

第八届

全国大学生集成电路创新创业大赛

报告类型\*： 设计、仿真报告

参赛杯赛\*： 中科芯杯

作品名称\*： Hydra-高速多端口共享缓存管理模块

队伍编号\*： CICC1651

团队名称\*： Hydra

# 1.概述

Hydra是一款高速多端口共享缓存管理模块，支持管理16端口独立同时读写32块SRAM缓存。

除实现了赛题的基础要求外，Hydra的特点有：

1）实现了完全意义上的动态分配内存；

2）实现了无需缓冲结构的零延迟写入机制；

3）实现了极低延迟的读出机制（一般小于4周期，极端情况也不超过16个周期）；

4）最大程度保证时序严谨性，并证明各子模块良好的鲁棒性；

5）除SRAM本体外仅需不超过200KB外部资源（校验、内存管理、队列管理等）；

6）基于最常用的16位宽SRAM，且不要求真双口，仅需伪双口；

7）代码简介清晰，符合规范，配有标准的注释。

Hydra的创新点与难点有：

1）融入了页表管理的模式，极大优化运行逻辑，维持体系简洁；

2）参考并延申了FIFO、链表的数据结构思想，实现了时间复杂度为O(1)的内存分配与回收；

3）通过缓解内存碎片化，极致压缩内存管理所需的外部资源；

4）拟定多方面低算力需求的分配策略，极大地降低了读写延迟。

# 2.模块基本参数

时钟频率：250Mhz

SRAM大小：256Kbits

SRAM数量：32

SRAM规格：宽16×深16384

端口数量：16

理论带宽：TODO

理论延迟：TODO

综合面积：TODO

综合能耗：TODO

最长线长：TODO

子模块目录

占用存储资源

# 3.模块实现策略

## 3.1缓存管理与分配

### 3.1.1页表管理

对于每个SRAM，物理地址宽度为14，要选取所有SRAM中一个半字需要5+14=19位宽的地址，不利于存储（原因见第三节跳转表）。

同时，若对每半字的数据进行校验运算，则由公式2^r>n+r+1，可是至少需要5位校验位，所有数据所需的校验资源高达总SRAM容量的5/16。

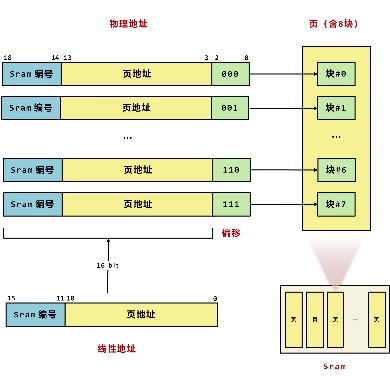
因此，为了便于存储地址、压缩校验功能所需的资源，将SRAM的资源进一步划分，每8半字为1页。

此时，指向SRAM中一页的地址被压缩成14-3=11位，选取所有SRAM中一页需要5+11=16位宽的地址，刚好与一半字对齐，便于存储。

校验也以页为单位，根据公式，每128位生成8位校验码，生成所需时间相比于16-5时间上的差别可忽略不计（3次异或运算的时间），且校验占用的资源大幅下降，只需总SRAM容量的1/16。

同时注意到一个数据包的长度折合后为4~64页/32~512半字。即只需用9位描述一个数据包的长度，即一个数据包有多少字节。9位长度信息与3位优先级信息、4位目的端口号信息结合，刚好为一个半字（9+3+4=16）。将其作为控制信号的一部分输入，可以使数据包的控制信息与有效数据之间能清晰地划分开来，处理时无需进行拼接操作，复杂度大大下降。

数据包存储占用的最后一页可能并不会被使用完全，即有几半字的区域无有效数据。为了对齐页的划分，这些空缺的半字并不会被利用（但并非永远浪费，数据包读出，占用的页被回收后，它们仍可以被新的数据包装填），经过计算，当数据包长度随机时，暂时被浪费的空间小于5%（TODO给出计算过程）。

由于模块数据传输均以半字为单位，一页的数据交互需要八个周期，故需要3位的批次计数器（batch）记录当前处理到页中的第几个半字的数据。页地址与批次计数器拼接即可得到半字的地址，根据其线性映射关系，我们称页地址为线性地址，线性地址11位拼接后得到物理地址14位，与前面的数据是吻合的。

### 3.1.2内存回收

传统的内存回收策略是为SRAM建立相应长度的位图（bitmap），其中每一位的数据分别对应SRAM中某单位（半字）是否有数据写入，1表示被占用，0表示未被占用，下称空闲。通过在写入数据时置1，读出数据时置0，即可描述SRAM的空闲位置，新来的数据只需直接写入空闲的位置。但是搜索位图中0的位置是一个时序性不良好的操作，即使经过独热码转化后，仍需通过遍历操作才能得到一个空闲位置。若要维护时序性，则需要等待较长的时间才能搜索到空闲位置，这与高速低延迟缓存管理模块的设计理念相悖。

Hydra采用的方案是为每一个SRAM维护一个“空闲队列”，其本质是一个FIFO，存储着空闲的线性地址，在写入数据时只需从队列头取出地址，读出数据时回收页，将地址插入队列尾，即可实现一个时间复杂度为O（1）的回收机制。不过其可观的时序性需要牺牲更多资源。目前采用的空闲队列FIFO深度为2048，宽度为11（线性地址的宽度），32块SRAM共需要88KB的存储资源。

### 3.1.3动态分配空间

Hydra基于空闲队列指导的内存回收，实现了完全动态的空间分配机制。具体的表现为Hydra可以在任意时刻（有空闲空间的时刻）无延时地申请一块空间，进行实时的数据存储。只要还剩下空闲的空间，任意端口就可以写入，与传统的静态分配策略相比，带宽更高且更灵活（例如吞吐量大的端口占有更多的资源）。

## 3.2数据包调度

### 3.2.1匹配较优SRAM

一个端口写入数据到SRAM时，有多个SRAM可用，若仅仅随机写入其中一个，会导致随着存入缓存的数据量增多，一个SRAM里会有各个端口（输出端口）的数据，不同端口同时输出数据需要访问SRAM，此时会发生冲突，一些端口的请求需要等待别的端口请求完数据才被受理，导致读出的延迟极高（少则几百个周期多则几千个周期）。

动态分配在该方面的缺陷无法避免，端口占用资源的不平衡总会使一个SRAM里有不同端口的数据。即使如此，我们仍能尽量缓解过高的读出延迟，将数据包开始存入SRAM前，Hydra会为其匹配一个较优的SRAM，具体的匹配规则如下：

硬性要求（不满足该规则的SRAM将会被忽略）：

SRAM容量充足

Hydra设计的框架规定一个数据包不得拆散在不同SRAM中，意味着SRAM剩余空闲空间必须大于等于新写入数据包的长度

SRAM未被其他端口锁定

锁定分两种情况：

SRAM正在被写

伪双口SRAM无法同时进行多次写操作

SRAM已被其他端口搜索过程匹配

若无此限制，不同端口可能同时认为一个SRAM“最优”，导致写冲突和复杂的仲裁逻辑

软性要求（越满足该规则的SRAM越被认为合适）：

包含数据包目的端口的数据较多

尽可能把目的端口的数据聚集在一起，这样缓解了一块SRAM极多端口数据混杂的情况

包含的数据对应的端口总数较少

同理，缓解了一块SRAM极多端口数据混杂的情况

为了维护良好的时序性，端口匹配SRAM的过程需要32周期，每个周期轮流询问一个SRAM，并建立一个中间寄存器保存已经搜索过的最优的SRAM，每次询问时，若硬性要求满足，则与当前最优的SRAM对比软性要求，若更优，则取而代之成为新的最优的SRAM。搜索完成后，可以得到较优的SRAM，接着再申请空闲页空间，开始SRAM的写入。

可见，从数据包开始进入Hydra，到真正开始写入SRAM，需要等待32周期的匹配过程，期间数据会流入端口前端模块的缓冲区中，在匹配完成后进一步送入后端模块中完成写入。

为了防止出现多个端口同时询问一个SRAM是否匹配的情况（会导致冲突，需要额外的仲裁逻辑），Hydra巧妙地设置了偏移量机制，保证每个端口每个周期搜索的SRAM刚好错开，互不干扰。

### 3.2.2跳转表维护端口队列

由于Hydra采用动态分配缓存的策略，故一个优先级队列中的数据包可能存放在不相邻的地方，因此需要专门存储所有数据包页地址。由于同一个优先级队列遵循先进先出的原则，可使用FIFO进行存储。

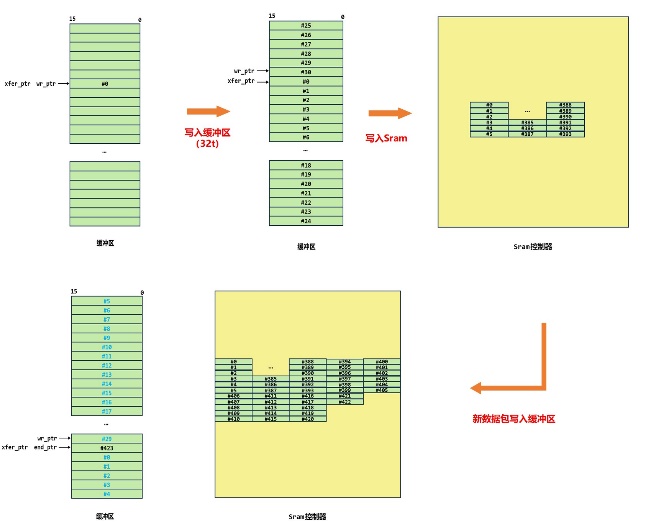
传统的方案便是给每个优先级队列建立一个FIFO管理队列中数据包的页地址，写入新的页时向FIFO末端插入页地址；读出页时从FIFO首端弹出页地址。但是由于动态分配的不平衡性，可能会存在一个优先级队列数据包极多，但是其他优先级队列几乎没有数据包的情况。若要支持最极端的条件（即所有空间都被一个优先级队列的数据包占用），每个队列的FIFO的深度需要65536，每个元素宽16位，记载了一个带SRAM编号的线性地址（5+11）。所有FIFO的存储资源共128\*65536\*16bits=16MB，完全无法接受。若酌情缩短FIFO的深度，则无法做到完全的动态分配，各队列有数据量限制。

Hydra采用的方案是为每个SRAM建立一个“跳转表”，将传统方案中的FIFO与链表数据结构的思想将结合，将队列中的页地址都视为一个结点，每个结点存储了下一结点的地址信息，这样就可以将所有优先级队列放在一起存储。读取队列中数据时，查找跳转表中当前页地址对应的内容，即可得到下一页的地址，再根据跳转表中下一页地址对应的内容，即可得到下下页的地址，以此类推，每个优先级队列只需维护队头的页地址，即可以进入的顺序访问队中的所有元素；向队列中写入数据时，只需将原来末端的跳转信息指向新的页地址，将新的页地址设置为新的末端即可。

值得注意的是，即使各端口同时独立地与SRAM进行高速数据交互，并不会存在同时访问/修改同一个SRAM的跳转表的问题，这是因为每个SRAM同时只会与一个端口进行交互（见SRAM匹配机制）。所以以SRAM为单位划分的跳转表可以被整合到BRAM资源中。

在多个端口同时向一个优先级队列末端插入数据时，可能会有冲突的情况，因此对于每个数据包，写入第一页时，暂时不和队列末端拼接，其后的所有页的跳转表信息正常更新，在数据包最后一页写完之后，数据包已经在跳转表中呈现为一条“断链”，只需将“断链”与队列末端拼接即可，为了避免同时来自多端口的“断链”拼接的问题，Hydra采用了轮询机制，即每个端口轮流进行拼接操作，即可保证跳转表和队列末端有条不紊地更新。

利用跳转表，Hydra可以在不增加读入、写出页地址所需时间复杂度的情况下，大大压缩地址管理所占用的资源，只需要32\*2048\*16=128KB。



### 3.2.3严格优先级调度

每个端口有8个优先级队列，Hydra建立了一个8位宽的二进制数，指示每个队列是否有数据。根据严格优先级，读出数据时应访问有数据的队列中最优先的队列。要实现这个操作，只需将该8位数转化为独热码，即可得到最高位“1”的位置，即最优先的有数据的队列。

得到当前应当读出哪个队列的数据后，根据队列头页地址得到应当访问哪个SRAM。考虑到可能会出现多个端口访问同一个SRAM的情况，此时需要引入轮询仲裁机制：得到端口应当输出哪个队列的数据后，向需要访问的SRAM端口号请求数据，每个SRAM都有一个16位宽的二进制数，指示了当前哪些端口正在请求数据，对其中为1的位进行轮询读取操作，每次轮询读出一页，宏观上表现即为访问同一SRAM的多个端口轮流输出128位数据，直至冲突结束。实现轮询操作使用了轮询位掩码，将其与请求指示数进行AND操作，即可得到当前应当处理哪个请求。

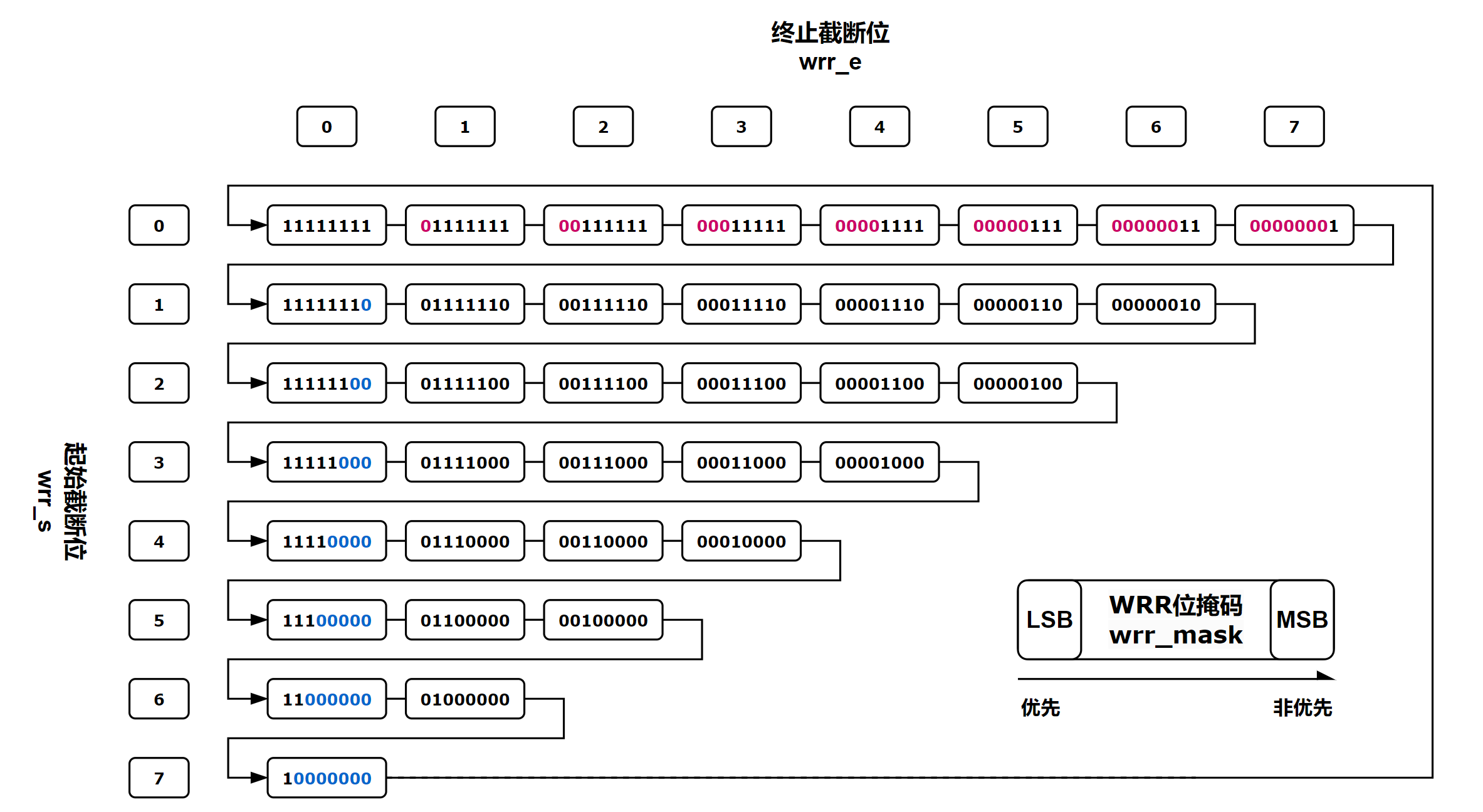
值得注意的一点是，若多端口同时访问同一SRAM时，数据以页的方式轮流被读取，这样可以缓解传统仲裁中，后面的请求需要等待前面的请求完全处理完才能开始处理的情况，大大缩短了从ready信号拉高到第一个rd\_vld拉高之间的延迟。

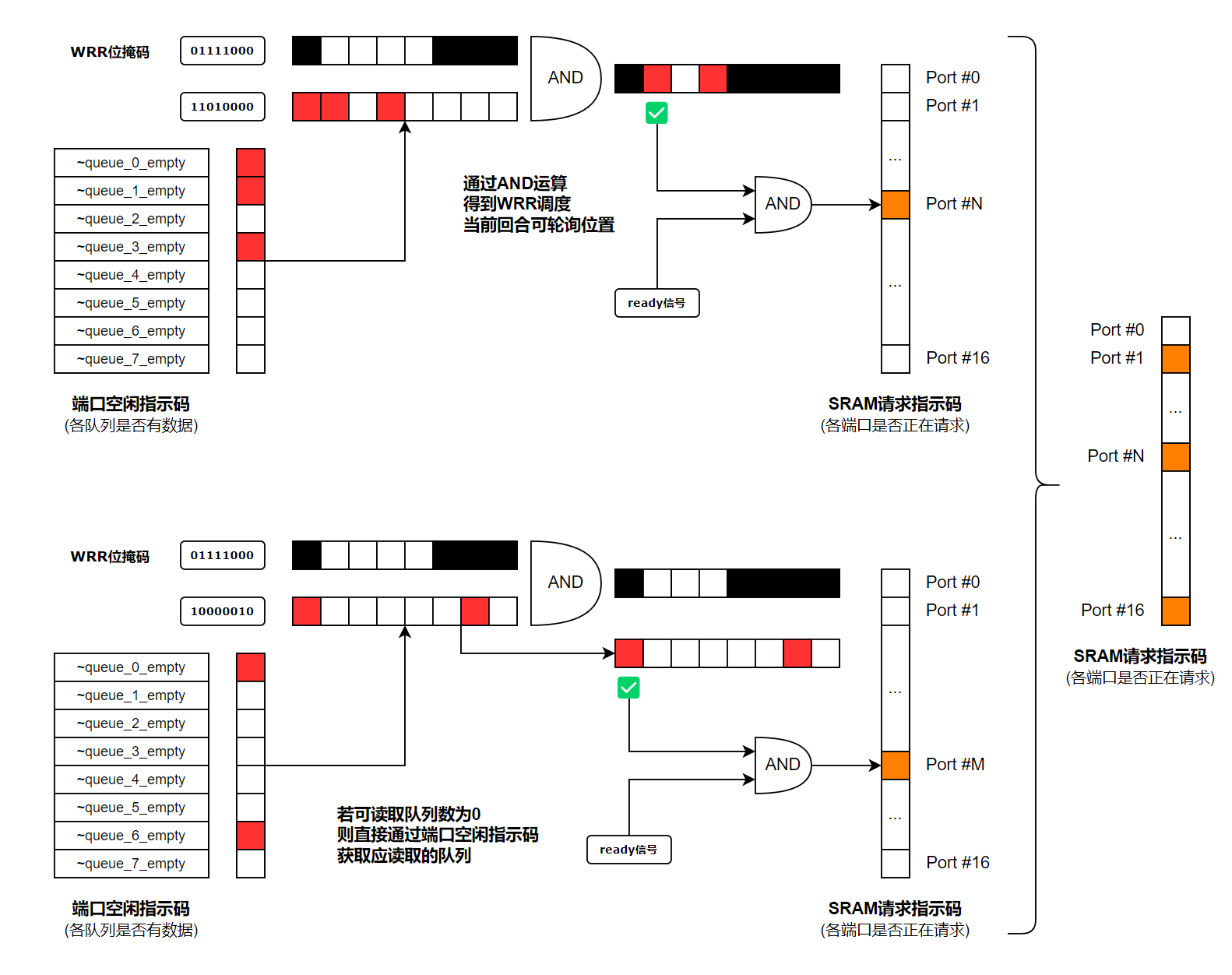
### 3.2.4 WRR调度

Hydra支持了使用最为广泛的一种WRR机制，即将输出数据包分为多个回合，第一回合八个优先级轮流输出，第二回合前七个优先级轮流输出，第三回合前六个周期优先级轮流输出……这样可以既考虑数据包的优先级，又缓解了低优先级数据包被一直堵塞无法输出的情况。

实现WRR调度采用了位掩码的方案，每个端口都有一个8位的WRR位掩码，由掩码头、掩码尾维护，每次输出新数据包时，将WRR位掩码与队列指示码取AND操作取最高位，即可得到当前应输出哪个队列的数据包。

通过引入掩码机制，Hydra成功将WRR调度从复杂的机制中解放，实现了占用资源极低（不到16位二进制数），时间复杂度极低（每次选择请求仅需1次AND运算所需的时间）。



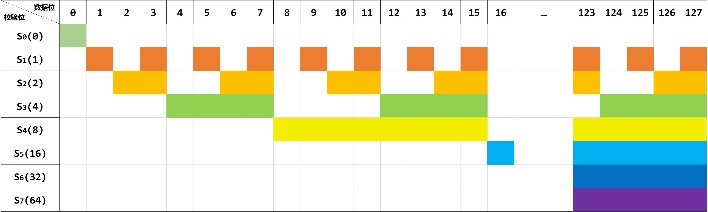


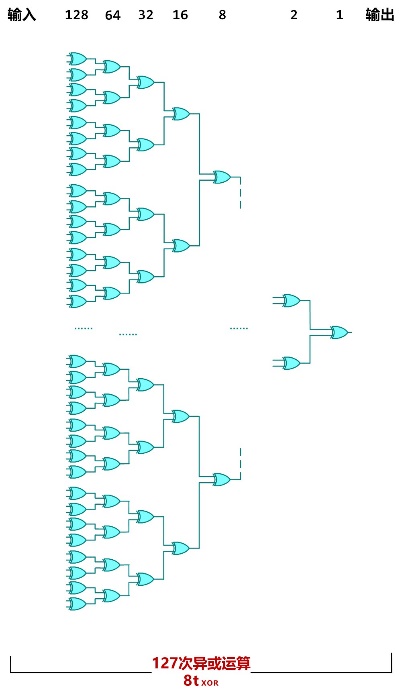
## 3.3 数据校验

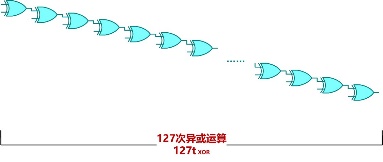
### 3.3.1汉明校验

Hydra采用了128-8的汉明校验，具体过程见下

采用了并行计算的思路，大大降低了127次异或运算所需的时间（只需7次并行异或运算的时间），使得校验模块的时序性得到了保障。







### 3.3.2校验信息存储

由于校验以页为单位，故校验信息的存储也以页为单位，每一个页地址指向了一个8位的校验码。每个SRAM都有一个校验存储空间，大小为2048\*8=2KB。由于同时只会有一个端口与SRAM进行交互，所以同时也只会有一个端口与校验存储空间进行交互，故校验存储可被置于片上资源的分布式RAM。

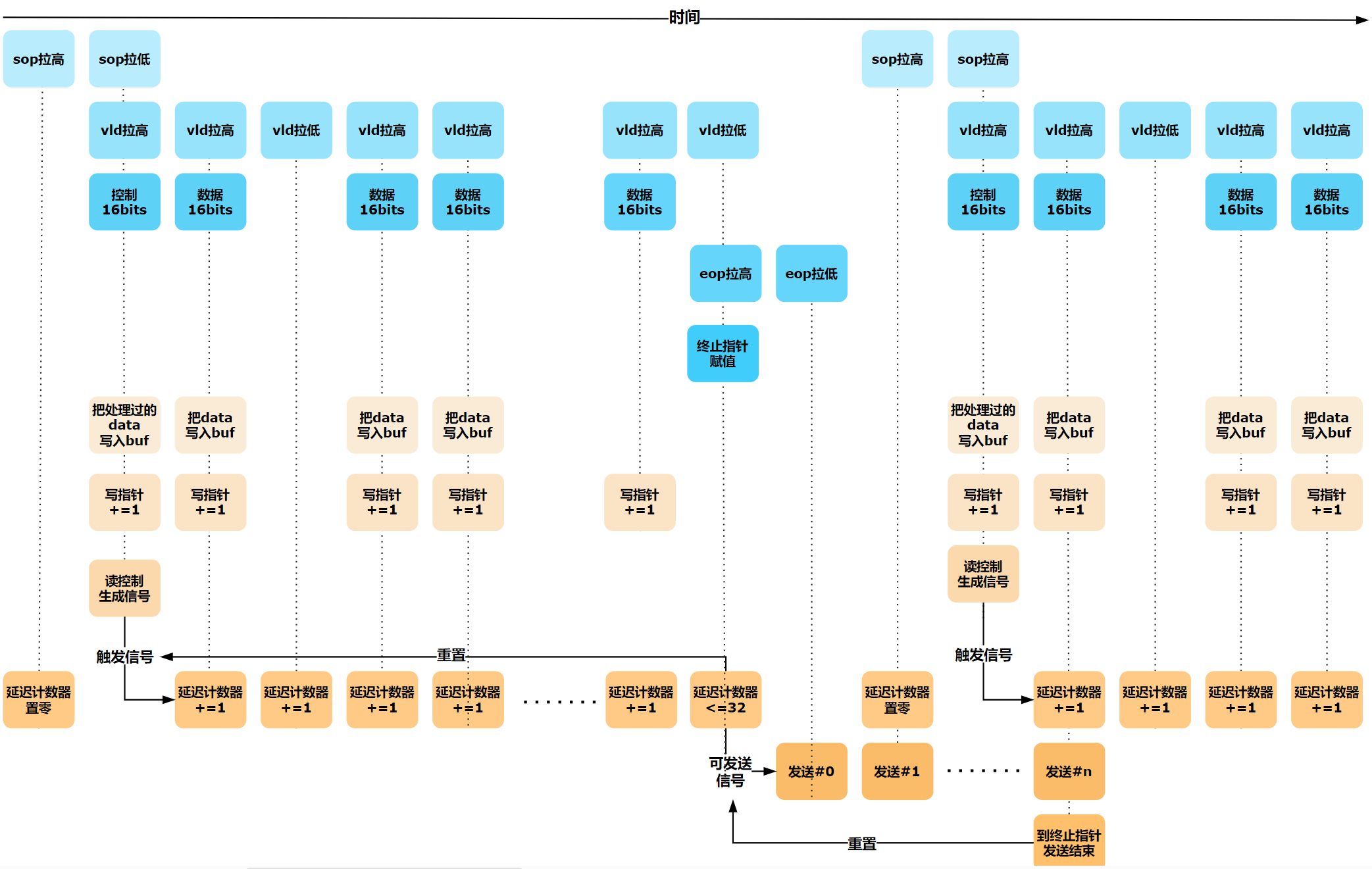
# 4.其他技术细节

## 4.1各模块寄存器说明

//TODO

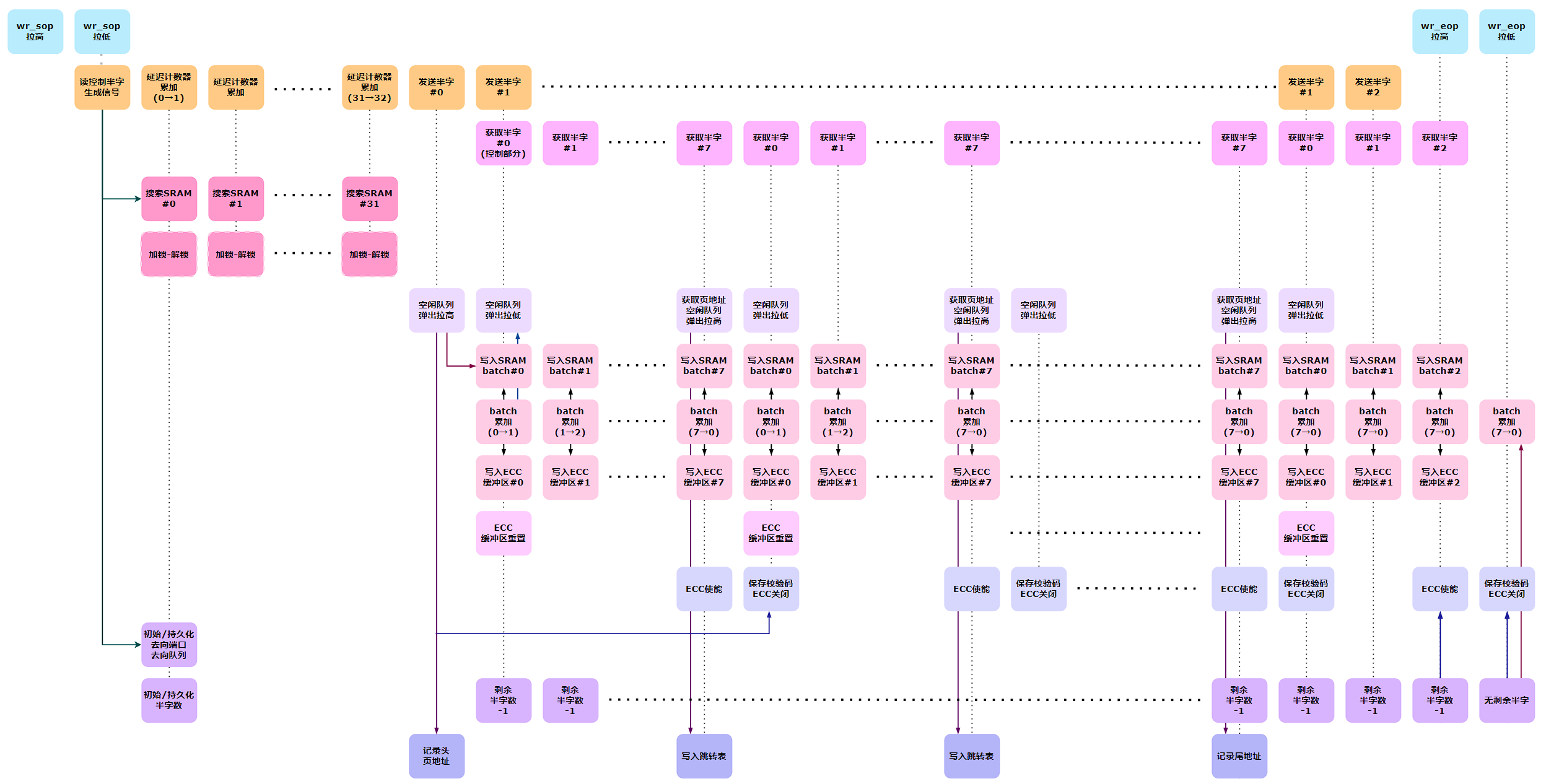
## 4.2重要逻辑说明

### 4.2.1前端预处理流水线



### 4.2.2后端读流水线

### 4.2.3后端写流水线



# 5.接口与配置

## 5.1接口

写控制IO口

写反馈IO口

读控制IO口

读反馈IO口

WRR使能口

## 5.2传输数据格式

控制信息组成

# 6.验证方法

## 6.1 RTL级仿真

### 6.1.1 RTL级电路图

### 6.1.2单模块仿真

### 6.1.3总模块仿真

给出三组测试样例

解析一组

## 6.2综合结果

### 6.2.1综合电路图

### 6.2.2 综合参数

//最好不要有warning，给出电路图

//给出面积与能耗参数、使用资源量

## 6.3 FPGA单模块测试

### 6.3.1伪双口SRAM

### 6.3.2 Ecc校验编解码模块

### 6.3.3端口前端模块

### 6.3.4 SRAM状态管理模块

# 7.设计优缺点

## 7.1优点

完全动态分配内存（队列无限制，按需分配）

时序性良好（流水化、无需位图）

偏好性存储降低读取延迟

可暂停发送等待腾出空间

代码风格良好，严格命名、清晰注释

## 7.2缺点

写入延迟32周期

需要额外的控制信息（数据包长度）

小额的浪费

回收功能占用资源较大

数据包只能在一个sram里

# 8.后续开发计划

## 8.1 “停止”接口

## 8.2更短的延迟，更佳的策略