时序
    2. 关键电路

*如果该层级只有例化，没有其他电路可不写。*

*主要包括Memory、流水线设计、仲裁、状态机等。多画图，配以文字说明*

只有例化和连线。

* + 1. 三级模块设计

*内容和要求同二级模块设计，如果还有4级模块需要一级级往下写。*

其余模块（除Prefetcher）均为其子模块。

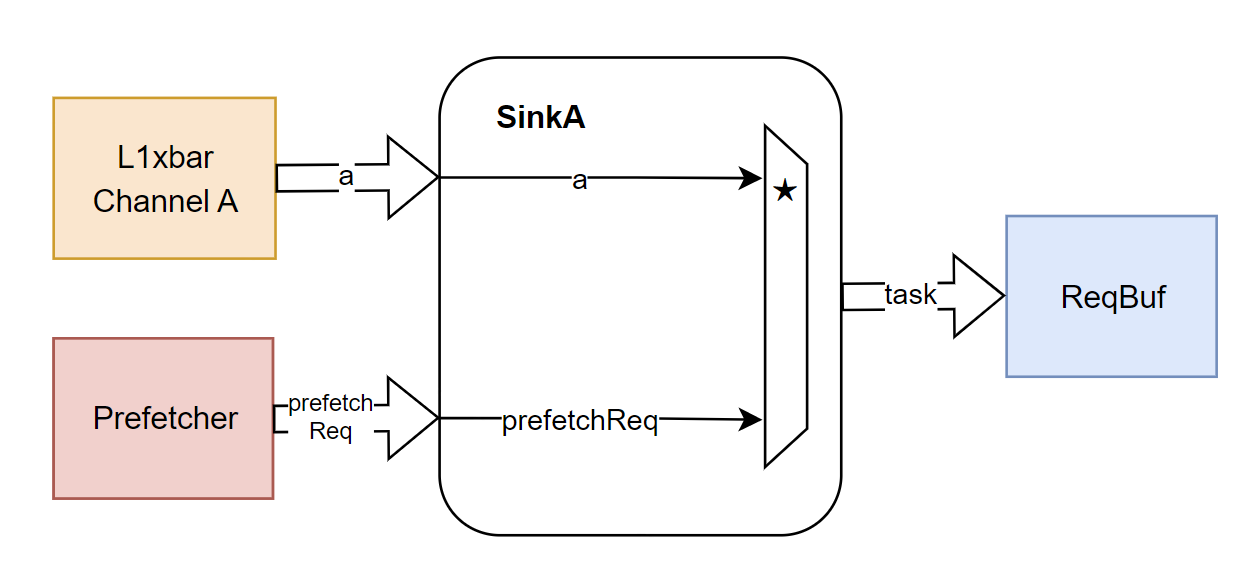
* 1. 通道控制器
     1. SinkA
        1. 功能

*简要描述该模块实现的功能*

SinkA把来自【①总线A通道，②预取器】的请求进行处理，转化为内部任务格式，然后发送给RequestBuffer。二者同时来请求时，总线A通道的请求有着更高的优先级。

* + - 1. 整体框图

*要求如4.1*



* + - 1. 接口列表

*只需要写IO信号*

* + - 1. 接口时序

S0：总线A/预取器请求握手，进入 SinkA；

当拍就通过 task 发送给 ReqBuf

处理结束。

* + - 1. 关键电路

无

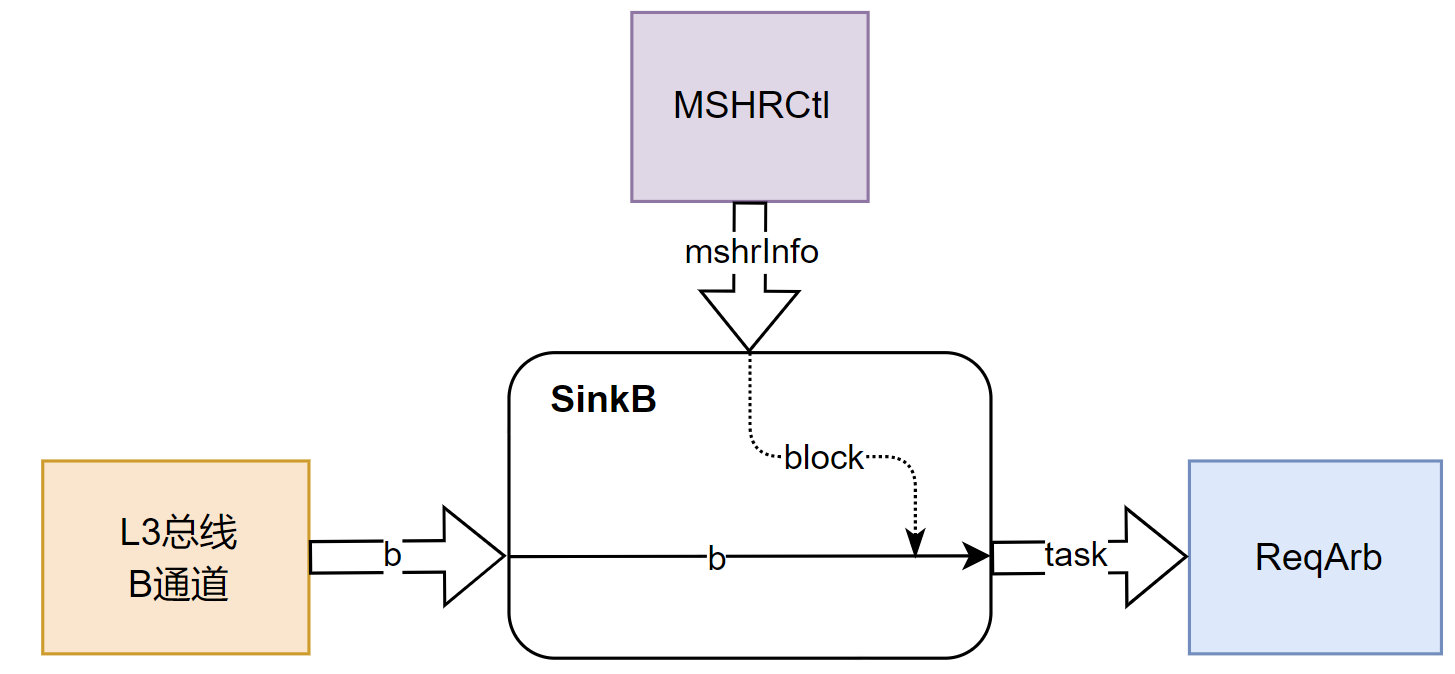
* + 1. SinkB
       1. 功能

*简要描述该模块实现的功能*

SinkB把来自总线B的请求进行处理，转化为内部任务格式，然后发送给ReqArb。同时如果 MSHR 中有正在进行的【①同地址请求，且不能被B请求嵌套；②准备替换的数据块地址与B请求地址相同】，则阻塞总线B请求进入SinkB。

* + - 1. 整体框图

*要求如4.1*



* + - 1. 接口列表

*只需要写IO信号*

* + - 1. 接口时序

S0：总线B请求握手，进入SinkB；

当拍就通过 task 发送给 ReqArb

处理结束。

* + - 1. 关键电路

无

* + 1. SinkC
       1. 功能

*简要描述该模块实现的功能*

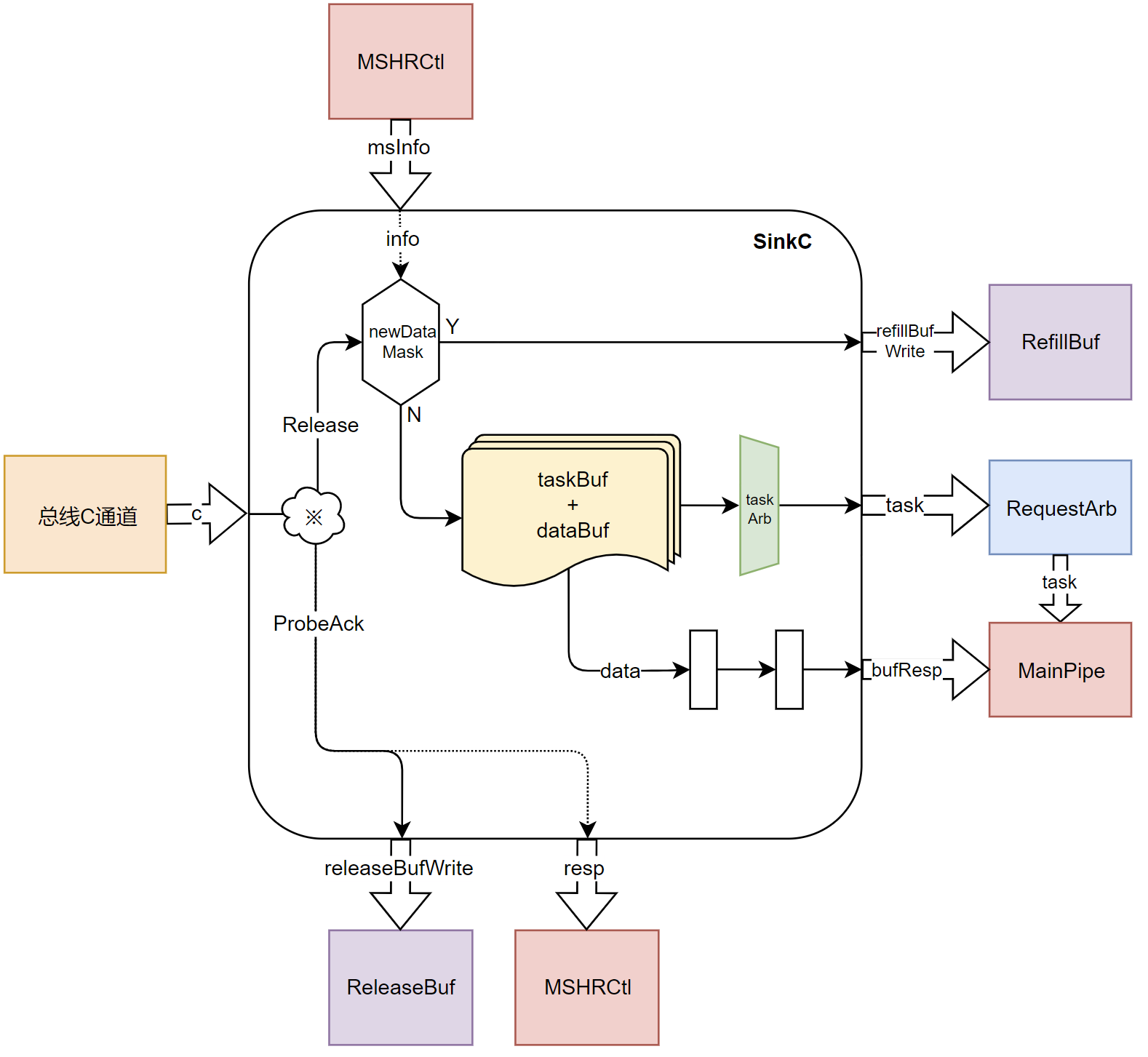
SinkC接收来自总线C通道的请求，进行如下操作：

* 1. 如果请求是 Release(Data)，则为其分配一个 buffer 项保存，等到流水线可以接收C请求的时候，就发往RequestArb进入主流水线；如果包含数据，则延迟两拍在请求进入到S3的时候将其数据发送给MainPipe；
  2. 如果请求是 ProbeAckData，则直接向MSHR发送反馈，同时将其数据写入ReleaseBuf 中。

另外，因为目前缺失重填操作会首先将数据返回给L1，然后再安排重填数据（在RefillBuf中）写入L2的DataStorage，二者之间存在一个时间差。如果在此期间L1就将脏数据释放下来，为了保证写入L2的最新数据，我们就让ReleaseData也将其数据同步写入一份到RefillBuf中，覆盖原有的重填数据。

* + - 1. 整体框图

*要求如4.1*



* + - 1. 接口列表

*只需要写IO信号*

* + - 1. 接口时序

**对于 ReleaseData：**

S0：总线C请求握手，进入SinkC；

S1：请求可以进入RequestArb；

S2：（如果需要写入 RefillBuf）在这一拍写 RefillBuf；

S3：请求的数据传给 MainPipe；

处理结束。

**对于 ProbeAckData：**

S0：总线C请求握手，进入SinkC，当拍发送给releaseBuf，同时给响应到MSHR；

处理结束。

* + - 1. 关键电路

无

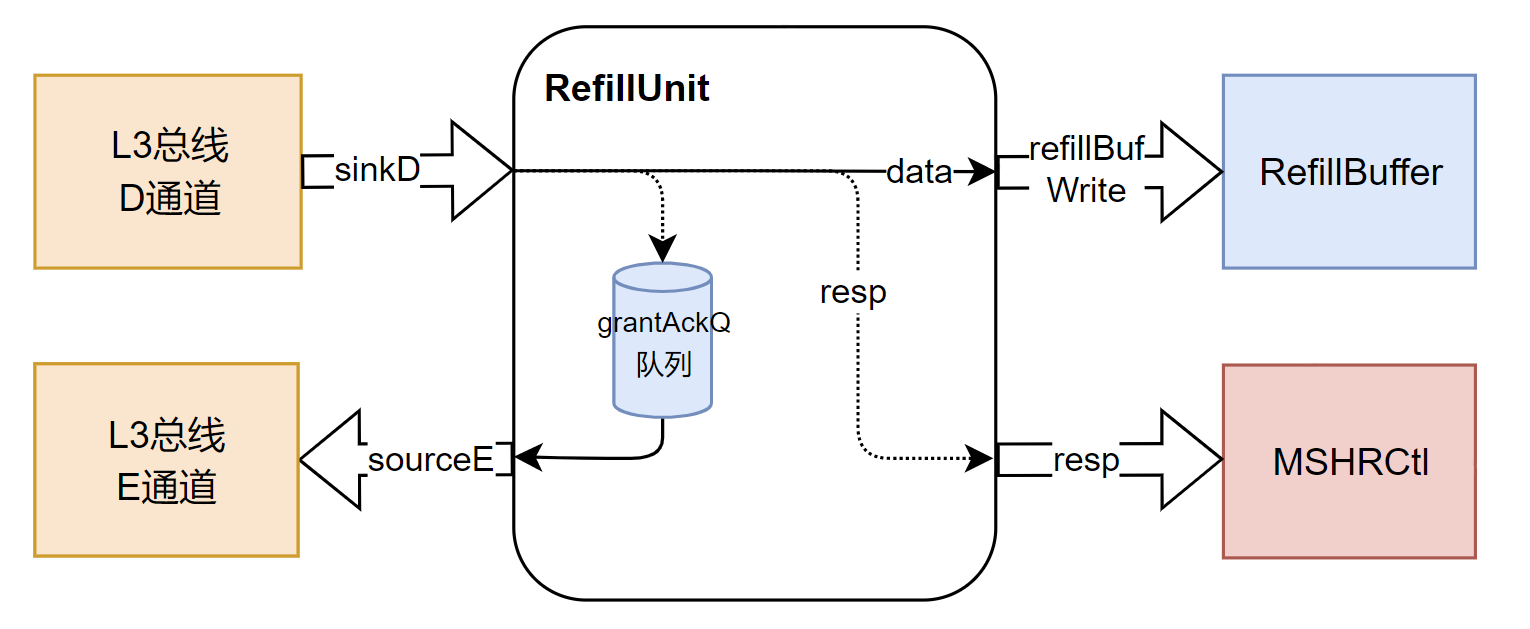
* + 1. RefillUnit
       1. 功能

*简要描述该模块实现的功能*

RefillUnit接收总线D通道的响应，将数据存入RefillBuffer，同时给出响应到MSHRCtl。RefillUnit也负责为收到的Grant/GrantData回复GrantAck，通过总线E通道向下发出。

* + - 1. 整体框图

*要求如4.1*



* + - 1. 接口列表

*只需要写IO信号*

* + - 1. 接口时序

*d*

* + - 1. 关键电路

GrantAck 会进入一个队列。

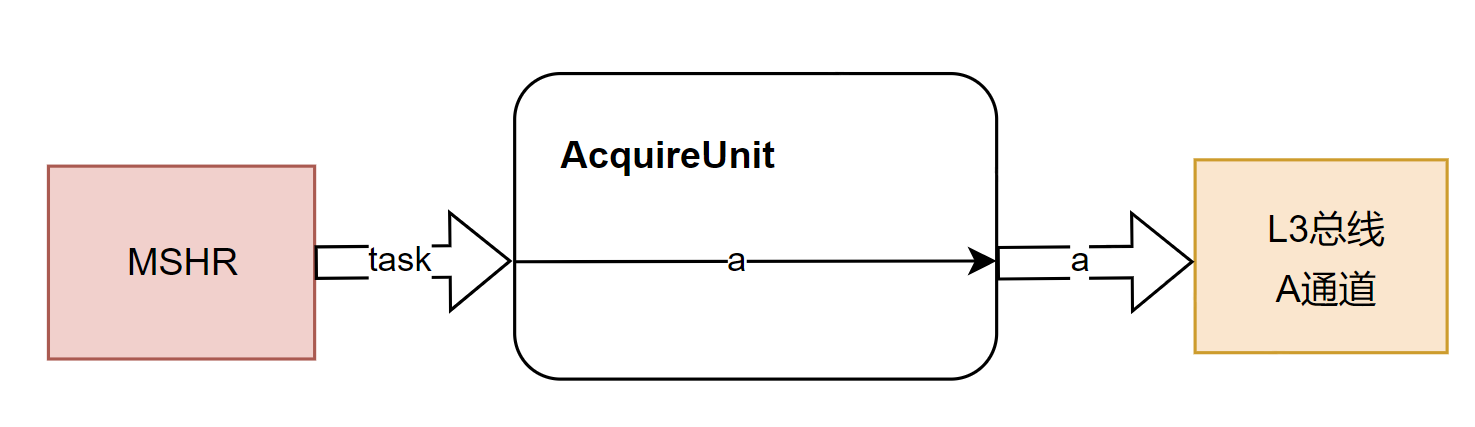
* + 1. AcquireUnit
       1. 功能

*简要描述该模块实现的功能*

AcquireUnit把来自MSHR的任务进行处理，转化总线请求，然后发送给向下的总线A通道。

* + - 1. 整体框图

*要求如4.1*



* + - 1. 接口列表

*只需要写IO信号*

* + - 1. 接口时序

S0：MSHR请求握手，进入AcquireUnit；

当拍就发送给L3总线A通道；

处理结束。

* + - 1. 关键电路

无

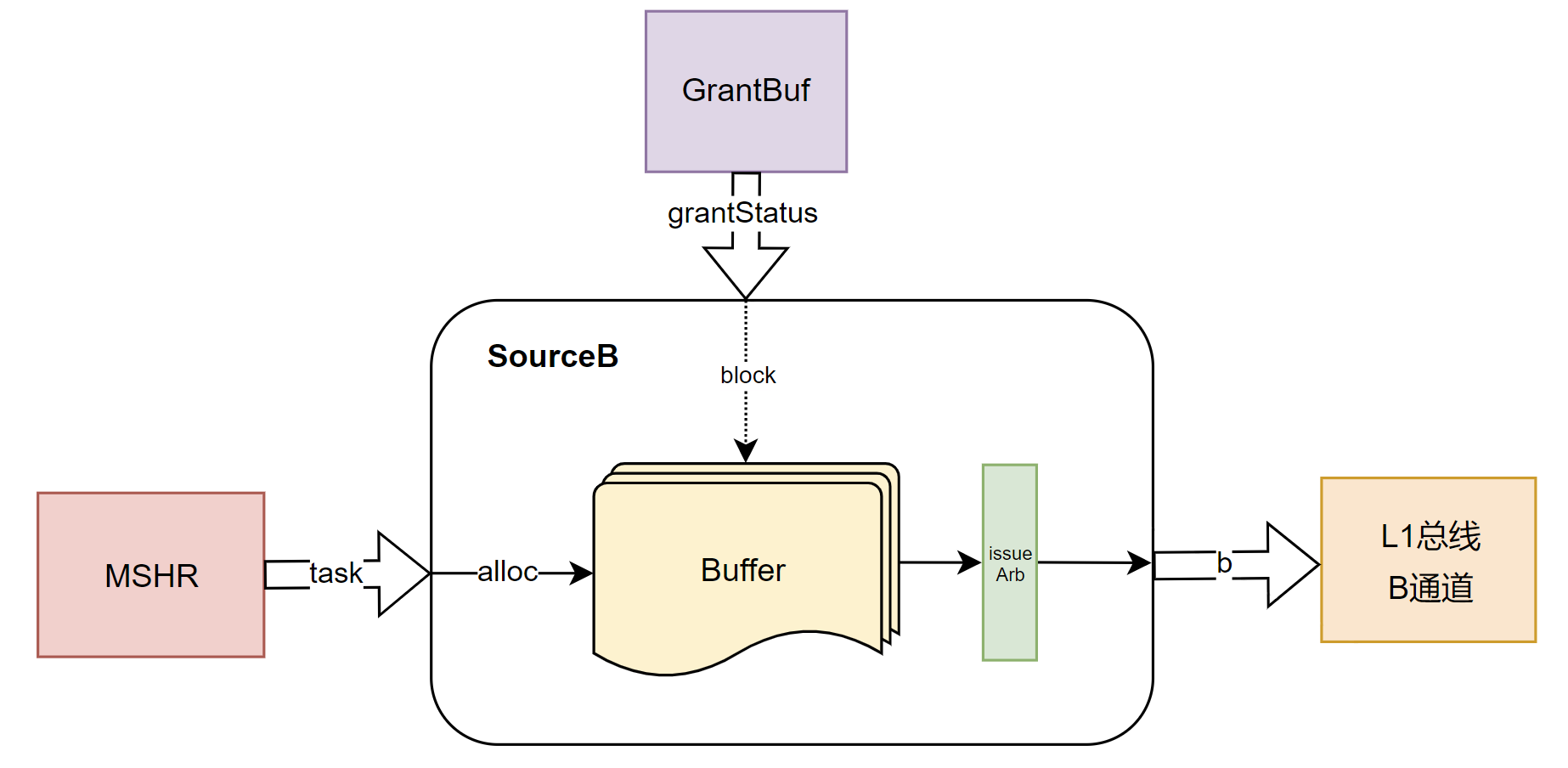
* + 1. SourceB
       1. 功能

*简要描述该模块实现的功能*

SourceB把来自MSHR的任务进行处理，转化总线请求，然后发送给向上的总线B通道。如果新进入的task发现，当前 GrantBuf 里有同地址的【已经发送Grant但是没有收到GrantAck】请求，则需要阻塞task发送，直到该请求收到GrantAck。

* + - 1. 整体框图

*要求如4.1*



* + - 1. 接口列表

*只需要写IO信号*

* + - 1. 接口时序

S0：MSHR请求握手，进入SourceB，为其分配一个 Buffer 项；

如果与GrantBuffer中同地址没收到GrantAck的请求冲突，则在 Buffer 中等待；

S1：如果没有冲突，在S1发送给L1总线B通道。

处理结束。

* + - 1. 关键电路

无。

* + 1. SourceC
       1. 功能

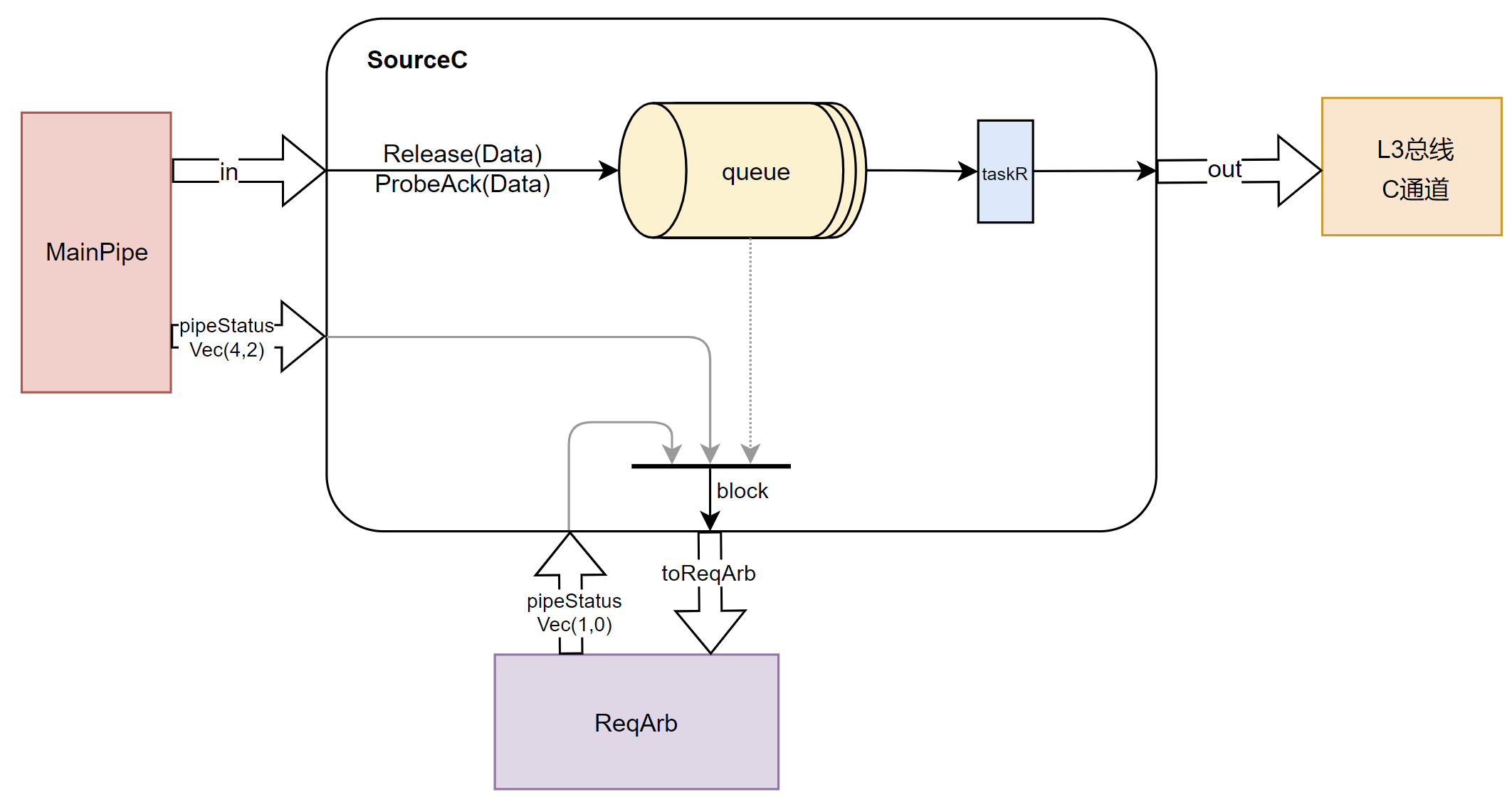
*简要描述该模块实现的功能*

SourceC接收来自MainPipe的任务，并转化为总线任务，向下级L3总线C通道转发。

另外，SourceC还会根据【流水线S1~S5级状态 + 内部 queue 状态】，来向ReqArb给出【请求入口的阻塞信息】，进行反压控制。

* + - 1. 整体框图

*要求如4.1*



* + - 1. 接口列表

*只需要写IO信号*

* + - 1. 接口时序

S0：入口in握手，任务进入queue；

S1：（如果任务在queue队头，且taskR为空）进入taskR，（若含数据）同时发送第一个数据beat；

S2：发送第二个数据 beat。

请求完成。

* + - 1. 关键电路

反压控制逻辑为：

如果【s1~s5级valid且channel=A/B的请求数量】+【SourceC中queue队列内请求数量】≥【SourceC中queue队列的容量】，则阻塞B请求进入流水线；

如果【s1~s5级valid且channel=A/B的请求数量】+【SourceC中queue队列内请求数量】≥【SourceC中queue队列的容量 - 1】，则阻塞MSHR任务进入流水线。

*（因为MSHR有第0级，所以要减一）*

*（流水线task的channel表示该任务进入CoupledL2时，是哪个总线通道；所以channel=A的任务可能是Acquire / MSHR Refill / MSHR Release）*

queue的出队条件为taskR空，所以如果这一拍发送了taskR的最后一个beat，下一拍会标记taskR空并安排出队，直到再下一拍才能继续通过taskR向总线发送下一个请求。

*（taskR相当于size=1且没有pipe的queue，会使最大吞吐减半）*

* + 1. GrantBuffer
       1. 功能

*简要描述该模块实现的功能*

GrantBuf接收来自MainPipe的任务，并根据任务类型进行转发。主要分为：

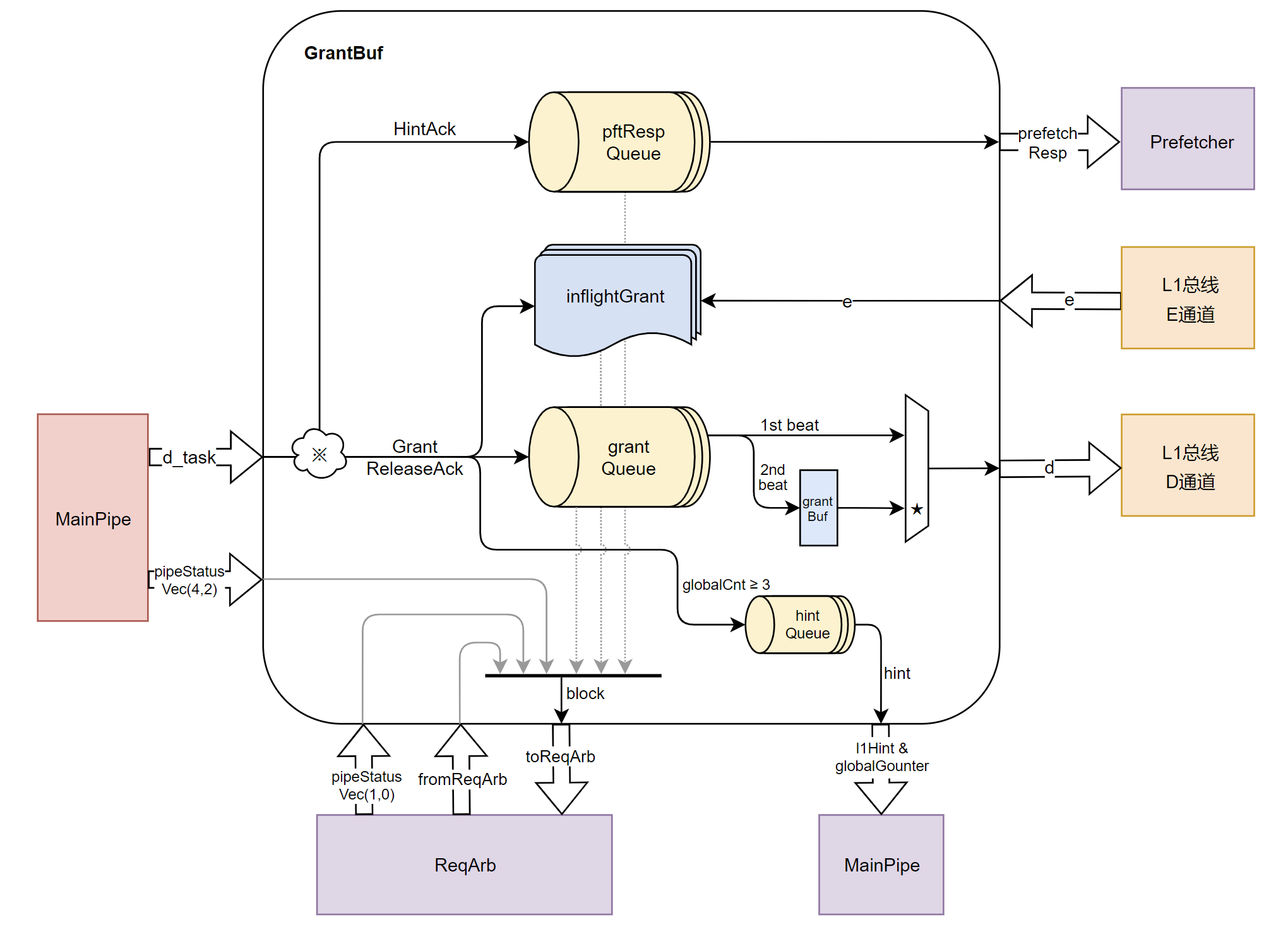
1. 预取响应（opcode = HintAck），会安排其进入预取响应队列 pftRespQueue，按FIFO顺序向预取器发出。
2. D通道响应（opcode = Grant/GrantData/ReleaseAck），会安排其进入grantQueue，按FIFO顺序向总线D通道发出；  
   如果是Grant/GrantAck，则还需要将其信息进入到inflightGrant缓冲区（表明Grant已发送但是还没收到GrantAck），等待L1返回GrantAck后再将其10时21分信息清除掉。
3. 融合请求（task.mergeA = true），同时执行①和②。

另外，GrantBuf 还会根据【流水线入口信息 + 流水线S1~S5级状态 + 内部 pftRespQueue，inflightGrant，grantQueue 状态】，来向ReqArb给出【请求入口的阻塞信息】。

最后，GrantBuf包含提前唤醒信号Hint相关逻辑。它维护了一个globalCounter，记录GrantBuf中还剩多少个beat需要发送。因为设计中Hint要早于Grant三拍发出，所以当globalCounter≥3时，我们就将刚进入的请求加入hintQueue，然后向MainPipe中的CustomL1Hint模块反馈globalCounter和要发送的Hint信息。

* + - 1. 整体框图

*要求如4.1*



* + - 1. 接口列表

*只需要写IO信号*

* + - 1. 接口时序

S0：MainPipe任务握手，进入GrantBuf，根据请求类型；

如果是HintAck，且pftRespQueue为空，则当拍可以发送到预取器；

S1：如果是Grant/ReleaseAck，且grantQueue为空，则在下一拍S1发送到总线；

处理结束。

阻塞的计算是当拍完成（根据当拍传入的流水线信息+内部信息，当拍给出阻塞情况）

* + - 1. 关键电路

pftRespQueue是flow的（可以当拍入队当拍出队），而grantQueue和hintQueue不是flow的。

对于包含一个beat的Grant/ReleaseAck，从grantQueue出队直接向总线发出。

对于包含两个beat的GrantData，在从grantQueue出队的时候，第一个beat的数据直接向总线发出，同时将第二个beat的数据存入grantBuf；接下来优先发送grantBuf里的数据。grantBuf发送完后，可以出队下一个grantQueue元素。

我们采用反压控制，详见SourceC。

阻塞条件：

* A入口：利用流水线S1~S5 A/C通道来统计反压，同SourceC
* B 入口：B请求地址在inflightGrant中
* C入口：利用流水线S1~S5 A/C通道来统计反压，同SourceC
* MSHR入口：利用流水线S1~S5 A/C通道来统计反压，但是不能大于容量-1，同SourceC
  1. RequestArb（ReqArb）
     1. 功能

*简要描述该模块实现的功能*

RequestArb 负责仲裁哪个请求应该进入主流水线，同一拍只允许一个请求进入。它会根据MSHRCtl，MainPipe，SourceC，GrantBuf传入的信息，判断哪个入口需要被阻塞。如果有多个入口都可以进入，则各入口仲裁优先级为 MSHR 任务 > C 请求 > B 请求 > A 请求。

RequestArb包含 CPL2 主流水线的前两级（S1, S2）。请求在S1进入主流水线，并向目录发起读请求。请求在S2向数据缓存区（ReleaseBuf 和 RefillBuf）发起读请求。

**关于入口的阻塞逻辑：**

TileLink 规定请求的优先级为 Release > Probe > Acquire。按照请求到达的先后顺序，MSHR 里发出的任务的优先级应当高于从通道控制器收到的请求。综合后，ReqArb 仲裁优先级为 MSHR > C > B > A。

又因为流水线是非阻塞式的，所以我们必须精细地安排其入口条件，

确保进入流水线的请求要么能够直接在流水线上被处理，要么能够为其分配一个 MSHR，不能出现需要 MSHR 但是 MSHR 满了的情况。

为此我们采用反压控制逻辑，当检测到【流水线前 3 级包含的请求数 + 已占 MSHR 数量 ≥ MSHR 总数】的时候，我们就阻止任何通道请求进入，这样能够保证流水线里的请求仍能有 MSHR。

同样的逻辑，我们也需要保证 SourceD 中的缓冲区不会溢出，即【流水线内 5 级包含的 Acquire/Release 请求数 + 已占缓冲区项数 ≥ 缓冲区总项数】（因为 Acquire 的响应和 Release 的响应都要走 SourceD）。

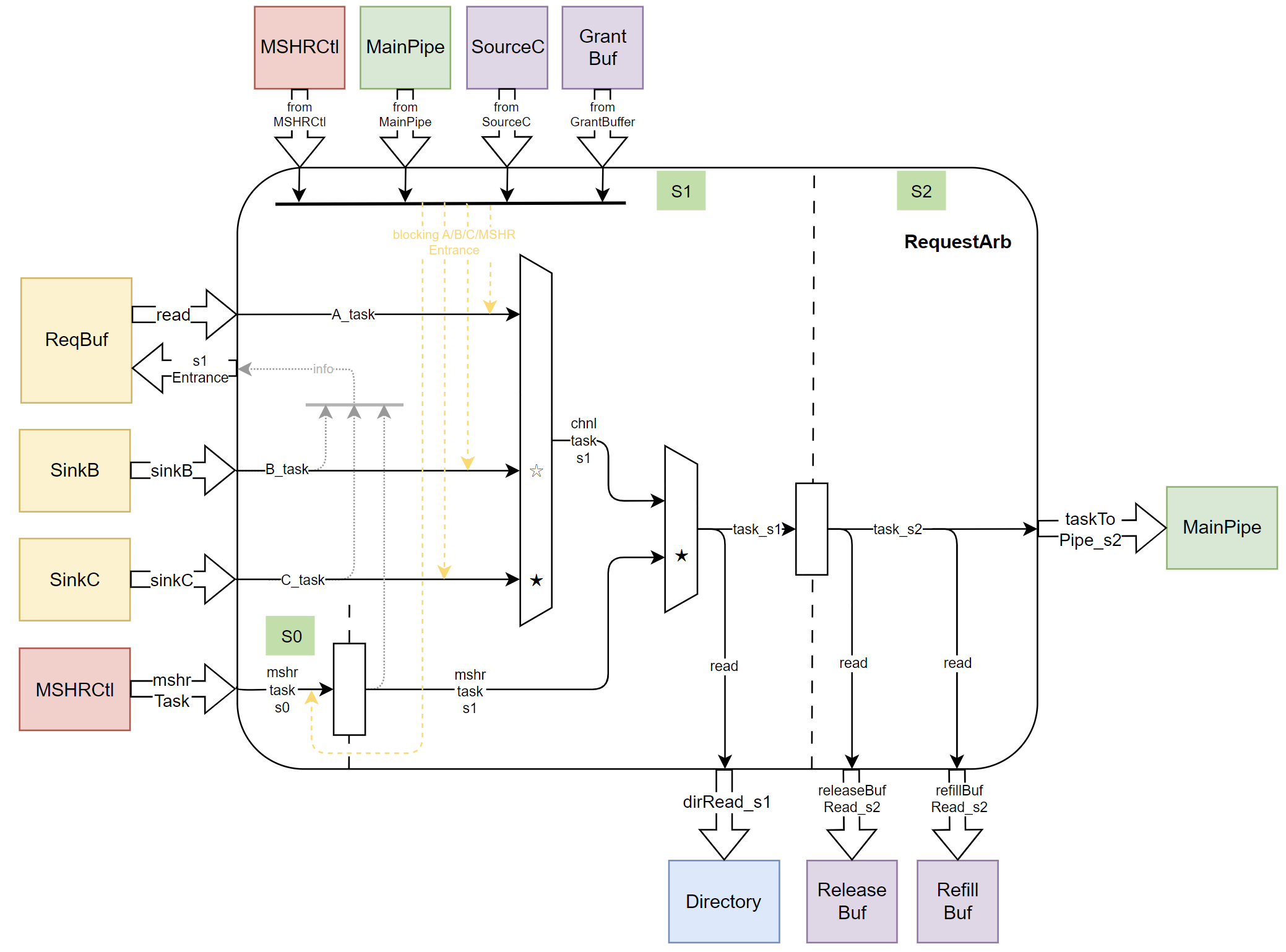
另外，因为我们的目录是在 S1 读，S3 写入，我们还需要防止写后读冲突，保证顺序（即 S1 的读必须能读到 S2/S3 将要写入的数据）。为了保证这一点，我们会进行 set 阻塞，如果入口新请求与流水线 S2/S3 请求的 set 相同，且 S2/S3 要写目录的话，我们就需要阻塞新请求进入。

之所以是按 set 阻塞，是因为只有同 set 内的请求会相互影响（因为替换只在 set 内进行），不同 set 间互不影响，所以只按 set 阻塞。

最后，如果入口新请求与流水线/MSHR 中某请求的地址完全相同（即要对同一个数据进行操作）,我们都进行阻塞，为了保证对同一个数据操作的一致性。

* + 1. 整体框图

*要求如4.1*



* + 1. 接口列表

*只需要写IO信号*

* + 1. 接口时序

S1：通道A/B/C请求/MSHR任务握手，进入RequestArb；

当拍给Directory发送读请求；

S2：请求向ReleaseBuf 和 RefillBuf发送读请求；

请求进入MainPipe；

处理结束。

* + 1. 关键电路

如果当拍接受了一条B/C请求/MSHR任务，RequestArb会通过s1Entrance通知RequestBuf。

* + 1. 三级模块设计

*内容和要求同二级模块设计，如果还有4级模块需要一级级往下写。*

* 1. Directory
     1. 功能

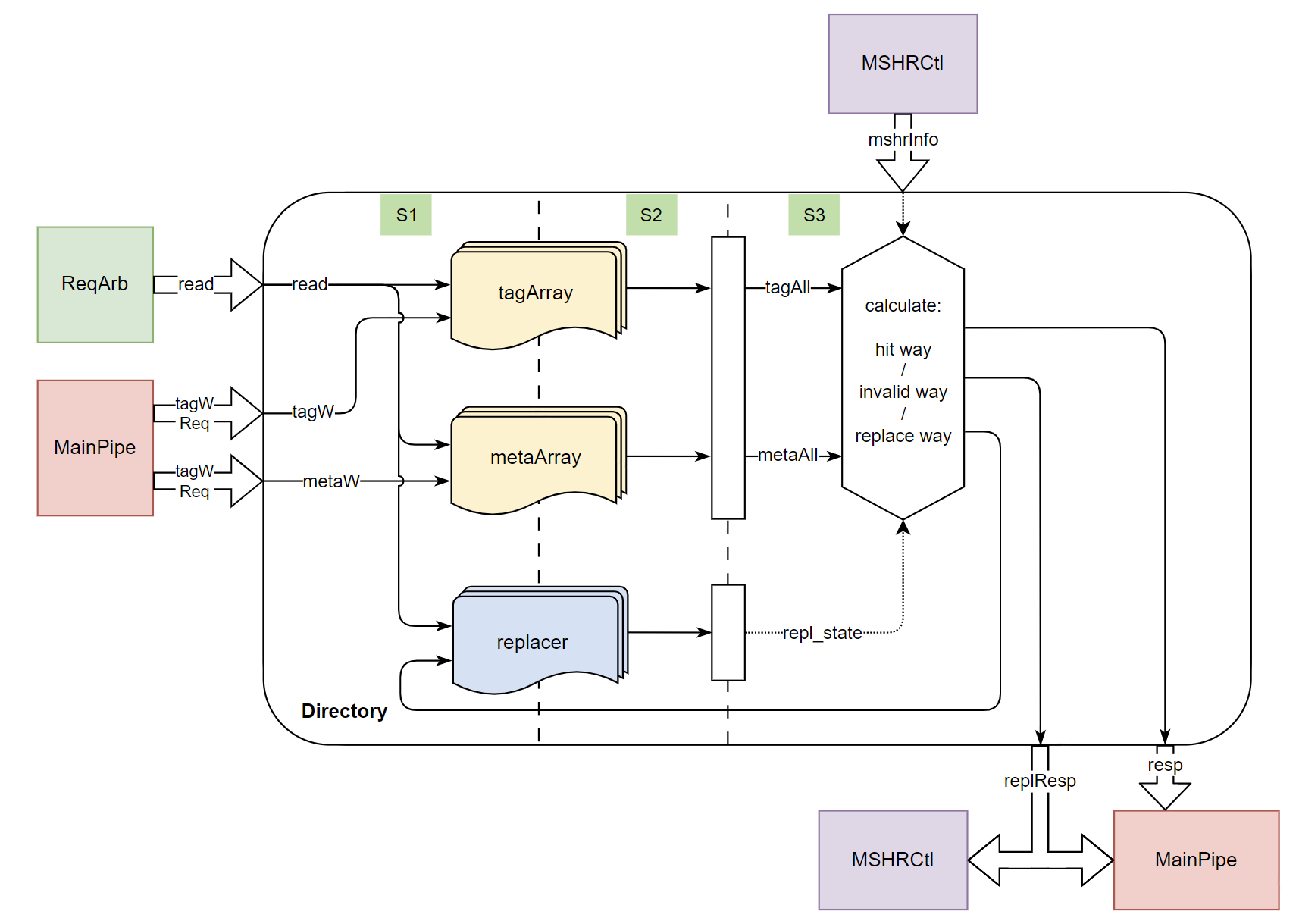
*简要描述该模块实现的功能*

CPL2采用基于目录的一致性实现方式，利用目录记录L2内数据块的元数据信息。

Directory 会根据读请求的tag&set，查找L2是否存储该数据块（是否命中）。如果命中，则返回该数据块的元数据信息。如果缺失，则挑选一个无效的路/被替换的路，返回该路数据的元数据信息。请求处理完成后，会将新的目录信息写入Directory进行更新。

* + 1. 整体框图

*要求如4.1*



* + 1. 接口列表

*只需要写IO信号*

* + 1. 接口时序

**读请求：**

接口时序第一级对应主流水线的S1级；

S1：读请求握手，当拍发送到SRAM端口；

S2：SRAM返回读数据，锁存；

S3：利用锁存数据计算命中/缺失替换信息，从resp和replResp返回数据；

请求完成。

**写请求：**

接口时序第一级对应主流水线的S3级；

S3：写请求握手，当拍发送到SRAM端口；

请求完成。

* 1. MainPipe
     1. 功能

非阻塞的主流水线负责接收和处理所有与Directory和DataStorage读写相关的通道请求和MSHR请求，并生成响应与分配MSHR。

***接收请求：***MainPipe在s2接收来自RequestArb的经过仲裁的A/B/C通道请求和MSHR请求。

***处理请求的流水线划分：***见表格

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **流程** | **MSHR Task** | **Opcode** | **s1** | **s2** | | **s3** | | | | | **s4** | **s5** | | |
| **读目录** | **读 refillBuf** | **读 releaseBuf** | **目录**  **结果** | **分配MSHR** | **写目录** | **读写数据** | **C/D** | **C/D** | **C/D** | **写 refillBuf** | **写 releaseBuf** |
| **A 请求**  **进入** | 0. | AcquireBlock | √ |  |  | 不命中 | √ |  |  |  |  |  |  |  |
| 命中 |  | √ | 读 |  |  | D |  |  |
| 命中+alias | √ |  | 读 |  |  |  |  | √ |
| 0. | AcquirePerm | √ |  |  | 不命中 | √ |  | 读 (L2 有 B) |  |  |  | √ (L2 有 B) |  |
| 命中 |  | √ |  | D | D | D |  |  |
| 命中 alias | √ |  | 读 |  |  |  |  | √ |
| 0. | Get | √ |  |  | 不命中 | √ |  |  |  |  |  |  |  |
| 命中 (无 client) |  |  | 读 |  |  | D |  |  |
| 命中 (有 client) | √ |  | 读 |  |  |  |  | √ |
| 0. | Hint | √ |  |  | 不命中 | √ |  |  |  |  |  |  |  |
| 命中 |  |  |  | D | D | D |  |  |
| **B 请求**  **进入** | 0. | Probe | √ |  |  | 不命中 |  |  |  |  |  |  |  |  |
| 命中 (无 client) |  | √ | 读 |  |  | C |  |  |
| 命中 (有 client) | √ |  | 读 |  |  |  |  | √ |
| **C 请求**  **进入** | 0. | ReleaseData | √ |  |  | - |  | √ | 写 | D | D | D |  |  |
| 和被替换的数据块发生嵌套 (rporbe - release) ，则需要写入 releaseBuf | | | | | | |  | √ (另有端口) |
| 已 Grant 但还没写入 DataStorage，就收到了上层 release，则需要写入 refillBuf | | | | | | | √ (另有端口) |  |
| **MSHR Grant** | 1. | Grant[Data] AcccessAckData HintAck | √ | √ |  | 选到忙路 | ***重发*** | | | |  |  |  |  |
| 选到无效路 |  | √ | 写【重填数据】 | D | D | D |  |  |
| 选到有效路 |  | √ | 读【被替换块】 | D | D | D |  | √ |
| **MSHR Release** | 1. | Release[Data] |  | √ | √ | 同 MSHR Grant 选到有效路 |  |  | 写【重填数据】 | C | C | C |  |  |
| **MSHR ProbeAck** | 1. | ProbeAck[Data] |  |  | √ | - |  | √ |  | C | C | C |  |  |

* + 1. 图示

       描述已自动生成整体框图
    2. 接口列表
    3. 接口时序

所有接口信息的接收和发送都是valid有效，sourceC和sourceD通道均有反压控制，因此不需要valid-ready握手。

接口taskInfo\_s1/ fromReqArb在对应请求的s1时刻进入MainPipe。taskFromArb\_s2在请求的s2时刻进入MainPipe。dirResp\_s3/ replResp/ toMSHRCtl/ fromMSHRCtl/ bufRead/ bufResp/ refillBufResp\_s3/ releaseBufResp\_s3/ toDS/ metaWReq/ tagWReq接口信息在对应请求的s3时刻有效。toSourceC/ toSourceD则在s3/s4/s5都可能有效。releaseBufWrite/ refillBufWrite在s5时刻有效。

* + 1. 关键电路
    2. 三级模块设计
  1. MSHRCtl
     1. 功能

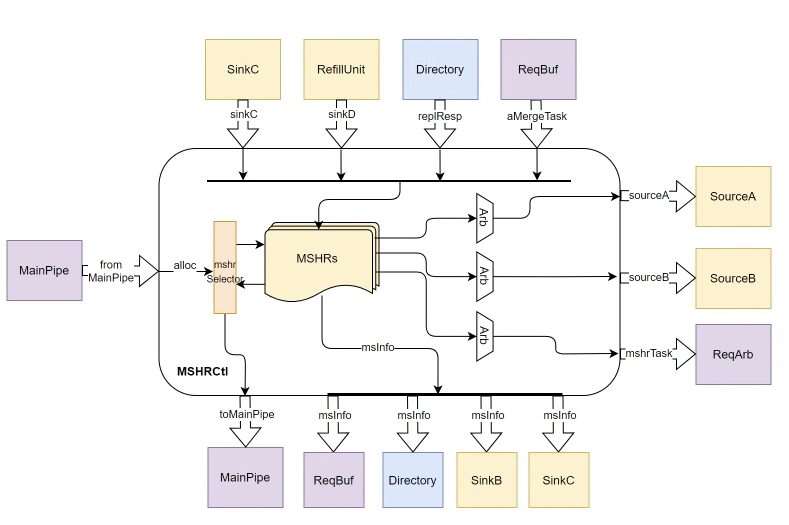
MSHRCtl模块主要负责为请求分配MSHR，将来自sinkC/sinkD/sourceC通道控制器等的信息传入目标MSHR，并对MSHR的请求进行仲裁，发送给对应的sourceA/sourceB通道控制器或者MainPipe（RequestArb）。

***MSHR分配***：当MainPipe发起MSHR分配请求时，MSHRCtl会根据当前MSHRs的占用情况，选出一个空闲的MSHR，将分配请求信息传入对应的MSHR，并将分配结果返回给MainPipe。

***接收响应信息***：当L2接收到来自sinkC/sinkD/sourceC/Replacer的响应以及aMerge请求时，MSHRCtl会将该信息传入响应的目标MSHR。

***MSHR请求仲裁***：对于MSHRs生成的task请求，MSHRCtl通过FastArbiter仲裁器仲裁后，发往sourceA/sourceB/MainPipe。

* + 1. 整体框图

**

* + 1. 接口列表
    2. 接口时序

MSHRCtl内部均为组合逻辑，所有接口信号都在valid当拍传入和传出MSHRs。

* + 1. 关键电路

关于任务仲裁：

从各个MSHR发出的任务要经过FastArbiter仲裁给到通道控制器，各个通道控制器之间是独立的，不存在优先级关系，而各MSHR任务的选择在FastArb中是类round robin的算法。

关于返回结果如何选择MSHR：

sinkC/sinkD/sourceC/Replacer的响应结果以及aMergeTask都和所有MSHR相连。sinkD/sourceC/Replacer/aMerge 都通过请求中的mshrId标识号来确定返回到哪个MSHR，sinkC则通过比较MSHR中的set, tag, valid等信息与通道控制器中的是否一致，得到OH独热编码找到对应的MSHR。

* + 1. 三级模块设计

*MSHRCtl模块内实例化了MSHR子模块，由于该模块较为重要，所以作为二级模块单独描述。*

* 1. MSHR
     1. 功能

当L2发生缓存miss或者命中但需要更改meta权限时，会为请求分配一项MSHR，用于记录请求所处的状态，并接收通道控制器与目录的响应，根据当前状态发出对应的控制请求。当所有控制请求完成后，释放MSHR。

***接收请求：***MSHR会从io.alloc接口接收MSHRCtl分配的请求，并根据MainPipe给出的初始化信息设置MSHR中的请求内容req、请求状态state、读目录结果dirResult。

***发送子任务：***根据请求内容和读目录结果，结合MainPipe给出的初始状态信息，完成一系列子请求。MSHR将这些要调度的请求和要等待的应答用一系列状态寄存器记录， s\_\*表示要调度的请求，w\_\*表示要等待的应答，这些寄存器都是低有效的，即默认值为true，值为false时表示需要执行。s\_\*状态寄存器与对应的请求如下：

s\_acquire – 需要向下发起Acquire请求，获取L3的数据或权限。

s\_pprobe – 需要向上发起Probe请求，修改L1数据的权限，由probe请求引起。

s\_rprobe – 需要向上发起Probe请求，修改L1数据的权限，由替换引起。

s\_release – 需要向下发起Release请求，将数据块写回L3。

s\_probeack – 需要向下发出ProbeAck响应，响应L3来的probe请求。

s\_refill – 需要向上发出Grant响应，完成数据重填并将数据返回给L1/响应预取器。

s\_retry – 当访问目录存在冲突时，需要MSHR重新发起请求。

***等待接收响应：***在完成子请求的过程中，需要等待上下游节点对发送请求的应答，w\_\*窗台寄存器与对应的等待响应如下：

w\_rprobeackfirst – 等待由替换引起的ProbeAck响应的第一个beat数据。

w\_rprobeacklast – 等待由替换引起的ProbeAck响应的最后一个beat数据。

w\_pprobeackfirst – 等待由probe请求引起的ProbeAck响应的第一个beat数据。

w\_pprobeacklast – 等待由probe请求引起的ProbeAck响应的最后一个beat数据。

w\_pprobeack – 等待由probe请求引起的ProbeAck响应。

w\_grantfirst – 等待Grant响应的第一个beat数据。

w\_grantlast – 等待Grant响应的最后一个beat数据。

w\_grant – 等待Grant响应。

w\_releaseack – 等待ReleaseAck响应。

w\_replResp – 当MSHR请求在MainPipe读目录结果为miss时，需要等待选择替换路的结果。

***释放MSHR：***当所有事件都完成后，该项MSHR就会被释放，即req\_valid拉低。

* + 1. 整体框图

见MSHRCtl

* + 1. 接口列表
    2. 接口时序

各task请求信息采用valid-ready握手方式完成发送。alloc和各resp响应以及nestedwb为valid有效。

status的valid表示MSHR中有请求且请求有效。

注意status的will\_free，它会在MSHR释放的前一拍拉高，释放后io\_status\_valid = false。

* + 1. 关键电路

关于嵌套请求：

C嵌套A —— 当L2首先收到对地址X的Acquire，miss且需要替换地址Y，或者为alias，那么A请求会进入 MSHR，同时读DS中Y的数据写入ReleaseBuffer。如果进入MSHR后L2又收到了L1对Y的ReleaseData请求，那么MainPipe会优先处理C

请求而非替换引起的Y rProbe，同时这个C请求可以嵌套进入当前MSHR，将嵌套C请求要写入的新数据ReleaseData更新到该MSHR对应的ReleaseBuffer中，从而在完成该MSHR任务时保证向下Release的是最新的数据。

C嵌套B —— 与C嵌套A类似，当L2在MSHR中处理Probe L1请求时，L1主动Release，那么C请求的数据需要通过MainPipe的nestedwbData写入当前MSHR项对应的releaseBuf。

B嵌套A —— 当A请求在MSHR中等待L3的Grant时，先收到了L3对同地址的Probe请求。那么MainPipe会在s3处理Probe请求时检查有无相同地址的MSHR，如果有则将写目录的结果更新到对应MSHR。

（或许还有要补充的）

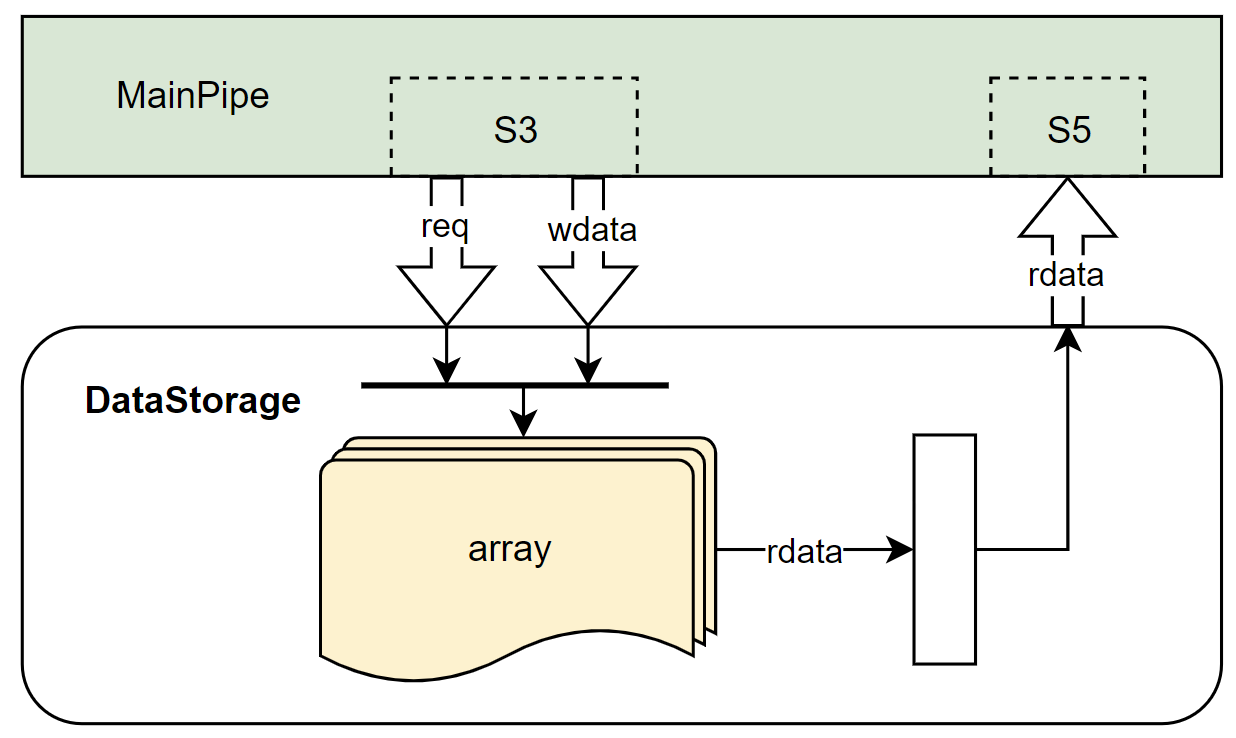
* + 1. 三级模块设计
  1. DataStorage
     1. 功能

*简要描述该模块实现的功能*

DataStorage模块负责Cache数据的存储与读写，采用单端口 SRAM 构建。DataStorage 只与 MainPipe 的 S3 级进行交互。MainPipe的S3 级每拍可以向 DataStorage 发送一个请求（读请求/写请求）。

* + 1. 整体框图

*要求如4.1*



* + 1. 接口列表

*只需要写IO信号*

* + 1. 接口时序

（S0 对应主流水线的 S3 级）

S0：请求握手，当拍也发送到 SRAM 端口；

如果是写请求，还需要提供写数据；写请求在S0已经完成；

S1：SRAM 返回读数据，锁存；

S2：数据从【读响应端口】返回；读请求完成。

* + 1. 关键电路

DataStorage 采用全频 SRAM 搭建，且不设置 Multi-Cycle Path。

读写请求一次均传输一个 Cache Block（512位）的数据。

* 1. RequestBuffer（ReqBuf）
     1. 功能

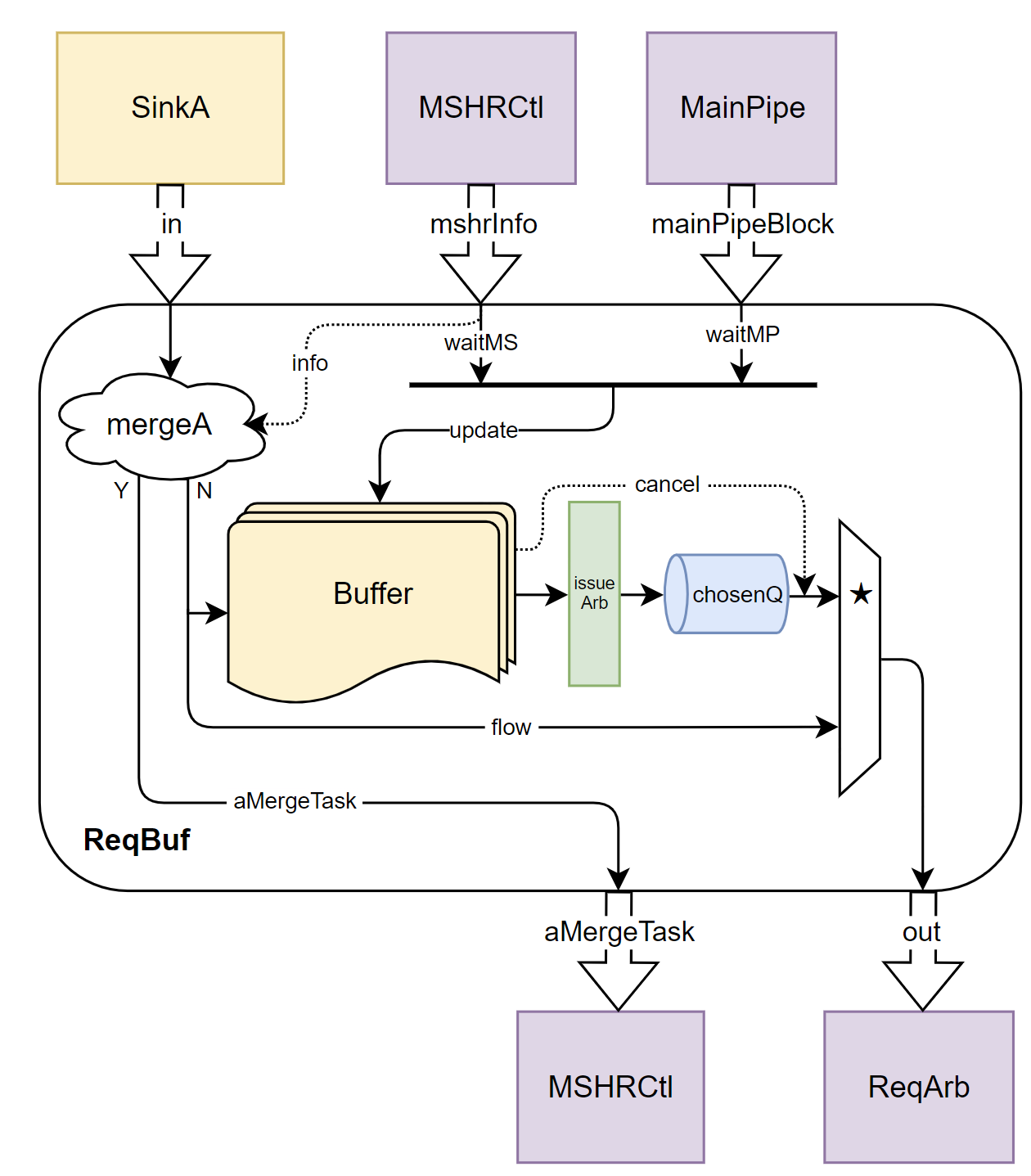
*简要描述该模块实现的功能*

Request Buffer用来缓冲那些暂时需要阻塞的 A 通道请求，同时让满足释放条件/不需要阻塞的 A 通道请求先进入主流水线。Request Buffer 可以防止需要阻塞的 A 请求堵住流水线入口，从而避免对后续请求的影响，提高了缓存的处理效率。

另外，如果有新到达的 Acquire 与 MSHR 中正在处理的预取请求地址相同，则可以进行请求融合，将 Acquire 的信息直接传到对应 MSHR 中，让 MSHR 处理完成后同时回复 L1 Acquire，从而加快了 Acquire 的处理流程，并减少了对 ReqBuf 和 MSHR 的占用。

* + 1. 整体框图

*要求如4.1*



* + 1. 接口列表

*只需要写IO信号*

* + 1. 接口时序

S0：in 握手，新请求进入 ReqBuf；

同时新请求的tag/set 通过 io\_ATag/io\_ASet 送给ReqArb，然后再送入MainPipe进行阻塞判断，最后通过 mainPipeBlock 将阻塞信息送回 ReqBuf；

如果判断可以进行请求融合，则当拍直接通过 aMergeTask 给到 MSHRCtl；

（如果当前buffer内没有请求）out 握手，请求离开 ReqBuf；（如果有则让buffer内请求先发送）

处理完成。

**其余**

MSHRInfo：当前 MSHR 状态，信号独立

s1Entrance：主流水线入口 S1 信息，信号独立

* + 1. 关键电路

***TODO***

Request Buffer 的设计类似于 CPU 中的保留站/发射队列，位于 SinkA 和 ReqArb之间，从 SinkA 接收新请求，并向ReqArb发射（进入ReqArb即进入了主流水线）。

Request Buffer 包含

如果 ReqBuf满，则不会再从 SinkA 接收新请求。

当新到来的请求需要阻塞时

**阻塞条件：**

1. 和 MSHR 中的请求

另外，还可以

* 1. RefillBuffer
     1. 功能

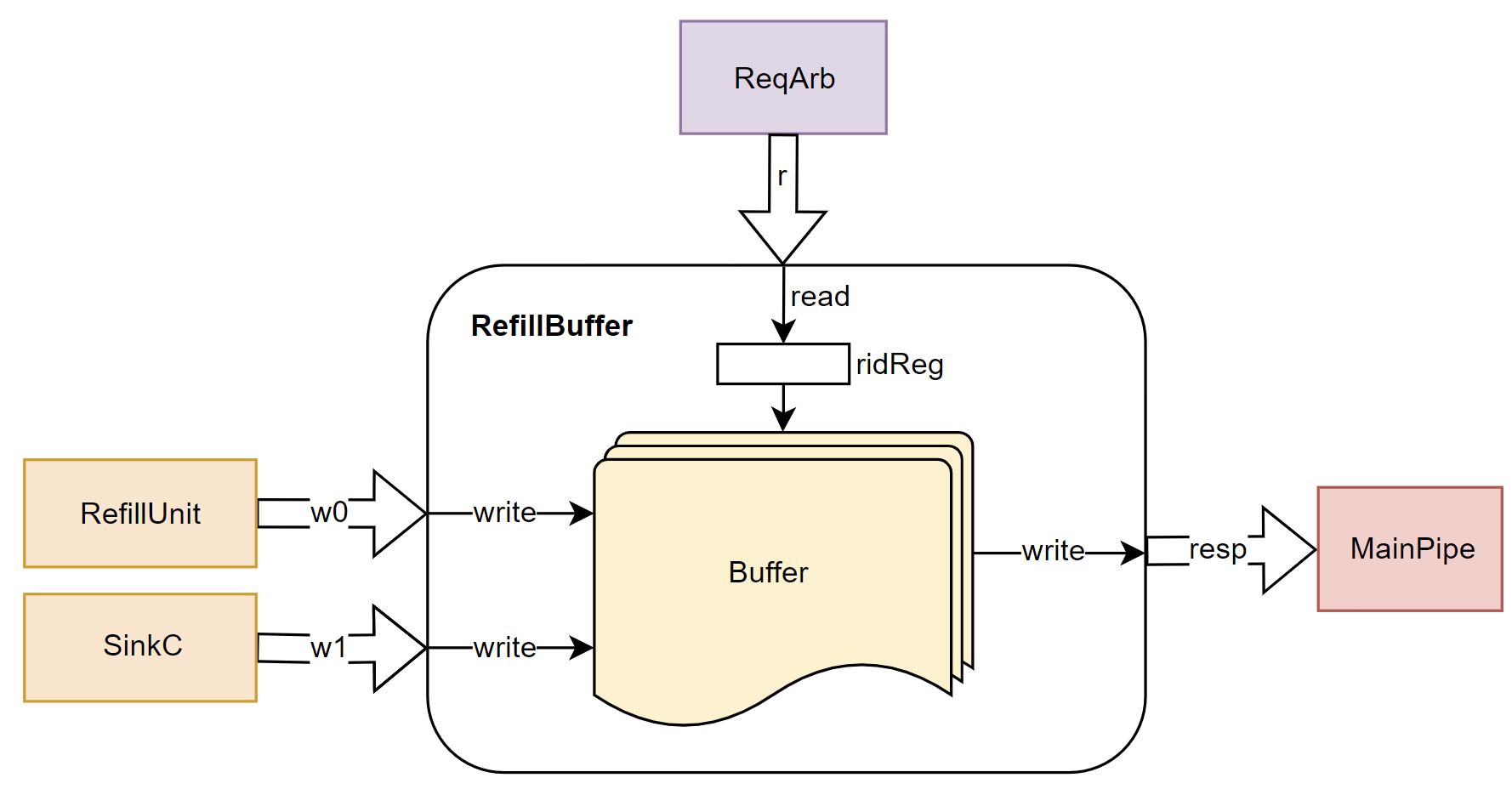
*简要描述该模块实现的功能*

用于暂存从下层缓存重填上来的数据。包含一个寄存器组，有两个写口和一个读口，读写口包含id用来索引是哪个寄存器。

两个写口可以独立写入。如果在同一拍中两个写口索引相同（要写入同一个数据项），则写口0的优先级高于写口1。

* + 1. 整体框图

*要求如4.1*



* + 1. 接口列表

*只需要写IO信号*

* + 1. 接口时序

**写请求：**

S0：写请求有效，当拍发到寄存器堆；写请求完成；

**读请求：**

S0：读请求有效，将其索引延迟一拍；

S1：利用延迟后的索引找到数据，通过resp返回给MainPipe；读请求完成。

* + 1. 关键电路

无。

* 1. ReleaseBuffer
     1. 功能

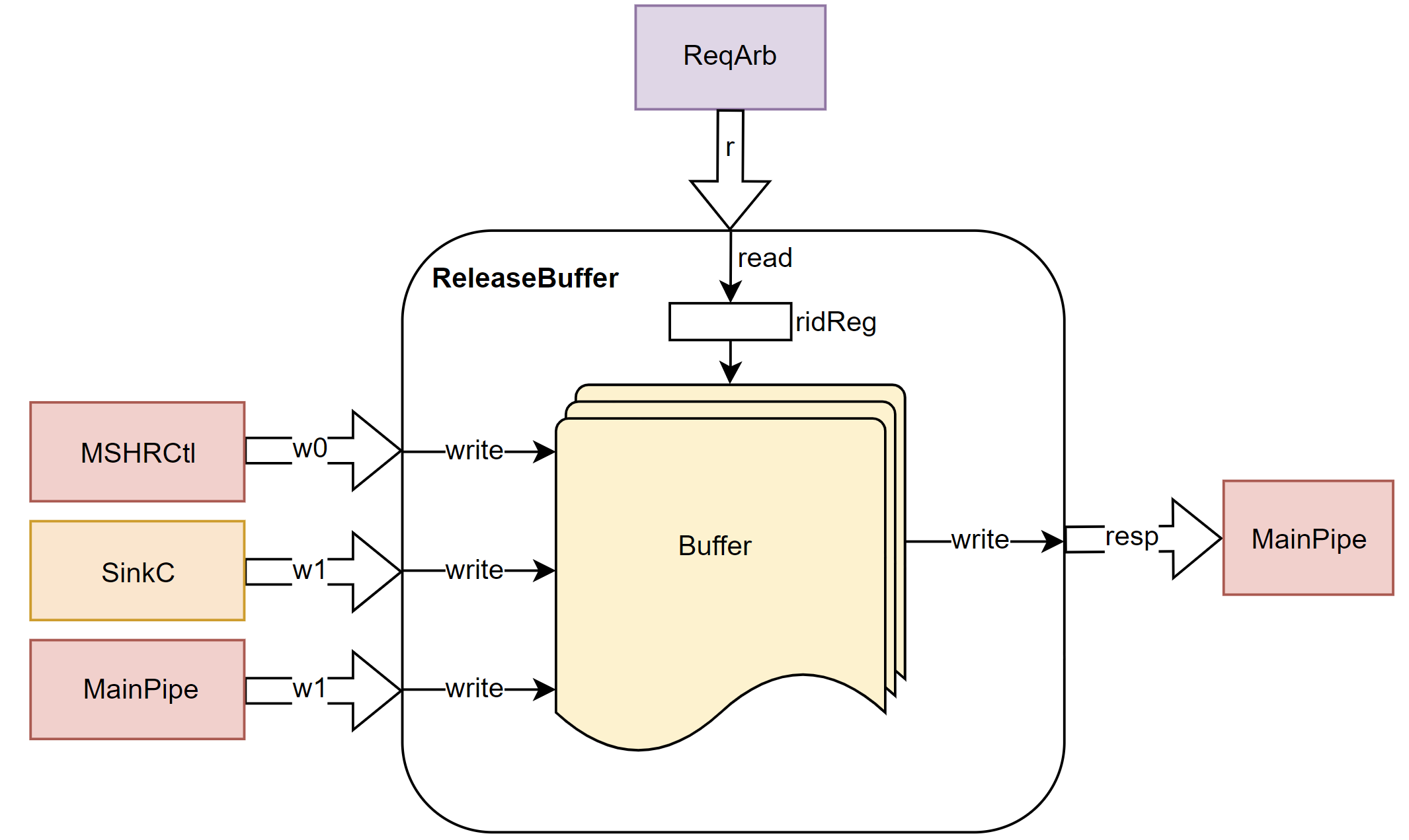
*简要描述该模块实现的功能*

用于暂存要释放给下层缓存的数据。包含一个寄存器组，有三个写口和一个读口，读写口包含id用来索引是哪个寄存器。

三个写口可以独立写入。如果在同一拍中有多于一个写口有效且索引相同（要写入同一个数据项），则优先级 写口0＞写口1＞写口2。

* + 1. 整体框图

*要求如4.13*



* + 1. 接口列表

*只需要写IO信号*

* + 1. 接口时序

**写请求：**

S0：写请求有效，当拍发到寄存器堆；写请求完成；

**读请求：**

S0：读请求有效，将其索引延迟一拍；

S1：利用延迟后的索引找到数据，通过resp返回给MainPipe；读请求完成。

* + 1. 关键电路

无。

* 1. Prefetcher
     1. 功能

*简要描述该模块实现的功能*

Prefetcher模块接收来自L2 Cache的预取训练数据以及对应的TP meta数据，并将训练数据分别发送给发送给BOP模块以及TP模块进行预取训练。同时，Prefetcher模块将来自L1 DCache的预取请求发送给PrefechReceiver模块，并进行筛选。

Prefetcher模块接收来自BOP模块、TP模块以及PrefechReceiver模块的预取请求，并将预取请求缓存至PrefechQueue中。当外部总线允许接收预取请求时，Prefetcher模块从PrefetchQueue中取出预取请求并发送。Prefetcher模块接收预取请求的响应，并发送至BOP模块以及TP模块。

Prefetcher模块接收TP模块完成训练后更新的TP meta数据，当外部总线允许时，发送更新的TP meta数据。

它包含两部分：

**一、训练数据：**

Prefetcher模块可以持续接收来自L2 Cache的训练数据（L1 DCache的预取请求）。训练数据包括虚拟地址vaddr、对应的Cache tag和set、训练数据来源、是否命中L2 Cache、地址是否需要TLB翻译、是否经过预取、预取请求来源等信息。Prefetcher接收到训练数据后直接将训练数据发送给BOP模块进行预取训练。同时，Prefetcher模块接收对应TP meta数据，与来自L2 Cache的训练数据一同发送给TP模块进行预取训练。

同时。Prefetcher模块接收TP模块完成训练后更新的TP meta数据，当外部总线允许时，发送更新的TP meta数据。

**二、预取请求：**

**2.1 预取请求生成**

Prefetcher模块可以持续接收来自BOP模块和TP模块的预取请求。预取请求以一个缓存块为单位，请求信息包括缓存块对应tag、set、是否需要TLB翻译、来源、以及预取来源等信息。

同时，Prefetcher模块持续接收来自L1 Cache的预取请求，并将预取请求发送至PrefetchReceiver模块接收。请求信息包含预取地址和预取来源，Prefetcher模块将L1 DCache预取器的预取请求中的地址划分为L2 Cache预取请求格式，并确保预取请求来自DCache。

**2.2预取请求缓存**

Prefetcher模块中例化PrefetchQueue模块，通过一个容量为16（inflightEntries？）的循环队列缓存尚未被发送的预取请求。

Prefetcher模块基于L1 DCache > BOP > TP的优先级顺序对同时产生的预取请求进行选择，并存入缓存队列队尾。当缓存队列已满时，删除最老的预取请求（头部），将新的预取请求存入队列。

**2.3预取请求发送**

当外部总线允许Prefetcher模块发起预取请求，且PrefetchQueue中缓存队列非空时，取出缓存队列头部的预取请求并发送。

* + 1. 整体框图

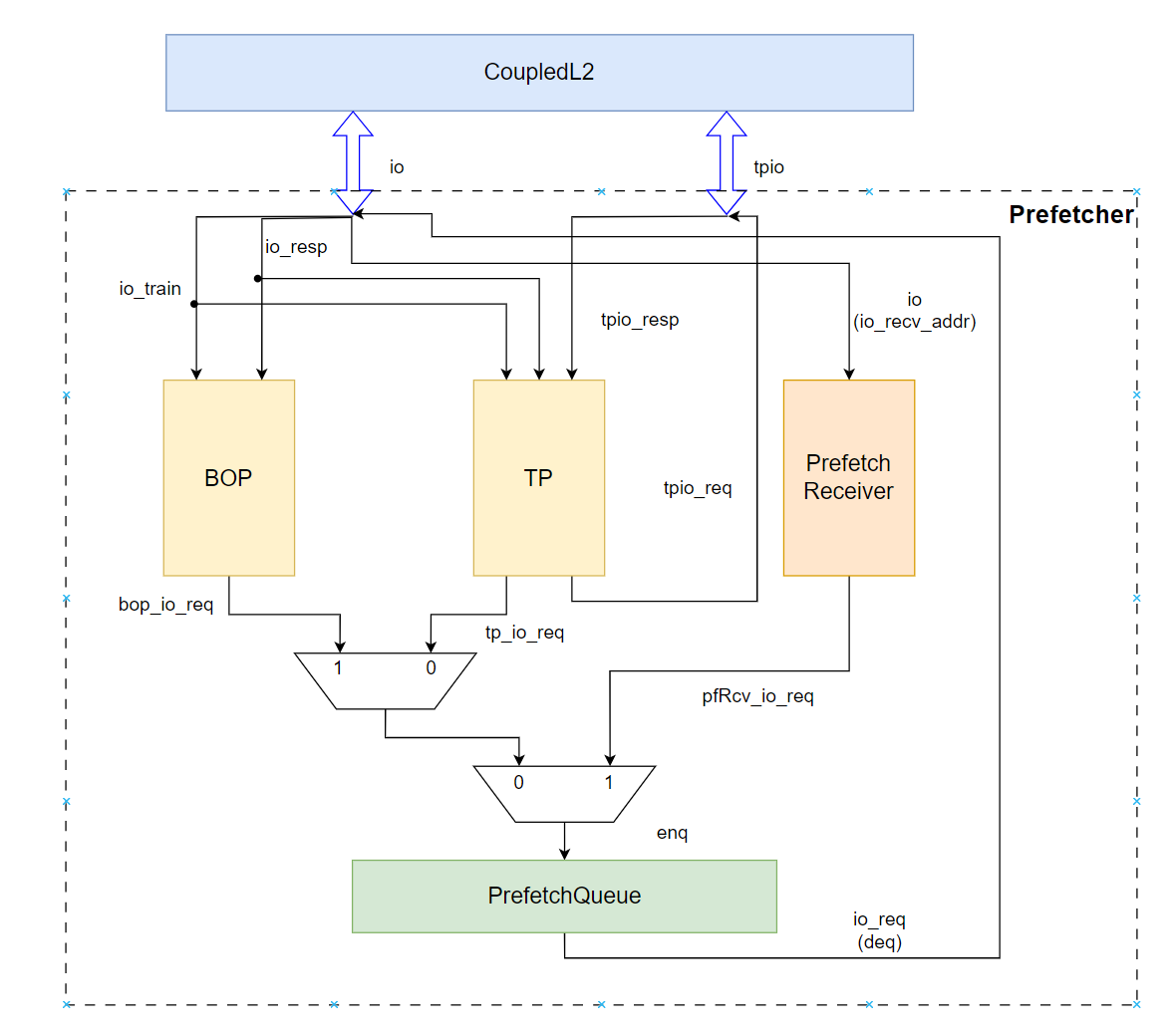
**

图4.1 Prefetcher整体框图

图4.1中，连线箭头表示数据传输方向。连线交汇处存在圆点的，表示相交。PrefechReceiver模块接口类型为PrefechIO，但实际从CoupledL2输入的有效信号仅有io\_recv\_addr（Prefecher模块接口io）。多路选择器中，1对应接口为高优先级输入；0对应接口为低优先级输入。

* + 1. 接口列表

表4.1 Prefetcher接口列表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **信号** | **方向** | **源** | **目的** | **信号组名** | **位宽** | **描述** |
| Clock | input |  |  |  | 1 | 时钟信号 |
| Reset | input |  |  |  | 1 | 异步复位信号 |
| io\_train\_ready | output | Prefetcher | CoupledL2 | io\_train  (PrefetchIO) | 1 | 训练数据可以接收；高有效 |
| io\_train\_valid | input | CoupledL2 | Prefetcher | io\_train  (PrefetchIO) | 1 | 训练数据有效；高有效 |
| io\_train\_tag | input | CoupledL2 | Prefetcher | io\_train  (PrefetchIO) | 26 | 训练数据对应Cache tag |
| io\_train\_set | input | CoupledL2 | Prefetcher | io\_train  (PrefetchIO) | 7 | 训练数据对应Cache set |
| io\_train\_needT | input | CoupledL2 | Prefetcher | io\_train  (PrefetchIO) | 1 | 训练数据是否需要TLB翻译。高需要 |
| io\_train\_source | input | CoupledL2 | Prefetcher | io\_train  (PrefetchIO) | sourceIdBits | 训练数据来源 |
| io\_train\_vaddr | input | CoupledL2 | Prefetcher | io\_train  (PrefetchIO) | 39 | 训练数据对应Cache set |
| io\_train\_hit | input | CoupledL2 | Prefetcher | io\_train  (PrefetchIO) | 1 | 训练数据是否命中L2 Cache。高命中 |
| io\_train\_prefetched | input | CoupledL2 | Prefetcher | io\_train  (PrefetchIO) | 1 | 训练数据是否预取过。高有效 |
| io\_train\_pfsource | input | CoupledL2 | Prefetcher | io\_train  (PrefetchIO) | 3 | 训练数据预取请求的预取器来源。0：nowhere；1：SMS；2：BOP；3：Stream；4：Stride；5：TP |
| io\_train\_reqsource | input | CoupledL2 | Prefetcher | io\_train  (PrefetchIO) | 4 | 训练数据预取请求来源。  0：nowhere  1：CPUInst  2：CPULoadData  3：CPUStoreData  4：CPUAtomicData  5：L1InstPrefetch  6：L1DataPrefetch  7：PTW  8：Prefetch2L2BOP  9：Prefetch2L2SMS  10：Prefetch2L2Stream  11：Prefetch2L2Stride  12：Prefetch2L2TP  13：Prefetch2L2Unknown  14：Prefetch2L3Unknown  15：ReqSourceCount |
| io\_req\_ready | input | CoupledL2 | Prefetcher | io\_req  (PrefetchIO) | 1 | 预取请求可以被接收；高有效 |
| io\_req\_valid | output | Prefetcher | CoupledL2 | io\_req  (PrefetchIO) | 1 | 预取请求有效；高有效 |
| io\_req\_tag | output | Prefetcher | CoupledL2 | io\_req  (PrefetchIO) | 26 | 预取请求缓存块tag |
| io\_req\_set | output | Prefetcher | CoupledL2 | io\_req  (PrefetchIO) | 7 | 预取请求缓存块set |
| io\_req\_needT | output | Prefetcher | CoupledL2 | io\_req  (PrefetchIO) | 1 | 预取请求地址是否需要TLB翻译；高需要 |
| io\_req\_source | output | Prefetcher | CoupledL2 | io\_req  (PrefetchIO) | sourceIdBits | 预取请求来源 |
| io\_req\_pfsource | output | Prefetcher | CoupledL2 | io\_req  (PrefetchIO) | 3 | 训练数据预取请求的预取器来源。0：nowhere；1：SMS；2：BOP；3：Stream；4：Stride；5：TP |
| io\_resp\_ready | output | Prefetcher | CoupledL2 | io\_resp  (PrefetchIO) | 1 | 预取请求响应可被接收；高有效 |
| io\_resp\_valid | input | CoupledL2 | Prefetcher | io\_resp  (PrefetchIO) | 1 | 预取请求响应有效；高有效 |
| io\_resp\_tag | input | CoupledL2 | Prefetcher | io\_resp  (PrefetchIO) | 26 | 预取请求响应数据对应缓存块tag |
| io\_resp\_set | input | CoupledL2 | Prefetcher | io\_resp  (PrefetchIO) | 7 | 预取请求响应数据对应缓存块tag |
| io\_resp\_addr | input | CoupledL2 | Prefetcher | io\_resp  (PrefetchIO) | 39 | 预取请求响应数据对应缓存块起始地址 |
| io\_recv\_addr \_valid | input | CoupledL2 | Prefetcher | io\_resp  (PrefetchIO) | 1 | L1 DCache预取请求有效；高有效 |
| io\_recv\_addr\_addr | input | CoupledL2 | Prefetcher | io\_recv\_addr | 64 | L1 DCache预取请求地址 |
| io\_recv\_addr\_pfSource | input | CoupledL2 | Prefetcher | io\_recv\_addr | 4 | 训练数据预取请求来源。  0：nowhere  1：CPUInst  2：CPULoadData  3：CPUStoreData  4：CPUAtomicData  5：L1InstPrefetch  6：L1DataPrefetch  7：PTW  8：Prefetch2L2BOP  9：Prefetch2L2SMS  10：Prefetch2L2Stream  11：Prefetch2L2Stride  12：Prefetch2L2TP  13：Prefetch2L2Unknown  14：Prefetch2L3Unknown  15：ReqSourceCount |
| tpio\_tpmeta\_port\_req\_ready | input | CoupledL2 | Prefetcher | tpio\_\_tpmeta\_port\_req | 1 | TP元数据写请求可被接收；高有效 |
| tpio\_tpmeta\_port\_req\_hartid | output | Prefetcher | CoupledL2 | tpio\_\_tpmeta\_port\_req | 1 | TP元数据写请求有效；高有效 |
| tpio\_tpmeta\_port\_req\_valid | output | Prefetcher | CoupledL2 | tpio\_\_tpmeta\_port\_req | 4 | TP元数据写请求硬件线程序号 |
| tpio\_tpmeta\_port\_req\_set | output | Prefetcher | CoupledL2 | tpio\_\_tpmeta\_port\_req | 32 | TP元数据写请求的缓存块set |
| tpio\_tpmeta\_port\_req\_way | output | Prefetcher | CoupledL2 | tpio\_\_tpmeta\_port\_req | 8 | TP元数据写请求的缓存块way |
| tpio\_tpmeta\_port\_req\_wmode | output | Prefetcher | CoupledL2 | tpio\_\_tpmeta\_port\_req | 1 | TP元数据读写模式。  0：读  1：写 |
| tpio\_tpmeta\_port\_req\_rawData | output | Prefetcher | CoupledL2 | tpio\_\_tpmeta\_port\_req | 16 \* 30 | TP元数据写请求数据：16个30位数据 |
| tpio\_tpmeta\_port\_resp\_valid | input | CoupledL2 | Prefetcher | tpio\_\_tpmeta\_port\_resp | 1 | TP元数据请求响应有效；高有效 |
| tpio\_tpmeta\_port\_resp\_hartid | input | CoupledL2 | Prefetcher | tpio\_\_tpmeta\_port\_resp | 4 | TP元数据请求响应硬件线程序号 |
| tpio\_tpmeta\_port\_resp\_rawData | input | CoupledL2 | Prefetcher | tpio\_\_tpmeta\_port\_resp | 16 \* 30 | TP元数据请求响应数据：16个30位数据 |

* + 1. 接口时序

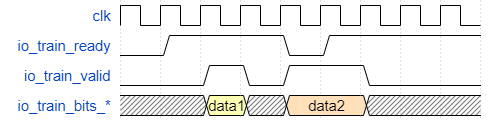


图4.2 io\_train接口时序图

io\_train接口：当ready和valid信号同时为高电平（有效）时，Prefetcher接收io\_train数据。Prefetcher模块中，io\_train接口直接与BOP、TP二级模块的相应接口连线，具体ready信号行为见BOP、TP二级模块相应时序图。

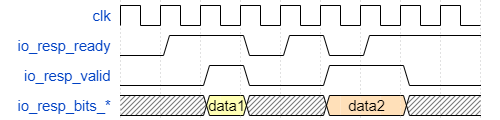


图4.3 io\_resp接口时序图

io\_resp接口：当ready和valid信号同时为高电平（有效）时，Prefetcher接收io\_resp数据。Prefetcher模块中，io\_resp接口直接与BOP、TP二级模块的相应接口连线，具体ready信号行为见BOP、TP二级模块相应时序图。

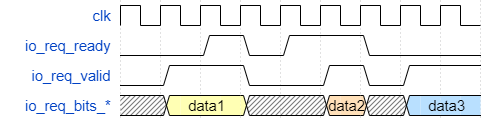


图4.4 io\_req接口时序图

io\_req接口：当ready和valid信号同时为高电平（有效）时，Prefetcher输出预取请求。Prefetcher模块中，io\_req接口通过Pipeline（Pipeline(io.req.bits.cloneType, 1)）与PrefetcherQueue二级模块的相应接口连线，具体valid信号行为见PrefetcherQueue二级模块相应时序图。

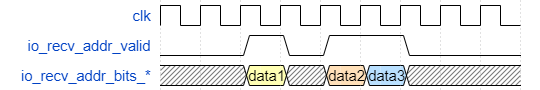


图4.5 io\_recv\_addr接口时序图

io\_recv\_addr接口：当valid信号为高电平（有效）时，Prefetcher接收io\_recv\_addr数据。Prefetcher模块中，io\_recv\_addr接口与PrefetcherReceiver二级模块的相应接口连接（延迟2拍）。

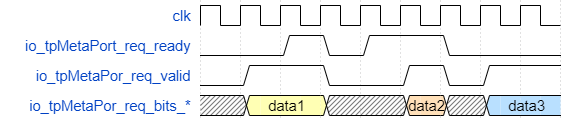


图4.6 io\_tpMetaPort\_req接口时序图

io\_tpMetaPort\_req接口：当ready和valid信号同时为高电平（有效）时，Prefetcher输出TP Meta请求。Prefetcher模块中，io\_tpMetaPort\_req接口通过与TP二级模块的相应接口连线，具体valid信号行为见PrefetcherQueue二级模块相应时序图。

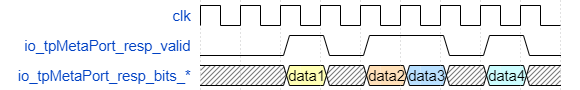


图4.7 io\_tpMetaPort\_resp接口时序图

io\_tpMetaPort\_resp接口：当valid信号为高电平（有效）时，Prefetcher接收外部总线返回的TP Meta数据。Prefetcher模块中，io\_tpMetaPort\_resp接口与TP二级模块的相应接口连接。

* + 1. 三级模块设计

Prefetcher模块包含PrefetchReceiver、PrefetchQueue、BOP、TP四个二级模块。

* + 1. 三级模块 PrefetchReceiver
       1. 功能

PrefetchReceiver模块仅根据输入的L1 DCache预取请求重新生成符合L2 Cache预取请求格式的信号（将recv\_addr转换为L2预取请求格式），无其他功能。

* + - 1. 接口列表

表5.1 PrefetchReceiver接口列表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **信号** | **方向** | **源** | **目的** | **信号组名** | **位宽** | **描述** |
| Clock | input |  |  |  | 1 | 时钟信号 |
| Reset | input |  |  |  | 1 | 异步复位信号 |
| io\_train\_ready | output | PrefetchReceiver | Prefetcher | io\_train  (PrefetchIO) | 1 | 训练数据可以接收；高有效 |
| io\_train\_valid | input | Prefetcher | PrefetchReceiver | io\_train  (PrefetchIO) | 1 | 训练数据有效；高有效 |
| io\_train\_tag | input | Prefetcher | PrefetchReceiver | io\_train  (PrefetchIO) | 26 | 训练数据对应Cache tag |
| io\_train\_set | input | Prefetcher | PrefetchReceiver | io\_train  (PrefetchIO) | 7 | 训练数据对应Cache set |
| io\_train\_needT | input | Prefetcher | PrefetchReceiver | io\_train  (PrefetchIO) | 1 | 训练数据是否需要TLB翻译。高需要 |
| io\_train\_source | input | Prefetcher | PrefetchReceiver | io\_train  (PrefetchIO) | sourceIdBits | 训练数据来源 |
| io\_train\_vaddr | input | Prefetcher | PrefetchReceiver | io\_train  (PrefetchIO) | 39 | 训练数据对应Cache set |
| io\_train\_hit | input | Prefetcher | PrefetchReceiver | io\_train  (PrefetchIO) | 1 | 训练数据是否命中L2 Cache。高命中 |
| io\_train\_prefetched | input | Prefetcher | PrefetchReceiver | io\_train  (PrefetchIO) | 1 | 训练数据是否预取过。高有效 |
| io\_train\_pfsource | input | Prefetcher | PrefetchReceiver | io\_train  (PrefetchIO) | 3 | 训练数据预取请求的预取器来源。0：nowhere；1：SMS；2：BOP；3：Stream；4：Stride；5：TP |
| io\_train\_reqsource | input | Prefetcher | PrefetchReceiver | io\_train  (PrefetchIO) | 4 | 训练数据预取请求来源。  0：nowhere  1：CPUInst  2：CPULoadData  3：CPUStoreData  4：CPUAtomicData  5：L1InstPrefetch  6：L1DataPrefetch  7：PTW  8：Prefetch2L2BOP  9：Prefetch2L2SMS  10：Prefetch2L2Stream  11：Prefetch2L2Stride  12：Prefetch2L2TP  13：Prefetch2L2Unknown  14：Prefetch2L3Unknown |
| io\_req\_ready | input | Prefetcher | PrefetchReceiver | io\_req  (PrefetchIO) | 1 | 预取请求可以被接收；高有效 |
| io\_req\_valid | output | PrefetchReceiver | Prefetcher | io\_req  (PrefetchIO) | 1 | 预取请求有效；高有效 |
| io\_req\_tag | output | PrefetchReceiver | Prefetcher | io\_req  (PrefetchIO) | 26 | 预取请求缓存块tag |
| io\_req\_set | output | PrefetchReceiver | Prefetcher | io\_req  (PrefetchIO) | 7 | 预取请求缓存块set |
| io\_req\_needT | output | PrefetchReceiver | Prefetcher | io\_req  (PrefetchIO) | 1 | 预取请求地址是否需要TLB翻译；高需要 |
| io\_req\_source | output | PrefetchReceiver | Prefetcher | io\_req  (PrefetchIO) | sourceIdBits | 预取请求来源 |
| io\_req\_pfsource | output | PrefetchReceiver | Prefetcher | io\_req  (PrefetchIO) | 3 | 训练数据预取请求的预取器来源。0：nowhere；1：SMS；2：BOP；3：Stream；4：Stride；5：TP |
| io\_resp\_ready | output | PrefetchReceiver | Prefetcher | io\_resp  (PrefetchIO) | 1 | 预取请求响应可被接收；高有效 |
| io\_resp\_valid | input | Prefetcher | PrefetchReceiver | io\_resp  (PrefetchIO) | 1 | 预取请求响应有效；高有效 |
| io\_resp\_tag | input | Prefetcher | PrefetchReceiver | io\_resp  (PrefetchIO) | 26 | 预取请求响应数据对应缓存块tag |
| io\_resp\_set | input | Prefetcher | PrefetchReceiver | io\_resp  (PrefetchIO) | 7 | 预取请求响应数据对应缓存块tag |
| io\_resp\_addr | input | Prefetcher | PrefetchReceiver | io\_resp  (PrefetchIO) | 39 | 预取请求响应数据对应缓存块起始地址 |
| io\_recv\_addr \_valid | input | Prefetcher | PrefetchReceiver | io\_resp  (PrefetchIO) | 1 | L1 DCache预取请求有效；高有效 |
| io\_recv\_addr\_addr | input | Prefetcher | PrefetchReceiver | io\_recv\_addr | 64 | L1 DCache预取请求地址 |
| io\_recv\_addr\_pfSource | input | Prefetcher | PrefetchReceiver | io\_recv\_addr | 4 | 训练数据预取请求来源。  0：nowhere  1：CPUInst  2：CPULoadData  3：CPUStoreData  4：CPUAtomicData  5：L1InstPrefetch  6：L1DataPrefetch  7：PTW  8：Prefetch2L2BOP  9：Prefetch2L2SMS  10：Prefetch2L2Stream  11：Prefetch2L2Stride  12：Prefetch2L2TP  13：Prefetch2L2Unknown  14：Prefetch2L3Unknown |

* + - 1. 接口时序

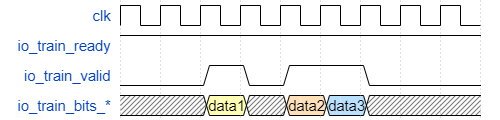


图5.1 PrefetchReceiver io\_train接口时序图

io\_train接口：PrefetchReceiver模块的ready信号始终为高电平，可接收train数据。但是在PrefetchReceiver模块内train数据未被使用。

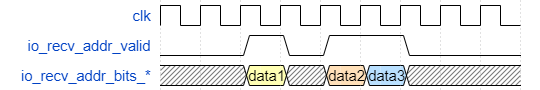


图5.2 PrefetchReceiver io\_recv\_addr接口时序图

io\_recv\_addr接口：当valid信号为高电平（有效）时，PrefetchReceiver接收io\_recv\_addr数据。PrefetchReceiver模块中，将recv\_addr数据转换为合法L2预取请求，通过io\_req接口发送。

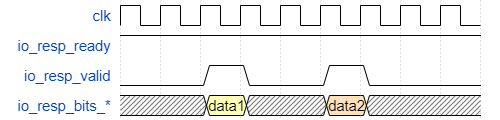


图5.3 PrefetchReceiver io\_resp接口时序图

io\_resp接口：PrefetchReceiver模块的ready信号始终为高电平，可接收resp数据。但是在PrefetchReceiver模块内resp数据未被使用。

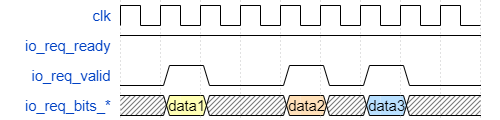


图5.4 PrefetchReceiver io\_req接口时序图

io\_req接口：PrefetchReceiver模块io\_req接口的ready信号始终为高电平，可持续向Prefetcher模块输出预取请求。

* + 1. 三级模块PrefetchQueue
       1. 功能

PrefetchQueue模块例化了一个容量为16（inflightEntries）的寄存器循环队列用于缓存预取请求。队列中元素类型为PrefetchReq（信号内容参考本模块接口列表），初始化为0。队列使用head、tail两个指针分别指向队首和队尾。该队列始终保留最新的预取请求，当队列已满时，新的预取请求直接替换队列中最老的预取请求（队首）。

预取请求输入：当输入有效的预取请求时（io\_enq\_valid信号拉高），将预取请求存入队尾。当队列不为空或者Prefetcher总线不允许输出预取请求时，将当前队尾的预取请求置为有效，并将tail指针加一。

预取请求输出：当队列不空，且Prefetcher总线允许输出预取请求时（io\_deq\_ready信号拉高），输出队首的预取请求，并将head指针加一、无效化已经输出的预取请求。当队列不空或输入有效的预取请求时，将io\_dnq\_valid信号拉高，可输出有效与去请求（队首预取请求或者此时传入的预取请求）。

* + - 1. 整体框图

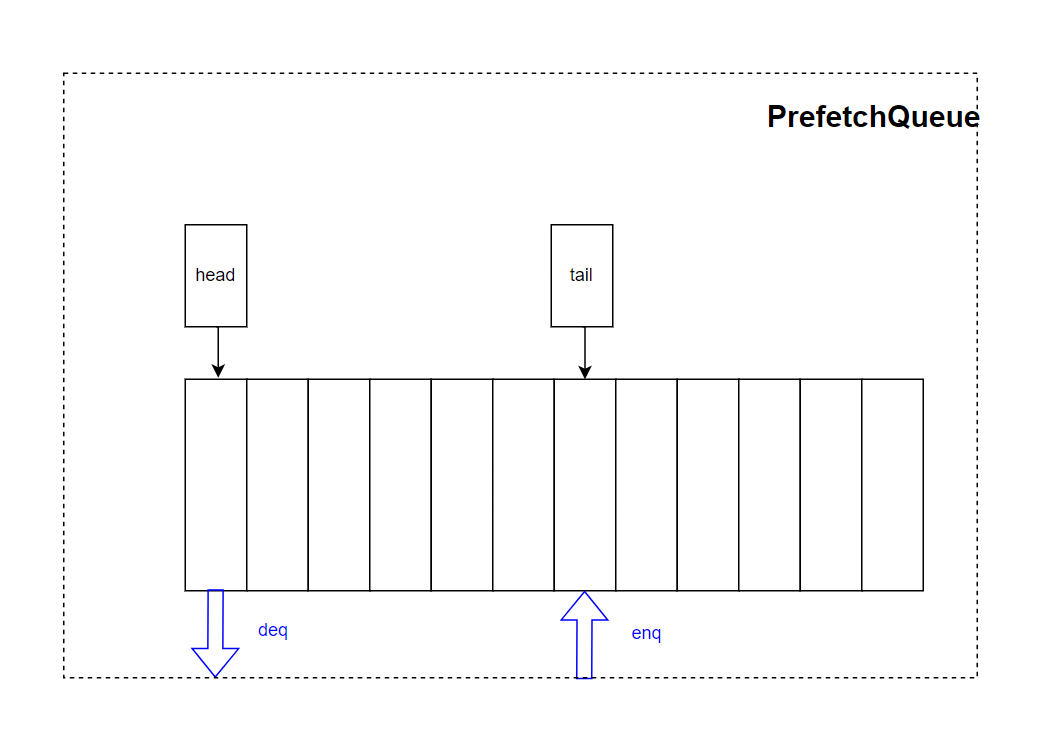


图5.5 PrefetchQueue整体框图

* + - 1. 接口列表

表5.2 PrefetchQueue接口列表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **信号** | **方向** | **源** | **目的** | **信号组名** | **位宽** | **描述** |
| Clock | input |  |  |  | 1 | 时钟信号 |
| Reset | input |  |  |  | 1 | 异步复位信号 |
| io\_enq\_ready | output | Prefetcher | PrefetchQueue | io\_enq  (PrefetchReq) | 1 | 输入的预取请求可以被接收；高有效 |
| io\_ enq \_valid | input | PrefetchQueue | Prefetcher | io\_enq  (PrefetchReq)) | 1 | 输入的预取请求有效；高有效 |
| io\_ enq \_tag | input | PrefetchQueue | Prefetcher | io\_enq  (PrefetchReq) | 26 | 输入的预取请求缓存块tag |
| io\_ enq \_set | input | PrefetchQueue | Prefetcher | io\_enq  (PrefetchReq)) | 7 | 输入的预取请求缓存块set |
| io\_ enq \_needT | input | PrefetchQueue | Prefetcher | io\_enq  (PrefetchReq) | 1 | 输入的预取请求地址是否需要TLB翻译；高需要 |
| io\_ enq \_source | input | PrefetchQueue | Prefetcher | io\_enq  (PrefetchReq)) | sourceIdBits | 输入的预取请求来源 |
| io\_ enq \_pfsource | input | PrefetchQueue | Prefetcher | io\_enq  (PrefetchReq) | 3 | 输入的训练数据预取请求的预取器来源。0：nowhere；1：SMS；2：BOP；3：Stream；4：Stride；5：TP |
| io\_deq\_ready | input | PrefetchQueue | Prefetcher | io\_deq  (PrefetchReq) | 1 | 输出的预取请求可以被接收；高有效 |
| io\_ deq \_valid | output | Prefetcher | PrefetchQueue | io\_deq  (PrefetchReq) | 1 | 输出的预取请求有效；高有效 |
| io\_ deq \_tag | output | Prefetcher | PrefetchQueue | io\_deq  (PrefetchReq) | 26 | 输出的预取请求缓存块tag |
| io\_ deq \_set | output | Prefetcher | PrefetchQueue | io\_deq  (PrefetchReq) | 7 | 输出的预取请求缓存块set |
| io\_ deq \_needT | output | Prefetcher | PrefetchQueue | io\_deq  (PrefetchReq) | 1 | 输出的预取请求地址是否需要TLB翻译；高需要 |
| io\_ deq \_source | output | Prefetcher | PrefetchQueue | io\_deq  (PrefetchReq) | sourceIdBits | 输出的预取请求来源 |
| io\_ deq \_pfsource | output | Prefetcher | PrefetchQueue | io\_deq  (PrefetchReq) | 3 | 输出的训练数据预取请求的预取器来源。0：nowhere；1：SMS；2：BOP；3：Stream；4：Stride；5：TP |
| io\_ deq \_ready | output | Prefetcher | PrefetchQueue | io\_deq  (PrefetchReq) | 1 | 输出的预取请求可以被接收；高有效 |
| io\_ deq \_valid | output | Prefetcher | PrefetchQueue | io\_deq  (PrefetchReq) | 1 | 输出的预取请求有效；高有效 |

* + - 1. 接口时序

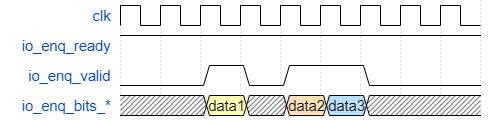


图5.6 PrefetchQueue io\_enq接口时序图

io\_enq接口：PrefetchQueue模块io\_enq接的ready信号始终为高电平，可持续接收Prefetcher模块中其他二级模块产生的预取请求（入队）。

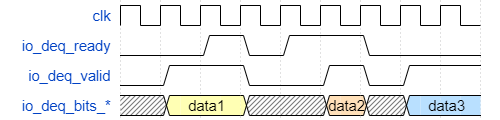


图5.7 PrefetchQueue io\_deq接口时序图

io\_deq接口：当ready和valid信号同时为高电平（有效）时，PrefetchQueue输出预取请求。Prefetcher模块中，当队列不空（有缓存的预取请求）或io\_enq\_valid为1（接收到新的预取请求）时，将valid信号置为高电平。

* + - 1. 寄存器配置

表5.3 PrefetchQueue寄存器说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器** | **地址** | **复位值** | **属性** | **描述** |
| head |  | 4’d0 | RW | 队列头指针 |
| tail |  | 4’d0 | RW | 队列尾指针 |

* + 1. 三级模块BestOffestPrefetch
       1. 功能

BOP模块接收来自Prefetcher模块的训练数据进行最佳偏移量预取。BOP模块包含RecentRequestTable（最近请求表，用于记录最近的BOP预取相关的内存访问请求）和OffsetScoreTable（偏移量分数表，用于训练、记录各个偏移量的分数，获取当前最佳偏移量）两个子模块。

BOP模块中预设37个偏移量，分别为-32, -30, -27, -25, -24, -20, -18, -16, -15, -12, -10, -9, -8, -6, -5, -4, -3, -2, -1, 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, 16, 18, 20, 24, 25, 27, 30。偏移量以缓存块为间隔单位，即地址运算时需要将偏移量左移8位（offsetBits）。

当OffsetScoreTable模块可接收新的训练地址且上一个预取请求已经可以被发送时，BOP模块可接收新的训练数据，并将训练数据的地址发送至OffsetScoreTable模块对各个偏移量进行训练。OffsetSocreTable模块中循环将训练数据的地址和个偏移量发送至RecentRequestTable模块发起查询，若在RecentRequestTable中命中，则增加对应偏移量的分数。该循环当达到循环最大次数或某一偏移量达到最大分数时终止，终止时OffsetSocreTable向BOP模块发送当前最大分数的偏移量。在下一拍，BOP模块将训练数据地址与当前最大分数的偏移量相加，生成新的预取请求。当预取的地址与训练地址在同一页时，置预取请求为有效，并向Prefetcher模块发送。

BOP模块接收来自Prefetcher模块的预取请求响应，将响应信号的地址减去当前最大分数的偏移量，生成最近内存访问请求的地址发送至RecentRequestTable模块更新最近访问请求记录。

* + - 1. 整体框图

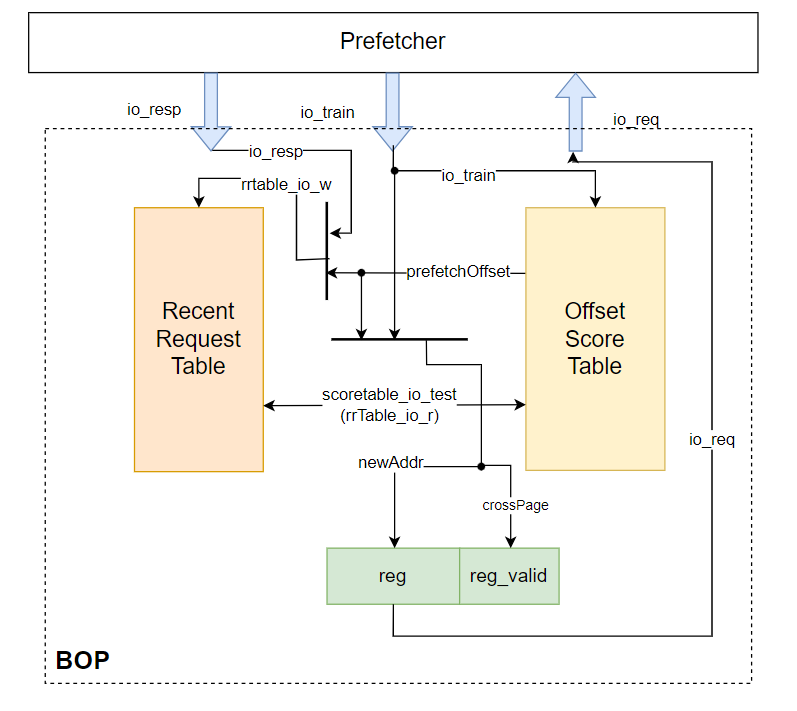


图5.8 BOP整体框图

上图中，连线箭头表示数据传输方向。连线交汇处存在圆点的，表示相交。加粗横线部分表示存在逻辑操作：prefetchOffset信号与io\_resp信号生成rrtable\_io\_w信号；prefetchOffset信号与io\_train信号生成newAddr与crossPage（判定预取地址与训练地址是否在同一页）。RecentRequestTable的io\_r（读）接口与OffsetScoreTable模块的io\_test接口直接相连。

* + - 1. 接口列表

表5.4 BOP接口列表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **信号** | **方向** | **源** | **目的** | **信号组名** | **位宽** | **描述** |
| Clock | input |  |  |  | 1 | 时钟信号 |
| Reset | input |  |  |  | 1 | 异步复位信号 |
| io\_train\_ready | output | BOP | Prefetcher | io\_train  (PrefetchIO) | 1 | 训练数据可以接收；高有效 |
| io\_train\_valid | input | Prefetcher | BOP | io\_train  (PrefetchIO) | 1 | 训练数据有效；高有效 |
| io\_train\_tag | input | Prefetcher | BOP | io\_train  (PrefetchIO) | 26 | 训练数据对应Cache tag |
| io\_train\_set | input | Prefetcher | BOP | io\_train  (PrefetchIO) | 7 | 训练数据对应Cache set |
| io\_train\_needT | input | Prefetcher | BOP | io\_train  (PrefetchIO) | 1 | 训练数据是否需要TLB翻译。高需要 |
| io\_train\_source | input | Prefetcher | BOP | io\_train  (PrefetchIO) | sourceIdBits | 训练数据来源 |
| io\_train\_vaddr | input | Prefetcher | BOP | io\_train  (PrefetchIO) | 39 | 训练数据对应Cache set |
| io\_train\_hit | input | Prefetcher | BOP | io\_train  (PrefetchIO) | 1 | 训练数据是否命中L2 Cache。高命中 |
| io\_train\_prefetched | input | Prefetcher | BOP | io\_train  (PrefetchIO) | 1 | 训练数据是否预取过。高有效 |
| io\_train\_pfsource | input | Prefetcher | BOP | io\_train  (PrefetchIO) | 3 | 训练数据预取请求的预取器来源。0：nowhere；1：SMS；2：BOP；3：Stream；4：Stride；5：TP |
| io\_train\_reqsource | input | Prefetcher | BOP | io\_train  (PrefetchIO) | 4 | 训练数据预取请求来源。  0：nowhere  1：CPUInst  2：CPULoadData  3：CPUStoreData  4：CPUAtomicData  5：L1InstPrefetch  6：L1DataPrefetch  7：PTW  8：Prefetch2L2BOP  9：Prefetch2L2SMS  10：Prefetch2L2Stream  11：Prefetch2L2Stride  12：Prefetch2L2TP  13：Prefetch2L2Unknown  14：Prefetch2L3Unknown |
| io\_req\_ready | input | Prefetcher | BOP | io\_req  (PrefetchIO) | 1 | 预取请求可以被接收；高有效 |
| io\_req\_valid | output | BOP | Prefetcher | io\_req  (PrefetchIO) | 1 | 预取请求有效；高有效 |
| io\_req\_tag | output | BOP | Prefetcher | io\_req  (PrefetchIO) | 26 | 预取请求缓存块tag |
| io\_req\_set | output | BOP | Prefetcher | io\_req  (PrefetchIO) | 7 | 预取请求缓存块set |
| io\_req\_needT | output | BOP | Prefetcher | io\_req  (PrefetchIO) | 1 | 预取请求地址是否需要TLB翻译；高需要 |
| io\_resp\_ready | output | BOP | Prefetcher | io\_resp  (PrefetchIO) | 1 | 预取请求响应可被接收；高有效 |
| io\_resp\_valid | input | Prefetcher | BOP | io\_resp  (PrefetchIO) | 1 | 预取请求响应有效；高有效 |
| io\_resp\_tag | input | Prefetcher | BOP | io\_resp  (PrefetchIO) | 26 | 预取请求响应数据对应缓存块tag |
| io\_resp\_set | input | Prefetcher | BOP | io\_resp  (PrefetchIO) | 7 | 预取请求响应数据对应缓存块tag |
| io\_resp\_addr | input | Prefetcher | BOP | io\_resp  (PrefetchIO) | 39 | 预取请求响应数据对应缓存块起始地址 |

* + - 1. 接口时序

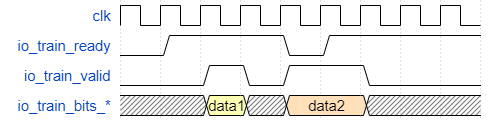


图5.9 BOP io\_train接口时序图

io\_train接口：当ready和valid信号同时为高电平（有效）时，BOP接收io\_train数据。BOP模块中，当且仅当OffsetScoreTable（三级）模块允许查询（其io\_req\_ready为1）且无等待输出的预取请求时，ready信号为高电平。不过，在Prefetcher模块中，始终允许接收来自BOP模块的预取请求，故ready信号仅与OffsetScoreTable模块相关。

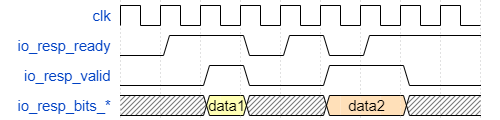


图5.10 BOP io\_resp接口时序图

io\_resp接口：当ready和valid信号同时为高电平（有效）时，BOP接收io\_resp数据。BOP模块中，当且仅当RecentRequestTable（三级）模块允许写请求（其io\_w\_ready为1），ready信号为高电平。

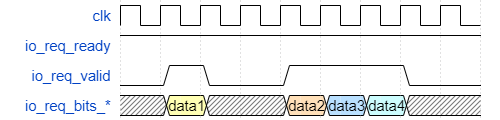


图5.11 BOP io\_req接口时序图

io\_req接口：BOP模块io\_req接口的ready信号始终为高电平，可持续向Prefetcher模块输出预取请求。

* + - 1. 寄存器配置

表5.5 BOP寄存器说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器** | **地址** | **复位值** | **属性** | **描述** |
| req |  | 36’d0 | RW | BOP预取请求，具体寄存器各位信号与接口表格中io\_req信号一致。 |
| req\_valid |  | 1’d0 | RW | BOP预取请求有效；高有效 |

* + - 1. 三级模块RecentRequestTable
         1. 功能

RecentRequestTable模块为最近内存访问请求记录表。RecentRequestTable模块例化了一个256项的随机静态存储器作为最近内存访问请求记录表。表中元素类型为rrTableEntry，包含一个12位的tag以及一个1位的valid信号。

当内存访问请求记录表允许写入且没有读请求时（读优先，避免读写冲突），RecentRequestTable模块可以接收来自BOP模块的写请求（最近内存访问地址），将地址的（15，8）与（23，16）位进行异或（哈希），获取内存访问请求记录表的index。当内存访问请求记录表没有读请求时，RecentRequestTable向内存访问请求记录表发起写请求，更新内存访问请求记录。

RecentRequestTable模块可以持续接收来自OffsetScoreTable模块的读请求。将读请求中的访问地址减去偏移量，生成内存访问请求记录表的读请求地址rAddr，对内存访问请求记录表发起访问，读出对应的tag。在下一拍，RecentRequestTable模块生成读请求的响应信号，并发送至OffsetScoreTable模块。响应信号中valid与ptr与读请求信号中对valid与ptr保持一致；当内存访问请求记录表读出的tag与rAddr的tag一致时，置响应信号中的hit为1，否则为0。

* + - * 1. 整体框图

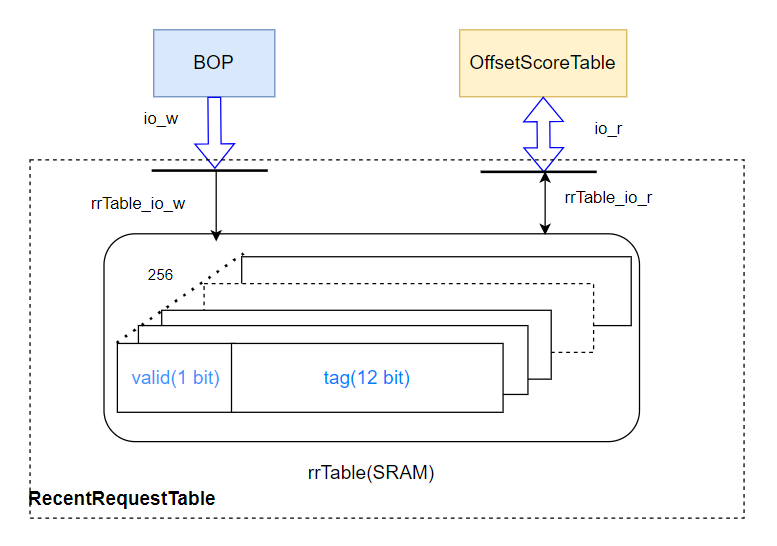


图5.12 RecentRequestTable整体框图

上图中，连线箭头表示数据传输方向。加粗横线部分表示存在逻辑操作：BOP模块输入的地址为全地址，需要进行5.3.6.1节中的哈希操作获取rrTable的索引，并截取地址的tag部分作为输入 ；OffsetScoreTable模块同理。

rrTable为一个的SRAM，读写端口为标准SRAM读写端口。rrTable为读优先策略。

* + - * 1. 接口列表

表5.6 RecentRequestTable接口列表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **信号** | **方向** | **源** | **目的** | **信号组名** | **位宽** | **描述** |
| Clock | input |  |  |  | 1 | 时钟信号 |
| Reset | input |  |  |  | 1 | 异步复位信号 |
| io\_w\_ready | output | RecentRequestTable | BOP | io\_w | 1 | 写请求可以接收；高有效 |
| io\_w\_valid | input | BOP | RecentRequestTable | io\_w | 1 | 写请求有效；高有效 |
| io\_w\_bits | input | BOP | RecentRequestTable | io\_w | 39  fulladddressbits | 写请求数据，最近访问地址 |
| io\_r\_req\_ready | output | RecentRequestTable | OffsetScoreTable | io\_r  (TestOffsetReq) | 1 | 读请求可以接收；高有效 |
| io\_r\_req\_valid | input | OffsetScoreTable | RecentRequestTable | io\_r  (TestOffsetReq) | 1 | 读请求有效；高有效 |
| io\_r\_req\_addr | input | OffsetScoreTable | RecentRequestTable | io\_r  (TestOffsetReq) | 39 | 读请求访问地址 |
| io\_r\_req\_testOffest | input | OffsetScoreTable | RecentRequestTable | io\_r  (TestOffsetReq) | 7 | 读请求当前训练的偏移量 |
| io\_r\_req\_ptr | input | OffsetScoreTable | RecentRequestTable | io\_r  (TestOffsetReq) | 6 | 读请求当前训练的偏移量对应分数的指针  （OffsetScoreTable） |
| io\_r\_resp\_ready | input | OffsetScoreTable | RecentRequestTable | io\_r  (TestOffsetResp) | 1 | 读响应可以接收；高有效 |
| io\_r\_resp\_valid | output | RecentRequestTable | OffsetScoreTable | io\_r  (TestOffsetResp) | 1 | 读响应有效；高有效 |
| io\_r\_resp\_ptr | output | RecentRequestTable | OffsetScoreTable | io\_r  (TestOffsetResp) | 6 | 读响应当前训练的偏移量对应分数的指针  （OffsetScoreTable） |
| io\_r\_resp\_hit | output | RecentRequestTable | OffsetScoreTable | io\_r  (TestOffsetResp) | 1 | 读请求是否命中最近访问记录表。1为命中 |

* + - * 1. 接口时序

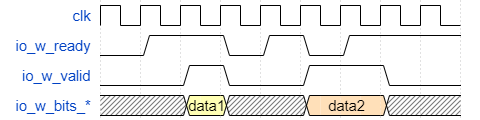


图5.13 RecentRequestTable io\_w接口时序图

io\_w接口（写请求）：当ready和valid信号同时为高电平（有效）时，RecentRequestTable接收写请求数据。RecentRequestTable模块中，当且仅当内部rrtable（SRAM）允许写入且RecentRequestTable模块没有读请求时，ready信号置为高电平（读优先策略）。

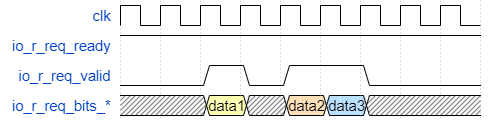


图5.13 RecentRequestTable io\_r\_req接口时序图

io\_r\_req接口（读请求）：ready始终为高电平（有效），RecentRequestTable可持续接收读请求数据（读优先策略）。

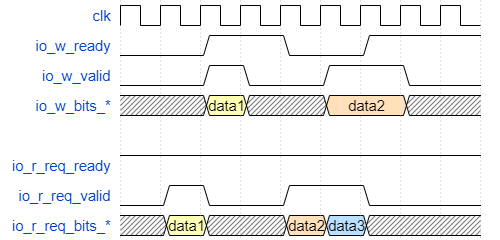


图5.14 RecentRequestTable 读写请求冲突时序图

由于RecentRequestTable模块的读优先策略，当io\_r\_req\_valid为高电平时，需要将io\_w\_ready置为低电平，暂停写请求响应直至无写请求。

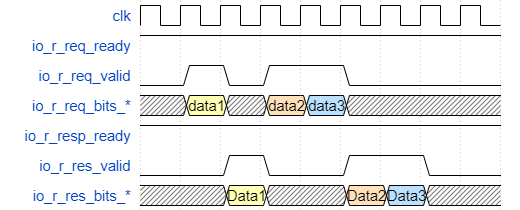


图5.15 RecentRequestTable io\_r\_resp接口时序图

io\_r\_req接口（读请求）：ready始终为高电平（有效），RecentRequestTable可持续输出响应数据。读响应晚于读请求一拍（SRAM读取数据需要一拍）。

* + - 1. 三级模块OffsetScoreTable
         1. 功能

OffsetScoreTable模块为偏移量分数训练模块。OffsetScoreTable模块中例化一个37项的队列寄存器组st用于存储每一个偏移量对应的分数。该寄存器组中每一个元素均为ScoreTableEntry类型，包含一个5位的分数数据。OffsetScoreTable模块使用一个寄存器ptr指向当前训练的偏移量。

OffsetScoreTable模块使用2个状态的状态机控制偏移量的分数训练：

* s\_idle：该状态为初始状态，将st中每一项的分数清零，ptr置0，训练循环计数器round置0，当前最大分数置为1，状态机下一个状态置为s\_learn。OffsetScoreTable模块向BOP模块输出的预取偏移量设置为先前的最高分数bestScore。
* s\_learn：该状态为训练状态，在该状态下，OffsetScoreTable模块可以持续接收BOP模块的训练请求，并使用两层循环进行训练。
  + 外层循环：训练循环计数器round进行计数，当达到最大循环层数数40时，终止循环，状态机下一个状态置为s\_idle。
  + 内层循环：将BOP模块的训练请求的地址依次与37个偏移量组合生成TestOffsetBundle（req & resp），向RecentRequestTable模块发起读请求。当读响应握手成功且读请求命中最近访问记录表（r\_resp\_hit信号为1）时，将对应偏移量的分数加一；若该分数高于当前最高分数，则更新最高分数bestScore以及最佳偏移量bestOffset；若该分数高于最大分数31，则终止循环，状态机下一个状态置为s\_idle。
    - * 1. 整体框图

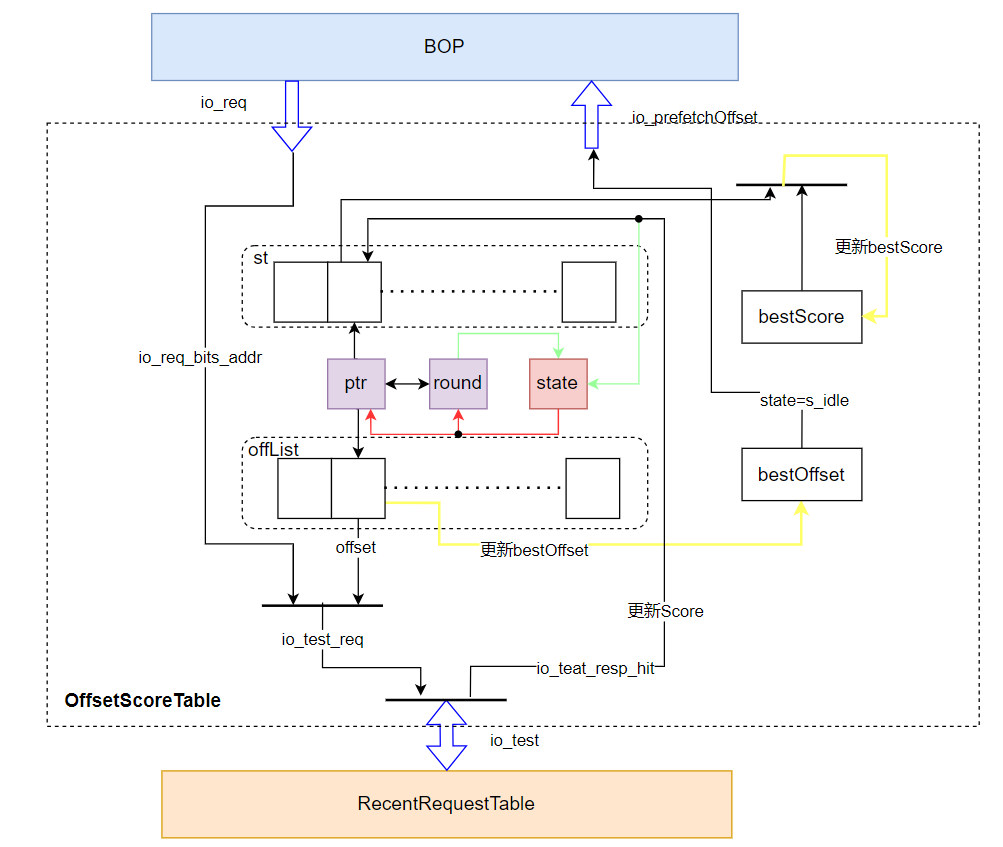


图5.16 OffsetScore整体框图

上图中，连线箭头表示数据传输方向。连线交汇处存在圆点的，表示相交。offList为设定偏移量寄存器组（具体值见5.3.7.1节），st为偏移量分数寄存器组。ptr为指针寄存器，指向当先训练offset与对应分数。round为循环次数寄存器。state为状态机寄存器。bestScore与bestOffset分别存储当前最大分数与最佳偏移量。

图中红色连线表示由state控制ptr、round行为；绿色连线表示round和分数更新时对状态机的反馈（s\_learn状态转移至s\_idle状态）；黄色连线表示当RecentRenquestTable反馈后当前训练偏移量分数超过当前最佳分数时，需更新bestScore与bestOffset寄存器。当状态机为s\_idle状态时，向BOP模块输出预取偏移量（io\_prefetchOffset）。

加粗横线部分表示存在逻辑操作：BOP模块输入的io\_req信号与ptr指向的offset组合生成传输至RecentRequestTable模块的io\_test\_req信号。

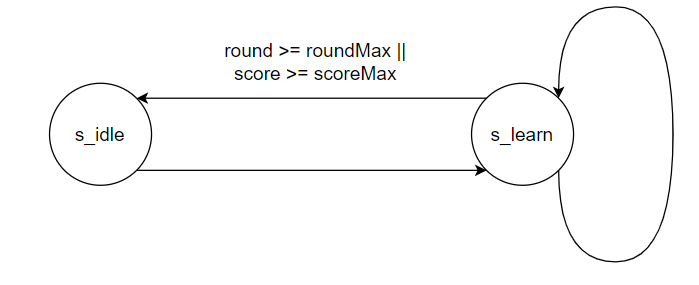


图5.17 OffsetScore状态机状态转移图

上图中，状态机从s\_idle状态无条件转移至s\_learn状态。若更新的分数大于分数上限（scoreMax）或者循环次数达到上限（roundMax），状态机从s\_learn状态转移至s\_idle状态；否则维持s\_learn状态。

* + - * 1. 接口列表

表5.7 OffsetScoreTable接口列表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **信号** | **方向** | **源** | **目的** | **信号组名** | **位宽** | **描述** |
| Clock | input |  |  |  | 1 | 时钟信号 |
| Reset | input |  |  |  | 1 | 异步复位信号 |
| io\_req\_ready | output | OffsetScoreTable | BOP | io\_req | 1 | BOP训练请求可以接收；高有效 |
| io\_req\_valid | input | BOP | OffsetScoreTable | io\_req | 1 | BOP训练请求有效；高有效 |
| io\_req\_bits | input | BOP | OffsetScoreTable | io\_req | 39  fulladddressbits | BOP训练请求数据，最近访问地址 |
| io\_prefetchOffset | output | OffsetScoreTable | BOP | io\_prefetchOffset | 6 | BOP预取偏移量 |
| io\_test\_req\_ready | input | RecentRequestTable | OffsetScoreTable | io\_r  (TestOffsetReq) | 1 | 读请求可以接收；高有效 |
| io\_test\_req\_valid | output | OffsetScoreTable | RecentRequestTable | io\_r  (TestOffsetReq) | 1 | 读请求有效；高有效 |
| io\_test\_req\_addr | output | OffsetScoreTable | RecentRequestTable | io\_r  (TestOffsetReq) | 39 | 读请求访问地址 |
| io\_test\_req\_testOffest | output | OffsetScoreTable | RecentRequestTable | io\_r  (TestOffsetReq) | 7 | 读请求当前训练的偏移量 |
| io\_test\_req\_ptr | output | OffsetScoreTable | RecentRequestTable | io\_r  (TestOffsetReq) | 6 | 读请求当前训练的偏移量对应分数的指针 |
| io\_test\_resp\_ready | output | OffsetScoreTable | RecentRequestTable | io\_r  (TestOffsetResp) | 1 | 读响应可以接收；高有效 |
| io\_test\_resp\_valid | input | RecentRequestTable | OffsetScoreTable | io\_r  (TestOffsetResp) | 1 | 读响应有效；高有效 |
| io\_test\_resp\_ptr | outp input ut | RecentRequestTable | OffsetScoreTable | io\_r  (TestOffsetResp) | 6 | 读响应当前训练的偏移量对应分数的指针 |

* + - * 1. 接口时序

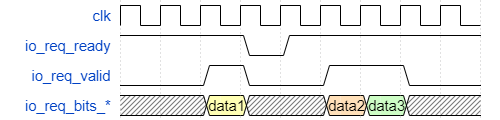


图5.18 OffsetScoreTable io\_req接口时序图

io\_req接口：当ready和valid信号同时为高电平（有效）时，OffsetScoreTable接收BOP模块的查询请求。OffsetScoreTable模块中，当状态机处于s\_learn状态（训练状态）时，将ready信号持续置为高电平。

io\_prefetchOffset接口：无时序特征（简单连线）。

io\_test（req/resp）接口：该组接口与RecentRequestTable模块的io\_r（req/resp）接口直接相连，具体时序特征可见5.3.6.4节。其中，当且仅当RecentRequestTable模块中状态机处于s\_learn状态且BOP模块发送有效查询请求时，io\_test\_req\_valid（对应RecentRequestTable模块的io\_r\_req\_valid）置为高电平。

* + - * 1. 寄存器配置

表5.8 OffsetScoreTable寄存器说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器** | **地址** | **复位值** | **属性** | **描述** |
| ptr |  | 6’d0 | RW | 偏移量offset对应分数指针：0~36（规定offset数量为37） |
| round |  | 6’d0 | RW | 循环计数器：0~39（最大循环数为40） |
| bestOffset |  | 6’d2 | RW | 当前最佳偏移量：-32~31 |
| bestScore |  | 5’d1 | RW | 当前最大分数：0~31 |
| state |  | 1’d0 | RW | 状态机：0为s\_idle；1为s\_learn |

* + 1. 二级模块TemporalPrefetch
       1. 功能

TP模块为基于时间相关性的预取模块。TP模块中例化一个16路1024组相联的静态随机存储器tpMetaTable，用于存储TP元数据的元数据信息（Meta2），tpMetaTable中每一项为tpMetaEntry类型，包含23位的triggerTag信号和1位的valid信号。TP模块中例化一个8项的队列dataReadQueue和5项的队列dataWriteQueue，分别用于缓存TP元数据的读请求以及写请求，其中每一项类型均为tpMetaReq。TP模块中例化一个8项的队列tpDataQueue，用于缓存TP元数据（预取请求数据），tpDataQueue每一项为tpDataEntry类型，包含一个16 \* 39的rawData。TP模块中例化一个4项的队列triggerQueue，用于缓存TemporalPrefetch的触发信号，triggerQueue中每一项为triggerBundle类型，包含虚地址vaddr、实地址paddr以及路选择信号way（tpMetaTable）。

TP模块使用3级流水结构进行预取训练：

* Stage 0：TP模块从Prefetcher模块接收训练数据。当训练请求握手成功后，根据预取类型（基于L2 Cache预取/基于L1 DCache预取请求的预取），使用vaddr或paddr确定（tag，set），向tpMetaTable发起查询请求，获取TP元数据的元数据信息（Meta2）
* Stage 1：TP模块基于tag比对是否命中tpMetaTable。若命中，则确定命中的路（way，数据更新）；否则，确定需要替换的路。
* Stage 2：Stage 1中命中情况下，TP模块发出TP元数据查询请求，生成预取请求；Stage 1中未命中情况下，触发预取训练。
  + Hit：
    - 获取TP元数据：TP模块将元数据查询请求入队dataReadQueue。当外部总线（io\_tpmeta\_port\_req）允许发送TP元数据读写请求且当前没有写请求时（写优先，避免读写冲突），dataReadQueue出队元数据查询请求并发送。当读写请求响应握手成功后，将外部总线（io\_tpmeta\_port\_resp）返回的rawData入队tpDataQueue。
    - 预取请求生成：当tpDataQueue已满或当前TP模块不处于发送预取请求状态（do\_sending为0）时，tpDataQueue出队数据sending\_data，并将发送计数器sending\_idx置为1（忽略第一个trigger），并置do\_sending为1。将sending\_data数组中的当前索引项sending\_idx与6位宽的0拼接，形成当前的发送数据current\_sending\_data，生成预取请求。当do\_sending为1，tpDataQueue不满，并且节流计数器sending\_throttle达到节流周期时，置预取请求为有效。在发送状态并且tpDataQueue不满，或者sending\_throttle不为0时，递增sending\_throttle计数器。成功发送一个预取请求后sending\_idx加1并重置sending\_throttle计数器。
  + Miss：此时dorecord\_s2为1，表示进行预取训练（记录训练数据）。TP模块采用trigger时（hitAsTrigger为1），将训练数据入队triggerQueue。
    - 预取训练：dorecord\_s2为1时，将训练数据的地址根据训练计数器recorder\_idx对应索引存入recorder\_data（16 \* 39的寄存器组），recorder\_idx自增。当recorder\_idx达到设定阈值recordThres后，将write\_record置为1一拍，出队triggerQueue作为当前recorder\_data的trigger，并重置recorder\_idx以及recorder\_data。
  + tpDataQueue的入队/出队以及dataReadQueue/dataWriteQueue出队均与流水线异步执行。
* Stage 3：Miss情况下，更新tpMetaTable信息。数据写回：write\_record为1时，将recorder\_data入队dataWriteQueue，并基于当前recorder\_data的trigger生成tpMetaTable的写请求，更新TP元数据的元数据信息。当外部总线（io\_tpmeta\_port\_req）允许发送TP元数据读写请求，dataWriteQueue出队元数据查询请求并发送。

* + - 1. 整体框图

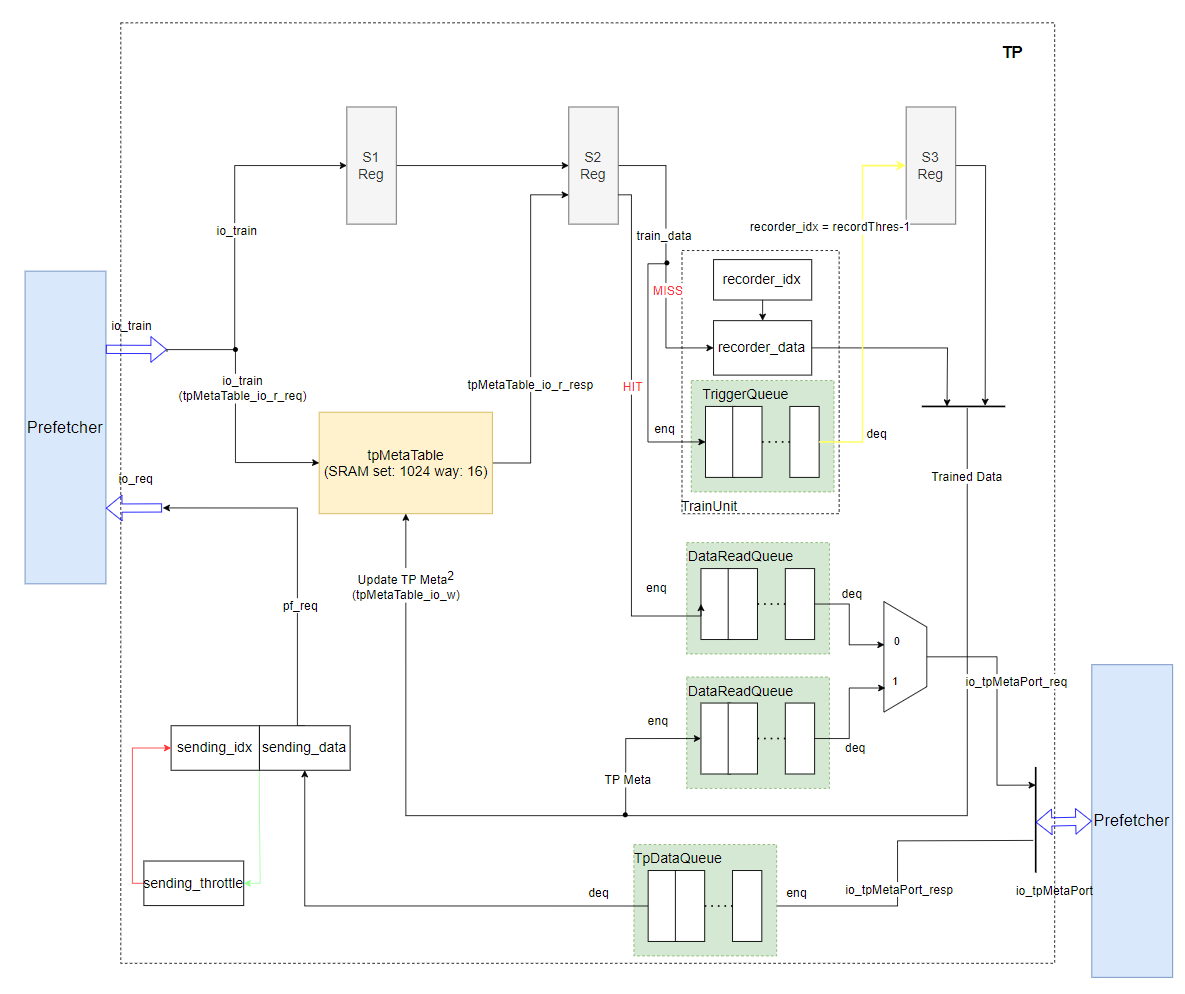


图5.19 TP整体框图

上图中，连线箭头表示数据传输方向。连线交汇处存在圆点的，表示相交。tpMetaTable为16路1024组相联的静态随机存储器。dataReadQueue、dataWriteQueue、tpDataQueue和triggerQueue均为（队列）寄存器组，用于缓存数据。recorder\_data寄存器组记录TP Meta训练数据。sending\_data寄存器组缓存预取请求的TP Meta。

图中红色连线表示由sending\_throttle寄存器控制预取请求发送行为；绿色连线表示预取请求发送情况对sending\_throttle的反馈调节；黄色连线表示当recorder\_idx达到阈值后（recordThres），将训练完成的TP Meta写回tpMetaTable中TP Meta2，

加粗横线部分表示存在逻辑操作：recorder\_data与TriggerQueue出队的Tirgger合并生成完整的训练好的数据。

* + - 1. 接口列表

表5.9 TP接口列表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **信号** | **方向** | **源** | **目的** | **信号组名** | **位宽** | **描述** |
| Clock | input |  |  |  | 1 | 时钟信号 |
| Reset | input |  |  |  | 1 | 异步复位信号 |
| io\_train\_ready | output | TP | Prefetcher | io\_train  (PrefetchIO) | 1 | 训练数据可以接收；高有效 |
| io\_train\_valid | input | Prefetcher | TP | io\_train  (PrefetchIO) | 1 | 训练数据有效；高有效 |
| io\_train\_tag | input | Prefetcher | TP | io\_train  (PrefetchIO) | 26 | 训练数据对应Cache tag |
| io\_train\_set | input | Prefetcher | TP | io\_train  (PrefetchIO) | 7 | 训练数据对应Cache set |
| io\_train\_needT | input | Prefetcher | TP | io\_train  (PrefetchIO) | 1 | 训练数据是否需要TLB翻译。高需要 |
| io\_train\_source | input | Prefetcher | TP | io\_train  (PrefetchIO) | sourceIdBits | 训练数据来源 |
| io\_train\_vaddr | input | Prefetcher | TP | io\_train  (PrefetchIO) | 39 | 训练数据对应Cache set |
| io\_train\_hit | input | Prefetcher | TP | io\_train  (PrefetchIO) | 1 | 训练数据是否命中L2 Cache。高命中 |
| io\_train\_prefetched | input | Prefetcher | TP | io\_train  (PrefetchIO) | 1 | 训练数据是否预取过。高有效 |
| io\_train\_pfsource | input | Prefetcher | TP | io\_train  (PrefetchIO) | 3 | 训练数据预取请求的预取器来源。0：nowhere；1：SMS；2：BOP；3：Stream；4：Stride；5：TP |
| io\_train\_reqsource | input | Prefetcher | TP | io\_train  (PrefetchIO) | 4 | 训练数据预取请求来源。  0：nowhere  1：CPUInst  2：CPULoadData  3：CPUStoreData  4：CPUAtomicData  5：L1InstPrefetch  6：L1DataPrefetch  7：PTW  8：Prefetch2L2BOP  9：Prefetch2L2SMS  10：Prefetch2L2Stream  11：Prefetch2L2Stride  12：Prefetch2L2TP  13：Prefetch2L2Unknown  14：Prefetch2L3Unknown  15：ReqSourceCount |
| io\_req\_ready | input | Prefetcher | TP | io\_req  (PrefetchIO) | 1 | 预取请求可以被接收；高有效 |
| io\_req\_valid | output | TP | Prefetcher | io\_req  (PrefetchIO) | 1 | 预取请求有效；高有效 |
| io\_req\_tag | output | TP | Prefetcher | io\_req  (PrefetchIO) | 26 | 预取请求缓存块tag |
| io\_req\_set | output | TP | Prefetcher | io\_req  (PrefetchIO) | 7 | 预取请求缓存块set |
| io\_req\_needT | output | TP | Prefetcher | io\_req  (PrefetchIO) | 1 | 预取请求地址是否需要TLB翻译；高需要 |
| io\_req\_source | output | TP | Prefetcher | io\_req  (PrefetchIO) | sourceIdBits | 预取请求来源 |
| io\_req\_pfsource | output | TP | Prefetcher | io\_req  (PrefetchIO) | 3 | 训练数据预取请求的预取器来源。0：nowhere；1：SMS；2：BOP；3：Stream；4：Stride；5：TP |
| io\_resp\_ready | output | TP | Prefetcher | io\_resp  (PrefetchIO) | 1 | 预取请求响应可被接收；高有效 |
| io\_resp\_valid | input | Prefetcher | TP | io\_resp  (PrefetchIO) | 1 | 预取请求响应有效；高有效 |
| io\_resp\_tag | input | Prefetcher | TP | io\_resp  (PrefetchIO) | 26 | 预取请求响应数据对应缓存块tag |
| io\_resp\_set | input | Prefetcher | TP | io\_resp  (PrefetchIO) | 7 | 预取请求响应数据对应缓存块tag |
| io\_resp\_addr | input | Prefetcher | TP | io\_resp  (PrefetchIO) | 39 | 预取请求响应数据对应缓存块起始地址 |
| tpio\_tpmeta\_port\_req\_ready | input | Prefetcher | TP | tpio\_\_tpmeta\_port\_req | 1 | TP元数据写请求可被接收；高有效 |
| tpio\_tpmeta\_port\_req\_valid | output | TP | Prefetcher | tpio\_\_tpmeta\_port\_req | 1 | TP元数据写请求有效；高有效 |
| tpio\_tpmeta\_port\_req\_hartid | output | TP | Prefetcher | tpio\_\_tpmeta\_port\_req | 4 | TP元数据写请求硬件线程序号 |
| tpio\_tpmeta\_port\_req\_set | output | TP | Prefetcher | tpio\_\_tpmeta\_port\_req | 32 | TP元数据写请求的缓存块set |
| tpio\_tpmeta\_port\_req\_way | output | TP | Prefetcher | tpio\_\_tpmeta\_port\_req | 8 | TP元数据写请求的缓存块way |
| tpio\_tpmeta\_port\_req\_wmode | output | TP | Prefetcher | tpio\_\_tpmeta\_port\_req | 1 | TP元数据读写模式。  0：读  1：写 |
| tpio\_tpmeta\_port\_req\_rawData | output | TP | Prefetcher | tpio\_\_tpmeta\_port\_req | 16 \* 30 | TP元数据写请求数据：16个30位数据 |
| tpio\_tpmeta\_port\_resp\_valid | input | Prefetcher | TP | tpio\_\_tpmeta\_port\_resp | 1 | TP元数据请求响应有效；高有效 |
| tpio\_tpmeta\_port\_resp\_hartid | input | Prefetcher | TP | tpio\_\_tpmeta\_port\_resp | 4 | TP元数据请求响应硬件线程序号 |
| tpio\_tpmeta\_port\_resp\_rawData | input | Prefetcher | TP | tpio\_\_tpmeta\_port\_resp | 16 \* 30 | TP元数据请求响应数据：16个30位数据 |

* + - 1. 接口时序

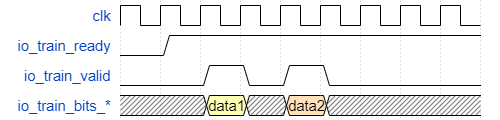


图5.20 TP io\_train接口时序图

io\_train接口：当ready和valid信号同时为高电平（有效）时，TP接收io\_train数据。BOP模块中，当完成tpMetaTable（SRAM）重置后，持续将ready信号拉高（流水线设计，允许持续进行预取（训练））。

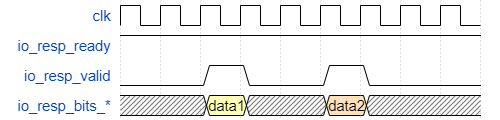


图5.21 TP io\_resp接口时序图

io\_resp接口：TP模块io\_resp接口的ready信号始终为高电平，可接收resp数据。但是在TP模块内resp数据未被使用。

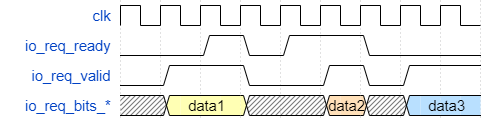


图5.22 TP io\_req接口时序图

io\_req接口：当ready和valid信号同时为高电平（有效）时，TP输出预取请求。TP模块中，预取请求发送受sending\_throttle寄存器（见5.4.1节）控制。输入的ready信号与BOP模块不同（BOP中始终为高电平），原因是在Prefetcher模块中BOP、L1预取请求的预取请求优先级高于TP，故当BOP或L1预取请求有效时，TP预取请求进行退让，延后发送。

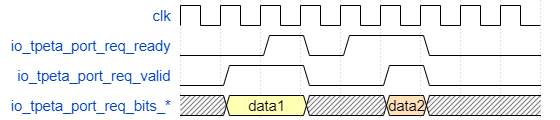


图5.23 TP io\_tpmeta\_port\_req接口时序图

io\_tpmeta\_port\_req接口：当ready和valid信号同时为高电平（有效）时，TP输出查询TP Meta请求。TP模块中，当dataReadQueue或dataWriteQueue可出队数据时，将valid信号置为高电平。此处采取写优先策略，优先dataWriteQueue出队数据。

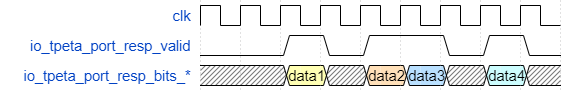


图5.23 TP io\_tpmeta\_port\_resp接口时序图

io\_tpmeta\_port \_resp接口：当valid信号为高电平（有效）时，IP接收外部总线返回的TP Meta数据。IP模块中，将返回的TP Meta数据写入tpDataQueue（需检查hartid一致）。

* + - 1. 寄存器配置

表5.10 TP寄存器说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **寄存器** | **地址** | **复位值** | **属性** | **描述** |
| resetFinish |  | 1’d0 | RW | tpMetaTable重置完成；高电平为重置完成 |
| resetIdx |  | 10’d1023 | RW | tpMetaTable重置索引寄存器：0~1023 |
| s1\_valid |  | 1’d0 | RW | 流水级s1有效信号；高电平为有效 |
| train\_s1 |  | 0 | RW | 流水及s1训练数据，具体信号与接口列表中的io\_train一致 |
| s2\_valid |  | 1’d0 | RW | 流水级s2有效信号；高电平为有效 |
| hit\_s2 |  | 1’d0 | RW | 流水级s2 tpMetaTable命中信号；高电平为命中 |
| way\_s2 |  | 4’d0 | RW | 流水级s2 tpMetaTable victim way |
| vset\_s2 |  | 10’d0 | RW | 流水级s2 训练数据虚拟地址set |
| pset\_s2 |  | 10’d0 | RW | 流水级s2 训练数据物理地址set |
| train\_s2 |  | 0 | RW | 流水及s2训练数据，具体信号与接口列表中的io\_train一致 |
| recorder\_idx |  | 6’d0 | RW | 训练记录计数器：0~recordThres-1（recordThres为设定阈值） |
| recorder\_data |  | 16\*39’d0 | RW | 训练数据记录寄存器 |
| write\_record |  | 1’d0 | RW | 训练记录写回寄存器；高电平写回 |
| write\_record\_trigger |  | 82’d0 | RW | 训练数据记录trigger寄存器；类型为triggerBundle |
| do\_sending |  | 1’d0 | RW | TP预取请求发送寄存器；高电平为发送中 |
| sending\_idx |  | 6’d0 | RW | TP预取请求发送计数器：0~recordThres-1  （0号对应数据不发送，为trigger） |
| sending\_data |  | 16\*39’d0 | RW | TP预取请求元数据寄存器 |
| sending\_throttle |  | 4’d0 | RW | TP预取请求节流计数器：0~tpThrottleCycles |

1. PPA

*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）*

1. 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”*

1. 【参考SinkC 模块所述写入 RefillBuffer 的条件】这种场景下设计是否存在 bug
2. 关于Acquire请求/预取请求对状态的更新是否正确（特别是针对如下情形 ①Acquire NtoB alias；② 上层缓存状态是 Branch，本层是 Tip）
3. 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”*

NA

1. 遗留问题

*需要跟踪的遗留问题*

*不涉及填“NA”*

NA