**Baiyang IP设计文档**

北京开源芯片研究院

2024.8.19

Baiyang IP详细设计

[1. 外部接口 4](#_Toc2133)

[2. 整体结构 5](#_Toc28102)

[3. AXI2UI设计说明 6](#_Toc24102)

[3.1. 模块功能 6](#_Toc8732)

[3.2. 接口信号 7](#_Toc29037)

[3.3. 设计思路 8](#_Toc25229)

[3.3.1. Token\_generator 9](#_Toc8164)

[3.3.2. Burst\_clip 9](#_Toc27624)

[3.3.3 Consistency 9](#_Toc6450)

[3.3.4 Read Cmd Buffer 10](#_Toc17666)

[3.3.5 Read Reorder Buffer 10](#_Toc12153)

[3.3.6 UI 11](#_Toc19343)

[3.3.7 优化点 11](#_Toc13649)

[4. Filter设计说明 12](#_Toc3247)

[4.1. 模块功能 12](#_Toc7320)

[4.2. 接口信号 12](#_Toc28420)

[4.3. 设计思路 13](#_Toc25789)

[4.3.1. 功能实现 13](#_Toc8775)

[4.3.2. 详细设计 13](#_Toc7286)

[4.3.3. 优化点 14](#_Toc10423)

[5. Addr Map设计说明 15](#_Toc6648)

[5.1. 模块功能 15](#_Toc12508)

[5.2. 接口信号 15](#_Toc26158)

[5.3. 设计思路 17](#_Toc27241)

[5.3.1. 静态映射 17](#_Toc31374)

[5.3.2. 优化点 18](#_Toc5969)

[6. Advanced Scheduler设计说明 19](#_Toc11877)

[6.1. Advanced Scheduler模块功能 19](#_Toc25151)

[6.2. 接口信号 19](#_Toc22498)

[6.3. 设计思路 21](#_Toc3603)

[6.3.1. 两个源的读写仲裁 21](#_Toc1577)

[6.3.2. CMDStation 21](#_Toc28182)

[6.3.3. bank state 22](#_Toc9516)

[6.3.4. 读写冲突 22](#_Toc21764)

[6.3.5. 数据处理，token处理 22](#_Toc8479)

[6.3.6. Schedule logic 23](#_Toc22807)

[7. SDRAM Command Generator设计说明 25](#_Toc3779)

[7.1. 模块功能 25](#_Toc32184)

[7.2. 接口信号 25](#_Toc18142)

[7.3. 设计思路 28](#_Toc6027)

[7.3.1. Fifo\_wrap 29](#_Toc5784)

[7.3.2. DataStorage 29](#_Toc9700)

[7.3.3. RdDataStorage 29](#_Toc29590)

[7.3.4. Mig\_wrap 29](#_Toc29936)

[7.3.5. Command Disassembly 30](#_Toc23043)

[7.3.6. WidthMatch 31](#_Toc13843)

[8. System Cache设计说明 33](#_Toc27637)

[8.1. 模块功能 33](#_Toc26700)

[8.2. 接口信号 33](#_Toc25795)

[8.3. 设计思路 35](#_Toc30875)

[8.3.1. Cache Wrapper 36](#_Toc18507)

[8.3.2. Request Queue 37](#_Toc705)

[8.3.3. Cache Array 38](#_Toc13632)

[8.3.4. MainPipe 40](#_Toc7376)

[8.3.5. RefillPipe 44](#_Toc15084)

[8.3.6. Miss Status Holding Registers (MSHR) 45](#_Toc11362)

[8.3.7. WriteCombineBuffer(WCB) 46](#_Toc22034)

# 外部接口

开源内存控制器 IP (玉泉第一代“白杨” IP , Baiyang) 包括地址映射、分流、系统缓存、调度等功能模块，支持DDR4-2400内存工作频率，支持AXI4 协议、DFI3.0协议。经测试，100万次读写内存，顺序访存序列读写内存正确，随机访存序列读写内存正确，12个香山核SPEC CPU2006程序访存序列切片读写内存正确。

Baiyang IP 其外部接口如下图所示，包括AXI总线接口和DFI总线接口，符合AXI4 协议、DFI3.0协议。

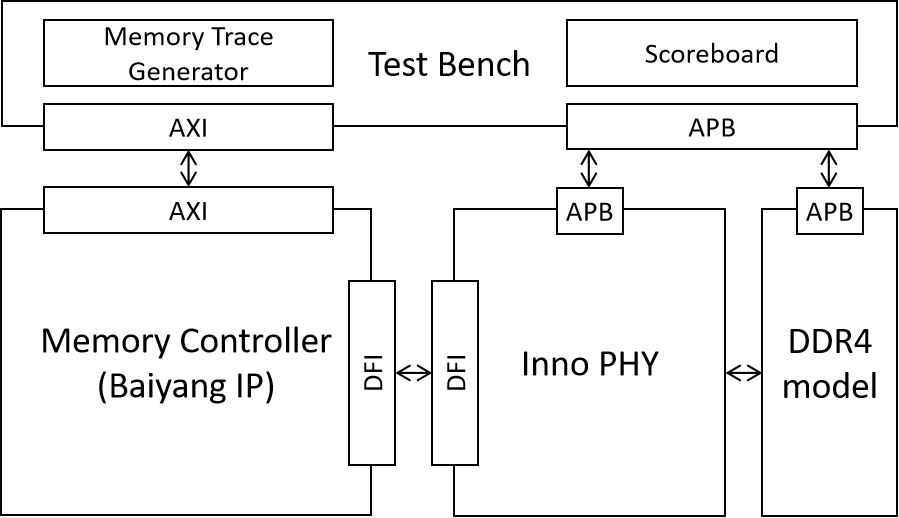


图1-1 Baiyang IP外部接口图

# 整体结构

Baiyang IP整体结构如下图所示，包括地址映射、分流、系统缓存、调度和命令生成功能模块。

AXI2UI模块包括ar/aw/r/w/b通道缓冲、仲裁、寄存等逻辑，将符合AXI4协议的读写请求拆分为UI(user interface)接口的读写请求，通过读写通道发送给下一级。从UI接口收取的读数据，要经过重排序返回给AXI接口。

AddrMap模块适应不同bank/row/col地址组合解析SDRAM地址(rank/bank/bank group/row/column)。

Filter模块获取读写请求地址后，根据filter策略，一部分送入Advanced Scheduler的read queue和write queue，一部分送入System Cache的cache request queue。

System Cache模块实现系统级缓存，提升读写性能。

Advanced Scheduler模块实现读写请求的多bank调度，充分利用内存颗粒的开关行特性，减少访存请求的执行时间。

SDRAM Command Generator模块实现DFI接口的命令调度和拆分，匹配内存颗粒的访存时序要求，保证读写内存正确性。

各模块的详细设计见后续章节。

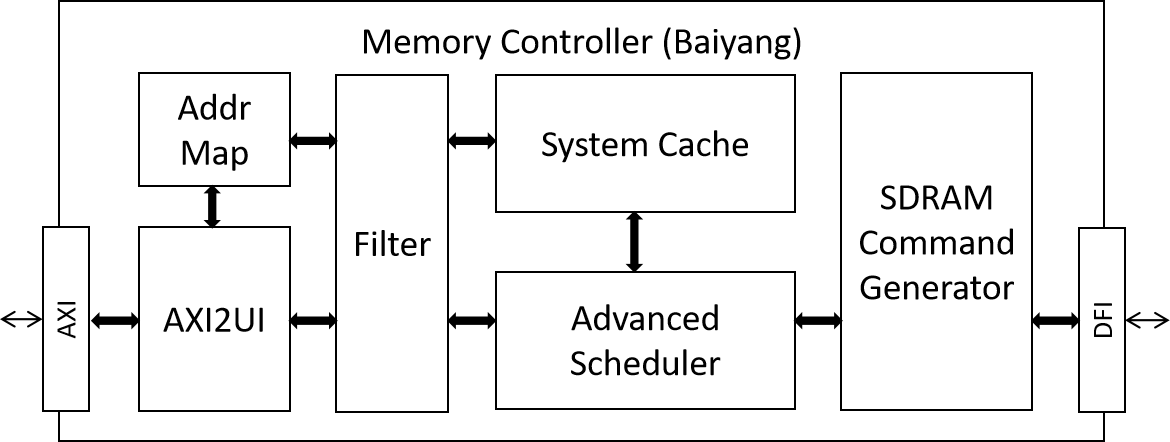


图2-1 Baiyang IP整体结构图

# AXI2UI设计说明

## 模块功能

AXI2UI模块内部结构图如下图所示。

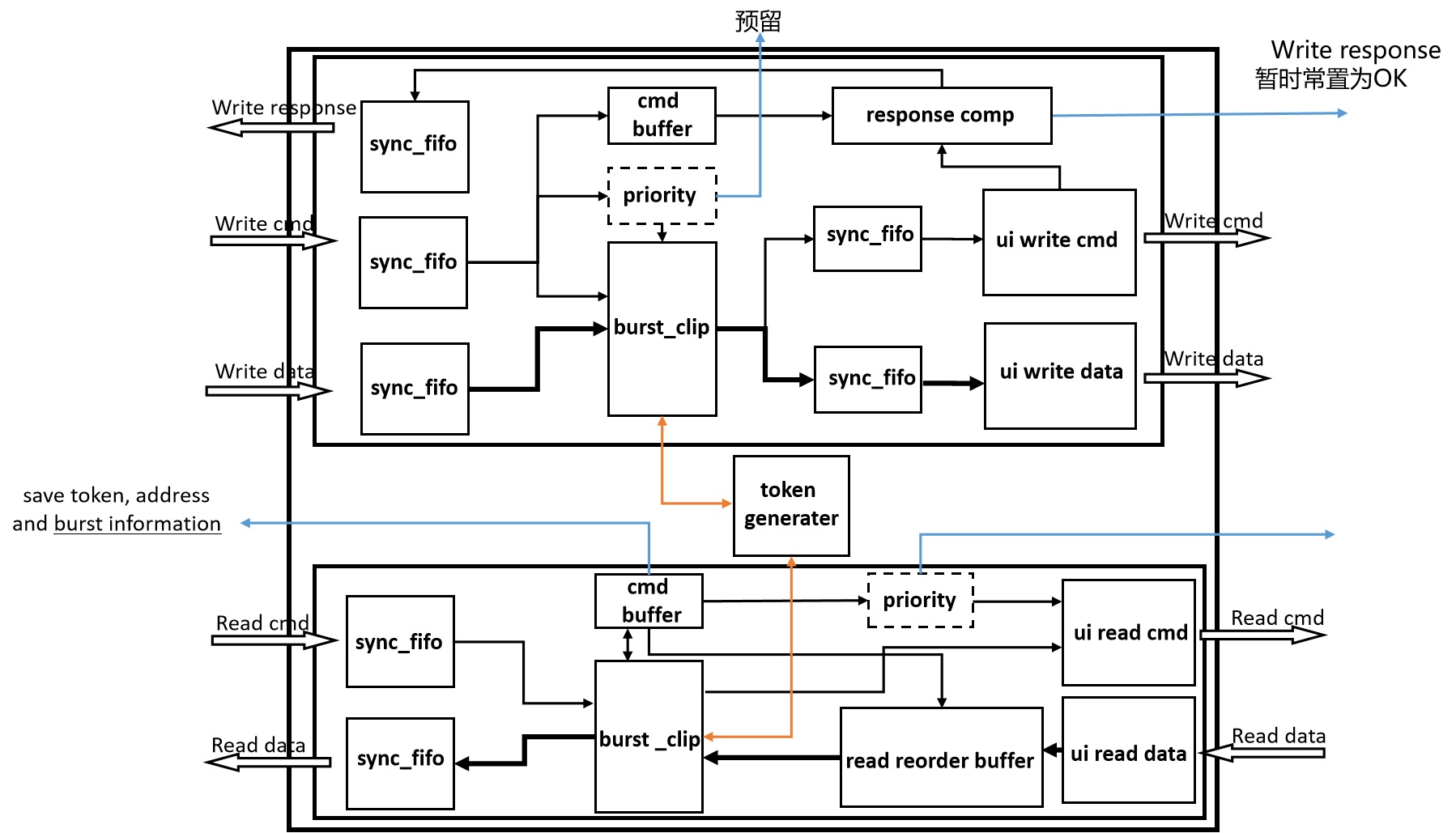


图3-1 AXI2UI模块内部结构图

AXI2UI的功能包括以下几点：

(1).读写Token的生成： Token用于标志命令的先后顺序与保持命令与数据的一致性。对于每个经过burst\_clip (无论读或写)裁剪完成后的命令与写数据，Token generator会按顺序为其打上Token。每一个写命令与对应写数据的Token值是相同的，且读写命令的Token共用一个计数队列，可用Token来辨别同一地址读写命令的先后顺序。

(2).AXI突发的裁剪：AXI突发裁剪在burst\_clip中进行，对于读写命令，burst\_clip根据AXI总线的data width (暂不支持带宽不满的AXI传输，且AXI总线data width小于UI总线的data width), 与awlen信号, UI总线的data width来将一个AXI cmd拆分为一个或多个UI cmd请求。

对于写数据，则将一次突发中的AXI data每寄存至相当于UI总线data width的数据量，打包成一个UI data。当遇到wlast信号时，则将现有寄存的AXI data直接打包，未满的数据位宽用掩码覆盖。对于读数据，则与写数据相反，将返回的UI data按AXI data width拆分为AXI data。拆分后的数据个数由cmd buff中缓存的命令数据决定，对于超出个数的数据，直接丢弃。

1. .Read Reorder Buffer： Read Reorder Buffer用于将DDRC内部返回的，乱序的读数据重排序为与AXI发送的读命令相同的顺序，再发送至burst\_clip中进行拆分。其主要结构与原理将在3.4 Read Reorder Buffer一节中介绍。

## 接口信号

AXI2UI和Filter模块之间存在信号互连，表 3‑1中列出了本模块的接口信号。

表3-1 AXI2UI接口信号列表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **方向** | **位宽** | **功能描述** |
| AXI | | | |
| **AXI write address channel** | | | |
| **awid** | input | [AXI\_IDW-1:0] | AXI write address ID |
| **awaddr** | input | [AXI\_ADDRW-1:0] | AXI write address |
| **awlen** | input | [AXI\_LENW-1:0] | AXI write burst length |
| **awsize** | input | [AXI\_SIZEW-1:0] | AXI write burst size |
| **awburst** | input | [AXI\_BURSTW-1:0] | AXI write burst type |
| **awuser** | input | [AXI\_USERW-1:0] | AXI write address User-defined signals（reserved） |
| **awqos** | input | [AXI\_QOSW-1:0] | AXI write address qos |
| **awvalid** | input | 1 | AXI write address valid |
| **awready** | output | 1 | AXI write address ready |
| **AXI write data channel** | | | |
| **wuser** | input | [AXI\_USERW-1:0] | AXI write data User-defined signals （reserved） |
| **wdata** | input | [AXI\_DATAW-1:0] | AXI write data |
| **wstrb** | input | [AXI\_STRBW-1:0] | AXI write strobes |
| **wlast** | input | 1 | AXI write last |
| **wvalid** | input | 1 | AXI write valid |
| **wready** | output | 1 | AXI write ready |
| **AXI write response channel** | | | |
| **bid** | output | [AXI\_IDW-1:0] | AXI write response ID |
| **bresp** | output | [AXI\_RESPW-1:0] | AXI write response |
| **buser** | output | [AXI\_USERW-1:0] | AXI write response User-defined signals （reserved） |
| **bvalid** | output | 1 | AXI write response valid |
| **bready** | input | 1 | AXI write response ready |
| **AXI read address channel** | | | |
| **arid** | input | [AXI\_IDW-1:0] | AXI read address ID |
| **araddr** | input | [AXI\_ADDRW-1:0] | AXI read address |
| **arlen** | input | [AXI\_LENW-1:0] | AXI read burst length |
| **arsize** | input | [AXI\_SIZEW-1:0] | AXI read burst size |
| **arburst** | input | [AXI\_BURSTW-1:0] | AXI read burst type |
| **aruser** | input | [AXI\_USERW-1:0] | AXI write address User-defined signals （reserved） |
| **arqos** | input | [AXI\_QOSW-1:0] | AXI read address qos |
| **arvalid** | input | 1 | AXI read address valid |
| **arready** | output | 1 | AXI read address ready |
| **AXI read data channel** | | | |
| **rid** | output | [AXI\_IDW-1:0] | AXI read ID |
| **rdata** | output | [AXI\_DATAW-1:0] | AXI read data |
| **rresp** | output | [AXI\_RESPW-1:0] | AXI read response |
| **ruser** | output | [AXI\_USERW-1:0] | AXI read data User-defined signals (no used) |
| **rlast** | output | 1 | AXI read last |
| **rvalid** | output | 1 | AXI read valid |
| **rready** | input | 1 | AXI read ready |
| **UI** | | | |
| **UI write cmd channel** | | | |
| **ui\_awaddr** | output | [UI\_ADDRW-1:0] | UI write command address |
| **ui\_awvalid** | output | 1 | UI write command valid |
| **ui\_awpri** | output | [UI\_PRIW-1:0] | UI write command valid priority |
| **ui\_awtoken** | output | [UI\_TOKENW-1:0] | UI write command token |
| **ui\_awlength** | output | [UI\_CMDW-1:0] | UI write command burst length (reserved) |
| **ui\_awready** | input | 1 | UI write command ready |
| **UI read cmd channel** | | | |
| **ui\_araddr** | output | [UI\_ADDRW-1:0] | UI read command address |
| **ui\_arvalid** | output | 1 | UI read command valid |
| **ui\_arpri** | output | [UI\_PRIW-1:0] | UIread command valid priority(reserved) |
| **ui\_artoken** | output | [UI\_TOKENW-1:0] | UI read command token |
| **ui\_arready** | input | 1 | UI read command ready |
| **UI read data channel** | | | |
| **ui\_rvalid** | input | 1 | UI read data valid |
| **ui\_rend** | input | 1 | UI read data end (reserved) |
| **ui\_rdata** |  | [UI\_DATAW-1:0] | UI read data |
| **ui\_rtoken** | input | [UI\_TOKENW-1:0] | UI read data token |
| **UI write data channel** | | | |
| **ui\_wready** | input | 1 | UI write data ready |
| **ui\_wdata** | output | [UI\_DATAW-1:0] | UI write data |
| **ui\_wstrab** | output | [UI\_STRBW-1:0] | UI write data mask |
| **ui\_wtoken** | output | [UI\_TOKENW-1:0] | UI write data token |
| **ui\_wvalid** | output | 1 | UI write data valid |

## 设计思路

本章节将介绍AXI2UI各模块的具体设计思路。AXI2UI主要包括以下几个模块：Token\_generator、Burst\_clip、Consistency、Read Cmd Buffer、Read Reorder Buffer、UI。

## Token\_generator

本模块设计思路以及功能实现如下：

Token generator会给每个从burst\_clip输出至FIFO的UI cmd与data打上token，一同送入UI之前的FIFO。

Token用于标志命令的先后顺序与保持命令与数据的一致性。对于同一地址的读写请求，DDRC在调度时具有保序的要求，故而读写请求共用一个计数队列。每当收到读请求或写请求时，token值加一，token的值为读请求与写请求之和。

当读写请求同时需求token时：按上述逻辑，读写请求对应的计数器都会加一，此时会导致两个token输出相同，即都为前一token + 2。因而，设计添加了一个简单的额外仲裁逻辑，当发生上述情况时，手动改变读请求token为加1。

防counter重置的额外设计：故而counter计数值设置为9 bit（暂定）位宽。在这一基础上，aw/w/ar\_counter额外设置了一个溢出信号，用于当溢出发生时防止计数器重置影响token值生成。该信号有效时token = 计数最大值 + 重置后计数值 + 另一对应计数器的值。

防token重置的额外设计：Token的位宽设置为counter + 1，其最高位为符号位，其余位为计数位。在DDRC中存在的最早的token值不会比最晚的token值早超过计数位最大值个数的计数。因而，在比较token先后时，当token符号位相同，计数位小的为更早生成的token；当符号位相反时，计数位大的为更早生成的token。

## Burst\_clip

本模块设计思路以及功能实现如下：

模块burst\_clip用于将AXI总线传输的事务格式转换为DDRC内部传输的事务格式。DDRC中写事务以一个cmd对应一个data的形式传输，读事务则单独以cmd的形式传输。其中cmd携带事务的地址、优先级、token等信息，而data则包括数据、掩码等信息。

写请求的burst\_clip模块主要可分为命令与数据两个通道。

对于读写命令通道，burst\_clip根据AXI总线的data width (暂不支持带宽不满的AXI传输，且AXI总线data width小于UI总线的data width), 与awlen信号, UI总线的data width来将一个AXI cmd拆分为一个或多个UI cmd请求。在read\_burst\_clip模块中，会将拆分之前的AXI cmd送入Read Cmd Buffer。

对于写数据通道，burst\_clip根据AXI总线的data width与UI总线的data width，将来自AXI总线的数据拼接成UI总线的数据。以128bit to 512bit为例，每收集4个来自AXI总线的数据，输出一个UI总线的数据。当本次传来的AXI总线数据带着有效的awlast信号时，则立刻输出包括本次数据在内，已收集的AXI总线数据，不满足UI总线data width的部分补0，并在ui\_strb信号中将补0的bytes对应位的拉高。

对于读数据通道，burst\_clip会根据Read Cmd Buffer的cmd fifo中记录的AXI cmd突发长度等信息，将从UI返回的读数据拆分位正确格式的AXI 读数据，并根据该信息为对应的AXI cmd打上rlast信号。

## 3.3.3 Consistency

本模块设计思路以及功能实现如下：

为了保证对同一地址读写事务的一致性，需要确保在被打上token之前，同一地址的读写事务顺序正确。因而在AXI端cmd进入模块后，会计算所得当前命令的起始与结束地址并存入到一个Consis Table中，读写通道各有一个Table。当一个通道有新的cmd到来时，先计算该cmd的首末地址，然后查找另一通道的Table（如，当读cmd到来时，查找写通道的Consis Table），判断是否发生冲突。

例如，此时AXI端进入了一个新的写命令，其首末地址分别为0x0000\_4000与0x0000\_4200，查找读通道的Consis Table，若存在起始地址或结束地址在0x0000\_4000与0x0000\_4200之间的cmd，则判断发生了冲突。此时，阻塞AXI端的写命令通道并挂起该写命令，直到burst\_clip模块中当前写命令完全下发

## 3.3.4 Read Cmd Buffer

本模块设计思路以及功能实现如下：

Read Cmd Buffer主要由两个FWFT(First-word Fall-Through)类型的FIFO：Token FIFO与Cmd FIFO组成。在read\_burst\_clip模块中，会将拆分之前的AXI cmd送入Cmd FIFO，将拆分后UI cmd的token送入Token FIFO。

Token FIFO将最先送入FIFO的token值展示给Read Reorder Buffer用于读数据的重排序。

而Cmd FIFO将最先送入FIFO的AXI cmd展示给read\_burst\_clip模块，用于将从UI返回的读数据拆分成正确格式的AXI读数据，再返还至AXI总线。

## 3.3.5 Read Reorder Buffer

本模块设计思路以及功能实现如下：

Read Reorder Buffer由两个memory与一个用于存储可用数据地址的队列(AAQ)，一个检索对应token是否返回的映射寄存器(ATK)组成。

AAR在上电或复位时初始化，从0至127(Data Memory最大存储空间)自动填充，并load可用计数器初始值。当UI中返回数据时，为其分配队列最前的可用地址Ad存入Data Memory中，计数值-1。同时将Token作为地址，Ad作为数据存入Token Addressed Memory (TAM)中，并占用对应bit的token的寄存器(ATR)。

需要注意的是，当计数至0时，说明内存控制器发生了预期之外的错误，最先进入的读命令始终没能返回读数据，导致RROB卡死。此时，将Token FIFO展示的最先读token丢弃，并向AXI返回read error信号。

Cmd Buffer中的Token FIFO会将一直需要取出的数据token展示给TAM，并映射到ATR中。当ATR检索到Token FIFO所给token对应的数据已从UI返回，发送指示到FIFO取出该token并释放该位寄存器。此时，TAM以token为地址取出Ad，并以Ad从Data Memory中取出数据。同时Ad返回至AAQ队列最末端。

Read Reorder Buffer (以下简称RROB)的整体结构如图3-2所示。

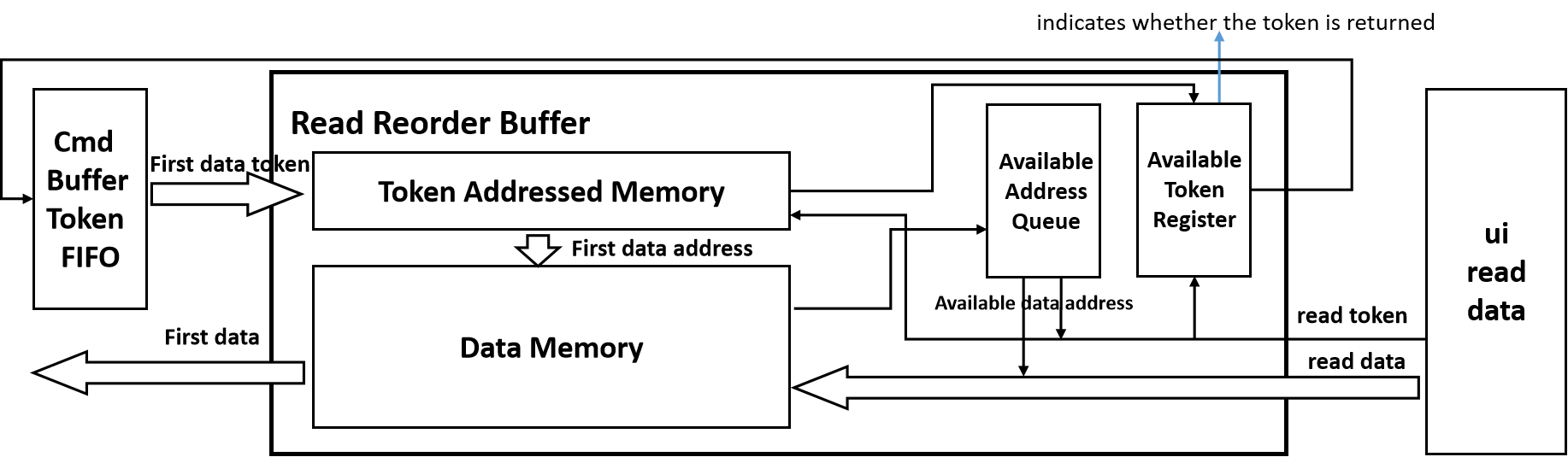


图3-2 Read Reorder Buffer结构图

## 3.3.6 UI

本模块设计思路以及功能实现如下：

UI (User Interface)将读写事务以特定的格式送入DDRC中。当RROB中所展示的最小的token值与token generator中生成的token值相差达到token计数位的最大值时，UI接口将被阻塞，直至最早读命令对应的读数据返还，并从RROB中将token取出。

## 3.3.7 优化点

1. 基于当前同一地址的cmd读写保序的情况下，Token与RROB的优化问题：当前通路，同一地址的读写命令下发时保序，遇到冲突无需用到token来进行先后顺序的判断，可以考虑将读写token分别计数，则可用节省RROB相连的CMD buffer中读token存储区域的资源消耗，改为使用计数器计数。
2. 避免AXI2UI与FILTER中都需要对冲突进行查找的优化方案：将Token分为父Token与子Token两部分。在AXI端的cmd进入模块时，便先打上父Token，以识别先后顺序。在cmd拆分后，打上子Token以确保同一读cmd在拆分，并进行乱序调度后，在返回至RROB时可以正确重排序。此外，在filter端cmd进入AS(Advanced Scheduler)时，将读写通道合并为一个通道，将其中命令按token顺序下发，即可避免AXI2UI到FILTER这一路径中对冲突地址进行查找操作。

# Filter设计说明

## 模块功能

在获取读写请求地址后先进行同步，然后根据filter策略，一部分送入Advanced Scheduler的read queue和write queue，一部分送入cache request queue。支持Bypassa AS 和 SC。

Filter在Baiyang IP中的功能包括以下几点：

（1）.可通过外部配置不同的功能，可配置选项有分流策略和，分流地址范围。

可通过mode接口进行功能配置：0（bypass cache），1（bypass schedule），2（split分流功能）。

分流地址范围由addr\_boundary接口决定，在范围内的分流至SC。

addr\_boundary[ADDR\_WIDTH\*2-1:ADDR\_WIDTH]决定分流的高位地址。 addr\_boundary[ADDR\_WIDTH-1:0]决定分流的低位地址。

（2）.wcache\_en、rcache\_en信号指示是否将读写命令数据分流至SC，高电平有效。

## 接口信号

Filter和两个模块之间存在信号互连，分别是UI模块和Addr Map模块。表4-1中列出了本模块的接口信号。

表4-1 Filter接口信号列表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **方向** | **位宽** | **功能描述** |
| **Global** | | | |
| **clk** | input | 1 | 时钟 |
| **rst\_n** | input | 1 | 同步复位信号 |
| **en** | input | 1 | 使能信号 |
| **mode** | input | 2 | 模式控制 |
| **addr\_boundary** | input | ADDR\_WIDTH\*2 | 分流至SC的地址范围 |
| **Filter2UI** | | | |
| **wcmd** | Input | ADDR\_WIDTH | 写命令地址 |
| **wdata** | input | DATA\_WIDTH | 写数据 |
| **rcmd** | input | ADDR\_WIDTH | 读命令地址 |
| **rui\_data** | output | DATA\_WIDTH | 读回来的数据 |
| **Filter2Addr** | | | |
| **wcache\_en** | output | 1 | 写相关数据是否分流至cache |
| **rcache\_en** | output | 1 | 读相关数据是否分流至cache |
| **raddr\_data** | Input | DATA\_WIDTH | 返回的读数据 |
| **wcmd\_addrmap** | output | ADDR\_WIDTH | 同步后的写命令 |
| **waddr\_data** | output | DATA\_WIDTH | 同步后的写数据 |
| **rcmd\_addrmap** | output | ADDR\_WIDTH | 同步后的的读命令 |

## 设计思路

本章节将介绍Filter模块的具体设计思路。Filter模块的整体架构图如图4-1：

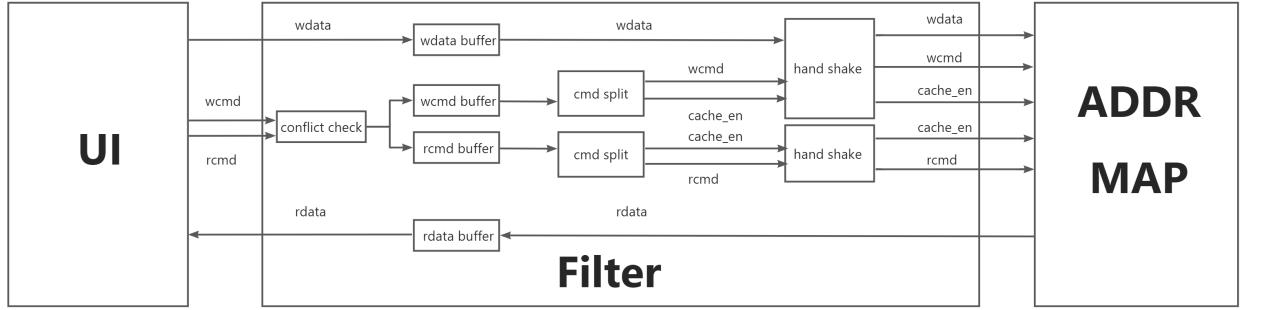


图4-1 Filter模块整体架构图

## 功能实现

在获取读写请求地址后先进行同步，然后根据filter策略，一部分送入Advanced Scheduler的read queue和write queue，一部分送入cache request queue。支持Bypass Advanced Scheduler模块 和 System Cache模块。

1、可通过外部配置不同的功能，可配置选项有分流策略和，分流地址范围。

可通过mode接口进行功能配置：0（bypass cache），1（bypass scheduler），2（split分流功能）。

2、分流地址范围由addr\_boundary接口决定，在范围内的分流至SC。

addr\_boundary[ADDR\_WIDTH\*2-1:ADDR\_WIDTH]决定分流的高位地址。 addr\_boundary[ADDR\_WIDTH-1:0]决定分流的低位地址。

因此分流至SC的地址范围为addr\_boundary[ADDR\_WIDTH\*2-1:0]

1. wcache\_en、rcache\_en信号指示是否将读写命令数据分流至System Cache，高电平有效。

## 详细设计

Filter模块分为读写两个通道，引入了命令冲突的情况，因此在进入fifo之前需要进行冲突查询，若有冲突地址则不将ready置高，待fifo中无冲突项的时候，才会进行握手操作。

冲突的定义是指：同地址的读写命令顺序错乱。假设输入filter的一个命令地址为A，这个地址的读命令AR先输入filter，存入读命令fifo当中，后续这个地址的写命令AW再输入filter，就要和读命令fifo当中的命令进行冲突查询，查询到读命令fifo当中有AR，那就需要把读命令fifo清空，保证AW是在AR之后输出，就像当时输入的顺序一样。同理，如果先输入filter的是写命令，那写命令存入写命令fifo中，后面读命令输入filter，则需要和写命令fifo进行冲突查询。

每个通道独立运行，数据和命令可能不会同时到达，需要对其做同步后才能输出到下一个模块，因此需要FIFO去进行数据的同步，当数据和命令FIFO中都有数据的时候也就是!empty信号有效的时候才会进行数据输出，此时根据命令FIFO的值和mode信号来确定分流到哪一个模块。

接收的读数据先用FIFO进行缓存，若UI准备好就可进行输出。

## 优化点

1、可以将分流地址分为多个区域进行映射。

2、优化冲突查询机制，在检测完冲突的时候可以将冲突项直接输出。

# Addr Map设计说明

## 模块功能

Addr Map在Baiyang IP中的功能包括以下几点：Addr Map主要实现地址解析功能。将输入的读写命令地址映射为SDRAM物理地址(rank, bank,bankgroup, row, column)，与Filter、Cache、AS分别有读写命令通道和读写数据通道进行交互，根据WrIsToAS/RdIsToAS指示将拆分后的命令与数据转发给CaChe/As，同时保留MEM\_ADDR\_MAP参数实现用户灵活配置。

Addr Map整体结构如图5-1所示。

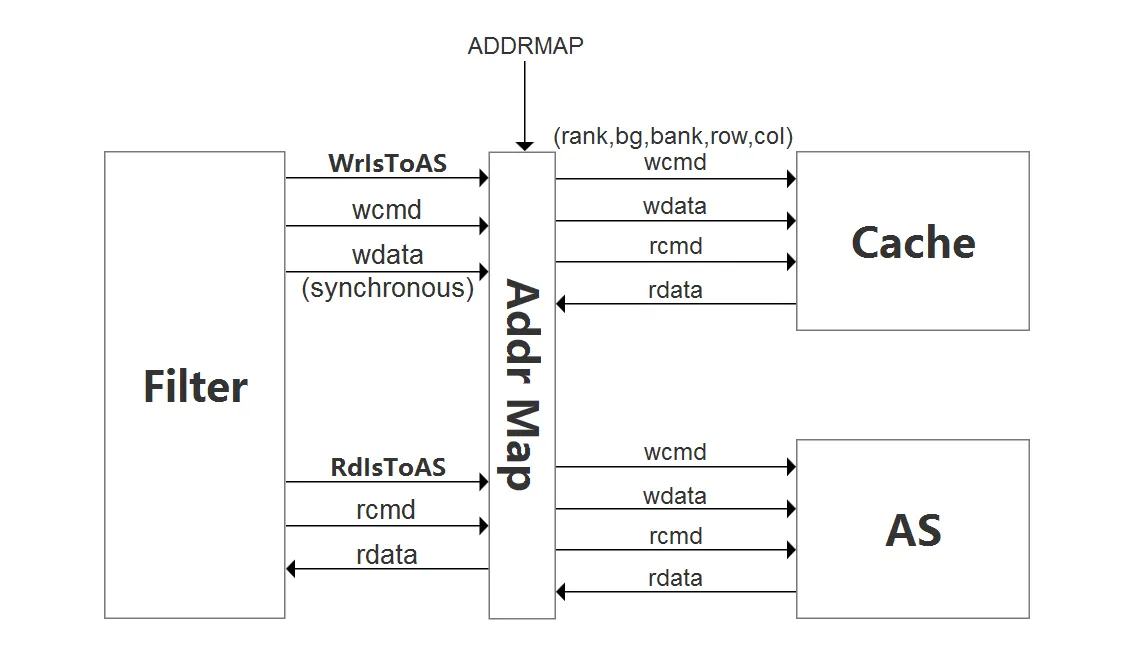


图5-1 Addr Map整体结构图

## 接口信号

Addr Map和3个模块之间存在信号互连，分别是Filter模块、Cache模块和AS模块。表5-1中列出了本模块的接口信号。

表5-1 Addr Map接口信号列表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **方向** | **位宽** | **功能描述** |
| **Global** | | | |
| **clk** | input | 1 | 全局时钟 |
| **rst** | input | 1 | 同步复位信号，高电平有效 |
| **Filter Interface** | | | |
| **WrIsToAs** | input | 1 | from Filter，高电平时表明写请求发给AS |
| **RdIsToAS** | input | 1 | from Filter，高电平时表明读请求发给AS |
| write cmd channel | | | |
| **awaddr** | input | ADDR\_WIDTH | from Filter，写命令地址 |
| **awvalid** | input | 1 | from Filter，写命令有效 |
| **awpri** | input | PRI\_WIDTH | from Filter，ui\_wvalid高电平时有效，优先级信号 |
| **awtoken** | input | TOKEN\_WIDTH | from Filter，写命令token标签 |
| **awlength** | input | CMD\_WIDTH | reserved |
| **awready** | output | 1 | to Filter，写命令反压信号 |
| write data channel | | | |
| **wready** | output | 1 | to Filter，写数据反压信号 |
| **wend** | inuput | 1 | reserved |
| **wdata** | input | DATA\_WIDTH | from Filter，写数据，与写地址同步 |
| **wstrab** | input | STRAB\_WIDTH | from Filter，写数据掩码 |
| **wtoken** | input | TOKEN\_WIDTH | from Filter，写数据掩码 |
| **wvalid** | input | 1 | from Filter，写数据有效 |
| read cmd channel | | | |
| **araddr** | input | ADDR\_WIDTH | from AXI2UI，读命令地址 |
| **arvalid** | input | 1 | from AXI2UI，读命令有效 |
| **arpri** | input | PRI\_WIDTH | from AXI2UI，ui\_rvalid高电平时有效，优先级信号 |
| **artoken** | input | TOKEN\_WIDTH | form AXI2UI，读命令token标签 |
| **arlength** | input | CMD\_WIDTH | reserved |
| **arready** | output | 1 | to AXI2UI，读命令反压信号 |
| read data channel | | | |
| **rvalid** | output | 1 | to Filter，读数据有效 |
| **rready** | input | 1 | from Filter |
| **rdata** | output | DATA\_WIDTH | to Filter，读数据 |
| **rtoken** | output | TOKEN\_WIDTH | to Filter，读数据token |
| **Cache Interface** | | | |
| write cmd channel | | | |
| **am\_wrank** | output | RANK\_WIDTH | to Filter，写rank地址 |
| **am\_wbg** | output | BG\_WIDTH | to Filter，写bank group地址 |
| **am\_wbank** | output | BANK\_WIDTH | to Filter，写bank地址 |
| **am\_wrow** | output | ROW\_WIDTH | to Filter，写row地址 |
| **am\_wcolumn** | output | COL\_WIDTH | to Filter，写column地址 |
| **am\_wpri** | output | PRI\_WIDTH | to Filter，am\_wvalid高电平有效 |
| **am\_wtoken** | output | TOKEN\_WIDTH | to Filter，写命令token标签 |
| **am\_wlength** | output | CMD\_WIDTH | reserved |
| **am\_wvalid** | output | 1 | to Filter，写命令有效 |
| **am\_wready** | input | 1 | from Filter，写命令反压信号 |
| write data channel | | | |
| 参考与Filter接口中的write data channel | | | |
| read cmd channel | | | |
| **am\_rrank** | output | RANK\_WIDTH | to Filter，读rank地址 |
| **am\_rbg** | output | BG\_WIDTH | to Filter，读bank group地址 |
| **am\_rbank** | output | BANK\_WIDTH | to Filter，读bank地址 |
| **am\_rrow** | output | ROW\_WIDTH | to Filter，读row地址 |
| **am\_rcolumn** | output | COL\_WIDTH | to Filter，读column地址 |
| **am\_rpri** | output | PRI\_WIDTH | to Filter，am\_rvalid高电平有效 |
| **am\_rtoken** | output | TOKEN\_WIDTH | to Filter，读命令token标签 |
| **am\_rlength** | output | CMD\_WIDTH | reserved |
| **am\_rvalid** | output | 1 | to Filter，读命令有效 |
| **am\_rready** | input | 1 | from Filter，读命令反压信号 |
| read data channel | | | |
| 参考与Filter接口中的read data channel | | | |
| **AS Interface** | | | |
| write cmd channel、write data channel | | | |
| read cmd channel、read data channel | | | |
| 参考Cache Interface | | | |
| **Parameters** | | | |
| MEM\_ADDR\_MAP | 3bits，可配置地址映射规则  -- 000，DDR4“ROW\_COLUMN\_BANK”Mapping（default）  -- 001，DDR4“BANK\_ROW\_COLUMN”Mapping  -- 010，DDR4“ROW\_BANK\_COLUMN”Mapping  -- others reserved | | |
| RANK\_WIDTH | rank位宽 | | |
| RANKS | rank个数 | | |
| ADDR\_WIDTH | 32 | | |
| PRI\_WIDTH | 1 | | |
| TOKEN\_WIDTH | 10 | | |
| CMD\_WIDTH | 3 | | |
| BG\_WIDTH | 2 | | |
| BANK\_WIDTH | 2 | | |
| ROW\_WIDTH | 15 | | |
| COL\_WIDTH | 10 | | |
| DATA\_WIDTH | 512 | | |
| STRAB\_WIDTH | 64 | | |

## 设计思路

本章节将介绍Addr Map各模块的具体设计思路。

## 静态映射

MEM\_ADDR\_MAP参数可选地址映射规则如下：

表5-2 DDR4“ROW\_COLUMN\_BANK”Mapping

|  |  |
| --- | --- |
| **SDRAM** | **ui\_addr Mapping** |
| Rank | -- singel rank |
| -- multiple rank |
| ui\_addr[COL\_WIDTH + ROW\_WIDTH + BANK\_WIDTH + BG\_WIDTH +: RANK\_WIDTH] |
| Row | ui\_addr[BANK\_WIDTH + BG\_WIDTH + COL\_WIDTH +: ROW\_WIDTH] |
| Column | ui\_addr[3 + BANK\_WIDTH + BG\_WIDTH +: COL\_WIDTH-3], ui\_addr[2:0] (addr最小步长为8) |
| Bank | ui\_addr[3 + BG\_WIDTH +: BANK\_WIDTH] |
| Bank Group | ui\_addr[3 +: BG\_WIDTH] |

表5-3 DDR4“BANK\_ROW\_COLUMN”Mapping

|  |  |
| --- | --- |
| SDRAM | ui\_addr Mapping |
| Rank | -- singel rank |
| -- multiple rank |
| ui\_addr[COL\_WIDTH + ROW\_WIDTH + BANK\_WIDTH + BG\_WIDTH +: RANK\_WIDTH] |
| Row | ui\_addr[COL\_WIDTH +: ROW\_WIDTH] |
| Column | ui\_addr[0 +: COL\_WIDTH] |
| Bank Group | ui\_addr[COL\_WIDTH + ROW\_WIDTH + BANK\_WIDTH +: BG\_WIDTH] |
| Bank | ui\_addr[COL\_WIDTH + ROW\_WIDTH +: BANK\_WIDTH] |

表5-4 DDR4“ROW\_BANK\_COLUMN”Mapping

|  |  |
| --- | --- |
| SDRAM | ui\_addr Mapping |
| Rank | -- singel rank |
| -- multiple rank |
| ui\_addr[COL\_WIDTH + ROW\_WIDTH + BANK\_WIDTH + BG\_WIDTH +: RANK\_WIDTH] |
| Row | ui\_addr[BANK\_WIDTH + BG\_WIDTH + COL\_WIDTH +: ROW\_WIDTH] |
| Column | ui\_addr[0 +: COL\_WIDTH] |
| Bank Group | ui\_addr[COL\_WIDTH + BANK\_WIDTH +: BG\_WIDTH] |
| Bank | ui\_addr[COL\_WIDTH +: BANK\_WIDTH] |

## 优化点

1. 可以增加动态映射的拓展功能
2. 参考Synopsys设计，通过base+offset方式，可部分映射到任意位置

# Advanced Scheduler设计说明

## Advanced Scheduler模块功能

Advanced Scheduler在Baiyang IP中的功能包括以下几点：

1、从Filter和Cache分别接收读/写命令，放入相应的队列中。

2、将调度选中的读/写命令发送到SDRAM Command Generator。

3、管理开行信息,从SCG获取开行信息。

4、接收Scg模块的resp并将resp返回到Filter和Cache模块。

5、APB配置接口。

其接口框图如图6-1所示

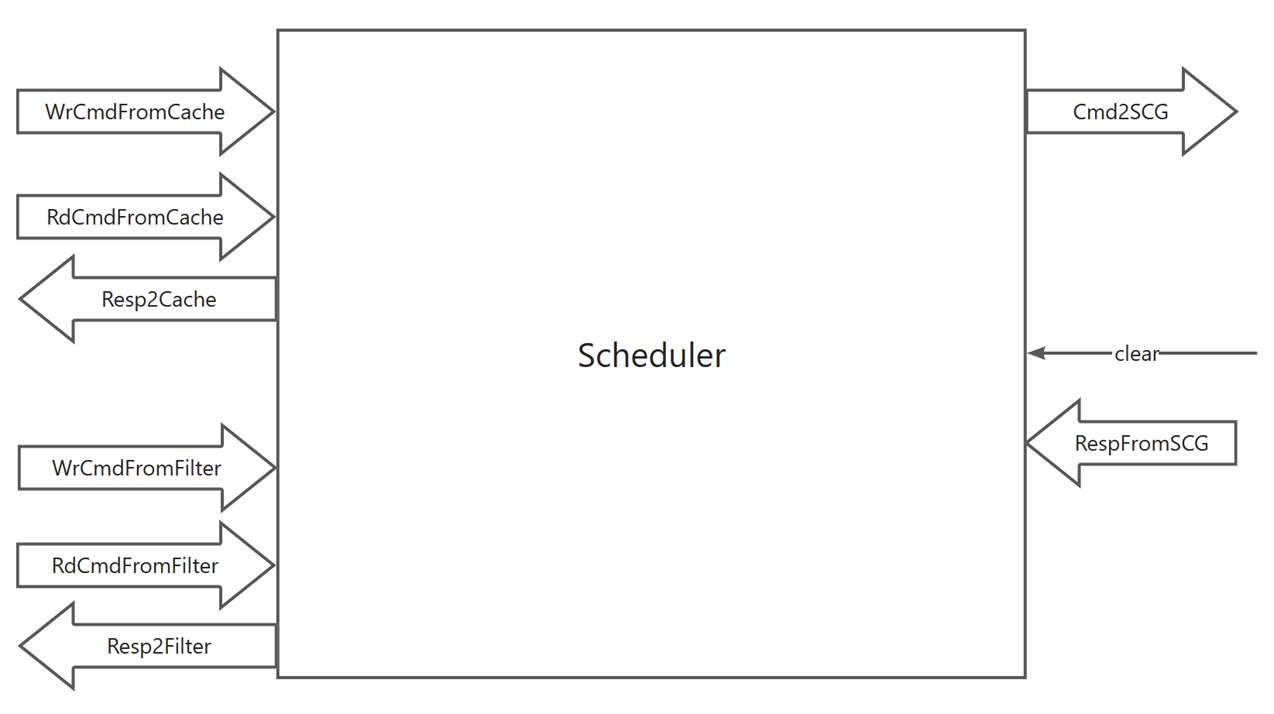


图6-1 Advanced Scheduler接口框图

## 接口信号

Advanced Scheduler和3个模块之间存在信号互连，分别是SCG模块、Filter模块、Cache模块，表6-1中列出了该模块使用的信号。

表6-1 Advanced Scheduler接口信号列表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **方向** | **位宽** | **功能描述** |
| **Global** | | | |
| **clk** | input | 1 | 全局时钟 |
| **rst** | input | 1 | 同步复位信号，高电平有效 |
| **Cmd2Scg** | | | |
| **col** | output | 10 |  |
| **group** | output | 2 |  |
| **rank** | output | 1 |  |
| **cmd** | output | 1 | 1读/0写命令 |
| **bank** | output | 2 | bank地址 |
| **row** | output | 18 |  |
| **token** | output | tokenWidth |  |
| **priority** | output | 1 |  |
| **wrData** | output | dataWidth |  |
| **cmdRdy** | input | 1 |  |
| **cmdValid** | output | 1 |  |
| WrCmdFromFilter | | | |
| **ready** | output | 1 |  |
| **valid** | input | 1 |  |
| **addr** | input | addrWidth |  |
| **token** | input | tokenWidth |  |
| **priority** | input | 1 |  |
| **data** | input | dataWidth |  |
| **RdCmdFromFilter** | | | |
| **ready** | output | 1 |  |
| **valid** | input | 1 |  |
| **addr** | input | addrWidth |  |
| **token** | input | tokenWidth |  |
| **priority** | input | 1 |  |
| **WrCmdFromCache** | | | |
| **ready** | output | 1 |  |
| **valid** | input | 1 |  |
| **addr** | input | addrWidth |  |
| **token** | input | tokenWidth |  |
| **priority** | input | 1 |  |
| **data** | input | dataWidth |  |
| **RdCmdFromCache** | | | |
| **ready** | output | 1 |  |
| **valid** | input | 1 |  |
| **addr** | input | addrWidth |  |
| **token** | input | tokenWidth |  |
| **priority** | input | 1 |  |
| **RespFromScg** | | | |
| **ready** | output | 1 |  |
| **valid** | input | 1 |  |
| **rdData** | input | dataWidth |  |
| **token** | input | tokenWidth |  |
| **ready** | output | 1 |  |
| **Resp2Filter** | | | |
| **ready** | input | 1 |  |
| **valid** | output | 1 |  |
| **rdData** | output | dataWidth |  |
| **token** | output | tokenWidth |  |
| **Resp2Cache** | | | |
| **ready** | input | 1 |  |
| **valid** | output | 1 |  |
| **rdData** | output | dataWidth |  |
| **token** | output | tokenWidth |  |
| **clear** | input | 1 | 清除所有bankState |

## 设计思路

本章节将介绍Advanced Scheduler各模块的具体设计思路。该模块的整体框图如图6-1：

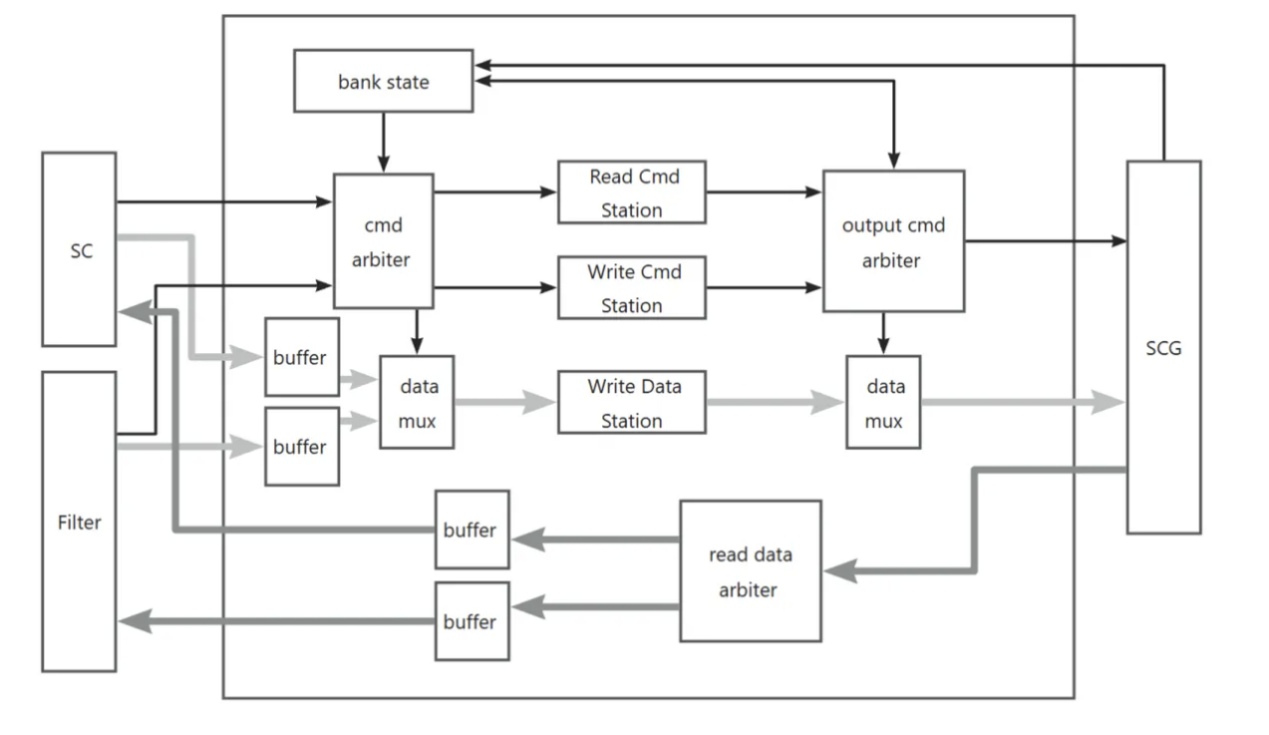


图6-1 Advanced Scheduler模块整体架构图

## 两个源的读写仲裁

本模块主要功能包括以下几点：

* (Cache, Filter)的读写cmd需要仲裁，可以先固定优先级，但是要解决饥饿问题。
* 分bank存储命令有个好处，如果两个源的bank地址不冲突，就可以同时存入。

## CMDStation

本模块主要功能包括以下几点：

分bank group存cmd，替代CAM寻址，4个bank group，暂定每个station里面存16项。结构是随机存取。读写分开存，好处是可以控制全局读写状态。

维护一个bank state，里面保存的是每个bank打开的row。与cmd记录中的各项比较，如果row命中，该项输出一个hit信号(与RS类似，(bank,row)相当于操作数)。

读/写状态切换：定为读优先，设置计时器，计时器超时切换到写状态。

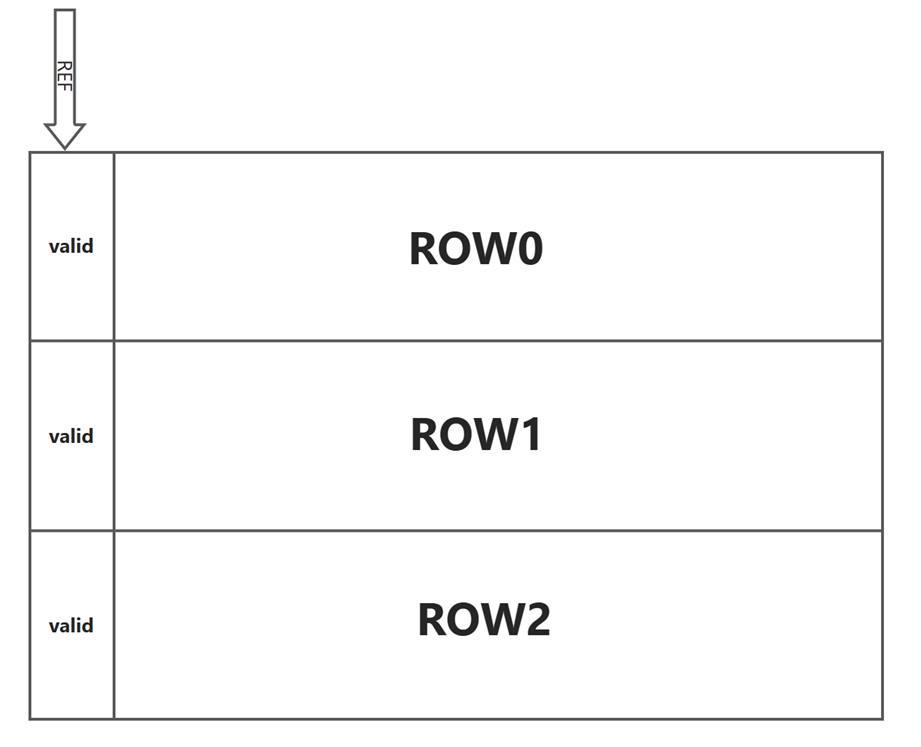
Cmd station维护的状态表如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| valid | bank | row | col | priority | token | hit |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

## bank state

Bank state记录每个bank目前打开的行，当前选择隐式管理，即在没有行打开时将调度选中的行认为是打开的行。

可以在SCG发出刷新命令时清除所有bank的打开行信息。



## 读写冲突

命令在进入对应的station前查询冲突，如果是读/读冲突不需要做处理，如果是读/写,写/读,写/写冲突就立即发出冲突的命令。

## 数据处理，token处理

可以记录冲突前bank state中的row地址，用于下次开行。

写命令与写数据怎么匹配，暂定Filter做命令与数据的同步。

往SCG模块发token时打上一位的标记来标记来源(cache, filter)，收到返回的数据时根据这一位返回给cache或者filter。

## Schedule logic

该模块实现了命令调度的功能，主要包括读写、命中、冲突的调度。根据命中的情况以及冲突等信号进行命令的调度。

该模块分别由conflict\_station、choose\_innerns、rwswitch\_logic、conflict\_logic等模块组成。其中conflict\_station是选出其中的冲突的station，choose\_innerns是选出每个station其中的项，rwswitch\_logic控制其的读写切换，conflict\_logic控制其处理冲突的发生。模块框图如图6-2所示：

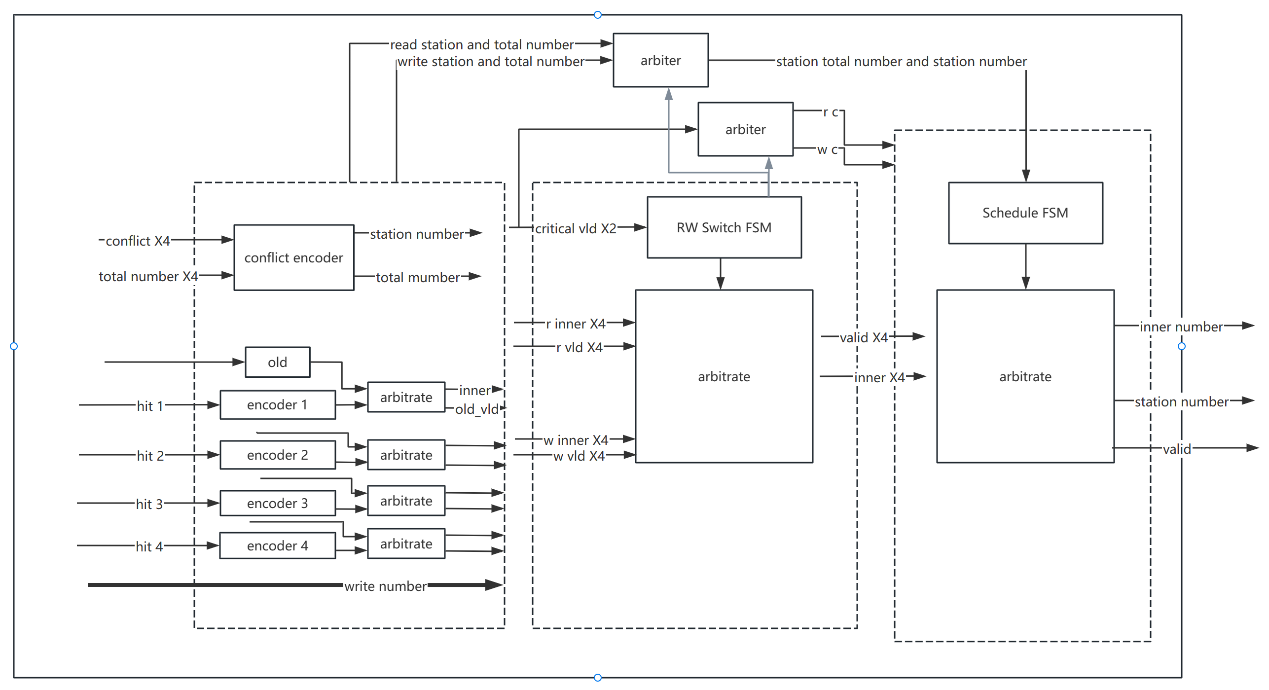


图6-2 Schedule logic 模块框架图

该模块使用接口信号如表6-2所示。

表6-2 Scheduler logic接口信号列表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **方向** | **位宽** | **功能描述** |
| **write\_timeout\_relode\_number** | input | WRITE\_TIMEOUT\_CNT\_WIDTH | 写饿死所花费的周期数 |
| **write\_clk\_relode\_number** | input | WRITE\_TIMEOUT\_CNT\_WIDTH | 饿死后需要保持在写状态的周期数 |
| **stall** | input | 1 | 阻塞信号 |
| **read\_old\_vld** | input | 1\*GROUP\_NUMBER | 每个station提供的old\_prefer是否有效 4X1.W |
| **read\_hit\_signal** | input | HIT\_WIDTH\*GROUP\_NUMBER | 每个station命中信号 4X16.W |
| **read\_old\_prefer** | input | OLD\_WIDTH\*GROUP\_NUMBER | 每个station最老的位置 4X4.W |
| **read\_conflict\_signal** | input | CSIGNAL\_WIDTH | 读冲突信号[1,2,3,4]4个station冲突信号拼接而成 4.W |
| **read\_station\_total\_number4** | input | TOTAL\_NUMBER\_WIDTH\*GROUP\_NUMBER | 每个station存在的命令数 4.W |
| **write\_old\_vld** | input | 1\*GROUP\_NUMBER | 每个station提供的old\_prefer是否有效 4X1.W |
| **write\_hit\_signal** | input | HIT\_WIDTH\*GROUP\_NUMBER | 每个station命中信号 4X16.W |
| **write\_old\_prefer** | input | OLD\_WIDTH\*GROUP\_NUMBER | 每个station最老的位置 4X4.W |
| **write\_conflict\_signal** | input | CSIGNAL\_WIDTH | 写冲突信号[1,2,3,4]4个station冲突信号拼接而成 4.W |
| **write\_station\_total\_number4** | input | TOTAL\_NUMBER\_WIDTH\*GROUP\_NUMBER | 每个station存在的命令数 4.W |
| **final\_inner\_number** | output | INNER\_NUMBER\_WIDTH | 需要调度的命令内部所在位置 |
| **final\_station\_number** | output | INNER\_NUMBER\_WIDTH | 所在的station |
| **outvalid** | output | 1 | 是否有效 |
| **write\_old\_prefer** | input | OLD\_WIDTH\*GROUP\_NUMBER | 每个station最老的位置 4X4.W |
| **write\_conflict\_signal** | input | CSIGNAL\_WIDTH | 写冲突信号[1,2,3,4]4个station冲突信号拼接而成 4.W |
| **write\_station\_total\_number4** | input | TOTAL\_NUMBER\_WIDTH\*GROUP\_NUMBER | 每个station存在的命令数 4.W |
| **final\_inner\_number** | output | INNER\_NUMBER\_WIDTH | 需要调度的命令内部所在位置 |

该模块的详细设计如下：

schedule按照bankgroup分为4个station，每个station都有一个conflict信号来指示是否发生冲突，并且提供此station的命中情况和里边的命令总数，以及其中最老的命令的地址。

conflict\_station：接收4个station发送过来的conflict拼接信号，按照低位优先的原则，找出其中第一个为1的station号和其中这个station中的命令的项数。

choose\_innerns：4个station每个分别提供1项，优先选出其中的老的命中项，其次是命中，最后是老的命令，也就是优先级为：hit&old ＞ hit ＞ old。

rwswitch\_logic：分为idle、read、write三个状态。当station全空的时候为idle状态；读写逻辑的切换，冲突和写timeout这三种情况都会引发读写切换；设置读冲突的优先级高于写冲突，也就是读写冲突同时发生的时候优先切换到读状态来优先处理读冲突，之后再切换到写状态来处理写冲突，而且当切换到写状态并且存在冲突的情况下不能被读状态抢占。

conflict\_logic：分为idle、normal、conflict三个状态。当station全空或被阻塞的时候为idle状态，当有冲突的时候跳转到conflict状态处理冲突，在冲突发生的时候conflict\_station根据rwswitich\_logic模块的状态提供读和写冲突station的total\_number，启动计数器当计数为0时冲突解除，此时跳转到normal状态等待再次被抢占。在normal状态时命令的发送通过roll\_cnt进行轮询发送。

# SDRAM Command Generator设计说明

## 模块功能

SDRAM Command Generator在Baiyang IP中的功能包括以下内容：

SDRAM Command Generator模块来自AS模块的命令以及数据、token，写数据和掩码根据BG存入不同的FIFO当中等待被读出；命令和数据经过MIG的调度产生一些列请求和与请求对应的地址，通过产生的写请求来将FIFO中的数据读出，同时命令被转换成DFI接口类型的命令并进行位宽匹配，再同步的向Command Disassembly模块发送。CD模块接收命令和数据之后进行判断，完成DFI协议要求的时序控制。

SDRAM Command Generator模块基本框图如下：

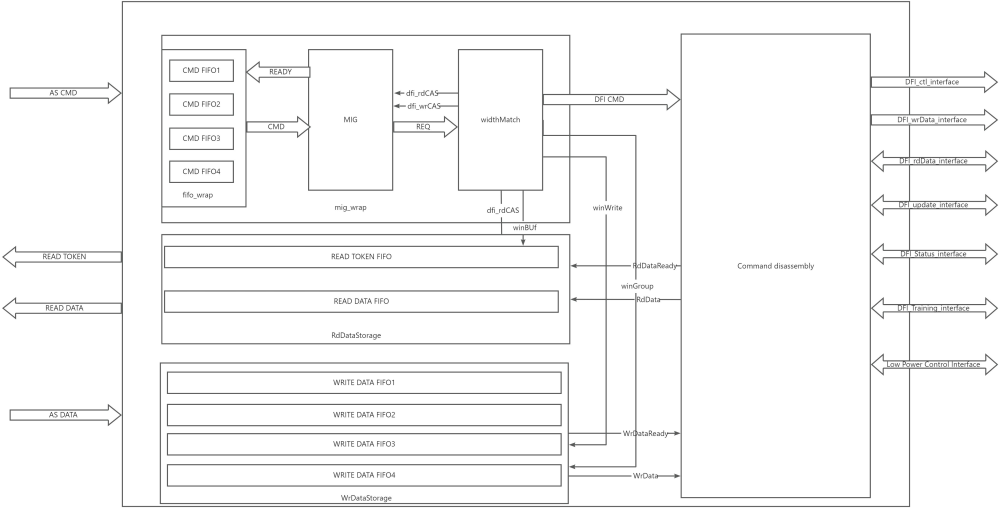


图7-1 SDRAM Command Generator模块架构图

## 接口信号

SDRAM Command Generator和AS之间存在信号互连，7-1中列出了本模块的接口信号。

表7-1 SDRAM Command Generator接口信号列表

|  |  |  |  |
| --- | --- | --- | --- |
| **端口** | **方向** | **位宽** | **功能描述** |
| **DFI interface** | | | |
| **control interface** | | | |
| **dfi\_address** | output | 36 | DFI地址总线 |
| **dfi\_bank** | output | 4 | DFIbank总线 |
| **dfi\_ras\_n** | output | 2 | DFI行地址选通 |
| **dfi\_cas\_n** | output | 2 | DFI列地址选通 |
| **dfi\_we\_n** | output | 2 | DFI写使能信号 |
| **dfi\_cs\_n** | output | 4 | DFI片选信号 |
| **dfi\_act\_n** | output | 2 | DFI激活信号 |
| **dfi\_bg** | output | 4 | DFI bank group |
| **dfi\_cid** | output | 1 | DFI Chip ID |
| **dfi\_cke** | output | 2 | DFI时钟使能 |
| **dfi\_odt** | output | 2 | DFI on-die termination control bus |
| **dfi\_reset\_n** | output | 2 | DFI复位总线 |
| **Write Data interface** | | | |
| **dfi\_wrdata\_en** | output | 16 | dfi\_wrdata ，dfi\_wrdata\_mask的使能信号 |
| **dfi\_wrdata** | output | 256 | dfi写数据总线 |
| **dfi\_wrdata\_cs\_n** | output | 32 | dfi写数据片选信号 |
| **dfi\_wrdata\_mask** | output | 32 | dfi写数据掩码 |
| **Read Data interface** | | | |
| **dfi\_rddata\_en** | output | 16 | dfi读数据掩码 |
| **dfi\_rddata** | input | 256 | dfi读数据总线 |
| **dfi\_rddata\_cs\_n** | output | 32 | dfi读数据片选信号 |
| **dfi\_rddata\_valid** | input | 16 | dfi读数据有效信号 |
| **dfi\_rddata\_dbi\_n** | input | 32 | dfi读数据DBI,只有在phydbi\_mode = 0 时使用 |
| **Update Interface** | | | |
| **dfi\_ctrlupd\_req** | output | 1 | mc通过该信号向PHY发起更新请求 |
| **dfi\_ctrlupd\_ack** | input | 1 | PHY对更新请求的回应 |
| **dfi\_phyupd\_req** | input | 1 | PHY向MC发起的更新请求 |
| **dfi\_phyupd\_type** | input | 2 | DFI支持最多4中更新模式，每种模式的不同只是在更新时DFI接口必须暂停的周期数 |
| **dfi\_phyupd\_ack** | output | 1 | MC对更新请求的回应 |
| **Status Interface** | | | |
| **dfi\_data\_byte\_disable** | output | 32 | 数据字节禁用，该信号只能在初始化阶段定义 |
| **dfi\_dram\_clk\_disable** | output | 2 | DRAM时钟禁用，有效时PHY到DRAM的时钟被禁止使用 |
| **dfi\_freq\_ratio** | output | 2 | DFI频率比，只能在初始化阶段定义 |
| **dfi\_init\_start** | output | 1 | 两种用途，在初始化阶段用于指示dfi\_data\_byte\_disable，dfi\_freq\_ratio是否被定义，在正常操作情况下用于指示MC发起改变频率的请求 |
| **dfi\_init\_complete** | input | 1 | 初始化完成指示信号，在初始化完成后被拉高，当频率变化请求有效后此信号被拉低，直到重新初始化完成再次拉高 |
| **dfi\_parity\_in** | output | 1 | 奇偶校验值，用于指示对命令的奇偶校验 |
| **dfi\_alert\_n** | input | 2 | 奇偶校验错误指示信号 |
| **Training Interface** | | | |
| **dfi\_rdlvl\_req** | input | 1 | PHY-initiated read data eye training request |
| **dfi\_phy\_rdlvl\_cs\_n** | input | 2 | 选择与当前dfi\_rdlvl\_req关联的目标chip |
| **dfi\_rdlvl\_en** | output | 1 | 在 read data eye training 期间拉高 |
| **dfi\_rdlvl\_resp** | input | 4 | Read training response. |
| **dfi\_rdlvl\_gate\_req** | input | 1 | PHY-initiated gate training request ， |
| **dfi\_phy\_rdlvl\_gate\_cs\_n** | input | 2 | 选择gate training 的目标chip |
| **dfi\_rdlvl\_gate\_en** | output | 1 | PHY gate training logic enable，在gate training 期间拉高 |
| **dfi\_wrlvl\_req** | input | 4 | PHY write leveling request. |
| **dfi\_phy\_wrlvl\_cs\_n** | input | 2 | Write leveling chip select,只在dfi\_wrlvl\_req拉高时有效 |
| **dfi\_wrlvl\_en** | output | 1 | PHY write leveling logic enable.在write leveling期间拉高 |
| **dfi\_wrlvl\_strobe** | output | 1 | Write leveling strobe |
| **dfi\_wrlvl\_resp** | input | 4 | Write leveling response.指示write leveling 已完成 |
| **dfi\_lvl\_pattern** | output | 8 | 用于确定当前读取训练操作应运行哪种训练模式 |
| **dfi\_lvl\_periodic** | output | 1 | 用于确定training 类型 |
| **dfi\_phylvl\_req\_cs\_n** | input | 1 | DFI PHY训练芯片选择请求 |
| **dfi\_phylvl\_ack\_cs\_n** | output | 2 | MC回应PHY发起的training 请求 |
| **Low Power Control Interface** | | | |
| **dfi\_lp\_ctrl\_req** | output | 1 | Low power opportunity control request. |
| **dfi\_lp\_data\_req** | output | 1 | Low power opportunity data request |
| **dfi\_lp\_wakeup** | output | 4 | Low power wakeup time. |
| **dfi\_lp\_ack** | input | 1 | Low power acknowledge |
| **Error Interface** | | | |
| **dfi\_error** | input | DFI Error Width | DFI Error |
| **dfi\_error\_info** | input | DFI Error Width \* x 4 | DFI Error Info. |
| **Advanced Scheduler interface** | | | |
| **CmdfromAS** | | | |
| **cmdRdy** | output | 1 |  |
| **cmdValid** | input | 1 |  |
| **cmd** | input | 1 | 读1/写0命令 |
| **bank** | input | 2 | bank地址 |
| **col** | input | 10 |  |
| **group** | input | 2 |  |
| **rank** | input | 1 |  |
| **row** | input | 18 |  |
| **token** | input | tokenWidth |  |
| **priority** | input | 1 |  |
| **wrData** | input | dataWidth |  |
| **dataMask** | input | dataWidth/8 |  |
| **Response2AS** | | | |
| **ready** | output | 1 |  |
| **valid** | input | 1 |  |
| **rdData** | output | dataWidth |  |
| **token** | output | tokenWidth |  |

## 设计思路

本章节将介绍SDRAM Command Generator各模块的具体设计思路。SDRAM Command Generator主要包括以下几个模块：fifo\_wrap模块、mig\_wrap模块、widthMatch模块、DataStorage模块、RdDataStorage模块、Command\_disassembly模块。

## Fifo\_wrap

* 功能描述

Fifo\_wrap接收来自Advanced Scheduler的读写命令和token以及地址信息，根据地址信息中的bank group分配到对应的fifo进行缓存。等待mig中的group模块ready时下发命令。

## DataStorage

* 功能描述

DataStorage通过四个FIFO来缓存四个bank group的写数据以及掩码，FIFO的写使能在write cmd 和BG与当前FIFO匹配两个条件同时满足时有效，将数据写入对应的FIFO；在经过group调度后的写命令生效时，根据BG从对应的FIFO读出数据，并将数据发送到Command Disassembly 模块。

## RdDataStorage

* 功能描述

RdDataStorage的目的是保证返回数据与token的一致性，该模块包含datafifo和tokenfifo。经过group调度后的读命令在下发到Command Disassembly模块的同时，其token被写入到tokenfifo中。又因为Command Disassembly模块不会再打乱读命令下发的顺序，因此从Command Disassembly模块返回的读数据总是与tokenfifo中最老的token所对应。datafifo用于缓存从Command Disassembly模块读回的数据，当Advanced Scheduler ready时将token和读数据一起返回。

## Mig\_wrap

* 内部架构

Mig\_wrap集成了mig mc的子模块和fifo\_wrap以及widthMatch。其中mig mc的内部架构如下图所示：

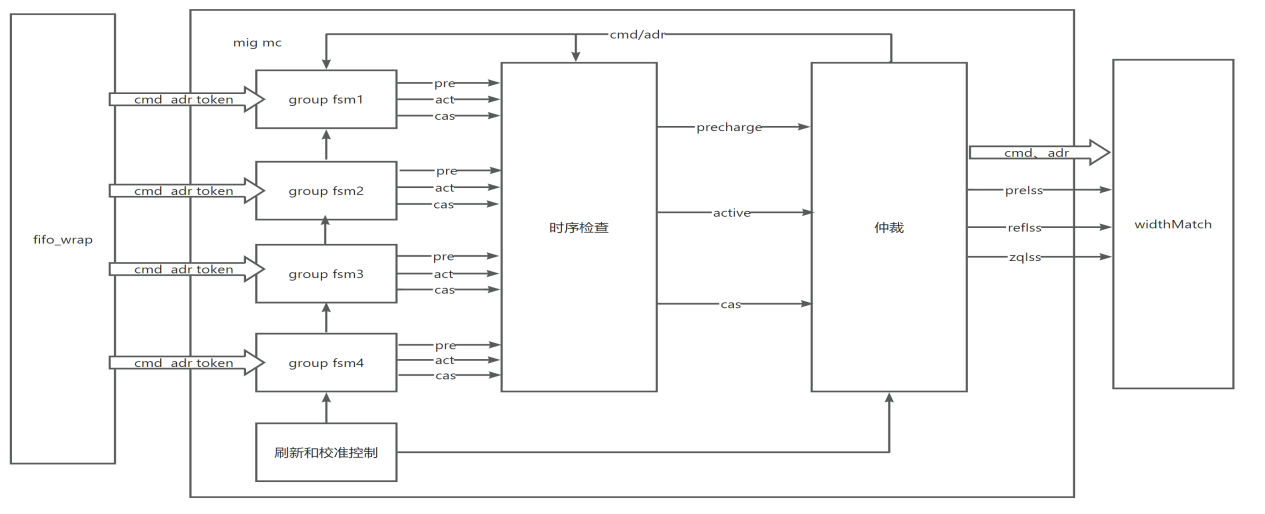


图7-2 mig mc内部架构图

* 功能描述

Mig mc的主要功能包括定期刷新和zq校准请求产生，act、precharge、write、read命令下发，时序检查和仲裁。

定期刷新和zq校准请求通过内部定时器产生，每当设定的时间到达时产生请求并挂起group fsm来进行刷新和zq校准流程。

Group模块可分为两个阶段：satge1 和 stage2，每个阶段都包含一个FIFO和一个FSM。Stage 1负责发送precharge和Activate命令，并且通过page表追踪DDR每个Bank的状态，Stage1 FIFO中存储来自上一级的读写事务，由Stage1 FSM根据当前命令对应的Page Status决定是否需要发送激活与预充电命令，以及何时将命令从Stage1 FIFO中pop出来并push到Stage2 FIFO中。Stage 2负责发送CAS命令与RMW事务，该阶段包含大量计时器以确保在最短的安全时间内完成命令调度，Stage2 FIFO输入为Stage1 FIFO输出。

时序检查部分主要是让命令按照规定的时序要求下发。

仲裁部分是根据优先级选出对应group fsm产生的命令下发，优先级的设置是依据当前的获得“胜利”的group来设置下一次仲裁的优先级。具体设置规则如下：  
 当此刻的“胜利”的group是group0时，此时的group0和group1之间1的优先级更高，group2和group3之间3的优先级更高 group01 和group23 之间group23优先级更高。

当此刻的“胜利”的group是group1时，此时的group0和group1之间0的优先级更高，group2和group3之间3的优先级更高 group01 和group23 之间group23优先级更高。

当此刻的“胜利”的group是group2时，此时的group0和group1之间0的优先级更高，group2和group3之间3的优先级更高 group01 和group23 之间group12优先级更高。

当此刻的“胜利”的group是group3时，此时的group0和group1之间0的优先级更高，group2和group3之间2的优先级更高 group01 和group23 之间group01优先级更高。

默认情况即当前没有“获胜”的group时。此时的group和group1之间1的优先级更高，group2和group3之间3的优先级更高 group01和group32之间group23优先级更高。

## Command Disassembly

* 功能描述

Command Dsiassembly模块的主要功能是接受上级模块传来的读写命令以及数据，将命令和数据以DFI协议规定的时序下发到PHY。

DFI模块相关参数介绍：

表7-2写传输相关参数：

|  |  |  |
| --- | --- | --- |
| **参数** | **最小值** | **描述** |
| **tphy\_wrlat** | 0 DFI PHY clock cycles | 从dfi控制接口发送写命令到拉高dfi\_wrdata\_en所需要的延时 |
| **tphy\_wrcslat** | 0 DFI PHY clock cycles | 从dfi控制接口发送写命令到拉高dfi\_wrdata\_cs\_n所需要的延时 |
| **tphy\_wrdata** | 0 DFI PHY clock cycles | 该参数规定了dfi\_wrdata\_en拉高到写入数据之间的延时 |
| **tphy\_wrcsgap** | 0 DFI PHY clock cycles | 两个目标芯片不同的写命令之间的延时 |

从dfi控制接口发出写命令到dfi写入数据之间的延时为tphy\_wrlat + tphy\_wrdata。

表7-3读传输相关参数：

|  |  |  |
| --- | --- | --- |
| **参数** | **最小值** | **描述** |
| **trddata\_en** | 0 DFI PHY clock cycles | 从dfi控制接口发送读命令到拉高dfi\_rddata\_en所需要的延时 |
| **tphy\_rdcslat** | 0 DFI PHY clock cycles | 从dfi控制接口发送读命令到拉高dfi\_rddata\_cs\_n所需要的延时 |
| **tphy\_rdcsgap** | 0 DFI PHY clock cycles | 两个目标芯片不同的读命令之间的延时 |
| **tphy\_rdlat** | 0 DFI PHY clock cycles | 该参数规定了dfi\_rddata\_en拉高到dfi\_rddata\_valid拉高的延时 |

从dfi控制接口发出读命令到dfi读回数据之间的延时为trddata\_en + tphy\_rdlat。

dfi\_wrdata\_en持续时间和数据流的长度保持一致；

dfi\_wrdata\_cs\_n的持续时间= tphy\_wrcslat + dfirw\_length + tphy\_wrcsgap;

dfirw\_length 通常等于burts\_length/2;

tphy\_wrcsgap定时参数指定当改变在dfi\_wrdata\_cs\_n信号上驱动的目标芯片选择时，命令之间需要的额外DFI PHY时钟的最小数量

Command Dsiassembly模块时序实现的整体思路是为每种信号都定义两组移位寄存器，第一组移位寄存器用来实现信号的偏移量和持续的时间（第一组移位寄存器的初值在读命令或者写命令发起时更新），第二组移位寄存器用来实现mc clock的整数拍延时。

## WidthMatch

* 功能描述

WidthMatch模块的主要功能是实现将mig产生的请求按照1：2的频率比进行转化。

* Mig产生请求的冲突解决

mig产生的激活，读/写，预充电请求可能会在同一拍出现，理论上极限情况可能会出现同一拍有三种请求，下发给DFI的两个phase会处理不过来。而preIss,refIss,zqIss在产生时会阻塞读、写、激活、预充电，所以不会出现冲突。为了解决极限情况下三种请求冲突问题，激活和预充电请求固定在对应的phase下发，在请求转化到DFI命令之间添加一个FIFO当三种命令冲突时把读写命令写入FIFO（在FIFO里有读写命令未处理时，新来的读写命令也存入FIFO，来保证读写命令的顺序），读出写/读命令放的phase取决于空闲的phase。同时为了解决读写请求可能的延后下发导致读/写到预充电时序不满足，增加DFI位宽匹配模块的反馈信号（反馈信息为DFI下发了读/写命令）来重载MC预充电仲裁模块计数器的初值。

# System Cache设计说明

## 模块功能

System Cache模块实现从filter收cmd和wdata，如果读cmd 命中，cache 将读数据直接返回给filter；如果cmd 未命中，cache需要发cmd和wdata 给下级 advanced scheduler， miss 的读请求数据会从advanced scheduler回来。

## 接口信号

System Cache和两个模块之间存在信号互连，分别是AS模块和Addr Map模块。表8-1中列出了本模块的接口信号。

表8-1 System Cache接口信号列表

|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **方向** | **位宽** | **功能描述** |
| **Global** | | | |
| **clk** | input | 1 | 时钟 |
| **rst** | input | 1 | 同步复位信号，高电平有效 |
| **Filter data path** | | | |
| **写data (filter vcache)** | | | |
| **flt2sc\_wdata\_valid** | input | 1 | 当前拍write data 有效 |
| **flt2sc\_wdata** | input | WDATA\_WIDTH | 当前拍write data 有效 |
| **flt2sc\_wdata\_end** | input | 1 | 当前拍为最后一个write data |
| **sc2flt\_wdata\_stall** | output | 1 | 当前拍为最后一个write data |
| **sc2flt\_wdata\_ptr\_valid** | output | 1 | 当前拍syscache返回的wdata\_ptr有效 |
| **sc2flt\_wdata\_ptr** | output | WDATA\_WIDTH | syscache返回的wdata\_ptr |
| **读 data (cache -> filter)** | | | |
| **sc2flt\_rd\_data** | output | RD\_DATA\_WIDTH | cache返回的read data |
| **sc2flt\_rd\_token** | output | RD\_TOKEN\_WIDTH | cache返回的rd token |
| **sc2flt\_rd\_data\_valid** | output | 1 | 当前拍rd\_data有效 |
| **sc2flt\_rd\_data\_end** | output | 1 | 当前拍为最后一个rd data |
| **Filter cmd path** | | | |
| **读cmd (filter -> cache)** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **flt2sc\_rrow** | input | ROW\_WIDTH | 地址拆分row |
| **flt2sc\_rcol** | input | COL\_WIDTH | 地址拆分col |
| **flt2sc\_rbank** | input | BANK\_WIDTH | 地址拆分bank |
| **flt2sc\_rgroup** | input | GROUP\_WIDTH | 地址拆分bank group |
| **flt2sc\_rrank** | input | RANK\_WIDTH | 地址拆分rank |
| **flt2sc\_rcmd\_type** | input | 2 | cmd 类型 |
| **flt2sc\_rcmd\_pri** | input | 2 | cmd 优先级 |
| **flt2sc\_rcmd\_valid** | input | 1 | 当前cmd 有效 |
| **flt2sc\_rdata\_mask** | input | 2 | data mask(同类型信号为cmd\_length) |
| **flt2sc\_rd\_token** | input | TOKEN\_WIDTH | UI发下来的读token |
| **sc2flt\_rcmd\_stall** | output | 1 | 拉高表示当前cache无法接收cmd(同类型信号ready) |
| **写cmd （filter -> cache)** | | | |
| **flt2sc\_wrow** | input | ROW\_WIDTH | 地址拆分row |
| **flt2sc\_wcol** | input | COL\_WIDTH | 地址拆分col |
| **flt2sc\_wbank** | input | BANK\_WIDTH | 地址拆分bank |
| **flt2sc\_wgroup** | input | GROUP\_WIDTH | 地址拆分bank group |
| **flt2sc\_wrank** | input | RANK\_WIDTH | 地址拆分rank |
| **flt2sc\_wcmd\_type** | input | 2 | cmd 类型 |
| **flt2sc\_wcmd\_pri** | input | 2 | cmd 优先级 |
| **flt2sc\_wcmd\_valid** | input | 1 | 当前cmd 有效 |
| **flt2sc\_wdata\_mask** | input | 2 | data mask(同类型信号为cmd\_length) |
| **flt2sc\_wdata\_ptr** | input | WDATA\_PTR\_WIDTH | UI发下来的写数据指针 |
| **sc2flt\_wcmd\_stall** | output | 1 | 拉高表示当前cache无法接收cmd(同类型信号ready) |
| **Advanced Scheduler cmd path** | | | |
| **读cmd (cache > advanced scheduler)** | | | |
| **sc2as\_rrow** | output | ROW\_WIDTH | 地址拆分row |
| **sc2as\_rcol** | output | COL\_WIDTH | 地址拆分col |
| **sc2as\_rbank** | output | BANK\_WIDTH | 地址拆分bank |
| **sc2as\_rgroup** | output | GROUP\_WIDTH | 地址拆分bank group |
| **sc2as\_rrank** | output | RANK\_WIDTH | 地址拆分rank |
| **sc2as\_rcmd\_type** | output | 2 | cmd 类型 |
| **sc2as\_rcmd\_pri** | output | 2 | cmd 优先级 |
| **sc2as\_rcmd\_valid** | output | 1 | 当前cmd 有效 |
| **sc2as\_rdata\_mask** | output | 2 | data mask(同类型信号为cmd\_length) |
| **sc2as\_rd\_token** | output | TOKEN\_WIDTH | UI发下来的读token |
| **as2sc\_rcmd\_stall** | input | 1 | 拉高表示当前as无法接收cmd(同类型信号为ready) |
| **写cmd (cache > advanced scheduler)** | | | |
| **sc2as\_wrow** | output | ROW\_WIDTH | 地址拆分row |
| **sc2as\_wcol** | output | COL\_WIDTH | 地址拆分col |
| **sc2as\_wbank** | output | BANK\_WIDTH | 地址拆分bank |
| **sc2as\_wgroup** | output | GROUP\_WIDTH | 地址拆分bank group |
| **sc2as\_wrank** | output | RANK\_WIDTH | 地址拆分rank |
| **sc2as\_wcmd\_type** | output | 2 | cmd 类型 |
| **sc2as\_wcmd\_pri** | output | 2 | cmd 优先级 |
| **sc2as\_wcmd\_valid** | output | 1 | 当前cmd 有效 |
| **sc2as\_wdata\_mask** | output | 2 | data mask(同类型信号为cmd\_length) |
| **sc2as\_wdata\_ptr** | output | WDATA\_PTR\_WIDTH | 可能是cache index，不一定是UI的指针 |
| **as2sc\_wcmd\_stall** | input | 1 | 拉高表示当前as无法接收cmd(同类型信号为ready) |
| **Advanced Scheduler data path** | | | |
| **读data （advanced scheduler > cache )** | | | |
| **as2sc\_rd\_data** | output | RD\_DATA\_WIDTH | as返回的read data |
| **as2sc\_rd\_token** | output | RD\_TOKEN\_WIDTH | as返回的rd token |
| **as2sc\_rd\_data\_valid** | output | 1 | 当前拍rd\_data有效 |
| **as2sc\_rd\_data\_end** | output | 1 | 当前拍为最后一个rd data |
| **写data (cache > advanced scheduler)** | | | |
| **sc2as\_wdata\_valid** | output | 1 | 当前拍write data 有效 |
| **sc2as\_wdata** | output | WDATA\_WIDTH | write data |
| **sc2as\_wdata\_end** | output | 1 | 当前拍为最后一个write data |
| **as2sc\_wdata\_stall** | input | 1 | 拉高表示当前as无法接收write data(同类型信号为ready) |
| **as2sc\_wdata\_ptr\_valid** | input | 1 | 当前拍as返回的wdata\_ptr有效 |
| **as2sc\_wdata\_ptr** | input | WDATA\_PTR\_WIDTH | as返回的wdata\_ptr |

## 设计思路

本章节将介绍System Cache模块的具体设计思路。System Cache模块的整体架构图如图8-1：

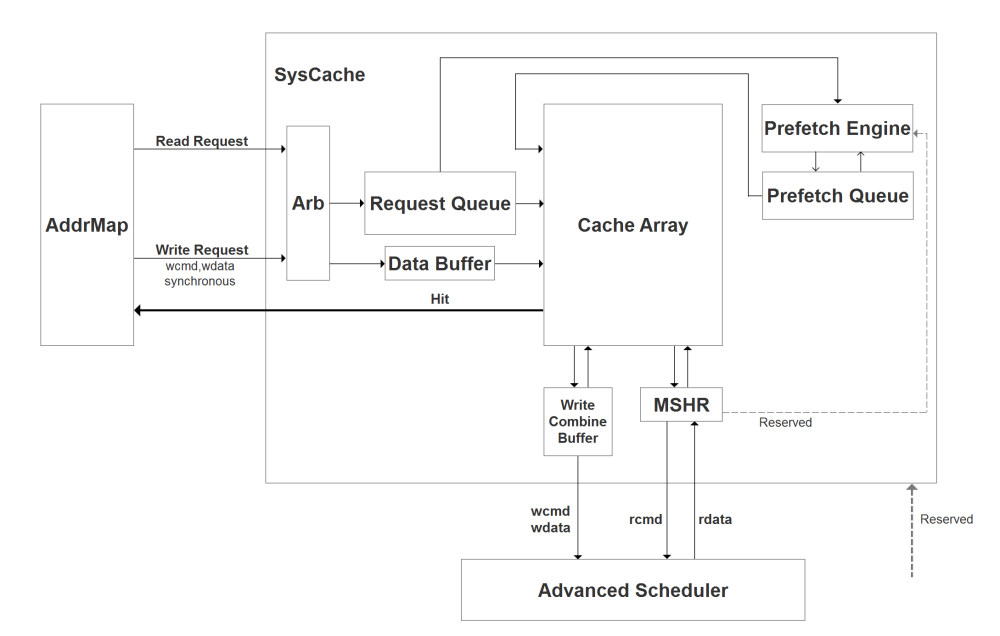


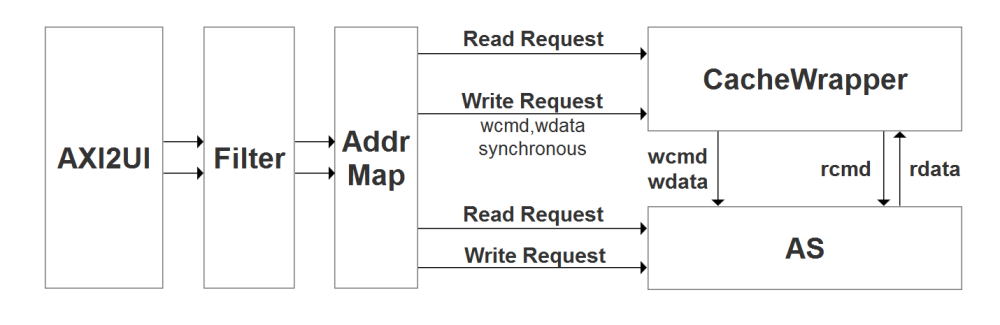
图8-1 System Cache模块整体架构图

## Cache Wrapper

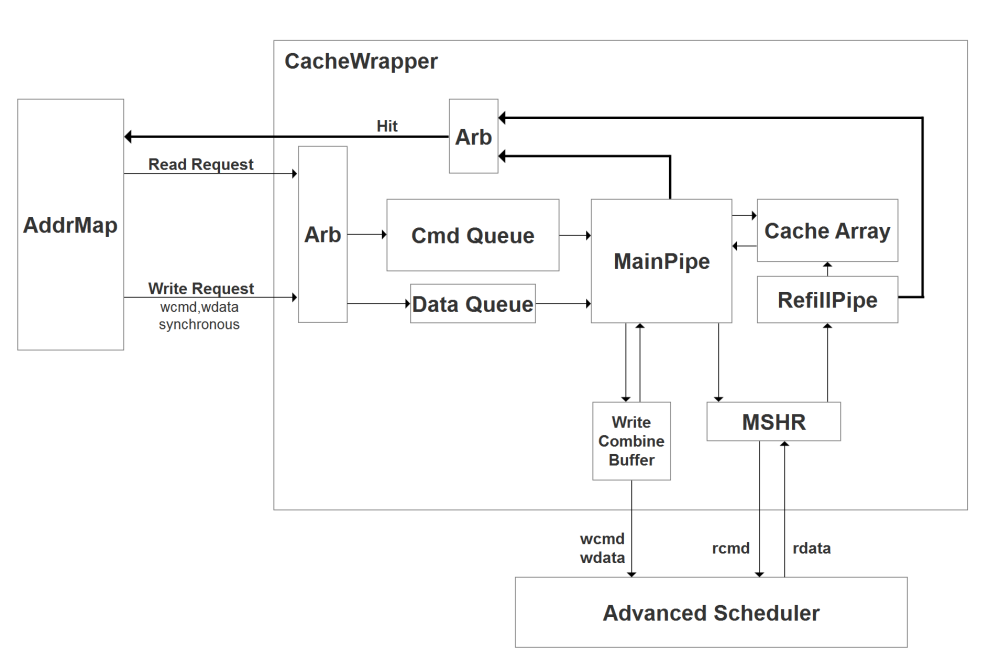
* **功能描述**

System Cache的顶层模块，集成所有Cache子模块

* **外部接口**



* **内部架构**



## Request Queue

* **功能描述**

RequestQueue模块的主要功能为：将来自AddrMap的读写两路数据合成一路后输入到MainPipeline。

内部包含CmdQueue和DataQueue两个队列，深度暂定64（参数化），分别用于存储读写请求、写数据，当输入接口同时收到读写请求时。

注意：

① 来自AddrMap的写请求与写数据同步

② Request Queue输出请求增加1比特指示请求类型，--0写，--1读

③ DataQueue深度 - CmdQueue深度 >= 2

stage1从CmdQueue中取写请求，写请求出stage2时才从DataQueue中取写数据

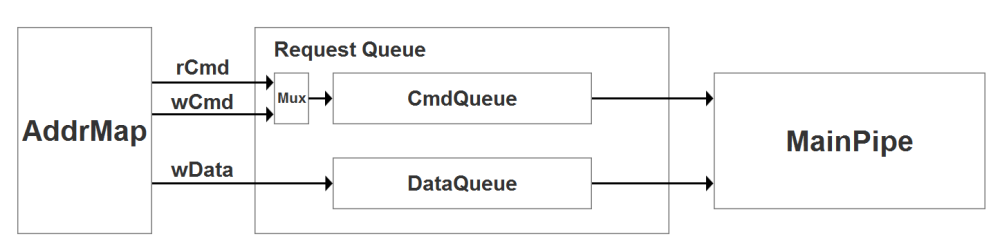
④ rcmd和wcmd同时有效时，token比较逻辑：

高位相同时表明是同一圈(1~1024)，token小的先收

高位不同时，token大的请求反而更早发出，先收token大的



* **架构设计**



* **顶层接口**

|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **方向** | **位宽** | **功能描述** |
| **Global** | | | |
| **clk** | input | 1 | 输入时钟 |
| **rst** | input | 1 | 复位信号，高有效 |
| **AddrMap Interface** | | | |
| **RdCmdFromAddrMap** | Flipped | SplitCmdIO Bundle | 拆分后的读请求 |
| **WrCmdFromAddrMap** | Flipped | SplitCmdIO Bundle | 拆分后的写请求 |
| **WrDataFromAddrMap** | Flipped | WrDataIO Bundle | 与写请求同步的写数据 |
| **MainPipe Interface** | | | |
| **RequestCmd** | Decoupled | SplitCmdIO Bundle + cmdType | 读写请求：0写，1读 |
| **RequestwData** | Decoupled | WrDataIO Bundle | 与写请求同步的写数据 |

## Cache Array

* **功能描述**

CacheArray位sysCache的存储部分，分为tagArray和dataArray，均分为8个bank并发来减少对同一个SRAM的读写冲突，每个bank包含1024个set，每个set包含4路。每个Cache行的大小为64B。Cache Size共为2MB。

Data Array暂定2MB，8 Banks，4 Ways（TagArray结构类似）

每个Bank由256KB单端口SRAM实现（1024 sets，每个set位4路64B数据）

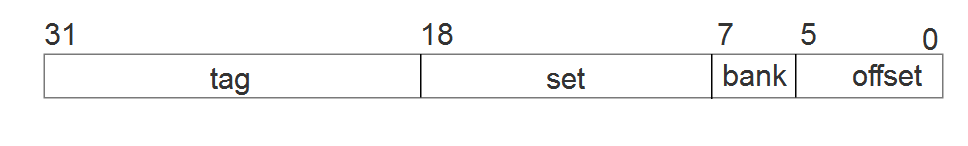
不同Bank读写请求可以并行，同一cycle内同bank的写请求会阻塞读请求

对外提供读写端口：

-- write：（bank，set，wdata，waymask）

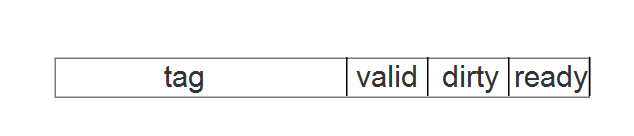
-- read：（bank，set）一拍后返回4路读数据

32位实地址划分如下图所示：



DataArray中每一项为64B数据

TagArray中每一项包含：tag，valid，dirty，ready

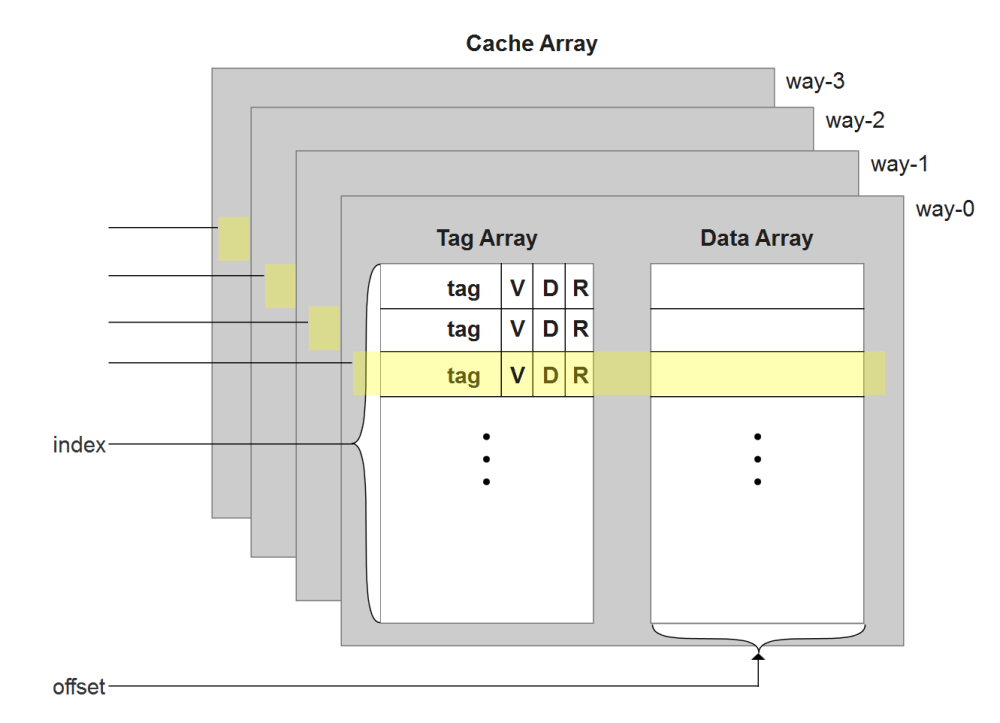


valid：高电平表示该项已经分配

dirty：高电平表示对应dataArray中数据为脏块

ready：低电平表示对应dataArray 中数据没有ready，miss读的数据还未回来，由refill完成回填后拉高

* **架构设计**



* **顶层接口**

**Data Array接口**

|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **方向** | **位宽** | **功能描述** |
| **clk** | input | 1 | 输入时钟 |
| **rst** | input | 1 | 复位信号，高有效 |
| **read** | Flipped | DataReadReq Bundle | 读data array请求，握手成功的一拍后返回读数据 |
| **write** | Flipped | DataWriteReq Bundle | 写data array请求，当拍写入，同一bank内的写请求将阻塞读请求 |
| **resp** | Decoupled | DataReadResult Bundle | 返回的4路读数据 |

**Tag Array接口**

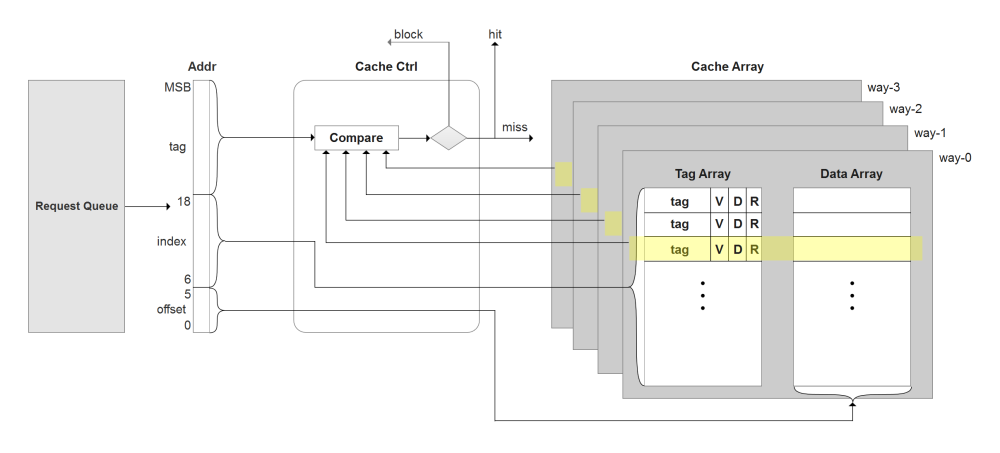
|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **方向** | **位宽** | **功能描述** |
| **clk** | input | 1 | 输入时钟 |
| **rst** | input | 1 | 复位信号，高有效 |
| **read** | Flipped | TagReadReq Bundle | 读tag array请求，握手成功的一拍后返回读数据 |
| **write** | Flipped | TagWriteReq Bundle | 写tag array请求，当拍写入，同一bank内的写请求将阻塞读请求 |
| **resp** | Decoupled | TagDataBundle Bundle | 返回的4路读数据 |

## MainPipe

* **功能描述**

MainPipe模块为sysCache的主要控制逻辑，分为三级流水线，通过读CacheArray信息判断请求是否命中，对于hit请求直接读写CacheArray，miss 请求发送给WCB模块或MSHR模块。

* **架构设计**



* **详细设计**

MainPipe的控制逻辑是一个三级流水的结构。

Stage1：读CacheArray

stage1接收来自Request Queue的请求，对请求进行地址划分，截取访存地址的bank号和index索引值，向CacheArray发送读请求，读取这个索引对应bank的Cache set，在下一拍返回数据。

注意：

① 当stage1接收到stage2阻塞反馈信号时，优先用阻塞请求地址读CacheArray，以等待回填完成后（ready=1）重新启动流水线

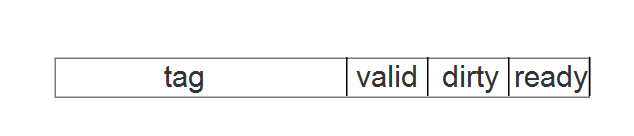
② CacheArray实现：每个bank为单独的单端口SRAM，相同bank内写请求阻塞读请求，不同bank可以并发

Stage2：判断是否命中

Stage2得到上一级流水线传递过来的访存请求信息，以及CacheArray返回的读数据，进行命中检查、判断是否阻塞，生成waymask掩码等，stage2功能点拆分如下：

**判断命中**

在stage2进行命中检查：截取访存地址的tag，并与该Cache组四路的tag进行比较，分为：hit，miss，block



-- miss：tag不匹配或者valid=0

-- hit：tag匹配并且valid=1，ready=1

-- block：case1，tag匹配，valid=1，但是ready=0；case2，tag不匹配但是四路全部ready=0

**判断阻塞**

对于block情况的说明：

① 在tagArray中增加ready位，低电平表示为miss read在Cache中分配了该项但是数据并未读回来，由refillPipe回填时置高。增加ready位的好处在于：不用查mshr冲突，可以增大mshr项数。

② stage2产生阻塞后将阻塞请求反馈给stage1，stage1优先使用阻塞请求地址读CacheArray

③ block请求在解锁将变为hit请求传递给stage3

**判断Forward**

stage1读CacheArray，stage3写CacheArray，因此对于连续两个同地址请求需要进行Forward判断，具体实现为：stage2连接到stage3写CacheArray总线，当写Cache地址与tag与当前请求匹配时，判断为isForward请求，并将forward请求数据传递给stage3。

**生成掩码**

按照如下规则生成waymask，优先级依次降低：

-- 如果hit，选择hit的那一路

-- 如果有未分配项（valid=0），选择未分配项

-- 如果有非dirty项，选择非dirty项

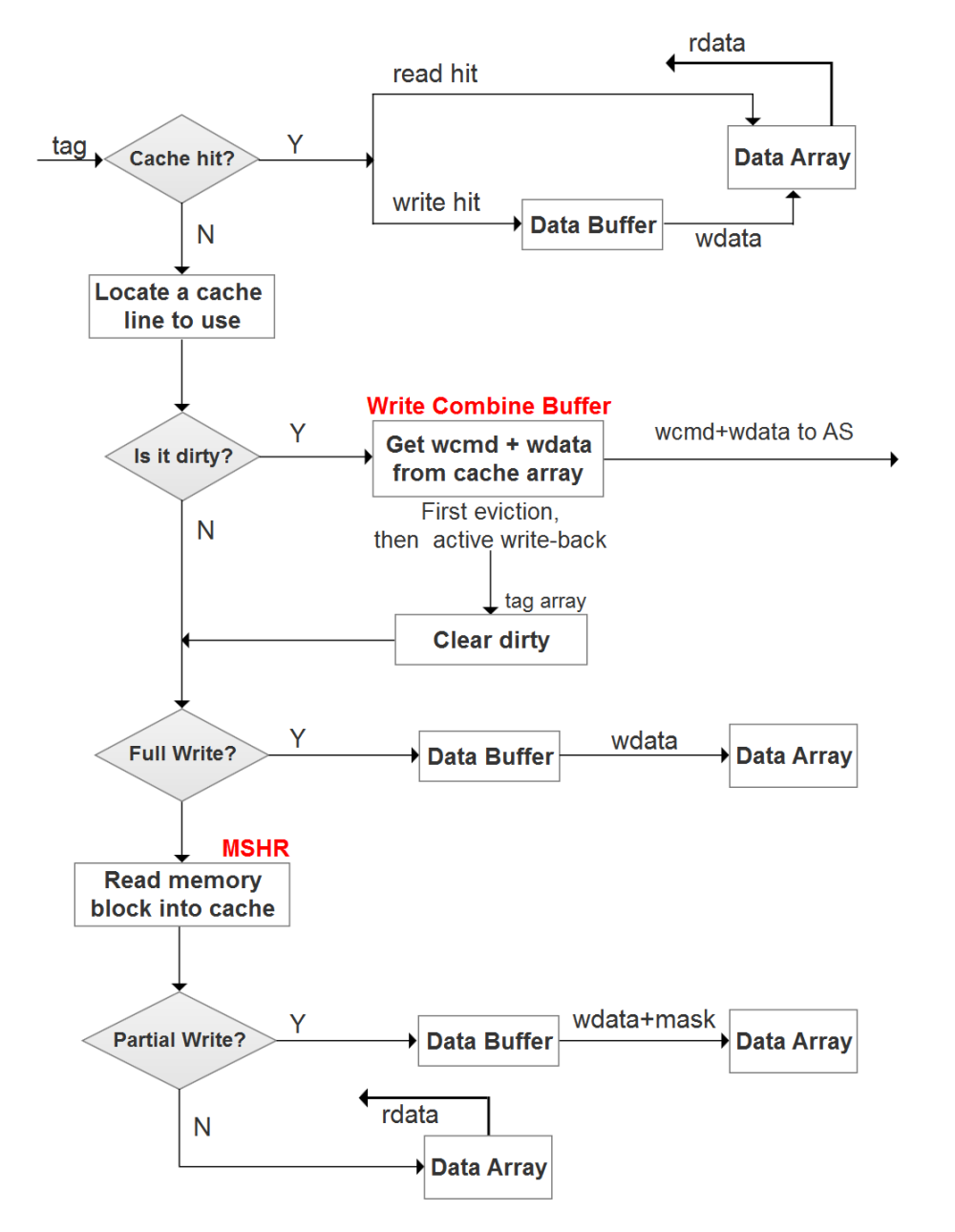
-- 如果全部ready，则随机选择一项；如果全部不ready，选择最先完成回填ready=1的一项；否则从ready项中选择最高项

**查WCB冲突**

stage2所有请求都会向WCB发起查冲突的请求，当拍返回查询结果（是否冲突，以及冲突数据）

stage3：处理请求

stage3根据stage2的结果对请求进行处理：



**HitWrite**

直接写tagArray，dataArray

**HitRead**

直接返回读数据

**Full MissWrite**

① 如果是dirty块，将dirty块写入WCB

② 如果WCB中有同地址请求，覆盖WCB中的数据

③ 更新tagArray和dataArray

**MissRead**

① 如果是dirty块，将dirty块写入WCB

② 如果WCB中无同地址请求，miss的读请求进MSHR

③ 如果WCB中有同地址请求，miss read变为hit read直接返回读数据

④ 更新tagArray和dataArray

对于CacheArray端口争用的说明：

目前只有MainPipe中stage1读CacheArray，MainPipe中stage3和RefillPipe都会写CacheArray，RefillPipe优先级更高

对于返回读数据端口争用的说明：

MainPipe和RefillPipe都会返回读数据，RefillPipe优先级更高

* **顶层接口**

|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **方向** | **位宽** | **功能描述** |
| **Global** | | | |
| **clk** | input | 1 | 输入时钟 |
| **rst** | input | 1 | 复位信号，高有效 |
| **Request Queue Interface** | | | |
| **RequestCmd** | Flipped | SplitCmdIO Bundle + cmdType | 从Request Queue出来的读写请求，进stage1 |
| **RequestwData** | Flipped | WrDataIO Bundle | 从Request Queue出来的写数据，进stage3 |
| **rdData2AddrMap** | Decoupled | RdDataIO Bundle | 读数据 |
| WCB Interface | | | |
| **searchReq** | Decoupled | SplitCmdIO Bundle + cmdType | 请求进stage2时，查询WCB中是否有同地址，当拍返回结果 |
| **conflictWCB** | input | 1 | 高电平表示WCB中有同地址 |
| **conflictWCBData** | input | CacheLineBits | WCB同地址的写数据 |
| **wcbReq** | Decoupled | WCBReqBundle | stage3踢dirty块进WCB |
| **clearConflict** | Decoupled | WrDataIO + overWrite | stage3请求处理完成后，释放WCB中同地址请求，overWrite高电平表示覆盖WCB中同地址数据 |
| **MSHR Interface** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **mshrReq** | Decoupled | MSHRInfo | stage3将miss read请求发送给mshr |
| **CacheArray Interface** | | | |
| **tagReadBus** |  | CacheTagArrayReadBus | tagArray读总线 |
| **dataReadBus** |  | CacheDataArrayReadBus | dataArray读总线 |
| **dataWriteBus** |  | CacheDataArrayWriteBus | dataArray写总线 |
| **tagWriteBus** |  | CacheTagArrayWriteBus | tagArray写总线 |
| **empty** | output | 1 | 高电平时表示stage2和stage3均无请求（reserved） |

## RefillPipe

* **功能描述**

RefillPipe模块为回填流水线，分为两级流水，主要功能为：将miss read读回的数据回填CacheArray，同时返回读数据给上层

* **详细设计**

**Stage1**

接收MSHR的回填请求，包含读数据、读token、对应miss read请求在CacheArray内的地址

mshr在收到AS读数据后，将该项miss read记录信息与读数据一起发送给refillPipe进行回填

**Stage2**

将读数据回填dataArraye，更新tagArray：valid=1，dirty=0，ready=1

返回读数据给AddrMap

对于CacheArray写端口争用的说明：

MainPipe中stage3和RefillPipe都会写CacheArray，RefillPipe优先级更高

对于返回读数据端口争用的说明：

MainPipe中stage3和RefillPipe都会返回读数据，RefillPipe优先级更高

* **顶层接口**

|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **方向** | **位宽** | **功能描述** |
| **Global** | | | |
| **clk** | input | 1 | 输入时钟 |
| **rst** | input | 1 | 复位信号，高有效 |
| **MSHR Interface** | | | |
| **mshrRefillReq** | Flipped | MSHR + rdata | mshr的回填请求，mshr在收到AS读数据后，将该项miss read记录信息与读数据一起发送给refillPipe进行回填 |
| **CacheArray Interface** | | | |
| **dataWriteBus** |  | CacheDataArrayWriteBus | dataArray写总线 |
| **tagWriteBus** |  | CacheTagArrayWriteBus | tagArray写总线 |

|  |  |  |  |
| --- | --- | --- | --- |
| **AddrMap Interface** | | | |
| **rdData2AddrMap** | Decoupled | RdDataIO | 返回读数据给AddrMap |

## Miss Status Holding Registers (MSHR)

* **功能描述**

MSHR是SysCache中专门用于处理miss读请求的模块。主要功能点包括：

1.接收MainPipe传来的读请求

2.将存储的读请求发送给AS

3.接收AS传回来的读数据

4.匹配读数据和读地址，发送给RefillPipe

* **架构设计**

MSHR采用RAM结构分别存储读数据和读请求，默认为64项（参数可配置）。

* **顶层接口**

|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **方向** | **位宽** | **功能描述** |
| **mpReq** | Flipped | MSHRInfo Bundle | Main pipeline 传给 MSHR的读miss 请求 |
| **asReq** | Decoupled | SplitCmdIO Bundle | MSHR 传给Advanced Scheduler的请求 |
| **DataFromAS** | Flipped | DataIO Bundle | Advanced Scheduler返回的读数据 |
| **rpReq** | Decoupled | MSHRInfo Bundle + BUNDLE\_PARAM.DATA\_WIDTH | MSHR给refill pipeline的miss 读请求和读数据 |

## WriteCombineBuffer(WCB)

* **功能描述**

WriteCombineBuffer(WCB)是SysCache中专门处理miss写请求的模块。主要功能点包括：

1.接收MainPipe传来的写请求和写数据

2.将模块内存储的写请求和写数据发送给AS

3.支持查找同地址冲突

* **架构设计**

WCB由寄存器组构成，默认为18个（数量可设置）。采用寄存器组的设计是为了方便MainPipe查找WCB中的同地址数据。

* **顶层接口**

|  |  |  |  |
| --- | --- | --- | --- |
| **信号** | **方向** | **位宽** | **功能描述** |
| **mpMissReq** | Flipped | WCBReqBundle | Main pipeline 传给 WCB的写请求 |
| **mpSearchReq** | Flipped | SplitCmdIO Bundle + Bool | Main pipeline传给WCB的查询请求 |
| **clearConflict** | Flipped | WrDataIO Bundle + Bool | 如果查到同地址写请求，使用该写数据覆盖原数据 |
| **isMatch** | Output | Bool | 是否找到和miss请求同地址的请求 |
| **isMatchData** | Output | CacheLineBits | 如果找到同地址请求，返回同地址请求的数据 |
| **asReq** | Decoupled | SplitCmdIO Bundle | WCB传给Advanced Scheduler的请求 |
| **asData** | Decoupled | WrDataIO Bundle | WCB传给Advanced Scheduler的数据 |