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

XX项目XX模块AS

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

昆明湖V1项目

2024年XX月XX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](#_Toc107389929)

[1.1 文档介绍 1](#_Toc107389930)

[1.2 参考文档 1](#_Toc107389931)

[1.3 术语说明 1](#_Toc107389932)

[1.4 技术背景 1](#_Toc107389933)

[2 设计规格 2](#_Toc107389934)

[3 功能描述 2](#_Toc107389935)

[4 总体设计 2](#_Toc107389936)

[4.1 整体框图 2](#_Toc107389937)

[4.2 接口列表 3](#_Toc107389938)

[4.3 接口时序 3](#_Toc107389939)

[4.4 时钟复位 4](#_Toc107389940)

[4.5 寄存器配置 4](#_Toc107389941)

[4.6 补充说明 5](#_Toc107389942)

[5 模块设计 5](#_Toc107389943)

[5.1 二级模块LoadPipe 6](#_Toc107389944)

[5.1.1 功能 6](#_Toc107389945)

[5.1.2 整体框图 6](#_Toc107389946)

[5.1.3 接口列表 6](#_Toc107389947)

[5.1.4 接口时序 6](#_Toc107389948)

[5.1.5 关键电路 6](#_Toc107389949)

[5.1.6 三级模块设计 6](#_Toc107389950)

[5.2 二级模块MainPipe 6](#_Toc107389951)

[5.3 关键电路 6](#_Toc107389952)

[5.3.1 时钟切换电路 6](#_Toc107389953)

[6 PPA优化设计 7](#_Toc107389954)

[7 验证关注点 7](#_Toc107389955)

[8 Floorplan 建议 7](#_Toc107389956)

[9 遗留问题 8](#_Toc107389957)

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*

*例如：*

本文当是XXX的AS文档，描述XXX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. XXXX
2. XXXX
   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** |
| CRU | Clock Reset Unit | 时钟复位单元 |
|  |  |  |
|  |  |  |
|  |  |  |

* 1. 技术背景

*可选项。简要介绍本模块的技术背景，比如协议说明、应用范围等。如果不需要则直接删除本小节。注意要根据模块特性进行提炼，不要大段的复制黏贴。*

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。*

*例如：*

*1. 支持从内存取值*

*2. 支持压缩指令*

*3. 支持单条指令跨预测块*

*4. 支持内存指令预译码*

*5. 支持5种类型的内存指令分支预测错误检查*

*6. 支持分支预测错误后重定向预测器*

*7. 支持从MMIO取值*

*8. 支持从外部刷新IFU流水线*

*9. 支持一次输出一个预测块的码流以及附加信息*

*10. 支持debug辅助*

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*

*IFU模块接收来自FTQ的取指令请求，并将请求发送给ICache。等到来自ICache返回至多两个缓存行的指令码后，进行切分产生取指令请求范围限定的初始指令码，并送到预译码器进行预译码和指令码扩展。下一拍根据预译码信息修正有效指令范围，同时将指令码和其他信息发送给IBuffer模块。当ICache查询地址属性发现是MMIO地址空间时，IFU需要将地址发送给MMIO处理单元取指令，这个时候处理器进入多周期顺序执行模式，IFU阻塞流水线直到收到来自ROB的提交信号时，IFU才允许下一个取指令请求的进行，同时IFU需要对跨页的MMIO地址空间32位指令做特殊处理（重发机制）。*

* 1. *功能详述*
     1. *非MMIO取指*

*非MMIO指令是指存放在内存中的指令。*

* + - 1. *取指*

*IFU可以持续接受FTQ的取值请求，请求以一个预测块为单位，这个预测块是一个地址连续的块，块里面可能有0~16条预测为不跳转指令，最多有1条预测为跳转的指令。块最大为32Byte，最多16条指令（非压缩是8条），最小为1条指令。IFU准备好后（对应的ready信号为高）接收FTQ发送出的预测块取指请求信息，包括预测块起始地址、起始地址所在cache line的下一个cache line开始地址、下一个预测块的起始地址、该预测块在FTQ里的队列指针、该预测块有无taken的CFI指令和该taken的CFI指令在预测块里的位置以及请求控制信号（请求是否有效和IFU是否ready）。*

*IFU接收到指令请求后从向icache发起取值请求，icahceh判断是否是MMIO指令，如果是会返回相应的指示，如果不是返回对应的1个或2个(取决于预测块数据是否跨cacheline，跨cacheline则需要请求2个cacheline的数据)cacheline的数据。*

*如果发生缺页异常或者是访问属性异常，icahce将会返回随机码，此时指令码无意义。*

* + - 1. *预译码*
      2. *指令扩展*
      3. *分支预测检查*

*IFU会根据预译码信息对分支预测结果进行检查，如果有错误可以提前通知预测器。主要检查的几种错误为：*

* * 预测块里面有jal指令，但是预测器预测为不跳转。（FTQ重定向到译码出来的目标地址）*
* * 预测块有ret指令，但是预测器预测为不跳转。（重定向到call调用时保存的地址）*
* * 在有效范围之内的非CFI指令被预测器预测为跳转。（重定向到PC+4）*

* 预测器指出的跳转指令的位置不是一条指令的开始位置。（重定向到PC+4）*

* * 预测器预测为跳转，但是给出的目标地址不正确。（与译码结果不符合，重定向到正确地址）。*

*如果发生了以上任意错误，当前预测块的范围会再次更新到从有效期起始地址到当前错误位置。*

*如果发生预测错误，IFU向FTQ发送预测检查错误指示信号，下一次FTQ从正确的跳转地址开始发送预测块取指请求。*

* + 1. *MMIO取指*
       1. *取指*
       2. *非完整指令处理*
       3. *回写FTQ*
       4. *重定向*

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. 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。*

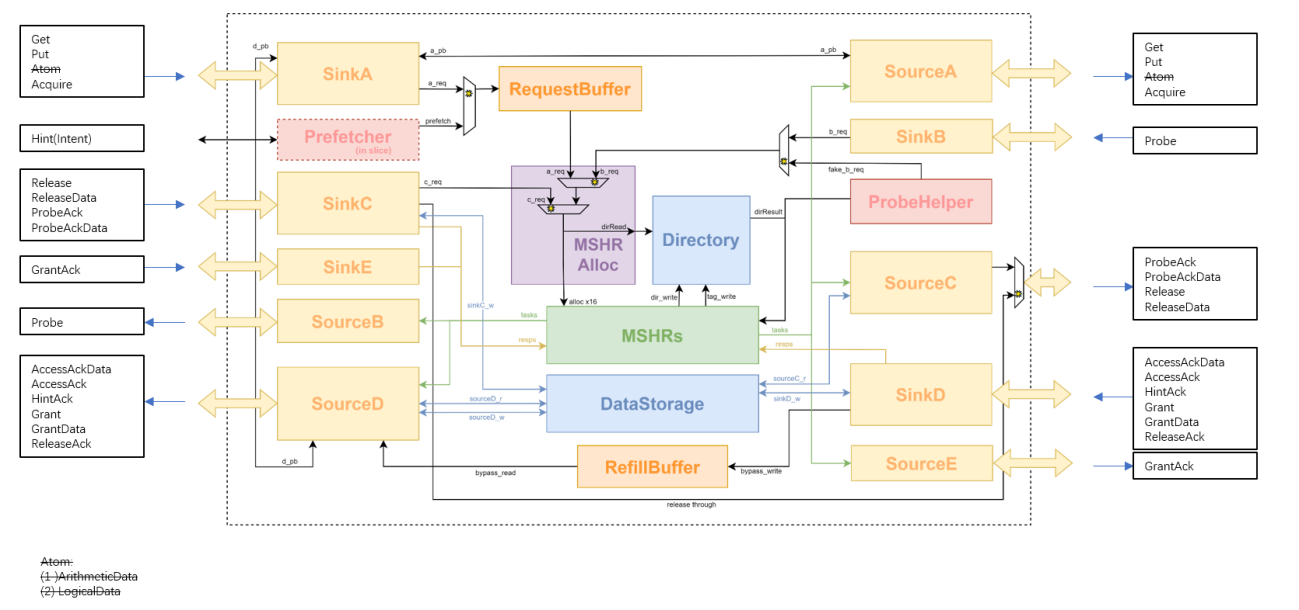
*图中不同属性的信号要注意区分开，比如控制信号与数据信号、一般信号与时钟复位信号等。*

*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。*

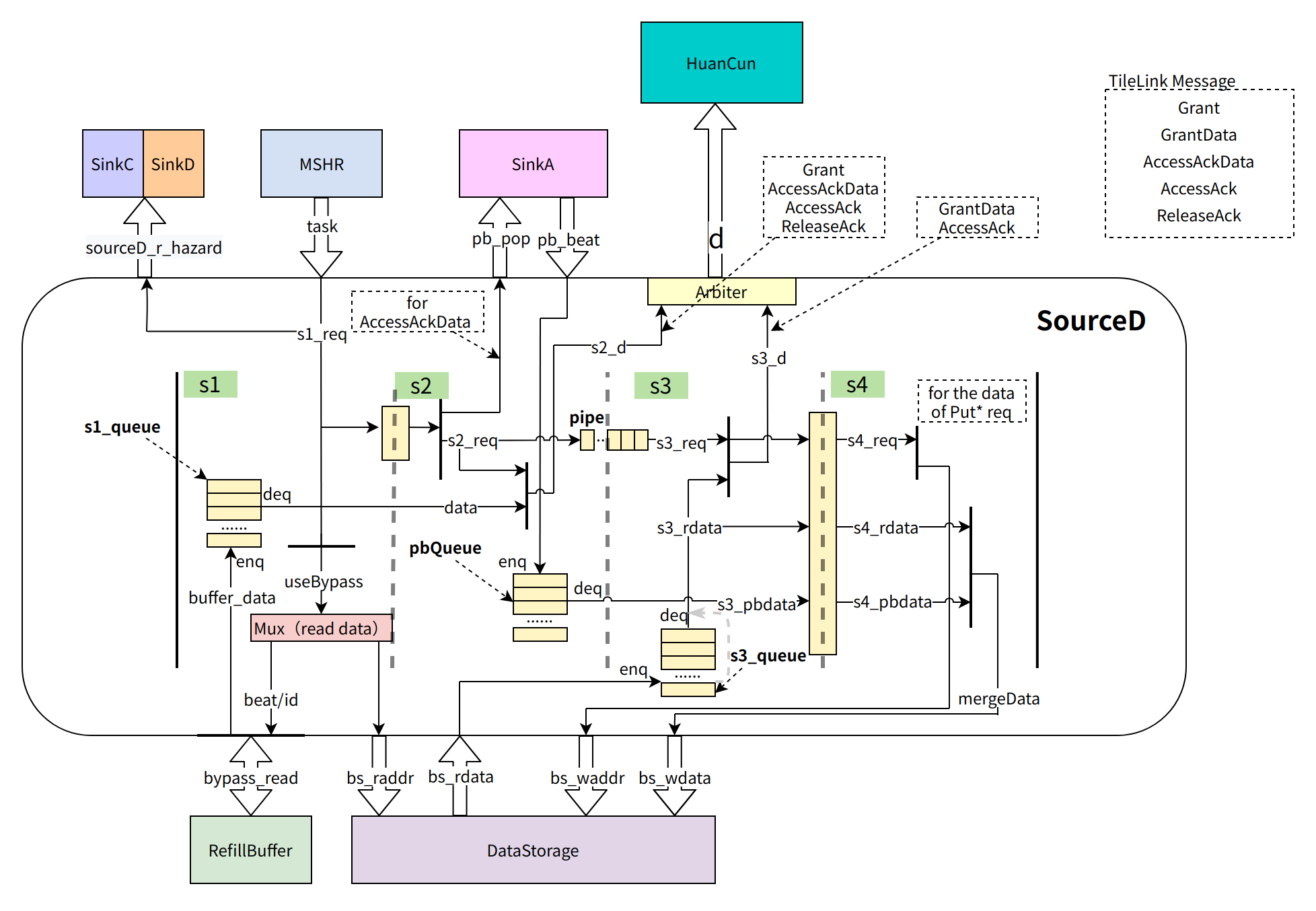
*简洁起见，图中可以使用代号标记信号组，然后在后文中详细说明每组信号组成，标准总线接口可不展开描述，比如使用axi\_m\_sig代表AXI master接口的所有信号。*

*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。*

*例如下面这些框图各自都有可取之处，可供参考：*



**



* 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））*

表4.1 XXXXIO接口列表

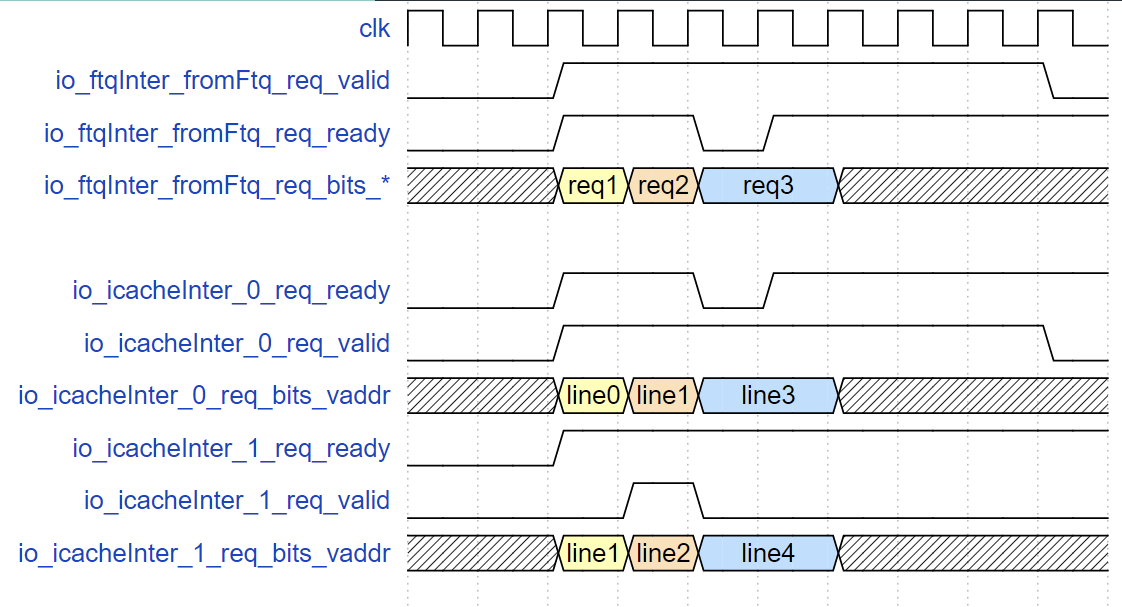
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **信号** | **方向** | **源** | **目的** | **信号组名** | **位宽** | **描述** |
| Clock | input |  |  |  | 1 | 时钟信号 |
| Reset | input |  |  |  | 1 | 异步复位信号 |
| io\_ftqInter\_fromFtq\_req\_ready | output | NewIFU | Ftq | ftq\_req | 1 | IFU可以接受请求，高有效。 |
| io\_ftqInter\_fromFtq\_req\_valid | input | Ftq | NewIFU | ftq\_req | 1 | Ftq请求有效i，高有效 |
| io\_ftqInter\_fromFtq\_req\_bits\_startAddr | input | Ftq | NewIFU | ftq\_req | 39 | Ftq请求表示的预测块起始地址。  按2byte对齐，即最低位为0。 |

* 1. 接口时序

*顶层接口时序图*

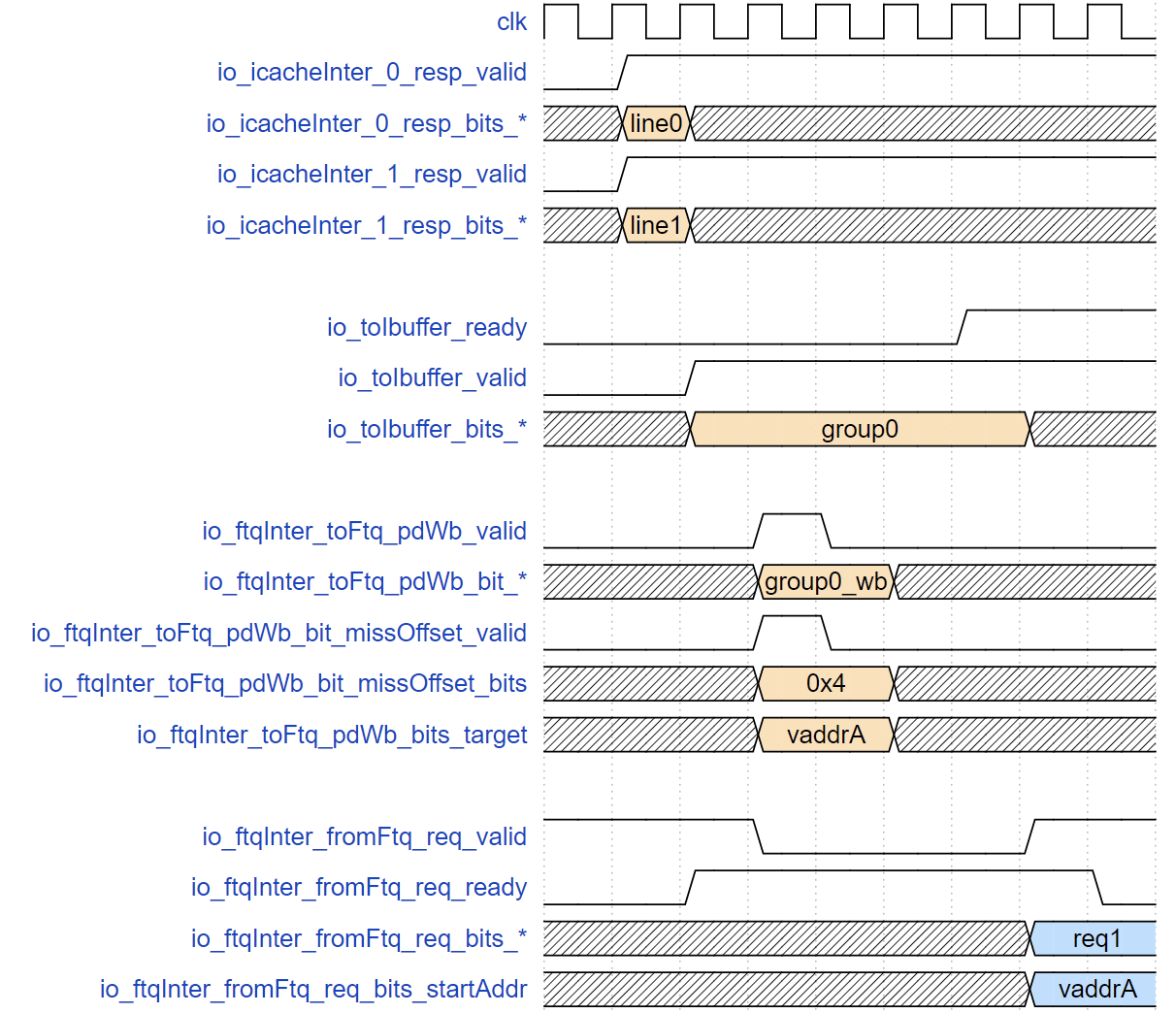
1. *非标准接口需要有关键信号的时序示意图*
2. *标准接口说明使用的接口协议即可（如AXI、APB等）*

*例:*

1. *FTQ请求接口时序示例*

*上图示意了三个FTQ请求的示例，req1只请求缓存行line0，紧接着req2请求line1和line2，当到req3时，由于指令缓存SRAM写优先，此时指令缓存的读请求ready被指低，req3请求的valid和地址保持直到请求被接收。*

1. *ICache返回接口以及到Ibuffer和写回FTQ接口时序示例*

*上图展示了指令缓存返回数据到IFU发现误预测直到FTQ发送正确地址的。。。。。*。

* 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 |

* 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*

* 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。*

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：*

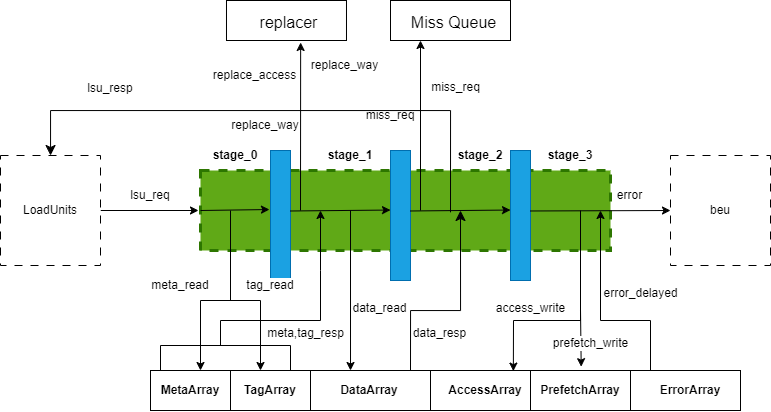
1. *Memory（ram、寄存器组、fifo等）：宽度、深度、接口含义、读写时序、data的详细描述、data在memory中存放的格式等信息*
2. *流水：有流水线框图、每一级流水线描述*
3. *仲裁：仲裁策略、优先级处理等*
4. *状态机：有状态机设计图，需要有每个状态描述、状态之间的跳转条件、复位状态等。*

*例如：*

* 1. 二级模块LoadPipe
     1. 功能

用流水线控制Load请求的处理，与Load访存流水线紧耦合，经过4级流水线读出目标数据或返回miss/replay响应。

* + 1. 整体框图



* + 1. 接口列表

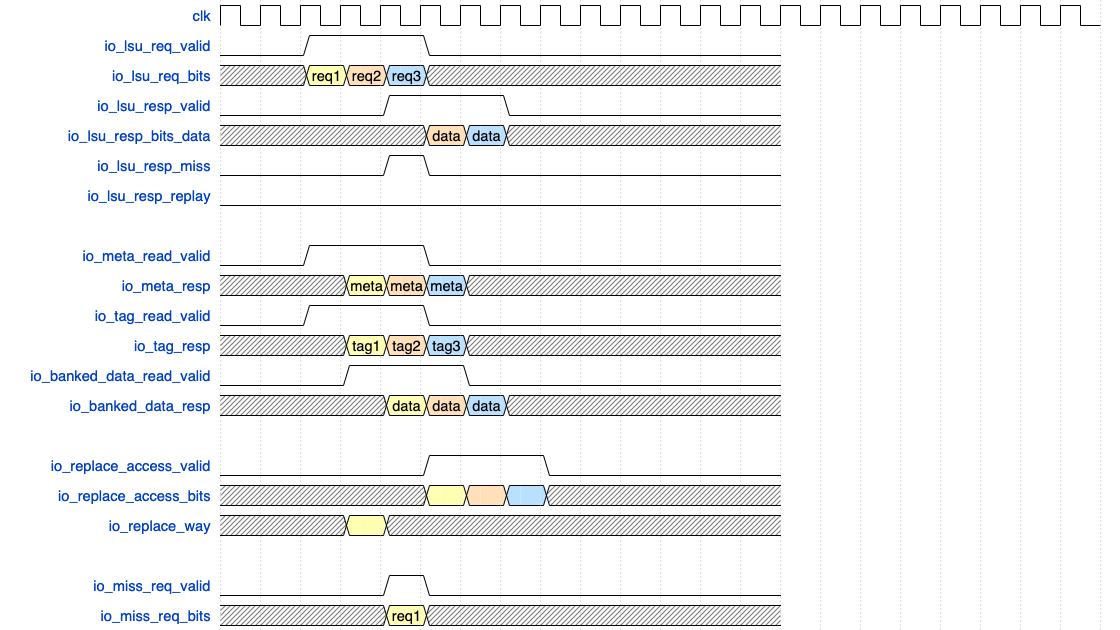
*写到 接口文档模板V1.0.xlsx 里*

【腾讯文档】io.miss.req.valid

<https://docs.qq.com/sheet/DUUZxaHVqeFVFUEZE?tab=t6row6>

* + 1. 接口时序

如下图所示，req1第一拍被loadpipe接收，读meta和tag；第二拍进行tag比较判断miss，通过PLRU替换算法选出替换路；第三拍向lsu返回响应，lsu\_resp\_miss拉高表示没有命中，暂时无法返回数据，同时向miss queue发出miss请求；第四拍检查报告是否有ecc错误，并更新replacer替换算法中的状态。req2和req3紧接着req1发出，同样在stage\_0被接收，读meta和tag；第二拍发现命中，发出data读请求；第三拍获得data，向lsu返回带load数据的响应；第四拍更新PLRU，报告ecc错误。



* + 1. 关键电路

LoadPipe 各级流水线功能：

Stage 0: 接收 LoadUnit 中流水线计算得到的虚拟地址：根据地址读tag 和 meta ;

Stage 1: 获得对应的 tag 和 meta 的查询结果；从load unit接收物理地址，进行tag比较判断是否命中；若缺失则根据PLRU信息选出替换路；根据地址读data；检查l2\_error；

Stage 2: 获得对应data结果；如果load miss则向miss queue发送miss请求，尝试分配MSHR项；向Load Unit返回load请求的响应；校验tag\_error；

Stage 3: 更新替换算法状态；向bus error unit上报1-bit ecc校验错误（包括dcache发现的data错误，dcache发现的tag错误，以及从L2获取数据块时已经存在的错误）。

* 1. 二级模块MissQueue
     1. 功能

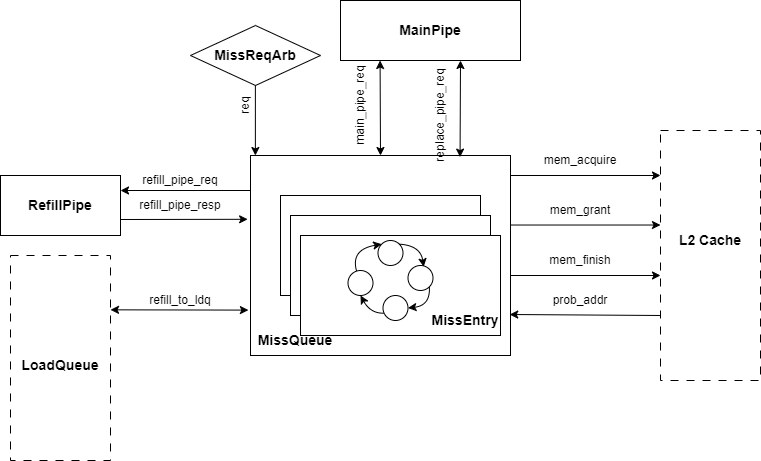
负责处理miss的load、store和原子请求，包含16项Miss Entry, 每一项负责一个请求，通过一组状态寄存器控制其处理流程。

对于miss的load请求，Miss Queue为它分配一项空的Miss Entry，并且可以在一定条件下合并请求或拒绝请求，分配后在Miss Entry中记录相关信息。根据 way\_en 所在的块是否有效, 判断是否需要替换，如果要替换则向 Main Pipe 发送 replace 请求；此外还会向 L2 发送 Acquire 请求，如果是对整个 block 的覆盖写则发送 AcquirePerm (L2 将会省去一次 sram 读操作)，否则发送 AcquireBlock；等待 L2 返回权限 (Grant) 或者数据加权限 (GrantData)；在收到 GrantData 每一个 beat 后要将数据转发给 Load Queue；在收到 Grant / GrantData 第一个 beat 后向 L2 返回 GrantAck；在收到 Grant / GrantData 最后一个 beat，并且 replace 请求已经完成后, 向 Refill Pipe 发送 refill 请求, 并等待应答, 完成数据回填；最后释放Miss Entry。

对于miss 的store请求，和load的流程基本一致, 区别在于不需要把回填的数据转发给 Load Queue，而是在最终完成回填后由Refill Pipe向 Store Buffer 返回应答, 表示 store 已完成。

对于miss的原子指令，在 Miss Queue 中分配一项空的 Miss Entry, 并在 Miss Entry 中记录相关信息；向 L2 发送 AcquireBlock 请求；等待 L2 返回 GrantData；在收到 GrantData 第一个 beat 后向 L2 返回 GrantAck；在收到 GrantData 最后一个 beat 后向 Main Pipe 发送请求, 在 Main Pipe 中同时完成替换和回填, 完成后向 Miss Entry 返回应答；最后释放 Miss Entry。

* + 1. 整体框图

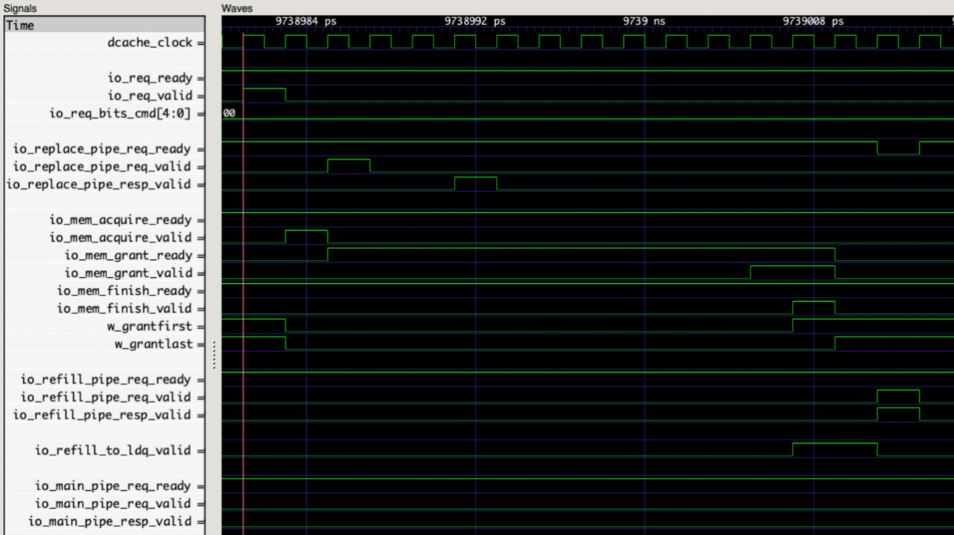


* + 1. 接口列表

*写到 接口文档模板V1.0.xlsx 里*

* + 1. 接口时序

下图展示了一个load miss请求进入MissQueue之后的接口时序。请求到达后分配miss entry，下一拍向L2发送acquire请求，第二拍再向main pipe发送replace请求；接收到grant数据的第一个beat之后，向ldq返回load响应，接收到grant数据的最后一个beat之后，下一拍向refill pipe发出回填请求。



* + 1. 关键电路

介绍 MissQueue 的入队（alloc/merge/reject 等等）逻辑及其响应逻辑；介绍 MSHR 给 LoadUnit 的数据前递逻辑；介绍预取相关信号的逻辑等。

1. MissQueue 入队处理：

MissQueue对于新入队请求，总的操作可分为响应和拒绝，而响应又可以分为分配和合并。Miss Queue 支持一定程度的请求合并, 从而提高 miss 请求处理的效率。

* 空项分配：如果新的miss请求不符合合并或者拒绝条件，则为该请求分配新的 Miss Entry。
* 请求合并条件：当已分配的 Miss Entry (请求 A) 和新的 miss 请求 B 的块地址相同时，在下述两种情况下可以将请求B合并：

1． 向L2的Acquire 请求还没有握手, 且 A 是 load 请求, B 是 load 或 store 请求，即A还未成功发起对L2的读请求前可以合并B，一起发送Acquire；

2． 向 L2 的 Acquire 已经发送出去，但是还没有收到 Grant(Data)，或收到 Grant(Data) 但还没有转发给 Load Queue，且 A 是 load 或 store 请求，B 是 load 请求，即新来的load请求可以在refill前合并，而store请求只能在acquire握手前合并。

* 请求拒绝条件：下述三种情况下需要将新的miss请求拒绝，该请求会在一定时间后重新发出：

1． 新的 miss 请求和某个 Miss Entry 中请求的块地址相同, 但是不满足请求合并条件；

2．新的 miss 请求的块和某个 Miss Entry 中请求的块地址不同, 但是在 DCache 中位于相同的slot (即具有相同的 set 和 way)；

2. MSHR给LoadUnit数据前递：

MissQueue支持数据前递，如果，lsq重发信号有效（具体重发逻辑请参考LoadQueueReplay部分，选出最合适的最老指令），在loadUnit的stage1，前递指定的mshrid以及地址，MissQueue拿到前递信息后，去比对，如果匹配，直接将重填的数据在LoadUnits的stage2传给LoadUnit，通过前递的方式更快地获得先前请求的数据，以减少加载指令的等待时间。

1. MissQueue 预取处理：
2. Miss Queue 中发出的替换请求：

这里提前进行替换操作，可以便于在收到回填数据之后立马进行写入。但是这里的替换并不是真的令替换块无效, 而是先把替换块的数据读出来, 并暂时放在写回队列中sleep。在替换请求 sleep 期间, 其他请求可以正常访问 DCache 中的替换块, 只要把对替换块的写同步一份到写回队列中即可（对Probe TtoB请求的额外检查详见writeback queue模块说明）。当得到回填数据块后，再唤醒写回队列里 sleep 的块向下release，同时Miss Queue 请求 Refill Pipe 完成回填。

* + 1. 三级模块 MissEntry
       1. 功能
       2. 整体框图
       3. 接口列表
       4. 接口时序
       5. 关键电路

1. Miss Entry 分配、合并、拒绝：

* 空项分配：如果新的miss请求不符合合并或者拒绝条件，则为该请求分配新的 Miss Entry。
* 请求合并条件：当已分配的 Miss Entry (请求 A) 和新的 miss 请求 B 的块地址相同时，在下述两种情况下可以将请求B合并：

1． 向L2的Acquire 请求还没有握手, 且 A 是 load 请求, B 是 load 或 store 请求，即A还未成功发起对L2的读请求前可以合并B，一起发送Acquire；

2． 向 L2 的 Acquire 已经发送出去，但是还没有收到 Grant(Data)，或收到 Grant(Data) 但还没有转发给 Load Queue，且 A 是 load 或 store 请求，B 是 load 请求，即新来的load请求可以在refill前合并，而store请求只能在acquire握手前合并。

* 请求拒绝条件：下述三种情况下需要将新的miss请求拒绝，该请求会在一定时间后重新发出：

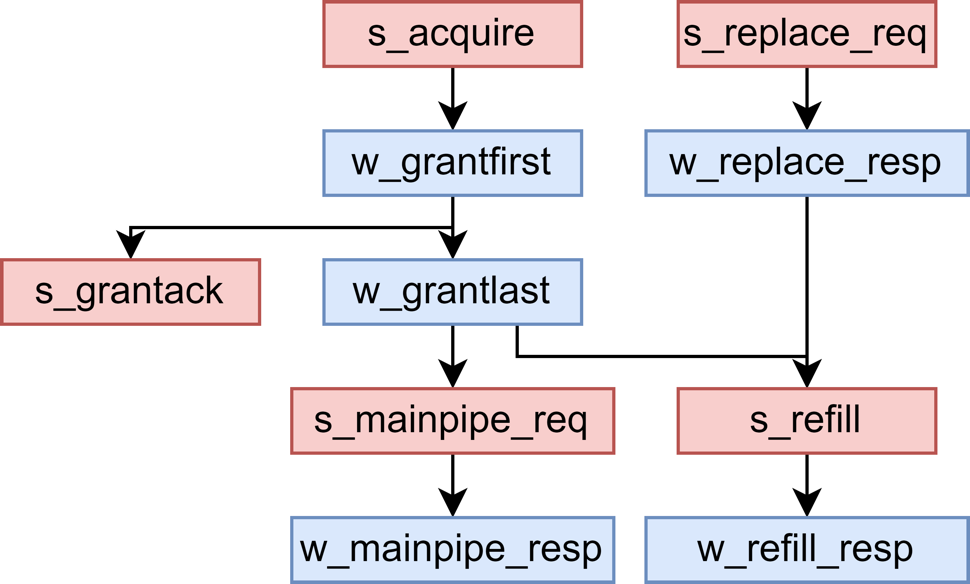
1． 新的 miss 请求和某个 Miss Entry 中请求的块地址相同, 但是不满足请求合并条件；

2．新的 miss 请求的块和某个 Miss Entry 中请求的块地址不同, 但是在 DCache 中位于相同的slot (即具有相同的 set 和 way)；

3．Miss Queue已满。

1. Miss Queue 状态设计：

Miss Entry由一系列状态寄存器控制操作的执行, 以及这些操作之间的顺序。s\_\* 寄存器表示需要调度的请求是否发送，w\_\* 寄存器表示要等待的应答是否返回，这些寄存器在初始状态下被置为 true.B, 在为请求分配一项 Miss Entry 时，会将相应的 s\_\* 和 w\_\* 寄存器置为 false.B，这表示请求还没有发出去，以及要等待的响应没有握手。

下面的图和表格展示了各个寄存器的含义描述以及执行的先后顺序：

|  |  |
| --- | --- |
| **状态** | **描述** |
| s\_acquire | 向 L2 发送 AcquireBlock / AcquirePerm请求 |
| w\_grantfirst | 等待接收到 GrantData 的第一个 beat，拉高表示接收到 |
| w\_grantlast | 等待接收到 GrantData 的最后一个 beat，拉高表示接收到 |
| s\_grantack | 在收到 L2 的数据后向 L2 返回应答, 在收到 Grant 的第一个 beat 时就可以返回 GrantAck |
| s\_mainpipe\_req | 向Main Pipe发送原子请求，将数据回填到 DCache |
| w\_mainpipe\_resp | 表示将原子请求发送到 Main Pipe 回填入DCache 后, 接收到 Main Pipe 的应答 |
| s\_replace\_req | 向MainPipe发送替换请求 |
| w\_replace\_resp | 完成替换 |
| s\_refill | load / store 请求需要发送到 Refill Pipe 进行回填 |
| w\_refill\_resp | 完成回填 |
| s\_write\_storedata | 是store请求，需要在数据从L2返回后将store\_data写入DCache |

1. MissEntry 别名处理

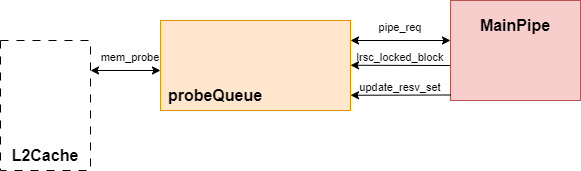
介绍 MissEntry 状态机 DAG；介绍 cache 别名有关的处理。

* 1. 二级模块 ProbeQueue
     1. 功能

负责接收并处理来自L2的一致性请求，包含 16 项 Probe Entry，每一项负责一个Probe 请求, 将 Probe 请求转成内部信号后发送到 Main Pipe, 由 Main Pipe 修改被 Probe 块的权限，等Main Pipe返回应答后释放Probe Entry。

* + 1. 整体框图

ProbeQueue只和L2通过B通道交互，以及与MainPipe互连。内部由16项entry组成，每一项通过一组状态寄存器控制请求信号的接收、转换以及发送。



* + 1. 接口列表
    2. 接口时序

下图展示了Probe Queue处理一个probe请求的接口时序，Probe Queue首先收到来自L2的probe请求，转换成内部请求并为其分配一项空的Probe Entry；经过一拍的状态转换可以向 Main Pipe 发送 probe 请求, 但由于时序考虑该请求会再被延迟一拍（probe queue里选择一项有一个arbiter， mainpipe入口也有一个arbiter选择各来源的请求，两次仲裁在一拍完成比较困难，因此在这里先锁存一拍），因此两拍后pipe\_req\_valid拉高；再下一拍即认为MainPipe返回了应答（这里实际上不需要等mainpipe返回resp），直接释放Probe Entry。

图形用户界面, 应用程序

描述已自动生成

* + 1. 关键电路

1. 别名问题：

KunMingHu架构采用了32KB的VIPT cache，从而引入了 cache 别名问题。为解决别名问题，L2 Cache的目录会维护在DCache中保存的每一个物理块对应的别名位。当DCache在某个物理地址上想要获取另一别名位的块时，L2 Cache会发起Probe请求，将DCache中原有的别名块probe下来，并且在TileLink B通道中记录其别名位。Probe Queue收到请求后会将别名位和页偏移部分进行拼接，转成内部信号发送到 Main Pipe, 由 Main Pipe 访问DCache存储模块读取数据。

2. 由原子指令引发的阻塞：

由于原子操作 (包括 lr-sc) 在 DCache 中完成，执行 LR 指令时会保证目标地址已经在 DCache 中，此时为了简化设计， LR 在 Main Pipe 中会注册一个 reservation set，记录 LR 的地址, 并阻塞对该地址的 Probe。为了避免带来死锁, Main Pipe 会在等待 SC 一定时间后不再阻塞 Probe (由参数 LRSCCycles 和 LRSCBackOff 决定), 此时再收到 SC 指令则均被视为 SC fail. 因此, 在 LR 注册 reservation set 后等待 SC 配对的时间里需要阻塞 Probe 请求对 DCache 进行操作。

介绍别名有关的处理；介绍 LR/SC 有关的阻塞逻辑

* + 1. 三级模块 ProbeEntry
       1. 功能
       2. 整体框图
       3. 接口列表
       4. 接口时序
       5. 关键电路

介绍 Probe 处理状态机

* 1. 二级模块 MainPipe
     1. 功能

用流水线控制Store, Probe, Replace以及原子操作的执行（即所有需要争用writeback queue向下层cache发起请求/写回数据的指令）。

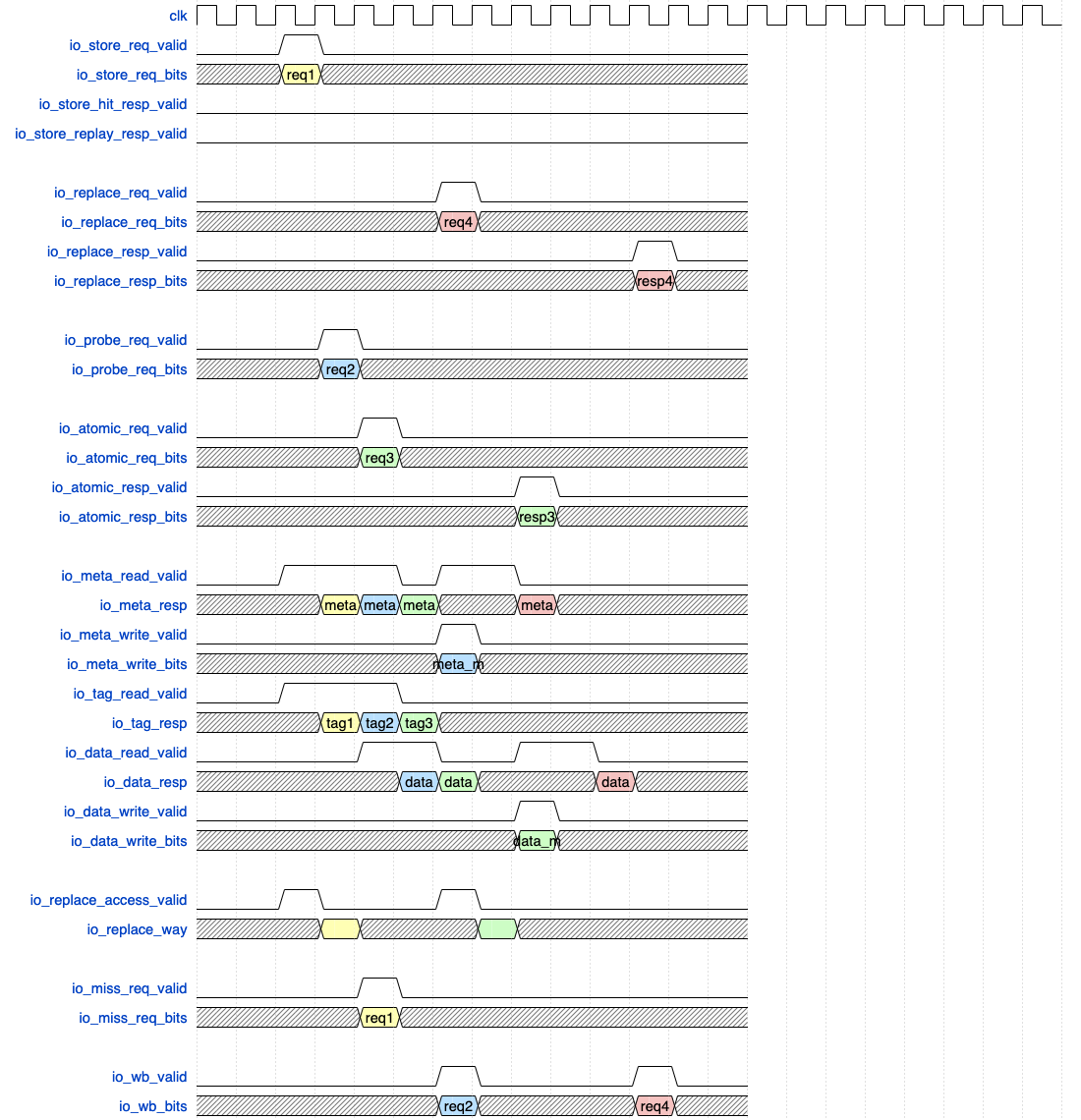
* + 1. 整体框图

*日程表

描述已自动生成*

* + 1. 接口列表
    2. 接口时序

接口时序如下图所示，req1为store请求，第一拍读meta和tag，第二拍进行tag比较发现请求miss，根据替换算法选出要替换的路，第三拍将miss请求发送给miss queue，第四拍因为miss，不会向store buffer返回响应。req2为probe请求，第一拍读meta和tag，第二拍读data，第三拍获取probe数据块结果，第四拍根据probe命令更新meta，并向writeback queue发起wb请求，返回probeAck应答。req3是amo指令，第一拍读meta和tag，第二拍进行tag比较命中，发出data读请求，第三拍获得data结果，第四拍和第五拍都处于stage\_3流水级，第四拍执行指令运算，第五拍发出data写操作更新原数据块内容，并向atomic unit返回响应。req4为req1对应的replace请求，miss queue发来replace\_req的第一拍读meta，由于此时req2正在进行meta写，而metaArray写优先于读，req4在stage\_0停留一拍，下一拍才能成功握手，第三拍stage\_1读data，由于此时req3正在进行data写，再在stage\_1停留一拍，第五拍stage\_2获取要被替换的数据块data，第六拍stage\_3向writeback queue发起wb请求，尝试让替换块提前进入wb队列，并向miss queue返回replace响应。



* + 1. 关键电路

1 MainPipe 各级流水线完成的功能：

Stage 0：仲裁传入的 Main Pipeline 请求选出优先级最高者；根据请求信息判断请求所需的资源是否就位；发出 tag, meta 读请求

Stage 1：获得 tag, meta 读请求的结果；进行 tag 匹配检查, 判断是否命中；如果需要替换, 获得 PLRU 提供的替换选择结果；根据读出的 meta 进行权限检查；提前判断是否需要执行 miss queue 访问

Stage 2：获得读 data 的结果, 与要写入的数据拼合；如果 miss, 尝试将这次请求信息写入 miss queue；检查tag\_error和l2\_error

Stage 3：根据操作的结果, 更新 meta, data, tag；如果命中则向lsu返回store响应；如果指令需要向下层 cache 发起访问/写回数据, 则在这一级生成 writeback queue 访问请求, 并尝试写 writeback queue；检查data\_error；对原子指令的特殊支持：AMO 指令在这一级停留两拍, 先完成 AMO 指令的运算操作, 再将结果写回到 dcache并返回响应；LR/SC 指令会在这里设置/检查其 reservation set queue。

2 mainpipe争用和阻塞：

Main Pipeline 的争用存在以下优先级: probe\_req > replace\_req > store\_req > atomic\_req

一个请求只有在其所请求的资源全部就绪, 不存在 set 冲突, 且没有比它优先级更高的请求的情况下才会被接受. 来自 committed store buffer 的写请求由于时序原因, 拥有单独的检查逻辑。

1. set阻塞逻辑：

确保并行执行的指令不会同时访问到同一个set中的不同行，以维护数据一致性和正确性即防止s3（或者s1,s2）处理的数据还没写完，s0进来的数据没读对这类情况发生。在各个阶段valid有效的情况下，MainPipe的set冲突要比对 s0和s1的，s0和s1，s0和s2的地址索引是否一致，如果一致则是出发了set冲突。阻塞s0。

1. meta更新
2. AMO指令处理
3. MainPipe写回

介绍各级流水线的关键设计点，包括不限于：

介绍各级流水线的功能

介绍 s0 的请求仲裁逻辑

介绍 s0 的 set 阻塞逻辑，并介绍阻塞的原因

介绍 s3 不同请求分别如何更新 meta

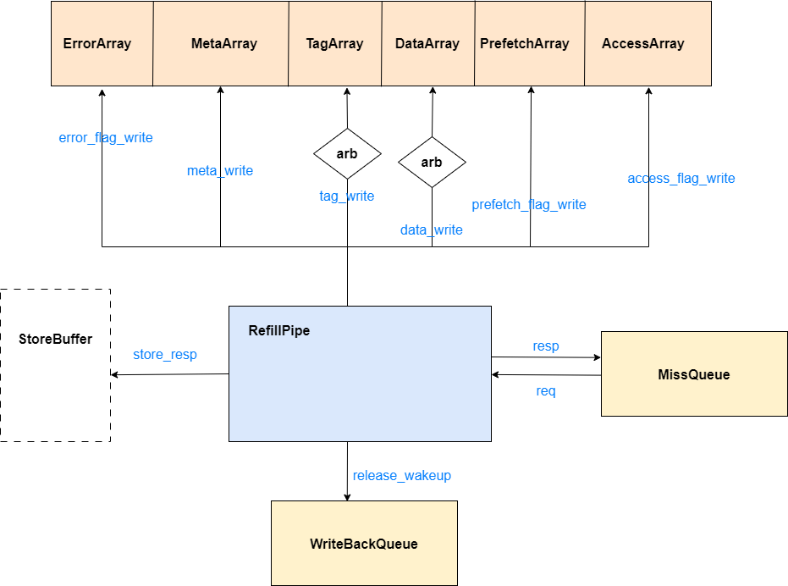
介绍 LR/SC 与 AMO 指令的处理

介绍写回逻辑，那些请求需要进入 wbq

* 1. 二级模块 RefillPipe
     1. 功能

RefillPipe专门负责把从L2 Cache返回的需要回填的数据块写入DCache，无需经过多拍主流水线，而是直接用一拍向DCache存储部件发出写请求，完成写操作。

* + 1. 整体框图



* + 1. 接口列表
    2. 接口时序

如下图所示，Refill Pipe在接收到回填请求时已经选出替换路并且获取替换数据块，因此当拍就可以向dataArray等发出写请求，向WritebackQueue发出release请求，向MissQueue返回响应，仅需一拍来完成对DCache的写操作。

*图形用户界面

描述已自动生成*

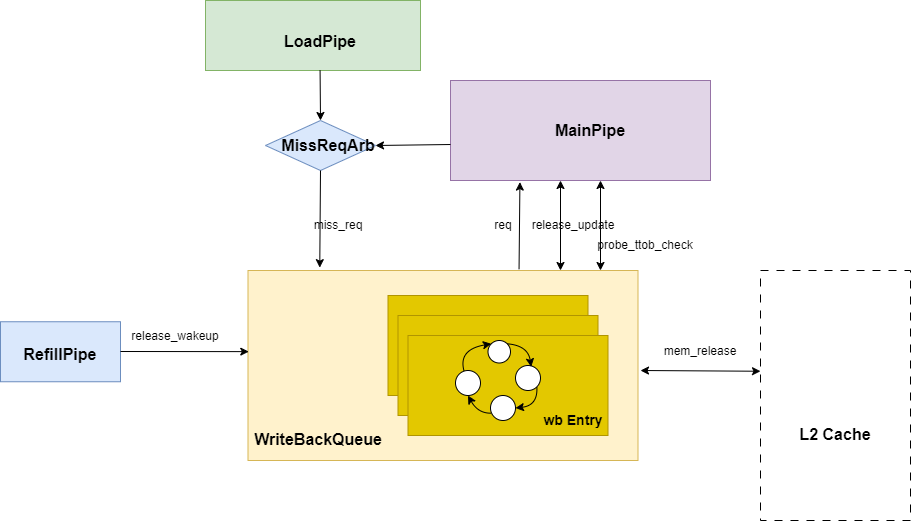
* + 1. 关键电路

由于Main Pipe和Refill Pipe都可以写入缓存数据，为保障读写一致性，Refill Pipe在下述情况下会被阻塞而暂时不能写入：

1. Refill Pipe的请求和Main Pipe的Stage 1存在set冲突；
2. Refill Pipe的请求和Main Pipe的Stage 2 / Stage 3存在set冲突且有相同的路使能信号way\_en。
   1. 二级模块WritebackQueue
      1. 功能

Writeback Queue包含18项WritebackEntry，负责通过TL-C的C通道向L2 Cache写回替换块(Release)，以及对Probe请求做出应答 (ProbeAck)，且支持Release和ProbeAck之间相互合并以减少请求数目并优化时序。

* + 1. 整体框图



* + 1. 接口列表
    2. 接口时序
    3. 关键电路

1. Writeback Queue空项分配、合并与拒绝：

为了时序考虑, 在 wbq 满的时候无论新请求是否能被合并都会被拒绝; 而当 wbq 不满的时候所有请求都会被接收, 此时或者为新请求分配空项, 或者将新请求合并到已有的 Writeback Entry 中, 后面在状态维护部分将会看到 Writeback Entry 任何时候都可以合并进新的 Release 或 ProbeAck 请求。因此 NanHu 架构中判断写回队列能否入队只需要看队列有没有空项即可。

2. 请求阻塞条件：

TileLink 手册对并发事务的限制要求如果 master 有 pending Grant (即还没有发送 GrantAck), 则不能发送相同地址的 Release. 因此所有 miss 请求在进入 Miss Queue 时如果发现和 Writeback Queue 中某一项有相同地址, 则该 miss 请求会被阻塞.

4. 对Probe TtoB请求的额外检查：

对于提前放入写回队列的release替换块，如果后续请求对该缓存块进行操作，通常的操作是在main pipe中处理后将写同步到写回队列的release块中。特别地，对于访问同一缓存块的Probe请求，如果Probe在Release发生之前到来，wb\_queue可以将ProbeAck合并到Release的wb\_entry中（toN），而同时在main pipe中的处理可能会对原cache\_line的一致性meta进行修改（toB），这会导致L1和L2中该缓存行的一致性状态不一致。因此，对于所有的probe请求需要在MainPipe里做额外的检查再更新meta\_coh，如果是TtoB probe请求，且wb queue中存在相应的cache\_line release请求等待写回，那么需要在main pipe中把对应的meta\_coh设为N。

5. WritebackQueue 项数必须大于 MissQueue 项数，因为每一个miss的请求通常对应一个需要writeback的替换块，而wb\_queue除了写回替换块之外，还需要对probe请求做出应答。为了避免请求在wbq造成拥堵，wbq的项数需要大于missqueue项数。

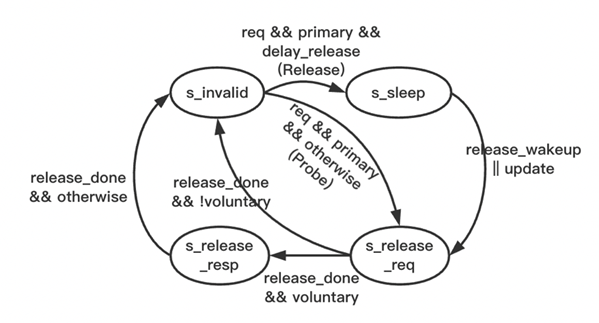
包括不限于：

介绍 wbq 入队逻辑（alloc/merge/reject）

1. 介绍对 Probe TtoB 请求的额外检查
2. 介绍 wbq 项数要求（必须大于 MissQueue 项数）及其原因
   * 1. 三级模块 WritebackEntry
        1. 功能
        2. 整体框图
        3. 接口列表
        4. 接口时序
        5. 关键电路

状态设计：Writeback Entry中的状态机设计如下图表所示：

|  |  |
| --- | --- |
| **状态** | **描述** |
| s\_invalid | 复位状态，该 Writeback Entry 为空项 |
| s\_sleep | 准备发送Release请求, 但暂时sleep并等待refill请求唤醒 |
| s\_release\_req | 正在发送 Release 或者 ProbeAck 请求 |
| s\_release\_resp | 等待ReleaseAck请求 |



介绍状态机，包括一般情况下的 ProbeAck 与 Release 处理流程，以及各种请求之间合并的处理流程。

* 1. 关键电路

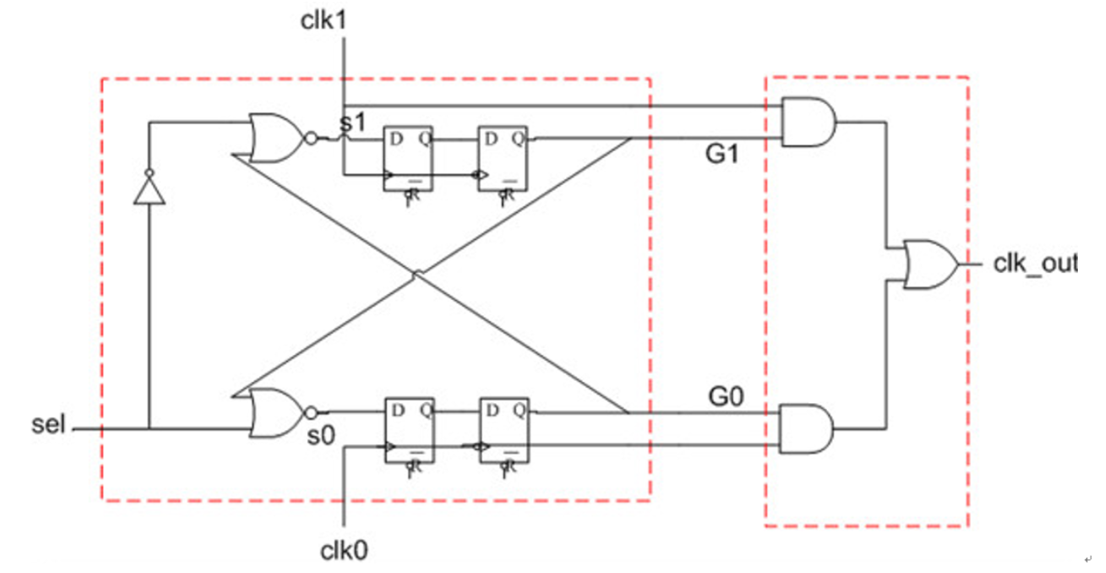
*例如：*

* + 1. 时钟切换电路

上述CRG框图中，紫色粗框内的clock\_mux是动态时钟切换模块，在用户程序配置PLL之后，通过配置寄存器cfg\_clk\_sel=1将系统时钟从晶振动态切换到PLL时钟。

为了保证时钟动态切换不会导致系统出错，需要使用无毛刺时钟切换电路，电路图如下：

图 2 无毛刺时钟切换电路图



上图中，下面两个寄存器的复位值为1，上面两个寄存器的复位值为0。复位时，clk\_out默认选择clk0时钟。

对于两级同步寄存器，同步器的第一级采用时钟上升沿触发，第二级采用时钟下降沿触发。

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. 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”*

1. 遗留问题

*需要跟踪的遗留问题*

*不涉及填“NA”*