

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

目录

[1.概述 4](#_Toc167062571)

[1.1题目再述 4](#_Toc167062572)

[1.2项目简介 4](#_Toc167062573)

[2.基本参数 5](#_Toc167062574)

[3.项目框架与策略 6](#_Toc167062575)

[3.1缓存管理与分配 6](#_Toc167062576)

[3.1.1页表管理 6](#_Toc167062577)

[3.1.2内存回收 7](#_Toc167062578)

[3.1.3动态分配空间 7](#_Toc167062579)

[3.2数据包管理 8](#_Toc167062580)

[3.2.1匹配较优SRAM 8](#_Toc167062581)

[3.2.2跳转表维护端口队列 9](#_Toc167062582)

[3.3数据包调度 11](#_Toc167062583)

[3.3.1严格优先级调度 11](#_Toc167062584)

[3.3.2 WRR调度 11](#_Toc167062585)

[3.4 数据校验 13](#_Toc167062586)

[3.4.1汉明校验 13](#_Toc167062587)

[3.4.2校验信息存储 13](#_Toc167062588)

[4.模块具体实现细节 14](#_Toc167062589)

[4.1各模块寄存器说明 14](#_Toc167062590)

[4.1.1 controller(顶层模块) 14](#_Toc167062591)

[4.1.2 dual\_port\_sram 15](#_Toc167062592)

[4.1.3 ecc\_encoder 15](#_Toc167062593)

[4.1.4 ecc\_decoder 16](#_Toc167062594)

[4.1.5 fifo\_null\_pages 16](#_Toc167062595)

[4.1.6 port\_frontend 17](#_Toc167062596)

[4.1.7 sram\_state 18](#_Toc167062597)

[4.2重要逻辑说明 19](#_Toc167062598)

[4.2.1写前端流水线 19](#_Toc167062599)

[4.2.2写后端流水线 19](#_Toc167062600)

[4.2.3读前后端流水线 20](#_Toc167062601)

[5.接口与配置 21](#_Toc167062602)

[5.1.写控制IO口 21](#_Toc167062603)

[5.1.1 IO口介绍 21](#_Toc167062604)

[5.1.2 使用方法 21](#_Toc167062605)

[5.2写反馈IO口 21](#_Toc167062606)

[5.2.1 IO口介绍 21](#_Toc167062607)

[5.2.2 使用方法 21](#_Toc167062608)

[5.3读控制IO口 22](#_Toc167062609)

[5.3.1 IO口介绍 22](#_Toc167062610)

[5.3.2 使用方法 22](#_Toc167062611)

[5.4读反馈IO口 22](#_Toc167062612)

[5.4.1 IO口介绍 22](#_Toc167062613)

[5.4.2 使用方法 22](#_Toc167062614)

[5.5 WRR使能配置 22](#_Toc167062615)

[6.验证方法 23](#_Toc167062616)

[6.1 RTL级仿真 23](#_Toc167062617)

[6.1.1 RTL级电路图 23](#_Toc167062618)

[6.1.2单模块仿真 23](#_Toc167062619)

[6.1.3总模块仿真 23](#_Toc167062620)

[6.2综合验证 23](#_Toc167062621)

[6.2.1 综合环境 23](#_Toc167062622)

[6.2.2 综合结果 24](#_Toc167062623)

[6.3 FPGA验证 26](#_Toc167062624)

[6.3.1 Ecc校验编解码模块 26](#_Toc167062625)

[7.设计优缺点 27](#_Toc167062626)

[7.1优点 27](#_Toc167062627)

[7.2缺点 27](#_Toc167062628)

[8.后续开发计划 28](#_Toc167062629)

[8.1 pause接口 28](#_Toc167062630)

[8.2读写延迟优化 28](#_Toc167062631)

[8.3 半动态模式 28](#_Toc167062632)

[8.4模块化框架 28](#_Toc167062633)

# 概述

## 1.1题目再述

赛题要求设计一款可对SRAM进行有效管理的SRAM控制器IP，如右图。具体要求如下：

1）支持管理至少32块256K bit的SRAM单元，总计至少8Mbits存储容量；

2）支持16个端口同时进行读写操作，每个端口的传输带宽需达到1Gbps；

3）每个端口支持8个优先级队列，实现按队列进行数据缓存；

**图1.1 IP示意图**

4）支持按数据包进行缓存和调度，数据包长度范围为64到1024字节；

5）支持多端口、多队列的动态共享缓存；

6）支持多个端口同时写入和写出数据，每个端口独立操作，相互之间不受影响；

7）支持数据校验，确保数据传输的准确性和完整性。

## 1.2项目简介

Hydra是一款高速多端口共享缓存管理模块，其基础功能包括但不限于：

1. 管理16个端口独立同时读写32块SRAM缓存；
2. 每端口8个优先级队列按优先级调度数据包；
3. 所有端口所有队列动态分配缓存；
4. 对SRAM存储的数据进行SEC数据校验；
5. 提供严格优先级、WRR两种调度模式。

Hydra在实现以上基础功能的同时，设计亮点有：

1. 引入页表管理的思想，优化逻辑的复杂程度，压缩模块空间占用；
2. 采用特殊的搜索机制，降低由于读出冲突导致的读延迟；
3. 引入链表的思想，降低维护优先级队列所需的资源；
4. 所有操作均流水化，在维护良好时序严谨性的同时提升性能；
5. 并行优化的数据校验加码-解码逻辑；
6. 采用位掩码实现极低复杂度的WRR机制。

综上所述，Hydra在完全实现题目基本要求的前提下，采用大量设计优化模块空间占用，提升模块性能，满足赛题背景中关于网络设备高速数据缓存的需求。

# 2.基本参数

**时钟频率**：250Mhz

**端口数量**：16

**SRAM数量**：32

**单块SRAM规格**：伪双口/真双口 256Kbits (16×16384)

**写入带宽**：3.7Gbps/端口

**写入延迟**：32~48周期

**读出带宽**：0.23~3.7Gbps/端口 （注：大多数情况下可跑满3.7Gbps）

**读出响应延迟**：12~140周期（注：大多数情况下仅需下限12周期）

**存储资源占用情况**（合计约2Mbits，320块片上BRAM）：

跳转表：32\*32Kbits = 1Mbits

空闲队列：32\*22Kbits = 704Kbits

ECC校验信息：32\*16Kbits = 512Kbits

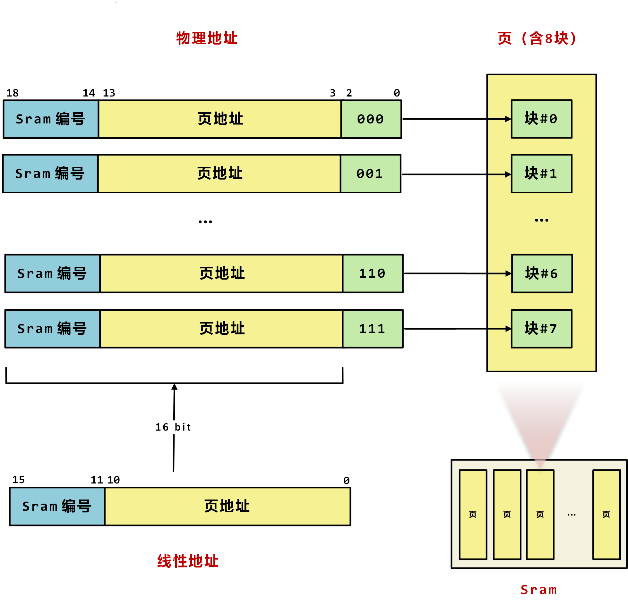
其他存储占用：< 8Kbits

# 3.项目框架与策略

## 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）。

 若对每半字的数据进行校验运算，由公式2^r>n+r+1可知至少需要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。

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。

**图3.1.1 页表管理中地址的对应关系**

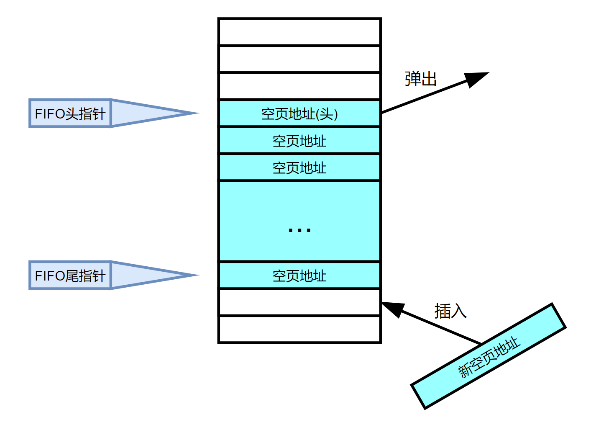
同时注意到一个数据包的长度折合后为4~64页/32~512半字。即只需用9位描述一个数据包的长度，即一个数据包有多少字节。9位长度信息与3位优先级信息、4位目的端口号信息结合，刚好为一个半字（9+3+4=16）。将其作为控制信号的一部分输入，可以使数据包的控制信息与有效数据之间能清晰地划分开来，处理时无需进行拼接操作，复杂度大大下降。

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），当数据包长度随机时，被浪费的空间小于2%，即使数据包均为不利好页表划分的小数据包，被浪费的空间也小于5%。

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。

**图3.1.2 空闲链表运作机制**

### 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数据包管理

### 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）

1）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。

2）SRAM未被其他端口锁定，锁定分两种情况：

I）SRAM正在被写；

伪双口SRAM无法同时进行多次写操作。

II）SRAM已被其他端口搜索过程匹配。

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）

1）包含数据包目的端口的数据较多；

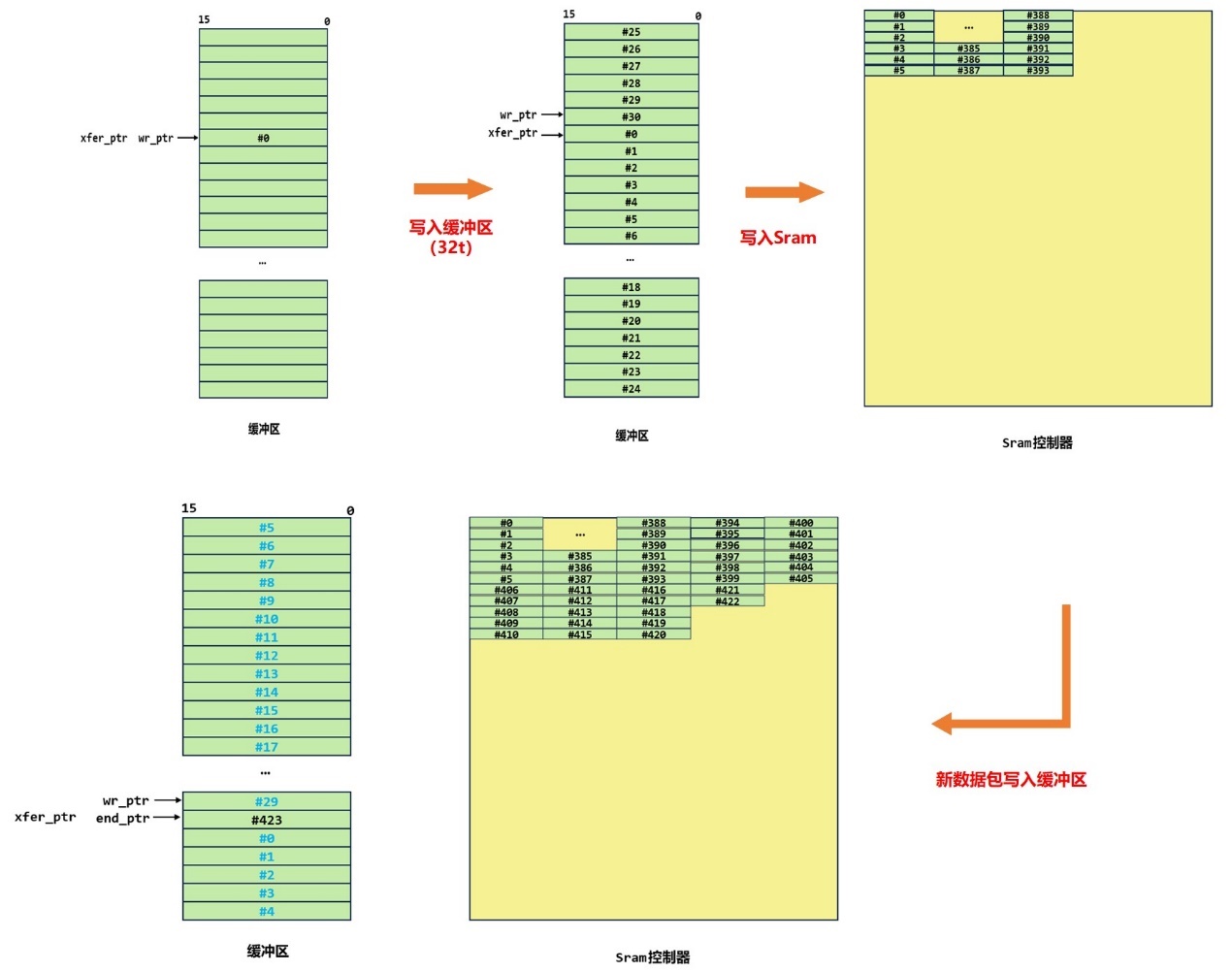
2）包含的数据对应的端口总数较少。

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。

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。



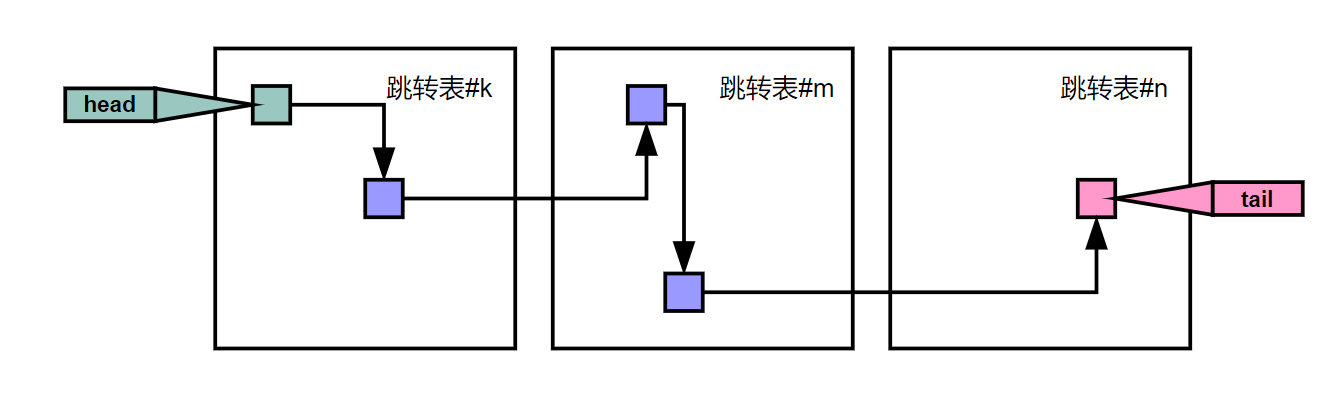
**图3.2.1 前端缓冲区机制**

### 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。

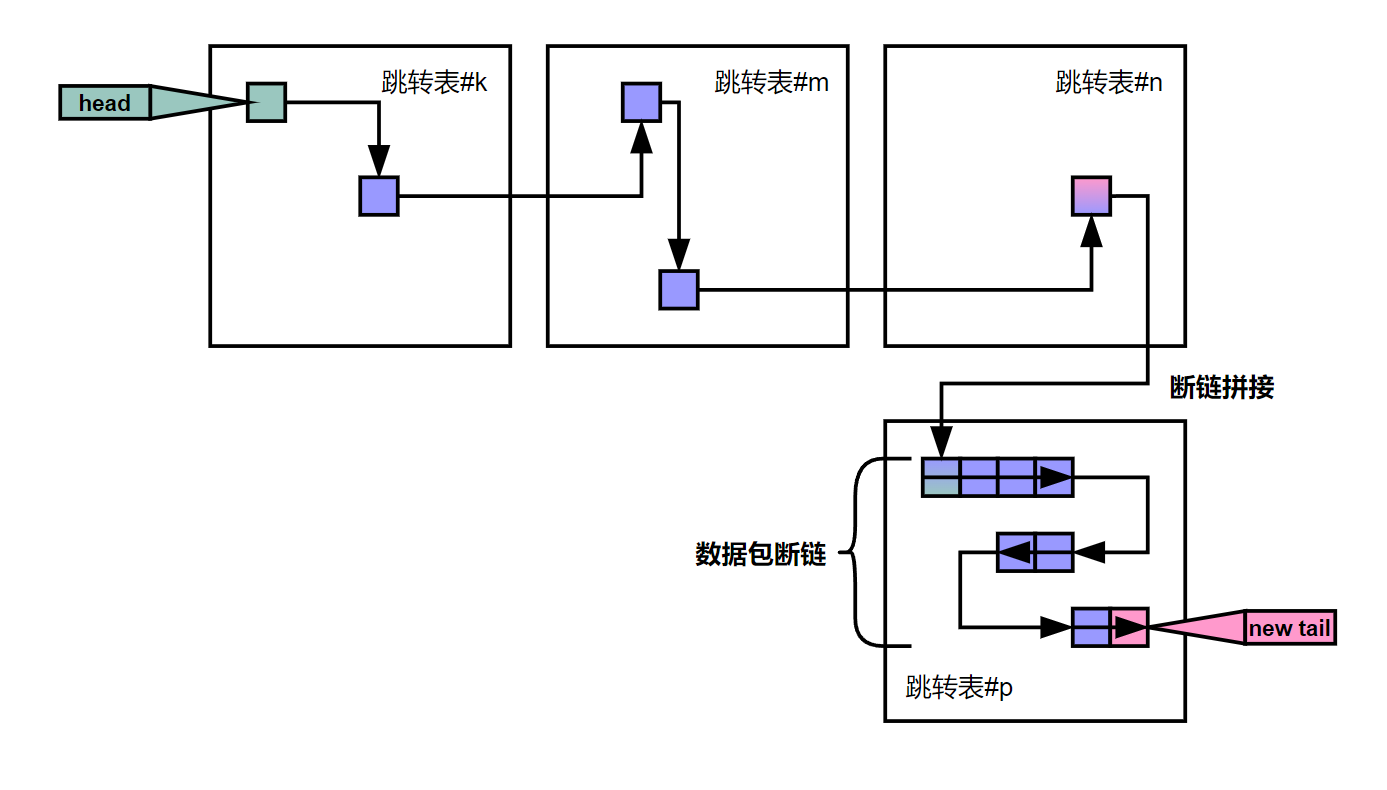
Hydra采用的方案是为每个SRAM建立一个“跳转表”，将传统方案中的FIFO与链表数据结构的思想将结合，将队列中的页地址都视为一个结点，每个结点存储了下一结点的地址信息，这样就可以将所有优先级队列放在一起存储。读取队列中数据时，查找跳转表中当前页地址对应的内容，即可得到下一页的地址，再根据跳转表中下一页地址对应的内容，即可得到下下页的地址，以此类推，每个优先级队列只需维护队头的页地址，即可以进入的顺序访问队中的所有元素；向队列中写入数据时，只需将原来末端的跳转信息指向新的页地址，将新的页地址设置为新的末端即可。



**图3.2.2(1) 跳转表维护优先级队列中的数据页**

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。



**图3.2.2(2) 并发写入情况下数据包入队机制**

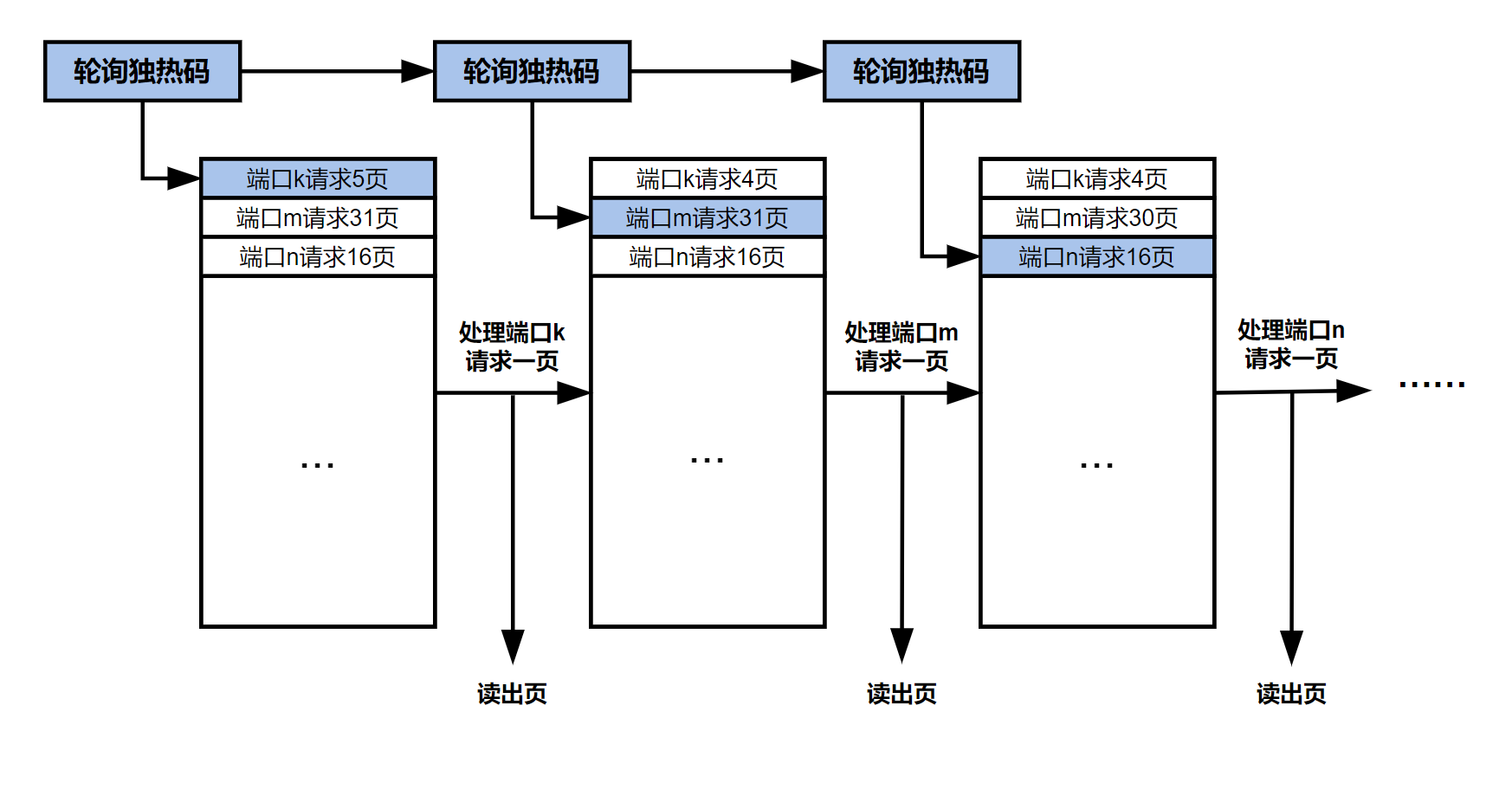
利用跳转表，Hydra可以在不增加读入、写出页地址所需时间复杂度的情况下，大大压缩地址管理所占用的资源，只需要32\*2048\*16=128KB。

## 3.3数据包调度

### 3.3.1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。

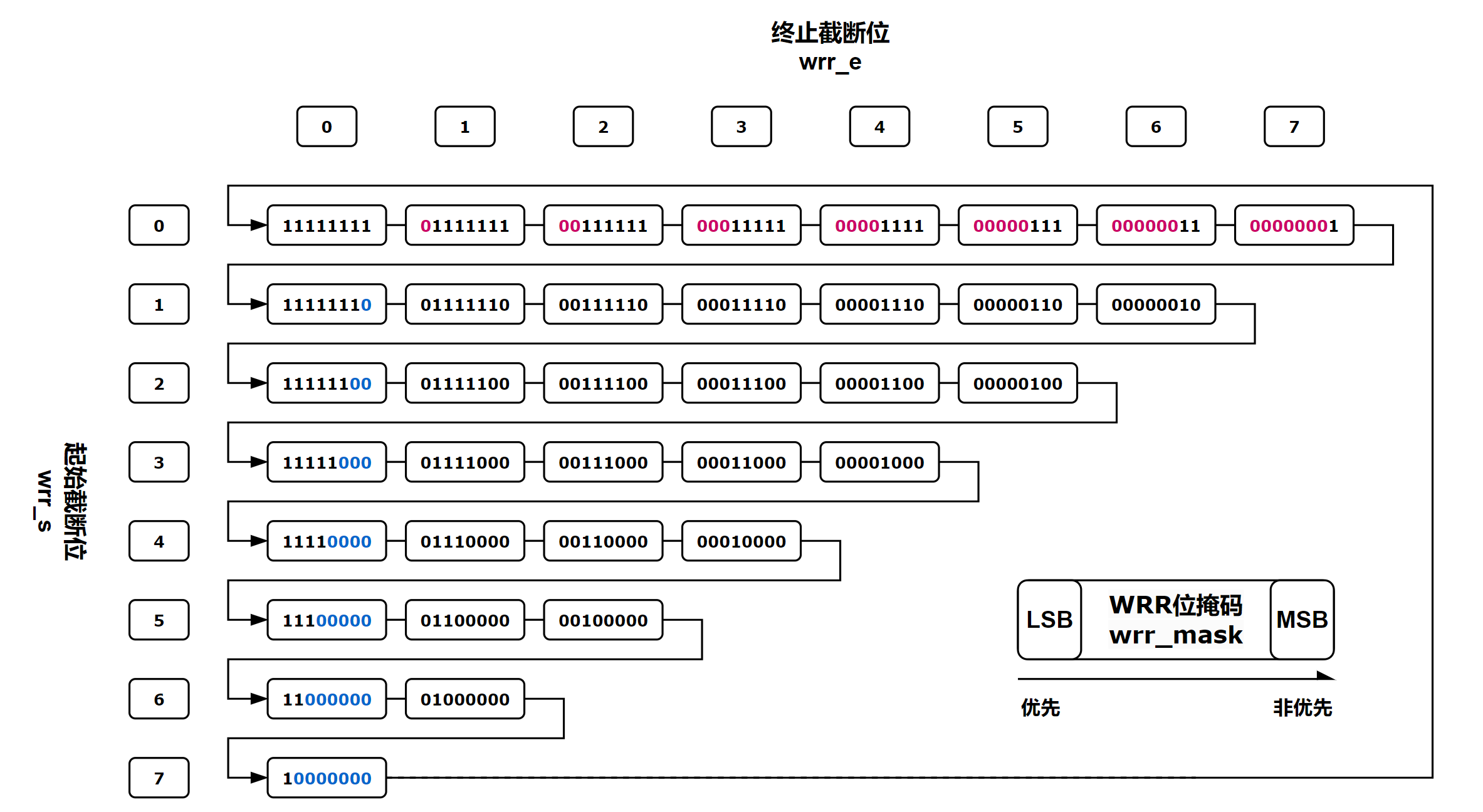


**图3.3.1 并发读取情况下轮询处理机制**

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。

### 3.3.2 WRR调度

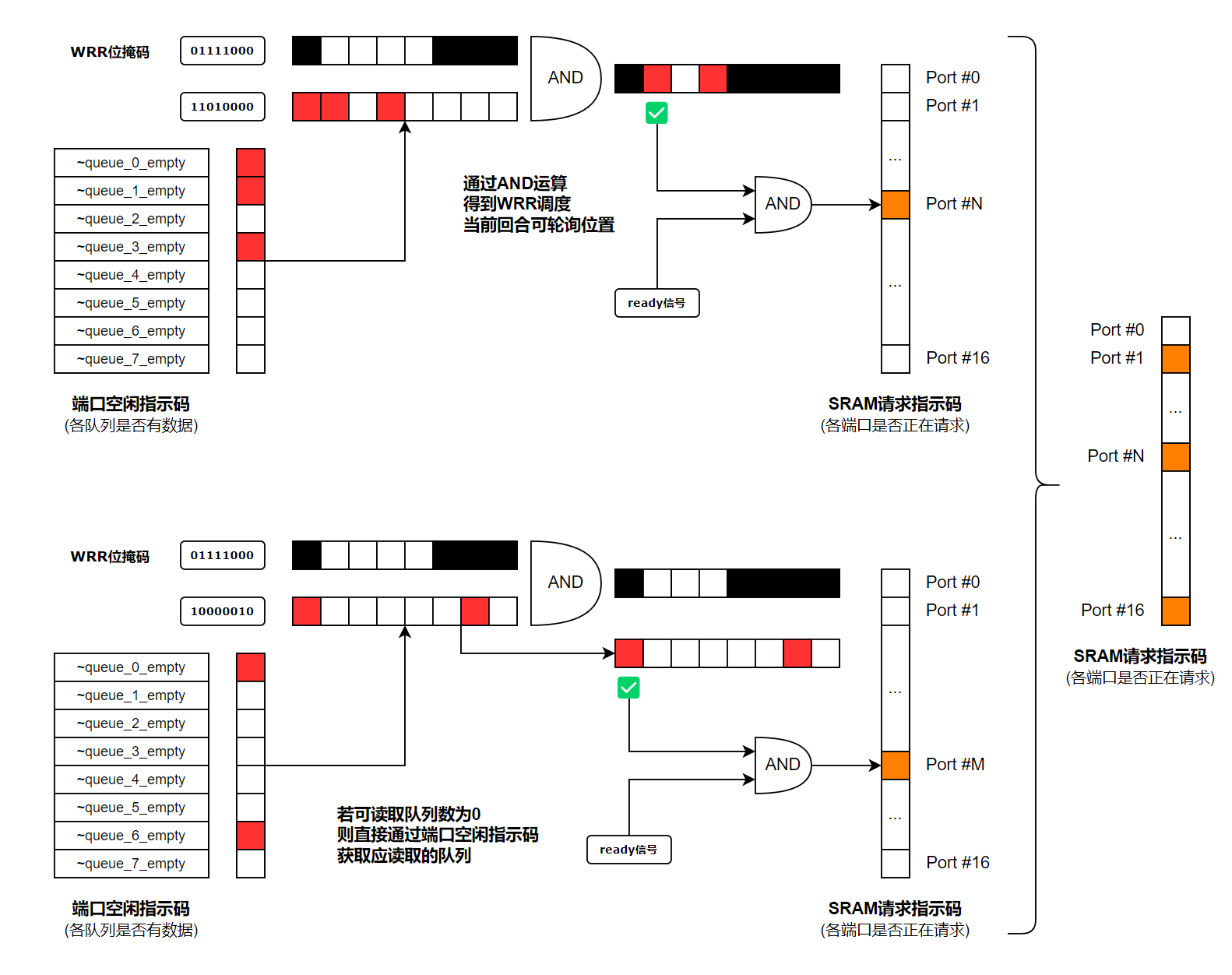
Hydra支持了使用最为广泛的一种WRR机制，即将输出数据包分为多个回合，第一回合八个优先级轮流输出，第二回合前七个优先级轮流输出，第三回合前六个周期优先级轮流输出……这样可以既考虑数据包的优先级，又缓解了低优先级数据包被一直堵塞无法输出的情况。



**图3.3.2(1) WRR掩码状态机**

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）。

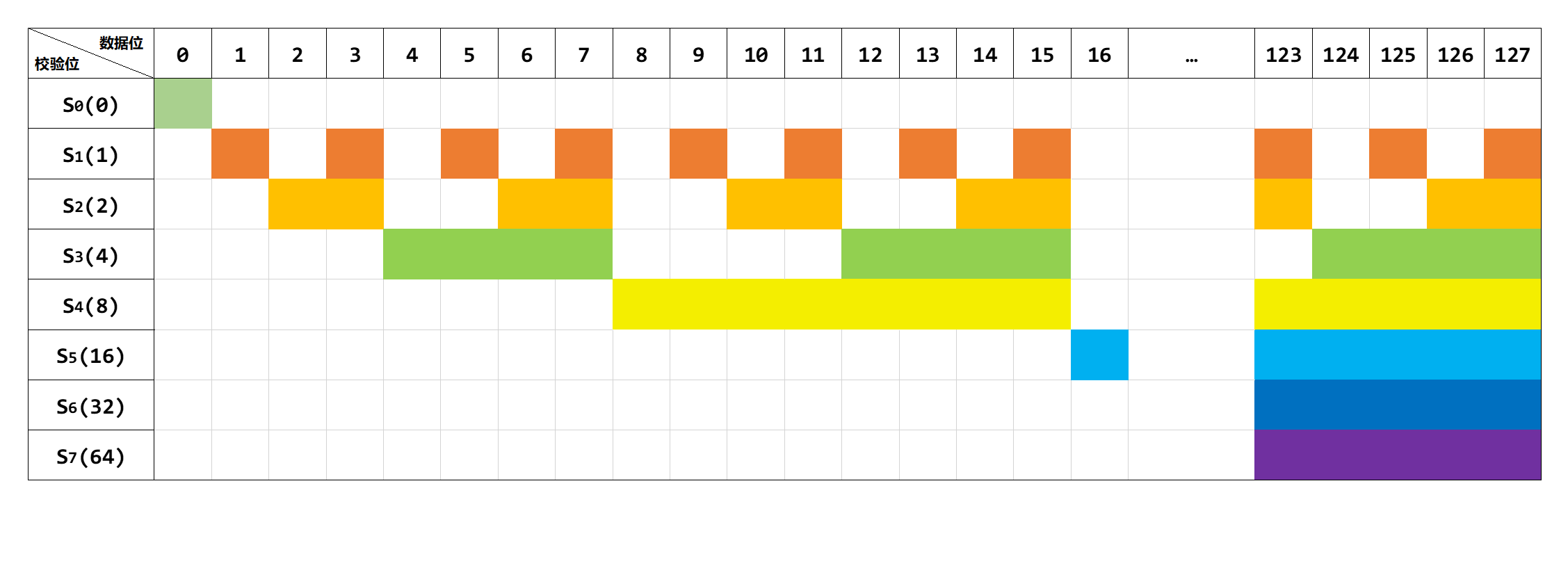


**图3.3.2(2) WRR掩码结合优先级判定队列号**

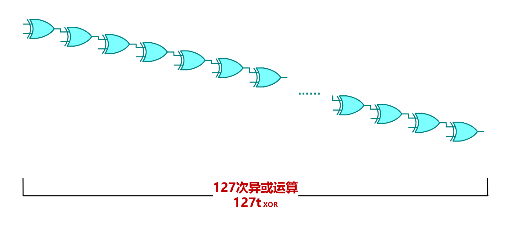
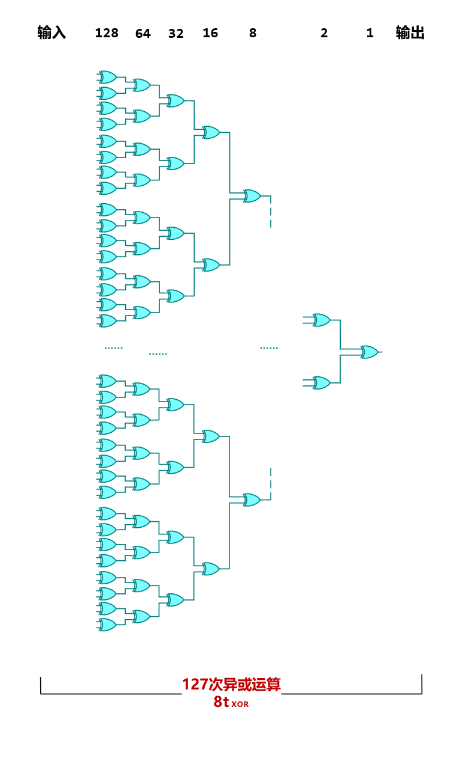
## 3.4 数据校验

### 3.4.1汉明校验

Hydra采用了(136,128)汉明校验，支持SEC（单错误纠错）。



**图3.4.1(1) 汉明码计算原理（可视化H矩阵）**

 由于需要进行的异或运算较多，Hydra将传统的串行计算改为并行计算，大大降低了编码所需的时间（只需7次异或门的时间），使得校验模块的时序性得到了保障。

**图3.4.1(2) 串并行计算校验码耗时比较**

### 3.4.2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Block RAM。

# 4.模块具体实现细节

## 4.1各模块寄存器说明

### 4.1.1 controller(顶层模块)

功能：主控制器模块，连接所有子模块并处理核心逻辑，如匹配SRAM，维护队列等操作。

**表4.1.1(1) controller模块IO口介绍**

| 类型 | IO口名称 | 含义 | 宽 |
| --- | --- | --- | --- |
| input | clk | 模块时钟信号 | 1 |
| input | rst\_n | 模块复位信号 | 1 |
| input | wr\_sop | 开始写入数据包信号 | 16\*1 |
| input | wr\_eop | 终止写入数据包信号 | 16\*1 |
| input | wr\_vld | 写入数据有效信号 | 16\*1 |
| input | wr\_data | 写入数据内容 | 16\*16 |
| output | full | 端口无法为数据包分配更多空间 | 16\*1 |
| output | almost\_full | 端口即将无法为数据包分配更多空间 | 16\*1 |
| output | rd\_sop | 开始读出数据包信号 | 16\*1 |
| output | rd\_eop | 终止读出数据包信号 | 16\*1 |
| output | rd\_vld | 读出数据有效信号 | 16\*1 |
| output | rd\_data | 读出数据内容 | 16\*16 |
| input | ready | 请求开始读出数据包信号 | 16\*1 |
| input | wrr\_en | WRR调度使能信号 | 16\*1 |

**表4.1.1(2) controller模块重要寄存器介绍**

| 寄存器名称 | 含义 | 宽 |
| --- | --- | --- |
| searching\_sram\_index | 端口正在匹配的SRAM | 16\*5 |
| searching\_distribution | 端口当前匹配的最优SRAM | 16\*5 |
| max\_amount | 端口当前最优SRAM的同端口数据量 | 16\*11 |
| search\_cnt | 端口匹配过程计数器 | 16\*6 |
| searching | 端口是否正在匹配 | 16\*1 |
| queue\_head\_sram | 端口各队列头页所在的SRAM | 16\*8\*5 |
| queue\_head\_page | 端口各队列头页的线性地址 | 16\*8\*11 |
| queue\_tail\_sram | 端口各队列尾页所在的SRAM | 16\*8\*5 |
| queue\_tail\_page | 端口各队列尾页的线性地址 | 16\*8\*11 |
| queue\_not\_empty | 端口各队列是否有数据包 | 16\*8\*1 |
| handshake | 读出后端向前端发出的握手信号 | 16\*1 |

### 4.1.2 dual\_port\_sram

功能：伪双口SRAM，提供独立工作的读写口各一个。

**表4.1.2(1) dual\_port\_sram模块IO口介绍**

| 类型 | 名称 | 含义 | 线宽 |
| --- | --- | --- | --- |
| input | clk | 模块时钟信号 | 1 |
| input | rst\_n | 模块复位信号（片选信号） | 1 |
| input | wr\_en | 写入数据使能信号 | 1 |
| input | wr\_addr | 写入数据物理地址 | 14 |
| input | din | 写入数据内容 | 16 |
| input | rd\_en | 读出数据使能信号 | 1 |
| input | rd\_addr | 读出数据物理地址 | 14 |
| output | dout | 读出数据内容 | 16 |

**表4.1.2(2) dual\_port\_sram模块重要寄存器介绍**

| 寄存器名称 | 含义 | 宽 |
| --- | --- | --- |
| d\_latches | SRAM存储的数据 | 16384\*16 |

### 4.1.3 ecc\_encoder

功能：生成一页（128位）数据的8位汉明校验码。

**表4.1.3 ecc\_encoder模块IO口介绍**

| 类型 | 名称 | 含义 | 线宽 |
| --- | --- | --- | --- |
| input | clk | 模块时钟信号 | 1 |
| input | enable | 编码使能信号 | 1 |
| input | data\_0 | 待校验数据#0 | 16 |
| input | data\_1 | 待校验数据#1 | 16 |
| input | data\_2 | 待校验数据#2 | 16 |
| input | data\_3 | 待校验数据#3 | 16 |
| input | data\_4 | 待校验数据#4 | 16 |
| input | data\_5 | 待校验数据#5 | 16 |
| input | data\_6 | 待校验数据#6 | 16 |
| input | data\_7 | 待校验数据#7 | 16 |
| output | code | 生成的校验码 | 8 |

### 4.1.4 ecc\_decoder

功能：根据8位校验码对一页（128位）待纠错数据纠错并输出。

**表4.1.4(1) ecc\_decoder模块IO口介绍**

| 类型 | 名称 | 含义 | 线宽 |
| --- | --- | --- | --- |
| input | clk | 模块时钟信号 | 1 |
| input | enable | 纠错使能信号 | 16 |
| input | data\_0 | 待纠错数据#0 | 16 |
| input | data\_1 | 待纠错数据#1 | 16 |
| input | data\_2 | 待纠错数据#2 | 16 |
| input | data\_3 | 待纠错数据#3 | 16 |
| input | data\_4 | 待纠错数据#4 | 16 |
| input | data\_5 | 待纠错数据#5 | 16 |
| input | data\_6 | 待纠错数据#6 | 16 |
| input | data\_7 | 待纠错数据#7 | 16 |
| input | code | 校验码 | 1 |
| output | cr\_data\_0 | 已纠错数据#0 | 16 |
| output | cr\_data\_1 | 已纠错数据#1 | 16 |
| output | cr\_data\_2 | 已纠错数据#2 | 16 |
| output | cr\_data\_3 | 已纠错数据#3 | 16 |
| output | cr\_data\_4 | 已纠错数据#4 | 16 |
| output | cr\_data\_5 | 已纠错数据#5 | 16 |
| output | cr\_data\_6 | 已纠错数据#6 | 16 |
| output | cr\_data\_7 | 已纠错数据#7 | 16 |

**表4.1.4(2) ecc\_decoder模块重要寄存器介绍**

| 寄存器名称 | 含义 | 宽 |
| --- | --- | --- |
| cr\_data | 已纠错数据 | 128 |
| cur\_code | 待纠错数据对应的校验码 | 8 |
| wrong\_pos | 错误位置 | 8 |

### 4.1.5 fifo\_null\_pages

功能：维护存储空闲页地址的“空闲队列”，实现O(1)的内存回收机制。

**表4.1.5(1) fifo\_null\_pages模块IO口介绍**

| 类型 | 名称 | 含义 | 线宽 |
| --- | --- | --- | --- |
| input | clk | 模块时钟信号 | 1 |
| input | rst\_n | 模块复位信号 | 1 |
| input | pop\_head | 弹出队头使能信号 | 1 |
| output | head\_addr | 当前队头页地址 | 11 |
| input | push\_tail | 插入队尾使能信号 | 1 |
| input | tail\_addr | 插入的页地址 | 11 |

**表4.1.5(2) fifo\_null\_pages模块重要寄存器介绍**

| 寄存器名称 | 含义 | 宽 |
| --- | --- | --- |
| fifo | 队列数据 | 2048\*11 |
| head\_addr | 队列头指针 | 11 |
| tail\_addr | 队列尾指针 | 11 |

### 4.1.6 port\_frontend

功能：对写入模块的数据进行初步处理并缓冲，并指导总控制模块进行匹配操作。

**表4.1.6(1) port\_frontend模块IO口介绍**

| 类型 | 名称 | 含义 | 线宽 |
| --- | --- | --- | --- |
| input | clk | 模块时钟信号 | 1 |
| input | wr\_sop | 开始写入数据包信号 | 1 |
| input | wr\_eop | 终止写入数据包信号 | 1 |
| input | wr\_vld | 写入数据有效信号 | 1 |
| input | wr\_data | 写入数据内容 | 16 |
| input | search\_get | 是否成功匹配到SRAM | 1 |
| output | dest\_port | 当前写入数据包的目的端口 | 4 |
| output | prior | 当前写入数据包的优先级 | 3 |
| output | length | 当前写入数据包的长度 | 9 |
| output | new\_packet\_into\_buf | 当前有新数据包写入（触发搜索） | 1 |
| output | data | 传输至后端的数据内容 | 16 |
| output | data\_vld | 传输至后端的数据有效信号 | 1 |

**表4.1.6(2) port\_frontend模块重要寄存器介绍**

|  |  |  |
| --- | --- | --- |
| 寄存器名称 | 含义 | 宽 |
| buffer | 被缓冲的数据 | 64\*16 |
| xfer\_en | 传输数据至后端使能 | 1 |
| xfer\_ptr | 正在传输的数据位置指针 | 6 |
| wr\_ptr | 正在写入的数据位置指针 | 6 |
| end\_ptr | 未传输完的上一数据包结束位置指针 | 6 |

### 4.1.7 sram\_state

功能：管理ECC存储、跳转表和SRAM端口数据量。

**表4.1.7(1) sram\_state模块IO口介绍**

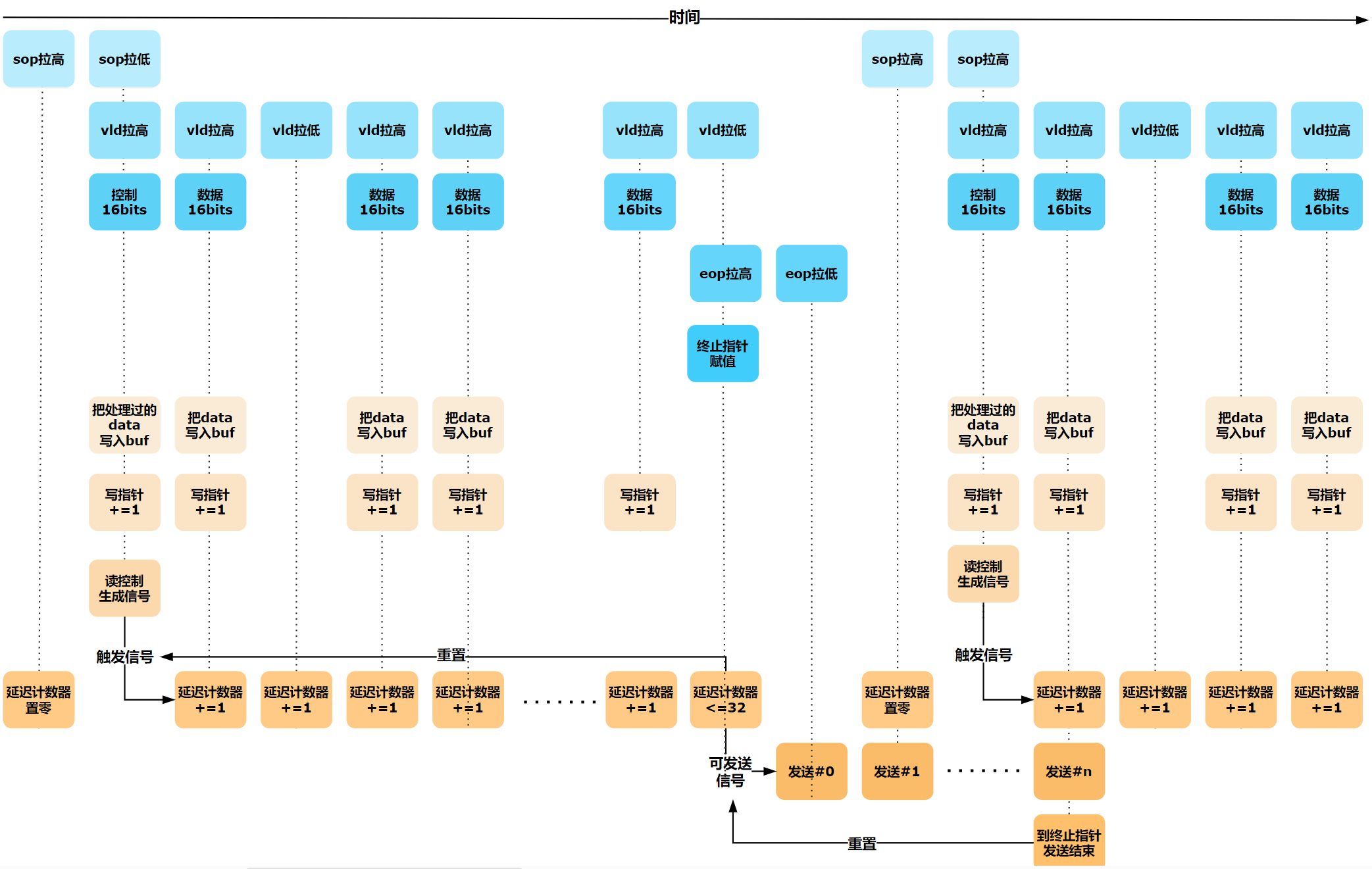
| 类型 | 名称 | 含义 | 线宽 |
| --- | --- | --- | --- |
| input | clk | 模块时钟信号 | 1 |
| input | rst\_n | 模块复位信号 | 1 |
| input | ecc\_wr\_en | ECC存储写入使能信号 | 1 |
| input | ecc\_wr\_addr | ECC存储写入页地址 | 11 |
| input | ecc\_din | ECC存储写入校验码 | 8 |
| input | ecc\_rd\_en | ECC存储读出使能信号 | 1 |
| input | ecc\_rd\_addr | ECC存储读出页地址 | 11 |
| output | ecc\_dout | ECC存储读出校验码 | 8 |
| input | jt\_wr\_en | 跳转表写入使能信号 | 1 |
| input | jt\_wr\_addr | 跳转表写入页地址 | 11 |
| input | jt\_din | 跳转表写入结点内容 | 16 |
| input | jt\_rd\_en | 跳转表读出使能信号 | 1 |
| input | jt\_rd\_addr | 跳转表读出页地址 | 11 |
| output | jt\_dout | 跳转表读出结点内容 | 16 |
| input | wr\_op | 写单页信号 | 1 |
| input | wr\_or | 写数据包信号 | 1 |
| input | wr\_port | 写入数据包的输出端口 | 4 |
| input | delta\_free\_space | 写入数据包时剩余空间改变量 | 12 |
| input | delta\_page\_amount | 写入数据包时端口统计该变量 | 12 |
| input | rd\_op | 读单页信号 | 1 |
| input | rd\_port | 读出数据包的输出端口 | 4 |
| input | rd\_addr | 读出的页地址 | 11 |
| input | request\_port | 请求查询数据量的输出端口 | 4 |
| output | page\_amount | 查询到的打开看看数据量 | 11 |
| output | null\_ptr | 空闲页指针 | 11 |
| output | free\_space | 剩余可写页 | 11 |

**表4.1.7(2) sram\_state模块重要寄存器介绍**

| 寄存器名称 | 含义 | 宽 |
| --- | --- | --- |
| ecc\_storage | ECC存储 | 2048\*8 |
| jump\_table | 跳转表 | 2048\*16 |
| port\_amount | 各端口的数据量 | 16\*11 |

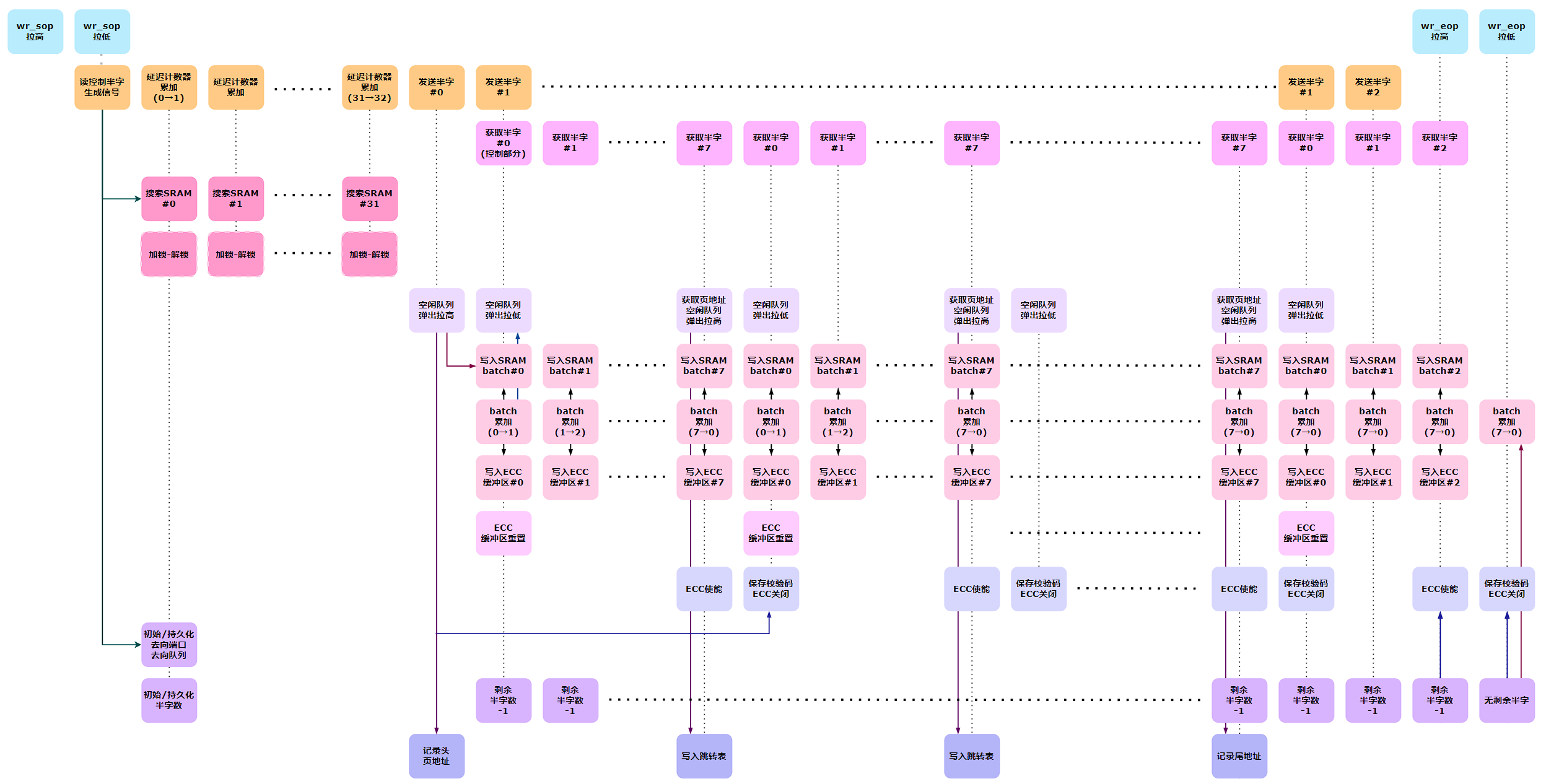
## 4.2重要逻辑说明

### 4.2.1写前端流水线



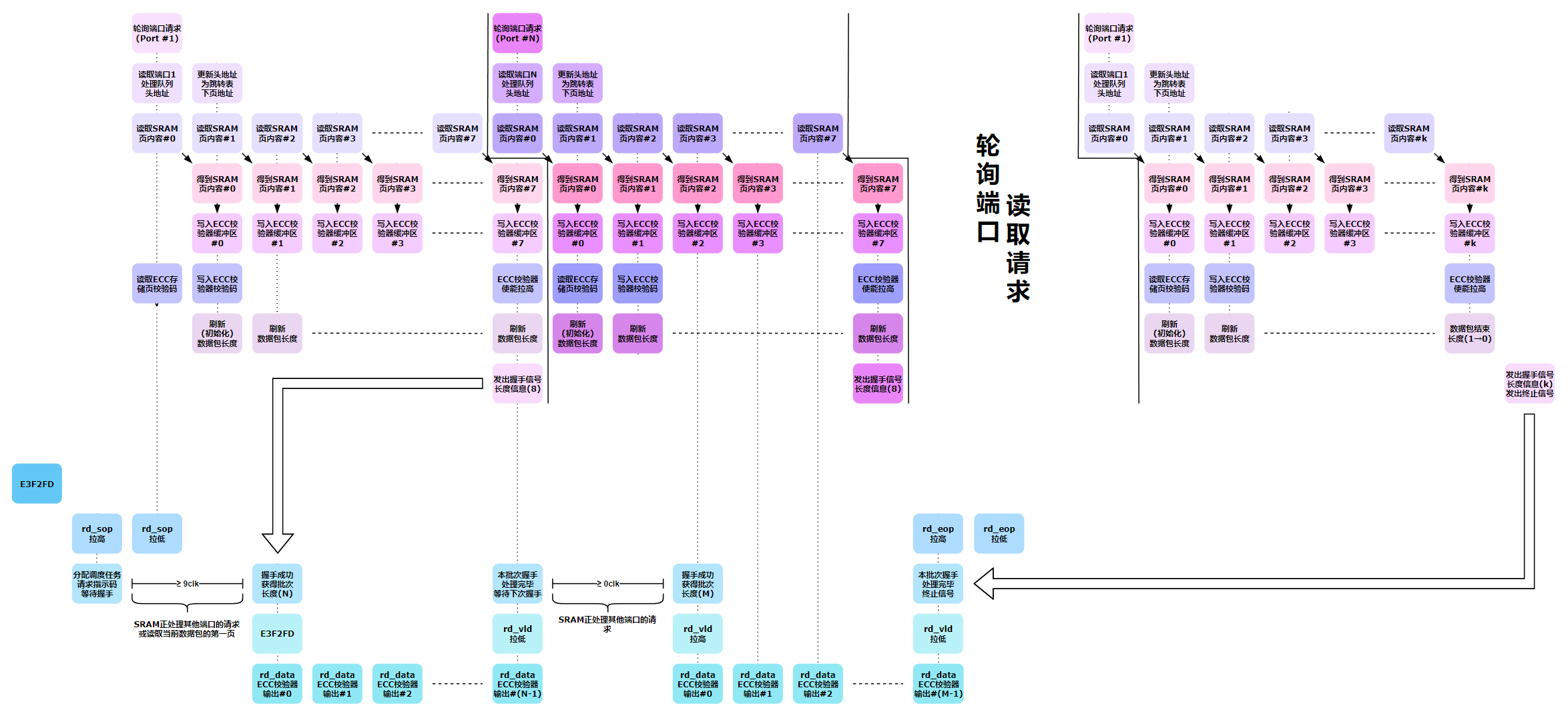
**图4.2.1 写前端流水线示意图 写控制IO口 前端逻辑**

### 4.2.2写后端流水线



**图4.2.2 写后端流水线示意图 写控制IO口 前端信号 后端逻辑**

### 4.2.3读前后端流水线



**图4.2.3 读前后端流水线示意图 读控制IO口/前端逻辑 后端逻辑**

# 5.接口与配置

## 5.1.写控制IO口

### 5.1.1 IO口介绍

**wr\_sop**：拉高1周期表示开始写入数据包。

**wr\_eop**：拉高1周期表示结束写入数据包。

**wr\_vld**：表示当前通过wr\_data写入的一半字数据有效。

**wr\_data**：写入一半字数据的内容，仅在wr\_vld为高的时候被认为有效。

### 5.1.2 使用方法

**数据包格式**：

控制部分（1半字）和数据部分（31~511半字）。

其中控制部分为16位，描述三个数据包信息：

1）高9位表示数据包的长度（半字）；

2）低4位表示目的端口；

3）剩余3位表示优先级。

**注意事项**：

1）wr\_sop和wr\_eop不能同时拉高。

2）wr\_vld只能在wr\_sop为高后、wr\_eop为高前的一段时间内为高。

## 5.2写反馈IO口

### 5.2.1 IO口介绍

**full**：当所有SRAM被占用/满时拉高，或端口无法匹配到可存储新包的SRAM时为高。

**almost\_full**：当端口没有占用SRAM，且可用SRAM数量小于端口数量时拉高。

### 5.2.2 使用方法

当某个端口的full拉高时，端口不应写入任何数据，正在写入的数据包应当立即停止写入，在合适的时机从头重新写入（已经写入的在缓冲里的数据将被“冲刷”）。

当某个端口的almost\_full拉高时，则做好准备在写入过程中拉高full的“冲刷”操作。

## 5.3读控制IO口

### 5.3.1 IO口介绍

ready：请求读出一个数据包的信号。

### 5.3.2 使用方法

ready拉高时，若该端口存在可读取的数据包，rd\_sop将会在下一周期立即拉高，表示开始读出数据包。读出数据包的过程中拉高ready无效，即只有在前一个数据包rd\_eop拉高后，拉高ready才会触发后一个数据包的读出。

## 5.4读反馈IO口

### 5.4.1 IO口介绍

**rd\_sop**：拉高1周期表示开始写入数据包。

**rd\_eop**：拉高1周期表示结束写入数据包。

**rd\_vld**：表示当前通过rd\_data写入的一半字数据有效。

**rd\_data**：写入一半字数据的内容，仅在rd\_vld为高的时候被认为有效。

### 5.4.2 使用方法

数据包格式与sop、eop、vld和data信号线的机制与写入时相同。

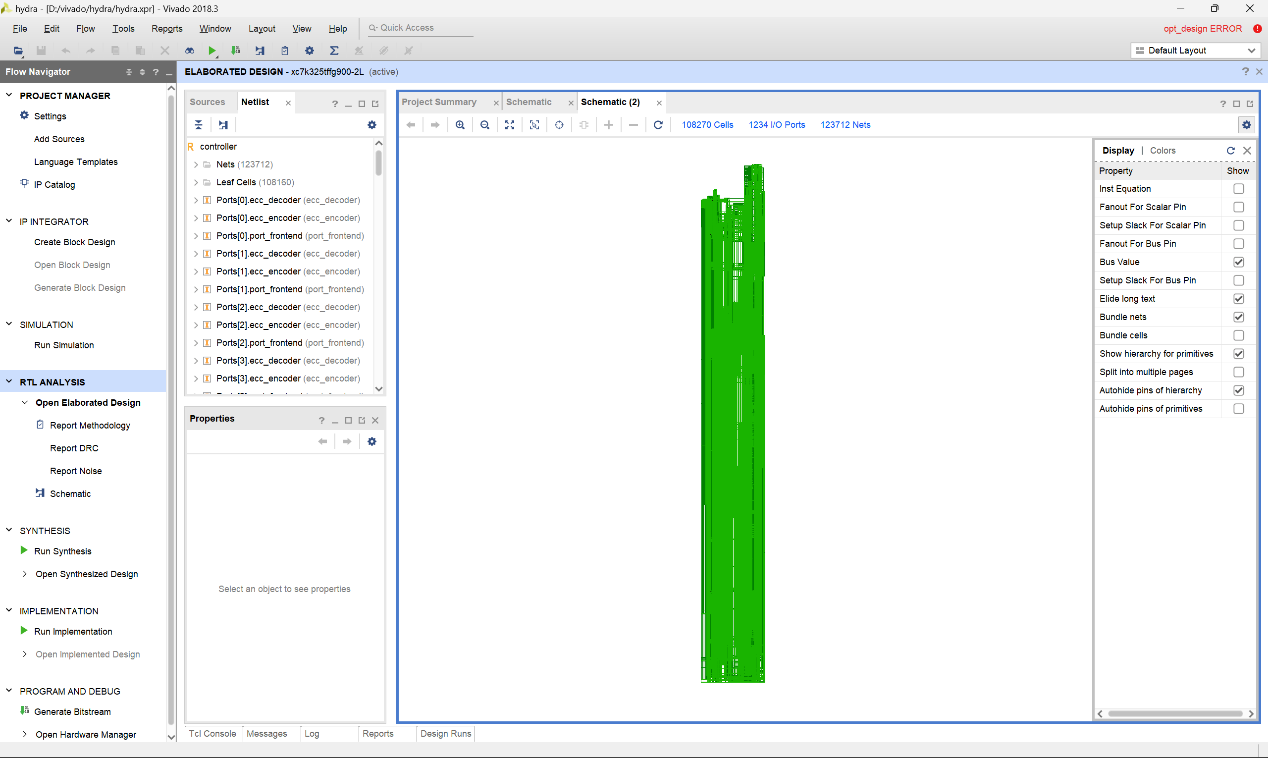
## 5.5 WRR使能配置

**wrr\_en**：每个端口各1个，表示是否开启该端口的WRR调度模式。

# 6.验证方法

## 6.1 RTL级仿真

### 6.1.1 RTL级电路图



**图6.1.1 RTL级电路**

### 6.1.2单模块仿真

见附件中的.tb测试文件和.wlf/.do仿真波形文件。

### 6.1.3总模块仿真

见附件中的.tb测试文件和.wlf/.do仿真波形文件。

## 6.2综合验证

### 6.2.1 综合环境

**操作系统**：Microsoft Windows 11

**Vivado版本**：2018.3

**板卡系列**：Kintex-7

**板卡型号**：xc7k325tffg900-2L

**片上资源**：**IO Pin**：900

**LUT**：203800

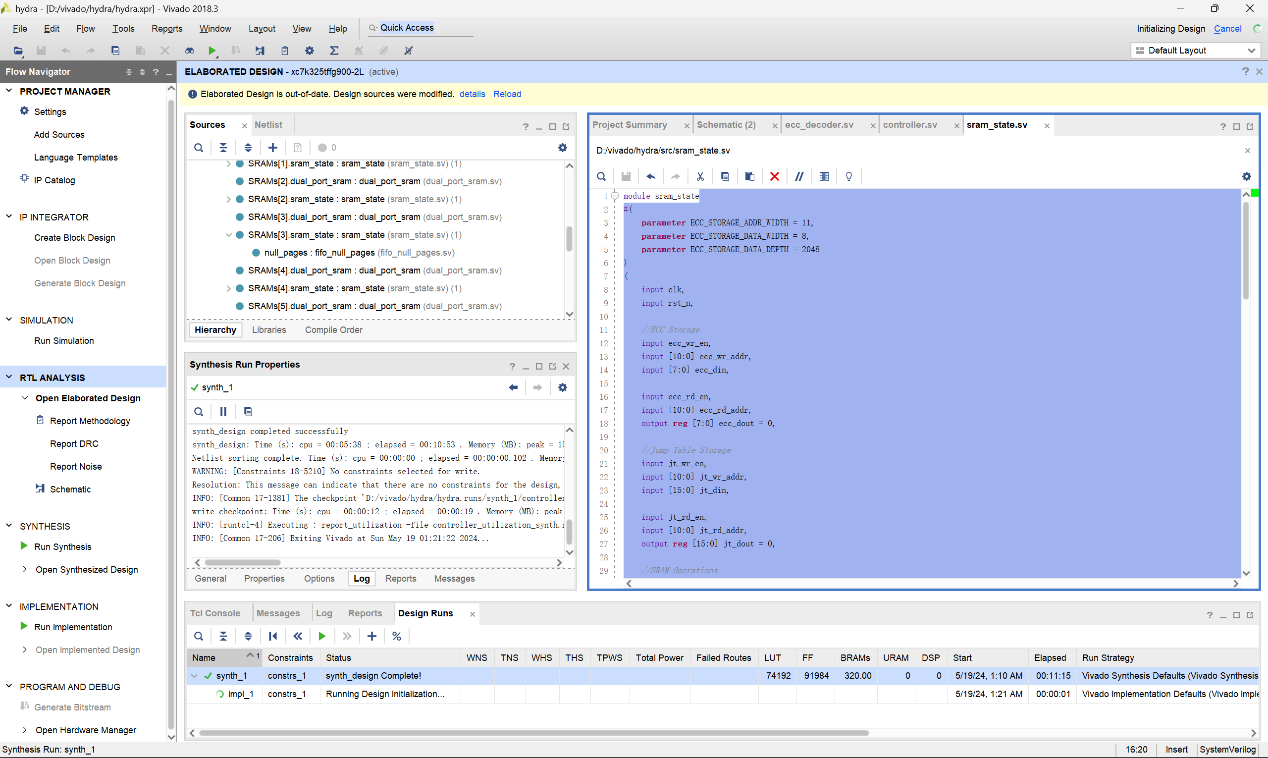
**FlipFlops**：407600

**Block RAMs**：445 \* 36Kbits

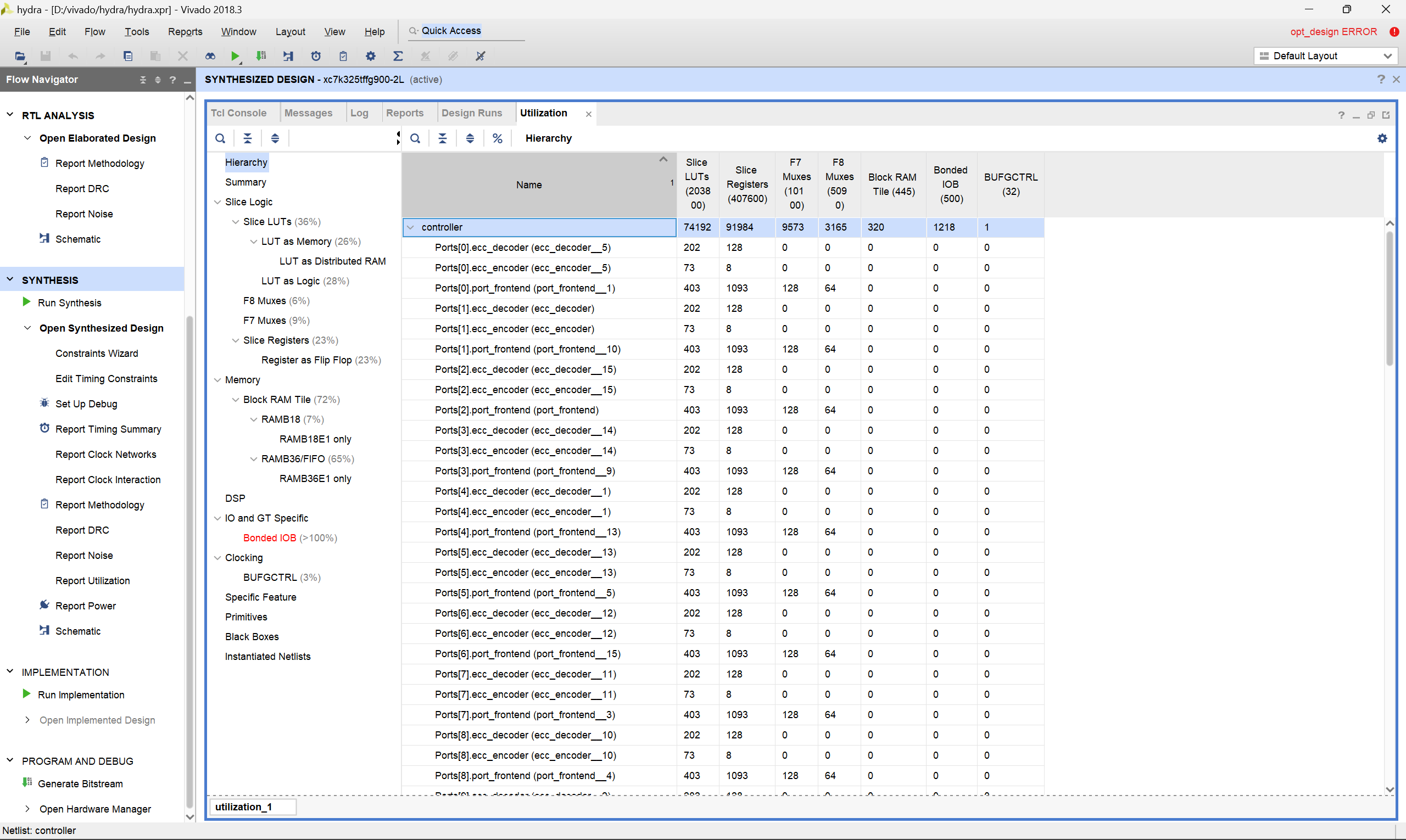
**Ultra RAMs**：0

**DSPs**：840

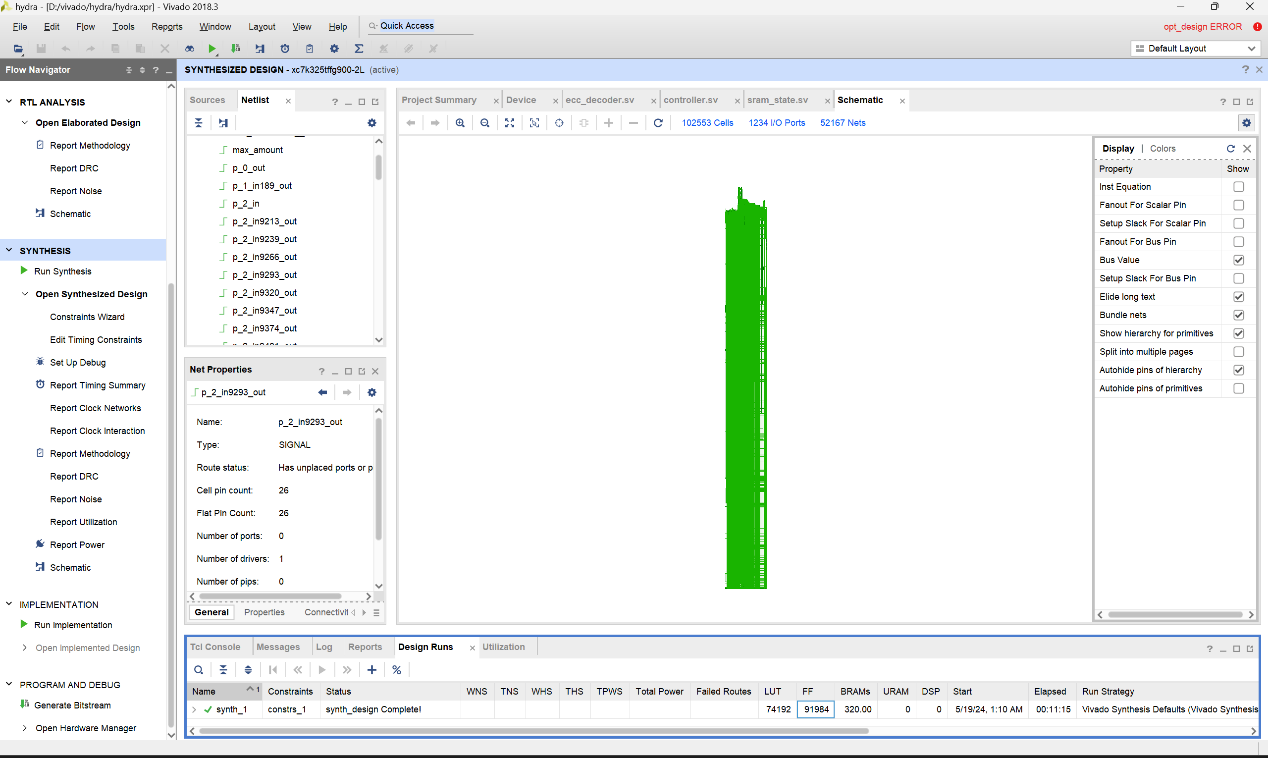
### 6.2.2 综合结果



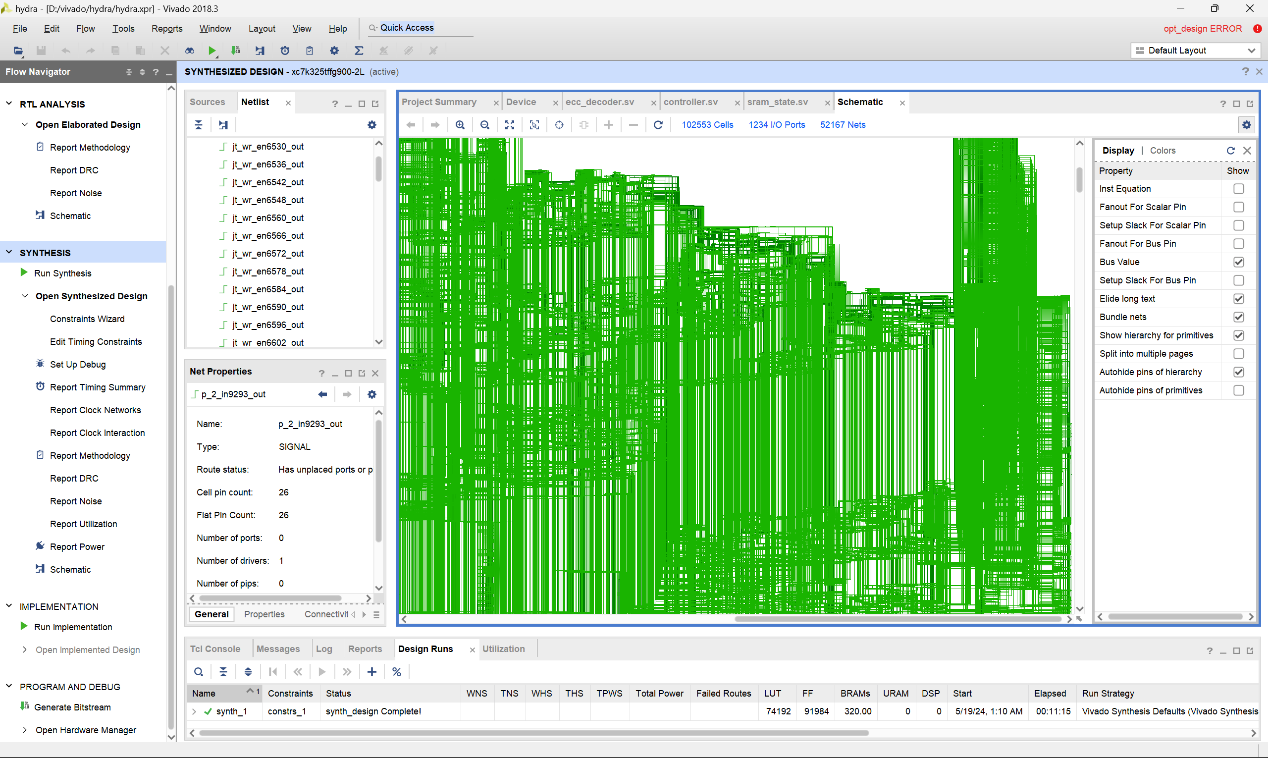
**图6.2.2(1) 综合成功结果**



**图6.2.2(2) 模块资源占用**



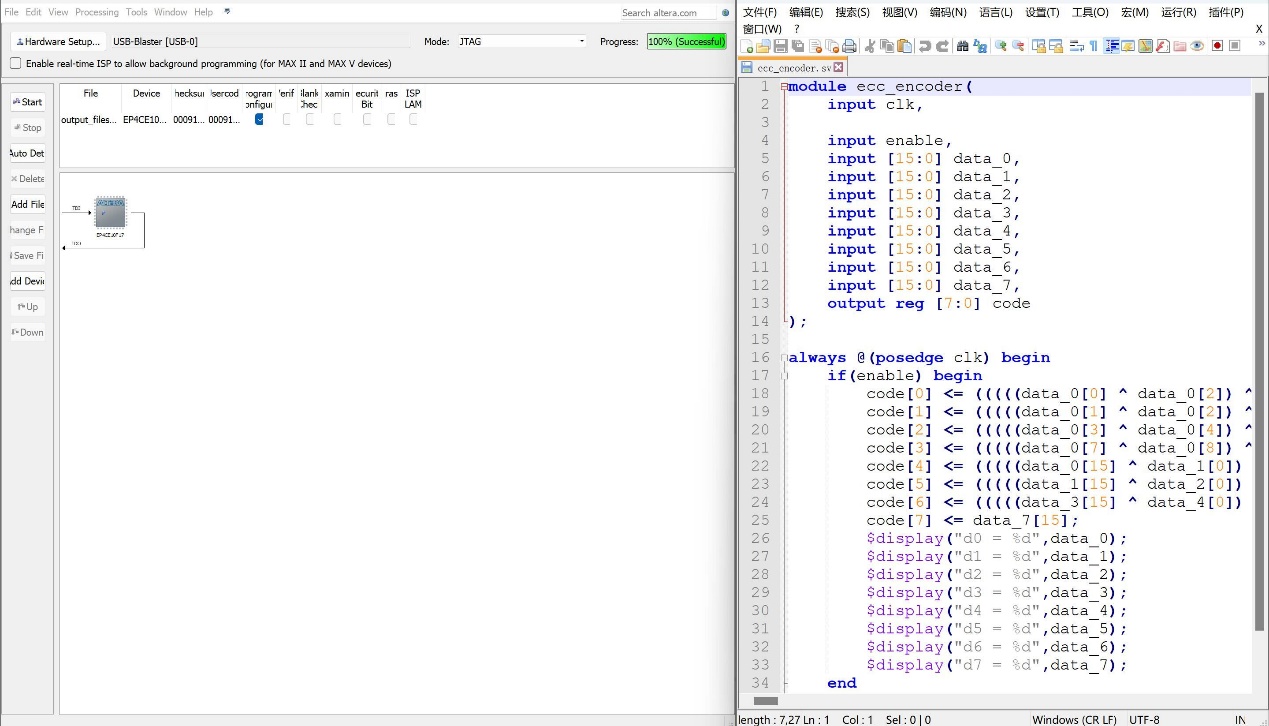
**图6.2.2(3) 综合电路（全貌）**



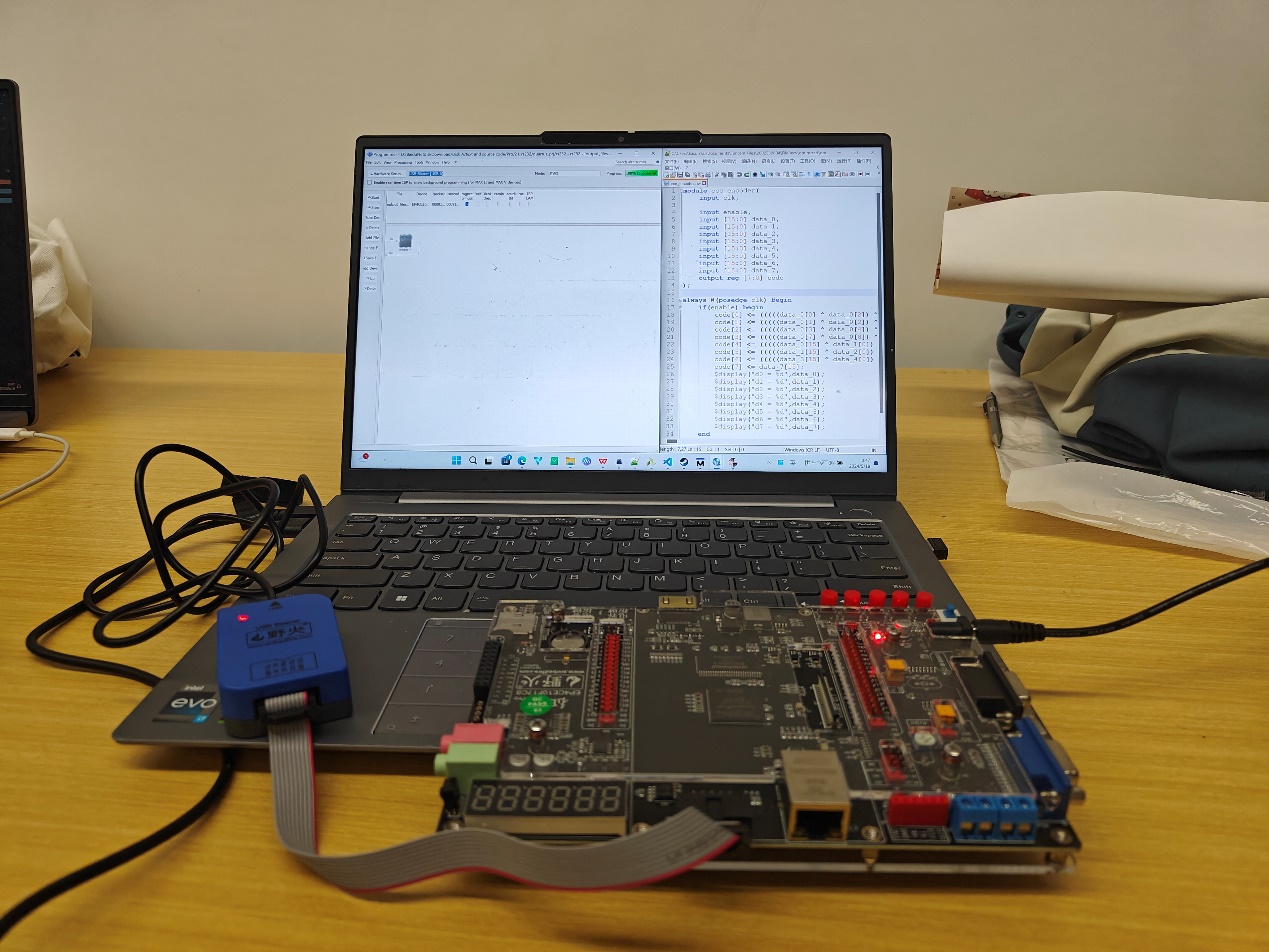
**图6.2.2(4) 综合电路（放大）**

## 6.3 FPGA验证

### 6.3.1 Ecc校验编解码模块



**图6.3.1(1) 程序烧录截图**



**图6.3.1(2) FPGA板运行图**

# 7.设计优缺点

## 7.1优点

1）完全动态分配内存，各端口、队列数据量无限制；

2）引入数据结构与算法思想，降低逻辑复杂度；

3）引入页表管理的思想，进一步划分SRAM，减少了存储时需要的控制信息；

4）时序性良好，使用跳转表替代时序性差的位图，各模块机制流水化；

5）偏好性存储降低读取延迟，使得大多数情况下端口读取带宽可以达到理论峰值；

6）在有可用SRAM时保证端口写入带宽总为理论峰值；

7）极大压缩模块的存储资源占用（相较于传统方案而言）；

8）代码风格良好，严格遵循命名规范，注释清晰。

## 7.2缺点

1）匹配过程使得数据包写入延迟32~48周期；

2）页表管理导致存储数据包时，SRAM部分空间（<5%）被暂时性浪费；

3）未匹配到SRAM时，写入强行终止，已被写入前端缓冲区的一部分数据会被冲刷。

# 8.后续开发计划

## 8.1 pause接口

见缺点3），当一个数据包开始发送时，Hydra会启动搜索，尝试匹配合适的SRAM对其进行存储，若未匹配到，则会拉高端口的full接口，强行终止此次发送，已发送进缓冲区的数据将会被冲刷。

Hydra未来会引入pause接口，一个新的写反馈接口，当出现上述情况时，pause将会被拉高，暂停本次传输，并一直启动搜索，直到原来被占用/满的SRAM被释放，匹配到合适的SRAM，pause被拉低，本次传输继续。这样不会出现在传输过程中拉高full接口，避免模块内部数据冲刷、模块外部设备重新发送数据包的复杂逻辑。

## 8.2读写延迟优化

Hydra未来会进一步优化流水线，并简化冗余的时序逻辑，预计可降低完全动态分配下读写延迟。

## 8.3 半动态模式

Hydra未来会引入新模式，旨在提供一种介于完全动态分配与静态分配内存之间的一种策略，即半动态模式。具体的设计为每个端口绑定一块SRAM，并共享剩余的16块SRAM。

半动态模式在保证带宽不变的前提下，牺牲单端口数据容量，极大减少读写延迟。在部分场合比完全动态模式表现更佳。

## 8.4模块化框架

Hydra未来将会拆解主控制模块，进一步提升代码的层次性与可读性。