

第四届

全国大学生集成电路创新创业大赛

CICIEC

项目设计报告

参赛题目： 智能口罩佩戴识别芯片及系统设计

队伍编号： ASC125149

团队名称： 旧的不去，芯的不来

摘要

本设计是一款用于口罩佩戴识别的前端SoC芯片，它将识别算法部署在FPGA上，在一定程度上克服了传统云端计算方案实时性差、对基础设施依赖性强等不足，能够实现更加灵活、方便的部署。

该系统旨在实现多目标人脸定位以及口罩佩戴判断结果显示的功能。其采用SSD卷积神经网络算法，综合考虑资源、访存量以及网络适应性提出了一种卷积计算阵列，并针对池化、padding以及网络结构的不规则性给出了合理的解决手段，配合灵活的状态机和丰富的控制信息，得到了一种能够适应多种网络模型的卷积池化加速器。为了提高网络结果后处理的执行时效，本设计又提出了一种将冒泡排序、预测解码和非极大值抑制等功能集于一体的后处理加速器，实现了后处理过程的全硬件化。在此基础上，引入摄像头和显示器件，配合软件实现的线性插值缩放法，达到了图像实时捕获和结果显示的效果。

目录

[摘要 1](#_Toc51494656)

[0 引言 4](#_Toc51494657)

[1 功能介绍与智能算法选择 5](#_Toc51494658)

[1.1 功能介绍 5](#_Toc51494659)

[1.2 智能算法选择 5](#_Toc51494660)

[1.2.1 方案分析与选择 5](#_Toc51494661)

[1.2.2 智能算法介绍 7](#_Toc51494662)

[1.2.3 神经网络模型介绍 8](#_Toc51494663)

[1.3 模型简化 10](#_Toc51494664)

[1.3.1 卷积层与BN层合并 10](#_Toc51494665)

[1.3.2 Sigmoid函数简化 11](#_Toc51494666)

[2 系统架构及软硬件功能划分 13](#_Toc51494667)

[2.1 系统结构 13](#_Toc51494668)

[2.2 SoC架构 13](#_Toc51494669)

[2.2.1 SoC总体架构 13](#_Toc51494670)

[2.2.2 Bus Matrix互联关系 14](#_Toc51494671)

[2.2.3 Memory Map 15](#_Toc51494672)

[2.2.4 DMAC 15](#_Toc51494673)

[2.2.5 DDR访问接口 16](#_Toc51494674)

[2.3 软硬件功能划分 16](#_Toc51494675)

[3 加速器（ACC）设计 17](#_Toc51494676)

[3.1 ACC结构组成 17](#_Toc51494677)

[3.1.1 ACC结构总览 17](#_Toc51494678)

[3.1.2 控制标志寄存器堆 17](#_Toc51494679)

[3.1.3 结果寄存器堆 19](#_Toc51494680)

[3.2 卷积池化算子设计 19](#_Toc51494681)

[3.2.1 数据量化、传输与计算方案 19](#_Toc51494682)

[3.2.2 卷积池化算子结构与功能 21](#_Toc51494683)

[3.2.3 卷积计算阵列及其计算过程 22](#_Toc51494684)

[3.2.4 卷积池化单元 24](#_Toc51494685)

[3.2.5 灵活池化 25](#_Toc51494686)

[3.2.6 FSM设计 27](#_Toc51494687)

[3.3 后处理（NMS）算子设计 28](#_Toc51494688)

[3.3.1 NMS算法介绍 28](#_Toc51494689)

[3.3.2 后处理算子结构与功能 28](#_Toc51494690)

[3.3.3 硬件冒泡排序 30](#_Toc51494691)

[3.3.4 硬件NMS 31](#_Toc51494692)

[3.4 适应模型的DDR分区与地址生成 32](#_Toc51494693)

[3.4.1 DDR分区 32](#_Toc51494694)

[3.4.2 地址生成策略 32](#_Toc51494695)

[4 软件设计 34](#_Toc51494696)

[4.1 中断请求信号 34](#_Toc51494697)

[4.2 ACC控制流程 34](#_Toc51494698)

[4.3 图像缩放 36](#_Toc51494699)

[4.4系统工作流程 37](#_Toc51494700)

[5 仿真与测试 39](#_Toc51494701)

[5.1 仿真 39](#_Toc51494702)

[5.1.1 推理时间计算与仿真验证 39](#_Toc51494703)

[5.1.2 卷积池化算子仿真 39](#_Toc51494704)

[5.1.3 后处理算子仿真 41](#_Toc51494705)

[5.1.4 DMAC仿真 42](#_Toc51494706)

[5.2 测试 42](#_Toc51494707)

[5.2.1 UART测试 42](#_Toc51494708)

[5.2.2 DDR读写测试 43](#_Toc51494709)

[5.2.3 ACC测试 45](#_Toc51494710)

[6 总结与展望 47](#_Toc51494711)

[致谢 48](#_Toc51494712)

[参考文献 49](#_Toc51494713)

0 引言

新冠肺炎疫情局势当下，病毒的传染速度和规模正在迅速增大，全民佩戴口罩行动刻不容缓。然而，在某些人力监管力度较低或人流量较大场所，很难保证口罩佩戴检查能够落实到每一个人，这成为了导致病毒扩散的安全隐患。这时，一个个以人工智能技术为核心的口罩佩戴识别装置便应运而生，他们被部署在社会的每个角落，对居民口罩佩戴起到了良好的监管作用。

纵观这些常见的口罩佩戴识别装置，它们大多都采用云计算的方案，即前端采集的影像数据被传送至云中心，由中心服务器计算得到结果后返回至前端显示。其运行虽然具有强大的资源优势，但随着监控需求的增大，识别装置部署的范围和数量也在增加，前端采集数据量增加，对服务器的性能如传输带宽、计算能力要求便越高，其成本会随之增加；且作为即时交互系统，随着数据规模增大带来的传输时延增加，其实时性愈难以保证；而且云计算方案对基础设施如网络质量的依赖性极强，当这些条件受到破坏时，系统功能将会受到严重的影响。因此，为了使系统更加可靠、实时性更强，部署更加灵活，口罩佩戴识别系统有待向边缘计算方向发展。

本文阐述了一种应用于口罩佩戴识别的前端SoC芯片设计方案，它以FPGA为载体，将摄像头采集的影像数据通过神经网络硬件加速引擎计算后，将结果实时显示在前端屏幕上，在运行过程中，只需与服务器进行很少的数据交互（如上电重启后的参数初始化、非期望目标的云端上传等）。该设计在一定程度上克服了传统云计算识别的诸多不足，且经仿真测试获得了良好的效果。

本设计可应用于任何由监管人力资源不足或人流量较大导致口罩佩戴监督力度与精度不足的场所，如商场、地铁站、走廊、天桥、电梯等，该装置轻便、体积小，因而能够被安装在隐蔽的角落，当检测到人员未佩戴口罩时，它能够在第一时间将该人员面部图像传输至云服务器，以便进行人脸的识别与记录。由于采用了边缘计算，该装置可以在很大程度上节省数据的传输带宽、减轻云服务器负担，从而能够得到更大规模、更加灵活的部署，为本次防疫阻击战作出重要贡献。

1 功能介绍与智能算法选择

1.1 功能介绍

本次设计拟实现的功能为：

1.通过摄像头采集图像，在FPGA上经过智能算法处理，实时定位人脸，并标记出每个人脸是否佩戴口罩。

2.将标记后的图像实时显示在屏幕上，并将未佩戴口罩的人脸传送至云端服务器，以便进行人脸的识别。

1.2 智能算法选择

1.2.1 方案分析与选择

目前主流的人脸识别算法主要有以下三种：Adaboost学习算法、基于人脸特征脸的识别算法和神经网络识别算法。  
 Adaboost算法的算法原理是通过调整样本权重和弱分类器权值，从训练出的弱分类器中筛选出权值系数最小的弱分类器组合成一个最终强分类器。该算法在样本训练集使用过程中，对其中的关键分类特征集进行多次挑选，逐步训练分量弱分类器，用适当的阈值选择最佳弱分类器，最后将每次迭代训练选出的最佳弱分类器构建为强分类器。因此，Aadboost 算法系统具有较高的检测速率，且不易出现过适应现象，但有两点主要的缺点：①需要较多的训练样本来达到较高的精度，在条件限制内，增大了训练的难度；②训练各个弱分类器的实质是使用贪心算法，因此取得的是局部的最优分类器，而将它们连接起来组成的强分类器不一定是全局最优解。  
 基于人脸特征脸的识别算法的基本思想是：从统计的观点，寻找人脸图像分布的基本元素，即人脸图像样本集协方差矩阵的特征向量，以此来近似地表征人脸图像。总的来说，特征脸的方法简单，快速，实用，但本质上依赖于训练集和测试集图像的灰度相关性即相似性，即要求测试图像与训练集图像比较像，这显然不符合本设计需要应用于人群检测的要求，因此该算法具有较大的局限性，不适于本设计。

最重要的是，上述两种方法一般应用于单纯的人脸识别，对于本设计来说，其目的更倾向于人脸定位和二分类，若采用上述两种方法，一方面，其较强的识别功能被应用于简单的二分类，会造成功能浪费；另一方面，它们无法首先完成人脸定位，识别便无从谈起。  
 而神经网络算法在人脸检测与识别上的应用比前两者有明显的优势，因为对人脸识别的许多规律进行显性描述是相当困难的，而神经网络算法则可以通过学习的过程对这些规律进行隐性描述，因此该方法的适应性更强，更容易实现，不仅可以轻松地实现定位功能，而且识别速度很快，唯一的缺点是识别率相对较低，但此次我们只需要区分人脸是否佩戴口罩而非精细化识别人脸。另外，随着深度学习的迅猛发展，各种高效算法层出不穷，互联网上的开源项目亦随处可见，如果能够对其加以参考，将会大大缩短本次项目的开发周期。综合考虑之下，我们选择了卷积神经网络算法。

由此我们拟定了以下三种神经网络算法方案：

方案一：采用当前流行的YOLO（You Only Look Once）算法，将佩戴口罩和未佩戴口罩划分为两个类别，配合NMS（非极大值抑制）后处理，达到同时定位和分类的目的。

方案二：采用SSD（Single Shot Detection）算法，同样配合NMS进行目标定位和识别。

方案三：采用MTCNN（Multi-task Convolutional Neural Network），即多任务卷积神经网络。该网络只适用于单类目标检测，因此还需采用相应的用于目标分类的后处理算法，如支持向量机（SVM）等进行目标分类，以达到减少虚警和漏报的目的。

方案分析：

YOLO虽然在目标检测领域有着突出的性能，但是其更偏向于多类目标检测的应用领域，对于本设计需求的两类目标检测，无疑是一种浪费，再加上YOLO的网络模型庞大，一旦将计算部署在FPGA上，在没有成熟的轻量化方案可利用时，时效会大大降低。

MTCNN在单类目标检测与识别领域中具有网络结构简单、图片大小自适应的优点，MTCNN采用了图像金字塔的思想，能够在不同尺度上对图片进行检测。其模型由P-Net，R-Net和O-Net三个网络级联构成，由于网络逐级加深，因此能够更大程度地提取目标的特征，从而达到目标识别的目的。但是该算法执行流程复杂，不适合硬件实现；且需要单独的后处理算法辅助分类，增大了设计的难度。

SSD采用多尺度特征图的方法，并且设有灵活比例和大小的预选框（anc hor），因而能够适应大范围、高密集度的目标检测，非常适合本设计所应用的人脸检测。SSD的主干网络采用VGG-16的网络模型，规则且卷积核大小固定，因此易于硬件实现。并且在多方面搜集资料后，我们掌握了一种SSD模型轻量化的成果，这更为我们选择SSD算法提供了坚实依据。

综上考虑，本设计采用方案二。

1.2.2 智能算法介绍

SSD相比于传统应用于目标检测的深度学习算法主要具有两个突出特点：多尺度特征映射和anchor多样化。前者指SSD网络将不同层的输出作为不同尺度的特征图，一并做出预测。这种做法克服了传统图像金字塔方法多循环、时效低的缺点，同时相比于单特征图的CNN算法能够适应更大范围的目标检测。后者指SSD的特征图每个单元都提供4个大小、比例不同的anchor，这在一定程度上增强了该算法检测的精准性。

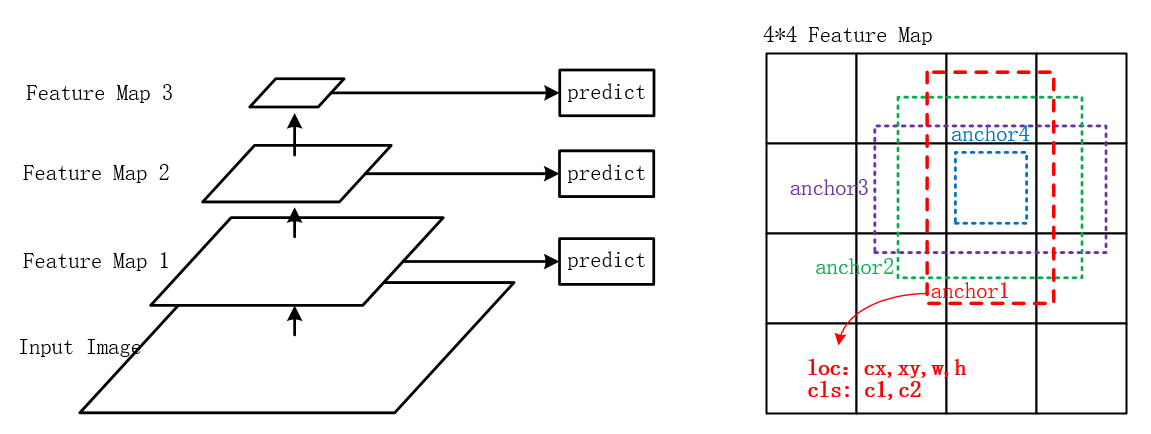


图 1-1 多尺度特征图映射(左) anchor多样化(右)

本算法采用33、17、9、5、3五个特征图尺度，标记Mask和No Mask两个类别，由于每个特征图单元包含4个anchor，每个anchor对应一个预测框，于是网络输出个预测框，每个预测框含有4个位置信息和2个置信度信息。

得到这些输出后，首先对每一个预测框的两个置信度取最大值（因为只有较大的值才能正确反映其类别），记录其类别后依据置信度阈值对置信度进行筛选。然后依照anchor的尺寸对位置信息解码，解码公式如下：

其中cx、cy、w和h分别代表中心x坐标、中心y坐标、宽度和长度。d、b和l分别代表anchor值、实际值和网络预测输出值。

解码后的位置信息经过NMS（详见2.2.3 NMS算子设计）处理后得到最终的人脸框坐标。 算法执行总流程如图1-2所示。

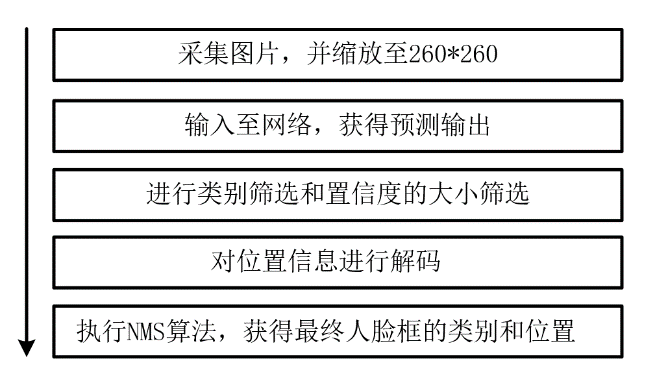


图 1-2 算法执行总流程

1.2.3 神经网络模型介绍

对VGG-16原始模型进行两方面修改：轻量化和全卷积化，使得其模型体积和参数数量大大降低，完整模型如图1-3所示（原图请见附件）。

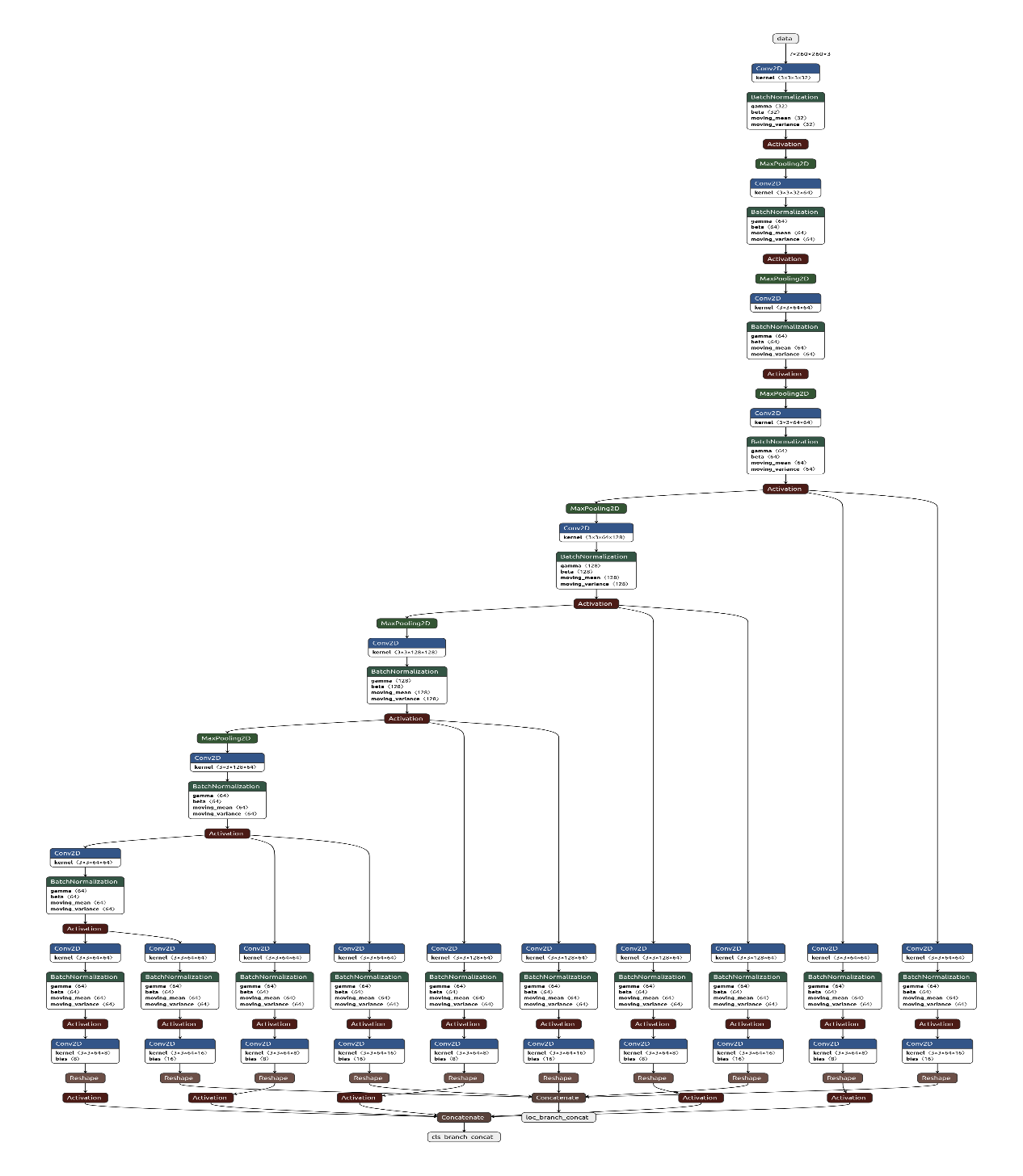


图 1-3 完整网络模型

该模型接收260\*260 RGB格式图片输入。若把padding、卷积、BN、池化等看作一大层，则该模型含有8个主干层和20个分支层，主干层用于生成不同尺度的特征图，分支层用于对不同尺度的特征图做出预测。

1.3 模型简化

1.3.1 卷积层与BN层合并

BN（Batch Normalization），是一种在神经网络中为了提高训练效率，对网络某一层输入数据的每个Batch进行正则化，从而控制其均值和方差的手段，一般作用在激活函数之前。它的处理过程由式给出：

其中，表示当前Batch的第组数据，为卷积的计算结果，为对归一化后得到的值，和为当前Batch的均值和方差；为对进行分布修正后的值，其值直接被送至激活函数，和为修正参数，由网络训练得到。

一旦网络训练结束，，，和都为确定的常数，这时卷积层和BN层可以总体地表示为（由于引入了BN层，卷积层舍去了bias参数）：

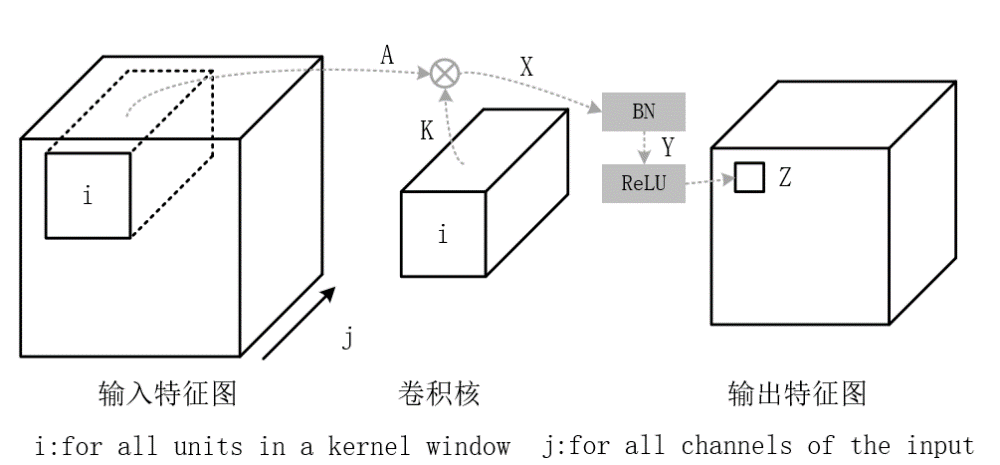


图 1-4 卷积层和BN层

在图1-4中所展示的卷积、BN计算过程中，有：

将式代入式可得：

令：

则：

式展示了使用一个参数为和的新卷积层代替原始卷积层和BN层，新卷积层的参数的计算方法由式给出。通过合并卷积层和池化层，简化了网络结构，减少了网络参数，在一定程度上降低了硬件设计难度，节省了推理时间和硬件资源。

1.3.2 Sigmoid函数简化

Sigmoid函数是神经网络中常用的激活函数，其表达式如下：

在本设计采用的算法中，需要使用Sigmoid函数激活置信度预测值。然而，Sigmoid函数的硬件实现方法非常复杂，如ROM查找表法、泰勒级数逼近法、cordic算法等，即使使用Xilinx的cordic IP核，也会造成较多资源的占用

值得注意，置信度的作用仅体现在被比较和筛选，因此，只要保证其大小关系不发生变化即可，而不需对其精确定量。Sigmoid函数是单调递增函数，因此其输入x和输出y是唯一映射的，假设置信度阈值为y\_thresh，则其对应的x\_thresh便被唯一确定；且对于多个置信度y1-ym，其对应的x1-xm大小关系不变。因此可以直接省去sigmoid函数，将原本的y\_thresh替换为x\_thresh。值得注意的是，由于网络中间结果采用定点数量化，因此需要保证其表示范围大于x\_thresh，当输出溢出时，可直接判定为保留该置信度。

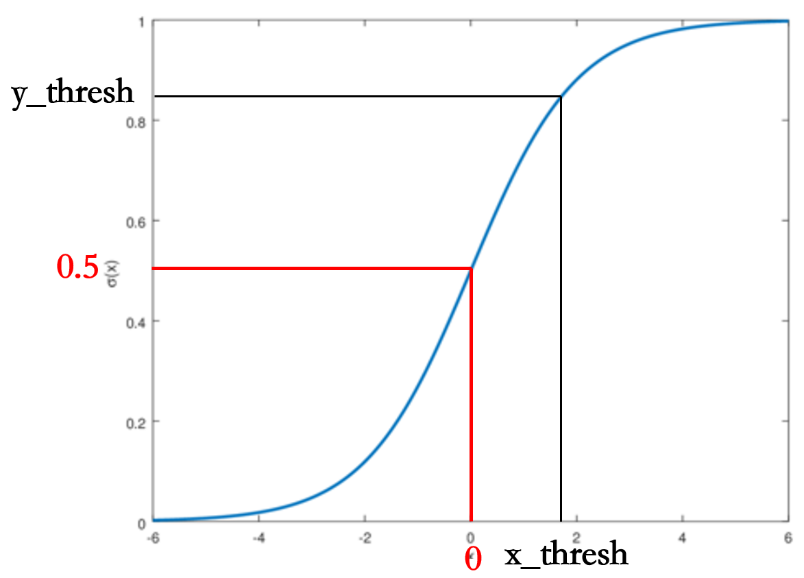


图 1-5 Sigmoid函数

通过在Pytorch框架上进行多次验证，我们得出y\_thresh为0.5时推理效果较好，于是x\_thresh可设置为0，这样原本复杂的Sigmoid函数便可直接被替换为简单的正负判断。

2 系统架构及软硬件功能划分

2.1 系统结构

本设计系统顶层结构图如图2-1。

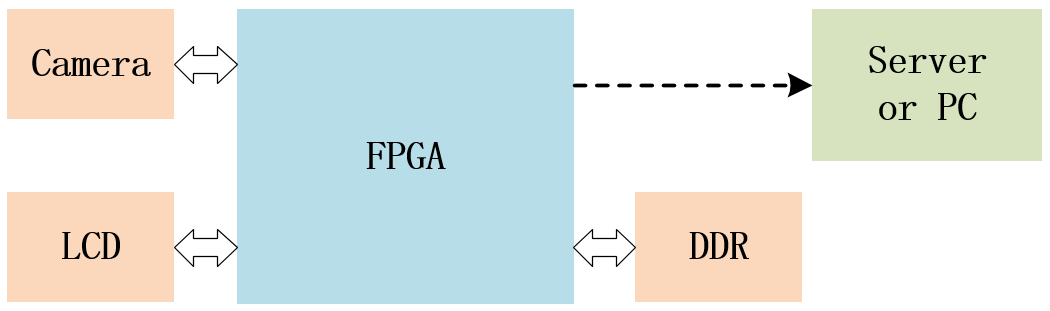


图 2-1 系统结构图

在本系统中，由于采用网络模型较大，参数和中间计算结果数据较为庞大，因此使用到了片外DDR用以存放模型参数和中间计算结果。

2.2 SoC架构

2.2.1 SoC总体架构

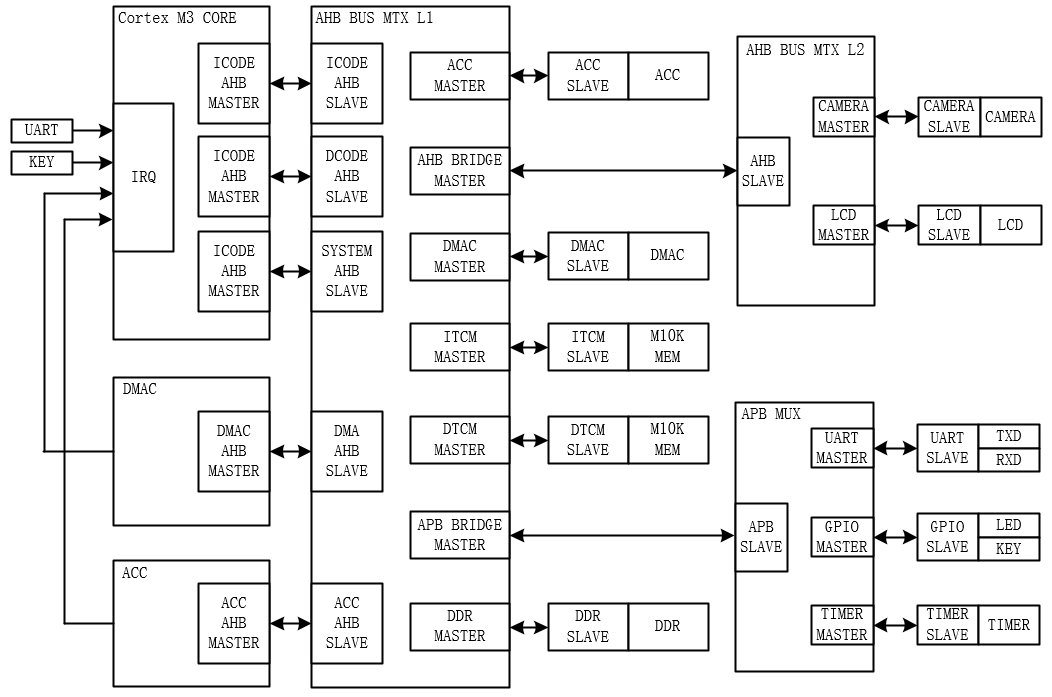


图 2-2 SoC总体架构

我们设计的SoC由如下几个主要模块构成，ACC、DMAC、DDR接口、CAMERA和LCD等模块为自主编写所得，其余模块由CMSDK生成或提供。

* Cortex-M3 Core
* Bus Matrix
* Single Channel DMAC
* ITCM(Store new program code from KEIL for debugging)
* DTCM(Stack and Heap)
* UART
* GPIO(LED and Switch)
* LCD
* CAMERA
* ACC
* Timer

2.2.2 Bus Matrix互联关系

图2-3展示了总线矩阵的互联关系。由于网络参数和中间结果均存放在DDR，ACC工作时仅需与DDR进行数据交互。

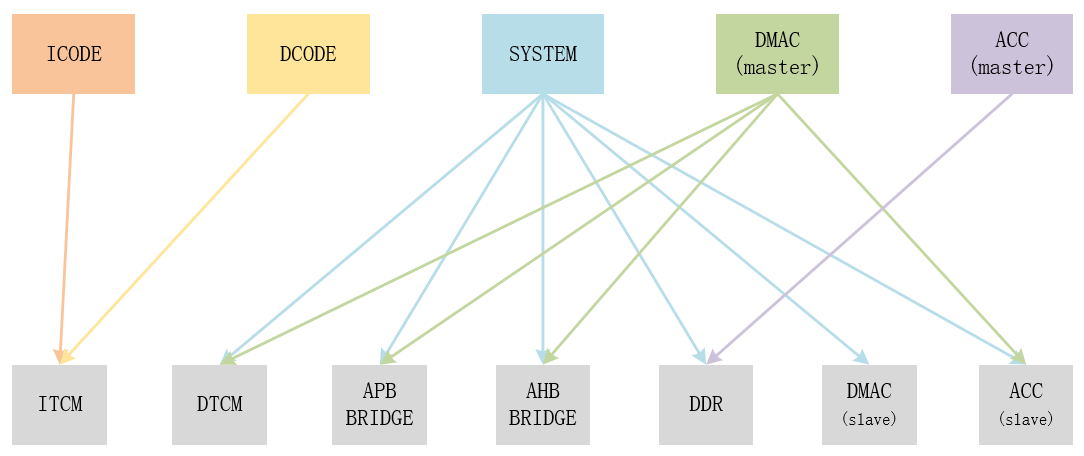


图 2-3 Bus Matrix 互联关系

2.2.3 Memory Map

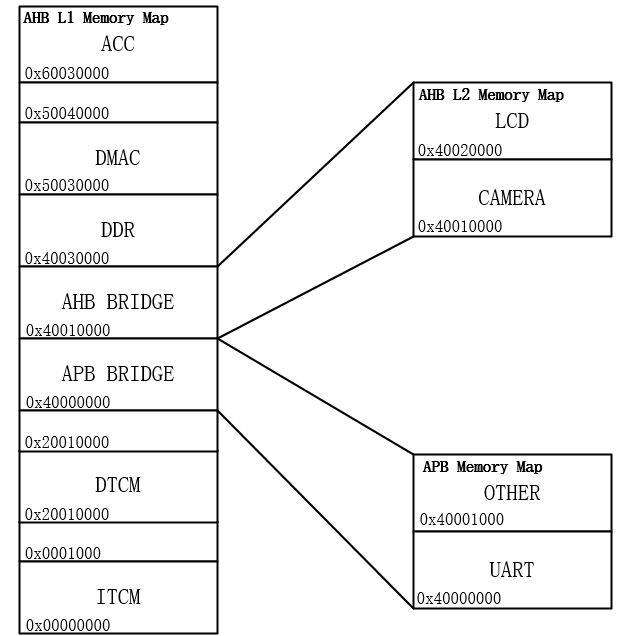


图 2-4 Memory Map

2.2.4 DMAC

我们设计的DMAC设有Master和Slave两个接口，均与AHB总线相连。内部设有Source、Destination、Length和Size四个寄存器，分别用于存放DMA传输时的源slave初始地址、目的slave初始地址、传输长度和传输位宽。

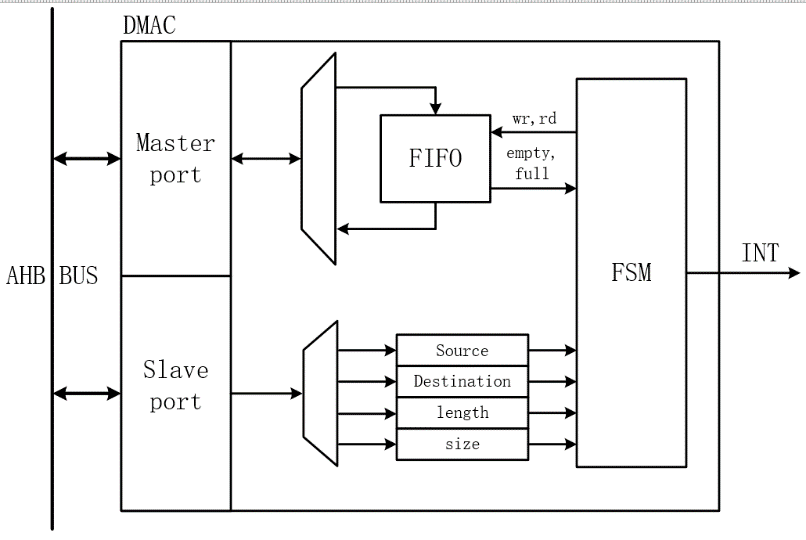


图 2-5 DMAC结构

2.2.5 DDR访问接口

我们在设计DDR访问接口时用到了Vivado提供的MIG（Memory Interface Generator）IP核，由于该模块采用的是AXI总线协议，我们另外使用了一个AHB to AXI的总线桥，并增加了一个异步FIFO作为数据缓存，其总体结构如图2-6所示。

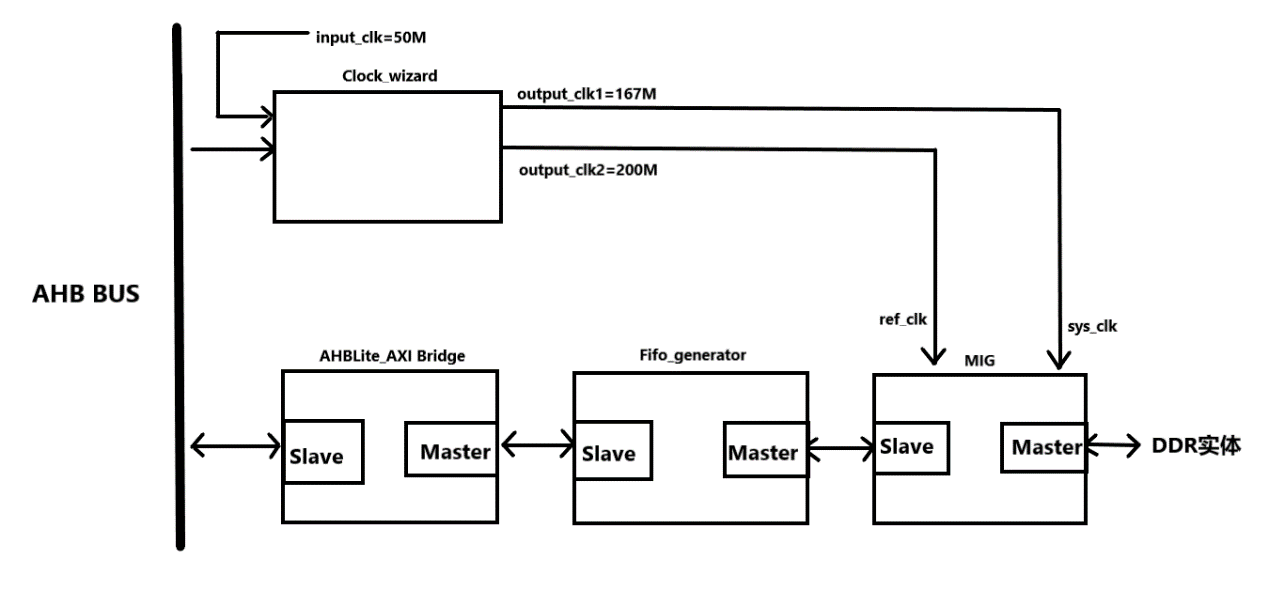


图 2-6 DDR访问接口结构

2.3 软硬件功能划分

在本设计中，SSD算法的计算部分几乎都由硬件自动完成，软件仅负责控制、协调和调度每个计算模块的运行，其具体功能划分如下：

硬件负责神经网络的所有计算、后处理运算以及几乎所有的数据搬运，如ACC与DDR、摄像头和DDR、DDR和LCD之间的数据搬运。软件负责在每次计算前向相应的硬件写入控制字，如ACC每层计算前和后处理运算前的寄存器配置、数据搬运前DMAC的寄存器配置。另外，计算结果的读取也由软件完成。总而言之，在本设计中，硬件功能承担了系统功能的绝大部分。

3 加速器（ACC）设计

3.1 ACC结构组成

3.1.1 ACC结构总览

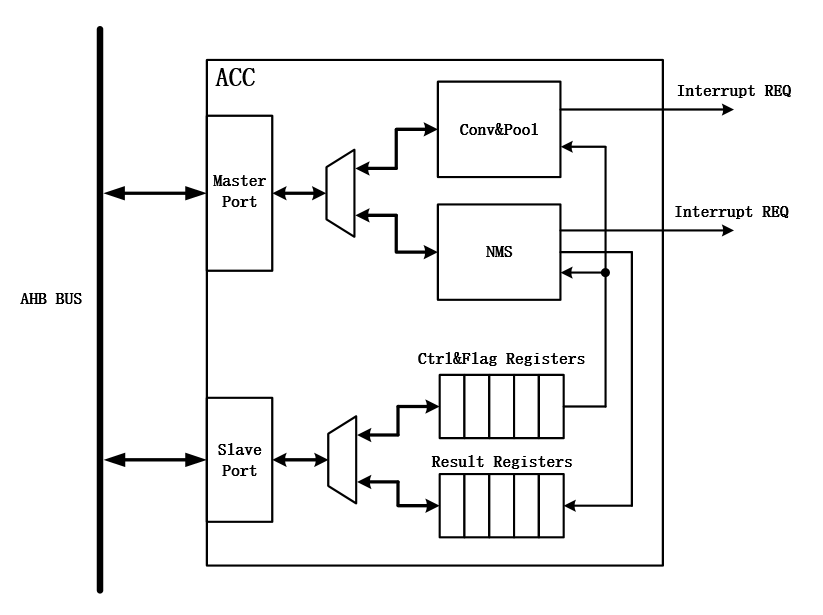


图 3-1 ACC结构组成

ACC设有master和slave两个接口，均遵循AHB总线协议。Master接口与卷积池化（Conv&Pool）算子和后处理（NMS）算子两个运算单元相连；Slave接口与ACC内部两个寄存器堆相连，它们分别是控制标志（Ctrl&Flag）寄存器堆和结果（Result）寄存器堆，前者存放ACC的配置信息与状态标志，后者存放推理得到的最终结果。另外，ACC的两个运算单元能够分别产生一个中断请求，用以指示当前卷积层或NMS运算的结束。

3.1.2 控制标志寄存器堆

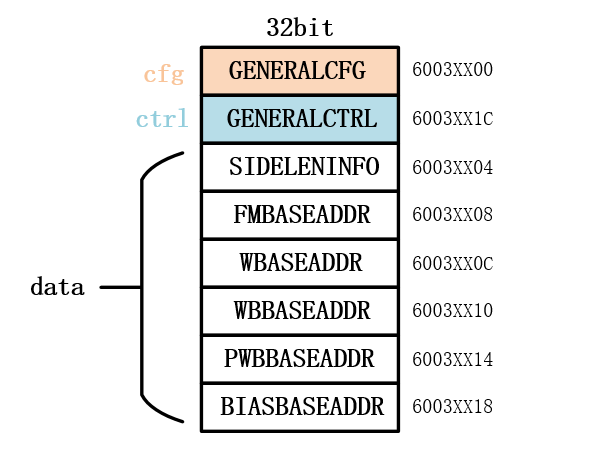


图 3-2 控制标志寄存器堆名称与地址映射

该寄存器堆包括8个32位寄存器，其中1个配置寄存器，1个控制寄存器，6个数据寄存器。各寄存器含义参照下表：

|  |  |  |
| --- | --- | --- |
| 寄存器名称 | 位 | 含义 |
| GENERALCFG | 6-0 | 输入通道数-1 |
| 12-7 | 一次pass中激活的输出通道数-1。硬件允许的最大并行输出通道数为64，但某些层的输出通道为128，则需要两个pass完成该层运算 |
| 18-13 | Δx,缓存窗边长 |
| 19 | 是否预写回。在网络分支处，卷积的结果经relu后预写回（pre-write-back）作为网络分支的输入，其再经过池化后写会(write-back)作为网络主干上下一卷积层的输入 |
| 20 | 是否为输出层。输出层的卷积运算比较特殊，没有经过relu激活 |
| 22-21 | 00：不是前三层；01：第一层；11：第二层；10：第三层。前三层输入特征图较大，因此被分割为多个缓存窗，因此其池化padding方式和输出padding方式都比较复杂，需要单独表示 |
| 23 | 输出是否padding。输出的padding即下一层卷积运算前的padding |
| 27-24 | n，表示缓存窗个数。输入特征图边长=n\*Δx |
| 28 | 运算需要的pass数-1 |
| 29 | 是否进行池化 |
| 30 | 0：进行卷积池化运算；1：进行NMS运算 |
| 31 | 保留 |
| GENERALCTRL | 0 | 卷积运算启动信号 |
| 1 | 卷积计数器(m\_FSM)使能信号 |
| 2 | 特殊标志 |
| 3 | NMS运算启动信号 |
| 4 | CPU告知ACC读结果完毕 |
| SIDELENINFO |  | 输入特征图，输出特征图的边长信息 |
| FMBASEADDR |  | 输入特征图在DDR的基地址 |
| WBASEADDR |  | 权重在DDR的基地址 |
| WBBASEADDR |  | 输出写回DDR的基地址 |
| PWBBASEADDR |  | 输出预写回DDR的基地址 |
| BIASBASEADDR |  | ACC内部Bias ROM基地址 |

表 3-1 控制标志寄存器堆明细

3.1.3 结果寄存器堆

ACC结果寄存器堆提供五个寄存单元用于寄存NMS运算结果。每个寄存单元包括6个只读寄存器：is\_valid（用于指示该单元的信息是否有效）、category（表示是否佩戴口罩）、xmin，ymin，xmax，ymax（人脸框位置信息）。

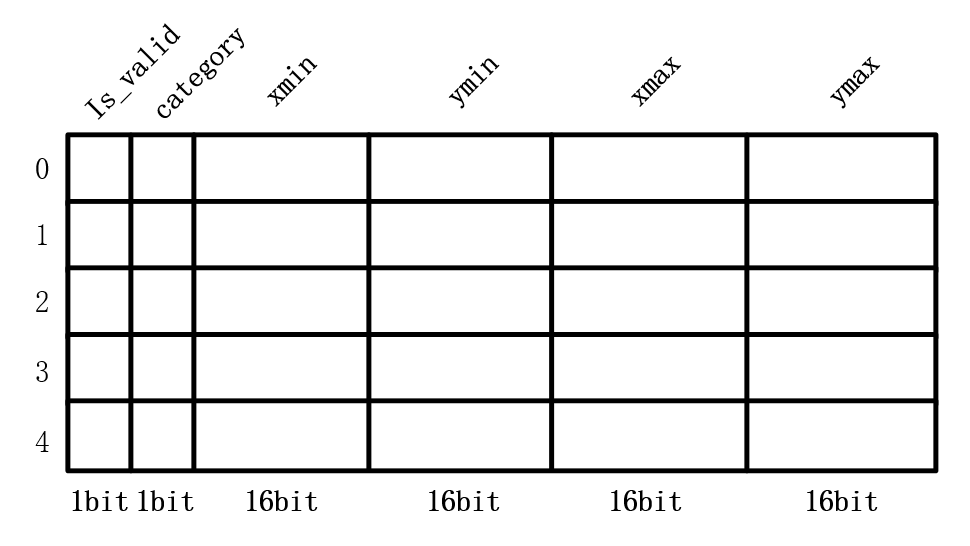


图 3-3 结果寄存器堆

3.2 卷积池化算子设计

3.2.1 数据量化、传输与计算方案

由于本设计采用的网络模型含有BN层，因此其每层中间结果几乎遵循相同的分布方式，这为我们采用定点数量化提供了极大的便利。

对于网络的中间结果和参数，经过在Pytorch框架上的反复推理验证，我们选择了一种适应其分布范围的16bit定点数量化方式：S+5+10,其中S为符号位，5为整数位数，10位小数位数。在进行计算时，乘法器得到的32bit结果理论上包含20bit小数，我们舍去其中低10bit，并截取整数部分的低5bit，便重新获得了S+5+10的卷积结果。在进行后处理运算时，由于这时的数据分布较为集中且对精度要求较高，我们灵活采用了S+1+14和S+2+13等多种量化方式，并通过补零和截断等操作来匹配计算，在此不再赘述。

[3]针对使用片外存储方式的CNN硬件加速引擎提出了其访存次数的下界条件。该文中将卷积运算表示成矩阵相乘的形式，在每次访存中，只获取输入特征图上的一个窗口（下称缓存窗）的部分通道，经计算得到一个部分和，然后通过多次访存和计算，得到完整和。其计算方式如图3-4所示。

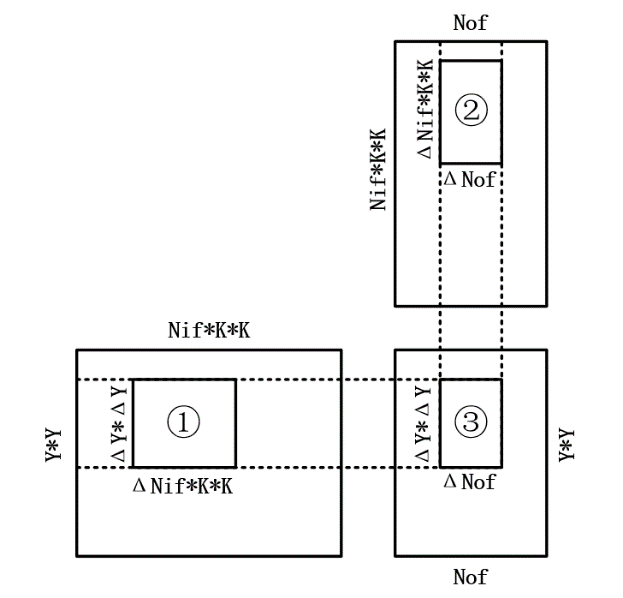


图 3-4 矩阵相乘表示的卷积运算

其中各变量含义如下：

|  |  |
| --- | --- |
|  | 输入特征图通道数 |
|  | 输出特征图通道数 |
|  | 卷积核边长 |
|  | 输出特征图边长 |
|  | 单次访存中获取的输入特征图通道数 |
|  | 单次访存中获取的输出特征图通道数 |
|  | 单次访存所得数据对应的输出特征图区域的边长 |

在一次访存中，获取输入特征图的①区域和权重矩阵的②区域，进行计算后，得到对应输出特征图③区域的一个部分和，称计算过程为一轮计算。

在本模型中，卷积核大小,因此输出特征图中边长为的区域对应输入特征图中区域的边长为，可得单次访存量为：

每得到一个完整和，需要访存的次数为：

总访存量为：

可以看出，在其他参数不变的情况下，随着单次访存中获取的输入特征图通道数和输出特征图通道数的增大，总访存量减小。然而和的增大会受到片内存储容量和硬件并行计算能力的限制。

本设计拟定了两个计算方案：

方案一：采用输出通道全并行计算且输入单通道读取的方式，即，。这种情况下，为了满足输出通道数最大（）的卷积层，应将逻辑资源的并行度设置为128，即设置128个独立的MAC单元。虽然资源数量允许，但是对于大部分的卷积层，这种方案会极大降低资源的利用率，造成资源的浪费。

方案二：将限制在64以下，但是令，并对单次访存中读取的所有输出通道和两个输入通道进行并行计算，即由方案一的128\*1转换为64\*2，这样在采用同等数量的MAC单元的条件下，提高了资源的利用率。

综上所述，本设计采用方案二。

3.2.2 卷积池化算子结构与功能

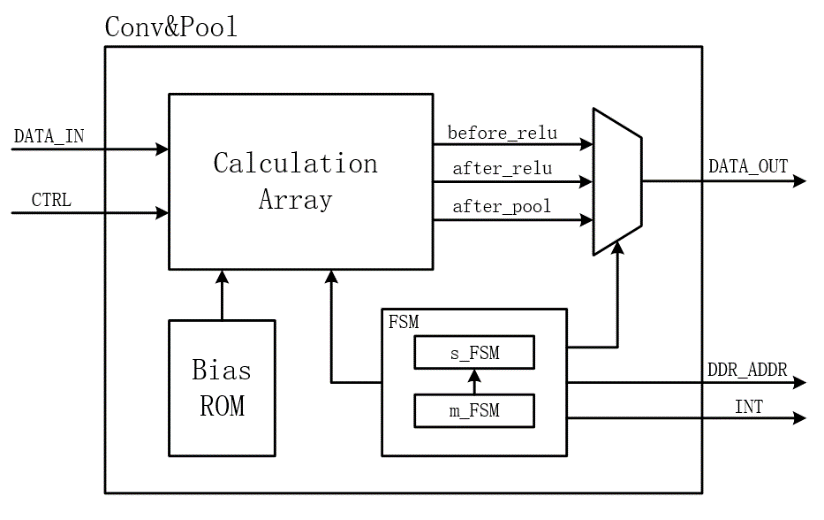


图 3-4 ACC结构组成

卷积池化算子结构如图3-4所示。其中，DATA\_IN为算子的数据输入总线，DATA\_OUT为数据输出总线，CTRL即控制标志寄存器堆表示的控制信息，Bias ROM用于存放卷积的偏移量参数。

该算子能够在一次运算中将输入特征图中相邻两个通道上的一个窗口（缓存窗）读入计算阵列，配合权重进行卷积计算、池化运算，并将运算结果（ReLU之前、ReLU之后或者池化后）经过padding后写回至DDR相应的区域。当计算遍历完输入特征图的所有缓存窗和通道，整层运算结束，产生中断请求。另外，可以通过ACC的控制寄存器来选择当前层是否需要池化、是否需要padding、是否预写回以及选择padding方式等。

3.2.3 卷积计算阵列及其计算过程

基于3.2.1中提出的数据传输和计算方案，我们设计出了相应的卷积计算阵列，如图3-5所示。

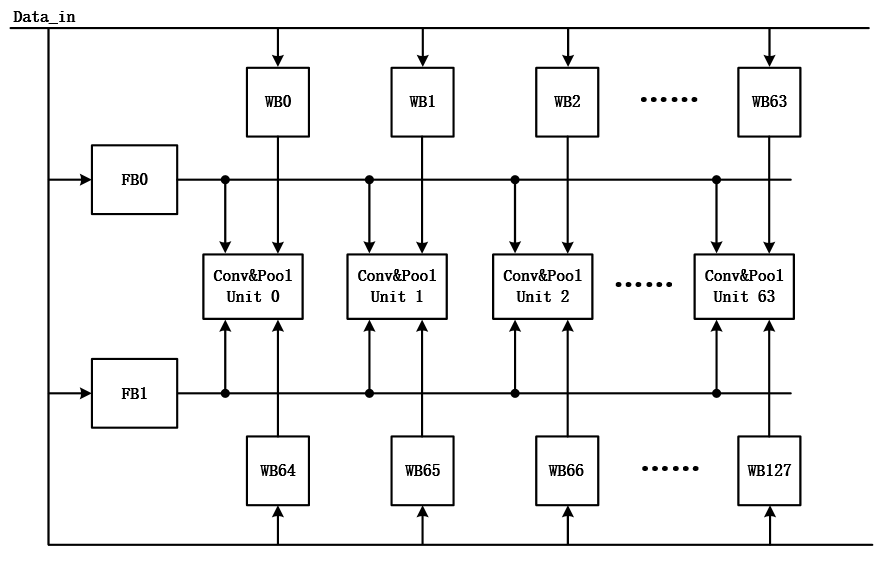


图 3-5 卷积计算阵列

在一轮计算中，ACC通过其Master接口将输入特征图的一个缓存窗读入FB0和FB1两个特征图缓存，将权重数据读入权重缓存WB0-WBx、WB64-WB(64+x)（x视该层激活的输出通道的并行度而定），其数据被送至卷积池化单元（Conv&Pool Unit）进行计算，每个卷积池化单元之间的计算并行进行。

图3-6展示了缓存窗和权重在Buffer中的存储方式。其中表示缓存窗边长。在存储过程中，输入缓存窗的宽、高和通道，卷积核的个数都由独立的计数器跟进，这些计数器通过组合逻辑计算生成DDR的读地址。

值得注意的是，为了保证多个缓存窗经卷积后的结果能够不重叠、不遗漏地组成完整的结果特征图，多个缓存窗之间应当有K-1的重叠。

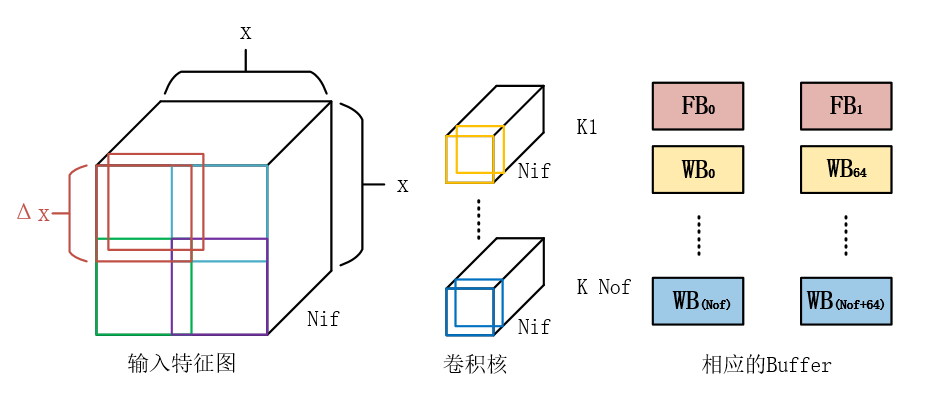


图 3-6 缓存窗和权重在Buffer中的存储方式（Nof全并行情况）

图3-7展示了一轮卷积计算的大致过程。在该过程中，当前卷积窗在缓存窗中的位置（宽、高）和当前正在计算单元在卷积窗中的位置（宽、高）都由独立的计数器跟进，这些计数器通过组合逻辑计算生成FBx和WBx的读地址。数据从FB和WB串行输出，进入乘累加单元，K\*K个时钟周期后，得到对应两个输入通道的一个部分和结果，经相加后写入RB（详见3.2.4），写入地址同样由上述计数器生成。

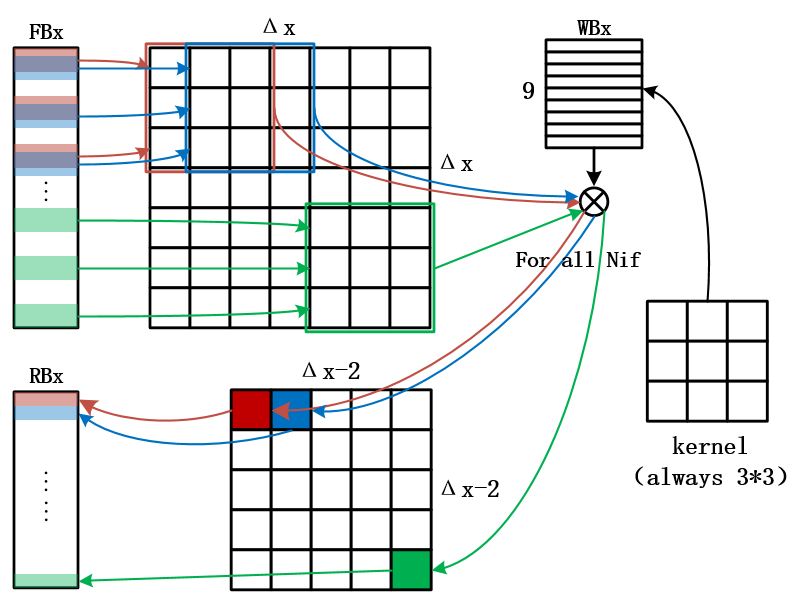


图 3-7 卷积计算过程

3.2.4 卷积池化单元

将图3-5中的每个卷积池化单元放大，得到图3-8所示结构。

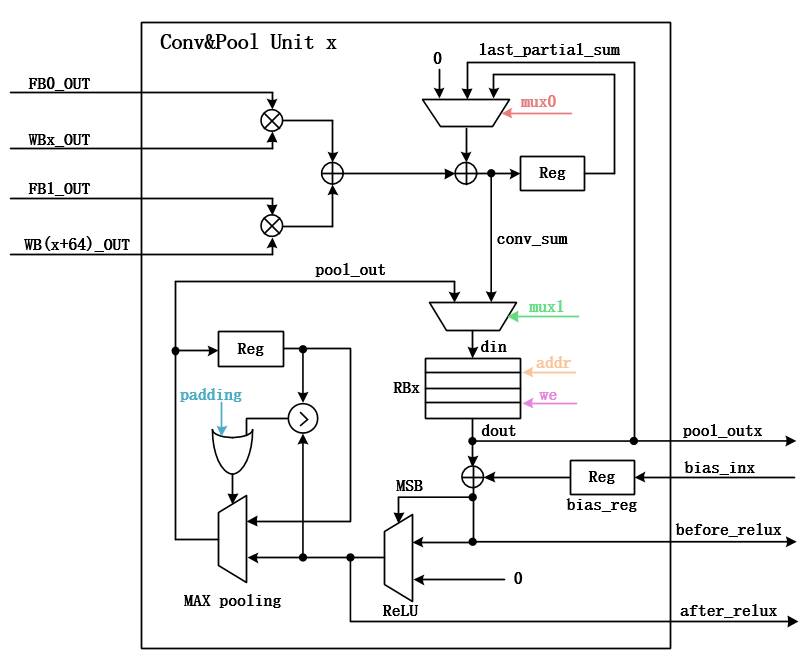


图 3-8 卷积池化单元

其中FB\_OUT和WB\_OUT分别为卷积计算阵列中FB和WB的输出端。RB为中间结果缓存，为了提高资源利用率，卷积与池化采用了RB分时复用的方式，即在卷积运算时，将卷积计算结果写入RB中；在池化运算时，池化算子从RB中读出数据，经padding和池化后重新写回RB。

在进行卷积计算时，FB0和FB1的计算结果被一直累加。在一轮计算之后，RB中存放的只是对应输入特征图两个通道上的部分和，因此，当下一轮的计算开始时，乘累加应该在上一轮计算结果的基础上进行，为此特地提供图中last\_partial\_sum数据通路。当遍历所有输入特征图通道，得到完整和时，便告知FSM进入池化状态，同时在下一次卷积计算时乘累加器清零。

在池化过程中，使用一个寄存器来寄存当前池化窗的最大值，当计数器告知硬件当前正处在padding状态时，该寄存器值保持不变，达到了padding的效果。当完成一个池化窗口的运算后，将池化结果写回至RB。池化过程可通过图3-9表示。

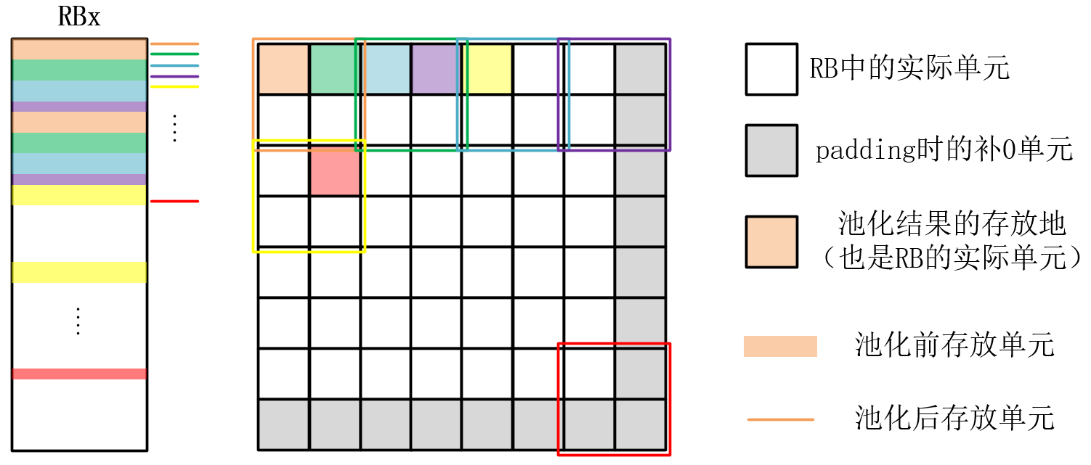


图 3-9 池化过程

为了适应不同层的需求，每个卷积池化单元都提供三种数据输出：池化后、ReLU前和ReLU后，所有卷积池化单元的这三路输出经过MUX后作为数据输出总线上的数据。

3.2.5 灵活池化

本设计采用的网络模型在池化方式上具有很大的不规则性，主要表现在：池化出现的随机性（即并非每次卷积都跟随池化）；池化padding的随机性（即某些特征图经卷积后边长为奇数，需要padding）；划分缓存窗带来的padding的多样化。针对上述难题，本设计通过增加控制信息，设计复杂的FSM以及定义多种padding方式设计出了一种能够适应多种网络结构、多种padding方式的灵活的池化方案。

当不划分缓存窗时，卷积和池化均以整个输入特征图为单位进行，这时对于某些卷积结果边长为奇数的层，只需进行半边式的padding（如图3-9）即可；然而在将输入特征图划分为多个缓存窗之后，卷积结果也被相应划分为多个区域（称为卷积结果窗），假设每个缓存窗对应的卷积结果边长为，完整的结果图边长为。

当为偶数时，也为偶数，这时每个卷积结果窗都不需要padding，如图3-10所示；当为奇数时，无论是否为偶数，在每个缓存窗对应的结果窗的边缘，缓存窗的划分都将原本连续的池化过程打断。这时如果每个卷积结果窗都采用相同的池化方式，则会造成池化结果的冗余和缺漏。因此本设计中定义了4中特殊的padding方式：00型、01型、10型和11型，如图3-11所示。

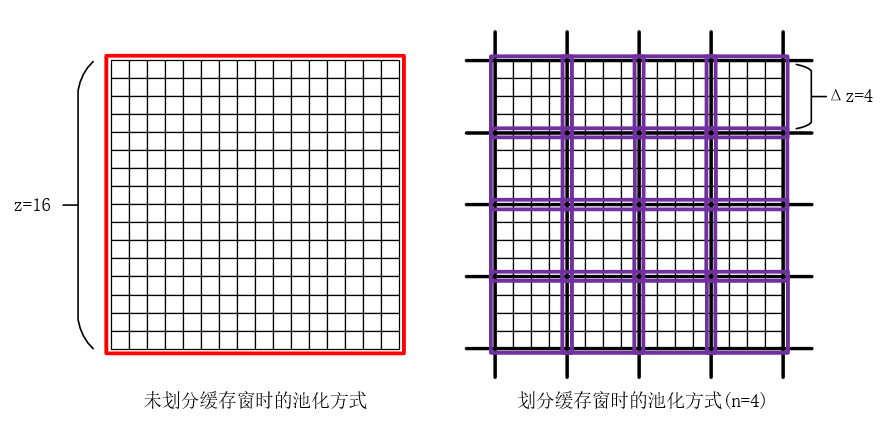


图 3-10 为偶数时的池化方式

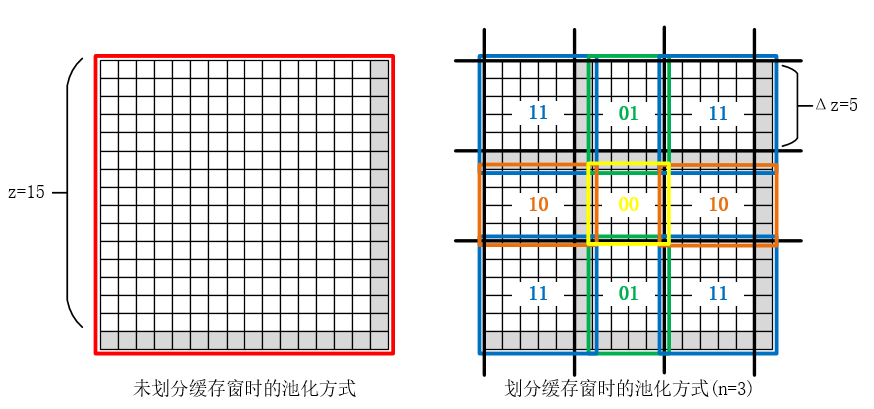


图 3-11 为奇数时的池化方式

采用这种定义，对同一个输入特征图得到的不同的卷积结果窗进行不同方式的池化，保证了池化结果的完整性。然而这种方式会造成结果窗某些信息的丢失，在缓存窗数量不多的情况下，考虑CNN网络的高容错性能，该误差基本可以忽略。

为了实现这种多方式池化，本设计中为卷积池化算子引入pad \_style这一控制信息，并对池化状态的内部计数器和用于生成RB访问地址的组合逻辑进行了补充，使其能够对不同的pad\_style产生不同的池化行为。

3.2.6 FSM设计

卷积池化算子的运行由两个Master\_FSM和Slave\_FSM共同跟进。前者包括p\_Nif、p\_nw、p\_nh和p\_k四个嵌套的计数器，分别用于指示当前计算所处的输入通道、缓存窗在输入特征图中的位置以及当前处在的pass（见表3-1）；后者包括conv\_FSM和pool\_FSM两个并行的计数器，conv\_FSM用于指示当前的卷积状态，分别为idle、wr\_fb（写FB）、wr\_wb（写WB）和calcu（卷积计算），pool\_FSM用于指示当前的池化状态，分别为idle、pre\_back（预写回）、pool（池化）和back（写回）。

上述FSM的时序图与状态转换图如图3-12、3-13、3-14。

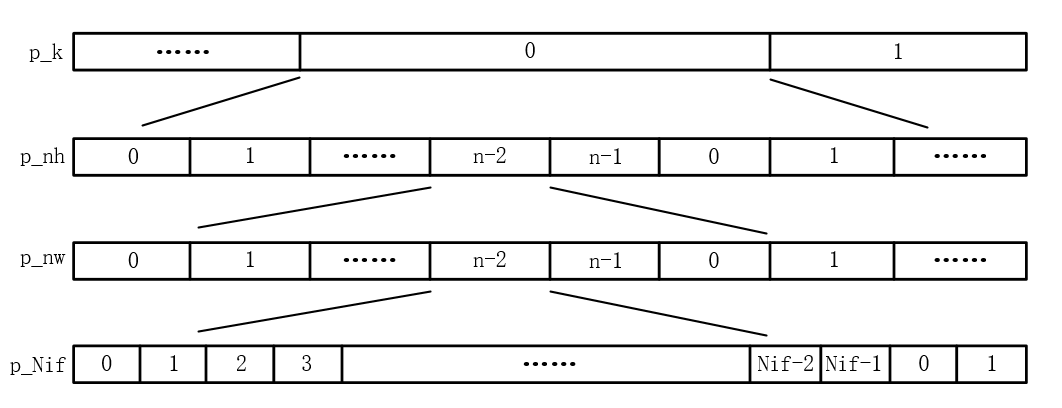


图 3-12 Master\_FSM时序图

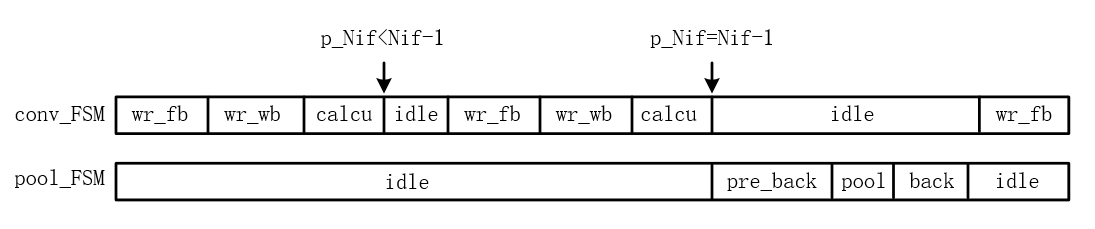


图 3-13 Slave\_FSM时序图

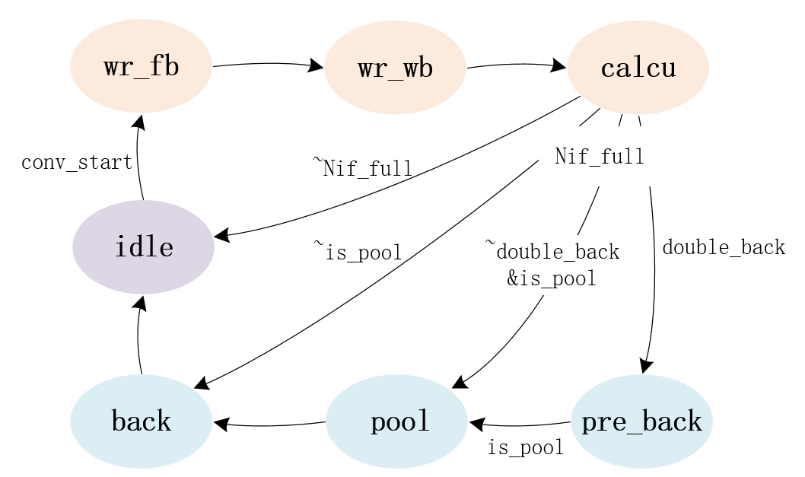


图 3-14 Slave\_FSM状态转换图

3.3 后处理（NMS）算子设计

3.3.1 NMS算法介绍

NMS（Non Maximum Suppression）即非极大值抑制，是应用于目标检测领域的深度学习常见的后处理算法。当得到多个目标定位预测框时，可能出现多个预测框表示同一个目标的情况，这时通过不断地选出最优框和剔除重叠框，就可以筛选出所有能够唯一表示每个目标的预测框。其一般执行流程如下：

①筛选出置信度大于阈值的所有预测框，并进行坐标解码

②对上述预测框按照置信度排序

③选出置信度最大的预测框作为最优框

④计算其余每一个预测框与最优框的交并比（IoU）, IoU大于阈值的记为重叠框

⑤记录最优框至结果，然后剔除最优框和所有的重叠框

⑥如果剩余预测框数量小于等于1，结束；否则，跳转至③

⑦return结果

图3-15展示了NMS的执行效果。

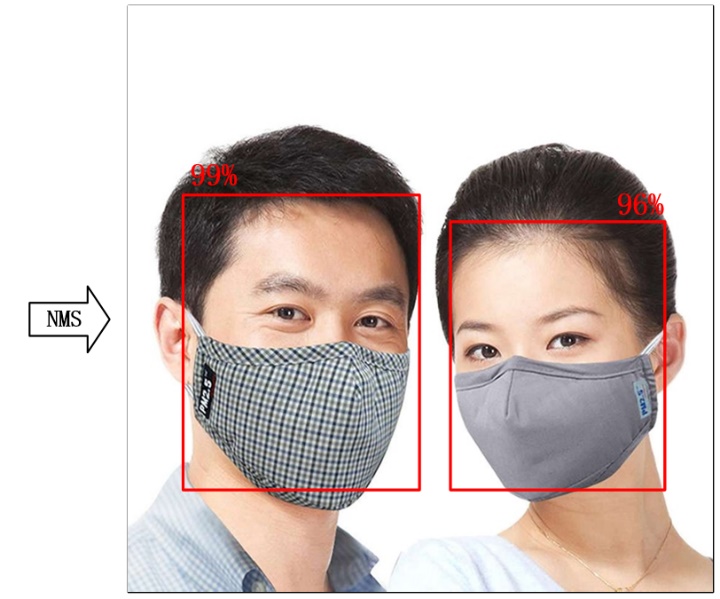


图 3-15 NMS执行效果图

3.3.2 后处理算子结构与功能

本设计中采用的NMS总流程分为四个状态：置信度筛选、置信度排序、预测框解码和非极大值抑制，四个状态分时运行，每个状态内部是并行或流水线执行的。

|  |  |
| --- | --- |
| 置信度筛选 | 从DDR中读取网络计算得到的原始置信度矩阵[5972,2] (5972为数量，2表示mask和no mask两个类别) |
| 通过纵向取最大值得到[5972,1]矩阵，同时利用置信度阈值筛选该矩阵，获得置信度数组cls[m,1]以及相应的类别数组class[m,1]和索引数组indx[m,1]（m为筛选后得到的元素数量） |
| 将cls、class和indx分别存入cls\_RAM和indx\_RAM |
| 置信度排序 | 对cls和class同时按照cls中置信度的大小进行冒泡排序 |
| 预测框解码 | 从indx\_RAM中读出indx数组，作为地址从DDR中读取原始的预测框偏移量矩阵 |
| 对照标准anchor反算得出预测框归一化的坐标矩阵loc（包括xmin，ymin，xmax和ymax四个位置信息）和每个预测框的归一化面积area |
| 将loc中每个预测框的xmin，ymin，xmax，ymax和area，连同indx存入main\_RAM中 |
| 非极大值抑制 | 通过不断地读写main\_RAM、计算和比较，最终得到NMS结果，并将其写入结果寄存器堆中 |

表3-2 硬件后处理执行流程

后处理算子的硬件结构如图3-16所示，它的每个功能区域对应表3-2中的一项功能。首先置信度数据通过DATA\_IN进入算子进行比较和筛选，筛选得到的数据连同其索引indx和类别class存放入相应的两个RAM中，idx\_cnt保存的为保留下来的预测框个数。在对cls RAM和indx RAM中的数据按照置信度排序后，按照indx RAM的输出依次从DDR中读取位置信息同时进行解码，解码公式由式至式给出。anchor ROM0和anchor ROM1中存放的分别为、和、。解码后的四个位置信息h、w、cx和cy被依次存放进四个首尾相连的寄存器中，当四个位置信息排列整齐后，通过组合逻辑生成新的五个位置信息xmin、ymin、xmax、ymax和area并存放进main RAM中。之后便交给NMS硬件算法，算法所得结果经过防边界溢出处理和绝对化之后被写入ACC的结果寄存器堆。

值得一提的是，这里为了减小产生时序问题的可能，回归框解码和NMS的计算电路都采用了两级并行流水线。

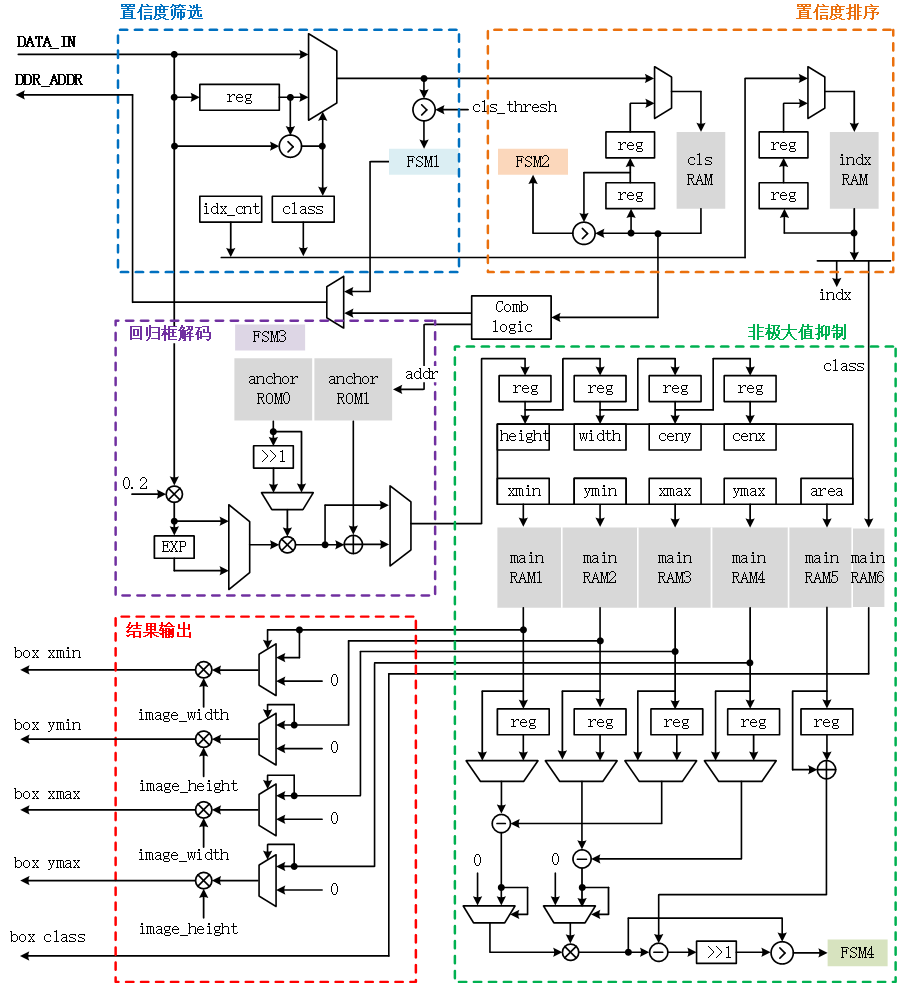


图 3-16后处理算子结构

3.3.3 硬件冒泡排序

硬件冒泡排序包括两个状态：读cls RAM和写cls RAM。初始时为读cls RAM状态，该RAM的地址按1递增，读出的数据被放入一级寄存器reg1，因此RAM的输出端dout和reg1中总是两个相邻地址的置信度值，如图24。当比较器检测到reg1小于dout时，FSM进入写cls RAM状态，这时原本递增的地址保持不变并被禁用，在原本地址基础上生成的交换地址被激活。两个时钟周期后，相邻的两个置信度被交换地址，FSM返回到读cls RAM状态，递增地址重新被激活，比较器继续检测下一组相邻的置信度。由于状态的转换需要延迟一个时钟周期，这里增加了二级寄存器reg2来匹配时序。

3.3.4 硬件NMS

软件实现NMS伪代码如下：

pick = [] #定义数组用于存放NMS结果

while box.length > 0:

#box初始为[m,4]数组，4代表预测框的4个位置信息，m个框已按照置信度从大到小排序

pick.append(box[0]) #先把最优框的保留下来

xmin=box[:,0] #得到每个框的左上角x坐标

ymin=box[:,1] #得到每个框的左上角y坐标

xmax=box[:,2] #得到每个框的右下角x坐标

ymax=box[:,3] #得到每个框的右下角y坐标

overlap\_xmin = max(xmin[0], xmin[1:]) #得到最优框与其他每个框重叠区的左上角x坐标

overlap\_ymin = max(ymin[0], ymin[1:]) #得到最优框与其他每个框重叠区的左上角y坐标

overlap\_xmax = min(xmax[0], xmax[1:]) #得到最优框与其他每个框重叠区的右下角x坐标

overlap\_ymax = min(ymax[0], ymax[1:]) #得到最优框与其他每个框重叠区的右下角y坐标

overlap\_w = np.maximum(0, overlap\_xmax - overlap\_xmin) #重叠区宽

overlap\_h = np.maximum(0, overlap\_ymax - overlap\_ymin) #重叠区高

overlap\_area = overlap\_w \* overlap\_h #重叠区面积

overlap\_ratio = overlap\_area / (area[0] + area[1:] - overlap\_area) #计算IoU

need\_to\_be\_deleted = (box[0] , box[where(overlap\_ratio>threshold)+1])

box = box.delete(need\_to\_be\_deleted) #删除最优框和重叠框

本设计中硬件NMS的运行状态如图3-17所示。

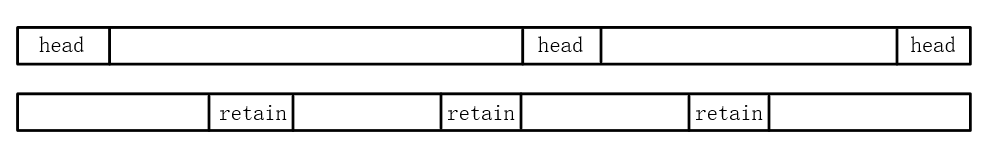


图 3-17 硬件NMS运行状态

在每一轮检测开始时，FSM进入head状态，如图3-17，从main RAM中读出的位置（面积）信息被存放至寄存器中，作为最优框；同时，这些信息连同class被送至NMS算子外部ACC的结果寄存器堆中。一个时钟周期后退出head状态，依次读取main RAM中其他框的信息，其坐标、面积的计算采用二级并行流水线。值得注意的是，这里采用IoU阈值为0.5，只需进行简单的移位和比较便可实现，避免了不必要的除法运算。当检测到IoU小于阈值时，FSM进入retain状态，将此时main RAM的输出信息重新写回至指定地址处，意为保留。一个周期后退出retain状态，继续检测IoU。当有多组信息需要保留时，其写回地址依次递增，这样在一轮检测执行结束时，新的box数组便自然诞生，其长度便为上一轮检测时最后一次写回的地址值。

如此反复，当检测到新的数组长度等于0时，发出nms\_done信号，标志着NMS结束；当检测到新的数组长度为1时，首先进入head状态，然后发出nms\_done信号。

3.4 适应模型的DDR分区与地址生成

3.4.1 DDR分区

本设计采用的网络模型分为主干和分支两部分，多个分支理论上可以并行运行，但是由于卷积池化算子同一时刻只能计算一个卷积池化层，所以包括分支在内的28个卷积池化层必须分时顺序进行，因此定义合适的网络的执行顺序是必要的。  
 我们定义网络分支优先的执行顺序，即在网络分支处，先执行两个分支线路，当分支网络执行完毕，得到最终结果后，返回执行主干线路。基于这种定义，现为DDR划分区域如表3-3：

|  |  |  |
| --- | --- | --- |
| 名称 | 包含子区域 | 作用 |
| W（weights） | 28 | 存储28个卷积层的权重 |
| IN（image\_in） |  | 存储输入图片 |
| T（trunk） | 2 | 在网络主干上用于“乒乓”存储中间结果 |
| B（branch） | 2 | 在网络分支上用于“乒乓”存储中间结果 |
| R（results） | 10 | 存储网络计算结果，即置信度和坐标信息 |

表 3-3 DDR分区

3.4.2 地址生成策略

当ACC作为Master访问DDR时，其访问地址采用基地址-起始地址-实时地址的三级地址生成策略。

表3-3中每个子区域都对应一个基地址。在卷积池化计算时，需要得知权重、输入特征图以及计算结果存放的区域，这时只需CPU在当前层运算之前向ACC的6个数据寄存器写入相应区域的基地址，作为整层运算的基地址。由于一层的运算分多个轮次进行，每轮计算的DDR访问起始地址都在基地址的基础上，通过p\_k、p\_nh、p\_nw和p\_Nif四个计数器生成。而访存时的实时地址都在起始地址的基础上由s\_FSM内部的计数器生成。

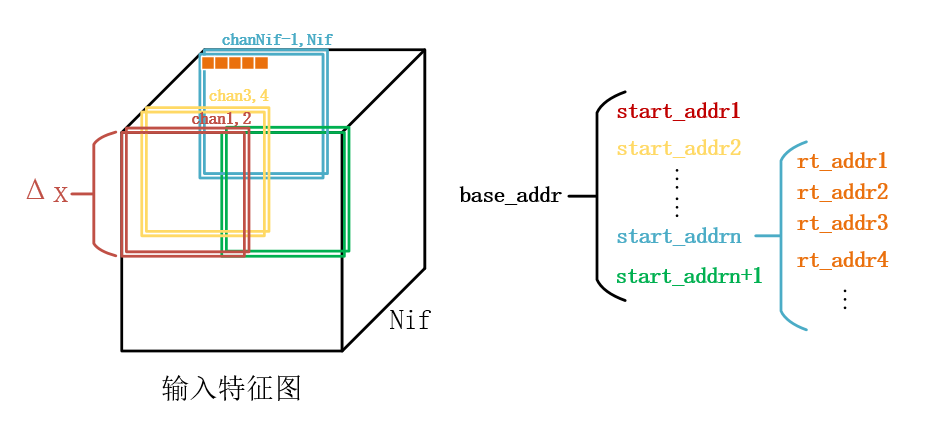


图 3-18 基地址-起始地址-实时地址三级地址生成策略

4 软件设计

4.1 中断请求信号

Cortex-M3使用了4种IRQ中断：

* 按键中断。由FPGA开发板上的按键经过消抖与上升沿检测得到；
* 计算中断。卷积计算结束（conv\_done），卷积池化算子的计算结束标志；未检测到目标（no\_face\_detected），标志着本次推理未检测到人脸；结果可读（you\_can\_rd\_result），标志着推理的结束，告知CPU结果寄存器堆可读；
* DMAC中断。由于Bus Matrix为静态仲裁，需要用WFI指令配合DMAC，使核主动交出总线控制权，因此DMAC工作完成后需要用中断唤醒核；
* UART中断。UART接收模块没有缓冲区，因此使用中断。

4.2 ACC控制流程

ACC硬件计算在整个推理计算过程中占据绝大部分，不需要太多的软件参与，但是需要软件在每层计算结束后配置下一层计算的控制信息，因此推理流程其实是中断服务程序的堆砌。

卷积池化层共28层，每层需要配置的控制寄存器内容不同，如果给每层网络设以单独的中断请求，那将会造成资源的浪费。因此我们只定义一个中断请求，使用软件计数＋分支选择的方式来实现对不同的层配置不同的控制信息。

当后处理运算筛选置信度结束时，如果idx\_cnt为0，代表没有人脸检测到，则返回IDLE状态，并产生no\_face\_detected中断请求。如果idx\_cnt不为0，则继续进行运算，直到产生结果，发出you\_can\_rd\_result请求。当CPU读取结果之后，需要重新写一次ACC的控制寄存器，以唤醒后处理算子的状态机，使其返回IDLE状态。ACC控制流程如图4-1所示。

当CPU读取结果寄存器堆时，需要首先判断每个单元的is\_valid位，若有效，则代表当前结果单元存放的为有效结果。读取结果流程如图4-2所示。

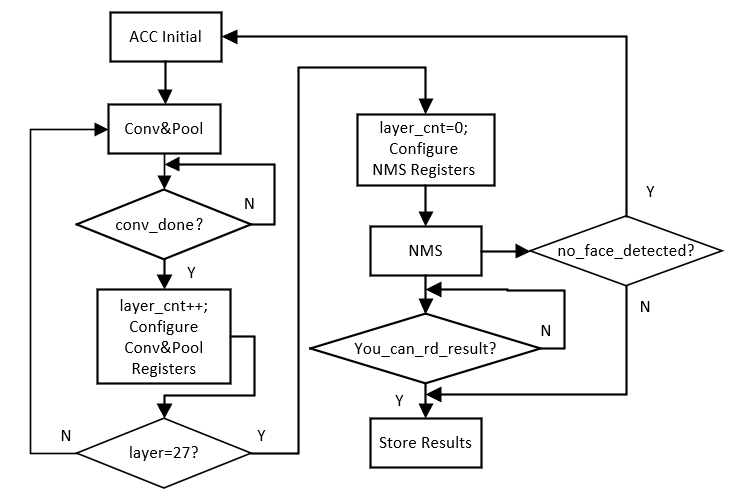


图 4-1 ACC软件控制流程

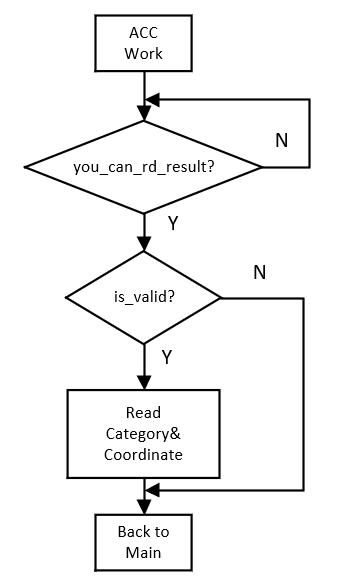
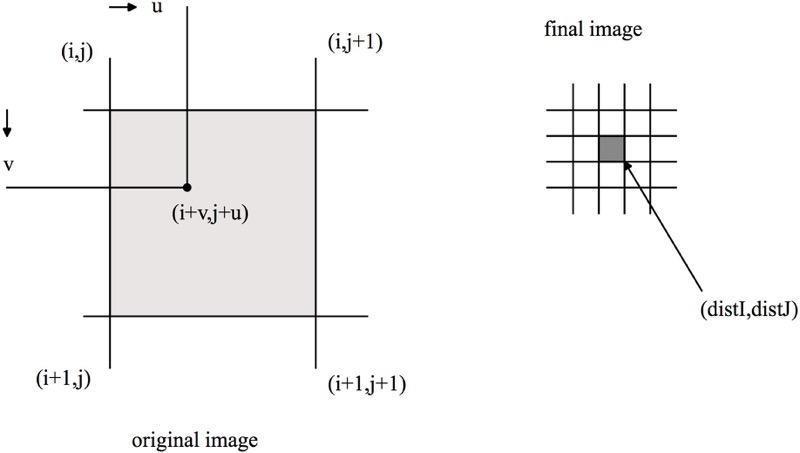


图 4-2 CPU读取结果流程

4.3 图像缩放

图片包含 游戏机

描述已自动生成图像处理中有三种常用的插值算法：最邻近插值、双线性插值和双立方（三次卷积）插值。相比于最邻近插值的粗糙和双立方插值的计算量大，双线性插值的效果比较折中，它计算量不是那么巨大，效果也比较好，可以作为应用中默认的图像处理算法。双线性插值计算较为简单，只涉及邻近的4个像素点，如图4-2所示

图4-4 图像缩放流程

图4-3 双线性插值图像映射关系

目标插值图中的某像素点（dstI，dstJ）在原图中的映射为（i+v，j+u）,而（i+v，j+u）处的值就是邻近4个像素点分别在x轴和y轴的权值和。其中u、v分别是列方向和行方向的偏差。则可得插值公式为

在本系统中具体应用双线性插值进行图像处理的流程如图4-4所示，将从摄像头采集到的RGB565格式的图像数据先按照RGB通道分开，然后再对每个通道进行图像处理，最后将处理后的数据交给ACC计算。

4.4系统工作流程

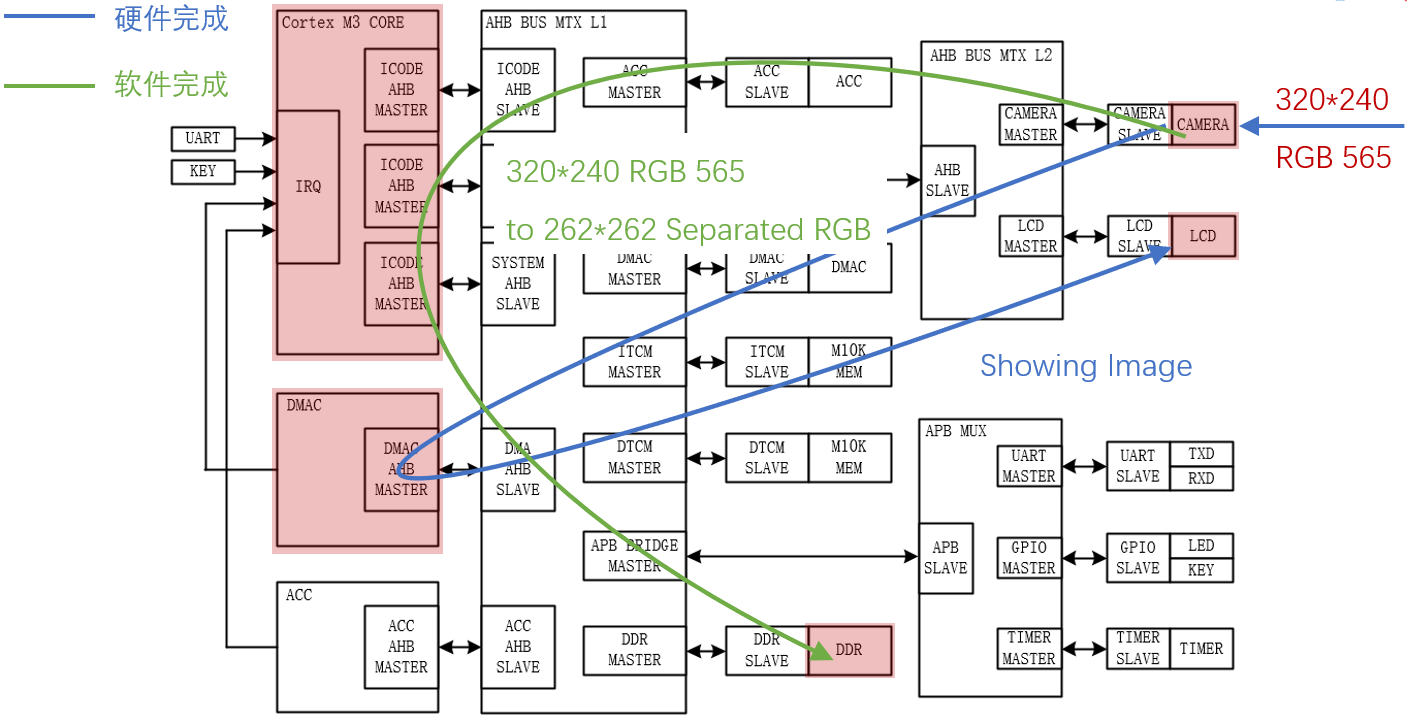
首先，CPU配置摄像头，开启一帧的捕捉，摄像头将捕获到的图片数据（RGB565）缓存在一个本地RAM中，然后CPU从该RAM读出数据，分散到RGB三个通道、缩放并进行16bit量化后写到DDR相应区域。当ACC开始计算后，DMA将该RAM中的RGB565数据搬运到LCD进行显示。

图 4-5 Step1图像捕捉与缩放

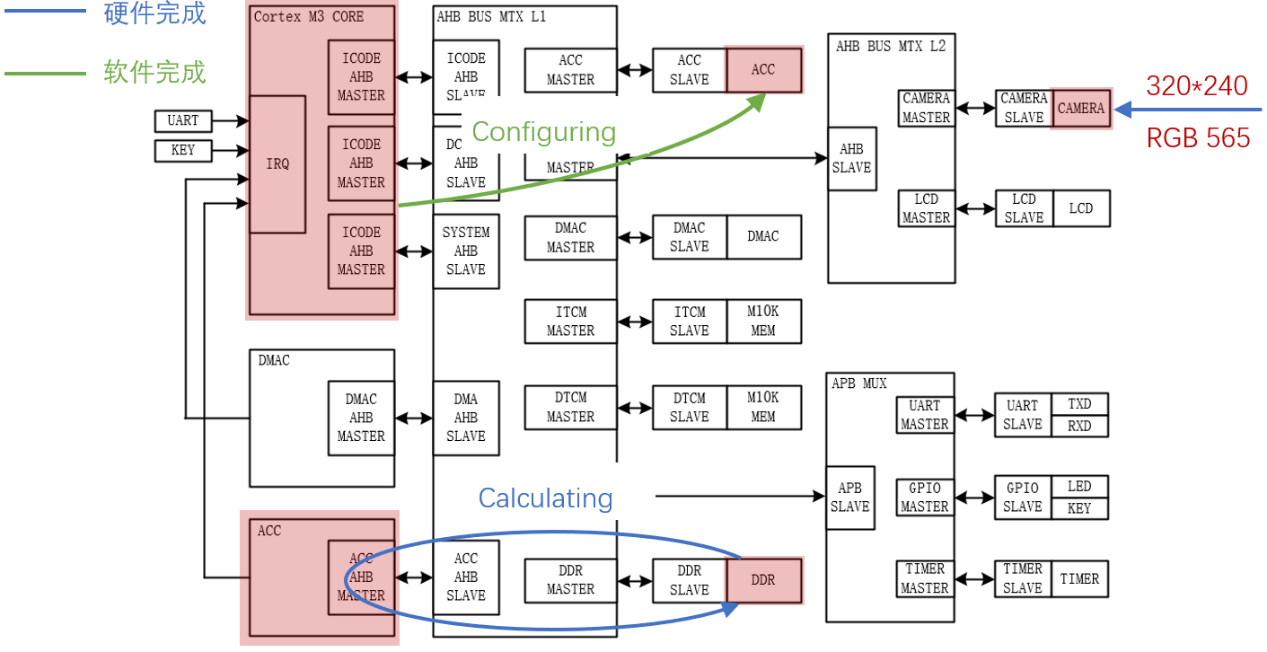
ACC的计算和DMA搬运数据显示是同时进行的，由于ACC计算耗时较长，当DMA搬运完毕后，这时摄像头本地RAM中的数据已经失效，CPU重新配置摄像头，开启新的一帧的检测。

图 4-6 Step2 ACC计算

ACC计算结束后，CPU从ACC的结果寄存器堆中读取计算结果，将其值直接显示在LCD上，一帧的识别到此结束。

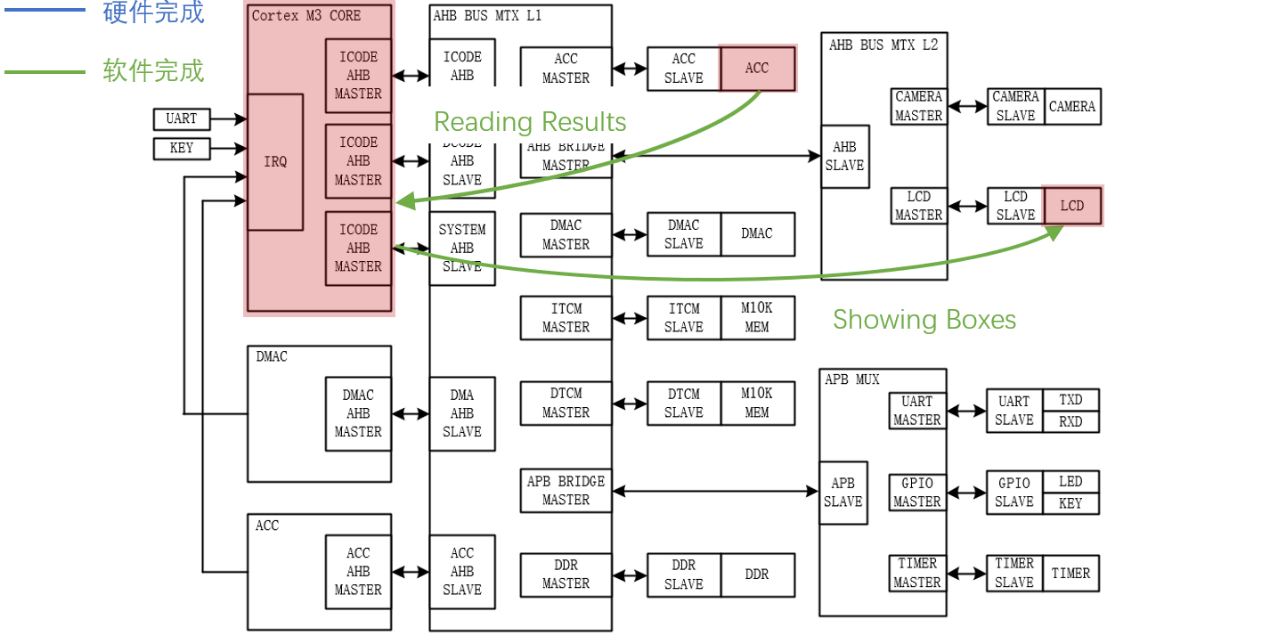


图 4-7 Step3 结果读取与显示

值得一提的是，这里为了提升单帧识别速度，采用了多过程“并行运行”的方式，如ACC计算与DMA搬运的并行运行、ACC计算与摄像头采集的并行运行、甚至是ACC计算与下一帧图片缩放的并行运行，之所以能够实现这种功能，是因为ACC计算的大部分工作都由硬件自动完成，而CPU处于空闲状态，因此可以用来控制、调度其他过程。然而，图片缩放需要CPU不断地访问DDR，这与ACC访问DDR产生了冲突，这时只需要设置合理的总线仲裁优先级，便可以做到有条不紊，最大程度地节省系统运行时间。

5 仿真与测试

5.1 仿真

5.1.1 推理时间计算与仿真验证

编写Python脚本对推理总时间进行估算，得出在50MHz的系统时钟条件下，单帧推理时间为0.382s左右。之后使用vivado 的testbench对第24层计算进行仿真，测得的计算时间与脚本得出的结果基本一致。

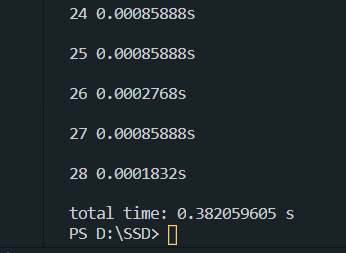
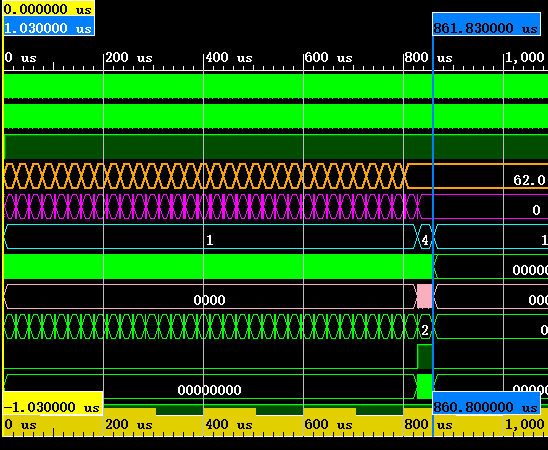
 

图 5-1 Python计算（左）和testbench仿真第24层（右）

5.1.2 卷积池化算子仿真

图5-2展示了卷积池化的状态机仿真，见p\_Nif、conv\_status、pool\_status三个状态机。HADDRM为ACC访问DDR的AHB总线地址。图5-3展示了第24卷积池化层的仿真，其中，conv\_out\_bus为写回的结果数据，cnt11-13代表当前正在返回的数据在输出特征图中所处的列、行和通道。图5-4展示了在Pytorch框架上推理得到的网络输出值，将其与conv\_ out\_bus对比可以发现，两者基本一致。（仿真/推理文件请见附件）

另外，我们将vivado testbench仿真得到的卷积池化计算结果和Pytorch上推理得到的结果分别导出到txt文件中，除去0值进行对比，得到了如图5-5所示的结果（数据已放大1024倍），可以看到，两者基本一致。

我们还使用Python对仿真计算结果中的1200个数据进行了统计，得出其与Pytorch上得到的实际结果平均相对误差仅为0.35%（见图5-6）。

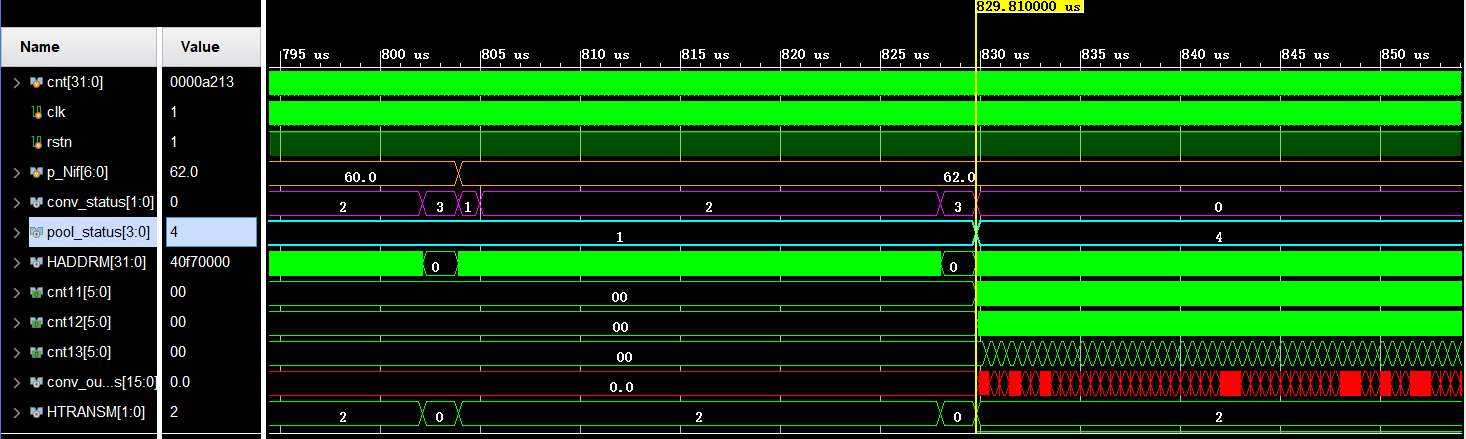


图 5-2 卷积池化算子状态机

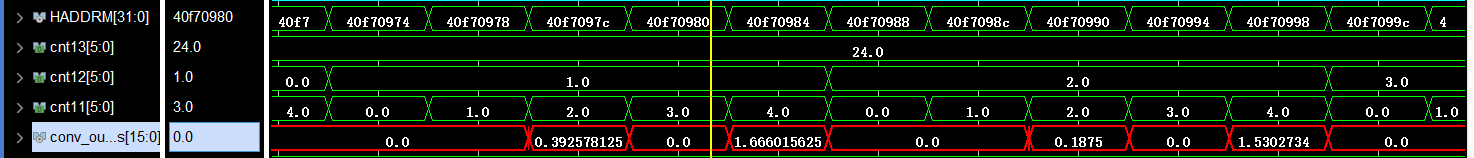




图 5-3 Layer24卷积池化算子仿真之计算结果

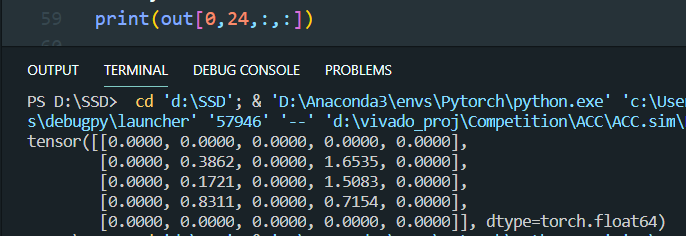


图 5-4 Layer24 Pytorch框架上推理计算结果

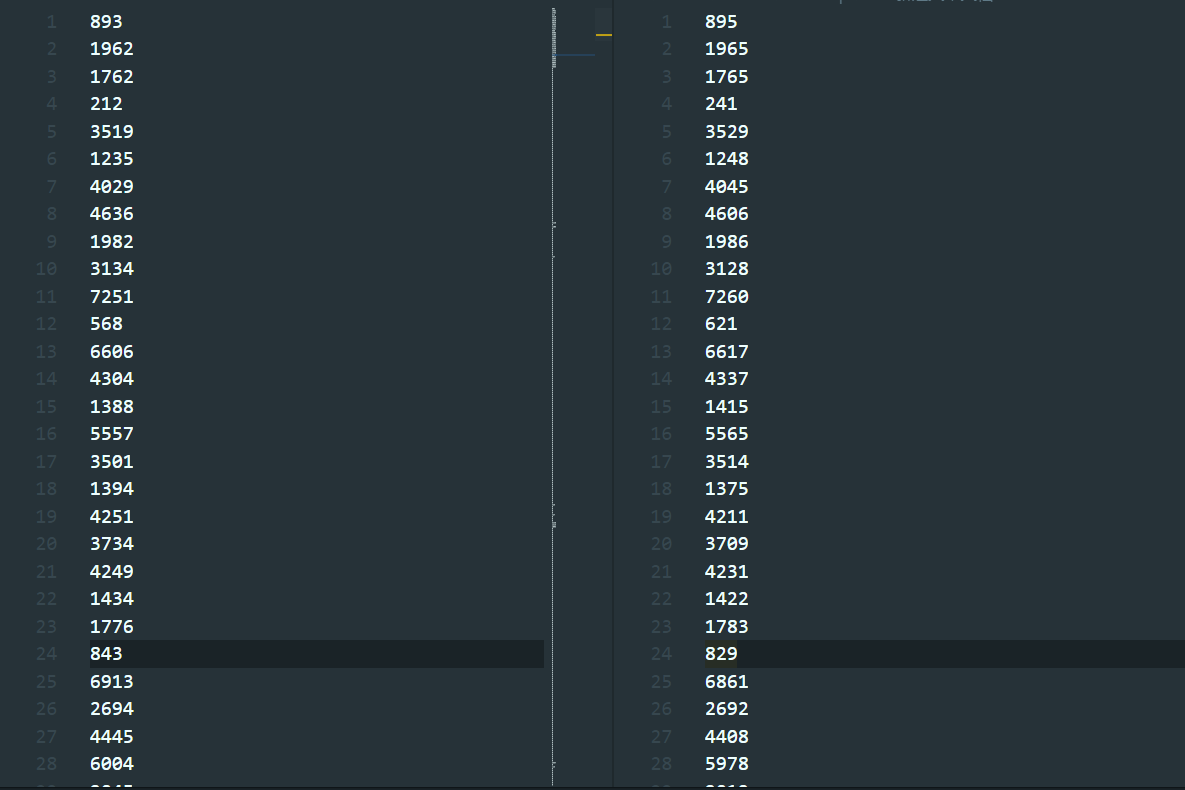


图 5-5 Pytorch上推理（左）和testbench仿真（右）

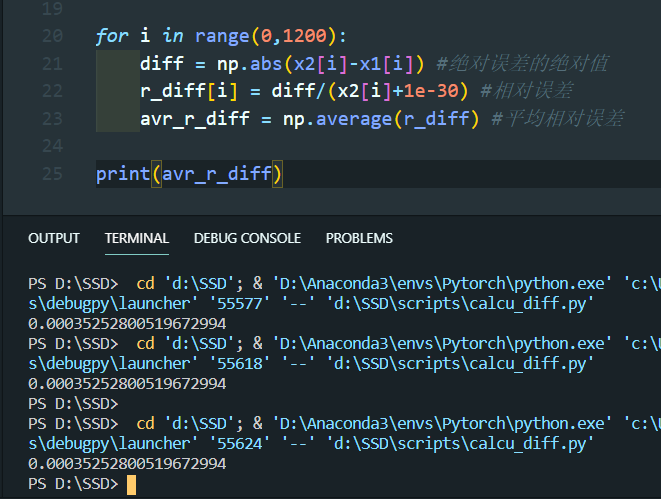
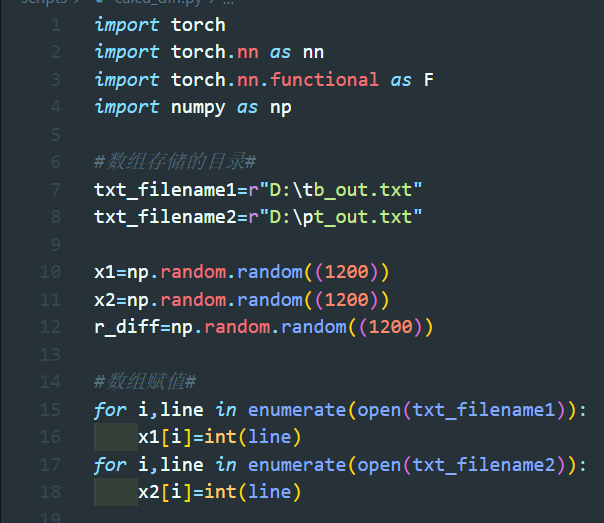


图 5-6 使用Python计算仿真结果的相对误差

仿真结果表明，卷积池化算子在进行运算时能够始终遵循AHB总线协议对DDR进行访问，且能够正确处理HREADYOUT拉低等突发情况，在此不再一一展示。

5.1.3 后处理算子仿真

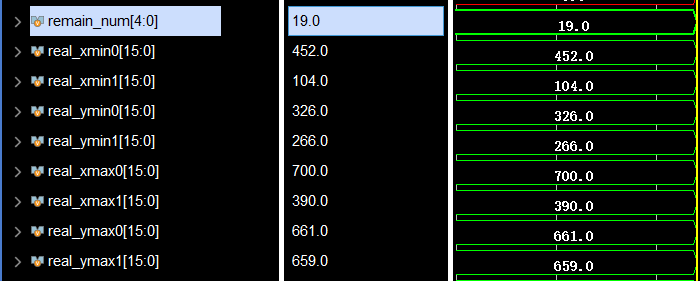


图 5-7 后处理算子的testbench仿真结果

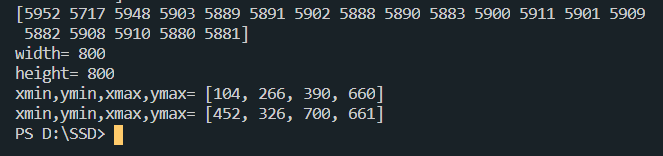


图 5-8 Pytorch上的后处理计算结果

我们将Pytoch上网络输出的置信度和预测框作为输入对后处理算子进行了仿真。仿真结果表明（图5-7，图5-8），后处理算子能够正确运算得出预测框的四个坐标值结果，并将结果写入ACC的结果寄存器堆。其计算结果与Pytor- ch上得出的结果几乎一致。另外，后处理算子能够正确产生no\_face\_detected和you\_can\_rd\_result两个中断请求信号。

5.1.4 DMAC仿真

通过给AHB总线特定的信号以实现从首地址为0x000F0000的地址空间传输一个包含20个32位数的数据块到首地址为0x000E0000的地址空间。经Vivado仿真，DMAC工作正常。仿真波形如下：电脑萤幕画面

描述已自动生成

图5-9 DMAC仿真波形

5.2 测试

5.2.1 UART测试

通过将Retarget.c程序移植到工程实现了printf和scanf函数重定向到UART，从而可以配合PC端的minicom进行UART读写，编写main函数如图5-10。它所实现的功能是在SoC启动时打印启动信息，并且将uart接收到的字符串重新发送两边，测试结果如图5-11。

屏幕上有字

描述已自动生成

图5-10 UART测试程序

手机屏幕的截图

描述已自动生成

图5-11 UART测试结果

5.2.2 DDR读写测试

编写汇编程序对DDR进行读写操作，从DDR首地址起写入15个32位数并将其一一读出，编写汇编程序如图5-12，测试结果表明，DDR能够正常被读写，说明我们的DDR访问接口时工作正常的，如图5-13。

电脑屏幕截图

描述已自动生成

图5-12 DDR测试汇编程序

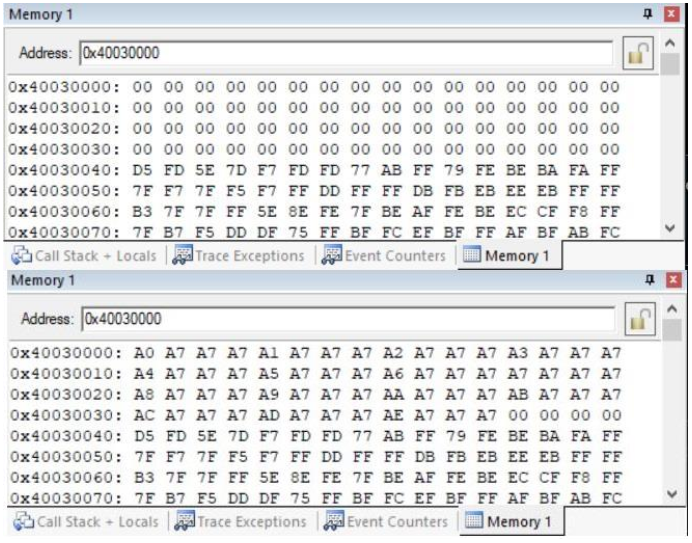


图5-13 DDR测试结果

5.2.3 ACC测试

我们对ACC的每个卷积池化层进行了测试。首先，在Pytorch上输入一张图片，获得网络特定层的中间结果和权重，将其导出并通过UART和CPU写到DDR的特定区域，作为测试层的输入。测试结果表明，ACC每个卷积池化层结果均与仿真完全一致，在此我们选用了第24层的结果加以展示，如图5-14和图5-15所示。

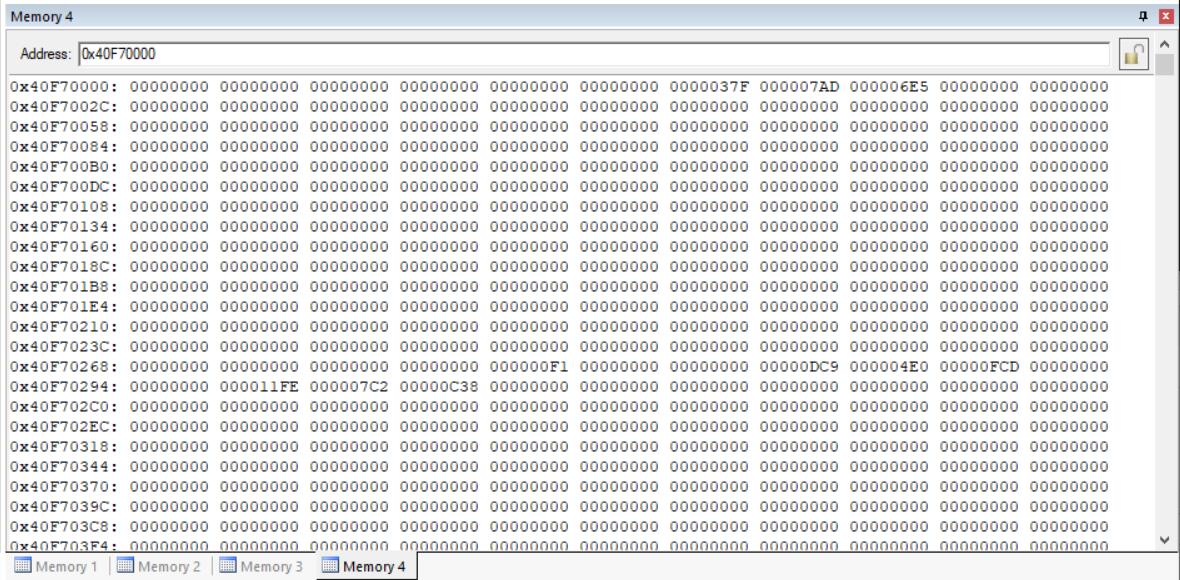


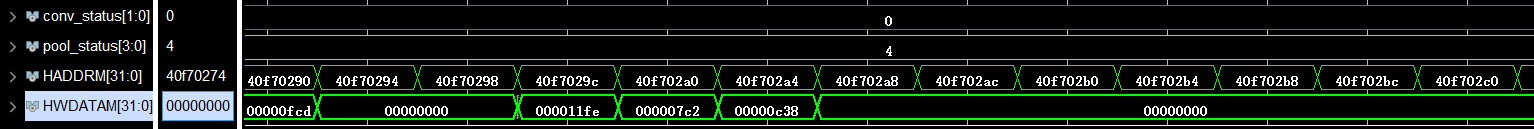
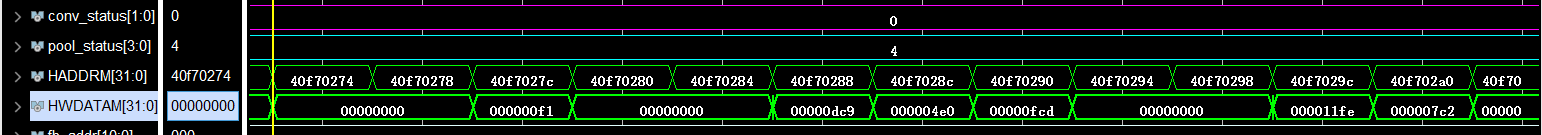
图 5-14 ACC Layer24实测结果

图 5-15 ACC Layer24仿真结果

之后，我们对ACC的单帧推理功能进行了测试。我们将Pytorch上经缩放后得到的262\*262尺寸的图片导出并通过UART发送至DDR的IN区，同时导出网络所有权重参数至DDR的W区。开启ACC，计算结束后查看结果寄存器堆的值。反复实验多次，在原图尺寸为800\*800的条件下，发现结果误差基本上不大于1，即回归框的位置与实际位置仅差0-1个像素。

这说明通过28层的运算，ACC能够将结果误差控制在0.2%以内，毫无疑问，该误差大小完全满足市场的精度需求。

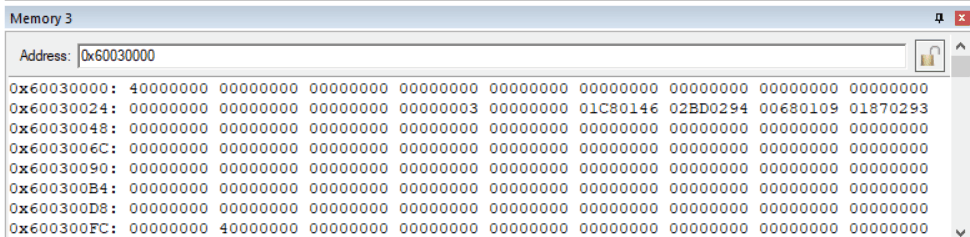


图5-16 图A推理结果 Pytorch（上）FPGA（下）



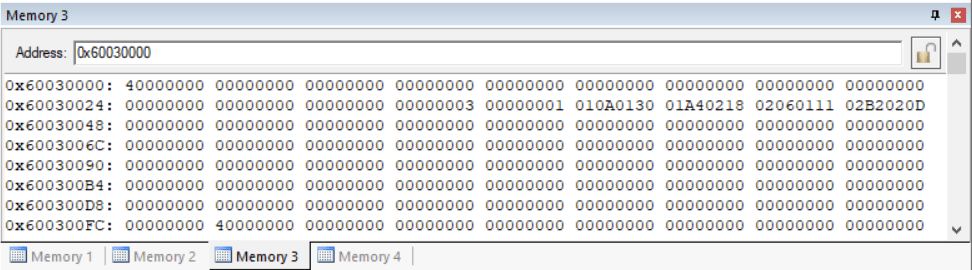


图5-17 图B推理结果 Pytorch（上）FPGA（下）

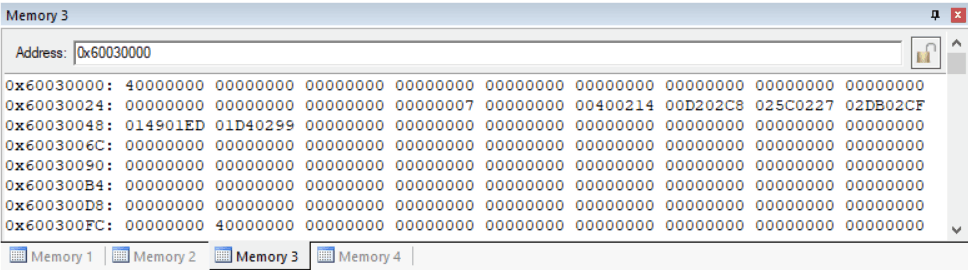
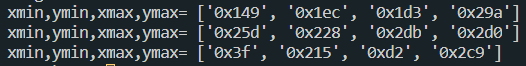


图5-18 图C推理结果 Pytorch（上）FPGA（下）

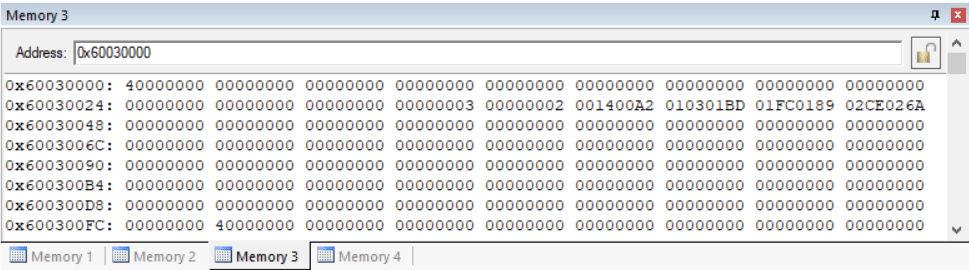


图5-19 图D推理结果 Pytorch（上）FPGA（下）

6 总结与展望

我们设计出了一款用于口罩佩戴识别的前端芯片，它基于SSD卷积神经网络，采用了轻量化VGG-16模型。我们针对此模型设计了相应的硬件加速器，它包括卷积加速器和后处理加速器两部分，实现了高时效、全硬件化的计算。

针对计算精度问题我们做出了三个方面的优化：参数导出四舍五入、乘累加四舍五入和后处理定点数动点法。经实测，优化后每层网络计算相对误差约为0.35%，推理总相对误差在0.2%以内。针对cordic IP核带来的大时序延迟，我们采用了状态机等待法做出了时序优化，给IP增加流水级，保证了系统在50MHz频率下的正常运行。针对原始图片与网络输入在比例尺寸、量化格式上的区别，我们采用线性插值法，使用软件对原始图片进行缩放与量化，保证了网络正常运行。在此基础上我们还提出了软件“并行化”，缩短了推理时间。

我们将ACC与DDR、CortexM3、摄像头、LCD和自主设计的DMAC等模块组合起来，搭建了SoC，成功实现了单张图片的推理与显示。

然而，该系统当前仍然有待于进一步优化：

1.尽管采用了轻量化网络模型，其参数和中间结果仍存在大量的0值，这说明网络结构还可以进一步简化，其推理时间和占用资源也将进一步减小。

2.目前DMAC与ACC访问DDR均只采用NSEQ的方式，由此带来的HR-EADYOUT低电平等待时间增大了时耗。可以通过进一步优化硬件，对连续地址采用BURST传输，进一步缩短推理时间。

3.目前ACC的读写size为16bit，及当ACC读写DDR时，高16bit是无效的，若修改网络参数在DDR中的存储方式，将16bit传输扩展至32bit传输，可进一步缩短推理时间。

接下来，我们将针对上述三个问题进行优化与完善，希望我们能够在总决赛中展现更加完美的系统功能！

致谢

感谢老师五个月以来的悉心陪伴与指导！

感谢AIZOOTech的开源项目FaceMaskDetection！

参考文献

[1]秦华标,曹钦平.基于FPGA的卷积神经网络硬件加速器设计[J].电子与信息学报,2019,41 (11):2599-2605.  
[2]童耀宗. 基于FPGA的卷积神经网络加速器的设计与实现[D].XX大学,2019.  
[3]Chen, Xiaoming, et al.”Communication Lower Bound in Convolution Accelerators.”2019.  
[4]ARM IHI 0033A，AMBA3 AHB-Lite Protocol[S]  
[5]ARM IHI 0024B，AMBA3 APB Protocol[S]  
[6]ARM DDI 0479C，Cortex-M System Design Kit[S]  
[7]ARM 100165\_0201\_01\_en，ARM Cortex-M3 Processor[S]