**部分总结**

大致理解了gemmini的构成，大致如下：

图示

描述已自动生成

1. **Controller Module 部分：**

图示

描述已自动生成

该部分主要负责Rocc指令的展开，Controller接收cmd后存入缓存队列，之后cmd通过LoopConv和LoopMatmul函数展开（暂时不知道展开的具体过程和结果，猜想是将一个矩阵乘法指令拆解为一系列load，store和execute的rocc指令），展开后的结果依旧是rocc指令，进入重排序缓存端元（ROB）Rocc的定义信息可以在”rocket-chip/src/main/scala/tile/LazyRoCC.scala”中找到，成分如下：

文本

描述已自动生成

为常量64

ROB会将指令发送至load\_controller单元，store\_controller单元以及execute\_controller单元。Load\_controller和store\_controller内部完成解读rocc指令的过程。其中load controller 和 store controller分别和Spad的dma控制单元交互。Spad的dma单元则通过内部writer与reader单元处理dma读写请求，wrtier与reader单元通过TLB页表缓存来将输入的虚地址（vaddr）与DRAM物理地址对应，从而完成片上与片外的交互。Execute\_controller 单元的流程还未分析，已知包含mesh阵列，完成脉动阵列计算，其接收来自Spad的数据。

1. **Spad 部分：**

**图示

描述已自动生成**

Spad的IO由3个控制单元：dma，srams和acc，TLB页表缓存和flush信号构成。Dma，srams和acc单元都有类似的req和resp两部分构造，req接收来自外部的请求，在spad内部完成后由resp输出信号。

Spad内部包含Spad\_banks, acc\_banks两种bank，与片外存储交互的reader和writer单元，vect\_mul单元和zero\_writer单元。Spad\_banks有4个，分别用于存储A,B,C,D四个矩阵。Acc\_banks有2个

Spad\_banks处理来自srams unit的读写请求，acc\_banks处理来自acc unit的请求。其输出会通过延迟队列分别连接到 srams的resp 和 acc的resp。另外，banks的输出都会通过WriteData单元输出给writer单元。

Dma unit的读写指令会先通过缓冲队列，缓冲队列会判断输入指令是否为garbage指令来判断是否需要0延迟传递garbage指令。之后如上文所述，dma读写指令会进入对应的writer和reader单元完成数据的读写。其中，reader会后接vec\_mul单元（应该是在一定条件下会进行向量乘法运算），vec\_mul单元会将结果输给sp\_bank和acc\_bank单元的写输入。dma读请求也会发送给zero\_writer单元，zero\_writer单元接acc\_banks和spad\_banks的写单元进行对应地址的清零操作，其输出会发送给vec\_mul单元。

1. **Controller 中的load controller 部分：**

**图示

描述已自动生成**

内部通过有限状态机控制单元的运作流程，controller的IO接口主要为指令输入和dma IO单元。Dma IO单元在controller的代码文件中完成和spad.dma.read IO单元的连接。

Controller内部主要为cmd\_tracker单元负责缓存cmd指令。有限状态机有三个状态等待指令（waiting\_for\_cmd）、等待dma请求处理（waiting\_for\_dma\_req）和发送行（sending\_rows）。

在waiting\_for\_cmd阶段，load controller会根据指令的funct参数判断是配置自己还是执行加载任务，在通过spad.dma unit执行加载任务之前，会将指令包含的总数据量（单位为字节）写入cmd\_tracker，进行登记。若内部有空间存储，则cmd\_tracker会在空闲的队列中存储该指令的总数据量并返回位置信息，cmd\_tracker的输出作为spad.dma.read.req的一个特征之一存储（用于标记）。若cmd\_tracker无内部空闲，则状态机保持直至cmd\_tracker有空闲。完成登记后机器根据上一个dma.read.req是否处理完毕进入waiting\_for\_dma\_req和sending\_row阶段

在waiting\_for\_dma\_req阶段，load Controller会等待SRAM处理上一个dma.read.req，完成后进入sending\_row阶段。

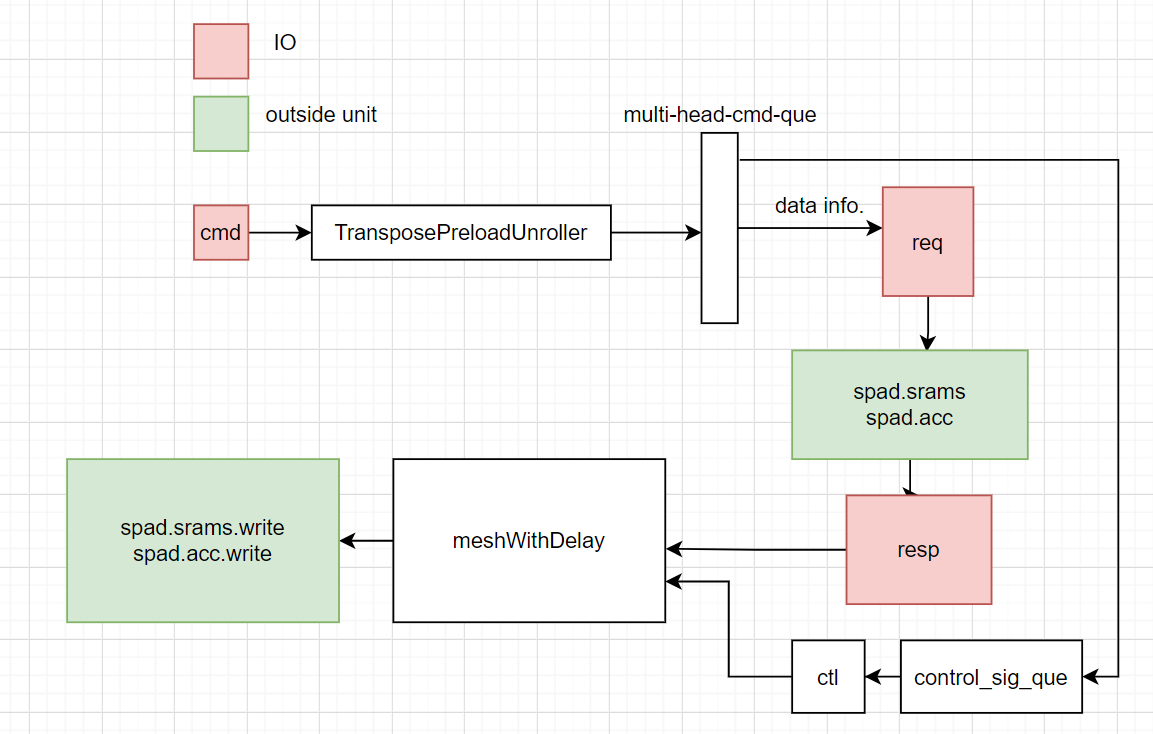
在sending\_row阶段，controller会根据是否最后一行来判断是否进入waiting\_for\_cmd阶段。Controller内部有row\_counter变量用来帮助controller判断是否为最后一行。row\_counter通过wrap add的机制将存入数据的位置控制在相对于vaddr下一定范围的栈内（但初始位置不确定）。每完成指令内部一行数据的load任务，dma.resp会从cmd\_tracker中减去对应数量的字节数。

1. **Exec\_controller**

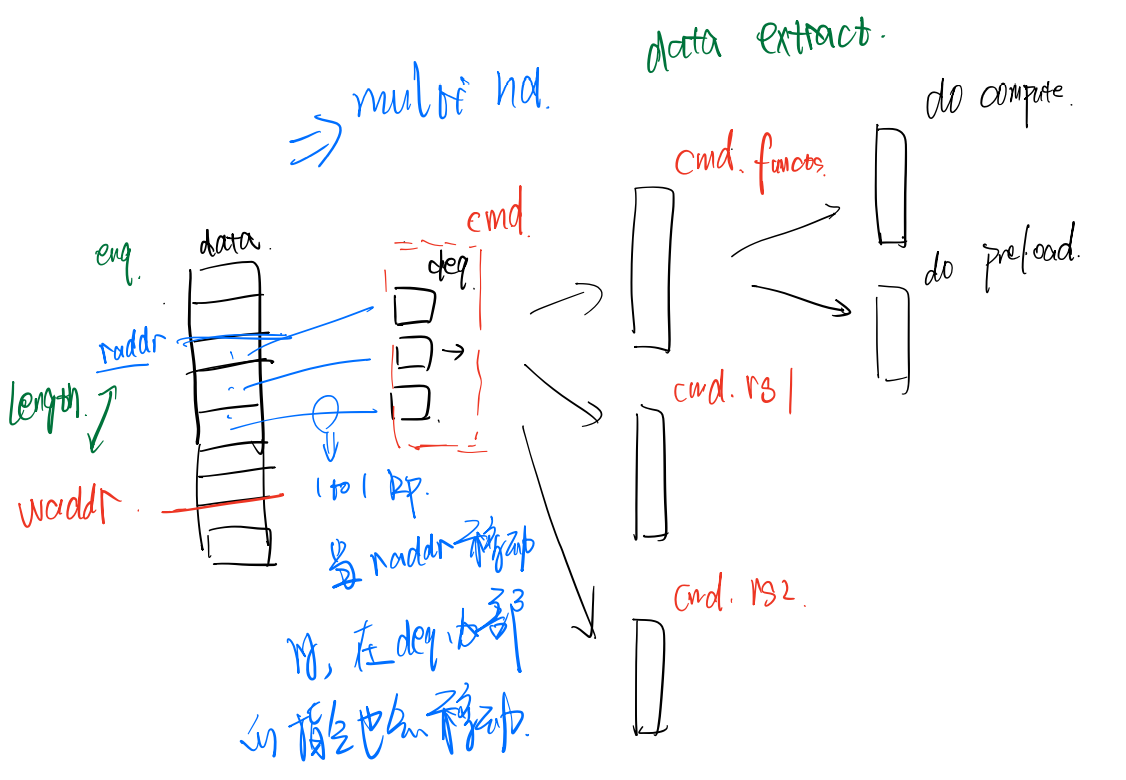
IO：input : resp from im2col and spad, cmd. Output: req to im2col and spad, complete signal.

FSM实现过程划分，waiting\_for\_cmd阶段：根据指令的类型判断是否进行config或指定compute状态的工作。或处于等待阶段（flush）直到脉动矩阵mesh完成计算。在compute阶段，会根据waiting\_for\_cmd阶段的控制信号执行overlap\_compute，single\_mul compute或Do\_preloads，从multi-head-cmdQue中提取所需的指令数量。

Multi-head-cmdQue，cmd缓冲区，cmd经过TransposePreloadUnroller 单元展开后会存入multi-head que中，que根据config被设置为3 head，其deq中存储raddr及其一下的两条地址的共三条指令（为一组），每次FSM中的compute阶段会执行其中的1-2条指令（为什么是直接访问而不是参数访问就很神奇）。提取出的指令为根据ISA的形式提取数据（a,b,d矩阵的地址）发送给req，进行与spad.srams或spad.acc的交互，spad的结果经由resp传输给输入，经过超出部分归零处理后进入脉动阵列计算。脉动阵列同时接收一个控制信号来标明存储的位置。



Exec\_contoller



Multi-head

1. **MushWithDelay:**

脉动阵列的最终形式，其中的shift函数实现的是将一个矩阵输入变成1周期延迟的形式。

文本, 信件

描述已自动生成

**计划阅读的部分**

1. **Controller中的两个loop函数**
2. **Exec\_controller中的unroll单元工作流程**