文档编号：

保密级别：内部

Cache 设计文档

Cache Design

第 X 卷: MSHR File 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

RISC-V L1DCache 是一款 Non-Blocking Cache，该 Cache 是通过设计 MSHRFile 模块来实现 non-blocking 特性。MSHRFile（Miss State Holding Register File）是 RISC-V L1DCache 中保存并处理失效指令的部件。MSHRFile 通过合理的组织结构，能够支持 miss under miss，提高 cache 的整体性能。通过提供一套较为完善的失效处理操作，能够完成 miss under miss 中所有类型的失效的处理。

MSHRFile 部件通过与 write back、meta array、data array 部件等的协同工作，使得 cache 拥有 non-blocking 的特性，通过隐藏由 cache 失效带来的访存延迟提高 AMAT（Average Memory Access Time）。

1.1功能简述

1.1.1 宏观特性

MSHRFile 具有如下特性：

- 将所有 cache 操作划分为两个类别：IntentWrite 与 not IntentWrite，设定前者的权限高于后者。MSHRFile 只需要保证对同一个 cache block 写有序，它支持松散序模型。
- MSHRFile 负责处理 cache miss，还负责修改 cache coherence state
- MSHRFile 包含多个 MSHR，能同时处理多个 cache block 失效请求

- MSHR 内设有 RPQ（Replay Queue），RPQ 的深度是可配置的（默认为 8），所以一个 MSHR 能处理相同 cache block 的多次失效
- MSHR 内设 MADR（Miss Access Deal Register），MADR 只保存一条权限最高的请求。MSHR 不需要依次处理 RPQ 中所有请求，而只要处理 MADR 保存的这条请求即可。
- 支持 miss under miss

1.1.2 主要功能

访存指令请求 cache 时，若出现一致性不满足或者 cache miss 的情况，该访存指令会被发送到 MSHRFile 模块，MSHRFile 保存指令并且进行相关处理。对于第一种情况，由一致性不满足而导致访问 cache 失败，只需要 MSHRFile 修改一致性状态。对于第二种情况，由 cache miss 而导致访问 cache 失败，需要 MSHRFile 请求下一级存储返回数据并更改一致性状态。一致性不满足和 cache miss 又可被细分，最终将访问 cache 失败的情形分为 4 类（见表 1-1）。

MSHRFile 内部包含多个 MSHR ,内设有多个仲裁器来协调各 MSHR 工作。MSHRFile 接受一个请求后 ,由仲裁器 alloc_arb 将这个请求分配到一个 MSHR 上处理。多个 MSHR 同时工作时，它们可能竞争同一个输出端口，这时需要输出仲裁器来控制各 MSHR 依次进行输出。

表 1-1 访问 cache 失败情形及描述

分类	失败情况简述	失败情况详细描述

一致性不满足	cache 命中，要修改本 cache 一致性	写请求，当前状态是 exclusiveclean 态，需要迁移到 exclusivedirty 态
	cache 命中，要修改其它存储层一致性	写请求，当前状态是 shared 态，需要先请求 mem 通知其它存储层变成 invalid 态，再将本 cache 变成 exclusivedirty 态
cache miss	cache 不命中，简单失效	替换行不需要写回 mem，比如，替换行是 invalid，shared，exclusiveclean 态
	cache 不命中，复杂失效	替换行要写回 mem，比如，替换行是 exclusivedirty 态

MSHR 是组成 MSHRFile 的核心部件，完成暂存请求、处理请求和重新启动等重要工作。一个 MSHR 能处理一个 cache block 的多次失效请求。MSHRFile 模块包含多个 MSHR，所以该模块能同时处理多个 cache block 上的多次失效。MSHR 处理失效和 Cache 接受 CPU 请求的过程是并行的，即非阻塞的特性，这样能隐藏访存延迟，提高 CPU 访存效率。

从组织结构上看，MSHR 是由状态机、RPQ、MADR 和其它组合逻辑电路构成。MSHR 将所有失效命令保存到 RPQ 中，同时将权限最高的命令又保存到 MADR 中，根据 MADR 中的命令来处理失效即可。由状态机来控制失效处理，处理完毕后，将 RPQ 中所有访存命令重新返回 cache 流水线进行存储访问。

MSHR 的主要功能包括：

● 暂存请求

MSHR 收到一条失效命令后，将其保存到 RPQ 中，等待重新返回 cache 流水线。

● 处理请求

状态机控制 MSHR 逐步完成各项操作 ,最终完成整个失效处理过程。表 1-2 对失效处理过程进行描述，表 1-3 对状态机各状态进行描述。

表 1-2 MSHR 处理失效过程

分类	失败情况描述	处理过程
一致性不满足	cache 命中，要修改本 cache 一致性	根据 MADR 中 cmd 和 old_meta_state 得到迁移后的一致性状态，并写入 meta array（meta array 是存储 coherence state 和 tag 的 cache 体）
	cache 命中 ,要修改其它存储层一致性	根据 MADR 中 cmd 和 addr 请求 mem 修改其它存储层的一致性状态
cache miss	cache 不命中 ,简单失效	根据 MADR 中 cmd 和 addr 请求 mem 返回数据和 mem 的一致性状态，再根据 MADR 中 cmd 得到迁移后的一致性状态
	cache 不命中 ,复杂失效	先将替换行写回，再同简单失效一样处理

表 1-3 状态描述表

状态	含义
s_invalid	等待接受失效请求

s_wb_req	请求 writeback 回写替换行
s_wb_resp	等待 writeback 回写替换行，并且，等待下一级存储的应答
s_meta_clear	清空 meta_line，使得对应的 cache line 处于 invalid 态
s_mem_req	请求读下一级存储
s_mem_resp	将下一级存储返回的数据写入 cache
s_meta_write_req	请求写 meta_line
s_meta_write_resp	写好了 meta_line
s_drain_rpq	失效请求重回流水线

- 重新启动

  当状态机处于 s_drain_rpq 状态时，MSHR 将 RPQ 中所有命令出队列，重新发往 cache 流水线。

1.2结构框图

多个 MSHR 组成 MSHRFile。与 MSHRFile 进行交互的模块包括：meta array (存储 coherence 和 tag)、data array (存储 cache line)、write back (负责将替换行写回)、mem (下一级存储)、cache 流水线 (发送 miss access 给 MSHR)。

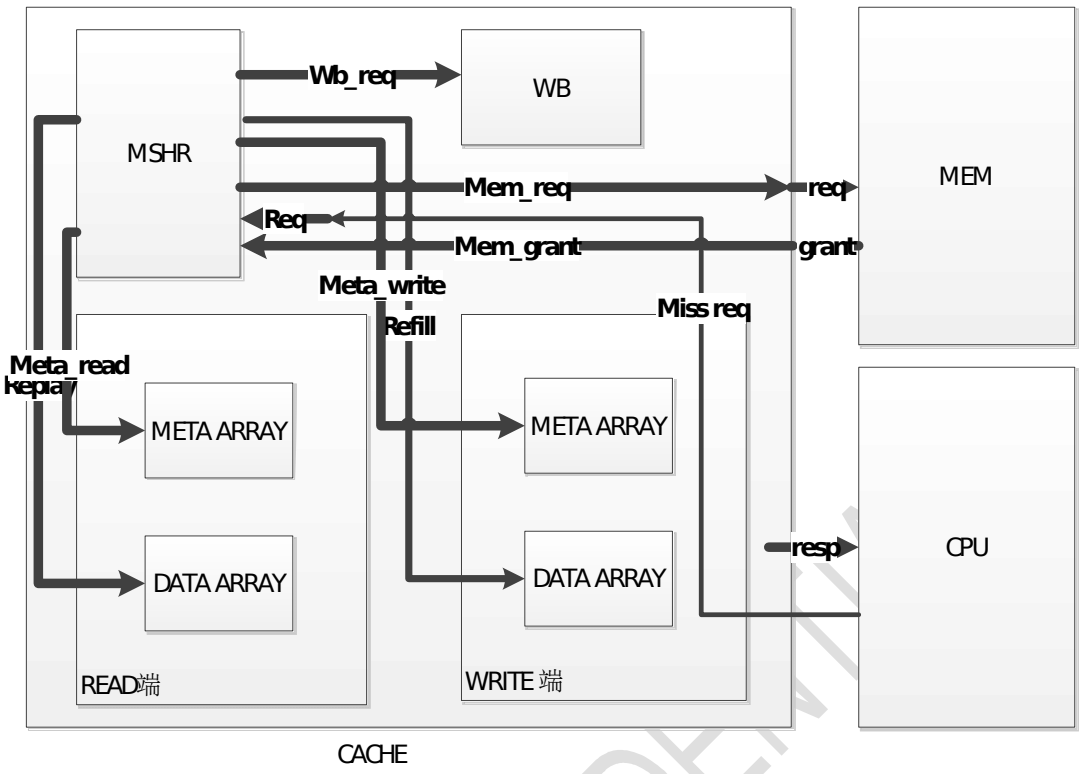

图 1-1 MSHRFile 与相关模块接口

1.3接口信号

MSHRFile 对外接口如下：

表 1-4 MSHRFile 对外接口

接口名称	接口描述	与相连
Req	接受失效命令的接口	Cache 流水线体
Wb	请求 wb 模块的接口	WriteBack 模块
Meta_read	请求 meta array 读端口的接口	Meta Array 读端口
Meta_write	请求 meta array 写端口的接口	Meta Array 写端口
Replay	请求 data array 读端口的接口	Data Array 读端口

Refill	请求 data array 写端口的接口	Data Array 写端口
Mem_req	请求 mem 的接口	MEM
Mem_grant	接受 mem 返回数据的接口	MEM

2 总体结构

2.1内部结构框图

内部结构框图如图 2-1 所示。

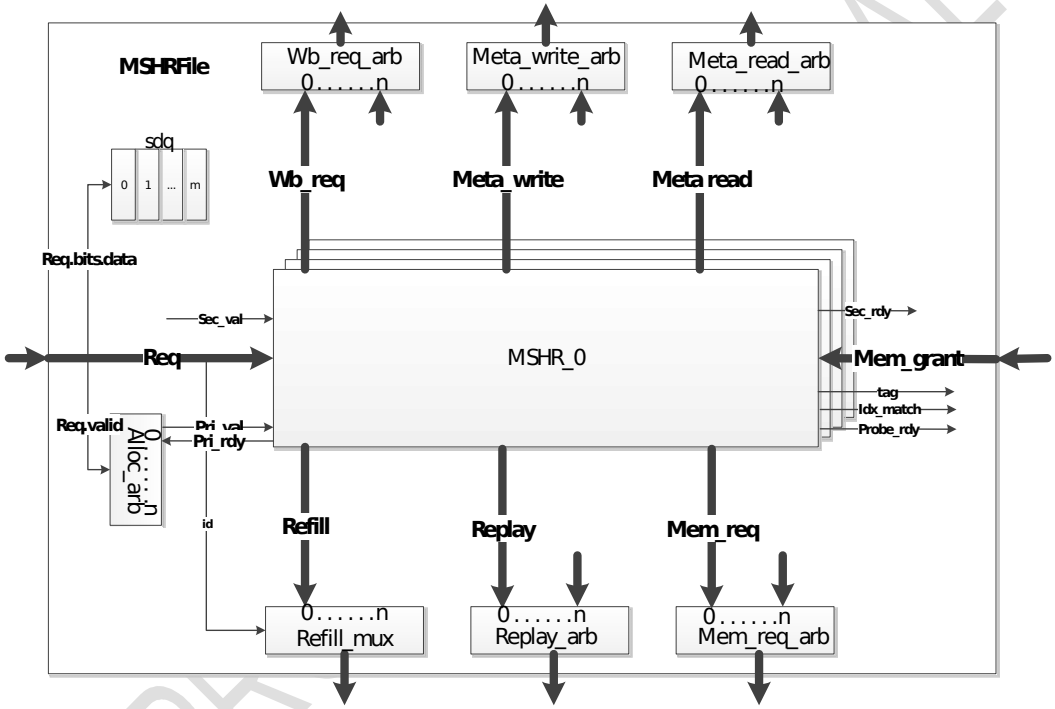

图 2-1 内部结构框图

2.2全局信号结构

整个 MSHR File 使用同样一个时钟，即系统时钟。

整个 MSHR File 使用同样一个异步 Reset 信号。

3 MSHRFile

3.1MSHRFile 结构设计

3.1.1 模块描述

内部实例化了 7 个 Arbiter，2 个 MSHR，一个 IOMSHR。这 7 个 arbiter 分别是 alloc_arb ， wb_req_arb ， mem_req_arb ， resp_arb ， meta_write_arb，meta_read_arb，replay_arb。

当一个失效请求到来时，alloc_arb 会从两个 MSHR 中分配一个。将请求信息链接到 MSHR，如果需要写回，两个 MSHR 将数据连接到 wb_req_arb，通过这个仲裁器将选择哪一个 MSHR 的请求，将数据通过 wb_req 请求信号级写回到下一级存储器。两个 MSHR 将 mem_req 信号发到 mem_req_arb 的 in 端口，经过 mem_req_arb 的选择后将其中的一个请求通过 out 端发到下一级存储器中。下一级存储器(也有可能是 IO)将数据通过 mem_grant 信号级发到两个 MSHR 以及 IOMSHR 中。两个 MSHR 同时连接着 meta_write_arb 的 in 端 ,meta_write_arb 仲裁器会选择一个 MSHR 将数据写到 cache 的 meta 中中，io_refill_way_en 信号会之初是哪一路的数据，io_refill_addr 会指定哪一个 beat。然后开始重启流水线，通过 meta_read_arb 仲裁器选择一个 MSHR 重启流水线。

该模块内部还有一个 sdq ,将存储 store 指令要存储的数据。预期对应的是一个 sdq_val，标识该对应的 sdq 一个单元是否有存储数据。0 表示未存储数

据 ,1 表示有存储数据。当 sdq_enq 有效并且找到了一个空闲的单元 ,会将 store 的数据存到 sdq 中。如果 free_sdq 有效，将此时将重启流水线 replay_arb_io_out_bits_sdq_id 对应的位置值存到 id 寄存器中。并将 sdq[id] 里的数据传给 io_replay_bits_data 信号线，输出到外部。

3.1.2 接口信号说明

与外部模块的接口信号

MSHRFile 与外部模块的接口信号，如表 3-1 所示：

表 3-1 MSHRFile 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	mem_req_arb MSHR_0 MSHR_1 IOMSHR	时钟信号	上升沿有效
reset	in	1	mem_req_arb MSHR_0 MSHR_1 IOMSHR	复位	1:有效
io_req_ready	out	1	IOMSHR	MSHRFile 是否准备好接收请求	1: 准备好 0: 未准备好
io_req_valid	in	1		req 请求信号是否有效	1:有效

io_req_bits_addr	in	40	MSHR_0 MSHR_1	IOMSHR	请求的地址位
io_req_bits_tag	in	9	MSHR_0 MSHR_1 IOMSHR		
io_req_bits_cmd	in	5	MSHR_0 MSHR_1 IOMSHR	命令类型	
io_req_bits_typ	in	3	MSHR_0 MSHR_1 IOMSHR		
io_req_bits_kill	in	1	MSHR_0 MSHR_1 IOMSHR		
io_req_bits_phys	In	1	MSHR_0 MSHR_1 IOMSHR		
io_req_bits_data	in	1		Store 指令要存的数据	
io_req_bits_tag_match	in	1	MSHR_0 MSHR_1	是否与 meta 中的 tag 匹配	
io_req_bits_old_meta_tag	in	20	MSHR_0 MSHR_1	要替换出去 line 的 tag	
io_req_bits_old_meta_coh_state	in	2	MSHR_0 MSHR_1	一致性状态信息	
io_req_bits_way_en	in	4	MSHR_0 MSHR_1	要替换出去哪一路	
io_resp_ready	in	1	resp_arb	外部是否准备好接受 io_resp 请求	1:有效

io_resp_valid	out	1	resp_arb	
io_resp_bits_addr	out	1	resp_arb	
io_resp_bits_tag	out	9	resp_arb	
io_resp_bits_cmd	out	5	resp_arb	
io_resp_bits_typ	out	3	resp_arb	
io_resp_bits_data	out	64	resp_arb	
io_resp_bits_nack	out	1	resp_arb	
io_resp_bits_replay	out	1	resp_arb	
io_resp_bits_has_data	out	1	resp_arb	
io_resp_bits_data_word_bypass	out	64	resp_arb	
io_resp_bits_store_data	out	64	resp_arb	
io_secondary_miss	out	1		是否是二次失效 1:是
io_mem_req_ready	in	1	resp_arb	下一级存储器好 1:是 mem 请求
io_mem_req_valid	out	1	resp_arb	mem_req 信号 1:是 是否有效
io_mem_req_bits_addr_block	out	26	resp_arb	tag+index
io_mem_req_bits_client_xact_id	out	2	resp_arb	那个 mshr 的请 求
io_mem_req_bits_addr_beat	out	2	resp_arb	一个 line 共有 4 个 beat, 标识哪

				一个 beat
io_mem_req_bits_is_builtin_type	out	1	resp_arb	
io_mem_req_bits_a_type	out	3	resp_arb	
io_mem_req_bits_union	out	17	resp_arb	
io_mem_req_bits_data	out	18	resp_arb	
io_refill_way_en	out	4		要填充哪一路
io_refill_addr	out	12		要填充的地址单元:地址后 12 位
io_meta_read_ready	in	1	meta_read_arb	外部是否准备好 1:是 接收 meta_read 请求
io_meta_read_valid	out	1	meta_read_arb	Meta_read 信号 1:是 是否有效
io_meta_read_bits_idx	out	6	meta_read_arb	哪一个 set
io_meta_read_bits_way_en	out	4	meta_read_arb	哪一路
io_meta_read_bits_tag	out	20	meta_read_arb	tag
io_replay_ready	in	1	meta_read_arb	外部是否准备好 1:是 重启流水线
io_replay_valid	out	1	meta_read_arb	Replay 信号是否 1:是 有效
io_replay_bits_addr	out	40	meta_read_arb	地址
io_replay_bits_tag	ou	9	meta_read_	Tag

	t		arb	
io_replay_bits_cmd	out	5	meta_read_arb	指令类型
io_replay_bits_typ	out	3	meta_read_arb	
io_replay_bits_kill	out	1	meta_read_arb	
io_replay_bits_phys	out	1	meta_read_arb	
io_replay_bits_data	out	64	meta_read_arb	Store 指令的 data
io_mem_grant_valid	in	1	MSHR_0 MSHR_1 IOMSHR	mem_grant 信号是否有效
io_mem_grant_bits_addr_beat	out	2	MSHR_0 MSHR_1 IOMSHR	返回的数据是哪一个 beat
io_mem_grant_bits_client_xact_id	out	2	MSHR_0 MSHR_1 IOMSHR	哪一个 mshr 的
io_mem_grant_bits_manager_xact_id	out	4	MSHR_0 MSHR_1 IOMSHR	
io_mem_grant_bits_is_builtin_type	out	1	MSHR_0 MSHR_1 IOMSHR	
io_mem_grant_bits_g_type	out	1	MSHR_0 MSHR_1 IOMSHR	
io_mem_grant_bits_data	out	128	MSHR_0 MSHR_1 IOMSHR	下一级存储器返回的数据
io_wb_req_ready	in	1	wb_req_arb	下一级存储器是否准备好接收写回请求 1: 准备好

io_wb_req_valid	out	1	wb_req_arb	写回请求信号是否有效 1:有效
io_wb_req_bits_addr_beat	out	2	wb_req_arb	哪一个 beat
io_wb_req_bits_addr_block	out	26	wb_req_arb	Tag+index
io_wb_req_bits_client_xact_id	out	2	wb_req_arb	哪一个 mshr 发出的
io_wb_req_bits_voluntary	out	1	wb_req_arb	被 assign 为 1
io_wb_req_bits_r_type	out	3	wb_req_arb	Shared 态的时候返回 0，非 shared 返回 3
io_wb_req_bits_data	out	128	wb_req_arb	要写回的数据
io_wb_req_bits_way_en	out	4	wb_req_arb	

3.1.3 内部结构

内部结构如图 2-1 所示。

3.2Sdq 设计

3.2.1 模块描述

时序部件：初值、读/写过程

clk 有效时，启动 sdq，sdq_val 初始化为 0。

若 sdq_enq 有效，即 indata 要入 sdq 时，首先把组合逻辑计算出的 sdq

空闲位置 alloc_id 赋值给 sdq_alloc_id，之后将输入数据 indata 保存在 sdq[alloc_id]中，更新 sdq 的 valid 寄存器 sdq_val=sdq_val|alloc_id_oh。

若 free_enq 有效 ,即要从 sdq 中取出 outdata 数据 ,根据输入信号 free_id 确定要从 sdq 中取数据的位置，将取出的数据保存在 outdata 中：outdata=sdq[free_id] ，之后更新 sdq 的 valid 寄存器 sdq_val=sdq_val&~alloc_id_oh。

组合电路：传输过程描述（最好细化到每根线的说明）

保存 indata：检查 sdq_rdy 信号，若 sdq_val 有空闲位，即 sdq 中有空闲部分可以接受 indata，则将空闲位置赋给 alloc_id，同时生成 alloc_id_oh 信号，记录 valid 寄存器中哪一位空闲。从 sdq 中取出数据，即 outdata：检查 free_id 信号以确定要释放的 sdq 位置并保存在 free_id_oh 中。

存数据过程：当 sdq_enq 有效，将 data 存储到 sdq，返回存储的位置（sdq_alloc_id），并且将保存 data 对应单元的 valid 域置为有效。

取数据过程 :当 free_sdq 有效 ,将 free_id 指向单元的 data 输出 ,将 valid 域置为无效。

3.2.2 接口信号说明

sdq 与外部模块的接口信号，如表 3-2 所示：

表 3-2 sdq 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	全部模块	时钟信号	控制时钟有效

sdq_enq	in	1	sdq	入队列使能	使能信号
free_sdq	in	1	sdq	出队列使能	使能信号
reg_sdq_rdy	out	寄存器	sdq	记录 queue 中空闲单元	保存 queue 中空闲单元 id
free_id	in	3	sdq	出队列 id	从 queue 中 id 单元取数据
sdq_alloc_id	out	寄存器	sdq	选择信号	选择 queue 中某个单元
indata	in	64	sdq	要存入 queue 的数据	64 位
outdata	out	寄存器	sdq	要从 queue 中取出的数据	输出到 64 位寄存器中

3.2.3 内部结构

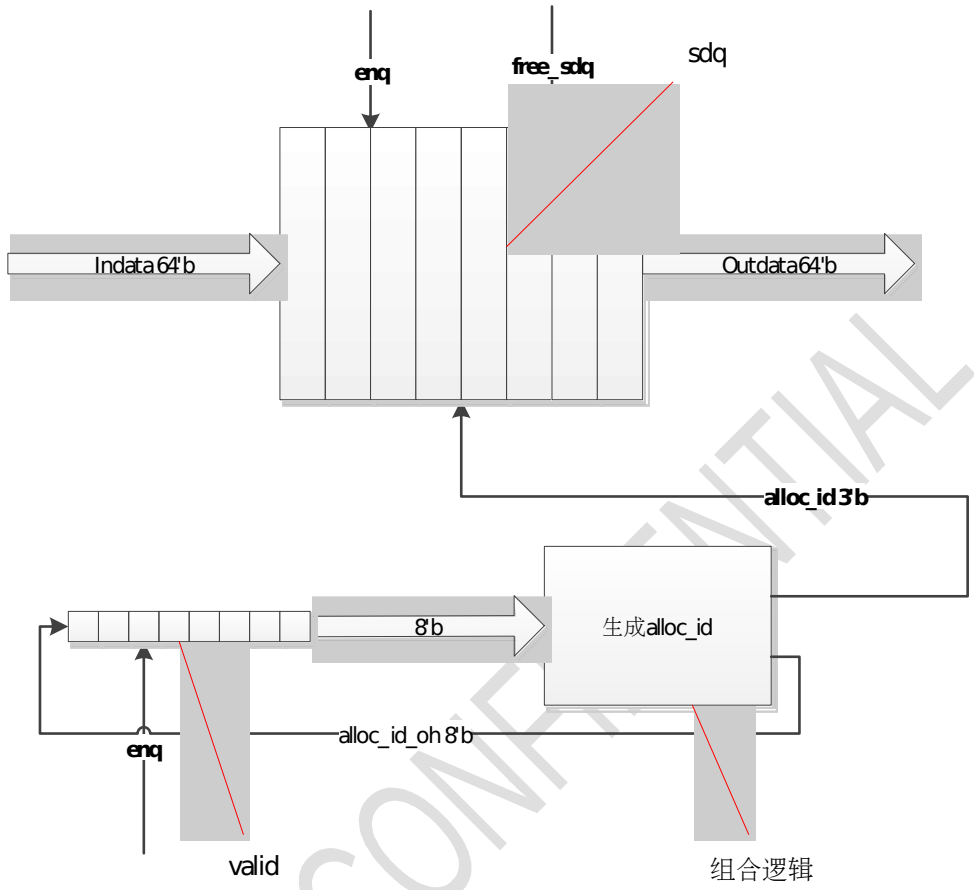

图 3-1 sdq 内部结构图

3.3仲裁器与多选器设计

3.3.1 模块描述

存失效请求过程 :当有一个失效请求进入 mshrfile 时 ,alloc_arb 首先启动，在 mshr 发来的空闲位置仲裁，找到一个存放失效请求的 mshr 并向 mshr 外部发送 alloc_arb.io.out 信号，标记已经完成失效请求的记录。

处理失效请求过程：从 mshr 中有可能向同一仲裁器发送编号不同的同种请求，剩下的 6 个不同的 arbiter 就是负责从发来的需要服务的请求中选择一个，打开与其相连的端口进行 mem_req、meta_read、meta_write、replay 请求的服务。

3.3.2 接口信号说明

仲裁器与多选器与外部模块的接口信号，如表 3-2 所示：

表 3-3 仲裁器与多选器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_in_0_valid	in	1	alloc_arb meta_read_arb meta_write_arb mem_req_arb wb_req_arb replay_arb	编号为 0 的 mshr 模块发出请求	若模块发出请求，则 valid 置 1
io_in_1_valid	in	1	alloc_arb meta_read_arb meta_write_arb mem_req_arb wb_req_arb replay_arb	编号为 1 的 mshr 模块发出请求	若模块发出请求，则 valid 置 1
io_out_chosen	out	1	alloc_arb meta_read_arb meta_write_arb mem_req_ar	仲裁器仲裁为哪个 mshr 模块	仲裁器选择出的为其提供服务的 mshr 模块编号

			b wb_req_arb replay_arb	服务	
io_in_0_ready	out	1	alloc_arb meta_read_arb meta_write_arb mem_req_arb wb_req_arb replay_arb	将与编号 0 相连的端口打开	为编号为 0 的 mshr 模块提供服务
io_out_1_ready	out	1	alloc_arb meta_read_arb meta_write_arb mem_req_arb wb_req_arb replay_arb	将于编号 1 相连的端口打开	为编号为 1 的 mshr 模块提供服务

3.3.3 内部结构

mshr 模块一共用到 6 个 arbiter，如图 3-3 所示：

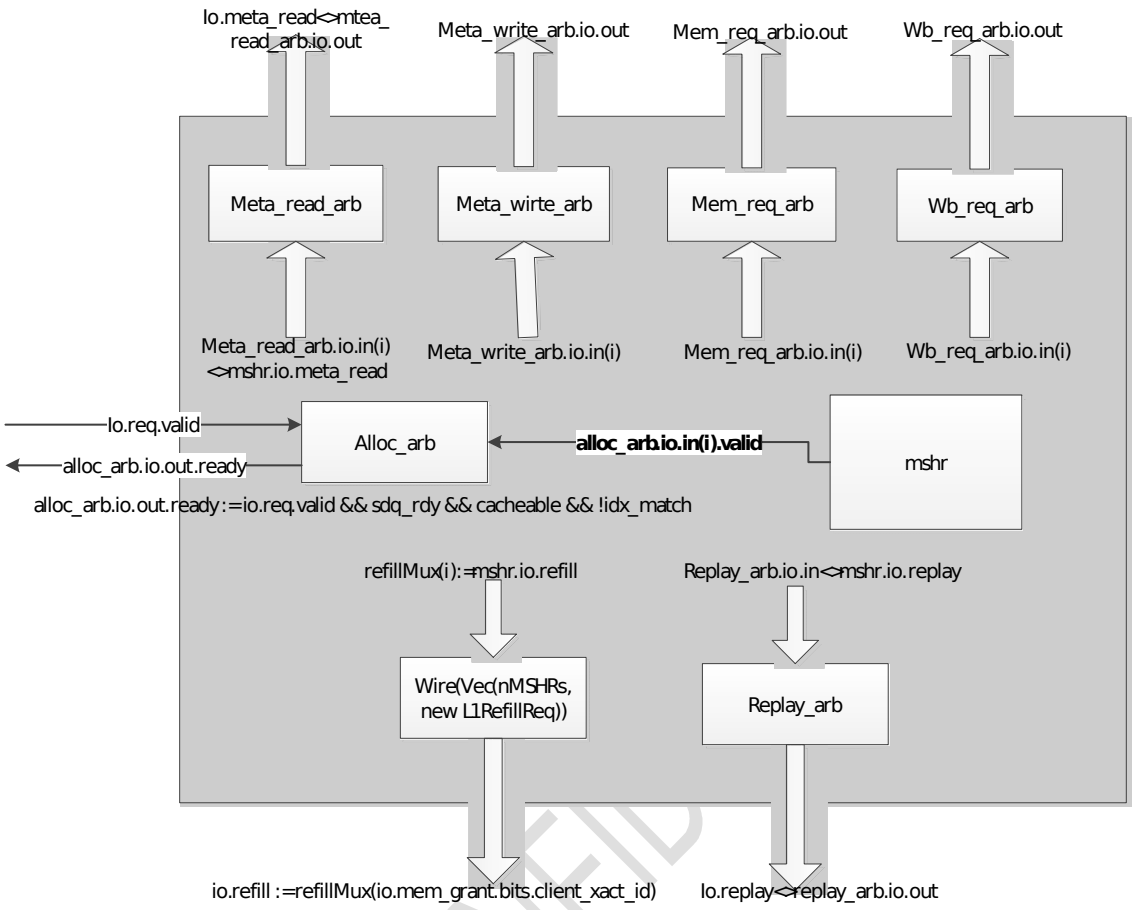

图 3-2 mshr 内部 arbiter

其中，alloc_arb、meta_read_arb、meta_write_arb、wb_req_arb、replay_arb 五个仲裁器模块都是从两个请求中仲裁出一个并为仲裁出的请求提供服务，它们的内部结构如图 3-3 所示：

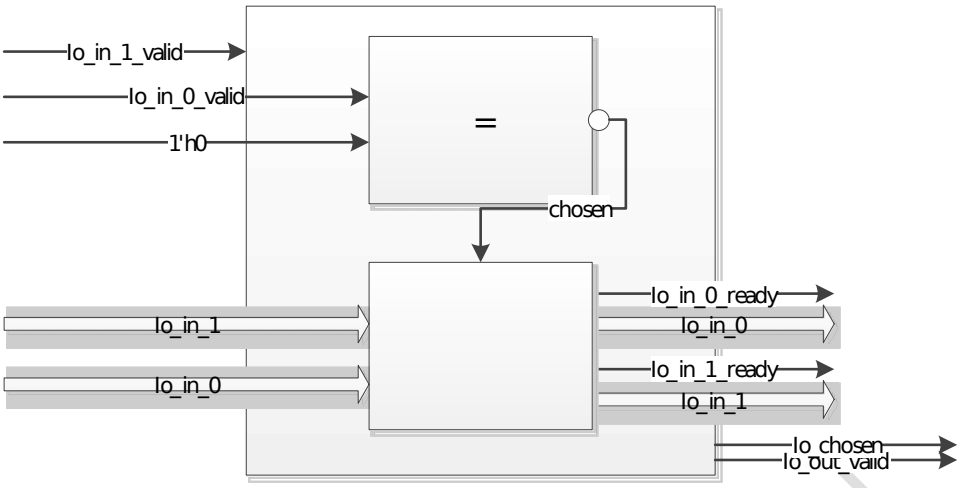

图 3-3 无锁存功能 arbiter

mem_req_arb 仲裁器模块较为特殊，它虽然也是只在两个请求中仲裁，但是由于它仲裁的请求都是请求在 cache 和内存之间传输数据的，所以有可能一个 burst 无法传输完全部需要传输的数据，所以这个仲裁器是一个带锁存功能的仲裁器（LockingArbiter）。在外部的两个请求到来时，它也要检查仲裁器自身是否被锁存，即上一次传输的数据是否传输完，若没有，则仲裁器被锁存，不从外部的两个请求中选择，直到上次的服务完成，即传输完要传输的数据后，才从外部的两个请求中仲裁出要为哪个请求服务。它的内部结构如图 3-4 所示：

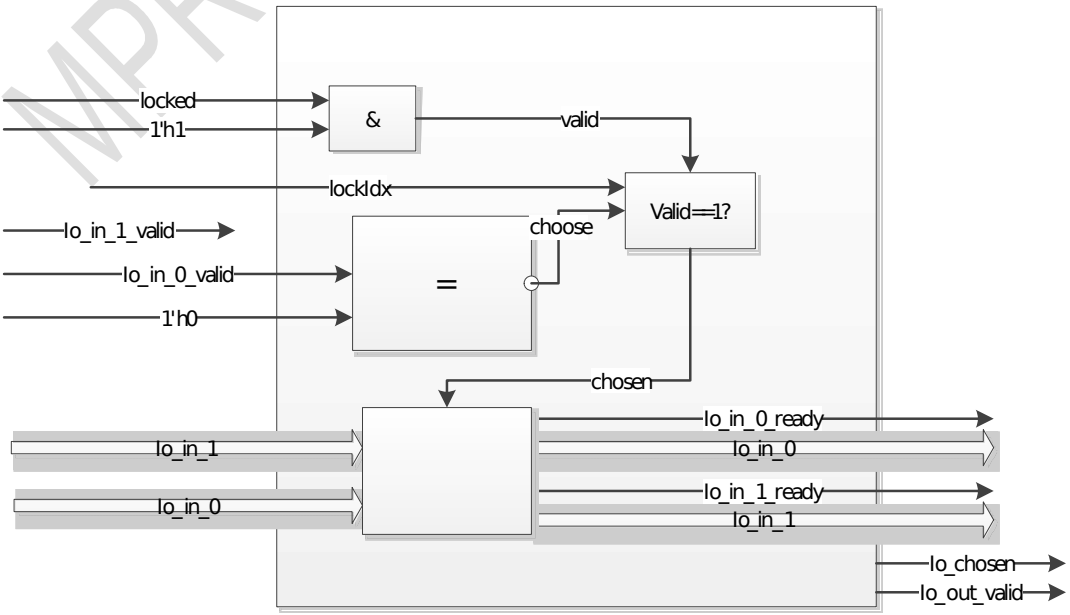

图 3-4 带锁存功能 arbiter

4 MSHR

4.1MSHR 结构设计

4.1.1 模块描述

MSHR 内部结构框图如图 4-1 所示。

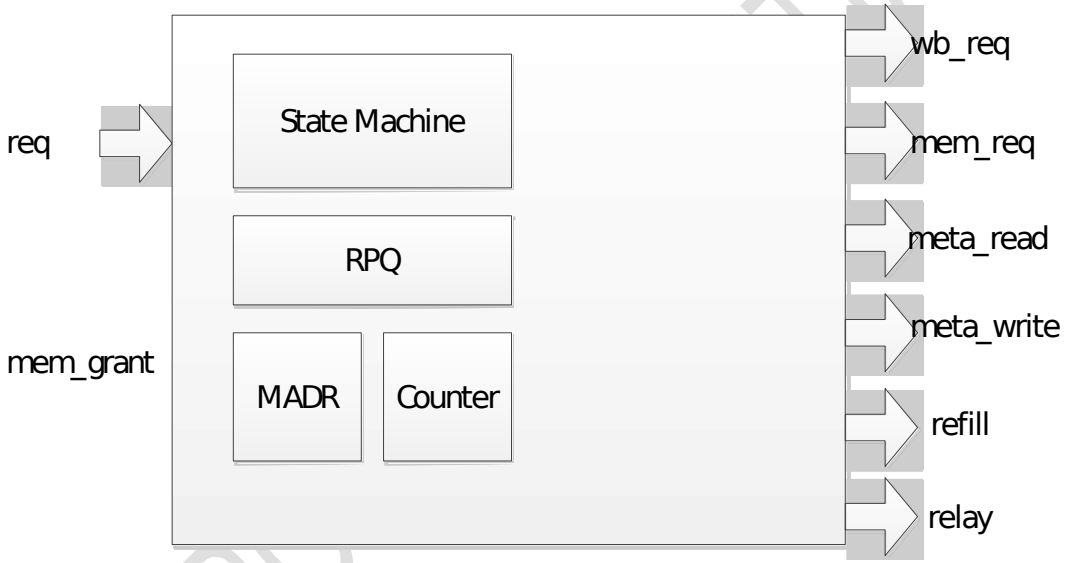

图 4-1 MSHR 内部结构框图

MSHR 的功能是保存失效命令，并利用状态机控制处理失效，最后将处理完成的失效命令重新发往 Cache 流水线。根据 MSHR 的功能来分析 MSHR 的设计结构：MSHR 的状态机用来控制处理失效的过程；MSHR 的 RPQ 用来保存所有失效命令；MSHR 的 MADR 用来保存当前所有接受请求中权限最高的请求，被用来处理失效；MSHR 的 Counter 用来对回填数据进行计数。

状态机是 MSHR 中非常重要的一部分，状态机的重要工作是控制 MSHR 有

序完成和其它模块交互，是通过发送控制信号完成。比如，MSHR 数据通路部分准备好了给 WriteBack 模块的数据，等状态机生成给 WriteBack 模块的使能信号后，WriteBack 才会接受这组数据信号，并进行工作。

MSHR 除去状态机的其它结构就剩下简单的数据通路。数据通路主要完成两个工作：生成给状态机的信号；组织要和其它模块交互的数据，这些数据基本上来自于 MSHR 的输入请求信号，而不需要 MSHR 模块内部进行复杂转化来生成，所以 MSHR 的数据通路部分比较简单。

4.1.2 接口信号说明

MSHR 外部模块的接口信号，如 4-1 所示。

表 4-1 MSHR 与外部模块的接口信号

所属接口	端口名称	方向	宽度	作用	描述
PRI_VAL	io_req_pri_val	in	1	首次失效有效信号	1：有效 0：无效
PRI_RDY	io_req_pri_rdy	out	1	响应接受首次失效	1：接受 0：不接受
SEC_VAL	io_req_sec_val	in	1	二次失效有效信号	1：有效 0：无效
SEC_RDY	io_req_sec_rdy	out	1	响应接受二次失效	1：接受 0：不接受

IDX_MATCH	io_idx_match	out	1	两次请求的 idx 是否一样	1：一样 0：不一样
TAG	io_tag	out	20	访存地址的 tag 位域	addr[31:12]
PROBE_RDY	io_probe_rdy	out	1		
REQ	io_req_bits_addr	in	40	访存地址	
	io_req_bits_tag	in	9	访存标签	
	io_req_bits_cmd	in	5	访存指令类型	
	io_req_bits_typ	in	3		
	io_req_bits_kill	in	1		
	io_req_bits_phys	in	1		
	io_req_bits_sdq_id	in	5	数据存放在 sdq 中的位置	
	io_req_bits_tag_match	in	1	是否命中	1：命中 2：失效
	io_req_bits_old_meta_tag	in	20	替换行的物理 tag	
	io_req_bits_old_meta_coh_state	in	2	替换行的一致性状态	0:Invalid 1:Shared 2:ExclusiveDirty 3:ExclusiveClean
	io_req_bits_way_en	in	4	替换行所在的 way	
MEM_REQ	io_mem_req_ready	in	1	mem 响应接	1：接受 0：暂未

			受请求	接受
io_mem_req_valid	out	1	请求 mem 有效信号	1：有效 0：无效
io_mem_req_bits_addr_block	out	26	访存地址的(tag+idx)域	addr[31:6]
io_mem_req_bits_client_xact_id	out	2	当前 mshr 的 id	
io_mem_req_bits_addr_beat	out	2	访存地址的（beat）位域	addr[5:4]
io_mem_req_bits_is_builtin_type	out	1	决定 g_type 关联的类型	0: g_type 取值 grantShared=0 grantExcluive=1 grantExclusiveACK=2 1：g_type 取值 voluntaryACKType=0 prefetchACKType=1 putACKType=3 getdataBeatType=4 getdataBlockType=5
io_mem_req_bits_a_type	out	3		
io_mem_req_bits_union	out	17		

	io_mem_req_bits_data	out	128	数据	
REFILL	io_refill_way_en	out	4	用于索引 DataArray 的 way	
	io_refill_addr	out	12	idx+beat 位域	addr[11:4]<<4
META_READ	io_meta_read_ready	in	1	读 meta 响应信号	1：完成 0：暂未完成
	io_meta_read_valid	out	1	读 meta 有效信号	1：有效 0：无效
	io_meta_read_bits_idx	out	6	用于索引 meta 的 idx	addr[11:6]
	io_meta_read_bits_tag	out	20		
META_WRITE	io_meta_write_ready	in	1	写 meta 响应信号	1：完成 0：暂未完成
	io_meta_write_valid	out	1	写 meta 有效信号	1：有效 0：无效
	io_meta_write_bits_idx	out	6	用于索引 meta 的 idx	addr[11:6]
	io_meta_write_bits_way_en	out	4	用于索引 meta 的 way	替换算法选出来的 way
	io_meta_write_bits_data_tag	out	20	写入 meta	addr[31:12]

				的 tag	
	io_meta_write_bits_data_coh_state	out	2	写入 meta 的 coh_state	0:Invalid 1:Shared 2:ExclusiveDirty 3:ExclusiveClean
REPLAY	io_replay_ready	in	1	replay 响应信号	1：完成 0：暂未完成
	io_replay_valid	out	1	replay 有效信号	1：有效 0：无效
	io_replay_bits_addr	out	40	访存地址	
	io_replay_bits_tag	out	9	访存标签	
	io_replay_bits_cmd	out	5	访存指令类型	
	io_replay_bits_typ	out	3		
	io_replay_bits_kill	out	1		
	io_replay_bits_phys	out	1		
	io_replay_bits_sdq_id	out	5	数据存放在 sdq 中的位置	
MEM_GRANT	io_mem_grant_valid	in	1	mem 返回数据有效信号	1：有效 0：无效
	io_mem_grant_bits_addr_beat	in	2	mem 返回数据所属 beat	

	io_mem_grant_bits_client_xact_id	in	2	当前 mshr 的 id	
	io_mem_grant_bits_manager_xact_id	in	4		
	io_mem_grant_bits_is_builtin_type	in	1	决定 g_type 关联的类型	
	io_mem_grant_bits_g_type	in	4	内存返回的一致性状态	0: shared 1 : exclusive 2 :exclusiveACK
	io_mem_grant_bits_data	in	128	数据	
WB_REQ	io_wb_req_ready	in	1	wb 响应接受	1 : 接受 0 : 暂不接受
	io_wb_req_valid	out	1	请求 wb 有效信号	1 : 有效 0 : 无效
	io_wb_req_bits_addr_beat	out	2	访存地址的（beat）位域	addr[5:4]
	io_wb_req_bits_addr_block	out	26	访存地址的(tag+idx)域	addr[31:6]
	io_wb_req_bits_client_xact_id	out	2	当前 mshr 的 id	
	io_wb_req_bits_voluntary	out	1		

	io_wb_req_bits_r_type	out	3	
	io_wb_req_bits_data	out	128	数据
	io_wb_req_bits_way_en	out	4	替换行所在 way

4.1.3 内部结构

MSHR 顶层连线电路图如图 3-1 所示：

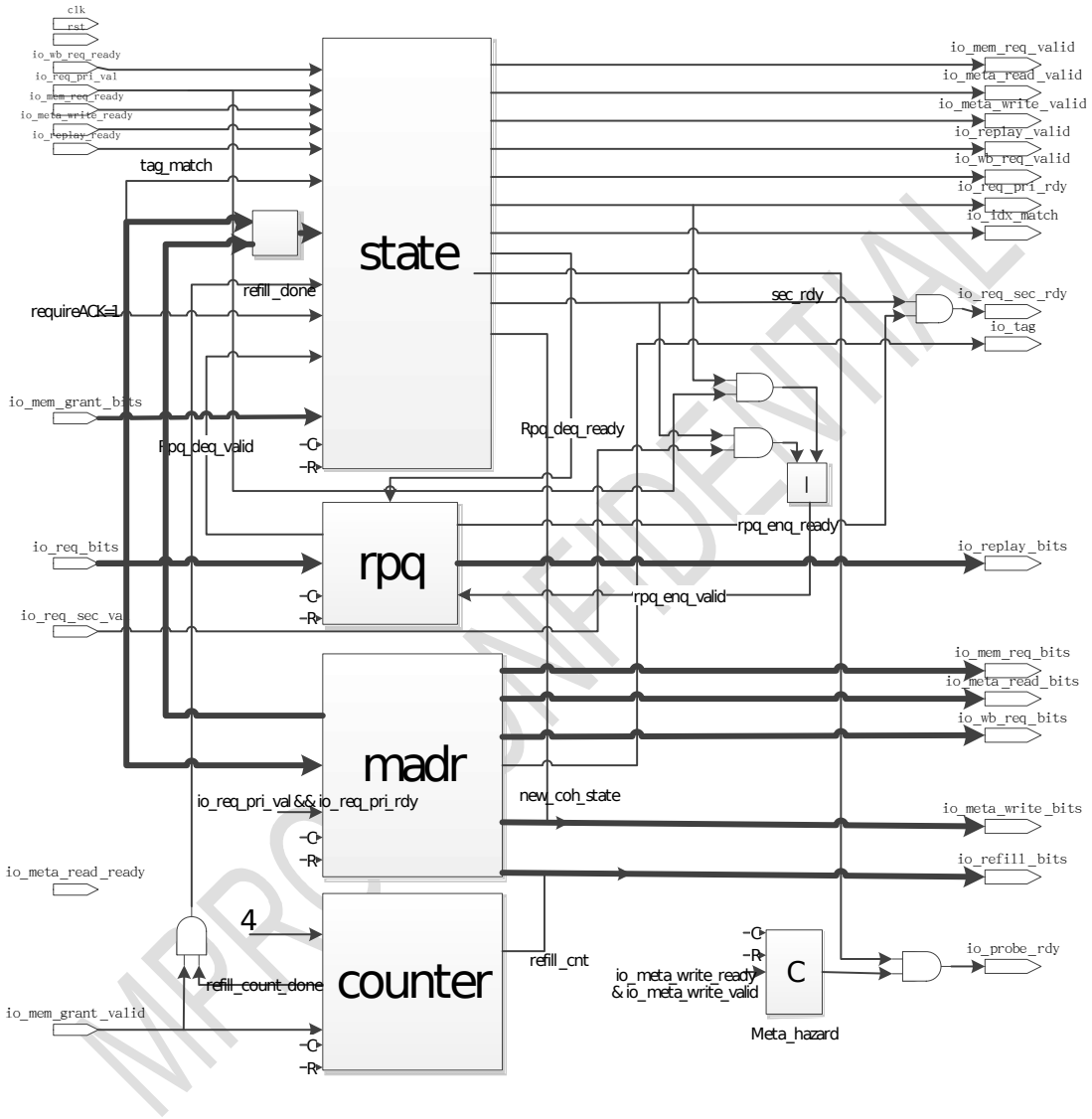

图 4-2 MSHR 顶层连线电路图

4.2失效处理状态机设计

4.2.1 模块描述

该模块是对失效请求的处理过程，状态机运转一周，能够完成 RPQ 中所有命令的处理。该失效状态机共分四种情况，如下所示：

- 情况一：Cache 不命中并且为复杂失效（替换行是 ExclusiveDirty 态）时，替换行要写回内存。
- 情况二：Cache 不命中并且为简单失效（替换行是 Invalid，Shared，ExclusiveClean 态）时，替换行不需要写回内存。
- 情况三：Cache 命中，要修改其它存储层一致性，即此次失效请求为写请求，当前状态是 Shared 态，需要先请求内存通知其它存储层变成 Invalid 态，再将本 Cache 变成 ExclusiveDirty 态。
- 情况四：Cache 命中，要修改本 Cache 一致性，即当前失效请求为写请求，当前状态是 ExclusiveClean 态，需要迁移到 ExclusiveDirty 态。

4.2.2 接口信号说明

与外部模块的接口信号

状态机与外部模块的接口信号，如表 4-2 所示：

表 4-2 状态机与外部模块的接口信号

端口名称	方	宽	相连模	作用	描述

	向	度	块名称	
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_pri_val	in	1	判断是否发射了一个失效请求	失效请求信号
io_req_bits_tag_match	in	1	判断 tag 是否匹配	tag 匹配信号
io_meta_write_ready	in	1	判断 meta 是否可写	meta 响应信号
io_replay_ready	in	1	流水线是否有效，是否可重返流水线	重返流水线信号
idx_match	in	1	判断是否有二次失效	idx 匹配信号
cmd_requires_second_acquire	in	1	判断是否有第二次失效请求	二次失效请求
coh_isHit	in	1	判断一致性状态是否命中	coh 命中信号
coh_require_wb	in	1	判断是复杂时效还是简单失效，从而判断是否需要写回	writeback 请求信号
wb_req_fire	in	1	判断是否接受了 writeback 请求	writeback 响应信号
wb_req_requireAck	in	1	判断是否需要等待	等待

			writeback 写回下一级存储	writeback 信号
io_mem_grant_valid	in	1	判断写回是否结束	mem writeback 完毕信号
mem_req_fire	in	1	判断 mem 是否接收了取数据请求	向 mem 取数据响应信号
rpq_deq_valid	in	1	判断失效请求是否发射完毕，rpq 是否为空	失效请求发射完毕
refill_done	in	1	判断数据是否写入完毕	数据写入完毕信号
coh_on_grant	in	2	对新一致性状态赋值	一致性信息
coh_on_hit	in	2	对新一致性状态赋值	一致性信息
io_wb_req_valid	out	1	判断是否进行了 writeback	writeback 响应信号
io_meta_write_valid	out	1	判断 meta 是否可写	写 meta 响应信号
io_mem_req_valid	out	1	判断是否从 mem 取了数据	mem 响应信号
io_meta_read_valid	out	1	判断 meta 是否可读	读 meta 响应信号

io_replay_valid	out	1	判断是否重回了流水线	重回流水线响应信号
io_idx_match	out	1	判断 idx 是否匹配	idx 匹配信号
io_req_pri_rdy	out	1	是否接收了失效请求	失效请求接收响应信号
rpq_dep_ready	out	1	是否发射失效请求	失效请求发射信号
new_coh_state	out	2	存储状态一致性信息	处理失效请求后的状态一致性信息
sec_rdy	out	1	是否接收二次失效请求	二次失效请求信号

4.2.3 内部结构

4.2.3.1 失效处理状态机图

失效处理状态转换如图所示：

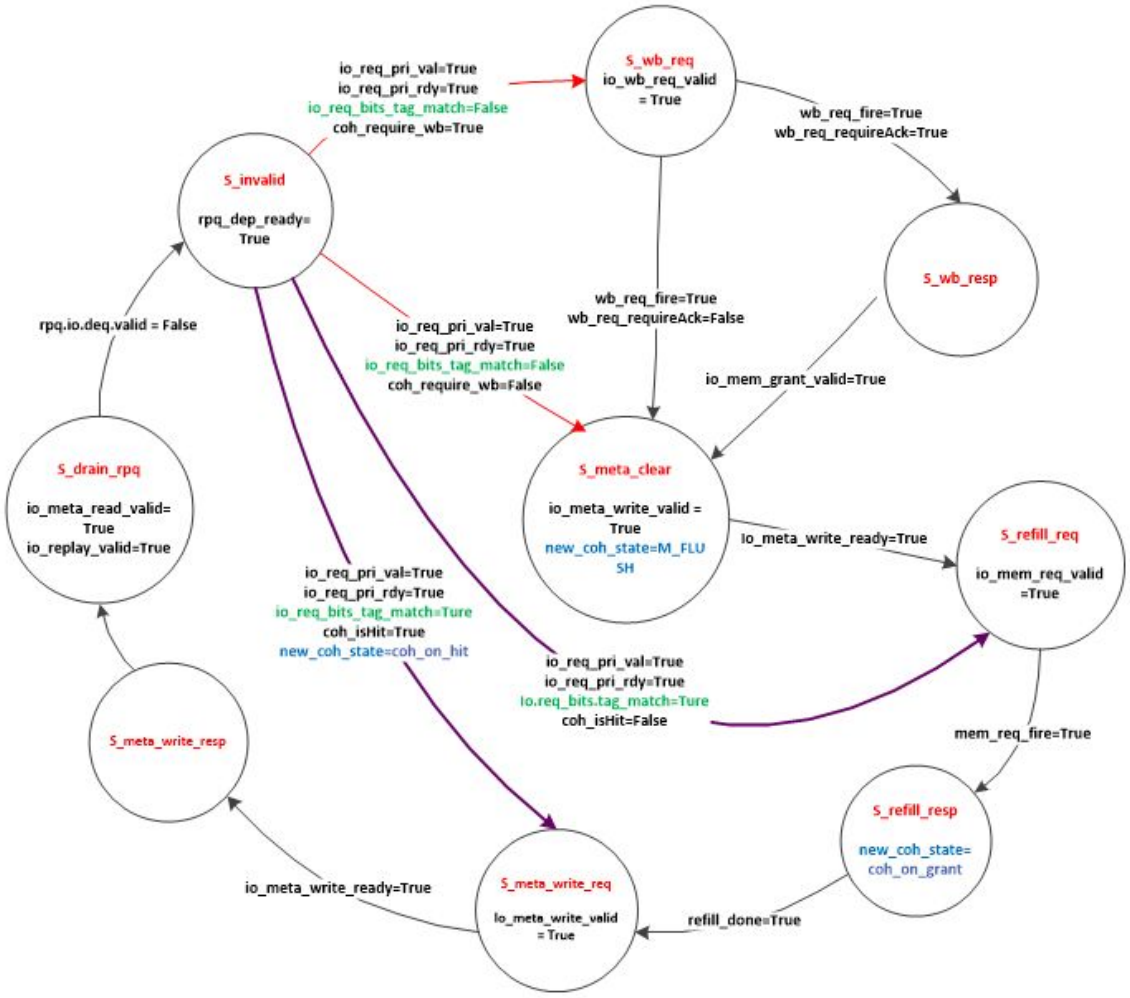

图 4-3 失效处理状态转换图

4.2.3.2 失效处理状态机各状态含义说明

失效处理状态机共有 9 个状态，各状态的含义如表所示：

表 4-3 失效处理状态机状态含义表

编号	状态名称	状态描述
0000	s_invalid	等待接受失效请求
0001	s_wb_req	请求 writeback 回写替换行

0010	s_wb_resp	等待 writeback 回写替换行，并且等待下一级存储的应答
0011	s_meta_clear	清空 meta_line，使得对应的 cache line 处于 invalid 态（可写）
0100	s_refill_req	请求读下一级存储
0101	s_refill_resp	将下一级存储返回的数据写入 cache
0110	s_meta_write_req	请求写 meta_line
0111	s_meta_write_resp	写 meta_line 完毕
1000	s_drain_rpq	失效请求重回流水线

4.2.3.3 失效状态机的状态转换表

- 状态转移一：

s_invalid→s_wb_req→s_wb_resp→s_meta_clear→s_refill_req→s_refill_resp→s_meta_write_req→s_meta_write_resp→s_drain_rpq

Cache 不命中并且为复杂失效（替换行是 exclusivedirty 态）时，替换行要写回 mem。然后清空 meta_line，再向 mem 读取数据写入 dataarray 中，接着写 metaline，最后失效请求重回流水线。

- 状态转移二：

s_invalid→s_meta_clear→s_refill_req→s_refill_resp→s_meta_write_req→s_meta_write_resp→s_drain_rpq

cache 不命中并且为简单失效(替换行是 invalid ，shared ，exclusiveclean 态) 时，替换行不需要写回 mem，直接清空 meta_line，再向 mem 读取数据写入 dataarray 中，接着写 metaline，最后失效请求重回流水线。

- 状态转移三：

s_invalid→s_refill_req→s_refill_resp→s_meta_write_req→s_meta_write_resp→s_drain_rpq

cache 命中，要修改其它存储层一致性，即此次失效请求为写请求，当前状态是 shared 态，需要先请求 mem 通知其它存储层变成 invalid 态，再将本 cache 变成 exclusivedirty 态。所以直接请求 mem，向 mem 读取数据写入 dataarray 中，接着写 metaline，最后失效请求重回流水线。

- 状态转移四：

  s_invalid→s_meta_write_req→s_meta_write_resp→s_drain_rpq

  cache 命中，要修改本 cache 一致性，即当前失效请求为写请求，当前状态是 exclusiveclean 态，需要迁移到 exclusivedirty 态。所以跳过请求 mem 直接请求写 metaline，最后失效请求重返流水线。

失效状态机的状态转换如表所示：

表 4-4 失效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
s_invalid	s_wb_req	io_req_pri_val=True io_req_pri_rdy=True io_req_bits_tag_match=False coh_require_wb=True	tag没有匹配上 并且是个复杂失效请求 需要写回mem
	s_meta_clear	io_req_pri_val=True io_req_pri_rdy=True io_req_bits_tag_match=False coh_require_wb=False	tag没有匹配上 并且是个简单失效请求 不需要写回mem
	s_refill_req	io_req_pri_val=True io_req_pri_rdy=True	tag匹配上了

		Io.req_bits.tag_match=True coh_isHit=False	但是cache line 处于invalid态
	s_meta_write_req	io_req_pri_val=True io_req_pri_rdy=True io_req_bits_tag_match=Ture coh_isHit=True new_coh_state=coh_on_hit	tag匹配上了 需要进行状态迁移
s_wb_req	s_wb_resp	wb_req_fire=True wb_req_requireAck=True	wtiteback响应， 需要等待 writeback写回 下一级存储
	s_meta_clear	wb_req_fire=True wb_req_requireAck=False	writeback响应， 不需要等待 writeback写回 下一级存储
s_wb_resp	s_meta_clear	io_mem_grant_valid=True	下一级存储返回 写完毕应答信号
s_meta_clear	s_refill_req	Io_meta_write_ready=True	meta_line清空 完毕
s_refill_req	s_refill_resp	mem_req_fire=True	下一级存储返回 数据
s_refill_resp	s_meta_write_req	refill_done=True	数据写入完毕

s_meta_write_req	s_meta_write_resp	io_meta_write_ready=True	写meta_line完毕
s_meta_write_resp	s_drain_rpq	无条件跳转	
s_drain_rpq	s_invalid	rpq.io.deq.valid = False	失效请求发射完毕

4.2.3.4 失效状态控制信号的生成

表 4-5 失效状态控制信号生成表

输出信号	默认	有效条件	含义
io_wb_req_valid	0	写回应答信号，即处于 s_wb_req 状态时，接受写回请求	writeback 响应信号
io_meta_write_valid	0	meta_line 清空完毕或者数据写入完毕，即处于 s_meta_write_req 、s_meta_clear 状态时，meta 可写	写 meta 响应信号
io_mem_req_valid	0	从 mem 取数据应答信号，即处于 s_refill_req 状态 ,接受从 mem 取数据请求	下一级存储返回数据响应
io_meta_read_valid	0	失效请求处理完毕即处于 s_drain_rpq 状态时，meta 可读	读 meta 响应信号
io_replay_valid	0	失效请求处理完毕并且已没有失效请求	重回流水线响应信

		即处于 s_drain_rpq 状态并且 rpq_deq_valid=0 时 ,失效请求重回流水线	号
io_idx_match	0	idx 匹配并且正在处理其他失效信息，即 idx_match=1 并且不处于 s_invalid 状态时为 1	idx 匹配信号
io_req_pri_rdy	0	处于 s_invalid 状态时，可接受失效请求	接受失效请求响应信号
rpq_dep_ready	0	处于 s_invalid 状态或此次失效请求处理完毕即处于 s_drain_rpq 状态并且重返流水线有效时，rpq 可接受失效请求	rpq 状态信号
new_coh_state	00	meta_line 的一致性状态	meta_line 一致性状态
sec_rdy	0	处于 s_wb_req 或 s_wb_resp 或 s_meta_clear 状态时，接受处理二次失效请求，处于 s_refill_req 或 s_refill_resp 时，没有捕获第二次请求的命令为 1	接受处理二次失效请求

4.3Rpq 设计

4.3.1 模块描述

replay queue，保存需要 replay 的命令（未命中 cache 指令）信息（包括命令中的地址，tag，命令类型及 store 命令数据存放的 sdq id 等信息），这些信息保存在队列 rpq 中。

实现原理：

1）存数据过程：当 enq 信号有效时（enq_valid&&enq_rdy==1），将 data（rpq.io.enq.bits）存储到 rpq 中，移动尾指针 enq_ptr。

2）取数据过程：当 deq 信号有效时（deq_valid&&deq_rdy==1），将队列头部单元（rpq.io.deq.bits）的 data 弹出，移动头指针 deq_ptr。

内部设计：

1.时序部件：

1）时序部件寄存器：q_reg

初值：q_reg—空，deq_valid—0，enq_rdy—1

读过程：当 deq 信号有效时（deq_valid&&deq_rdy==1），将队列头部单元弹出，更新 rpq.io.deq.bits 为队列头单元数据。

写过程：当 enq 信号有效时（enq_valid&&enq_rdy==1），将 rpq.io.enq.bits 放入队列尾。

2）时序部件寄存器：m_reg

初值：m_reg—0

当 do_enq!=do_deq 时，m_reg=do_enq，否则值不变。表示当前时刻

队列的 maybe_full 状态。

2.组合电路：

若队列为空 rpq_deq_valid 为 0 ,否则为 1 ;若队列为满设 rpq_enq_ready 为 0，否则为 1。

4.3.2 接口信号说明

与外部模块的接口信号

仲裁器与多选器与外部模块的接口信号，如表 4-6 所示：

表 4-6 rpq 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	
Rst	in	1		复位	
entries	in	qdepth（参数 2）		队列容量	
enq_bits	in	bits_width（参数 1）		入队列元素	
enq_valid	in	1		表示是否有元素要入队列	有则为 1，无为 0
deq_rdy	in	1		表示队列头元素是否出队列	出队列为 1，否则为 0

deq_bits	out	bits_width	队尾元素	
count	out	qdepth	当前队列元素数	
enq_ready	out	1	队列非满	为 1 表示队列非满，可入队列元素
deq_valid	out	1	队列非空	为 0 表示队列非空，可出队列元素

4.3.3 内部结构

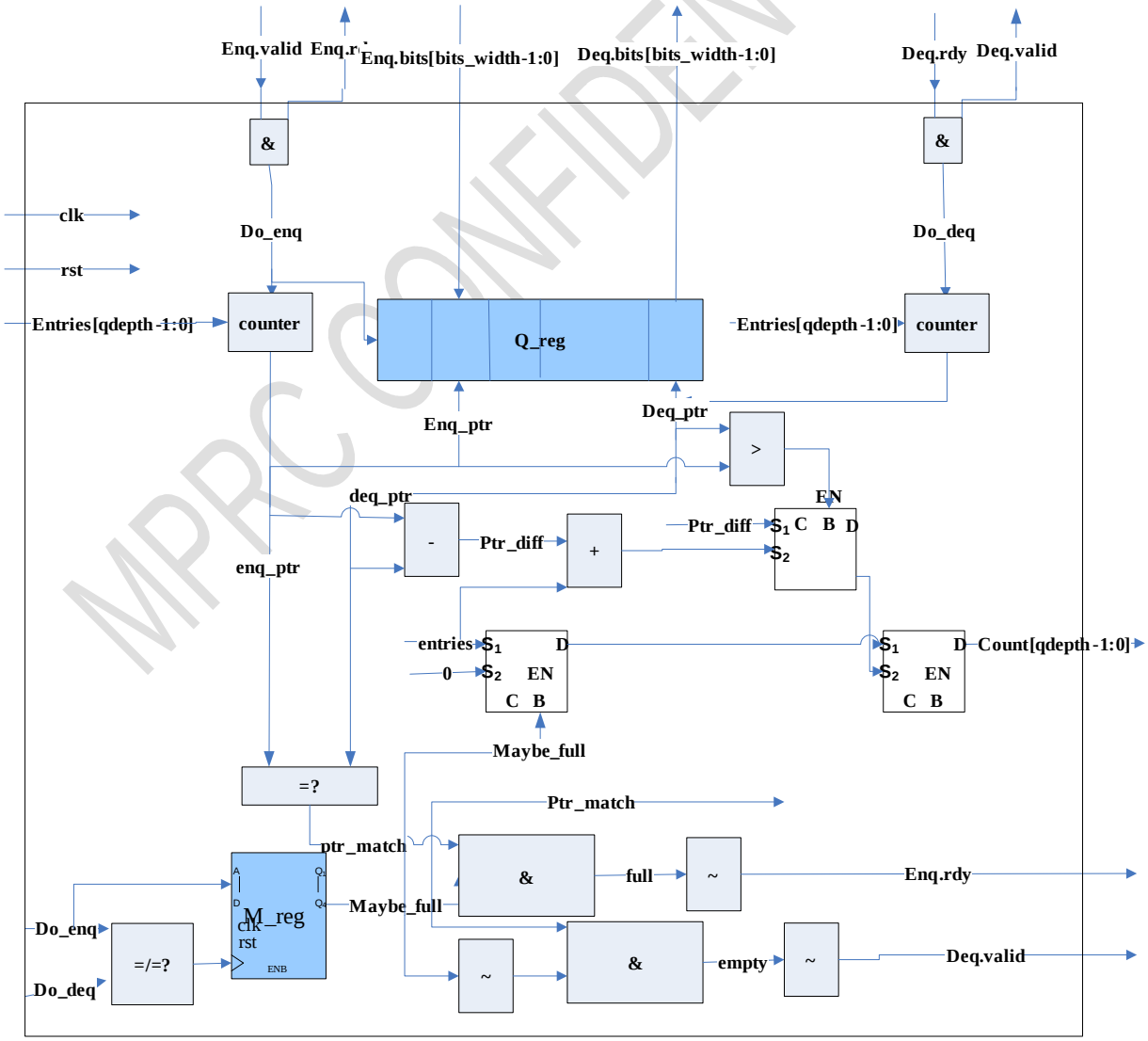

图 4-4 rpq 内部图

4.4计数器设计

4.4.1 模块描述

使能信号 en 有效（=1）时计数，判断计数器当前值 value，若达到计数器计数上限则 c_complete 设 true，计数器 value 清零，否则 c_complete 设 false，计数值 value 加 1。en 无效暂停计数，value 不变，c_complete 不变。

内部设计：

1.时序部件：

时序部件寄存器：value

初值： value—0

计数过程：

enq 有效，value=v_in.

enq 无效，停止计数，值不变。

2.组合电路：

1）判断计数值 v_out 是否等于 n-1，若为 n-1，表示此次计数到此完成，输出 wrap 为 1，v_in 为 0，否则 wrap 为 0， v_in=v_out+1。

2）c_complete=wrap&enq。

4.4.2 接口信号说明

与外部模块的接口信号

计数器与外部模块的接口信号，如表 4-7 所示：

表 4-7 计数器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	
rst	in	1		复位	
n	in	depth（参数）		计数器计数上限	计数为从 0—n-1
En	in	1		使能信号	使能为 1 ,计数器工作
v_out	out	depth		当前计数器值	
c_complete	out	1		完成信号	当计数器计数值达到 n-1 , out1 , 表示计数完成，其它情况为 0.

4.4.3 内部结构

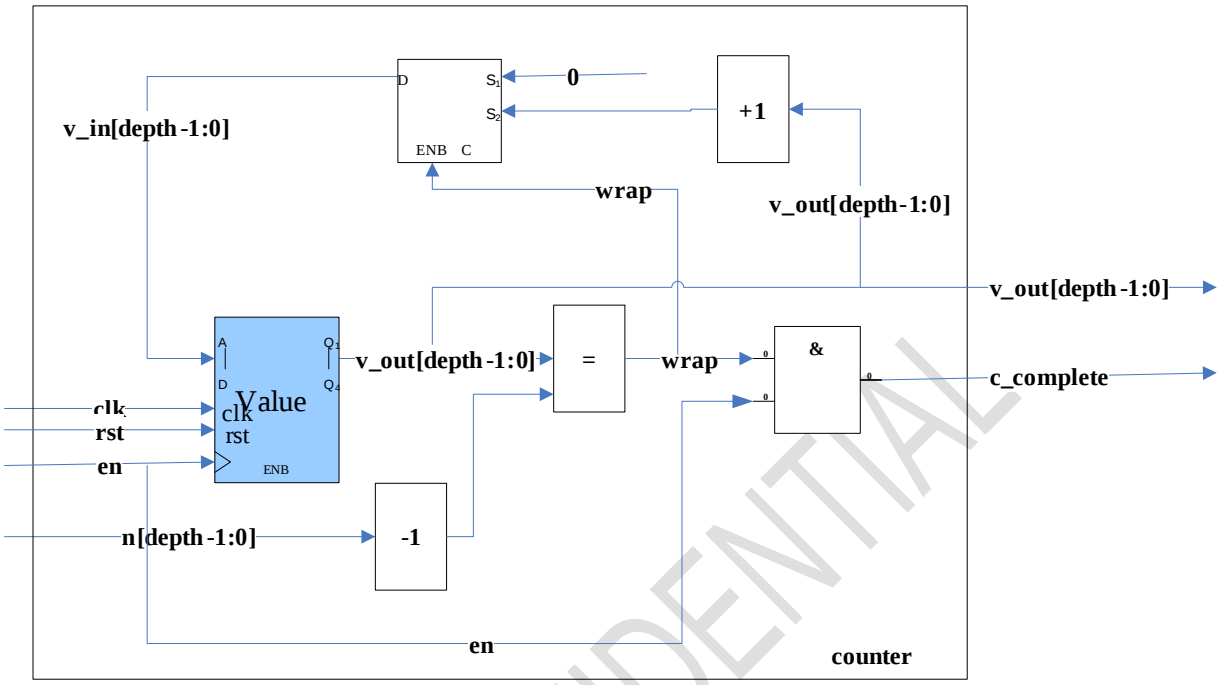

图 4-5 计数器内部结构图