文档编号：

保密级别：内部

Cache 设计文档

Cache Design

第 X 卷: L1DCache 设计文档

Book X: MSHR File Design

版本说明

版本号	日期	作者	描述

目 录

图目录

表目录

1 概述

一级数据缓存（L1 DCache）是位于 CPU 与二级缓存（L2 Cache）之间的小容量高速缓冲存储部件。L1 DCache 通过合理的组织结构和映象规则，根据高效的块查找方法、替换算法和写策略等，能够以对用户程序透明的方式实现数据的基本存储访问过程，提高系统的整体性能。通过提供一套较为完善的

Cache 操作，能够满足操作系统在系统初始化、进程创建、切换和终止等过程中所需要的基本功能。

L1 DCache 部件通过与 L2 Cache、存储管理部件 (MMU)等的协同工作，能够填补 CPU 和主存在速度上的差距，提高处理器访问主存的平均速度，完成数据的快速高效访问。

1.1 功能简述

1.1.1 宏观特性

L1 DCache 有如下特性：

大小为 16KB，组织结构为 4 路组相连，总共 64 组

每个 Cache 行包括 16 个字（64 Bytes）、Tag（20 bits）、状态位（2 bits）

每个 Cache 行又分为 4 个子块，每个子块大小为 16 Bytes

双端口 SRAM、多 Bank 设计

采用虚拟索引、物理标签（Virtual-Index、Physical-Tag）寻址和 L1 DTLB 并行查找

采用 Pseudo-Least-Recently-Used（Pseudo-LRU）替换算法

采用写返回（Write-Back）与按写分配（Write-Allocate）的写策略

支持 Misses under Misses 非阻塞访问

四级的 Cache 流水操作

支持 ECC 校验检查

支持 Modified/Exclusive/Shared/Invalid（MESI）一致性协议

支持原子存储操作

支持松散（relax）序模型？？？（chendongwei）

1.1.2 主要功能

L1 DCache 的主要功能包括：

读 Data Array 和 Meta Array

Data Array 包含存储 cache 数据主体的物理实体（Data SRAM Wrapper），其作用是对 Data SRAM Wrapper 进行相应的读写操作。

Meta Array 包含存储 cache tag 域和 state 域的物理实体（Metadata SRAM Wrapper），其作用是对 Metadata SRAM Wrapper 实体进行相应的读写操作。

DTLB 进行虚实地址转换

DTLB（Data Translation Look-aside Buffers），其作用是内存页表的缓存，因此又称为“快表”。其主要功能是虚实地址转换，并在转换过程中进行地址检查、权限检查等。其接收 Cache 发出的访存请求（访存地址、访存读写类型和 passthrough 机制使能信号），若 DTLB 命中则输出转换后的物理页号给 Cache，否则向 PTW 请求失效页表项，等待 PTW 返回页表项并更新当前 DTLB 内容。

命中检查

命中检查分两步完成：Tag 比较、一致性检查。从 Meta Array 读出四路 Tags，将它们与 DTLB 转换得到 Tag 进行比较，若 Tag 比较不

命中，判断为失效。若 Tag 比较命中，检验命中行的一致性状态是否能满足当前操作，若一致性不满足，判断为失效。

数据校验

对命中行的数据进行 ECC 校验，它能够检测并纠正单比特错误和检测多比特错误。若检测错误，将纠正的数据写回 Data Array，而此次请求被发往第一级流水线来重新访问 Cache。

读返回：当前是读 Cache 请求，若 Cache hit 并且校验正确，则将数据返回给 CPU

写操作：当前是写 Cache 请求，若 Cache hit 并且校验正确，则进行写操作，修改后的数据后续被存入 Data Array。

失效处理

命中检查单元判定当前请求失效，该请求将被 MSHRFile 模块保存并处理。MSHRFile 对失效情况进行分类，不同情况的失效将进行执行不同的处理过程：

- Tag 比较不命中且为简单失效

  Tag 比较不命中，替换行不是 dirty 态，则替换行不用写回下一级存储，称这种失效为简单失效。

- Tag 比较不命中且为复杂失效

  Tag 比较不命中，替换行是 dirty 态，需要将替换行回写到下一级存储，称这种情况为复杂失效。

- Tag 比较命中且只需修改本地 Cache 一致性信息

  当前是写请求，Cache 行的状态是 exclusive 态，需要迁移到

dirty 态，此时，只需要修改本地 Cache 的一致性信息。

■ Tag 比较命中且需要修改其它 Cache 一致性信息

当前是写请求，Cache 行的状态是 shared 态，需要先 invalidate 其他存储对这个数据块的备份，再将本 Cache 变成 dirty 态。

脏行的回写操作

若 Tag 比较不命中，当前请求将被 MSHRFile 保存并处理，MSHRFile 会根据替换行的状态来判断其是否为脏行，若替换行是脏行，则 MSHRFile 请求 WriteBack 模块将脏行回写。WriteBack 模块请求 Data Array 读取替换行，并控制发送给下一级存储。

Cache 行的填充操作

Tag 比较不命中，或命中行为 invalid 态，Cache 会请求下一级存储返回数据。下一级存储响应并返回数据后，由 MSHRFile 控制将数据回填到 Data Array 中。Cache 和下一级存储的数据交换基本单位是 Beat（16 Bytes），而一个 Cache Block 是 64 Bytes，所以，回填一个 Cache Block 需要进行四次数据传输。

1.2 结构框图

L1 DCache 与其他模块之间的模块接口示意如图 1. 2-1 所示

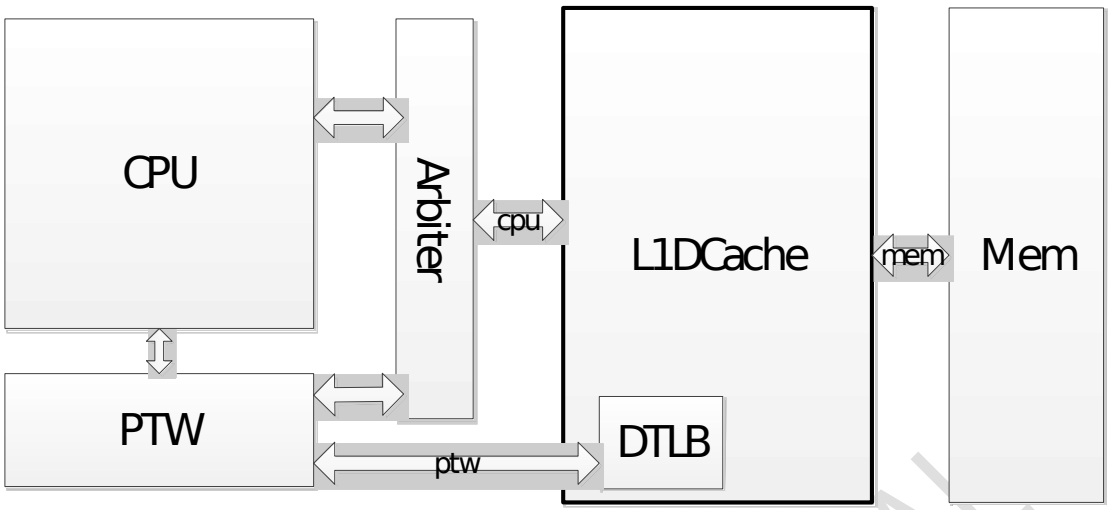

图 1.2-1 L1 DCache 与相关模块接口

1.3 接口信号

L1 DCache 连接的模块有 CPU、PTW（Page Table Working）和下一级存储（通常是 L2 Cache），图 1-1 的 Mem 表示下一级存储。DTLB 实现在 L1 DCache 里，它需要和 PTW 配合来完成页表管理和虚实地址转换等工作。CPU 和 PTW 都要请求 L1 DCache 读写内存数据，若请求的数据不在 L1 DCache 中，L1 DCache 需要请求 Mem 获取数据。

L1 DCache 对外接口有三类：CPU 和 PTW 请求 L1 DCache 的接口、L1 DCache 中 DTLB 与 PTW 交互接口、L1 DCache 请求 Mem 的接口。三类接口具体说明见 3.2 节。

2 总体结构

2.1 内部结构框图

L1 DCache 内部可划分为四级流水线部分和其它三个辅助功能模块 ,包括：MSHRFile、Prober 和 WriteBack 模块。LIDCache 内部结构框图如图 2.1-1 所示。

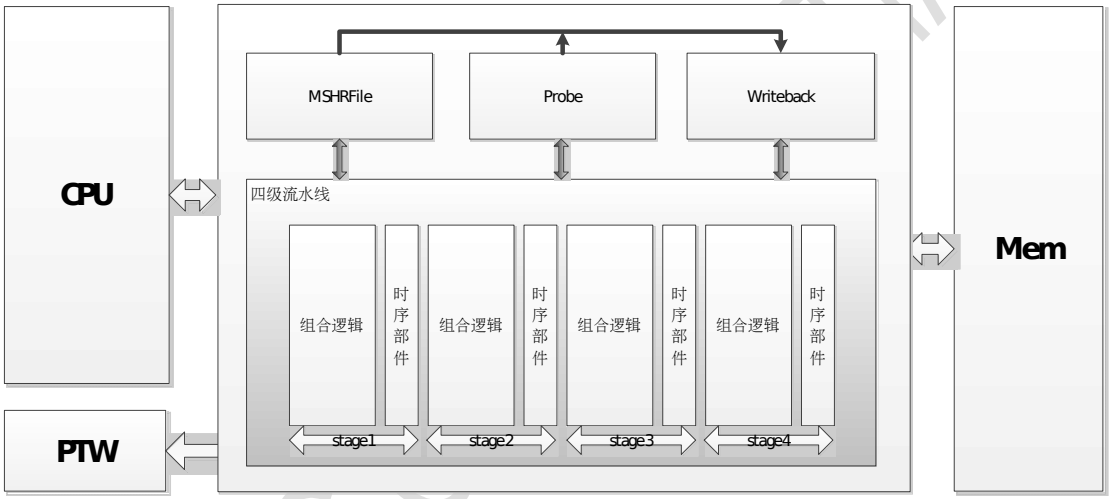

图 2.1-1 L1 DCache 内部结构框图

2.2 全局信号结构

整个 L1 DCache 使用同样一个时钟，即系统时钟。

整个 L1 DCache 使用同样一个异步 Reset 信号。

3 L1 DCache 设计

3.1 模块描述

L1 DCache 的数据放在 Data Array 体中，一致性信息和物理标签放在 Meta Array 体中，这两个 Cache 存储体都是双端口，即读端口和写端口分开。L1 DCache 采用虚拟索引物理标签的设计策略，并且将 dtlb 设计到该 Cache 内部。

L1 DCache 采用流水线设计，各级流水的功能描述如下：

第一级流水 :用请求地址的 idx 字段索引读 Data Array 和 Meta Array 体，获得一组（set）的数据、标签和一致性信息。

第二级流水：dtlb 将请求地址的虚拟标签转化为物理标签，再用此物理标签进行 tag 比较来取得命中路。

第三级流水：判断是否命中，若 Cache hit 并且当前是读 Cache 请求，则进行数据校验并将数据返回给 CPU；若 Cache hit 并且当前是写 Cache 请求，则将数据校验并进行写操作；若 Cache miss，则将失效请求发送给 MSHR 模块，MSHR 模块保存失效请求并进行相应失效处理。

第四级流水：将修改后的数据写入 Cache 存储体。

L1 DCache 除了主流水线部分，还包括 MSHRFile、Probe 和 WriteBack

等三个子模块，这三个模块的功能描述如下：

MSHRFile：保存由 Cache miss 或一致性不满足而导致访问失效的请求，并且控制处理失效，处理完毕后，将保存的请求重新发往 Cache 流水线进行 Cache 访问。

Probe：控制多核 Cache 一致性，比如，其它核的 Cache 对某个数据块进行了写操作，它通过给本地 Probe 模块发送 invalidate 信号来清除本地 Cache 对那个数据块的缓存 ;其它核的 Cache 访问某个数据块发生 Cache miss，若它访问的数据块只在本地 Cache 中有缓存，它会请求本地 Probe 模块将该数据块写回 memory ,memory 再将数据块发送给请求的那个处理器核。

WriteBack：控制将数据块（Cache block）写回下一级存储，比如，MSHRFile 请求 Writeback 模块写回替换行；Probe 请求 Writeback 写回指定数据块。

下图 3.1-1 是 L1 DCache 的内部结构见图：

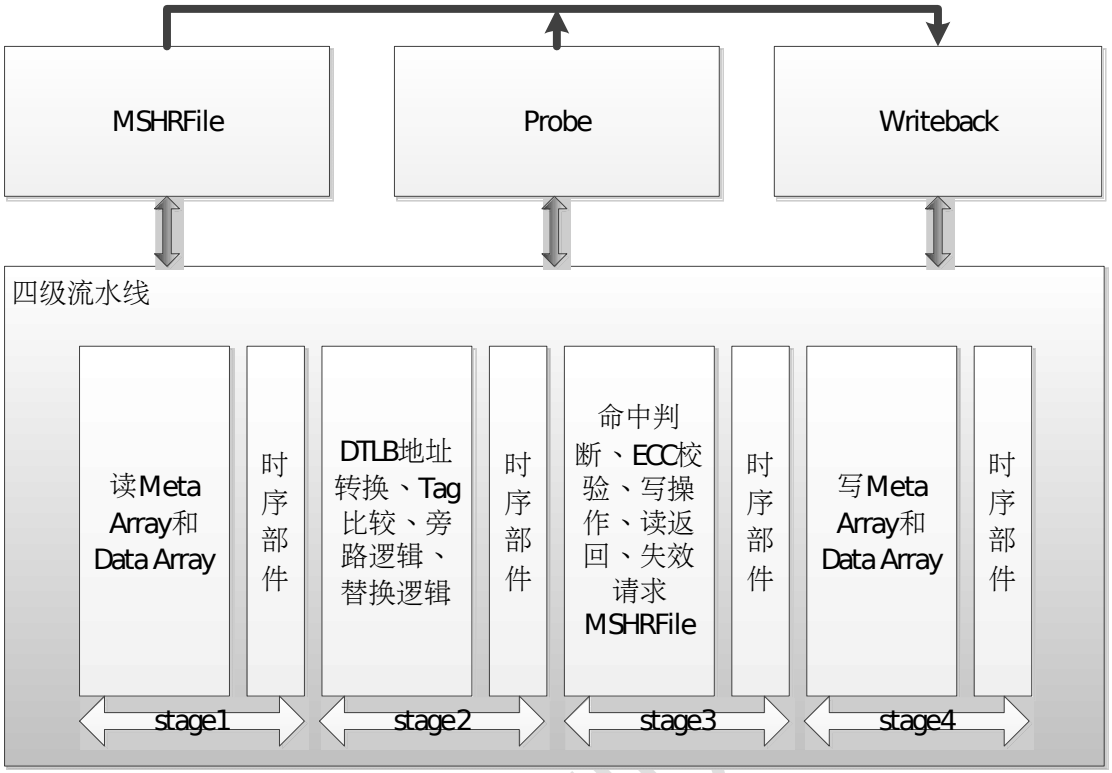

图 3.1-1 L1 DCache 内部结构简图

3.2 接口信号说明

L1 DCache 数据通路与外部模块的接口信号，如表 3.2-1 所示：

表 3.2-1 L1 DCache 数据通路与外部模块的接口信号

端口名称	方向	宽度	相连模块名称	作用	描述
clk	in	1		时钟信号	上升沿有效
Reset	in	1		复位信号	0:复位 1:正常工作
io_cpu_req_ready	out	1		通知 cpu 是	0:cpu 不能

			否可以发送请求	发送请求 1:cpu 可以发送请求
io_cpu_req_bits_addr	in	40	请求的地址	
io_cpu_req_bits_tag	in	9		
io_cpu_req_bits_cmd	in	5	命令类型	
io_cpu_req_bits_typ	in	3	操作数粒度	
io_cpu_req_bits_kill	in	1	请求是否被杀掉	0:不被杀掉 1:被杀掉
io_cpu_req_bits_phys	in	1	请求地址是否是实地址	0:不是实地址 1:是实地址
io_cpu_resp_valid	out	1	Cache 的相应信息是否有效	0:无效 1:有效
io_cpu_resp_bits_addr	out	40	响应请求的地址	
io_cpu_resp_bits_tag	out	9		
io_cpu_resp_bits_cmd	out	5	相应请求的命令类型	
io_cpu_resp_bits_typ	out	3	响应请求的操作数粒度	

io_cpu_resp_bits_data	out	64	响应请求的数据
io_cpu_resp_bits_nack	out	1	是否是否定应答 0:肯定应答 1:否定应答
io_cpu_resp_bits_replay	out	1	是否是 replay 的 0:否 1:是
io_cpu_resp_bits_has_data	out	1	是否给 cpu 返回数据 0:不返回 1:返回
io_cpu_resp_bits_data_word_bypass	out	64	未经处理的数据或者处理过的一个 word 的数据
io_cpu_resp_bits_store_data	out	64	写入 cache 体的数据
io_cpu_replay_next_valid	out	1	???
io_cpu_replay_next_bits	out	9	???
io_cpu_xcpt_ma_ld	out	1	是否发生读地址未对齐异常 0:否 1:是
io_cpu_xcpt_ma_st	out	1	是否发生写地址未对齐异常 0:否 1:是

io_cpu_xcpt_pf_ld	out	1	tlb miss 或者请求读数据访问权限不够	0:没有发生该异常 1:发生了改异常
io_cpu_xcpt_pf_st	out	1	tlb miss 或者请求写数据访问权限不够	0:没有发生该异常 1:发生了改异常
io_cpu_invalidate_lr	in	1		
io_cpu_ordered	out	1		
io_ptw_req_ready	In	1	PTW 就绪信号	0：未就绪 1：就绪
io_ptw_req_valid	Out	1	TLB 请求有效信号	0：无效 1：有效
io_ptw_req_bits_addr	Out	27	TLB 请求的虚拟页号	27 位的 VPN
io_ptw_req_bits_prv	Out	2	当前机器所处的特权级	00:user 01:supervise 10:hypervisor 11:machine
io_ptw_req_bits_store	Out	1	Store 操作标识	0：非 store 操作 1：store 操

				作
io_ptw_req_bits_fetch	Out	1	取指操作标识	0：非取指操作 1：取指操作
io_ptw_resp_valid	In	1	PTW 响应有效信号	0：无效 1：有效
io_ptw_resp_bits_error	In	1		
io_ptw_resp_bits_pte_ppn	In	20	PTE 物理页号	20 位物理页号
io_ptw_resp_bits_pte_reserved_for_software	In	3	PTE 软件保留域	2 位保留域
io_ptw_resp_bits_pte_d	In	1	PTE Dirty 位	0：页表非脏 1：页表脏
io_ptw_resp_bits_pte_r	In	1	PTE Readable 位	0：页表可读 1：页表不可读
io_ptw_resp_bits_pte_typ	In	4		
io_ptw_resp_bits_pte_v	in	1	PTE Valid 位	0：页表项无效 1：页表项有效
io_ptw_status_sd	in	1	mstatus sd 位域，用	0：无需保存 用户扩展内

			于上下文保存和恢复	容到内存 1：需保存用户扩展内容到内存
io_ptw_status_zero2	in	31		
io_ptw_status_sd_rv32	in	1		
io_ptw_status_zero1	in	9		
io_ptw_status_vm	in	5	mstatus vm 位域，显示当前的虚拟存储方式	0：Mbare 1：Mbb 2：Mbbid 3-7：reserved 8：Sv32 9：Sv39 10：Sv48 11：Sv57 12：Sv64 13-31：reserved
io_ptw_status_mprv	in	1		
io_ptw_status_xs	in	2	mstastus 的 xs 位域，用于保存所有附加用户模式扩展和	0:ALL off 1:None dirty or clean，some on 2:None dirty，some

			关联的当前状态	clean 3:Some dirty
io_ptw_status_fs	in	2	mstastus 的 fs 位域，用于保存浮点单元的当前状态	0:off 1:Initial 2:clean 3:dirty
io_ptw_status_prv3	in	2		
io_ptw_status_ie3	in	1		
io_ptw_status_prv2	in	2		
io_ptw_status_ie2	in	1		
io_ptw_status_prv1	in	2		
io_ptw_status_ie1	in	1		
io_ptw_status_prv	in	2		
io_ptw_status_ie	in	1		
io_ptw_invalidate	in	1		
io_mem_acquire_ready	in	1	下一级存储器是否准备好	1:准备后 0:未准备好
io_mem_acquire_valid	out	1	Mem_req 信号组是否有效	1:有效 0:无效
io_mem_acquire_bits_addr_block	out	26	tag+index	
io_mem_acquire_bits_client_xact_id	out	2	哪个 mshr 的请求	
io_mem_acquire_bits_addr_beat	out	2	一个 line 共有 4 个	

			beat, 标 识 哪 一 个 beat
io_mem_acquire_bits_is_builtin_type	out	1	
io_mem_acquire_bits_a_type	out	3	
io_mem_acquire_bits_union	out	17	
io_mem_acquire_bits_data	out	128	
io_mem_grant_ready	out	1	是否准备好 1:准备好 接收下一级 0:未准备好 存储的响应
io_mem_grant_valid	in	1	下一级存储 1:相应有效 的响应信号 0:响应无效 是否有效
io_mem_grant_bits_addr_beat	in	2	返回的数据 是 哪 一 个 beat
io_mem_grant_bits_client_xact_id	in	2	响应哪一个 mshr
io_mem_grant_bits_manager_xact_id	in	4	
io_mem_grant_bits_is_builtin_type	in	1	
io_mem_grant_bits_g_type	in	1	
io_mem_grant_bits_data	in	128	下一级存储 返回的数据
io_mem_probe_ready	out	1	
io_mem_probe_valid	in	2	
io_mem_probe_bits_addr_block	in	26	请求地址块

			的地址
io_mem_probe_bits_p_type	in	2	请求 prober 的类型
io_mem_release_ready	in	1	
io_mem_release_valid	out	1	
io_mem_release_bits_addr_beat	out	2	请求的第几个 beat
io_mem_release_bits_addr_block	out	26	请求 release 的地址块
io_mem_release_bits_client_xact_id	out	2	接收 release 信号的 client id 号
io_mem_release_bits_voluntary	out	1	
io_mem_release_bits_r_type	out	3	请求 release 的类型
io_mem_release_bits_data	out	128	请求 release 的数据
init	in	1	

3.3 内部结构

L1 DCache 整体的内部结构，如图 3.3-1 所示：

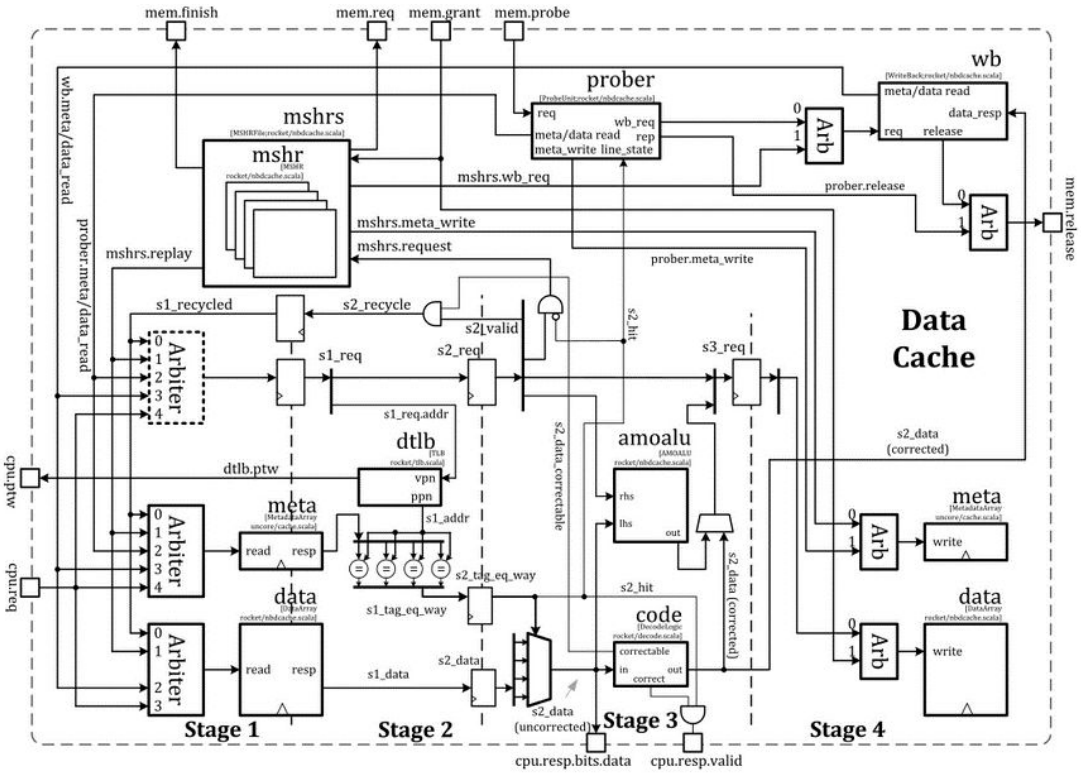

图 3.3-1 L1 DCache 数据通路图

L1 DCache 的内部可以划分为如下 10 个子功能单元，简述如下：

- **Meta Array**：包含存储 cache tag 域和 state 域的物理实体（Metadata SRAM Wrapper），其作用是对 Metadata SRAM Wrapper 实体进行相应的读写操作。
- **Data Array**：包含存储 cache 数据主体的物理实体（Data SRAM Wrapper），其作用是对 Data SRAM Wrapper 进行相应的读写操作。
- **DTLB**：完成虚实地址转换（虚拟页号 VPN 到物理页号 PPN 转换），将转换后的 PPN 发给 Cache，转换过程中进行地址检查、权限检查等异

常检测操作。

- **ECC 校验单元**：其功能是对读出的数据进行 ECC 校验，.它能够检测并纠正单比特错误和检测多比特错误。
- **AMOALU 单元**：该模块有两个功能分别是处理原子存储操作指令和普通存储指令。AMOALU 模块支持 RISC-V 提供的所有的 AMO 指令，包括整数加、逻辑 AND、逻辑 OR、逻辑 XOR 和有符号、无符号整数最大值和最小值。另一方面 AMOALU 模块根据数据操作类型对写 cache 体的数据进行相应的操作
- **Tag 域比较单元**：其功能是比较从 Tag SRAM 中读出的 Tag 域和访问 DCache 的物理地址 Tag 域，判断 DCache 访问是否命中。
- **替换行选择单元**：当进行 Reload 操作时，根据 PLRU 替换算法产生所需要替换的行。
- **MSHRFile** :保存由 Cache miss 或一致性不满足而导致访问失效的请求，并且控制处理失效，处理完毕后，将保存的请求重新发往 Cache 流水线进行 Cache 访问。
- **Probe**：控制多核 Cache 一致性，比如，其它核的 Cache 对某个数据块进行了写操作，它通过给本地 Probe 模块发送 invalidate 信号来清除本地 Cache 对那个数据块的缓存 ;其它核的 Cache 访问某个数据块发生 Cache miss，若它访问的数据块只在本地 Cache 中有缓存，它会请求本地 Probe 模块将该数据块写回 memory ,memory 再将数据块发送给请求的那个处理器核。
- **WriteBack**：控制将数据块（Cache block）写回下一级存储，比如，

MSHRFile 请求 Writeback 模块写回替换行；Probe 请求 Writeback 写回指定数据块。

DCache 设计为四级流水，以下具体介绍这些子功能单元和主流水线设计。

3.3.1 DCache 四级流水线

3.3.1.1 流水线第一级组合设计

3.3.1.1.1 功能描述

Stage1 模块的对外接口信号如下图所示

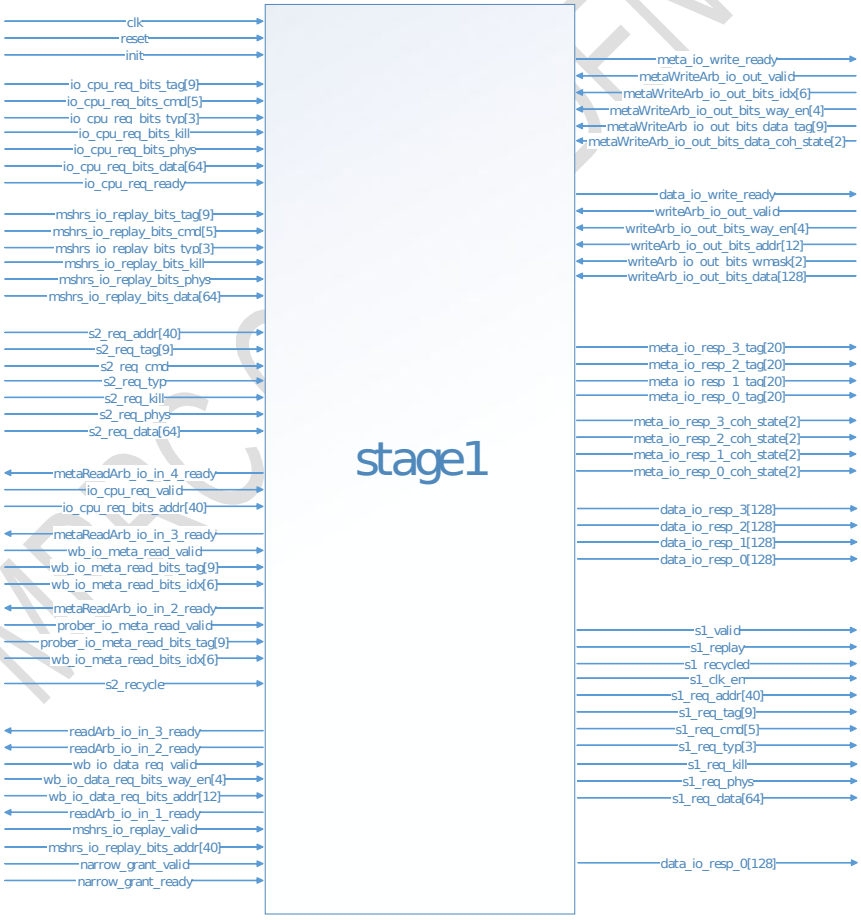

图 3.3.1-1 Stage1 模块的对外接口信号

本级流水完成下面几个功能:

1. **metaReadArb 仲裁读 meta 的请求**: metaReadArb 仲裁器从五个请求中仲裁出一个请求：这五个请求分别是 s2_recycle，mshrs_io_meta_read，probe_io_meta_read，wb_io_meta_read，io.cpu.req。这五个请求的优先级依此降低，优先级高的请求会被优先选择接受。
2. **readArb 仲裁读 data 的请求:** readArb 仲裁器从四个请求中仲裁一个请求：这四个请求分别是 s2_recycle，mshrs_io_replay，wb_io_data_req，io_cpu_req。这四个请求的优先级依此降低，优先级高的请求会被优先选择接受。
3. **流水线寄存器的寄存**：state1/stage2 流水线寄存器包括如下：
   1) s1_valid:标志当前是否有来自 cpu 的请求
   2) s1_replay:标志当前是否有来自 mshrs 的请求
   3) s1_recycled:标志当前是否有来自 s2_recycle 的请求
   4) s1_clk_en:标志 metaReadArb 是否接受了一个有效的请求
   5) s1_req_addr:接受的请求要访问的内存地址，来源有 5 个
   6) s1_req_phy:接受的请求的 s1_req_addr 是否是物理地址，来源有 5 个
   7) s1_req_tag:接受的请求的 tag，来源有 3 个
   8) s1_req_cmd:接受的请求的命令类型，来源有 3 个
   9) s1_req_typ:接受的请求的读写粒度，来源有 3 个
   10) s1_req_kill:接受的请求是否要 kill 掉上一次的请求，来源有 3 个
   11) s1_req_data:接受的请求（对于写请求）要写的数据，来源有 3 个

3.3.1.1.2 数据通路图

第一级流水线数据通路如图 3.3.1-1 所示。

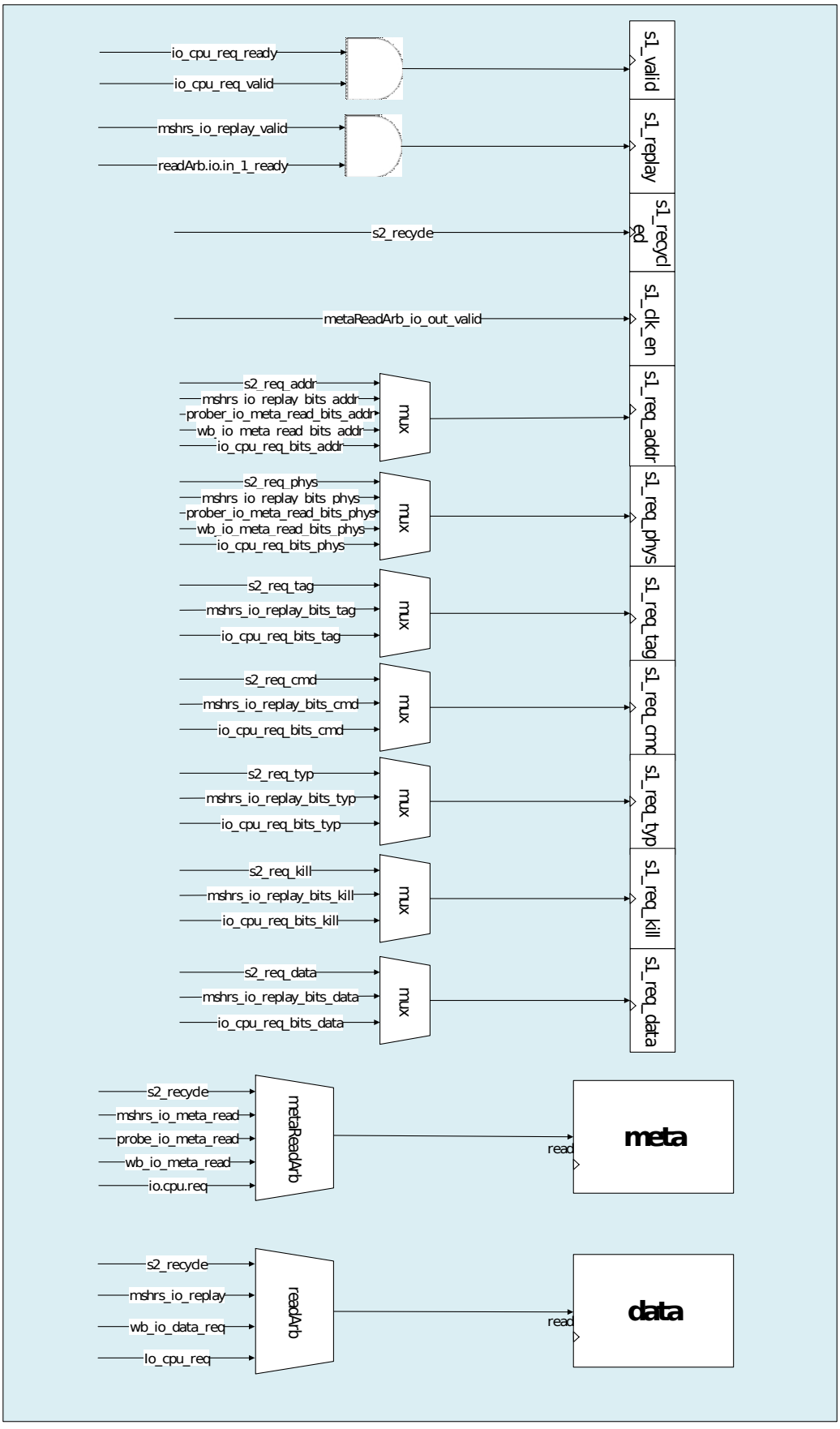
io_cpu_req_ready
io_cpu_req_valid
s1_valid
mshrs_io_replay_valid
readArb.io.in_1_ready
s1_replay
s2_recycle
s1_recycled
metaReadArb_io_out_valid
s1_clk_en
s2_req_addr
mshrs_io_replay_bits_addr
prober_io_meta_read_bits_addr
wb_io_meta_read_bits_addr
io_cpu_req_bits_addr
mux
s1_req_addr
s2_req_phys
mshrs_io_replay_bits_phys
prober_io_meta_read_bits_phys
wb_io_meta_read_bits_phys
io_cpu_req_bits_phys
s1_req_phys
s2_req_tag
mshrs_io_replay_bits_tag
io_cpu_req_bits_tag
s1_req_tag
s2_req_cmd
mshrs_io_replay_bits_cmd
io_cpu_req_bits_cmd
s1_req_cmd
s2_req_typ
mshrs_io_replay_bits_typ
io_cpu_req_bits_typ
s1_req_typ
s2_req_kill
mshrs_io_replay_bits_kill
io_cpu_req_bits_kill
s1_req_kill
s2_req_data
mshrs_io_replay_bits_data
io_cpu_req_bits_data
s1_req_data
s2_recycle
mshrs_io_meta_read
probe_io_meta_read
wb_io_meta_read
io.cpu.req
metaReadArb
read
meta
s2_recycle
mshrs_io_replay
wb_io_data_req
Io_cpu_req
readArb
read
data

图 3.3.1-2 第一级流水的数据通路

3.3.1.2 流水线第一级输出寄存器

第一级流水线输出寄存器描述如表 3.3.1-1 所示。

表 3.3.1-1 DCache 流水线第一级输出寄存器

寄存器名称		宽度	功能	描述
s1_valid		1	当前是否有来自 cpu 的请求	1：有 0：无
s1_replqy		1	当前是否有来自 mshr 的请求	1: 有 0: 无
s1_recycle		1	当前是否有来自 s2_recycle 的请求	1:有 0:无
s1_clk_en		1	当前 stage1 的 metaReadArb 是否接受了一个有效的请求	1:是 0:否
s1_req	addr	39	当前请求要读内存的地址	有可能是虚拟地址 ,也有可能是物理地址
	tag	9	指令标签	
	cmd	5	操作类型	操作类型及编码见表 3.3.5-3
	typ	3	操作粒度	操作粒度及编码见表 3.3.5-2
	kill	1	当前请求是否要杀掉上一个	0：否

		cpu 请求	1：是
phys	1	是否为实地址请求	0：否 1：是
data	64	对于写请求来说要写的数据	

3.3.1.3 流水线第二级组合设计

3.3.1.3.1 功能描述

Stage2 的模块对外就接口信号如下图所示：

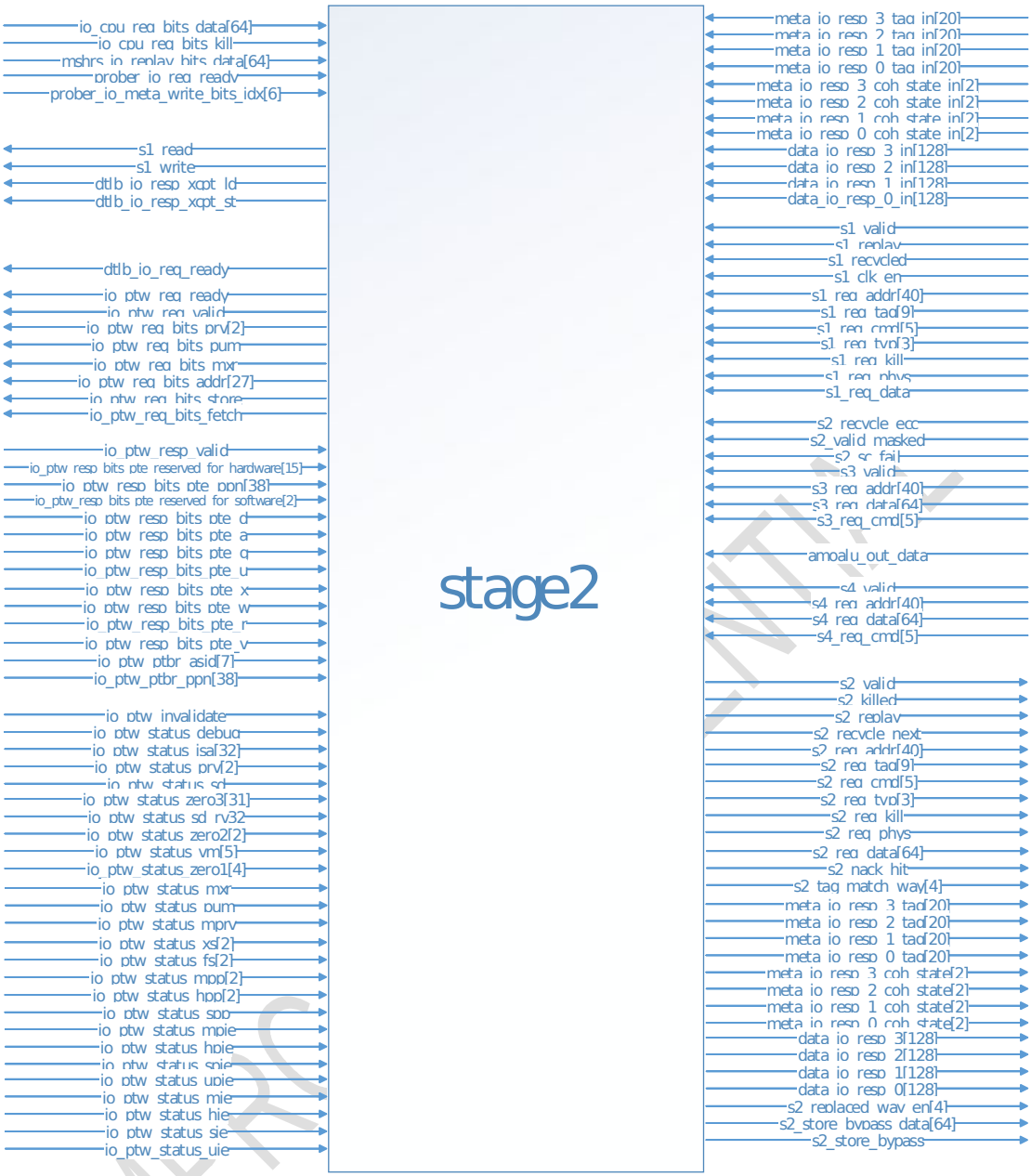

图 3.3.1-3 Stage2 的模块对外就接口信号

本级流水线主要完成下面几个功能:

dtlb 完成虚实地址转换:dtlb 将输入的 28 位虚拟页号转成 20 位的物理页号，然后与原来地址的低 12 位拼成 40 位的物理地址，其中高 8 位补 0。

tag 比对: 经 dtlb 得到物理地址 tag 与 meta 中的一组 tag（4 路）

进行比较,看是否匹配,并得到相应的 OneHot 编码寄存。

从 data 中取数据: 在上面过程进行中会同时根据 index 字段从 data 体中取出一组（4 路）的数据集，寄存到 data_io_resp 寄存器中。

判断是否存在 s1_nack: s1_nack（否定应答）包括两种情况,一种是 dtlb 发生了 miss,dtlb 会通过 dtlb_io_resp_miss 信号为 1 反应出来. 另外一种是此时当 prober 处于非 invalid 状态时，请求读 meta 的地址与 prober 要写往 meta 的地址相同,发生了冲突。

替换策略（伪 LRU）产生一个要替换出去的位置: 当前的请求是有效的但是发生了 miss（没有一路命中）,替换车略 replace 通过伪 LRU 算法产生一个 onehot 编码,标识要替换出去的是哪一块。

对请求读写的数据进行旁路: 当前请求是读请求时很容易理解，当前请求是写请求时，因为要先动 data 中读出要写的数据对应的单元，经过 amoalu 合并后，然后写到 data 体重。此时从 data 体重读的数据单元并不一定是最新的，最新的有可能还没有写到 data 体重，所以也要进行 bypass 处理。bypass 数据的来源有三个分别是: amoalu_out_data, s3_req_data, s4_req_data. 当读请求的地址与要写往 data 体中数据的地址的高 37 位相同时,旁路逻辑用一个优先级多选去选择出最新的数据,旁路的数据会寄到到 s2_store_bypass_data 中去。

3.3.1.3.1 数据通路图

第二级流水线数据通路如图 3.3.1-2 和 3.3.1-3 所示。

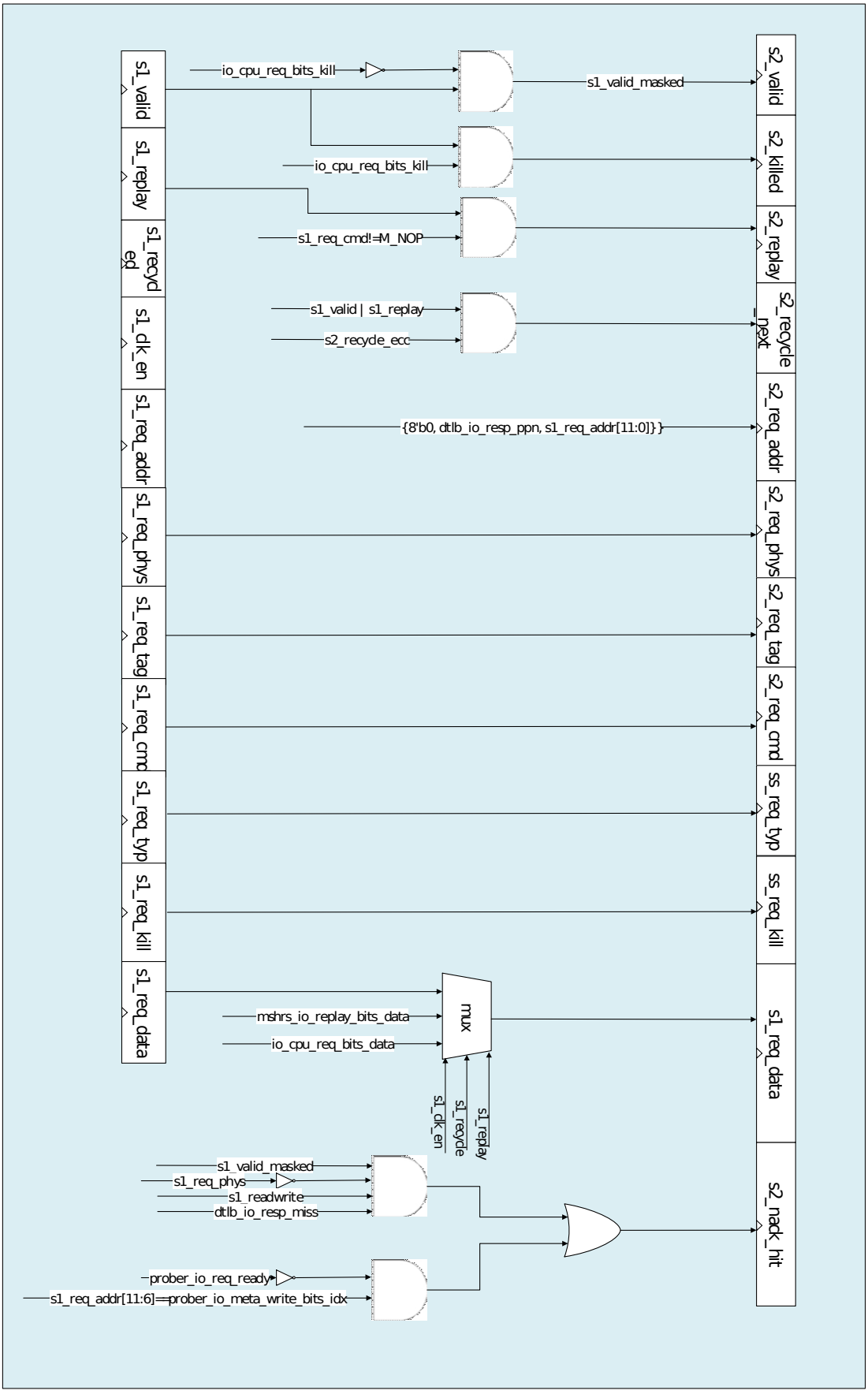
s1_valid
s1_replay
s1_recycled
s1_clk_en
s1_req_addr
s1_req_phys
s1_req_tag
s1_req_cmd
s1_req_typ
s1_req_kill
s1_req_data
io_cpu_req_bits_kill
io_cpu_req_bits_kill
s1_req_cmd!=M_NOP
s1_valid | s1_replay
s2_recycle_ecc
s1_valid_masked
{8'b0, dtlb_io_resp_ppn, s1_req_addr[11:0]}
mshrs_io_replay_bits_data
io_cpu_req_bits_data
mux
s1_clk_en
s1_recycle
s1_replay
s1_valid_masked
s1_req_phys
s1_readwrite
dtlb_io_resp_miss
prober_io_req_ready
s1_req_addr[11:6]==prober_io_meta_write_bits_idx
s2_valid
s2_killed
s2_replay
s2_recycle_next
s2_req_addr
s2_req_phys
s2_req_tag
s2_req_cmd
ss_req_typ
ss_req_kill
s1_req_data
s2_nack_hit

图 3.3.1-4 第二级流水的数据通路

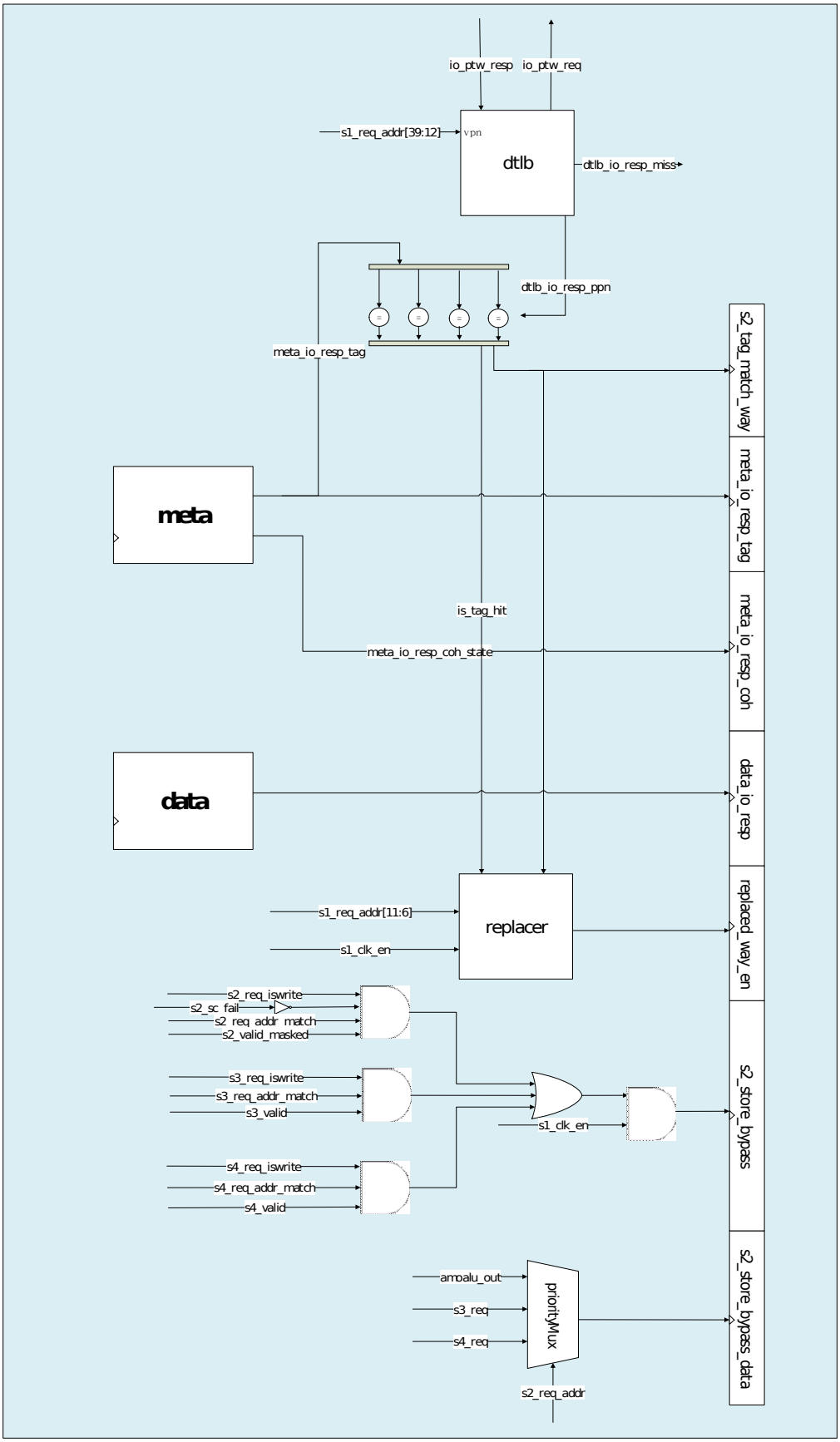

图 3.3.1-5 第二级流水的数据通路

3.3.1.4 流水线第二级输出寄存器

第二级流水线输出寄存器描述如表 3.3.1-2 所示。

表 3.3.1-2 DCache 流水线第二级输出寄存器

寄存器名称	宽度	功能	描述
s2_valid	1	有效的 cpu 请求信号且没有被 kill 掉	1：有效 0：无效
s2_replay	1	从 s1_replay 寄存器流出来判断不是空操作并再次寄存	1: 有效 0: 无效
s2_killed	1	当前有效的 cpu 请求是否被下下一个 cpu 请求 kill 掉’	1:被 kill 掉 0:未被 kill 掉
s2_recycle_next	1	保存下一级流水线的 s2_recycle_ecc 信号	1:有效 0:无效
s2_nack_hit	1	表明第二级流水线阶段是否发生 nack,如果 dtlb miss 或者读 meta 的地址和 probe 写往 meta 的地址发送冲突,则说明该流水级发生了 nack	1:第二级流水线发生了 nack 0:第二级流水线未发生 nack
meta_io_resp_0_coh_state	2	第 0 路 cache 块的一致性状态信息	
meta_io_resp_1_coh_state	2	第 1 路 cache 块的一致性状态信息	

meta_io_resp_2_coh_state		2	第 2 路 cache 块的一致性状态信息	
meta_io_resp_3_coh_state		2	第 3 路 cache 块的一致性状态信息	
meta_io_resp_0_tag		20	第 0 路 cache 块的 tag 信息	
meta_io_resp_1_tag		20	第 1 路 cache 块的 tag 信息	
meta_io_resp_2_tag		20	第 2 路 cache 块的 tag 信息	
meta_io_resp_3_tag		20	第 3 路 cache 块的 tag 信息	
s2_repalce_way_en		4	替换策略产生的要替换的哪一块的 onehot 编码	为 1 的位置对应为要替换出去的块
s2_store_bypass_data		64	旁路逻辑得到的要旁路出去的数据	
s2_req	addr	40	tlb 转后后得到物理页号，与 s1_req_addr 的低 12 位拼接得到物理地址，高 8 位补 0	
	tag	9	指令标签	
	cmd	5	操作类型	操作类型及编码见表 x-x
	typ	3	操作粒度	操作粒度及编码见表 x-x
	kill	1	是否要 kill 掉上一个 cpu 请求	0：否 1：是
	phys	1	地址是否为物理地址	0：不是

			1：是
	data	64	写请求操作也写的数据

3.3.1.5 流水线第三级组合设计

3.3.1.5.1 功能描述

Stage3 的模块对外就接口信号如下图所示：

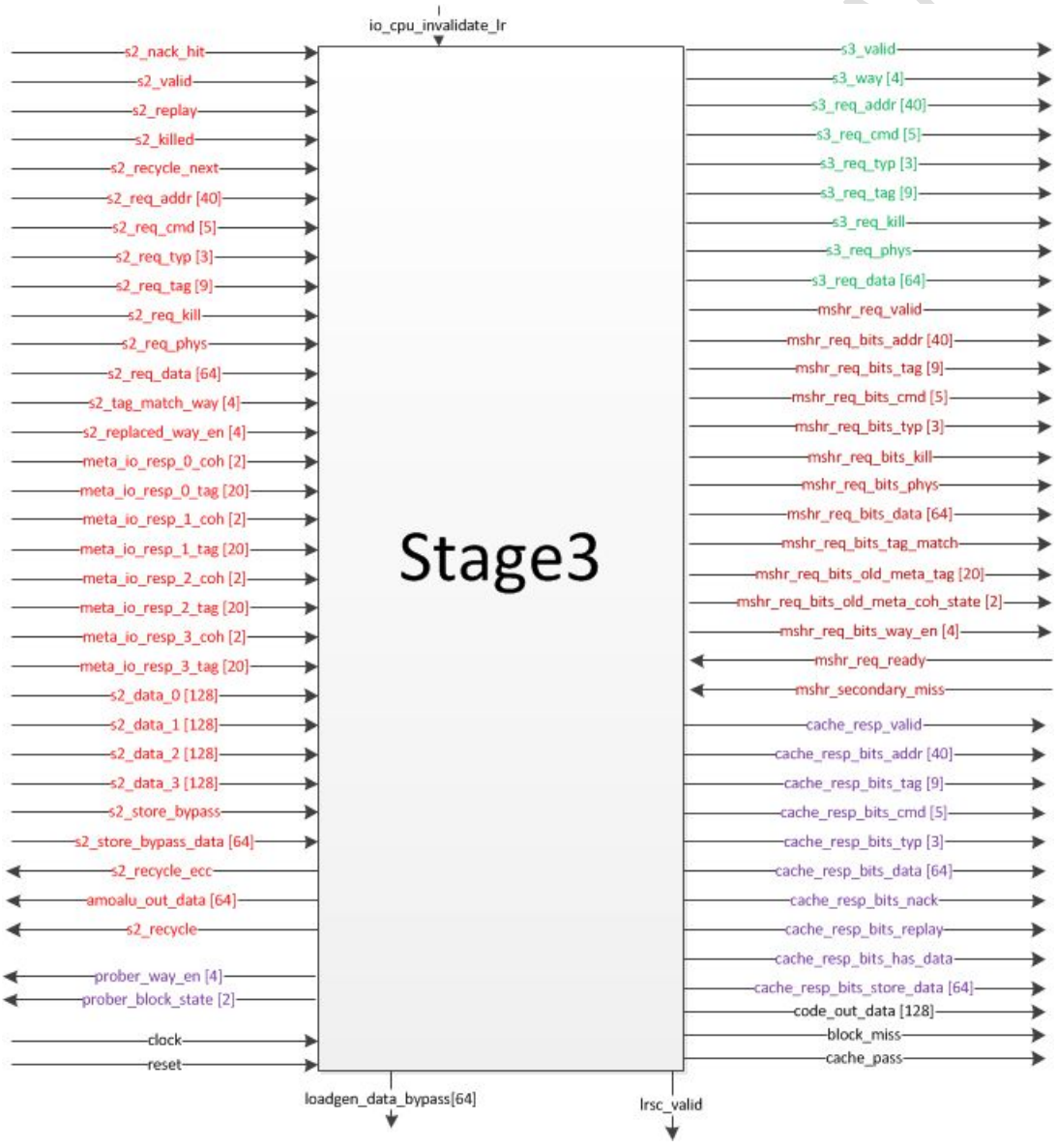

图 3.3.1-6 Stage3 的模块对外就接口信号

本级流水完成下面 5 个功能：

判断是否命中：若当前请求的数据在 Cache 体中，并且此次操作不会导致一致性状态的迁移，则为命中（cache hit），否则为失效（cache miss）。

失效请求 MSHRFile：若 Cache miss，则将失效请求发送给 MSHRFile 模块，MSHRFile 模块保存失效请求并进行相应失效处理。

ECC 校验：将请求的数据进行 ECC 校验，若校验失败，则纠错并写入 Data Array，而此次请求被发往第一级流水线来重新访问 Cache。

读返回：当前是读 Cache 请求，若 Cache hit 并且校验正确，则将数据返回给 CPU

写操作：当前是写 Cache 请求，若 Cache hit 并且校验正确，则进行写操作，修改后的数据在第四级流水线被存入 Data Array。

3.3.1.5.2 数据通路图

第三级流水的数据通路如图 3.3.1-4 所示：

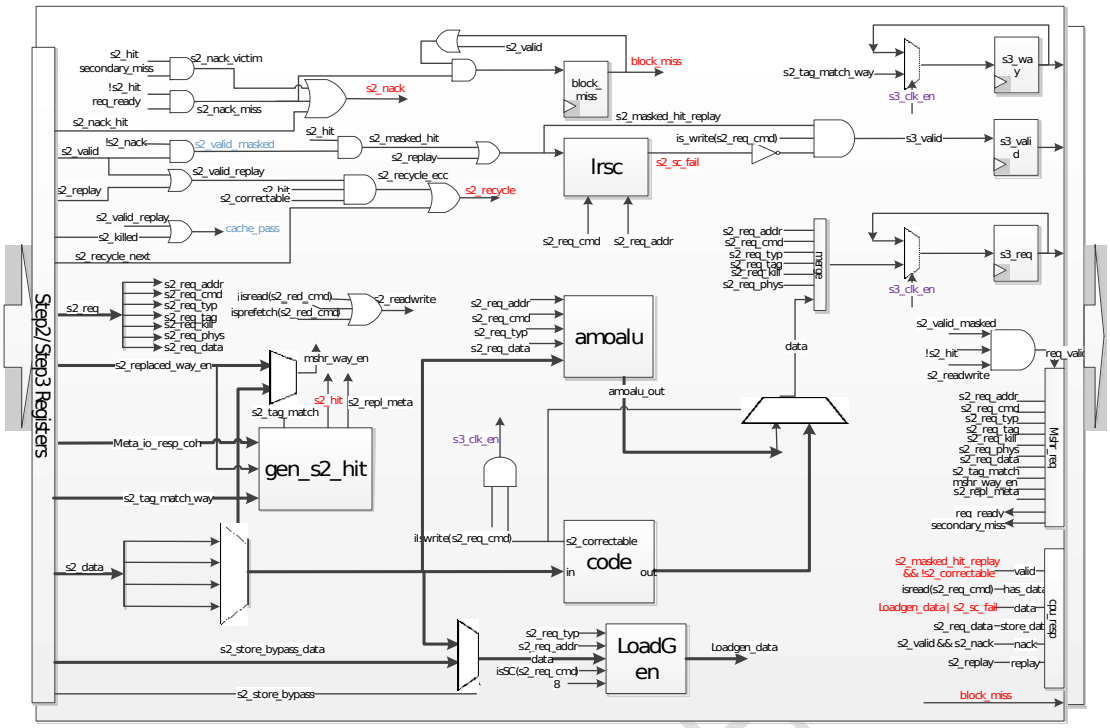

图 3.3.1-7 第三级流水的数据通路

lrsc 模块

- 支持 load-reserve 和 store-conditional 指令
- 数据通路如图 3.3.1-5：

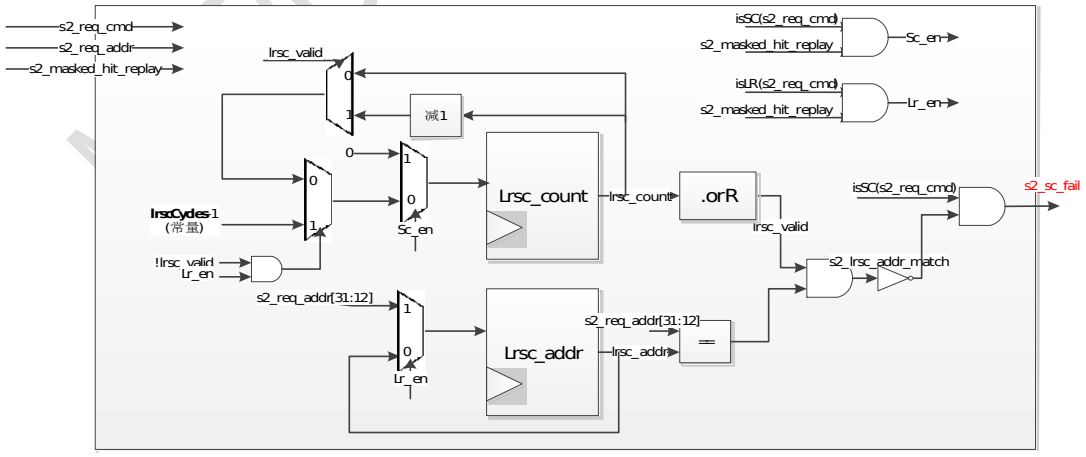

图 3.3.1-8 lrsc 模块图

gen_s2_hit 模块

■ 命中判断逻辑

■ 数据通路图 3.3.1-6：

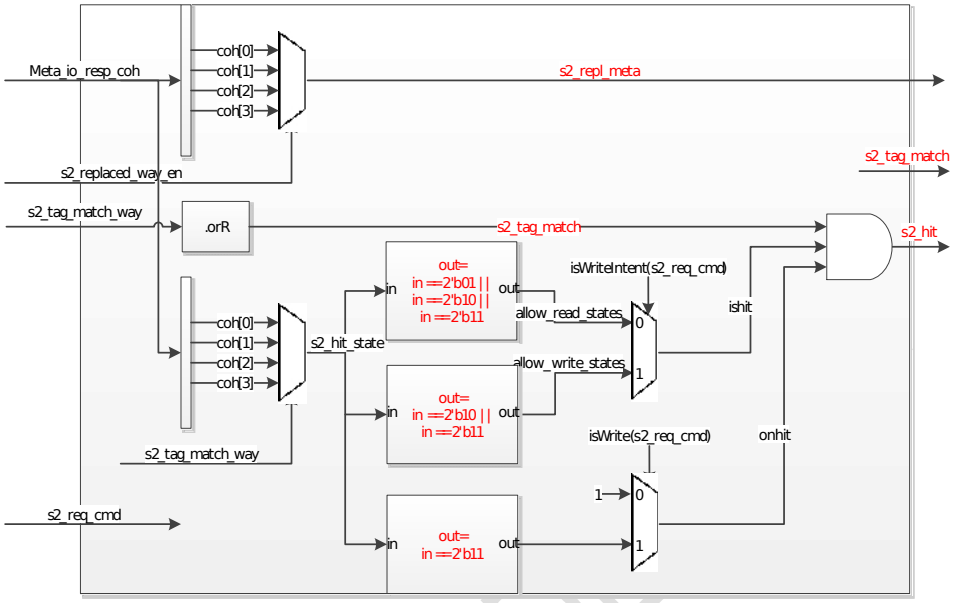

图 3.3.1-9 gen_s2_hit 模块图

3.3.1.6 流水线第三级输出寄存器（完成）

该组寄存器的功能是寄存第三级流水的输出信号，用于访问第四级流水线，信号包括：写 Data Array 有效信号、路选择信号、数据和数据地址，详见表所示 3.3.1-3。

表 3.3.1-3 DCache 流水线第一级输入寄存器

寄存器名称	宽度	功能	描述
s3_valid	1	写 Data Array 有效信号	1：有效 0：无效
s3_way	4	路选择信号	0000：无命中 0001：第 0 路

				0010：第 1 路 0100：第 2 路 1000：第 3 路
s3_req	addr	39	索引 Data Array 的地址	[5:0] block off [11:6] idx [31:12] tag
	tag	9	指令标签	
	cmd	5	操作类型	操作类型及编码见表 x-x
	typ	3	操作粒度	操作粒度及编码见表 x-x
	kill	1	请求是否被杀掉	0：没有 1：有
	phys	1	是否为实地址请求	0：不是 1：是
	data	128	写入 Data Array 的数据	

3.3.1.7 流水线第四级组合设计（完成）

3.3.1.7.1 功能描述

Stage4 的模块对外就接口信号如下图所示：

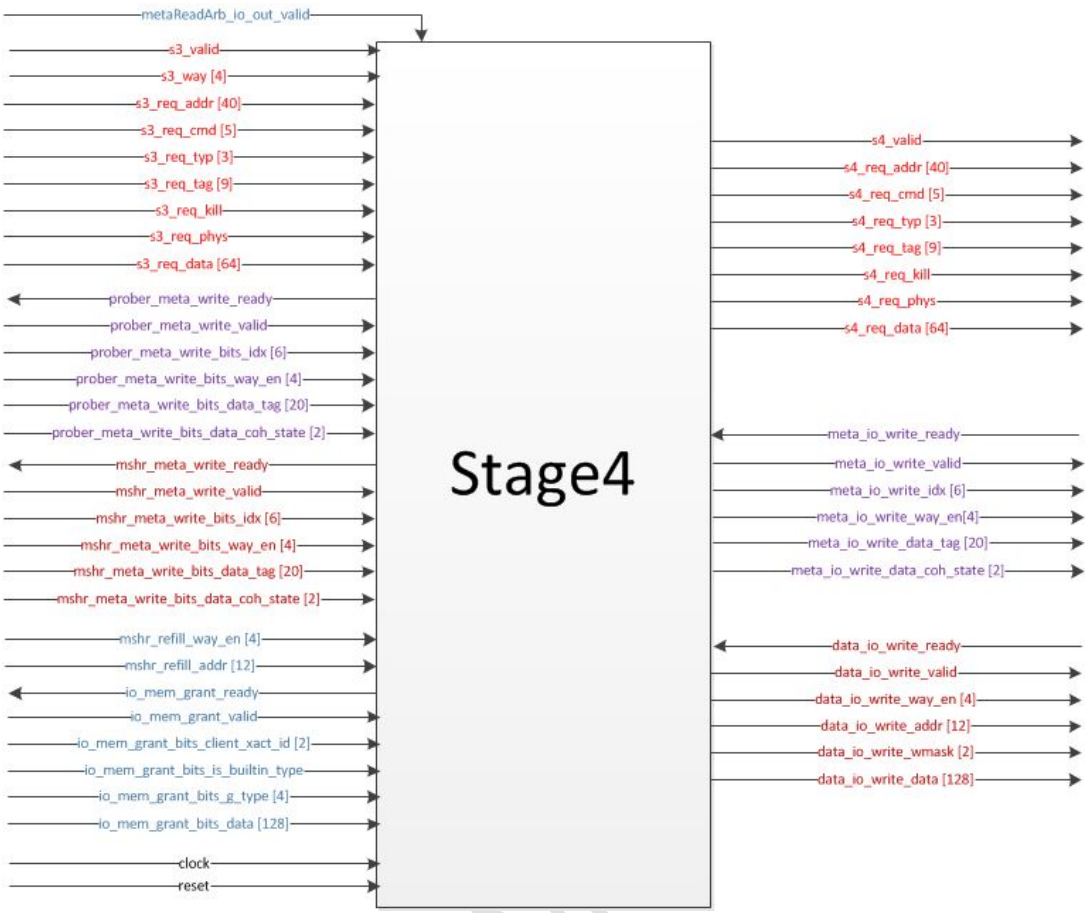

图 3.3.1-10 Stage4 的模块对外就接口信号

本级流水的功能是写 Meta Array 体和 Data Array 体。

写 Meta Array 的两种情况：

- 由本地 CPU 访问 Cache 失败，请求被保存到 MSHRFile，MSHRFile 处理失效时请求修改 Meta Array。
- 下一级存储请求 Probe 修改 Meta Array，来维护多核一致性。

写 Data Array 的两种情况：

- 本地 CPU 的写操作进入第四级流水请求将修改后的数据写入 Data Array。
- MSHRFile 请求下一级存储返回数据，MSHRFile 控制将返回的数据填入 Data Array。

3.3.1.7.2 数据通路图

第四级流水的数据通路如图 3.3.1-7 所示：

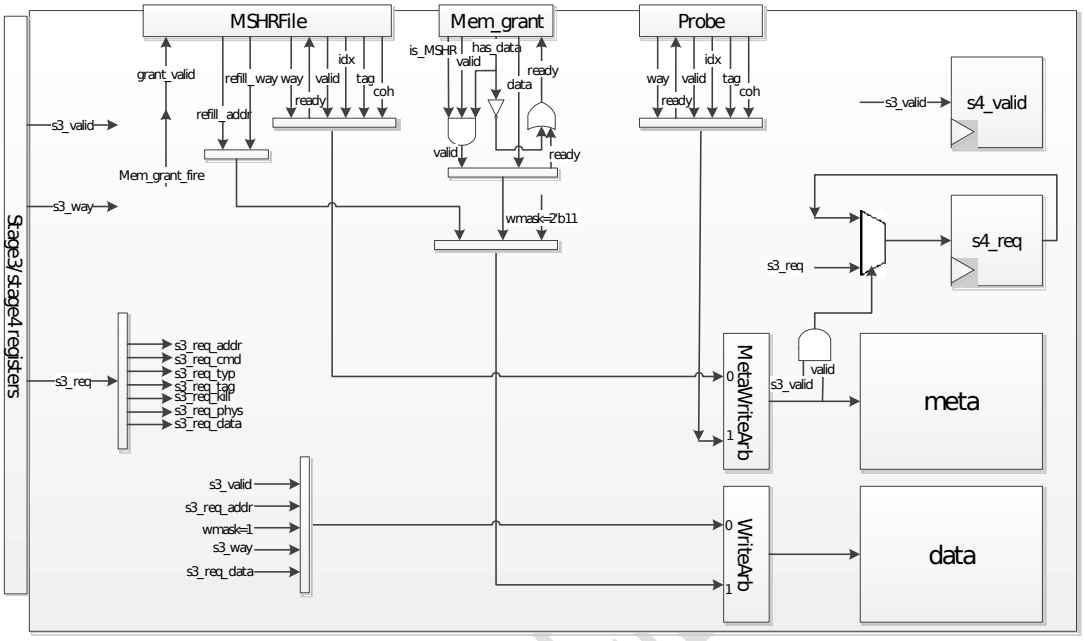

图 3.3.1-11 第四级流水的数据通路

3.3.1.8 流水线第四级输出寄存器（完成）

该组寄存器的功能是寄存第四级流水的输出信号，用于 Bypass 逻辑，信号包括：写 Data Array 有效信号、数据和数据地址，详见表 3.3.1-4 所示：

表 3.3.1-4 DCache 流水线第一级输入寄存器

寄存器名称		宽度	功能	描述
s4_valid		1	写 Data Array 有效信号	1：有效 0：无效
s4_req	addr	39	索引 Data Array 的地址	[5:0] block off [11:6] idx [31:12] tag

	tag	9	指令标签	
	cmd	5	操作类型	操作类型及编码见表 x-x
	typ	3	操作粒度	操作粒度及编码见表 x-x
	kill	1	请求是否被杀掉	0：没有 1：有
	phys	1	是否为实地址请求	0：不是 1：是
	data	128	写入 Data Array 的数据	

3.3.2 Meta Array

整个模块为 MetadataArray，该模块包含存储 cache tag 域和 state 域的物理实体（Metadata SRAM Wrapper），并实现对该实体相应读写操作。它由读写信号产生单元、Metadata SRAM Wrapper 单元与数据产生单元三部分组成。Metadata SRAM Wrapper 则由 1 个 Metadata SRAM 组成。

Metadata SRAM Wrapper 单元根据来自 Metadata Array 读写信号产生单元的读写控制信号完成对 Metadata SRAM 的读写访问，数据产生单元将从 Metadata SRAM 读出的数据进行分割，产生输出到下一模块的数据。

Metadata Array 模块整体结构框图如下图 3.3.2-1 所示。

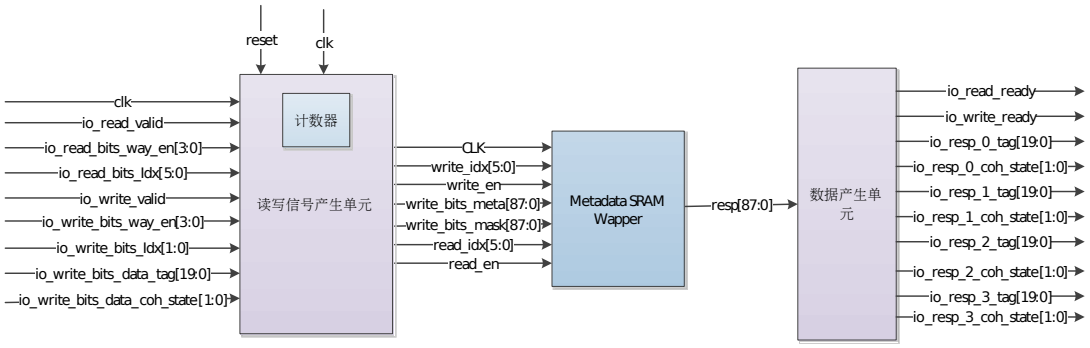

图 3.3.2-1 Metadata Array 模块整体结构框图

Metadata SRAM 的功能是保存 L1 Dcache 4 路数据所对应的 Tag 域和 state 域，Tag 域对应的是数据所在物理地址的高 20 位（即物理页号），state 域为 2 位对应 cache 块的 4 种一致性状态。L1 DCache 包含 1 个 Metadata SRAM，具体说明如下：

- 功能：保存 L1 Dcache 4 路数据所对应的 Tag 域和 state。
- 组织结构：宽度 88bit，对应每个 set 中 4 路 DCache 的 Tag 域和 state 域；深度 64，对应 L1 DCache 的 64 个 sets。

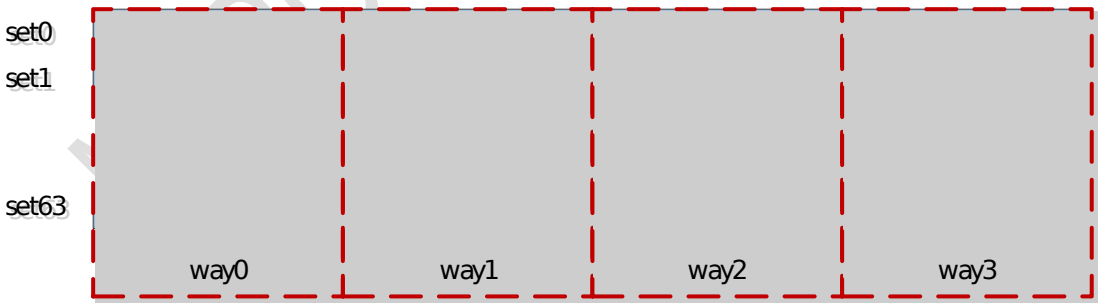

图 3.3.2-2 Metadata SRAM 组织结构

- 读写口数量：Dual Port SRAM，其中 1 个读端口 1 个写端口，可在同一周期内进行读写

下面分别介绍这三个单元。

3.3.2.1 读写信号产生单元

3.3.2.1.1 模块描述

该单元产生发送给 Metadata SRAM Wrapper 的读写信号，包括：写地址、写有效、写数据、写掩码、读有效、读地址。

3.3.2.1.2 接口信号

读写信号产生单元与外部模块的接口信号如表 3.3.2-1 所示：

表 3.3.2-1 读写信号产生单元与外部模块的接口信号

端口名称	方向	宽度	相连模块	作用	描述
clk	in	1		时钟信号	全局时钟
reset	in	1	arbiter	复位信号	
io_read_valid	in	1	arbiter	读有效	是否可读
io_read_bits_way_en	in	4	arbiter	读路使能信号	4 路中读哪一路
io_read_bits_idx	in	6	arbiter	读索引地址	要读哪一路的哪一个 row
io_write_valid	in	1	arbiter	写有效	是否可写
io_write_bits_way_en	in	4	arbiter	写路使能信号	4 路中写哪一路

io_write_bits_idx	in	6	arbiter	写索引地址	要写哪一路的哪一个 row
io_write_bits_data_tag	in	20	arbiter	写 tag 数据	写入 Metadata SRAM 的 tag 数据
io_write_bits_data_coh_state	in	2	arbiter	写 meta 数据	写入 Metadata SRAM 的 meta 数据

读写信号产生单元与内部模块的接口信号如表 3.3.2-2 所示：

表 3.3.2-2 读写信号产生单元与内部模块接口信号

端口名称	方向	宽度	相连模块	作用	描述
clk	out	1	Metadata SRAM Wrapper	DCache 时钟	全局时钟
write_idx	out	6	Metadata SRAM Wrapper	写地址	确定写入 Metadata SRAM 的哪个 set
write_en	out	1	Metadata SRAM Wrapper	写有效	Metadata SRAM 是否写有效
write_bits_meta	out	88	Metadata SRAM Wrapper	写数据	写入 Metadata SRAM 的数据（tag 和 state）
write_bits_mask	out	88	Metadata SRAM Wrapper	写掩码	写数据对应的掩码

read_idx	out	6	Metadata SRAM Wrapper	读地址	确定读出 Metadata SRAM 的哪个 set
read_en	in	1	Metadata SRAM Wrapper	读有效	Metadata SRAM 是否读有效

3.3.2.1.3 数据通路图

如果 reset，则在 64 个时钟周期内，依次将每一个 set 初始化为 0，初始化结束后再根据输入的写地址、写数据与写有效等信号写相应的 tag 域和 state 域。此过程中读操作不受影响。

读写信号产生：

Metadata SRAM Wrapper 读写信号产生如下。

- 初始化：
  - 如果 reset，则在 64 个时钟周期内，依次将每一个 set 初始化为 0，此过程中读操作不受影响。
  - 写地址为计数器的值
  - 写有效信号：有效
  - 写数据：88'b0
  - 写掩码：88 位 1
- 初始化完成后：
  - 写地址：为传入 MetadataArray 的写索引地址 io_write_bits_idx。
  - 写有效信号：传入 MetadataArray 的写有效信号有效时，其有效。
  - 写数据：将传入 MetadataArray 的 tag 与 state 拼接为 88bits，

即为写数据

- 写掩码的产生：该位路使能信号有效，写掩码为 1；该位路使能信号无效，写掩码为 0。

- 读有效信号的产生：
  - 为传入 MetadataArray 的读有效信号。
- 读地址的产生：
  - 为传入 MetadataArray 的读索引地址 io_read_bits_idx。

数据通路图：如图 3.3.2-3 所示

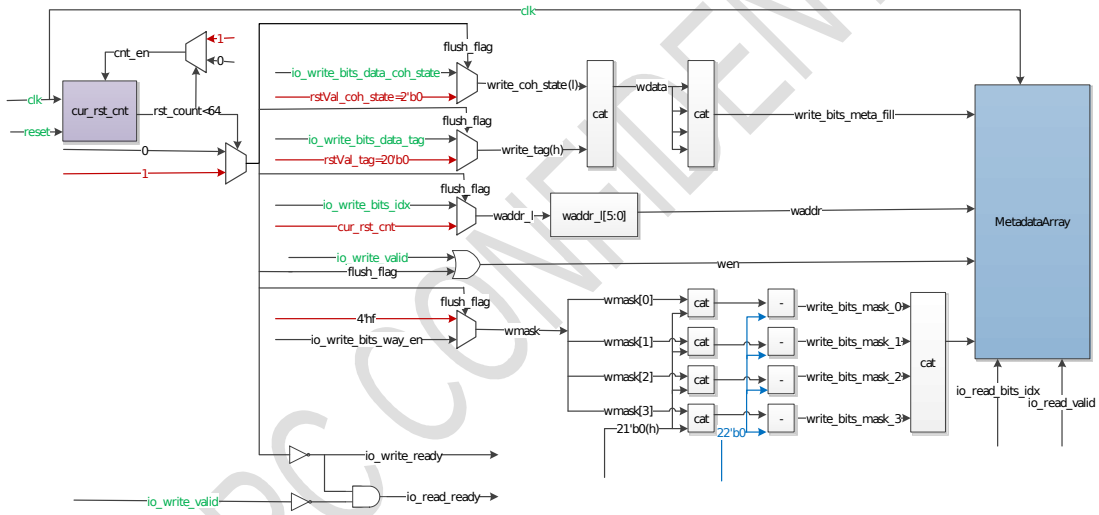

图 3.3.2-3 读写信号产生单数据通路图

3.3.2.2 Metadata SRAM Wrapper 模块

3.3.2.2.1 模块描述

Metadata SRAM Wrapper 单元根据来自 MetadataArray 读写信号产生单元的读写控制信号完成对 Metadata SRAM 的读写访问。

3.3.2.2.2 接口信号

Metadata SRAM Wrapper 的接口信号示意图如图 3.3.2-4 所示：

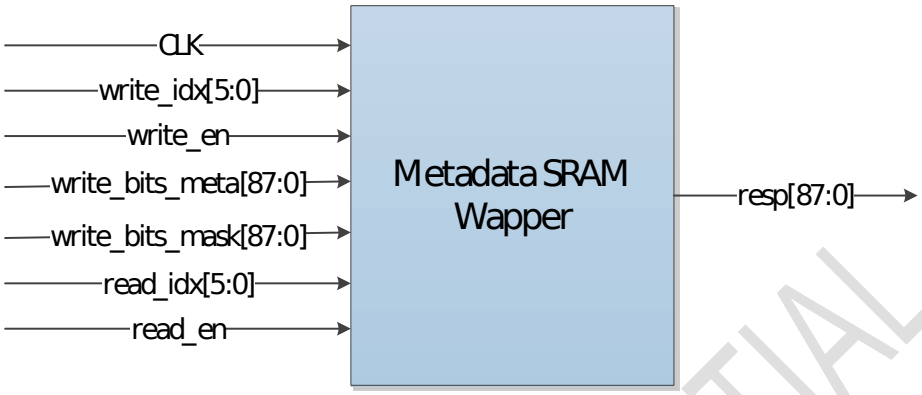

图 3.3.2-4 Metadata SRAM Wrapper 接口框图

Metadata SRAM Wrapper 的接口信号说明如下表所示：

表 3.3.2-3 Metadata SRAM Wrapper 模块与内部单元的接口信号

端口名称	方向	宽度	相连模块	作用	描述
clk	in	1	读写信号产生单元	DCache 时钟	全局时钟
write_idx	in	6	读写信号产生单元	写地址	确定写入 Metadata SRAM 的哪个 set
write_en	in	1	读写信号产生单元	写有效	Metadata SRAM 是否写有效
write_bits_meta	in	88	读写信号产生单元	写数据	写入 Metadata SRAM 的数据（tag 和 state）
write_bits_mask	in	88	读写信号产生	写掩码	写数据对应的掩码

			单元		
read_idx	in	6	读写信号产生单元	读地址	确定读出 Metadata SRAM 的哪个 set
read_en	in	1	读写信号产生单元	读有效	Metadata SRAM 是否读有效
resp	out	88	数据产生单元	读出的数据	从 Metadata SRAM 读出的数据

3.3.2.2.3 数据通路图

若 read_en 有效（读有效），则读出 Metadata SRAM 中读地址对应的数据。若 write_en 有效（写有效），则将 write_bits_meta（写数据）写入 Metadata SRAM 中写地址对应的位置。

3.3.2.3 数据产生单元

3.3.2.3.1 模块描述

该单元接收 Metadata SRAM Wrapper 读出的 tag 和 state 数据，将数据进行分割产生判断是否命中的 tag 和 state 数据。

3.3.2.3.2 接口信号

数据产生单元与外部模块的接口信号如表 3.3.2-4 所示：

表 3.3.2-4 数据产生单元与外部模块的接口信号

端口名称	方向	宽度	相连模块	作用	描述

io_read_ready	out	1	第二级流水线	读完毕信号
io_write_ready	out	1	第二级流水线	写完毕信号
io_resp_0_tag	out	20	第二级流水线	读出的第 0 路 tag 数据
io_resp_1_tag	out	20	第二级流水线	读出的第 1 路 tag 数据
io_resp_2_tag	out	20	第二级流水线	读出的第 2 路 tag 数据
io_resp_3_tag	out	20	第二级流水线	读出的第 3 路 tag 数据
io_resp_0_state	out	2	第二级流水线	读出的第 0 路 state 数据
io_resp_1_state	out	2	第二级流水线	读出的第 1 路 state 数据
io_resp_2_state	out	2	第二级流水线	读出的第 2 路 state 数据
io_resp_3_state	out	2	第二级流水线	读出的第 3 路 state 数据

数据产生单元与内部模块的接口信号如表 3.3.2-5 所示：

表 3.3.2-5 数据产生单元与内部模块接口信号

端口名称	方向	宽度	相连模块	作用	描述
resp	out	88	Metadata SRAM Wrapper	读出的数据	从 Metadata SRAM 读出的数据

3.3.2.3.3 数据通路图

如果 reset 则首先依次将每一个 set 用 0 写满，再根据写入的地址写 tag 和 state，此过程中读操作不受影响。

读数据产生单元接收 Metadata SRAM Wrapper 读出的 tag 和 state 数据，将数据进行分割产生判断是否命中的 tag 和 state 数据。其中 Meta SRAM Wrapper 返回的数据为 88 位数据即 4 路的 state 和 tag。

- 数据分割过程：
  - way0 的 state 和 tag 分为 Meta SRAM Wrapper 返回数据的[0,1]和[2:21]位；
  - way1 的 state 和 tag 分为 Meta SRAM Wrapper 返回数据的[22:23]和[24:43]位；
  - way2 的 state 和 tag 分为 Meta SRAM Wrapper 返回数据的[44:45]和[46:65]位；
  - way3 的 state 和 tag 分为 Meta SRAM Wrapper 返回数据的[66:67]和[68:87]位。

数据通路如下图所示：

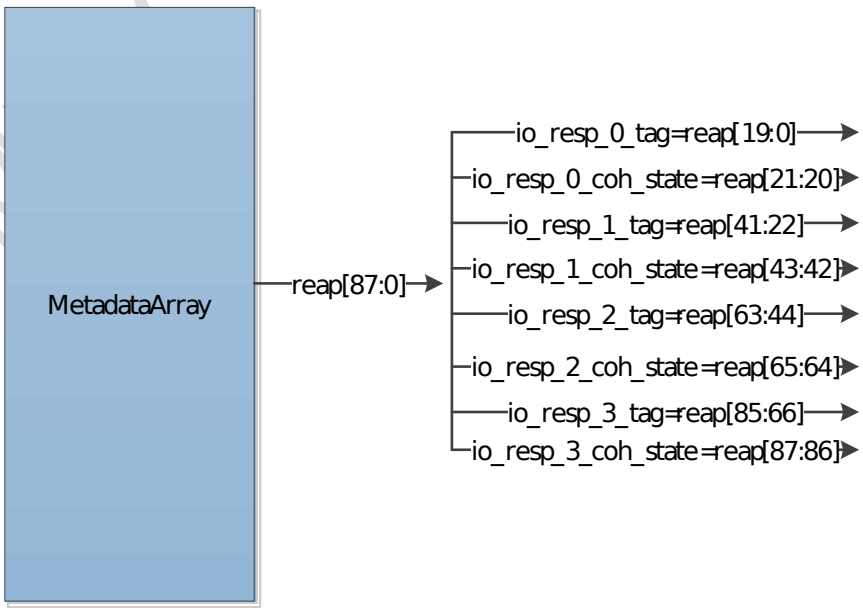

图 3.3.2-5 数据产生单元数据通路图

数据产生示意图：

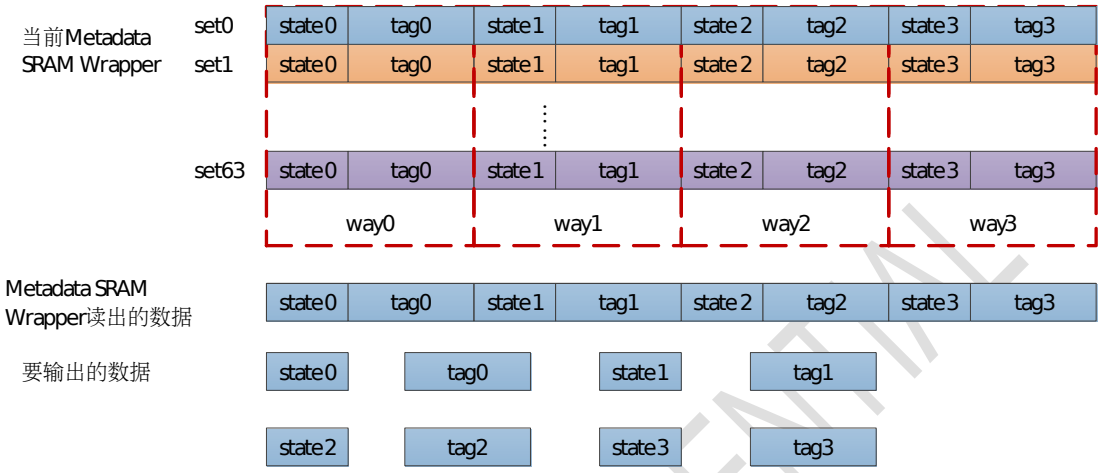

图 3.3.2-6 读操作数据产生图

3.3.3 Data Array

整个模块为 DataArray ,该模块包含存储 cache 数据主体的物理实体(Data SRAM Wrapper) , 其作用是对 Data SRAM Wrapper 进行相应的读写操作。该模块由读写信号产生单元、Data SRAM Wrapper 模块与数据产生单元三部分组成。Data SRAM Wrapper 则由 4 个 Data SRAM 组成 ,每个 Data SRAM 存储一路数据。

Data SRAM Wrapper 单元根据来自 DataArray 读写信号产生单元的读写控制信号完成对 Data SRAM 的读写访问 ,数据产生单元将从 Data SRAM 读出的数据进行重新组合，产生输出到下一模块的数据。

DataArray 模块整体结构框图如下图所示。

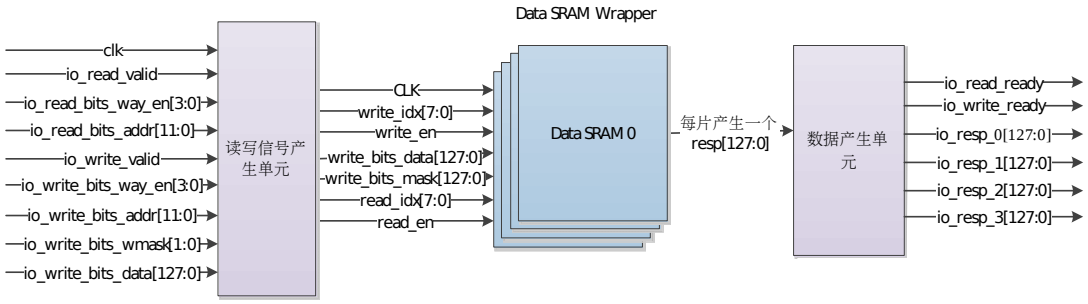

图 3.3.3-1 DataArray 模块整体结构框图

Data SRAM 的功能是保存 L1 DCache 各路的数据，预定义的 L1 Dcache 中路数为 4，所以包含 4 个 Data SRAM。每一个 Data SRAM 的具体说明如下：

- 功能：保存 L1 Dcache 各路的数据。
- 组织结构：每个 Data SRAM 有 64 个 sets，即 64 个 block，每个 block 分为 4 个 row。所以每个 Data SRAM 共有 256 个 row，每个 row 为 16byte=128bit。Data SRAM 的宽度为 128bit（即一个 row 的大小），深度为 256bit（即 row 的个数）。组织结构如下图所示

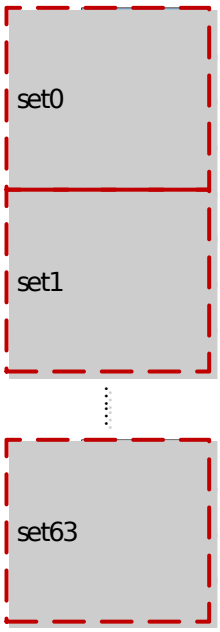

图 3.3.3-2 Data SRAM 组织结构

- 读写口数量：双端口（Dual Port），一读一写，所以一个时钟周期内可同时完成读写操作。
- 访存粒度：按 row 的大小进行访存，所以访存粒度为为 128bit。
- 读写特点：

（1）多种读写模式

按照读写模式分类：

- 写模式：写 128 位，写高 64 位，写低 64 位。
- 读模式：读 128 位，读高 64 位。

按照请求类型分类：

- cpu 读写请求：读高 64 位请求、读低 64 位请求；写高 64 位请求、写低 64 位请求。
  - √ 读高 64 位请求：由读高 64 位读模式实现。
  - √ 读低 64 位请求：由读 128 位读模式实现（在流水线会有截取

低 64 位处理）。

√ 写高 64 位请求：由写高 64 位写模式实现。

√ 写低 64 位请求：由写低 64 位写模式实现。

■ memory 读写请求：读 128 位请求；写 128 位请求。

√ 读 128 位请求：由读 128 位读模式实现。

√ 写 128 位请求：由写 128 位写模式实现

（2）读写数据时会进行重新组合。重新组合方式如下所示（以读写第 0 路和第 1 路为例）

memory 读写请求：

■ memory 写 128 位请求：dataarray 采用写 128 位模式

◆ 写第 0 路：将其低 64 位写入第 0 个 Data SRAM 低 64 位，将其高 64 位写入第 1 个 Data SRAM 低 64 位。

◆ 写第 1 路：将其低 64 位写入第 0 个 Data SRAM 高 64 位，将其高 64 位写入第 1 个 Data SRAM 高 64 位。

■ memory 读 128 位请求：dataarray 采用读 128 位模式

◆ 读第 0 路：将第 0 个 Data SRAM 低 64 位（作为低 64 位）与第 1 个 Data SRAM 低 64 位（作为高 64 位）拼接后的数据读出。

◆ 读第 1 路：将第 0 个 Data SRAM 高 64 位（作为低 64 位）与第 1 个 Data SRAM 高 64 位（作为高 64 位）拼接后的数据读出。

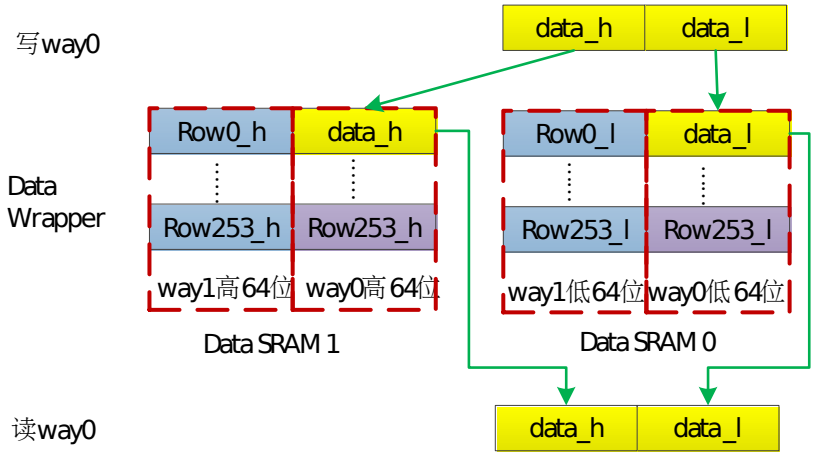

图 3.3.3-3 memory 读写 way0

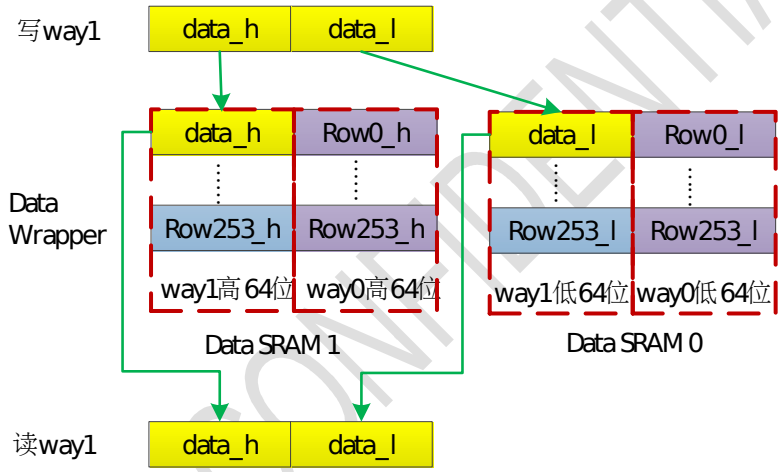

图 3.3.3-4 memory 读写 way1

cpu 读写请求

■ cpu 写高 64 位请求：dataarray 采用写高 64 位模式

通过读高 64 位读模式读出 128 位数据，截取其低 64 位，并与要写入 cache 的数据进入 AMOALU 模块，进行一定的操作后产生真正要写入 cache 的 64 位数据，将该数据复制两份拼接为 128 数据，写入 cache。

◆ 写第 0 路 :忽略其低 64 位 ,将其高 64 位写入第 1 个 Data SRAM 低 64 位。

◆ 写第 1 路 :忽略其低 64 位 ,将其高 64 位写入第 1 个Data SRAM 高 64 位。

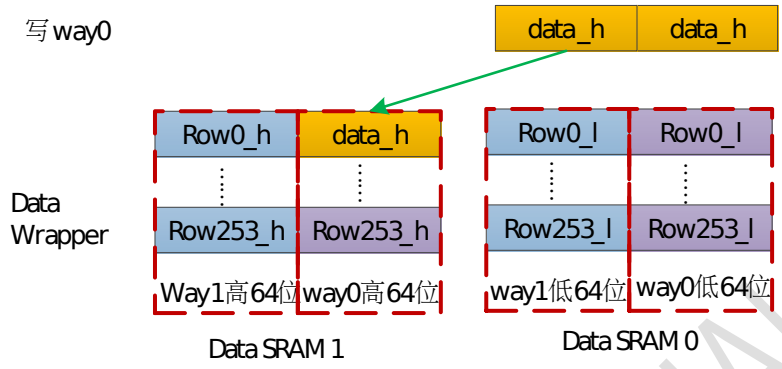

图 3.3.3-5 cpu 写 way0 高 64 位

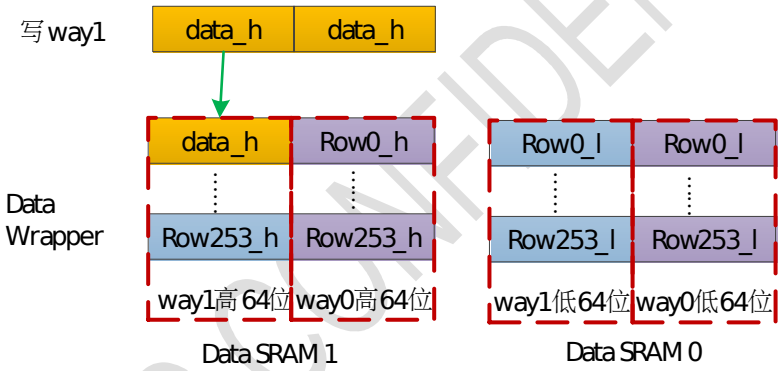

图 3.3.3-6 cpu 写 way1 高 64 位

■ cpu 读高 64 位请求：dataarray 采用读高 64 位模式：

◆ 读第 0 路：将第 1 个 Data SRAM 低 64 位数据拼接成 128 位后的数据读出。

◆ 读第 1 路：将第 1 个 Data SRAM 高 64 位数据拼接成 128 位后的数据读出。

将读出的 128 位数据截取低 64 位，返回给 cpu。

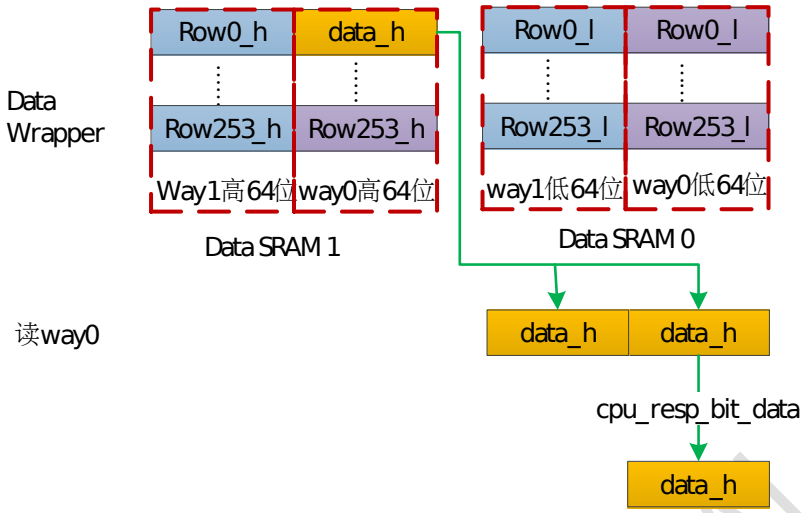

图 3.3.3-7 cpu 读 way0 高 64 位

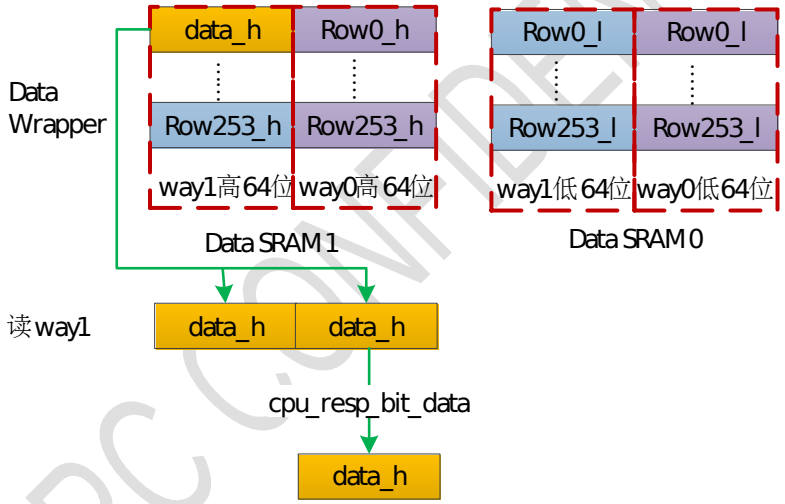

图 3.3.3-8 cpu 读 way1 高 64 位

■ cpu 写低 64 位请求：dataarray 采用写低 64 位模式

通过读 128 位读模式读出 128 位数据，截取其低 64 位，并与要写入 cache 的数据进入 AMOALU 模块，进行一定的操作后产生真正要写入 cache 的 64 位数据，将该数据复制两份拼接为 128 数据，写入 cache。

◆ 写第 0 路 :忽略其高 64 位 ,将其低 64 位写入第 0 个 Data SRAM

低 64 位。

◆ 写第 1 路 :忽略其高 64 位 ,将其低 64 位写入第 0 个 Data SRAM 高 64 位。

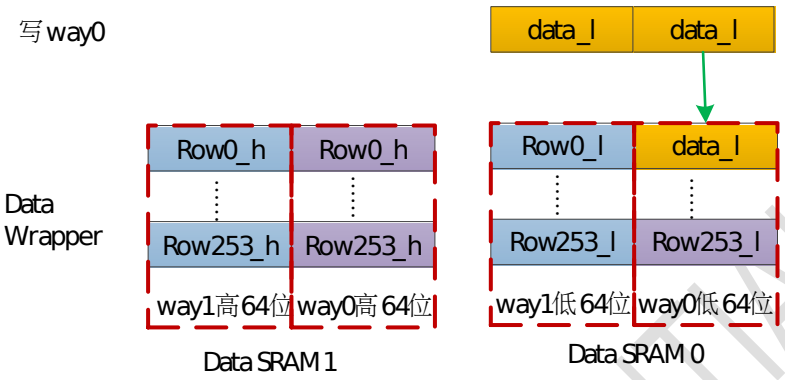

图 3.3.3-9 cpu 写 way0 低 64 位

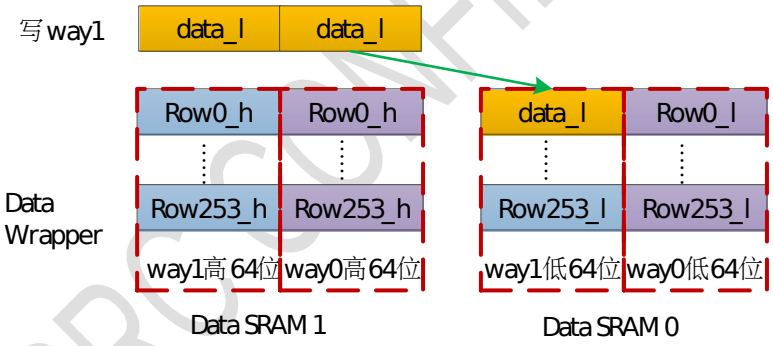

图 3.3.3-10 cpu 写 way1 低 64 位

■ cpu 读低 64 位请求：dataarray 采用读 128 位模式：

◆ 读第 0 路：将第 0 个 Data SRAM 低 64 位（作为低 64 位）与第 1 个 Data SRAM 低 64 位（作为高 64 位）拼接后的数据读出。

◆ 读第 1 路：将第 0 个 Data SRAM 高 64 位（作为低 64 位）与第 1 个 Data SRAM 高 64 位（作为高 64 位）拼接后的数据读

出。

将读出的 128 位数据截取低 64 位，返回给 cpu。

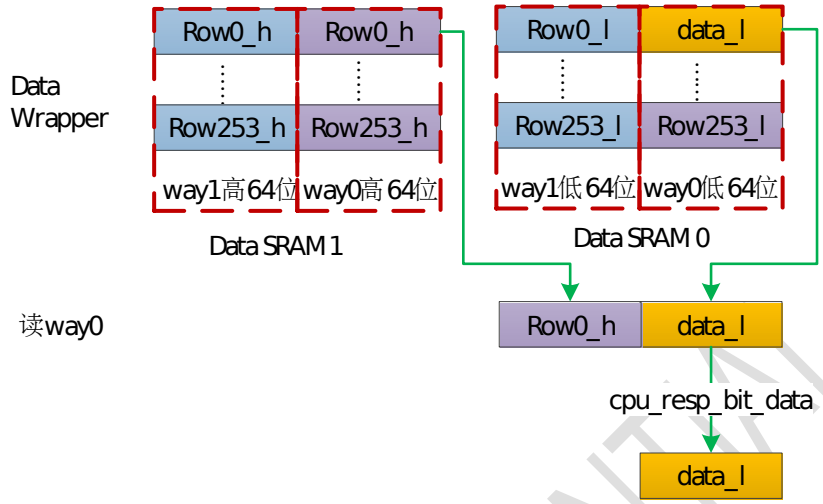

图 3.3.3-11 Cpu 读 way0 低 64 位

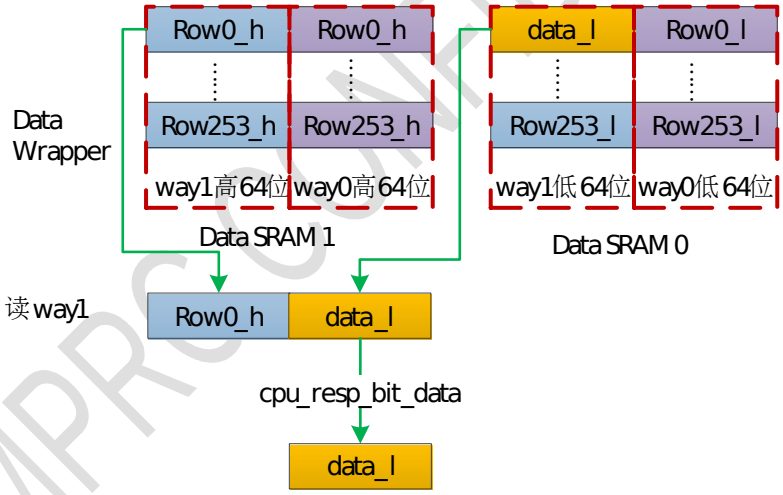

图 3.3.3-12 Cpu 读 way1 低 64 位

综上所述 ,可以将 Data SRAM0 和 Data SRAM1 的低 64 位分别看做 way0 的低 64 位和高 64 位；将 Data SRAM0 和 Data SRAM1 的高 64 位分别看做 way1 的低 64 位和高 64 位；将 Data SRAM2 和 Data SRAM3 的低 64 位分别看做 way2 的低 64 位和高 64 位；将 Data SRAM2 和 Data SRAM3 的高

64 位分别看做 way3 的低 64 位和高 64 位。也就是说 Data SRAM0 存储 way0 和 way1 低 64 位，Data SRAM1 存储 way0 和 way1 高 64 位，Data SRAM2 存储 way2 和 way3 低 64 位，Data SRAM3 存储 way2 和 way3 高 64 位。如下图所示：

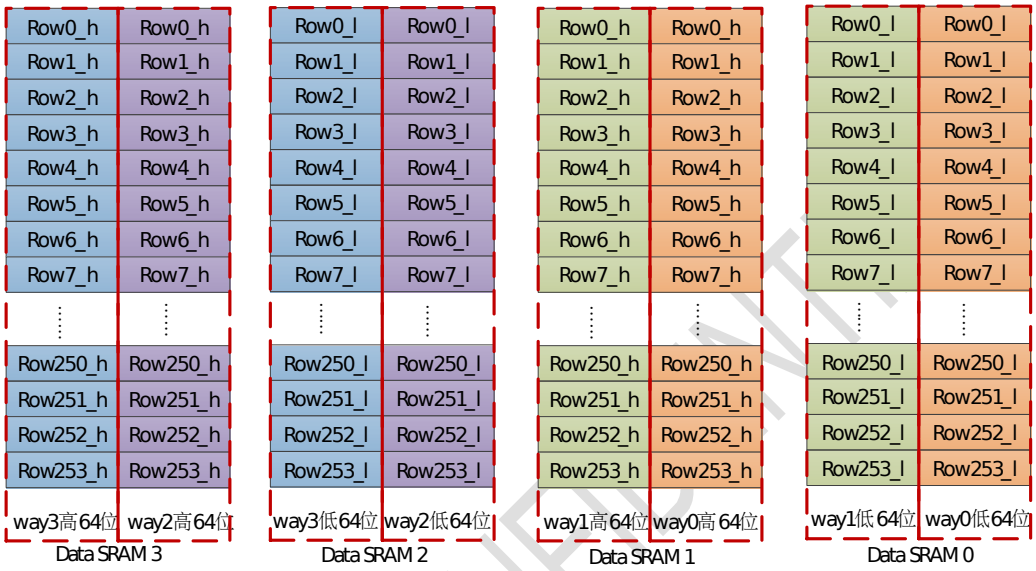

图 3.3.3-13 Data SRAM Wrapper 组织结构

这种读写模式的优缺点分析：

优点：（1）将数据备份，数据冗余提高 cache 可靠性，实例化末级 cache 时可提高性能。（2）留作 float 等 double word 类型的数据进行读写。（3）和 mem 直接进行 128Byte 交互，提高效率。

缺点：（1）每次读写都要使能两路能耗会增大。（2）cache 接口信号会比较复杂。

3.3.3.1 读写信号产生单元

3.3.3.1.1 模块描述

该单元产生发送给 Data SRAM Wrapper 的读写信号，包括：写地址、写

使能信号、写数据、写掩码、读使能信号、读地址。

3.3.3.1.2 接口信号

读写信号产生单元与外部模块的接口信号如表 3.3.3-1 所示：

表 3.3.3-1 读写信号产生单元与外部模块的接口信号

端口名称	方向	宽度	相连模块名称	作用	描述
clk	in	1		时钟	全局时钟
io_read_valid	in	1	arbiter	读有效	是否可读
io_read_bits_way_en	in	4	arbiter	片读使能信号	4 路中读哪一路
io_read_bits_addr	in	12	arbiter	读地址	要读哪一路的哪一个 row io_read_bits_addr[3] =0：读 128 位 io_read_bits_addr[3] =1： 读高 64 位
io_write_valid	in	1	arbiter	写有效	是否可写
io_write_bits_way_en	in	4	arbiter	路使能信号	4 路中写哪一路
io_write_bits_addr	in	12	arbiter	写地址	要写哪一路的哪一个 row
io_write_bits_wmask	in	2	arbiter	写高地位掩码	11：写 128 位 10：写高 64 位

				01：写低 64 位
io_write_bits_data	in	128	arbiter	写数据

读写信号产生单元与内部模块的接口信号如表 3.3.3-2 所示：

表 3.3.3-2 读写信号产生单元与内部模块的接口信号

端口名称	方向	宽度	相连模块		作用	描述
clk	out	1	Data Wrapper	SRAM	DCache 时钟	全局时钟
write_idx	out	8	Data Wrapper	SRAM	写地址	要写数据对应的 row 地址
write_en	out	1	Data Wrapper	SRAM	写使能	高电平有效 ,每个字节一个写使能位
write_bits_data	out	128	Data Wrapper	SRAM	写数据	写入 Data SRAM 的数据
write_bits_mask	out	128	Data Wrapper	SRAM	写掩码	
read_en	out	1	Data Wrapper	SRAM	读使能	
data_idx	out	8	Data Wrapper	SRAM	读地址	要读数据对应的 row 地址

3.3.3.1.3 数据通路图

Data SRAM Wrapper 读写信号产生如下（以读写 way0 和 way1 为例）。

- 读使能的产生：

- 读第 0 或 1 路时，因为要将 Data SRAM0 和 Data SRAM1 数据读出后进行拼接，所以 Data SRAM0 和 Data SRAM1 读使能位都有效。（读 128 位或者高 64 位是在数据产生单元进行操作的）

● 读地址的产生：

- 这个地址指的是读目标 row 的地址，由索引地址偏移 4 位（一个 block 中 row 的个数）产生。

● 写地址的产生：

- 这个地址指的是写目标 row 的地址，由索引地址偏移 4 位（一个 block 中 row 的个数）产生。

● 写数据的产生：

- 写入 Data SRAM0 数据由要写数据的低 64 位拼接而成。
- 写入 Data SRAM1 数据由要写数据的高 64 位拼接而成。

根据写模式的不同，Data SRAM0 和 Data SRAM1、写掩码、写使能信号的值如表 3.3.3-3 所示。根据表可看出：

● 写掩码的产生：由路使能决定。

- 若路使能信号第 0 位为 1，那么写第 0 路，数据要写到 Data SRAM0 和 Data SRAM1 低 64 位，所以 Data SRAM0 和 Data SRAM1 掩码低 64 位全为 1，否则为 0。
- 若路使能信号第 1 位为 1，那么写第 0 路，数据要写到 Data SRAM0 和 Data SRAM1 高 64 位，所以 Data SRAM0 和 Data SRAM1 掩码高 64 位全为 1，否则为 0。

● 写使能信号的产生：由路使能、写有效、写高低 64 位掩码决定。

■ 若写 way0 或 way1 的低 64 位，并且写有效，那么 Data SRAM0 的写使能信号有效，即：

wen_0=(io_write_bits_way_en[1:0] != 2'b00)&io_write_valid&io_write_bits_wmask[0]

■ 若写 way0 或 way1 的高 64 位，并且写有效，那么 Data SRAM1 的写使能信号有效，即：

wen_1=(io_write_bits_way_en[1:0] != 2'b00)&io_write_valid&io_write_bits_wmask[1]

信号说明：

io_read_bits_way_en——0001：写 way0；0010：写 way1；

io_write_bits_wmask——11：写 128 位；10：写高 64 位；01：写低 64 位

io_read_bits_addr[3] ——0：读 128 位；1：读高 64 位

表 3.3.3-3 不同读写模式下读写信号的值

信号 \ 请求分类		cpu 写请求				mem 写请求	
		写 way0 低 64 位	写 way0 高 64 位	写 way1 低64位	写 way1 高 64 位	写 way0	写 way1
输入信号	io_write_bits_way_en	4’b0001	4’b0001	4’b0010	4’b0010	4’b0001	4’b0001
	io_write_bits_wmask	2’b01	2’b10	2’b01	2’b10	2’b11	2’b11
	io_read_bits_addr[3]	1’b0	1’b1	1’b0	1’b1	1’b0	1’b0
访问 Data SRAM0 信号	wen_0	1	0	1	0	1	1
	write_bits_mask (每个数字代表 64 位)	01	10	01	10	01	10
访问 Data	wen_1	0	1	0	1	1	1

SRAM0 信号	write_bits_mask (每个数字代表 64 位)	01	10	01	10	01	10

数据通路图如下

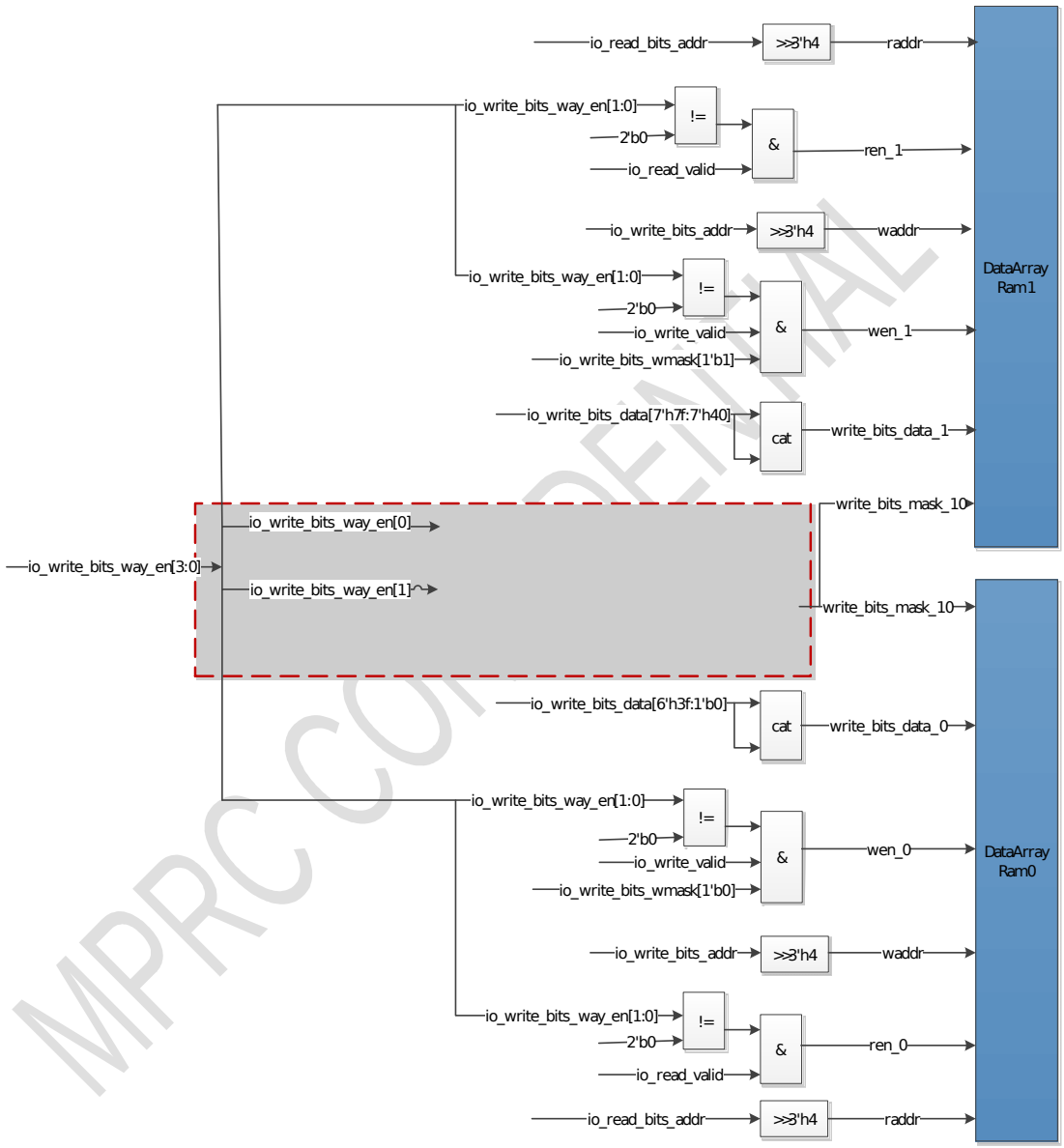

图 3.3.3-14 读写信号产生单元数据通路图

3.3.3.2 Data SRAM Wrapper

3.3.3.2.1 模块描述

Data SRAM Wrapper 的功能是根据来自 DataArray 读写信号产生单元的读写控制信号完成对 Data SRAM 的读写访问。

3.3.3.2.2 接口信号

Data SRAM Wrapper 的接口信号示意图如图 3.3.3-15 所示：

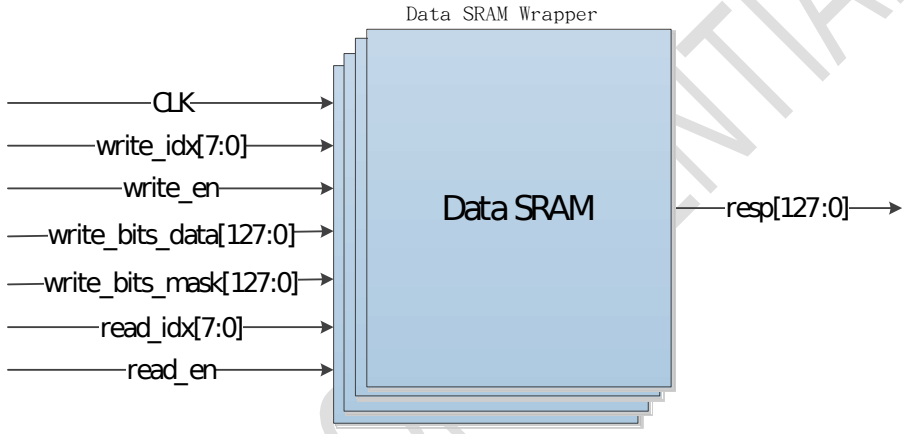

图 3.3.3-15 Data SRAM Wrapper 接口框图

Data SRAM Wrapper 的接口信号含义如表 3.3.3-4 所示：

表 3.3.3-4 Data SRAM Wrapper 与内部单元接口信号

端口名称	方向	宽度	相连模块	作用	描述
clk	in	1	读写信号产生单元	DCache 时钟	全局时钟
write_idx	in	8	读写信号产生单元	写地址	要写数据对应的 row 地址
write_en	in	1	读写信号产生单	写使能	高电平有效 ,每个字节

			元		一个写使能位
write_bits_data	in	128	读写信号产生单元	写数据	写入 Data SRAM 的数据
write_bits_mask	in	128	读写信号产生单元	写掩码	
read_en	in	1	读写信号产生单元	读使能	
data_idx	in	8	读写信号产生单元	读地址	要读数据对应的 row 地址
resp	out	128	数据产生单元	读出的数据	从 Data SRAM 中读出的数据

3.3.3.2.3 数据通路图

若 read_en 有效（读有效），则读出 Data SRAM 中读地址对应的数据。若 write_en 有效（写有效），则将 write_bits_data（写数据）写入 Data SRAM 中写地址对应的位置。Data SRAM Wrapper 数据通路如下图所示：

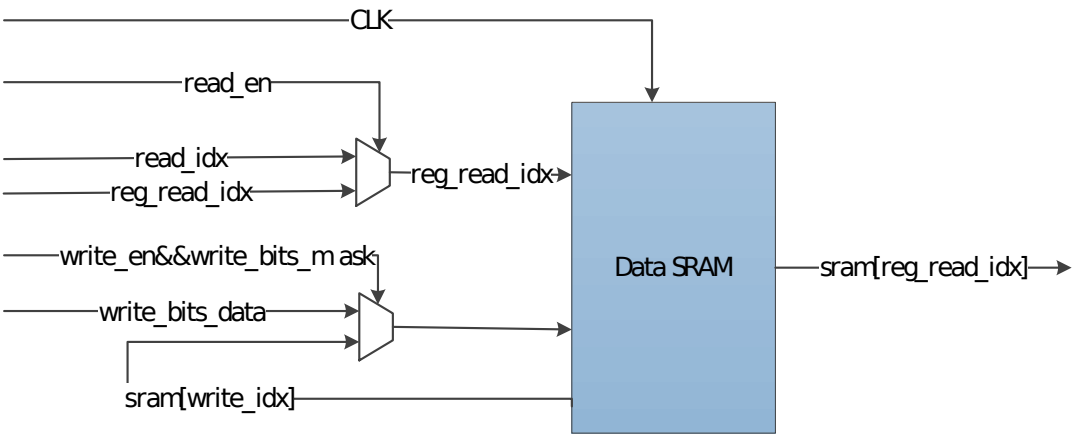

图 3.3.3-16 Data SRAM Wrapper 数据通路

3.3.3.3 数据产生单元

3.3.3.3.1 模块描述

该单元接收从 Data SRAM Wrapper 读出的数据，将数据重新组合产生 DCache 读访问命中时需要返回给下一模块的数据。

3.3.3.3.2 接口信号

数据产生单元与外部模块的接口信号如表 3.3.3-5 所示：

表 3.3.3-5 数据产生单元与外部模块的接口信号

端口名称	方向	宽度	相连模块名称	作用	描述
io_read_ready	out	1	第二级刘流水线	读完毕信号	
io_write_ready	out	1	第二级刘流水线	写完毕信号	
io_resp_0	out	128	第二级刘流水线	读出的第 0 路数据	
io_resp_1	out	128	第二级刘流水线	读出的第 1 路数据	

io_resp_2	out	128	第二级刘流水线	读出的第 2 路数据
io_resp_3	out	128	第二级刘流水线	读出的第 3 路数据

数据产生单元与内部模块的接口信号如表 3.3.3-6 所示：

表 3.3.3-6 数据产生单元与内部模块接口信号

端口名称	方向	宽度	相连模块	作用	描述
resp0	in	128	Data SRAM Wrapper	读出的数据	从 Data SRAM0 中读出的数据
resp1	in	128	Data SRAM Wrapper	读出的数据	从 Data SRAM1 中读出的数据
resp2	in	128	Data SRAM Wrapper	读出的数据	从 Data SRAM2 中读出的数据
resp3	in	128	Data SRAM Wrapper	读出的数据	从 Data SRAM3 中读出的数据

3.3.3.3.3 数据通路图

读数据产生单对从 Data SRAM Wrapper 中读出的数据进行拼接，产生要输出的数据。

数据拼合过程如下：

- 128 位读写模式：
  - 第 0 个 Data SRAM 低 64 位 (作为第 0 路低 64 位) 与第 1 个 Data

SRAM 低 64 位（作为第 0 路高 64 位）拼接后的数据即为要输出的第 0 路的数据。

- 第 0 个 Data SRAM 高 64 位（作为第 1 路低 64 位）与第 1 个 Data SRAM 高 64 位（作为第 1 路高 64 位）拼接后的数据即为要输出的第 1 路的数据。

- 读高 64 位模式：
  - 第 1 个 Data SRAM 低 64 位拼接成 128 位后的数据即为要输出的第 0 路的数据。
  - 第 1 个 Data SRAM 高 64 位拼接成 128 位后的数据即为要输出的第 1 路的数据。

数据产生示意图如下：

1） 128 位读模式，读第 0 个 row

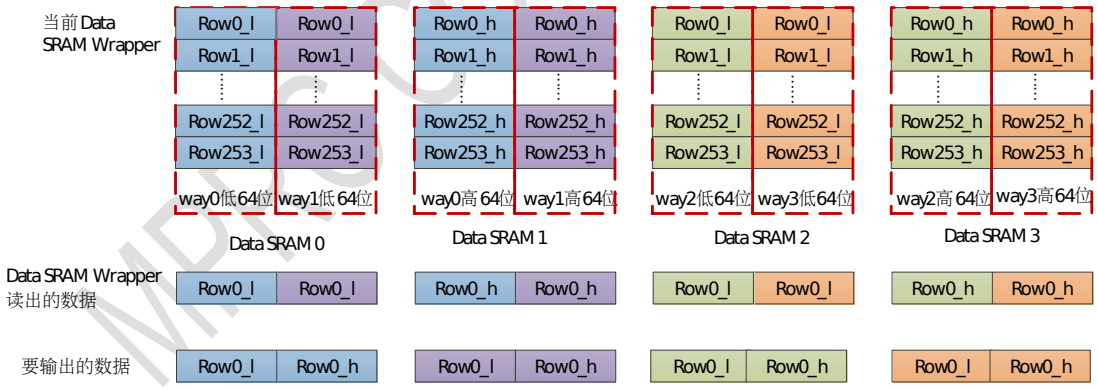

图 3.3.3-17 128 位读模式读第 0 个 row 数据产生图

2） 高 64 位写模式，读第 0 个 row

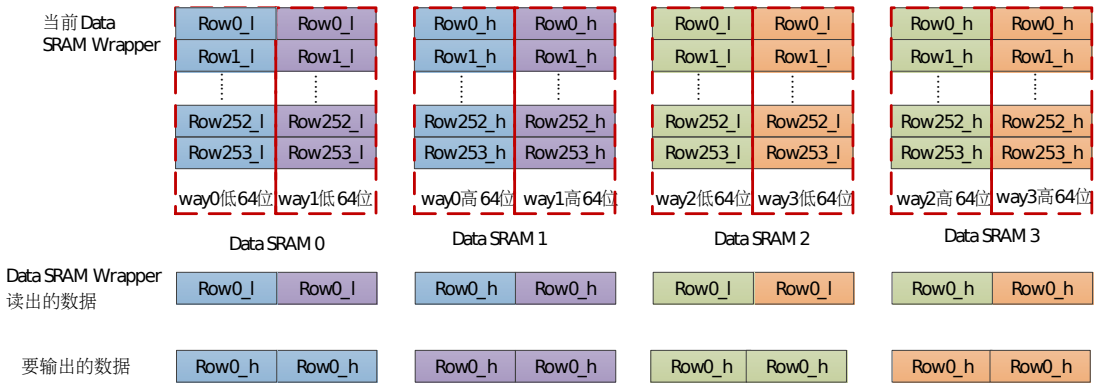

图 3.3.3-18 高 64 位读模式读第 0 个 row 数据产生图

3.3.4 DTLB

见 DTLB 详细设计文档。

3.3.5 AMOALU

3.3.5.1 模块描述

AMOALU 模块接收第二级流水线请求信号，对写 cache 体的数据与从 cache 体中读出的数据进行一定的操作，产生最终要写入 cache 体的数据。

AMOALU 有两个功能分别是处理原子存储操作指令和普通存储指令。RISC-V 提供了 fetch-and-op 风格的原子存储操作（AMO，Atomic Memory Operation），即将内存中的数据、与寄存器的数据进相应的原子操作，将结果写入到原内存中的地址，该操作执行期间不能被打断。AMOALU 模块支持 RISC-V 提供的所有的 AMO 指令，包括整数加、逻辑 AND、逻辑 OR、逻辑 XOR 和有符号、无符号整数最大值和最小值。另一方面 AMOALU 模块根据数据操作类型对写 cache 体的数据进行相应的操作。

如果指令操作类型为原子存储操作则对写 cache 体的数据和从 cache 体中

读出的数据施加一个相应的二进制原子操作 ,否则根据数据操作类型对写 cache 体的数据进行重新组合。AMOALU 模块整体结构框图如下图所示：

s2_req 为第二级流水线请求，s2_data 为从 cache 体中读出的数据，s3_req 为第三级流水线请求。

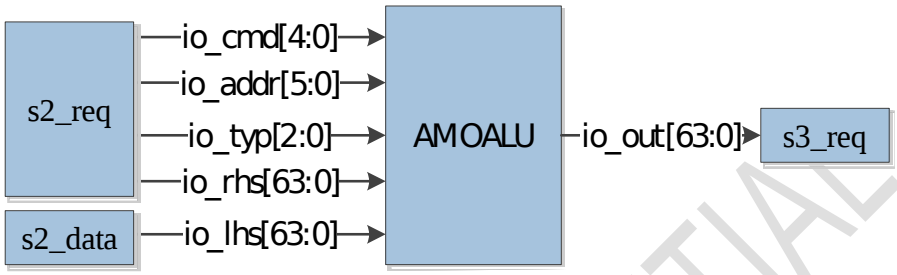

图 3.3.5-1 AMOALU 模块整体结构框图

3.3.5.2 接口信号

AMOALU 与外部模块的接口信号，如下表所示：

表 3.3.5-1 AMOALU 与外部模块的接口信号

端口名称	方向	宽度	相连模块名称	作用	描述
io_addr	in	6	第二级流水线	内存地址	0-2 位进行地址对齐操作
io_cmd	in	5	第二级流水线	命令类型	详见操作粒度说明表
io_typ	in	3	第二级流水线	粒度大小	详见指令类型说明表
io_lhs	in	64	第二级流水线	left hand side	从 cache 体中读出的数据
io_rhs	in	64	第二级流水线	right hand side	cpu 请求中携带的数据
io_out	out	64	第三级流水线	写入 dataarray 的	如果 code 模块校验不正

	数据	确，则将该值输入到 dataarray

（1）操作粒度说明

表 3.3.5-2 操作粒度定义与编码

	操作粒度大小 io_typ			
	字节	半字	字	双字
有符号	MT_B = 3'h0	MT_H = 3'h1	MT_W = 3'h2	MT_D = 3'h3
无符号	MT_BU = 3'h4	MT_HU = 3'h5	MT_WU = 3'h6	MT_Q = 3'h7

（2）指令类型说明

表 3.3.5-3 指令类型定义与编码

分类	指令 io_cmd	说明
普通读写	M_XRD = 5'h0	读
	M_XWR = 5'h1	写
预取读写	M_PFR = 5'h2	预取读
	M_PFW = 5'h3	预取写
空操作	M_NOP = 5'h5	空
原子操作	M_XA_SWAP = 5'h4	交换
	M_XLR = 5'h6	load-reserved
	M_XSC = 5'h7	store-conditional
	M_XA_ADD = 5'h8	加
	M_XA_XOR = 5'h9	异或

	M_XA_OR = 5'ha	或
	M_XA_AND = 5'hb	与
	M_XA_MIN = 5'hc	有符号最小
	M_XA_MAX = 5'hd	有符号最大
	M_XA_MINU = 5'he	无符号最小
	M_XA_MAXU = 5'hf	无符号最大
一致性操作	M_FLUSH = 5'h11	刷新行
	M_PRODUCE = 5'h12	清空写
	M_CLEAN = 5'h13	清空读和写

3.3.5.3 数据通路图

（1）原子存储操作

如果指令操作类型为原子存储操作则对写 cache 体的数据（io_rhs）和从 cache 体中读出的数据（io_lhs）施加一个相应的二进制操作。内存地址必须与操作数的大小对齐，如果地址没有自然对齐，将会产生一个非对齐地址异常。

（2）非原子存储操作

如果指令操作类型为非原子存储操作则根据数据操作粒度对写 cache 体的数据（io_rhs）进行重新组合。如果操作粒度为 byte 则将 io_rhs 低 8 位复制 8 份拼接成 64 位数据 ,如果操作粒度为 halfword 则将 io_rhs 低 16 位复制 4 份拼接成 64 位数据 ,如果操作粒度为 word 则将 io_rhs 低 32 位复制 2 份拼接成 64 位数据，如果操作粒度为 doubleword 则直接为 io_rhs。

最后根据操作粒度与地址对齐，将上面产生的数据（out）与从 cache 体中读出的数据（io_lhs），施加一个 mask 操作，得出真正要写入 cache 体的数据。

Mask 产生：

io_addr 低三位负责地址对齐，若 io_addr[2:0]=n，则地址按第 n 位对齐。所以地址对齐、数据操作粒度及 mask 之间的关系如下表所示

表 3.3.5-4 数据对齐与操作粒度

mask / io_addr	Byte	HalfWord	Word	DoubleWord
000	00000001	00000011	00001111	11111111
001	00000010			
010	00000100	00001100		
011	00001000			
100	00010000	00110000	11110000	
101	00100000			
110	01000000	11000000		
111	10000000			

由上表可总结出 mask 编码算法为（chisel 语法）：

```
var res = UInt(1)
for (i <- 0 until log2Up(maxSize)) {
        val upper = Mux(addr(i), res, UInt(0)) | Mux(size >= UInt(i+1),
UInt((BigInt(1) << (1 << i))-1), UInt(0))
        val lower = Mux(addr(i), UInt(0), res)
        res = Cat(upper, lower)
}
Res
```

具体数据通路图如下所示：

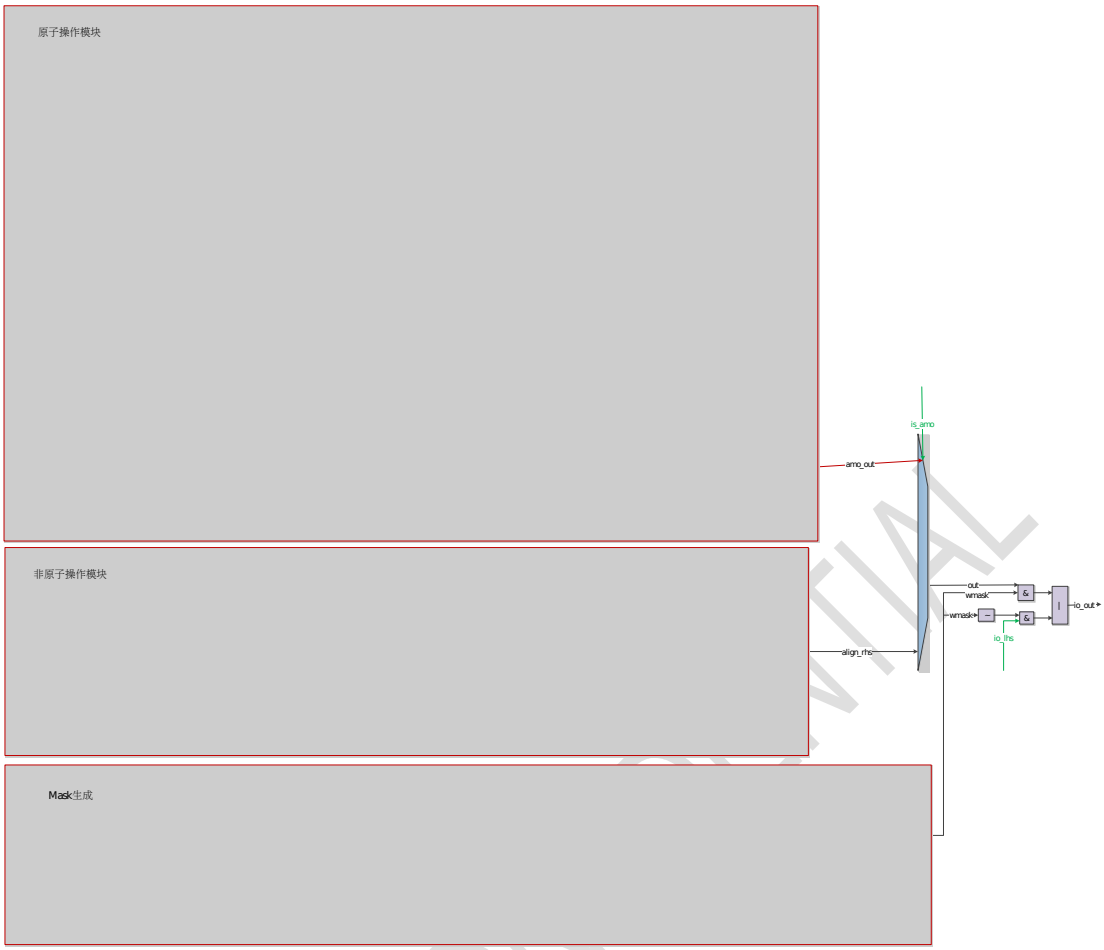

图 3.3.5-2 amoalu 数据通路图

3.3.6 替换行选择单元

3.3.6.1 模块描述

替换行选择单元的功能是当进行 Reload 操作时 ,根据 PLRU 替换算法产生所需要替换的行。

3.3.6.2 接口信号

替换行选择单元分为 cache 部分的替换和 TLB 部分的替换 , 它们与内部模

块的接口信号分别入表 3.3.6-1、表 3.3.6-2 所示：

表 3.3.6-1 cache 替换行选择单元与内部模块的接口信号

端口名称	方向	宽度	相连模块名称	作用	描述
set	in	6	内部寄存器	当前操作是哪个 set	用于选择 set
valid	in	1	内部信号线	当前模块是否开始工作	0 :替换行选择单元不工作 1 :替换行选择单元工作
hit	in	1	内部信号线	cache 是否命中	0：miss 1：hit
way_in	in	2	内部信号线	当前要访问所选 set 的哪一路	4 路组相联
reset	in	1	内部使能信号	是否重置当前模块	0：不重置 1：重置
way_out	out	2	外部模块	选择的替换行	替换行的编号

表 3.3.6-2 TLB 替换行选择单元与内部模块的接口信号

端口名称	方向	宽度	相连模块名称	作用	描述
hits	in	8	内部寄存器	当前 hit 向量	提供此次命中表

					项地址
reset	in	1	内部使能信号	是否重置当前模块	0：不重置 1：重置
state_reg	in	8	内部寄存器	伪 LRU 记录的值	8 位，记录上次做完替换的 PseudoLRU 状态
GEN_PseudoLRU_reg	out	8	外部模块	计算得到该更新 PseudoLRU 寄存器的值	更新后 PseudoLRU 状态
has_invalid_entry_in	in	1	内部使能信号	TLB 表项是否有 invalid 行	0 :没有 invallid 行 1 :有 invalid 行
has_invalid_entry_out	out	1	外部模块	TLB 表项是否有 invalid 行	0 :没有 invallid 行 1 :有 invalid 行
invalid_entry	in	8	内存寄存器	TLB 表项有 invalid 行时有效	invalid 行的入口地址

3.3.6.3 数据通路图

cache 的替换行选择单元实现了 PLRU 替换算法：

- 首先依次查看选中 Cache 组的 0~3 路的数据是否为有效的，如果存在无效的行数据，则替换该行。
- 如果选中 Cache 组内所有的 4 路的数据都是有效的，则根据 3 位的 PLRU 值产生需要被替换的行。

cache 的替换行选择单元的数据通路图如图 3.3.6-1 所示：

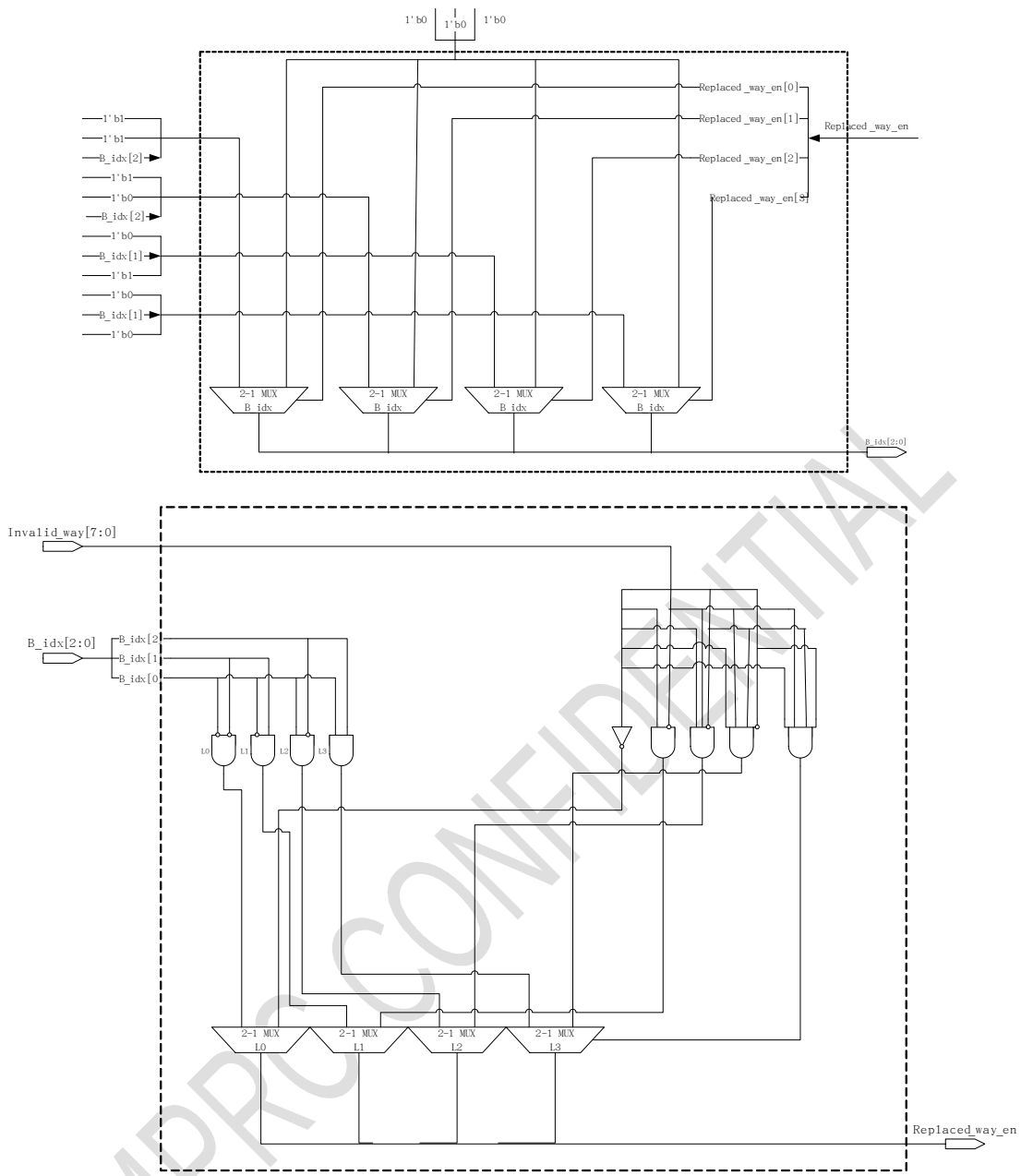

图 3.3.6-1 cache 替换行选择单元电路图

TLB 的替换行选择单元实现了 PseudoLRU 替换算法：

- 首先依次查看选中 Cache 组的 0~7 路的数据是否为有效的，如果存在无效的行数据，则替换该行。
- 如果选中 Cache 组内所有的 8 路的数据都是有效的，则根据 7 位的 PLRU 值产生需要被替换的行。

Cache 采用的是 4 路组相联方式，替换行选择单元的数据通路图与 TLB 替换行选择单元类似，只不过把 TLB 的 8 路改成了 4 路，TLB 实现起来较为复杂，cache 较为简单，所以给出了 TLB 替换行选择单元的数据通路图作为实力，cache 替换行选择单元数据通路图可参考图 3-1 的 TLB 替换行选择单元。

TLB 的替换行选择单元的数据通路图如图 3.3.6-2 所示：

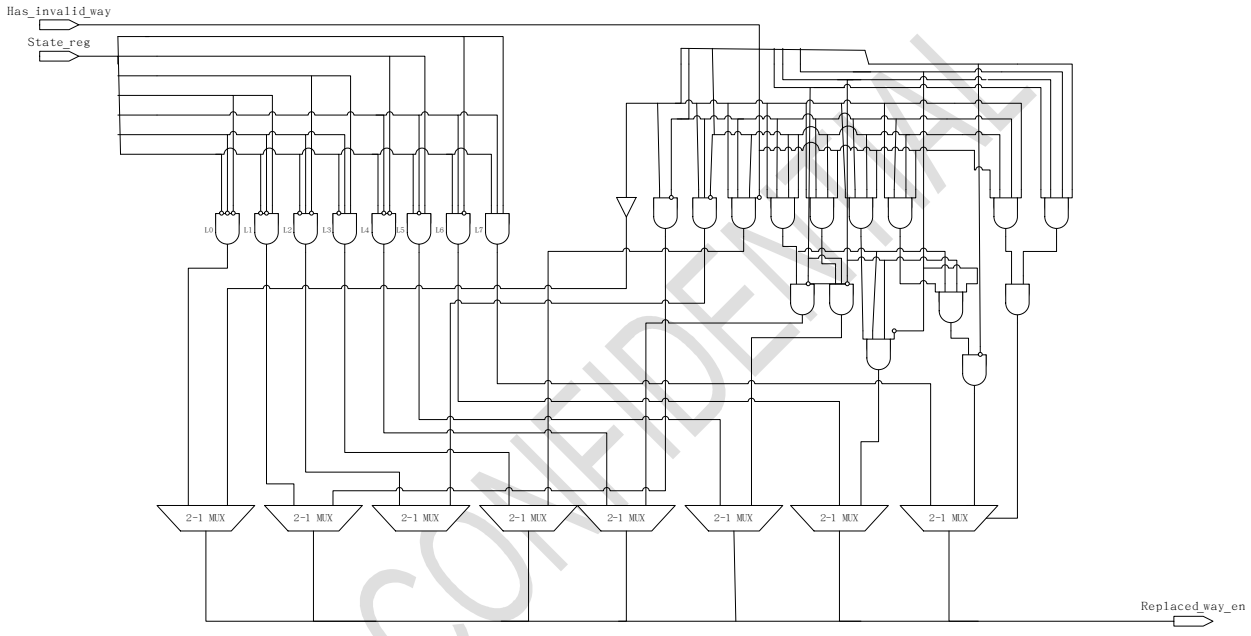

图 3.3.6-2 TLB 替换行选择单元的数据通路图

3.3.7 MSHRFile

见 MSHRFile 详细设计文档。

3.3.8 Prober

3.3.8.1 模块描述

控制多核 Cache 一致性，处理 memory 发来的 prober 请求，即处理其他

cache 块修改本 cache 块一致性信息。比如，其它核的 Cache 对某个数据块进行了写操作，它通过给本地 Probe 模块发送 invalidate 信号来清除本地 Cache 对那个数据块的缓存；其它核的 Cache 访问某个数据块发生 Cache miss，若它访问的数据块只在本地 Cache 中有缓存，它会请求本地 Probe 模块将该数据块写回 memory，memory 再将数据块发送给请求的那个处理器核。

prober 模块整体结构框图如图 3.3.8-1 所示：

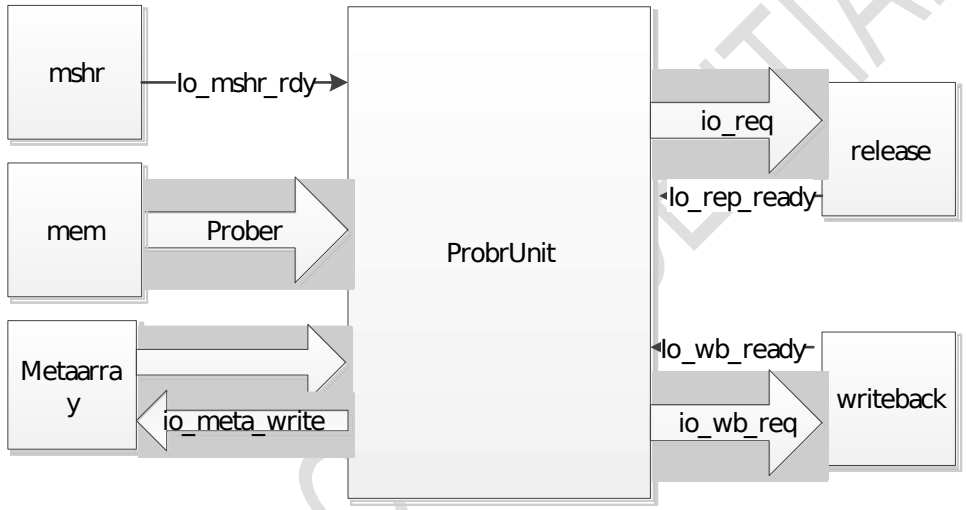

图 3.3.8-1 prober 模块整体结构框图

3.3.8.2 接口信号

表 3.3.8-1 prober 与外部模块的接口信号

端口名称	方向	宽度	相连模块	作用	描述
clk	in	1		状态机时钟	全局时钟
reset	in	1		状态机复位	1:复位

					0:正常工作
io_req_valid	in	1	mem	请求有效信号，表示可以发送请求	
io_req_bits_addr_block	in	26	mem	请求地址块的地址	tag+index
io_req_bits_p_type	in	2	mem		
io_way_en	in	4	s2_tag_match_way	路使能信号	
io_block_state	in	2	s2_hit_state	块的状态	
io_rep_ready	in	1	mem release	替换信号是否准备好	
io_meta_read_ready	in	1	meta	meta 可读信号	
io_meta_write_ready	in	1	meta	meta 可写信号	
io_wb_req_ready	in	1	mem	写回请求准备好，说明要写回	
io_mshr_rdy	in	1	mshr	mshr 准备好信号	若 mshr 处理的 set 和 prober 请求的额 set 是同一个并且 mshr 还未将改 set flush 则该信号为 0
io_req_ready	out	1	mem	请求信号准备好，可进行处理	
io_rep_valid	out	1	mem release	替换有效	可进行替换

io_rep_bits_addr_beat	out	2	mem release	要替换的 beat 地址	
io_rep_bits_addr_block	out	26	mem release	要替换的 block 地址	
io_rep_bits_client_xact_id	out	2	mem release	响应的 client 的 id 号	
io_rep_bits_voluntary	out	1	mem release	release 接收的操作类型	
io_rep_bits_r_type	out	3	mem release	Release 类型	
io_rep_bits_data	out	128	mem release	要替换的数据	
io_meta_read_valid	out	1	meta	读 meta 有效	可进行读 meta 操作
io_meta_read_bits_idx	out	6	meta	读 meta 的索引地址	确定读哪一个 set
io_meta_read_bits_tag	out	20	meta	读 meta 的 tag	确定该 set 中的哪一路 tag
io_meta_write_valid	out	1	meta	写 meta 有效	可进行写 meta 操作
io_meta_write_bits_idx	out	6	meta	写 meta 的索引地址	确定要写 meta 的地址
io_meta_write_bits_way_en	out	4	meta	写 meta 的路使能	确定要写 meta 的哪一路
io_meta_write_bits_data_tag	out	20	meta	写 meta 的 tag 信	写入的 tag 数据

				息	
io_meta_write_bits_data_coh_state	out	2	meta	写 meta 的一致性状态信息	写入的一致性数据
io_wb_req_valid	out	1	mem	写回有效	可进行写回
io_wb_req_bits_addr_beat	out	2	mem	要写回的 beat 地址	
io_wb_req_bits_addr_block	out	26	mem	要写回的 block 地址	
io_wb_req_bits_client_xact_id	out	2	mem	client 的 id 号	
io_wb_req_bits_voluntary	out	1	mem		
io_wb_req_bits_r_type	out	3	mem	mem 接收的操作类型	
io_wb_req_bits_data	out	128	mem	要写回的数据	写回 0
io_wb_req_bits_way_en	out	4	mem	写回使能位	确定写回哪一路数据

3.3.8.3 数据通路图

3.3.8.3.1 内部状态机

首先读本地 cache，查看其是否持有某个 cache 块，若持有该 cache 块，则将其一致性状态读出。

- 若 MSHR 正在处理 prober 请求的 set ,在 prober 读 meta 之前 mshr 还未 flush miss block，读的数据是无效的，需要进入 s_meta_read

状态重新读 meta。

- 若含有这个 cache 块，并且这个 cache 块为脏（ExclusiveDirty），也就是其它核的 Cache 访问某个数据块发生 Cache miss ,若它访问的数据块只在本地 Cache 中有缓存，那么进入 wb 模块进行处理，并修改此 cache 的一致性状态。
- 若含有这个 cache 块 ,并且这个 cache 块不为脏(Invalid 、Shared 、ExclusiveClean)，直接进入 release 模块进行处理，并修改此 cache 一致性状态。
- 若不含这个 cache 块，那么直接进入 release 模块进行处理即可。

处理状态转换如图 3.3.8-2 所示：

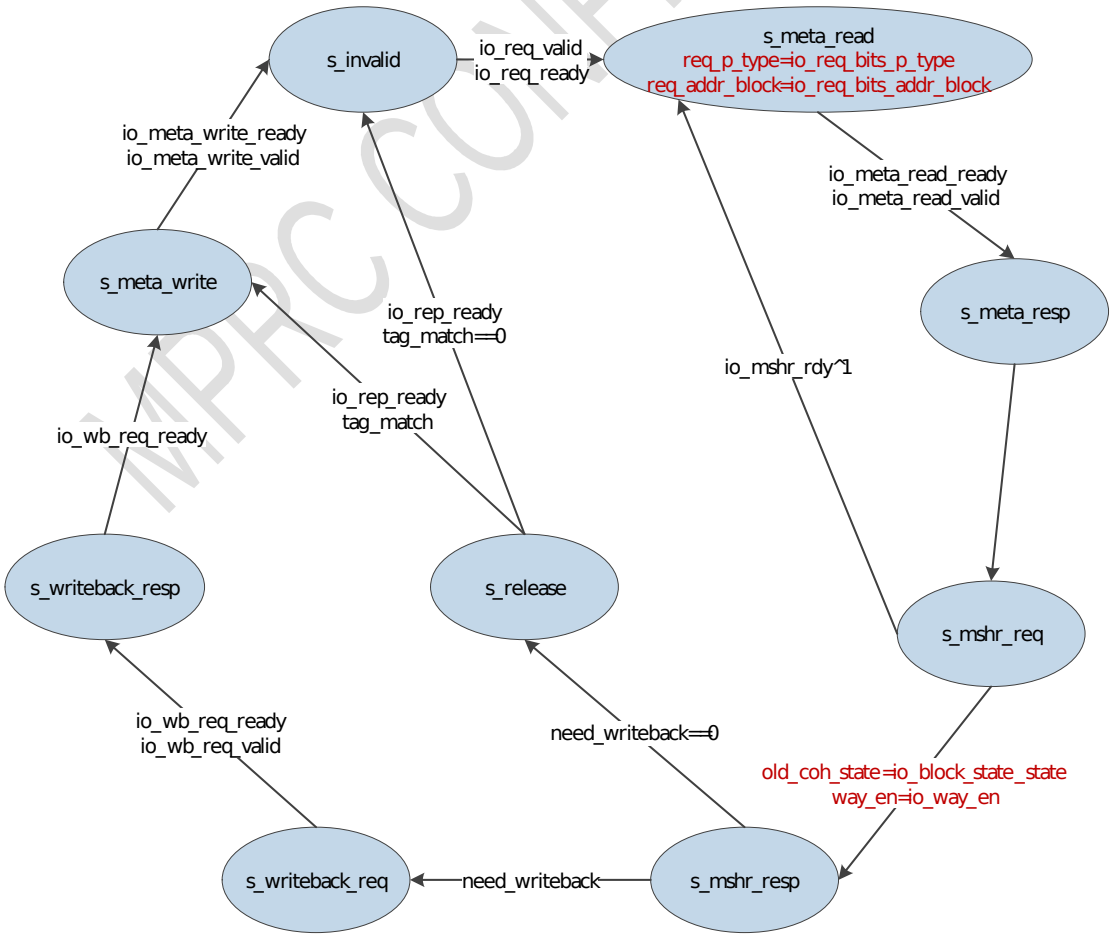

图 3.3.8-2 prober 处理状态转换图

3.3.8.3.2 状态机中状态含义说明

状态机共有 9 个状态，各状态的含义如表 3.3.8-2 所示：

表 3.3.8-2 prober 状态机中状态含义说明表

编号	状态名称	状态描述
0000	s_invalid	初始状态
0001	s_meta_read	读 meta，确认本地 cache 是否含有某个 cache 块
0010	s_meta_resp	读 meta 完毕
0011	s_mshr_req	向 mshr 发送请求，确认失效请求是否处理完毕
0100	s_mshr_resp	收到 mshr 响应
0101	s_release	请求 release 模块
0110	s_writeback_req	写回请求，将 cache 中的数据写回
0111	s_writeback_resp	写回响应
1000	s_meta_write	写 meta，修改 meta 中某个 cache 块的状态

3.3.8.3.3 状态机的状态转换表

- 状态转移一：release

  s_invalid→s_meta_read→s_meta_resp→s_mshr_req→s_mshr_resp →s_release→s_invalid

- 状态转移二：release

  s_invalid→s_meta_read→s_meta_resp→s_mshr_req→s_mshr_resp →s_release→s_meta_write→s_invalid

- 状态转移三：wb

s_invalid→s_meta_read→s_meta_resp→s_mshr_req→s_mshr_resp →s_writeback_req→s_writeback_resp→s_meta_write→s_invalid

状态转换如表 3.3.8-3 所示：

表 3.3.8-3 prober 状态机的状态转换表

起始状态	下一状态	前移条件	前移条件描述	备注
s_invalid	s_meta_read	io_req_vaild io_req_ready	prober请求准备好 并且有效	req_p_type req_addr_block
s_meta_read	s_meta_resp	io_meta_read_ready io_meta_read_valid	读meat请求准备好 并且有效	
s_meta_resp	s_mshr_rep	无条件跳转		
s_mshr_rep	s_mshr_resp	io_mshr_rdy!=0	失效状态处理完毕	old_coh_state way_en
	s_meta_read	io_mshr_rdy==0	失效状态未处理完毕	
s_mshr_resp	s_writeback_req	need_writeback	含有该cache块，并且一致性状态为ExclusiveDirty	
	s_release	need_writeback==0	不含有该cache块，或者一致性状态为Invalid 、Shared 、ExclusiveClean	
s_release	s_meta_write	io_rep_ready tag_match	prober请求准备好并且含有该cache块	
	s_invalid	io_rep_ready && tag_match==0	prober请求准备好并且不含该cache块	

s_writeback_req	s_writeback_resp	io_wb_req_ready io_wb_req_valid	写回请求准备好 并且有效
s_writeback_resp	s_meta_write	io_wb_req_ready	写回请求已准备好
s_meta_write	s_invalid	io_meta_write_ready io_meta_write_valid	写meta请求准备好 并且有效

3.3.8.3.4 状态控制信号的生成

表 3.3.8-4 prober 状态控制信号的生成

输出信号	值	含义
io_req_ready	当前状态为 s_invalid 有效	请求信号准备好，可进行处理
io_rep_valid	当前状态为 s_release 有效	替换有效
io_rep_bits_addr_beat	2'h0	要替换的 beat 地址
io_rep_bits_addr_block	为输入的 block 地址： io_req_bits_addr_block	要替换的 block 地址
io_rep_bits_client_xact_id	2'h0	
io_rep_bits_voluntary	1'h0	
io_rep_bits_r_type	见 3.4.2 章节	
io_rep_bits_data	128'h0	要替换的数据
io_meta_read_valid	当前状态为 s_meta_read 有效	读 meta 有效
io_meta_read_bits_idx	输入 block 地址的 0-5 位	读 meta 的索引地址
io_meta_read_bits_tag	输入 block 地址向右偏移 6 位	读 meta 的 tag

io_meta_write_valid	当前状态为 s_writeback_req 有效	写 meta 有效
io_meta_write_bits_idx	为输入的 block 地址：io_req_bits_addr_block	写 meta 的索引地址
io_meta_write_bits_way_en	输入的路使能信号	写 meta 的路使能
io_meta_write_bits_data_tag	输入 block 地址向右偏移 6 位	写 meta 的 tag 信息
io_meta_write_bits_data_coh_state	见表 3.3.8-5	写 meta 的一致性状态信息
io_wb_req_valid	当前状态为 s_writeback_req 有效	写回有效
io_wb_req_bits_addr_beat	2'h0	要写回的 beat 地址
io_wb_req_bits_addr_block	为输入的 block 地址：io_req_bits_addr_block	要写回的 block 地址
io_wb_req_bits_client_xact_id	2'h0	
io_wb_req_bits_voluntary	1'h0	
io_wb_req_bits_r_type	见表 3.3.8-6	
io_wb_req_bits_data	128'h0	要写回的数据
io_wb_req_bits_way_en	为输入的路使能信号	路使能

1) **写 Meta 一致性信号生成**

写 meta 的一致性信号由 prober 的操作类型 io_req_bits_p_type 决定。io_req_bits_p_type 的类型有三种：probeInvalidate、probeDowngrade、probeCopy 分别编码为 00、01、10。Cache 的一致性类型有四种：Invalid 、Shared 、ExclusiveClean、ExclusiveDirty 分别编码为 00、01、10、11。

- probeInvalidate 表示其他核的 cache 进行写操作 ,那么需要把本地该 cache 块一致性置为 Invalid。
- probeDowngrade 表示其他核的 cache 进行读操作 ,那么需要把本地该 cache 块一致性置为 Shared。
- probeCopy 表示进行 Copy 操作 ,不改变本地 cache 块的一致性。(还没有弄明白)

如表 3.3.8-5 所示：

表 3.3.8-5 Meta 一致性信号生成

Meta_coh_state 的生成	
io_req_bits_p_type (in)	Meta_coh_state(out)
00	00
01	01
10	old_coh_state

2) **writeback 与 release 模块的操作类型生成**

Release 模块的操作类型 r_type 由 meta 的一致性状态和 prober 的操作类型决定。r_type 的类型有 6 种：releaseInvalidateData 、releaseDowngradeData、releaseCopyData 、releaseInvalidateAck 、releaseDowngradeAck 、releaseCopyAck 分别编码为 000、001、010、011、100、101。

- releaseInvalidateData 、releaseDowngradeData 、releaseCopyData 表示 prober 相应的操作类型 probeInvalidate、probeDowngrade、probeCopy，并且携带了数据也就是进行了写回

操作（meta 的一致性为 ExclusiveDirty 时进行写回），也就是此时 meta 的一致性为 ExclusiveDirty。

- releaseInvalidateAck 、 releaseDowngradeAck 、 releaseCopyAck 表示 prober 相应的操作类型 probeInvalidate、probeDowngrade、probeCopy，并且不携带数据也就是没有进行写回操作（meta 的一致性为 Invalid 、Shared 或 ExclusiveClean 时不进行写回），也就是此时 meta 的一致性为 Invalid 、Shared 或 ExclusiveClean。

如表 3.3.8-6 所示：

表 3.3.8-6 r_type 的生成

r_type 的生成		
io_req_bits_p_type (in)	Meta_coh_state(in)	r_type(out)
00	11	000
01	11	001
10	11	010
00	!11	011
01	!11	100
10	!11	101

3.3.9 WriteBack

3.3.9.1 模块描述

wb 模块的主要功能有两部分：

（1）接收 mshr 的替换行信息，将替换行信息（主要是地址和数据，包含在 release 信号簇中）写回下一级存储；

（2）接收 probe 模块的查询请求，并用 release 信号簇响应该查询请求。若查询到的数据是脏数据，则将脏数据根据要写回的地址写回下一级存储。

3.3.9.2 接口信号

wb 模块接口信号表如表 3.3.9-1 所示：

表 3.3.9-1 wb 模块信号描述表

信号名称	方向	宽度	作用	描述
reset	in	1	为 1 时 wb 模块重置	置位信号
active	wb 内部	1	为 1 时代表 wb 模块处于工作活跃状态	代表 wb 模块是否处于工作状态
beat_cnt	wb 内部	2	从 wb 发向下一级存储的数据写回请求处在第几拍	是一个 2 位的寄存器
r1_req_data_fired	wb 内部	1	请求读取 meta、data 信息的第一拍	读取 meta
r2_req_dta_fired	wb 内部	1	请求读取 meta、data 信息的第二拍	根据 meta 读取 data
data_req_cnt	wb 内部	2	还需要从 cache 中取几次数据	是一个 2 位的寄存器 ,每次读取 128 位数据，每有一

				个读取 meta、data 的请求，data_req_cnt 就加 1
req	in	163	Wb 模块的写回请求信号	该信号中包含的 req.ready 为 1 时，表示有写回请求向 wb 模块发出
ready	in	1	为 1 时代表有有效的请求信号进入 wb 模块	集成在 wb.io.req 信号簇中，是从 prober.req 或 mshr.req 中发向 wb 模块的请求信号
req_valid	out	1	为 1 时代表 wb 模块响应 req 写回请求信号	wb 模块是否已经开始处理输入的 req 信号，是从 wb 模块发向请求 req 的模块的信号
data_resp	in	128	wb 模块接收到的经过校验的要写回的数据	每次接收经过 code 模块校验的 128 位数据；处理一次写回请求要从 code 接收 4 次 128 位数据
meta_read_ready	in	1	wb 模块的输出信号，与 metaReadArb.io.in(3)相连	是 metaReadArb 的输入信号，为 1 时表示请求读 meta 的请求信号有效

meta_read	out	27	wb 模块的输出信号，与 metaReadArb.io.in(3)相连	是 metaReadAr 的输入信号簇，集成了许多与读 meta 相关的信号信息
data_req_ready	out	1	wb 模块的输出信号，与 readArb.io.in(2)相连	是 readArb 的输入信号，为 1 时表示请求读 data 的请求信号有效
data_req	out	17	wb 模块的输出信号，与 readArb.io.in(2)相连	是 readArb 的输入信号，集成了许多与读 data 相关的信号信息
release	out	163	wb 模块的输出信号，与 releaseArb.io.in(0)相连	与 prober_release 共同作为 releaseArb 的输入信号，竞争 mem_release 服务
release_ready	out	1	为 1 时代表有向下一级存储写回数据的请求从 wb 发出	集成在 wb.io.release 信号簇中，是从 wb 模块发向下一级存储的信号
release_valid	in	1	为 1 时代表下一级存储接收到了写回数据请求并已经响应该请求，开始接收要写回的数据	集成在 wb.io.release 信号簇中，是从下一级存储发向 wb 模块的信号

3.3.9.3 数据通路图

图 3.3.9.1 是 wb 模块结构图：

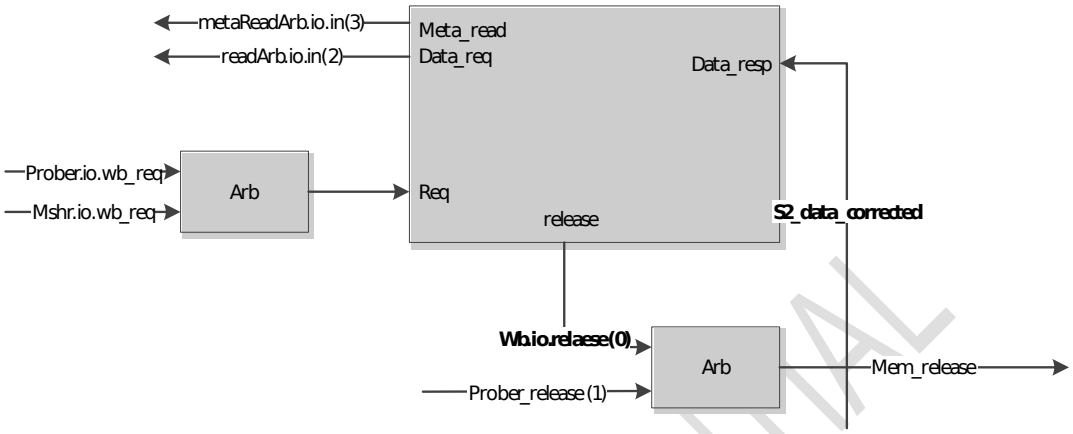

图 3.3.9-1 wb 模块结构图

图 3.3.9.2 是 wb 模块电路图：

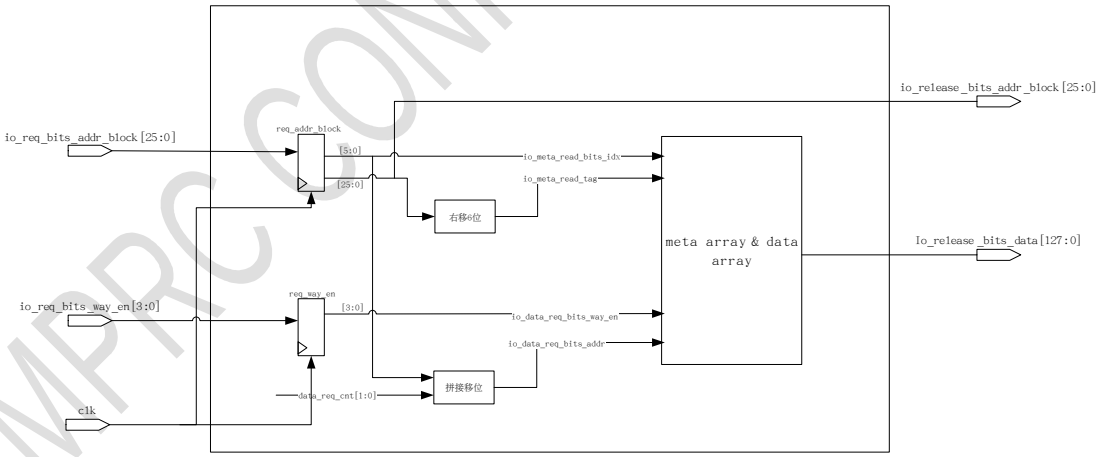

图 3.3.9-2 wb 模块电路图

具体说明：

1) wb 模块的输入信号 req 是从 mshr 和 prober 的写回请求中仲裁出的，即它在同一时刻只能为 mshr 的 req 信号或者 prober 的 req 信号服务；而进入 wb 模块的 req 信号中包含的 128 位的 io_req_bits_data 在此

处自始至终没有用到，要写到 mem 的数据是 io_release_bits_data=data_resp，其中 data_resp 是经过 code 校验的数据，即在 wb 模块，要写回的数据是根据 req 中的信息从 cache 中取回的经过校验的数据，而不是 req 中直接携带的 128 位数据；

2) mshr 替换行信息封装在 req 中，通过 req 传入 wb 模块进行处理；

3) 根据一致性协议要求，prober 处理的是下一级共享存储发出的查询命令，无论是否查询到需要的 cache 块，都会发一个 release 来响应此次查询，如果查询到的是 cache 脏数据，就会通过 wb 的处理写回下一级存储；

4) Wb 每次从 cache 取回 128 位数据，用 data_req_cnt 控制，一共取 4 次，取回数据后将数据传递给 io_release_bits_data，通过 release 向下一级存储写回数据，向下一级存储写回数据的过程中，用 beat_cnt 控制操作，一共 4 次，每次也是操作 128 位数据。

对于 Primary Cache Miss，需要在 MSHRs 寄存器组中获得一个空闲 Entry，并填写相应的 Block Address，设置 Block Valid Bit，同时向其下级存储器系统发送 Fetch 请求；当 Fetch 的 Cache Block 返回时，会并行查找 Address Stack，如果 Match，返回的数据将进入相应的 Cache，可能还会传递给某个寄存器。如果此时要被替换的 cache 块已经被改写，即处于(exclusive 状态)，mshr 就会向内存发出 wb_req，这个就是 mshr 向 wb 模块提出的数据写回请求，即由于 cache 自身的替换造成的写回请求。

与 wb 模块交互的 prober 模块的主要作用：接收外部 manager(一般情况下是共享的 L2cache) 发出的查询请求，查询是否有某一 cache 块，并把响应

信息封装在 release 中，该 cache 块带有脏数据，经过 wb 模块处理后写回下一级存储，即根据一致性协议产生的查询 cache 块及写存储器操作。

Mshr 和 prober 提出的 wb_req 会进行通过 arbiter 竞争 wb 模块的服务。

经过仲裁器和 wb 模块处理后，wb 模块通过向总线提出写回事务把新数据写回下一级存储。