## 智能ip

初期制定了两套方案，其中方案一是基于传统算法，内部有状态机控制，自动流水处理，该方案已经完成；

项目大部分时间都在完成方案二的设计，方案二是基于卷积神经网络（CNN）的通用AI加速方法，拟设计一个基于输入指令控制的通用卷积神经网络推理加速硬件，可支持任意BP神经网络和基本卷积神经网络，如Alexnet、VGGnet，支持不规则取数（借此支持空洞卷积）支持各种尺寸的卷积核（3\*3、5\*5），不过该架构数据的调度非常复杂，资源消耗大、几乎用了全部DSP，在200M时钟下理论峰值算力达到约20GFLOPS，大部分卷积神经网络除了第一层只能达到37.5%算力利用率以外，其余层基本都能达到80%~100%利用率。目前各模块rtl已写完，但还未进行功能、时序验证和顶层封装验证和debug。

|  |  |  |
| --- | --- | --- |
| 方案 | 优点 | 缺点 |
| 方案一（基于传统算法） | 算法简单、计算量少、易于硬件实现 | 准确度相对低一点、泛化性不足 |
| 方案二（通用AI加速） | 准确率高、泛化性强 | 计算量大、缓存容量需求大、数据调度复杂 |

### 方案一（传统算法）

#### 方案简介

该方案是为了防止AI方案不能正常完成而预先制定和完成的一个备用方案，整体策略相对简单。该方案由内部状态机自行控制，通过二值化节省硬件资源、并行处理提高处理速度，典型工作流程为：

读取摄像头写入的图片数据；

根据亚洲人人体肤色的常见RGB值范围，对图片数据进行二值化；

使用模板匹配法将二值化后的数据与所有模板数据进行匹配，计算出误差；

通过比较器模块比较出误差最小的数diff\_min和模板id；

最后通过阈值筛选，当diff\_min小于阈值，则成功检测到手势，通过id得到匹配结果，并通过AHB接口传出分类结果，当diff\_min大于阈值，则认为当前未检测到手或者手势无效。

#### 架构介绍

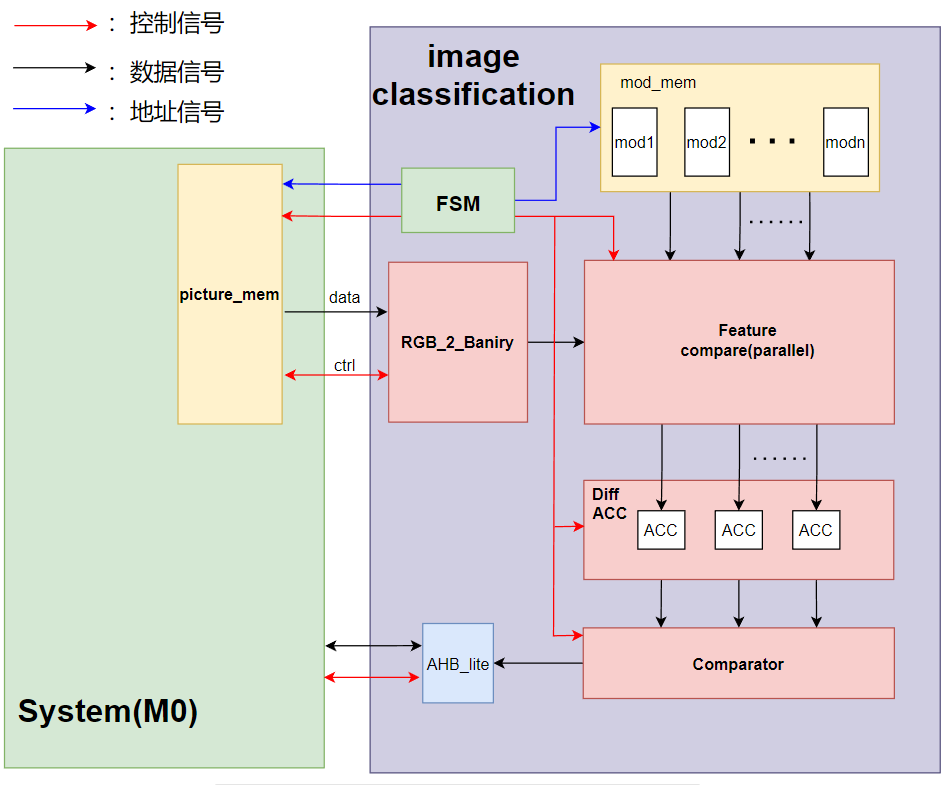
顶层原理框图如下所示，该方案设计的硬件部分为紫色框部分。

自动从picture\_mem中读取图片数据，每拍读16个16bit数（RGB565），通过RGB2B模块完成肤色筛选和二值化得到1个16bit数据B，同时从模板存储mod\_mem中每拍读出对应像素位置的n个16bit数MOD；

在Feature compare模块中，B分别和n个MOD数据同时并行处理，得到这一部分的差异值diff\_p；

在读取完一帧图像之前，diff\_p会在Diff ACC模块中进行累加，统计整幅图像与各模板数据的差异值diff；

通过Comparator模块计算出最小的diff值，并与阈值对比，最后得到分类结果并向系统发出检测完成信号，输出结果会在下一帧检测完成之前被寄存。



### 方案二（AI ，CNN加速）

#### 2.1 方案简介

该方案的设计目标是一个通用的卷积网络加速器，通过指令调度完成卷积网络推理，通过配置不同指令来完成不同尺寸特征图、不同个数卷积核的卷积运算，从而实现各种基本的卷积神经网络加速。该方案特点如下：

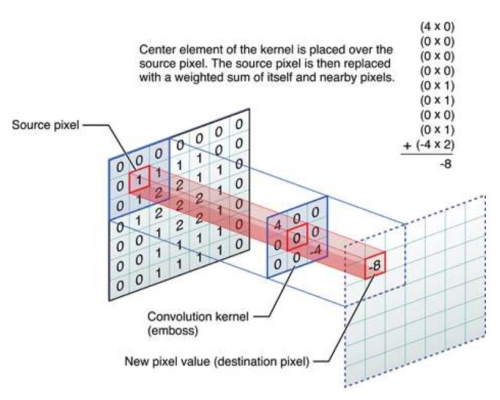
1. 参考了谷歌TPU的脉动阵列架构以加速卷积运算；
2. 采用int8量化方案大幅减少网络参数数据量和计算资源需求；
3. 根据该项目采用的摄像头的特点、游戏需要的手势识别功能等应用场景采集自定义数据集；
4. 根据板子dsp数量和存储的资源设计合适的自定义神经网络结构，采用深度可分离卷积以减少缓存的需求和中间数据累加位宽需求；
5. 整个电路通过指令包控制，没有设置状态机，支持不规则取数。

（业界的AI项目通常会配套AI编译器，编译器读入神经网络模型文件，解析后得到对应其AI硬件的机器码，但在本项目中考虑到时间有限且只需做一个demo，选择不做配套编译器，而是直接选定某一个神经网络后，手动分析出其对应与本项目AI加速硬件的指令包。）

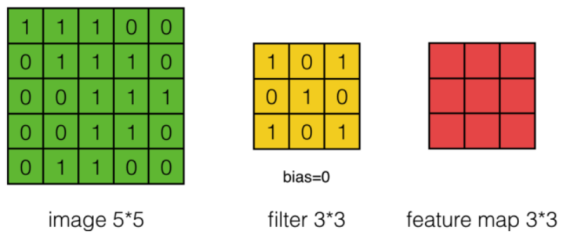
#### 2.2 基本原理介绍

##### 2.2.1 卷积计算和卷积神经网络

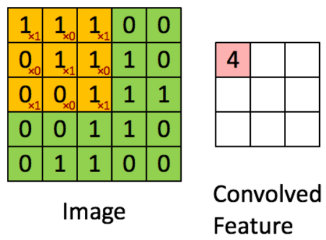
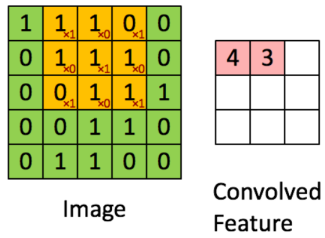
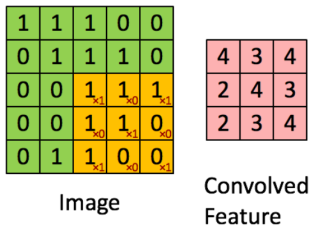
AI中的卷积计算指2D卷积，其最基本运算是一个k\*k的卷积核和一个k\*k的输入数据对应位置相乘再累加，下图为一个3\*3单次卷积，输出一个结果-8，神经网络中卷积核会滑动遍历一个n\*n的图像，最终得到（n-2）\*(n-2)的新featuremap。



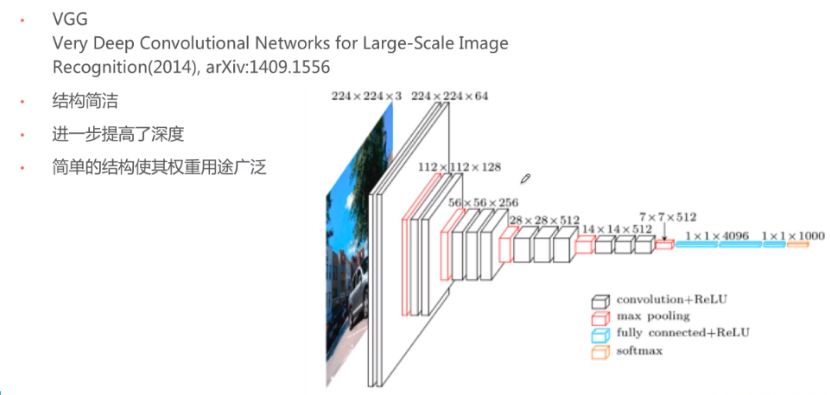
对于一张输入图片, 将其转化为矩阵, 矩阵的元素为对应的像素值. 假设有一个5 × 5的图像，使用一个3 × 3的卷积核进行卷积，可得到一个3 × 3的特征图。



具体过程如下：

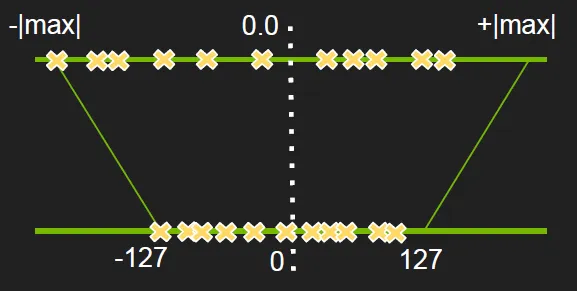
  

下图为典型卷积神经网络VGG16的结构图，整个网络有13层卷积层和3层全连接层，模型参数数量约138M，对应存储为138M\*4bytes，推理一张图片对应的计算量约30.96GFlops。



##### 2.2.2 int8量化

由上可知神经网络模型参数量和计算量通常非常大，所以业界提出了低精度量化方法来减少存储需求和计算资源，典型的是int8量化，统计出feature map的最大值max后，将浮点数映射为-128~127的8位定点数。



用r指代量化前数据，q指代量化后数据：

对一个数据量化：\*q ，S即为Scale系数，S= 。

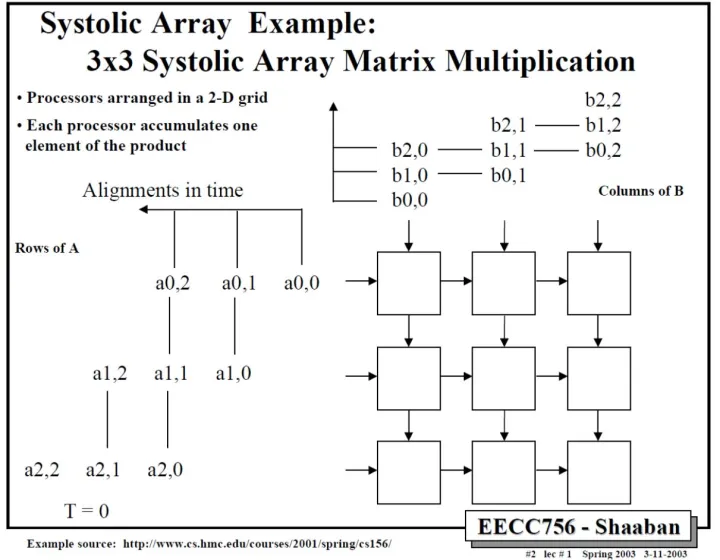
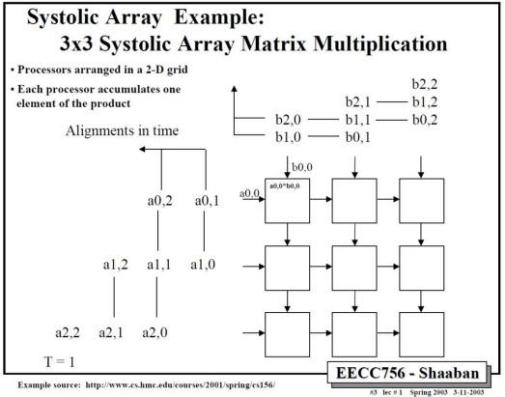
量化前一次卷积：

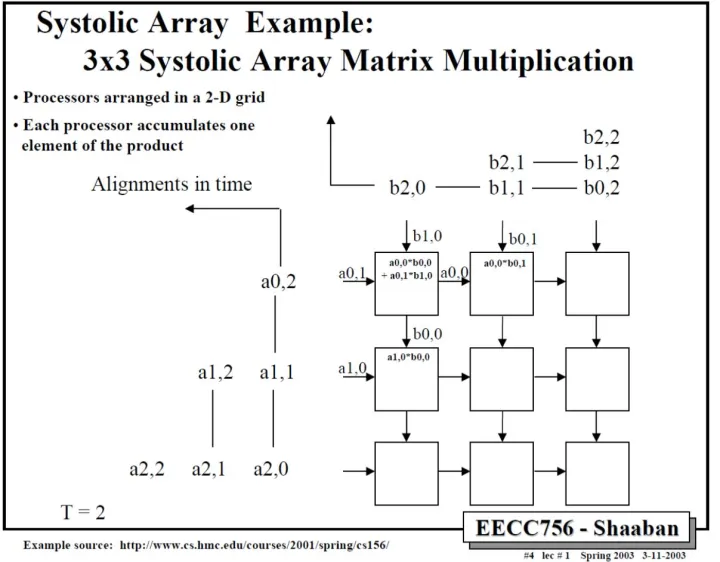
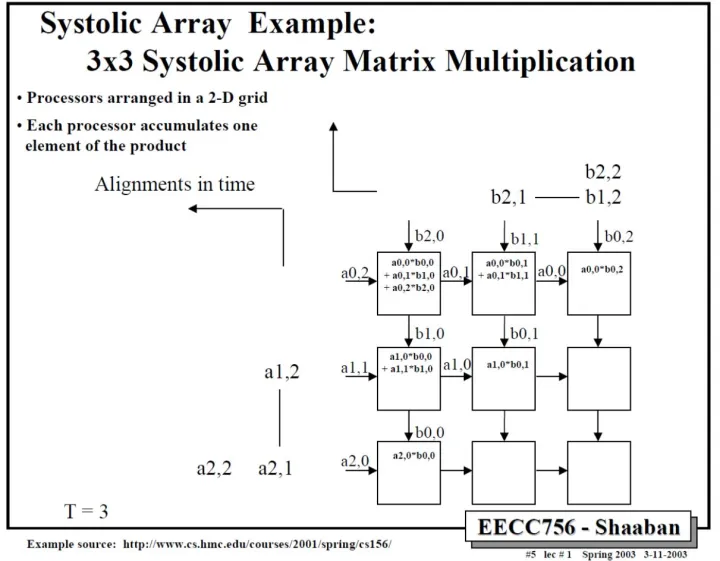
量化后：

= S\*

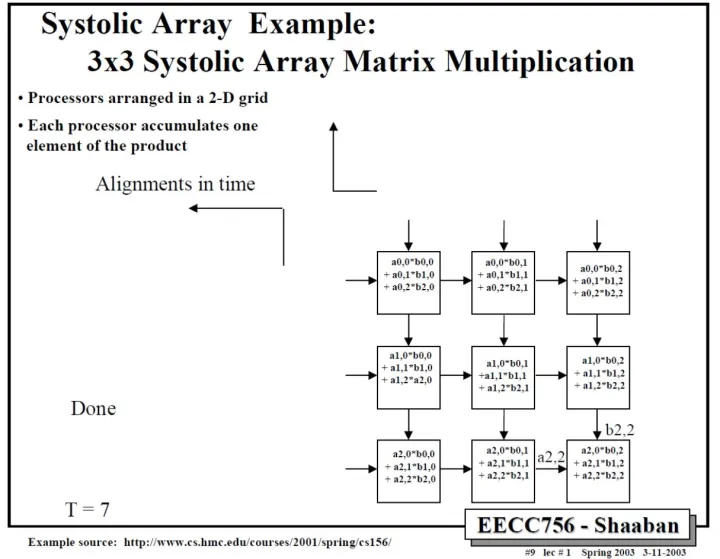
##### 2.2.3 脉动阵列

脉动阵列提出与上世纪80年代，可用于矩阵乘，下面介绍一个脉动阵列实现矩阵乘的例子，A、B都是3\*3矩阵，T代表时钟。

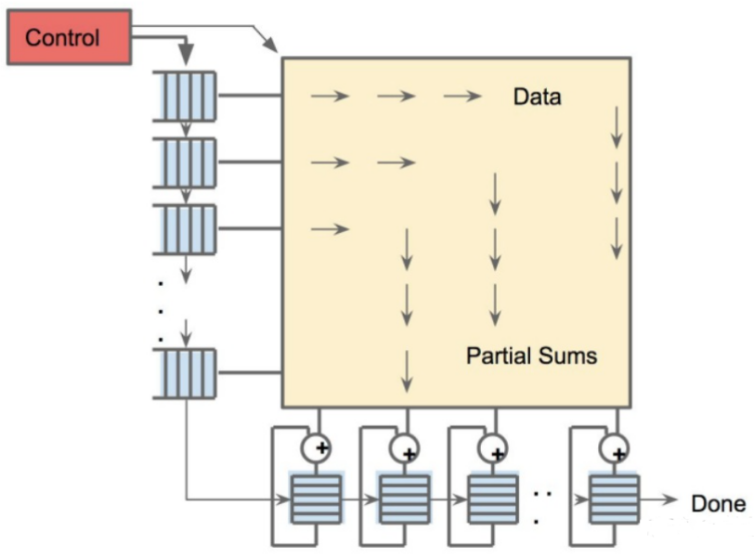
 

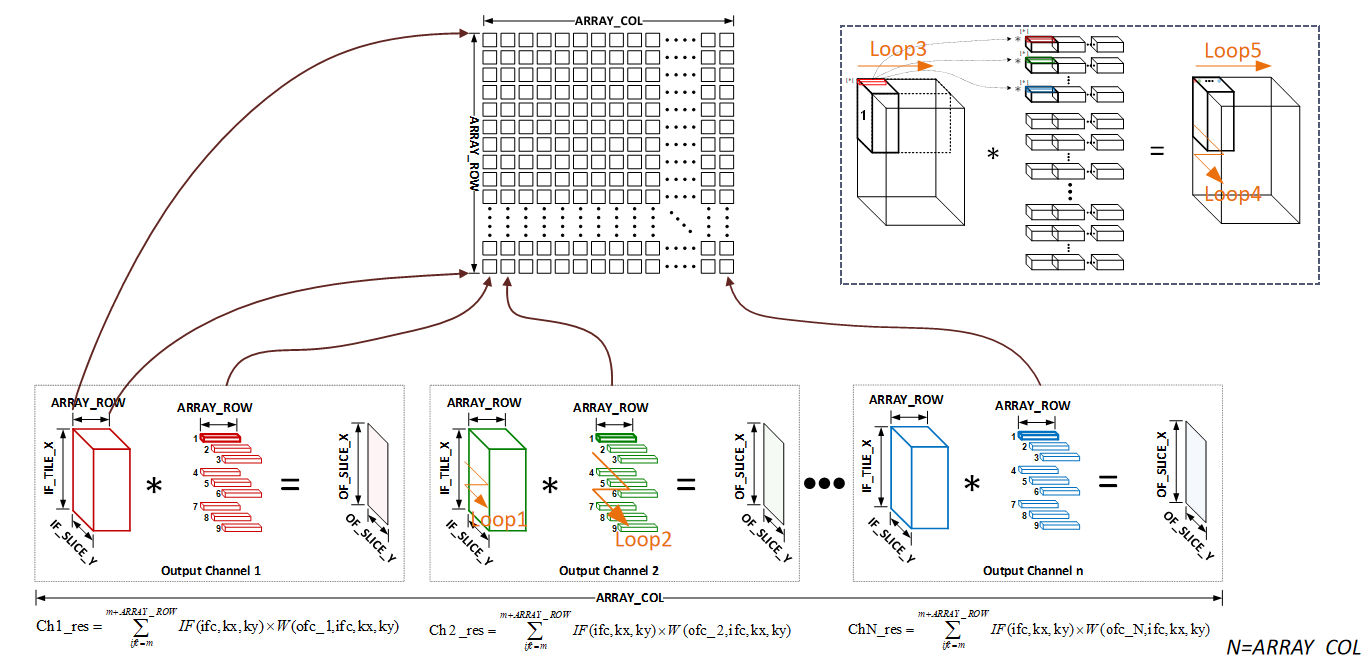
省略中间几步，得到最后结果如下：



而卷积计算可以等效转换为矩阵乘运算，谷歌的TPU中就应用脉动阵列来加速神经网络，对上图介绍的数据流做了变化，原本单列从上到下传递的是矩阵B的数据，TPU中列之间传递的是上面乘加后的结果，从每列最下面一个PE出数，这样改动后使脉动阵列适应卷积的运算特点，输入数据可以源源不断的送入，每一列每一拍都会出一个固定长度向量乘累加的结果。这个结果对应的是3\*3卷积中的一个1\*1卷积，之后更新8次权重，重复输入8次一样的输入，并将所有结果累加起来，就得到了3\*3卷积对应的结果feature map。

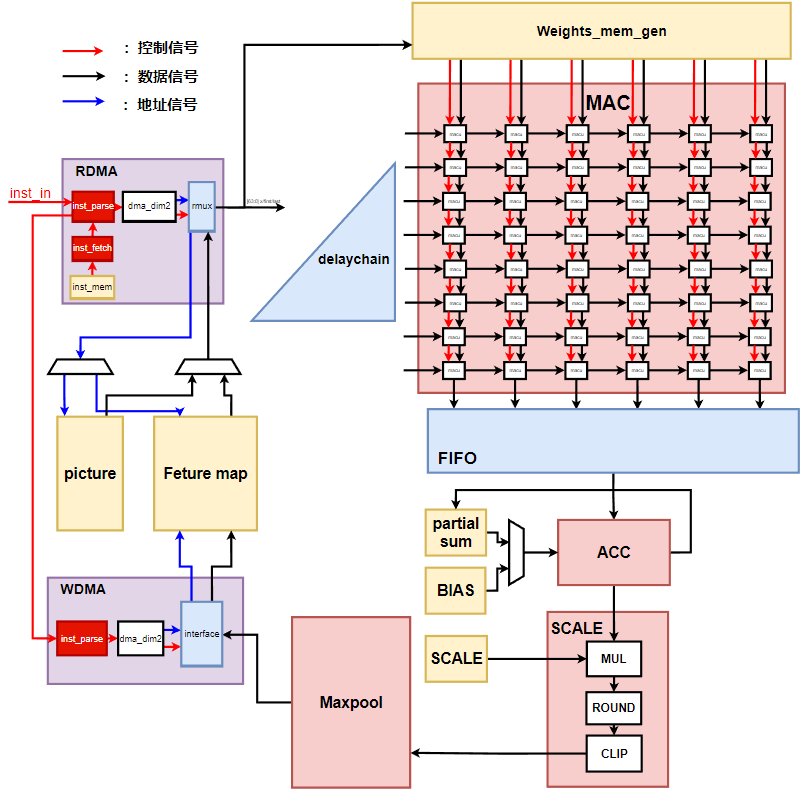


2.3.4 TPU的卷积运算



#### 2.3 架构介绍

顶层原理框图如下所示：



其典型工作流程如下：

从inst\_In口或inst\_mem里读取指令到RDMA中，RDAM解析指令包得到dma控制信息和后续模块控制信号，控制2维dma取数据（每拍取8个8bit数）；

由于脉动阵列每列从上到下传递，而输入数据是同时并行读取的，所以要将输入数据通过倒三角延时链delay，其中第一个数直通，第2个数延时1拍，第3个数延时2拍，以此类推，第8个数延时7拍；

为了让输出同步，脉动阵列的输出接了FIFO（也可以用一个5拍的倒三角延时链）；

在一次9轮（3\*3conv）调度中，脉动阵列输出会在ACC模块中累加，在最后一轮时使能输出；

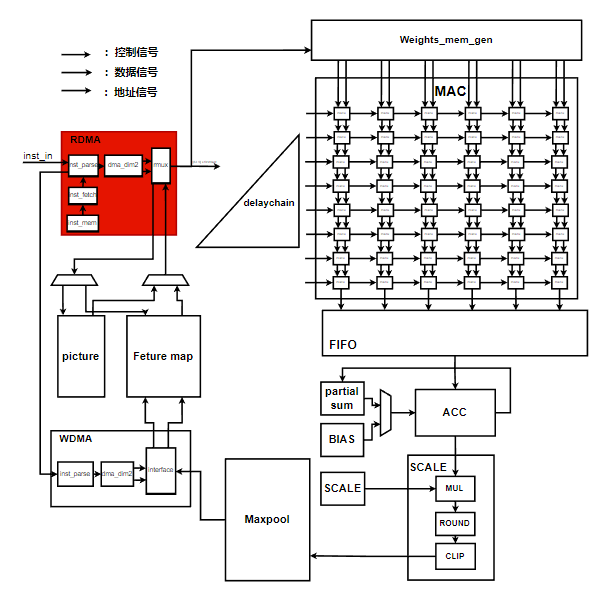
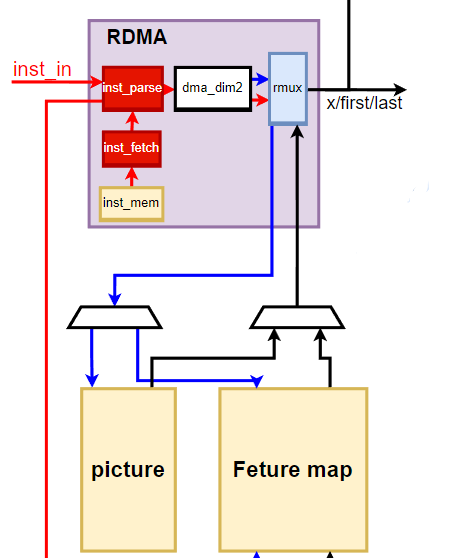
Scale模块将ACC累加结果压缩回（-128,127）的范围；

之后数据通过maxpool模块得到池化后的最终结果，并将数据送给WDMA的接口；

WDMA会接收来自RDMA的指令流，解包指令后将maxpool传入的结果数据写入指令指定的地址。至此，完成了一次一层卷积网络的前向推理。之后仍通过指令循环调度，完成整个网络的计算。

#### 2.4 子模块介绍

##### 2.4.1 RDMA&WDMA

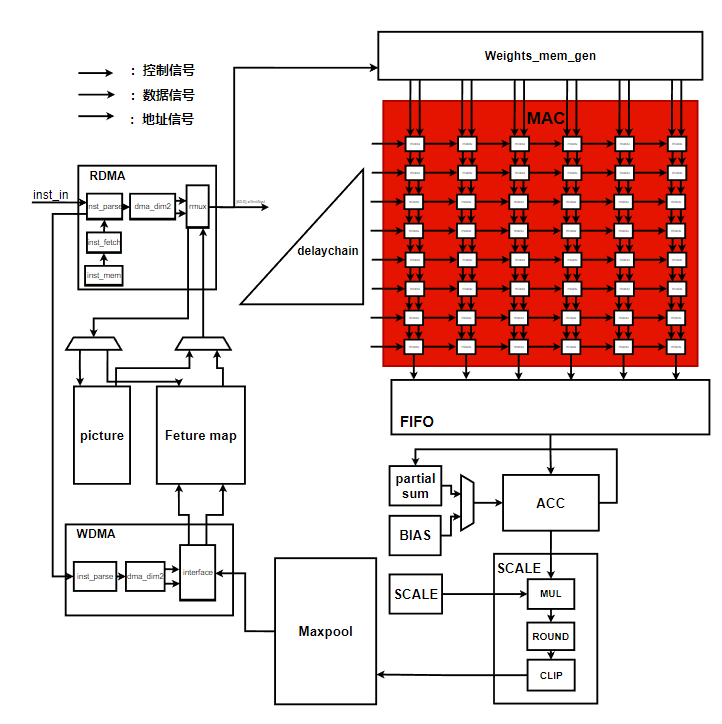
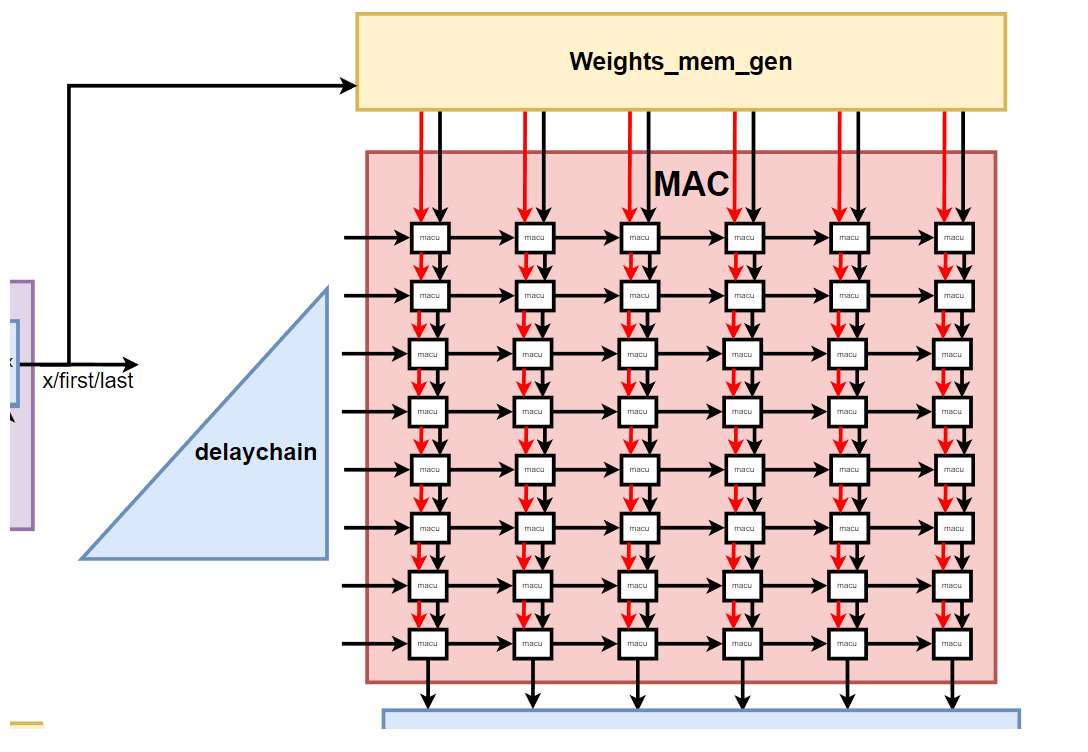
从inst\_In口或inst\_mem里读取指令到RDMA中，指令有ID信息，如果不是当前设备ID，则指令会传给WDMA，如果指令ID是RDMA的ID，则RDAM解析指令包，得到的dma控制信号会控制一个2维dma从图片mem或特征图mem中取数据（每拍取8个8bit数），同时得到其他模块的控制信号；

一次调度需要多条指令组成的指令包完成。指令位宽为36位，每条指令都包含器件ID信息、指令ID信息共6bit，除了ID信息，其余30bit包含控制信息，控制信息规则与指令ID有关。如果指令ID不等于本地ID，指令流会由inst\_parse模块传递给外部其他指令分析模块。

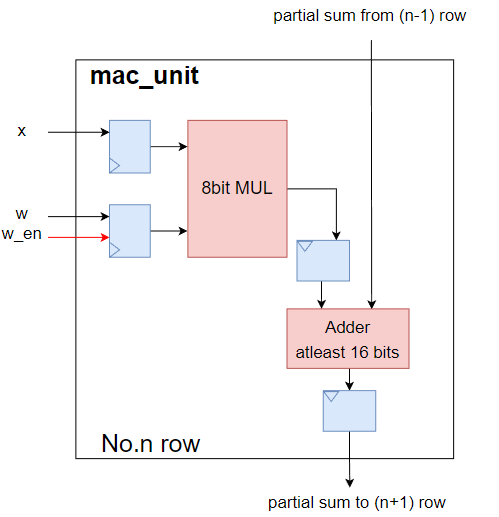
指令包的pack由inst\_parse完成，pack完成会输出指令数组到RDMA顶层unpack，同时给出start valid信号，dma\_dim2模块会在取完指令指定的数时返回一个start ready信号。为了提高效率对指令组做了一级缓存，在上一组数据read完成后，dma\_dim2返回start\_ready，立刻可以做下一次指令unpack，通过valid和ready信号进行axi握手控制指令流。

rmux是地址和数据的接口，传递地址给2部分memory，并接收2路读数据，根据控制信号决定选择哪一路数据，再根据控制信号对数据进行相应的变换。

##### 2.4.2 MAC

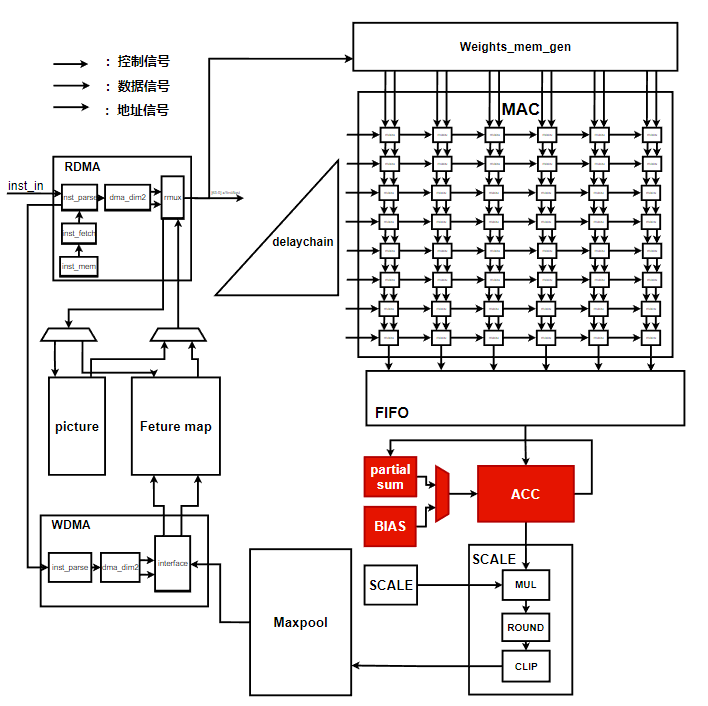
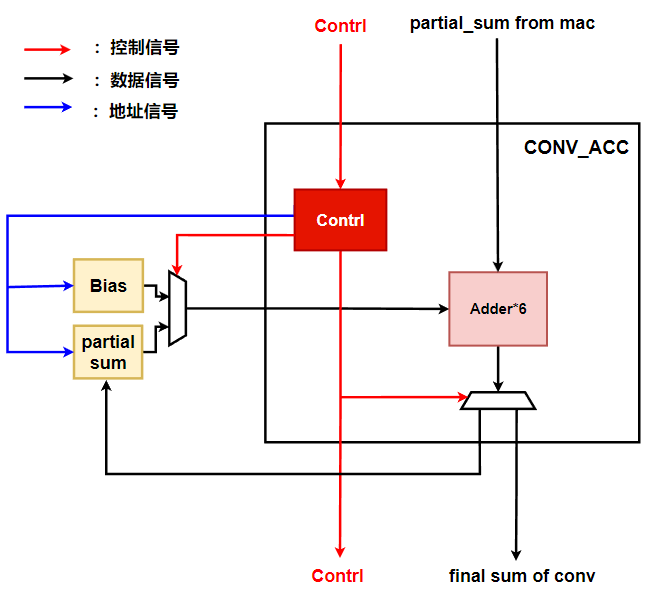
MAC阵列为8行\*6列，总共48个mac单元，其基本单元结构如下：



不同列有各自的w和w\_en信号，但会共用同一组8个8位输入x ，防止路径和多负载可能导致的驱动和布线问题，没有采用直接广播的方法，而是采用打拍的方法，每列之间都会有一组x的寄存器打拍。

其中8bit的乘法器和其输入输出寄存器由EG4S20板子上的DPS实现，DSP只支持9bit乘法，所以在输入之前会对x和w符号位扩展一位。加法器由lut实现。

##### 2.4.3 ACC

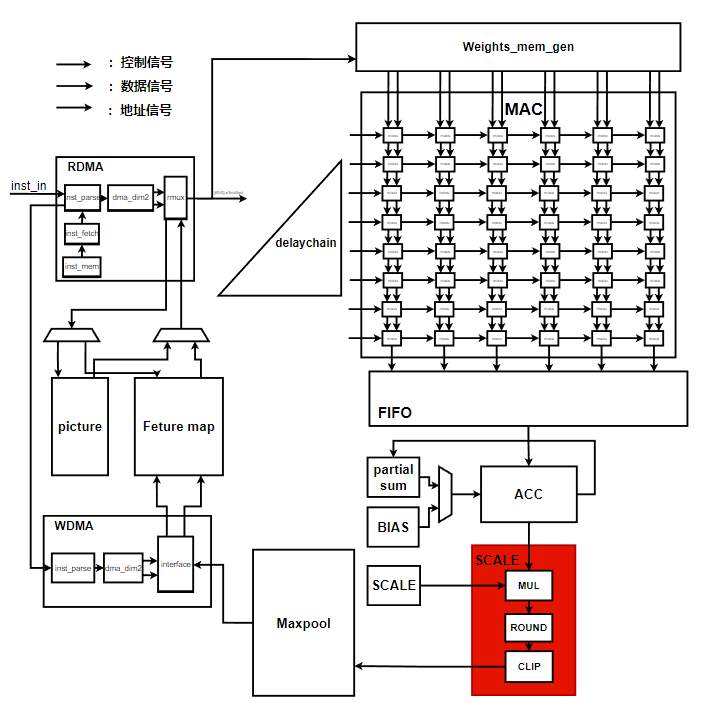
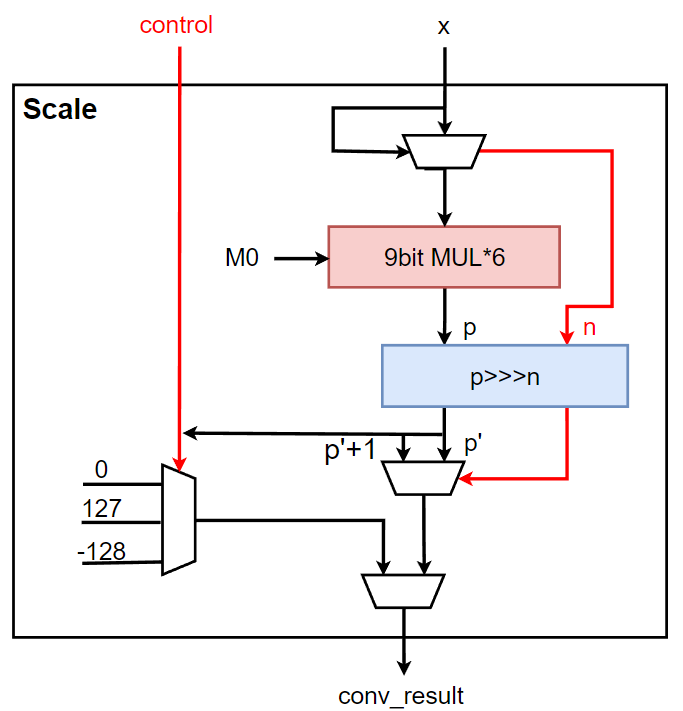
 

该模块简单来看就是6个加法器，分别对应脉动阵列的6列输出，另一个加数由控制信号得到相应的读地址。控制信号带有first\_k、last\_k信号。

在第1轮调度中，脉动阵列输出partial\_sum会和bias memory读取到的偏置数据相加，在第2-9轮调度中，脉动阵列输出partial\_sum会和前轮累加结果partial\_sum相加，9轮后得到真正的卷积结果并使能输出；

由于脉动阵列输出为19bit数，所以3\*3卷积总共会进行8次19bit加法，于是加法器的位宽选用22bit，所以partial\_sum\_from\_mac会先扩展3位符号位。

##### 2.4.4 SCALE

Scale模块的作用是将上述累加后的结果乘上一个缩放因子，使其数值重新压缩回-128~127，对结果做一次round（四舍五入）和clip（截断至-128~127）即完成一次结果与fp32等效的int8卷积计算。由于relu也是一种特殊的clip操作，所以cilp模块留出了一个relu的接口，由指令控制决定当前组数据是否进行relu；

如之前所说，上一级电路输出结果为22bit数，理论上来说需要22bit乘法器，但由于scale数据通常为0.001-0.01这个量级，通常会转换为一个定点数(或整形)+移位信息n0，所以会这个乘法会视为定点乘法，小数点后2位之后的低位数不会对结果产生影响，经过简单计算，如果只用高15位做乘法就可以得到完全一样的数。

又考虑到剩余可用dsp资源，进一步做近似，根据输入数据的大小决定选取其中某8位+符号位（共9位），同时得到一个倍数因子n1，令n=n0-n1，在9位乘法结束后右移n位即可得到近似结果，举个例子

如：scale = 0.0017 = 0.0000,0000,0110,1111,0， 取0,1101,1110 = 222 ，n0=17

x = 65536 = 00,0001,0000,0000,0000,0000

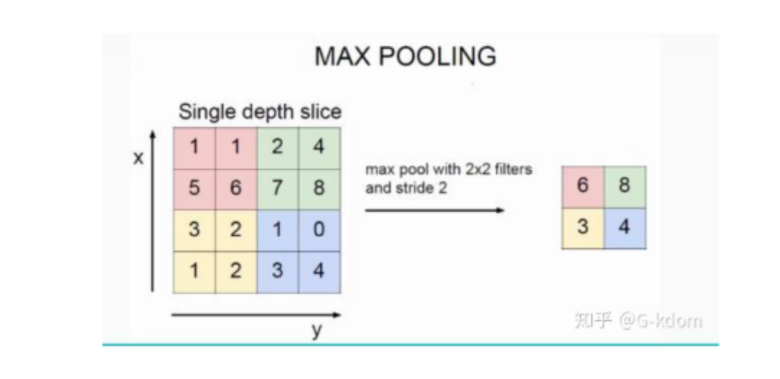
理论结果：x\*scale=65536 \*0.0017=111.4，四舍五入后为111=0110,1111

选取9位：x= 0,0000,1000 \* 2^13 = 8 \* 2^13 =65536，即n1=13，n=17-13=4

近似乘法：0,1101,1110 \* 0,0000,1000 = 222 \* 8 =1776=0110,1111,0000

右移n位结果：0110,1111，与理论结果一致。

##### 2.4.5 maxpool



该模块完成maxpooling操作，maxpooling就是在2\*2的矩阵里选出最大值，由于按行取数据，选择在运算时缓存一行的结果进行比较。