分类号:TN492　　学校代码:10363
密　级:公开　　学　　号:2210330143

安徽工程大学
Anhui Polytechnic University

硕士学位论文

题目　基于深度学习的目标检测系统设计与实现

论文作者　章咸斌
指导教师　张肖强
学科（专业）　电子信息
研究方向　系统集成技术及应用

论文提交日期：2024 年 5 月 31 日

分类号：TN492　　单位代码：10363
密　级：公开　　学　　号：2210330143

题　　目 基于深度学习的目标检测系统设计与实现

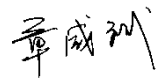
题　　目 DESIGN AND IMPLEMENTATION OF OBJECT DETECTION SYSTEM BASED ON DEEP LEARNING

学 生 姓 名：章咸斌
校 内 导 师：张肖强（副教授）
校 外 导 师：朱标（高级工程师）
专业学位类别：电子信息（085400）
研究方向(领域)：系统集成技术与应用

论文答辩日期：2024 年 5 月 22 日

二、安徽工程大学学位论文原创性声明

本人郑重声明：我恪守学术道德，崇尚严谨学风。所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已明确注明和引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品及成果的内容。论文为本人亲自撰写，我对所写的内容负责，并完全意识到本声明的法律后果由本人承担。

学位论文作者签名： 

日期：2024 年 5 月 31 日

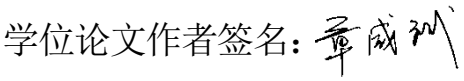
三、安徽工程大学学位论文版权使用授权书

学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅或借阅。本人授权安徽工程大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密 □，在 ____ 年解密后适用本版权书。

本学位论文属于

不保密 ☑。

学位论文作者签名：  　　指导教师签名：

日期：2024 年 5 月 31 日　　日期：2024 年 5 月 31 日

基于深度学习的目标检测系统设计与实现

摘 要

目标检测是计算机视觉领域中的一个重要研究方向，其主要任务是从图像或视频中自动识别并定位感兴趣的目标。随着人工智能时代的到来，基于深度学习的目标检测方法在实时性、准确性和鲁棒性等方面不断取得突破，并广泛应用于图像分析、视频监控、自动驾驶等领域。然而，深度学习作为一种计算密集型方法，对硬件计算能力有着极高的要求。目前，基于深度学习的目标检测大多依赖于计算型显卡来进行。但高性能的计算服务器价格昂贵，功耗较高，导致搭建成本高、扩展性差。因此，在综合考虑了各种硬件平台的优缺点后，本文选择 FPGA+ARM 架构的 ZYNQ 硬件平台来设计并实现了一个低功耗、低延迟的目标检测系统。本文的主要工作内容与创新点如下：

（1）针对 YOLOv3-tiny 中存在的漏检、重复检测以及误检的问题，提出了一种改进的 YOLOv3-tiny 算法。该算法在特征提取网络中添加了 3 个 3×3 的卷积层和 1×1 的卷积层，从而增强了特征的表达能力。此外，采用 K-means 聚类算法进行先验框的生成，使网络更容易学习。实验结果表明，该方法在保持与原 YOLOv3-tiny 相当的检测速度的同时，平均精度比原 YOLOv3-tiny 提高了 4.69%。

（2）搭建了通用卷积神经网络加速器架构，通过循环展开和乒乓缓存等优化技术实现了计算单元的加速。为了节省 FPGA 的存储资源，对输入特征图、片上权重、中间结果以及片外权重进行缓存设计。为进一步提高加速器性能，本文通过优化行缓存结构提高数据传输效率；通过乒乓技术和全流水设计降低系统延时。实验结果表明，在 250MHz 时钟

频率下，本设计的性能达到 144.2GOPS。与其他基于 FPGA 的研究工作相比，本文的加速器体现了较低的资源密度。

（3）设计并实现了目标检测系统的其他相关模块，如图像缩放模块、摄像头采集模块、图像显示模块和片上 Linux 系统等。为了验证系统的功能与性能，本文在 Xilinx 的 ZYNQ-XCZU3EG 平台上实现了改进的 YOLOv3-tiny 目标检测算法。实验结果表明，提出的系统能够准确检测出口罩和头盔，检测速度达到 18FPS，吞吐率、能效比和计算密度分别达到了 81.11GOPS、18.06GOPS/W 和 0.27GOPS/DSP。相比于现有设计，本系统具有低延时和高性能的优势。

关键词：目标检测，FPGA，ARM，ZYNQ，YOLOv3-tiny，卷积神经网络加速器

DESIGN AND IMPLEMENTATION OF TARGET DETECTION SYSTEM BASED ON DEEP LEARNING

ABSTRACT

Target detection is an important research direction in the field of computer vision, and its main task is to automatically identify and localize targets of interest from images or videos. With the advent of the artificial intelligence era, deep learning-based target detection methods have been making breakthroughs in real-time, accuracy and robustness, and are widely used in image analysis, video surveillance, automatic driving and other fields. However, deep learning, as a computationally intensive method, has extremely high requirements on hardware computational power. Currently, target detection based on deep learning mostly relies on computational graphics cards to perform. But high-performance computing servers are expensive and have high power consumption, resulting in high build cost and poor scalability. Therefore, after comprehensively considering the advantages and disadvantages of various hardware platforms, this paper selects the ZYNQ hardware platform with FPGA+ARM architecture to design and implement a low-power, low-latency target detection system. The main contents and innovations of this paper are as follows:

(1) An improved YOLOv3-tiny algorithm is proposed to address the problems of missed detection, repeated detection, and false detection in YOLOv3-tiny. The algorithm adds three 3×3 convolutional layers and 1×1 convolutional layers to the feature extraction network, which enhances the feature representation. In addition, the K-means clustering algorithm is used for a priori frame generation, which makes the network easier to learn. Experimental results show that the method improves the average accuracy by

5.69% over the original YOLOv3-tiny while maintaining a detection speed comparable to that of the original YOLOv3-tiny.

(2) A general-purpose convolutional neural network accelerator architecture is built, and the acceleration of computational units is achieved by optimization techniques such as loop unrolling and ping-pong caching. In order to save the storage resources of FPGA, the input feature map, on-chip weights, intermediate results, and off-chip weights are cached and designed. To further improve the gas pedal performance, this paper improves the data transfer efficiency by optimizing the line cache structure; and reduces the system delay by ping-pong technique and full-flow water design. Experimental results show that the performance of this design reaches 144.2GOPS at 250MHz clock frequency. Compared with other FPGA-based research works, the accelerator in this paper reflects a lower resource density.

(3) Other related modules of the target detection system, such as the image scaling module, the camera acquisition module, the image display module and the on-chip Linux system, are designed and implemented. In order to verify the functionality and performance of the system, this paper implements the improved YOLOv3-tiny target detection algorithm on Xilinx ZYNQ-XCZU3EG platform. The experimental results show that the proposed system is able to detect the mask and helmet correctly with a detection speed of 18 FPS, and the throughput rate, energy efficiency ratio, and computational density reach 81.11 GOPS, 18.06 GOPS/W, and 0.27 GOPS/DSP respectively. Compared with the existing designs, the proposed system has the advantages of low latency and high performance.

KEY WORDS: target detection, FPGA, ARM, ZYNQ, YOLOv3-tiny, convolutional neural network accelerator

目 录

第 1 章 绪论

1.1 研究背景及意义

目标检测作为计算机视觉领域的一个研究热点，旨在识别和定位图像或视频中的特定对象[1]。随着数字化和智能化技术的不断发展，目标检测技术已成为现代技术应用中不可或缺的一部分。在日常生活中，目标检测技术的应用已成为常态。如火车和高铁站的面部识别技术可以快速验证乘客身份；在安全监控系统中，其能够实时检测和跟踪人员或车辆；在医疗影像分析中，目标检测技术可以帮助医生更准确地诊断疾病。此外，目标检测技术在军事和国防领域中的应用也是至关重要的，如导弹定位和战机追踪，这些技术的进步直接关系到国家安全和战略地位。因此，目标检测技术不仅是连接理论研究与实际应用的桥梁，也是推动社会进步、保障人类福祉的关键技术。随着技术的不断发展和完善，目标检测将在更多领域展现出不可替代的价值。

目前，目标检测算法可分为传统的目标检测算法和基于深度学习的目标检测算法。传统的目标检测主要依赖于人工设计的特征来识别和定位图像中的对象，检测效率不高且准确度有限[2]。而基于深度学习的目标检测算法则是利用卷积神经网络（Convolution Neutral Network，CNN）对图像特征进行端到端的学习，能够自动提取到更具有区分性的特征，并在训练过程中逐渐提高检测性能[3]。因此，基于深度学习的目标检测算法在准确性和效率上往往更具优势，已成为目标检测领域的主流方法。

为了实现基于 CNN 的目标检测算法，许多嵌入式系统及相关软件工具相继出现。其中，CPU（Central Processing Unit，中央处理器）通过使用指令集执行运算，适合运行软件程序，但在处理大规模乘累加运算方面表现不佳，并且内部缺少专门用于计算 CNN 的硬件模块[4]。GPU（Graphic Processing Unit，图像处理单元）拥有并行处理和快速进行浮点计算的能力，成为 CNN 加速的常见平台[5]。但过高的能耗和较大的芯片面积，限制了在嵌入式系统中的应用。ASIC（Application Specific Circuit Integrated，专用集成电路）拥有高运算速度和低功耗的优点，体积也相对较小，但定制成本昂贵且开发灵活性不足。FPGA（Field Programmable Gate

Array，现场可编程门阵列）是一款具备高灵活性、低成本、高集成度和低功耗的高速并行通用器件，其通过并行数据处理方式支持多种计算任务，因此非常适合用来加速 CNN 计算[6]。ARM（Advanced RISC Machine，高级精简指令集计算机）采用了精简而高效的指令集，通过优化指令设计和流水线执行等技术实现了出色的性能表现，具有高能效、低功耗和灵活性强等特点。FPGA+ARM 的硬件架构结合了 FPGA 的灵活硬件可编程性与 ARM 处理器的高效序列处理能力，提供互补性能并增强功能集成，可以解决各种硬件平台的不足。

因此，基于 FPGA+ARM 架构的深度学习目标检测系统具有重要的研究意义和应用价值。

1.2 国内外研究现状

1.2.1 基于深度学习的目标检测算法研究现状

目前，基于深度学习的目标检测算法分为以 R-CNN（Region-based Convolution Neural Networks，基于区域的卷积神经网络）系列为代表的两阶段和以 YOLO（You Only Look Once，你只看一次）系列为代表的单阶段目标检测两类[7]。两阶段目标检测首先通过区域提议网络生成可能包含目标的候选区域，然后利用分类器和回归器对这些候选区域进行分类和定位，最终得到目标检测结果。单阶段目标检测则通过直接从输入图像中预测目标的类别和边界框来完成目标检测任务。图 1-1 为基于深度学习的目标检测算法发展历程图。

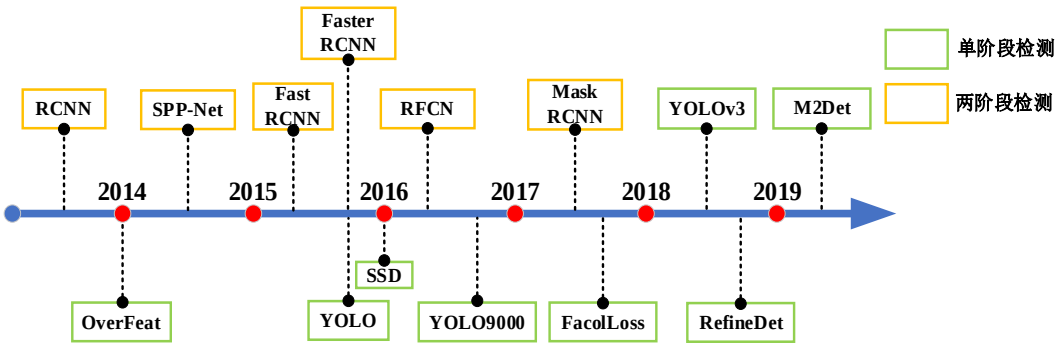


图 1-1 基于深度学习的目标检测算法发展历程图

2014 年，Girshick 等人提出了 R-CNN 模型[8]，该模型将目标检测任务分解为候选区域生成和特征提取、分类与回归两个阶段，使得 PASCAL VOC 数据集的检测率从 35.1%提升到了 53.7%，标志着目标检测领域迎来了深度学习的革命。He 等人于 2015 年提出 SPP-Net（Spatial Pyramid Pooling Networks，空间金字塔池化网络），通过引入空间金字塔池化层解决了 R-CNN 中固定尺寸的输入图像限制的

问题，使得网络能够处理任意尺寸的输入图像[9]。2015 年，Girshick 等人在 R-CNN 的基础上又提出了 Fast-R-CNN[10]，通过共享特征提取器和区域池化层实现端到端的训练，避免了 R-CNN 中的重复卷积操作，同时利用多任务损失函数优化分类和边界框回归。同年，Ren 等人提出 Faster-R-CNN，通过引入 RPN（Region Proposal Network，区域提议网络）完全实现端到端的训练并达到了更快的推理速度[11]。R-CNN 系列的目标检测算法具有较高的检测精度，但在实时性方面仍存在挑战。

2016 年，Redmon 团队提出了 YOLO 算法[12]，该算法将目标检测任务转化为回归问题，通过单个神经网络同时预测图像中的目标类别和位置边界框。为了解决 YOLO 算法在小尺寸目标检测方面的不足，Redmon 等人于 2017 年再次提出了 YOLOv2[13]，在 YOLO 算法的基础上引入多尺度预测机制和锚框技术，通过批标准化技术稳定网络训练，同时采用 Darknet-19 来深化网络结构，进一步提高了检测的精度和准确性。2018 年，Redmon 等研究员又设计出了 YOLOv3 和 YOLOv3-tiny 网络[14]。YOLOv3 引入了多尺度预测、改进的锚框技术以及特征级联等新方法，采用了更深的 Darknet-53 网络作为特征提取器，增强了特征学习能力，提高了检测性能。而 YOLOv3-tiny 则是 YOLOv3 的轻量级版本，通过简化网络结构和模块设计，实现了更小的模型体积和更低的计算复杂度，适用于资源受限的设备和实时应用场景。

因此，本文在权衡检测精度和速度后，选择了 YOLOv3-tiny 作为基础检测算法，并针对口罩和头盔佩戴场景对该算法进行改进和优化，以满足高精度和实时性的要求。

1.2.2 卷积神经网络加速器研究现状

从架构上看，卷积神经网络加速器主要包括单计算引擎和流式架构两种[15]。

流式架构中通常包含多个处理单元或流水线，用于同时处理多个数据流或任务。复旦大学的 Li 等人[16]设计出了一种全流水线架构，将每一层网络都映射到单独的硬件模块，并为每个模块采取针对性的并行策略，从而实现 565.94GOPS 和 391FPS 的峰值性能。Zhang 等人提出了一种基于列的细粒度流水线架构 DNNBuilder，将延迟和 BRAM 利用率分别降低了 7.7 倍和 43 倍[17]。韩国的研究者利用流水线展开、K 均值聚类的权重复用以及乘累加压缩三种优化技术实现了一个流式架构加速器[18]，并在 MNIST 和 Cifar-10 数据集上分别实现了 1531K 和 205K 的帧率。Nguyen 等人通过量化网络和优化数据路径来消除中间数据的片外

访问，提出了一种用于实时目标检测的高性能硬件流式架构，在 200MHz 下实现了 1.877 兆次/秒的吞吐量[19]。综上，流式架构可根据不同层的计算特性优化每个硬件内核的实现和并行方案，以提高性能。但需要针对不同的网络模型设计新的硬件计算单元，灵活性较低。且随着卷积层数的增加，会消耗大量的硬件资源和功耗。

单计算引擎架构是设计一个通用的计算单元依次处理每一层网络，同时利用共享的计算资源对不同层进行加速。Zhang 等人[20]利用循环平铺和转换等优化技术，对 CNN 加速器的计算吞吐量和所需内存带宽进行了定量分析，并利用 Roofline 模型搜索了硬件参数设计空间中的最优方案。最后，通过此方案设计出一款基于脉动阵列的通用卷积加速器，该加速器在 100MHz 工作频率下达到了 61.62GOPs 的峰值性能。Chen 等人提出了一种基于行固定（Row Stationary，RS）的数据流空间架构[21]，通过最大限度地利用处理单元的本地缓存、各单元间的直接通信以及空间并行性来减少各种类型的数据移动，从而显著减少了功耗。清华大学的 Qiu 等人采用卷积层内并行的方式设计 CNN 加速器，通过分时复用的方法提高了资源利用率，在 150MHz 工作频率下，该加速器的平均性能为 137.0GOPS[22]。Guo 等人提出了一种可编程、灵活的 CNN 加速器架构 Angel-Eye[23]，通过编译器将不同的 CNN 模型映射到指令序列，从而加速各种模型，且无需重新配置。综上，单计算引擎可以有效节省硬件资源，并具有通用性，但这种逐层计算的方式会产生较大的延时，且硬件利用率也不高。

由于应用场景为移动嵌入式设备，故本文采用单计算引擎架构来实现目标检测系统，以在性能和成本之间取得平衡；同时对架构进行优化，从而降低延迟，提高性能。

1.3 研究内容

本文的主要内容是在优化后的加速器架构的基础上，设计了一种基于 ZYNQ 平台的目标检测系统。以 YOLOv3-tiny 为基础算法，对其改进和优化并成功部署到 FPGA+ARM 的硬件平台上。本系统主要由多核 Contex-A53、XCZU3EG FPGA、OV5640 摄像头数据输入、HDMI（High-Definition Multimedia Interface，高清多媒体接口）视频输出显示、AXI（Advanced eXtensible Interface，高级可扩展接口）数据传输、图像缩放、VDMA（Video Direct Memory Access，视频直接存储访问）缓存控制以及卷积加速器等模块组成。其中，FPGA 主要负责视频图像的采集、

图像预处理、算法模型中各层的加速处理，并将处理后的结果通过 HDMI 接口传输给显示器。ARM 主要完成对外接驱动和卷积加速器的调用、片上 Linux 系统的搭建以及控制数据的发送和接收等。本文的具体研究内容如图 1-2 所示。

（1）YOLOv3-tiny 在目标检测中会存在漏检、重复检测和误检的问题，这些问题可能源自于其卷积层较少，导致提取的特征较为有限。这些特征包括局部特征和全局特征，但未经融合，使得特征表达能力有所不足。因此，为提高检测精度，在特征提取网络中添加了 3 个 3×3 的卷积层和 1×1 的卷积层，实现对模型结构的改进。除此之外，采用 K-means 聚类算法生成先验框能够有效提升网络的收敛速度和目标检测的准确性。为减少将改进的 YOLOv3-tiny 部署到 FPGA 上的逻辑资源，将其从 32 位浮点模型量化为 8 位定点模型。

（2）针对传统行缓存结构中存在无效数据传输时间的问题，提出了一种带有列缓存功能的行缓存结构。同时，为了提高卷积加速器的通用性和灵活性使其能够处理不同大小的特征图，提出了一种深度可配置的行缓存结构。然后构建出优化后的加速器模块，并对其进行仿真测试，验证了功能的正确性。

（3）设计并实现目标检测系统中的其他模块，并对部分模块进行仿真测试，最终在 ZYNQ 平台上搭建目标检测系统并进行了实际的板载验证，分析实验结果，验证加速器用于检测口罩和头盔的可行性。

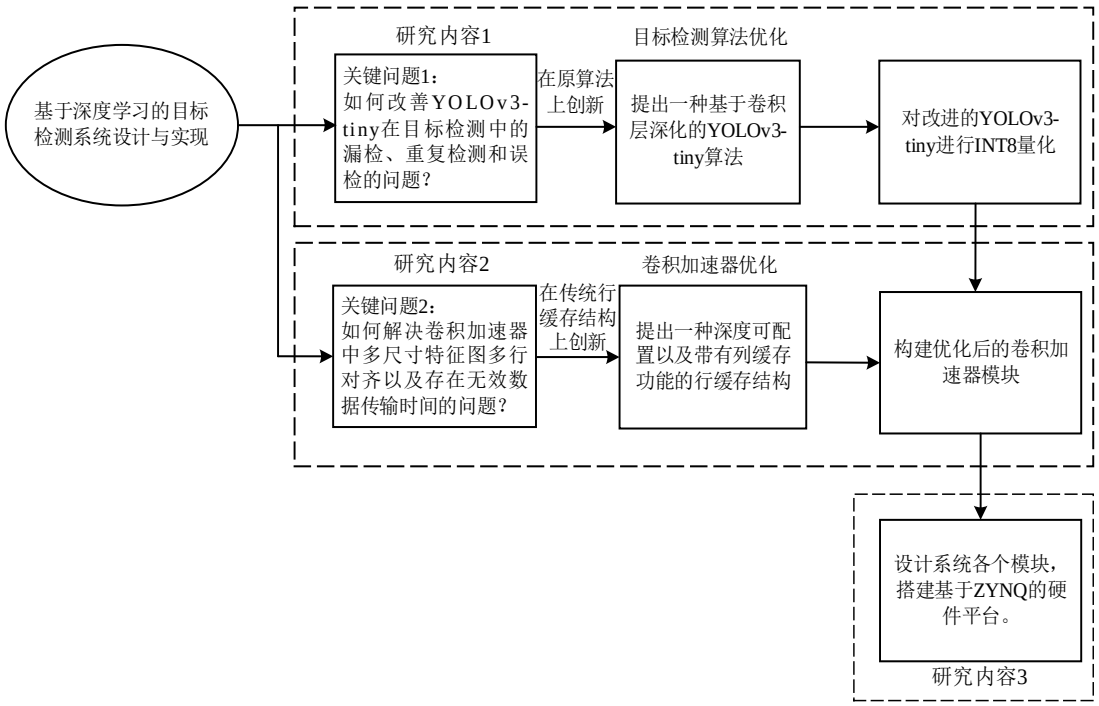


图 1-2 研究内容框架

1.4 论文结构安排

基于以上对课题的背景知识和研究内容的详细介绍，本文共分为五个章节，各章节内容如下：

第一章是绪论。首先叙述了目标检测技术的研究背景及意义，比较了几种常用平台在 CNN 加速上的性能优劣。其次介绍了基于深度学习的目标检测算法和卷积神经网络加速器的国内外研究现状。最后阐述了论文的研究内容和结构安排。

第二章是目标检测算法理论分析和硬件基础。主要分成三部分：首先对基于 YOLOv3-tiny 的目标检测算法的计算原理进行分析。其次对 CNN 加速器的相关理论进行阐述。最后对 YOLOv3-tiny 原模型存在的问题进行结构上的改进，并介绍了使用的动态定点量化方法和批量归一化层合并技术。

第三章是通用卷积神经网络加速器设计。首先介绍了卷积加速器的整体硬件架构，并对计算单元中卷积、池化、量化和激活函数模块以及缓存单元等硬件模块的设计进行详细说明。控制单元通过 PS（Processing System，处理系统）端的配置参数来实现。接着分别介绍了行缓存结构，乒乓技术和全流水思想以及并行度设计的优化方案。最后对本文设计的加速器进行功能性测试，并与其他加速器进行资源利用率的对比。

第四章是目标检测系统实现与验证。首先介绍了系统的整体架构和硬件开发平台，然后分别叙述了 PL（Programmable Logic，可编程逻辑）部分和 PS 部分的实现以及片上 Linux 系统的搭建。最后对系统进行调试，并将本文的设计与其他设计对比，进行性能评估。

第五章是全文的总结与展望。对本文的工作内容进行总结，分析其中的不足，并对后续改进内容进行展望。

第 2 章 目标检测算法理论分析和硬件基础

本章首先从卷积神经网络的角度分析和研究了 YOLOv3-tiny 算法的计算原理。然后介绍了关于卷积神经网络加速器的相关概念，分析了卷积加速器的基本架构以及卷积和池化计算的并行性，为后面通用卷积加速器的设计奠定了理论基础。接着针对 YOLOv3-tiny 存在的漏检、误检、小物体检测精度低等问题，对 YOLO 算法进行了改进。最后，通过浮点数定点化和批量归一化层融合技术优化改进后的网络，使其更适合部署到 FPGA 上。

2.1 YOLOv3-tiny 算法原理分析

2.1.1 卷积层

卷积层通过执行卷积操作，滑动卷积核于输入图像之上以计算局部区域的加权总和，从而实现输入数据特征的提取，然后生成输出特征图。卷积的主要目的是通过学习权重和特征映射来捕获输入数据的空间结构信息，实现对图像、语音或其他数据的抽象表达[24]。其数学表达式如下：

$$Y(i,j)=\sum_{m=0}^{M-1}\sum_{n}^{N-1}X(i+m,j+n)\cdot W(m,n)+b \tag{2-1}$$

其中，X 是输入数据，Y 是输出数据，W 是卷积核权重，用于学习特征的表示，b 为偏置项，用于调整每个位置的输出，i 和 j 分别表示输出特征图中的位置，M 和 N 是卷积核的尺寸。

2.1.2 激活层

激活层通过在神经网络中加入非线性变换，使得网络能够在每个神经元的输出上应用激活函数，从而使网络有能力获取非线性关系并表达更为复杂的数据模式[25]。常见的激活函数有：

（1）Sigmoid 函数

Sigmoid 函数[26]将输入映射到[0,1]区间，其公式如下：

$$Sigmoid(x)=\frac{1}{1+e^{-x}} \tag{2-2}$$

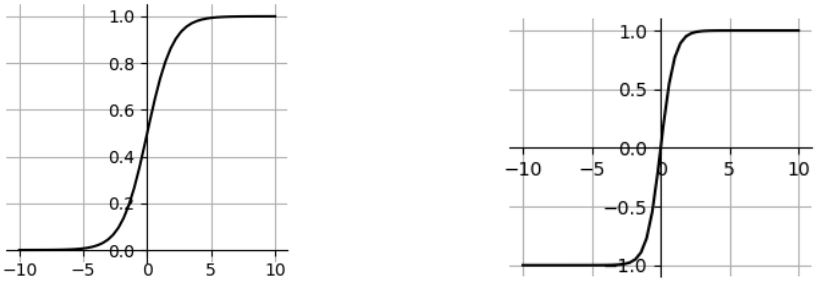
Sigmoid 函数在处理二分类任务的输出层中得到了广泛使用，因为其输出介于 0 和 1 之间，适合表达为概率。不过，当输入值过大或过小时，Sigmoid 函数可能会造成梯度消失的现象，这会使训练过程变得困难。该函数图像如图 2-1（a）

所示。

（2）Tanh 函数

图 2-1（b）展示了 Tanh 函数，该函数把输入值转换到[-1,1]的范围内，并且在零点处是对称的[26]。与 Sigmoid 函数相比，Tanh 函数提供了更广的输出区间。然而，Tanh 函数也面临着梯度消失的挑战。其表达式为

$$\mathrm{Tanh}(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{2-3}$$



（a）Sigmoid函数　　（b）Tanh函数

图 2-1 Sigmoid 和 Tanh 函数

（3）ReLU 函数

ReLU 函数对于正数输入直接返回输入值，而对于负数输入则返回零[27]。其表达式如式（2-4）所示。

$$\mathrm{Re}\,LU(x)=\begin{cases}x, & x\geq 0\\ 0, & x<0\end{cases} \tag{2-4}$$

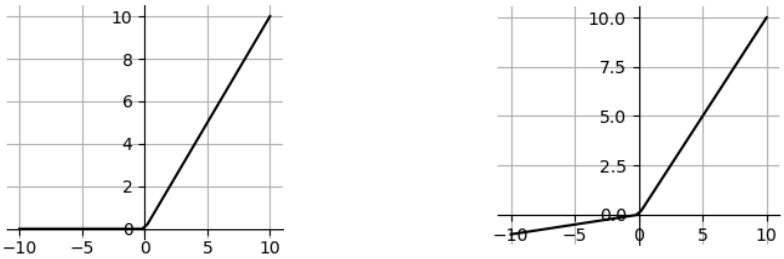
ReLU 函数的主要优势在于其计算的高效性，有助于提高模型训练的速率。不过，ReLU 可能会引发神经元“死亡”的现象，即训练时部分神经元可能不再对任何数据激活。该函数图像如图 2-2（a）所示。

（4）Leaky ReLU 函数

Leaky ReLU 函数[28]对 ReLU 函数进行了优化，通过赋予负数输入一个微小的正梯度，增强了模型的处理能力。数学上可以表示为

$$Leaky(x)=\begin{cases}x, & x\geq 0\\ \lambda x, & x<0\end{cases}\quad(0<\lambda<1) \tag{2-5}$$

Leaky ReLU 函数有效缓解了 ReLU 函数中可能导致的神经元失活问题，从而提高了网络训练阶段的稳定性。如图 2-2（b）所示为$\lambda=0.1$时的函数图像。



（a）ReLU函数　　（b）Leaky ReLU函数

图 2-2 ReLU 和 Leaky ReLU 函数

2.1.3 池化层

池化层的主要功能是通过减少输入特征的空间尺寸来降低模型的参数量和运算成本，同时维护重要的特征信息[29]。池化操作主要有最大池化和平均池化，分别用于提取区域内的最大值和平均值。

最大池化是在每个池化窗口中选择最大值作为输出。常见的最大池化操作是在 2×2 的池化窗口中选择四个值中的最大值。这有助于保留图像中最显著的特征，同时减小数据的空间维度。

平均池化通过计算池化窗口中所有元素的均值来生成输出。与最大池化相比，平均池化有助于降低噪音，对输入数据进行平滑处理，提高网络的稳定性。

两者的数学表达式分别如下：

$$Y(i,j)=max(X(2i,2j),X(2i,2j+1),X(2i+1,2j),\ X(2i+1,2j+1)) \quad (2\text{-}6)$$

$$Y(i,j)=\frac{1}{4}(X(2i,2j)+X(2i,2j+1)+X(2i+1,2j)+X(2i+1,2j+1)) \quad (2\text{-}7)$$

其中，$Y(i,j)$ 表示池化后的输出值，$X(2i,2j)$，$X(2i,2j+1)$，$X(2i+1,2j)$ 和 $X(2i+1,2j+1)$ 表示输入数据的相应位置。

2.1.4 批量归一化层

在 YOLOv3-tiny 中，批量归一化通常用于卷积层中，对每个卷积层的输出进行归一化操作，提高网络的训练效果和收敛速度[30]。假设卷积层的输出特征图为 X_{ijk}，其中包含多个通道，对于每个通道，批量归一化公式如下：

$$\hat{X}_{ijk}=\frac{X_{ijk}-\mu_i}{\sqrt{\sigma_i^2+\varepsilon}} \quad (2\text{-}8)$$

$$\mu_i=\frac{1}{H\times W}\sum_{j=1}^{H}\sum_{k=1}^{W}X_{ijk} \quad (2\text{-}9)$$

$$\sigma_i^2=\frac{1}{H\times W}\sum_{j=1}^{H}\sum_{k=1}^{W}(X_{ijk}-\mu_i)^2 \quad (2\text{-}10)$$

$$Y_{ijk} = \gamma_i \hat{X}_{ijk} + \beta_i \tag{2-11}$$

其中，$\hat{X}_{ijk}$ 表示归一化处理后的特征图，i 表示通道，j 和 k 表示特征图的空间位置。μ_i 是对每个通道的特征图计算的均值，表示该通道的平均值。σ_i^2 是对每个通道的特征图计算的方差，表示该通道数据分布的离散程度。ε 是用于避免除零错误的常数。H 和 W 分别表示特征图的高度和宽度。γ_i 和 β_i 分别用于缩放和平移归一化后的特征。

2.1.5 检测层

YOLOv3-tiny 的检测层位于神经网络的最后几层，负责生成目标检测的结果。该层输出一个三维张量，包含网络对输入图像中目标的位置和类别的预测。通过预定义的锚框，检测层预测每个锚框的边界框参数，包括中心偏移、宽高缩放因子和置信度分数。此外，对每个格子进行类别预测，表示目标属于各个类别的概率。输出经过后处理后，转换为在输入图像上的实际边界框和目标类别的概率。最后，为提高准确性，采用非极大值抑制算法去消除重叠的边界框[31]。

2.2 加速器相关概念介绍

2.2.1 典型的卷积加速器结构

卷积加速器是一种专门为卷积神经网络推理而设计的硬件加速器，其典型结构如图 2-3 所示，主要包括 DMA（Direct Memory Access，直接储存访问）、计算单元、外部存储器以及片上缓存等[32]。首先，输入特征图从外部存储器加载到输入特征图缓存中。与此同时，卷积核的权重参数也被加载到权重缓存中。然后将输入特征图和权重送到计算单元进行计算，计算结果被保存在输出特征图缓存中，并在需要时传输到外部存储器中。其中，加速器的延时主要包括计算延时、数据传输延时以及存储访问延时[33]。减少延时的方法有流水线设计、循环展开和循环分块等[34]。流水线设计是将计算划分成多个阶段，使得在一个时钟周期内可以同时执行不同阶段的计算，从而提高整体吞吐量。循环展开是将循环中的多个迭代合并成一个大的迭代，从而减少循环开销，提高计算效率。通过循环展开，将多个乘法和累加操作组合成一个大的计算块，以提高并行性。循环分块利用缓存的数据重用性以提高计算效率。在卷积神经网络的卷积操作中，循环分块将输入数据划分成较小的块，通过对这些块进行计算，有利于降低内存访问的延迟，增强数据的局部性。

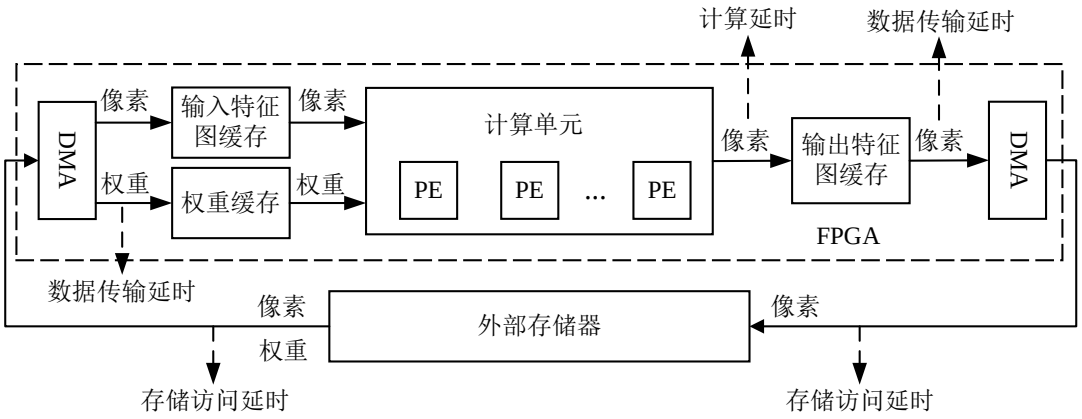


图 2-3 典型的卷积加速器结构

2.2.2 卷积计算并行度分析

卷积计算并行性分析在于识别并实施有效的并行处理策略，以最大化计算资源的利用率，减少执行时间，并提高计算效率。在卷积神经网络的前向传播中，通过卷积窗口沿输入数据滑动，执行卷积运算，从而体现了其计算过程的高度并行性。这种并行性能够在同一时间内同时处理多个卷积窗口的计算，充分发挥硬件资源，从而提高整体计算性能，快速完成训练和推理任务。

如图 2-4 所示，卷积操作涉及四个循环层次：Loop-1 用于展开卷积核，Loop-2 展开输入特征图的通道，Loop-3 处理输入特征图内部的展开，而 Loop-4 负责展开输出特征图的通道[35]。下面依次介绍这四种循环展开方式。

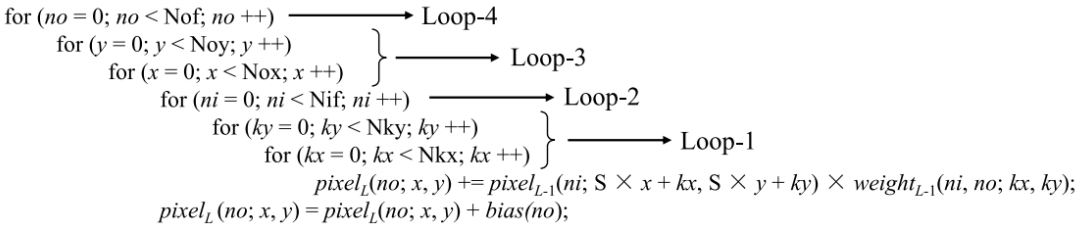


图 2-4 卷积计算的四重循环

（1）卷积核展开

如图 2-5 所示，卷积核展开是最常见和直观的并行计算方案，其中 Pkx 和 Pky 分别表示其展开的并行参数。在卷积核展开中，每个周期都会计算 $Pkx \times Pky$ 个输入特征图像素和 $Pkx \times Pky$ 个卷积核权重的内积。这种展开操作的目的是通过并行计算多个位置的内积来提高计算效率。为了实现这个目标，需要一个具有 $Pkx \times Pky$ 个并行分支的加法树，用于对 $Pkx \times Pky$ 个乘法结果进行累加。这个累加的输出被送入累加器中，与之前的部分和相加，从而得到最终的累加结果。

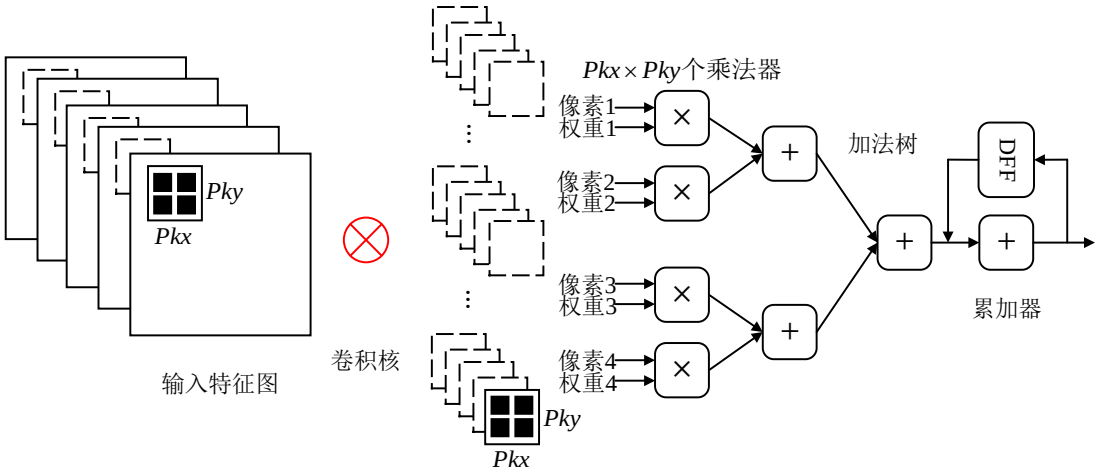


图 2-5 卷积核展开及其计算架构

（2）输入特征图通道展开

如图 2-6 所示，输入特征图通道展开是利用不同输入特征图通道之间的并行性，其中 *Pif* 为并行参数。在输入特征图的展开中，为了计算内积，每个计算周期需要来自 *Pif* 个不同特征及其对应卷积核的像素和权重。Loop-2 与 Loop-1 展开具有相同的计算结构，但加法树的分支数不同，分支数为 *Pif* 。

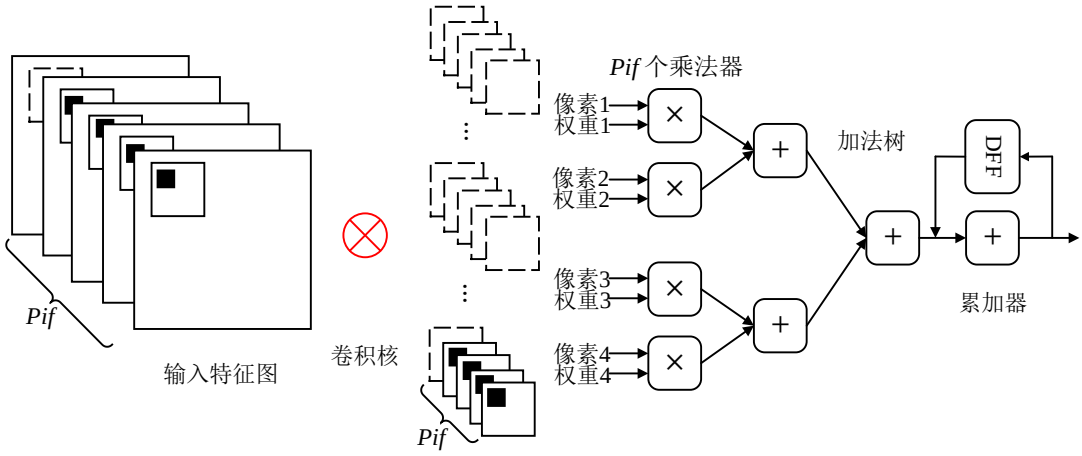


图 2-6 输入特征图通道展开及其计算架构

（3）输入特征图内展开

如图 2-7 所示，输入特征图内展开是利用同一输入特征图内不同像素的并行性，其中 *Pix* 和 *Piy* 为其展开的并行参数。对于输入特征图内展开，每个周期在同一特征图不同位置选取 *Pix*×*Piy* 个像素与同一个权重相乘，从而实现对该权重 *Pix*×*Piy* 次复用。这种并行的 *Pix*×*Piy* 次乘法操作使得 *Pix*×*Piy* 个独立输出生成，所以要用 *Pix*×*Piy* 个累加器分别对这些输出进行累加，不需要加法树。

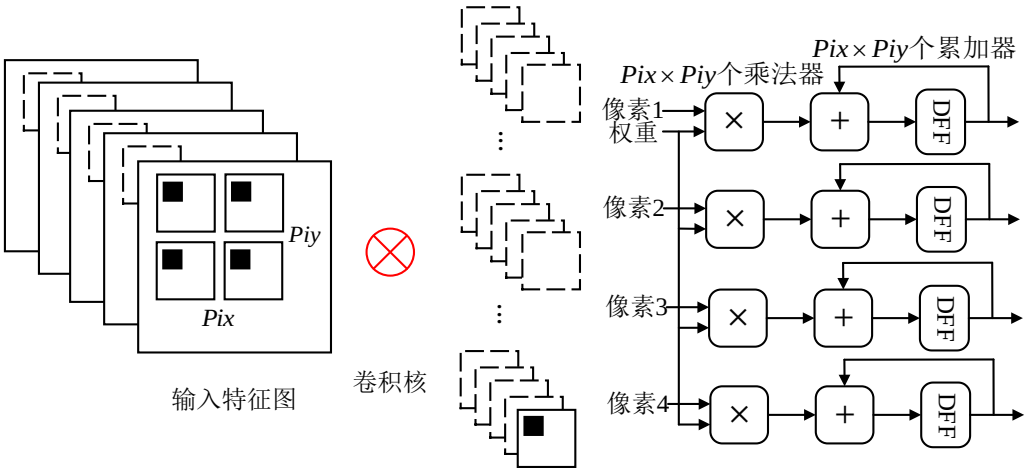


图 2-7 输入特征图内展开及其计算架构

（4）输出特征图通道展开

如图 2-8 所示，输出特征图通道展开是利用不同输出特征图通道间卷积核权重的并行性，其中 *Pof* 为并行参数。在输出特征图展开中，每个周期会计算一个像素与 *Pof* 个相同位置权重的乘积，且像素会被重用 *Pof* 次。计算结构与使用 *Pof* 个乘法器和累加器的 Loop-3 展开相同。

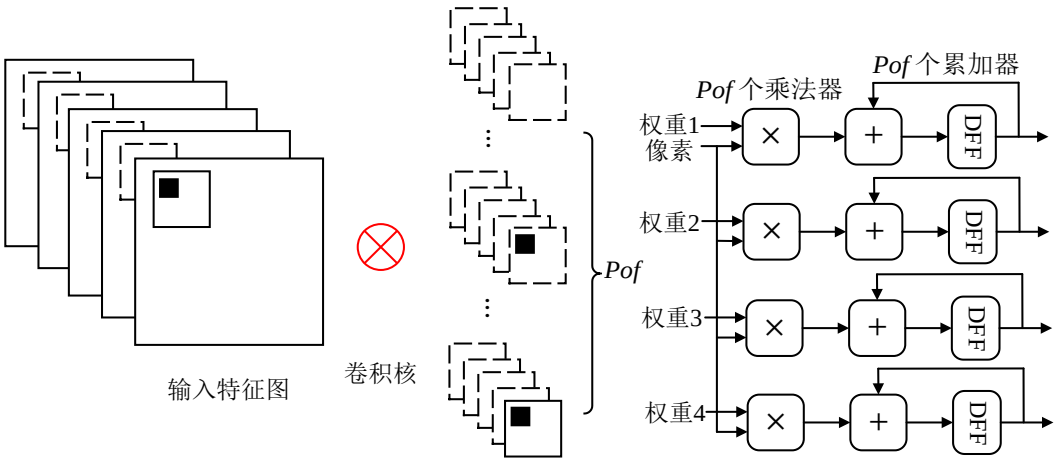


图 2-8 输出特征图通道展开及其计算架构

2.2.3 池化计算并行度分析

池化计算并行度分析在于评估并优化池化操作中的并行处理能力。池化操作在每个特征图通道中独立进行，所以其本身就具备高度的并行性。为了进一步提高并行性，可以将输入特征图分割成多个单独块，并同时进行操作。在进行最大池化时，需要对每个单独块中的元素比较并得到最大值。因此，在每个时钟周期内，需要比较单独块中的全部元素，以确定单独块中的最大值。通过这种方式可以加速池化操作，实现高效的并行计算。其原理如图 2-9 所示。

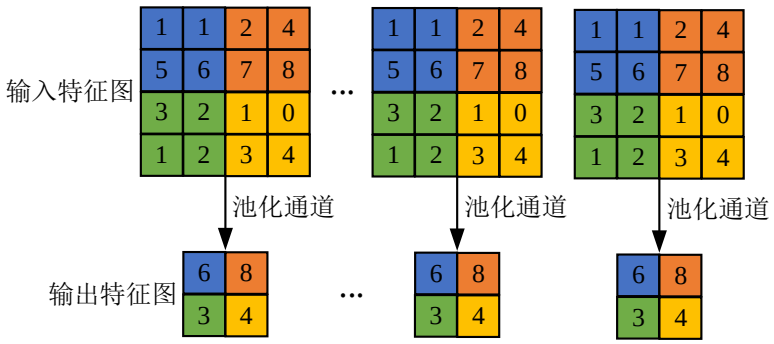


图 2-9 池化并行性计算原理图

2.3 YOLOv3-tiny 算法的改进和优化

2.3.1 YOLOv3-tiny 网络的改进

针对 YOLOv3-tiny 存在的漏检、误检、小物体检测精度低等问题，本文对 YOLOv3-tiny 进行了如下改进。首先为了让网络更容易学习，使用 K-means 聚类算法[36]在训练数据集的边界框上重新计算锚框或先验框。然后对原有的 YOLOv3-tiny 的网络结构进行深化，使其能够更好地提取图像的特征，从而提高检测精度。本文所检测的目标对象为口罩和头盔。

（1）锚框的改进

在原始的 YOLOv3-tiny 中，有 6 个具有不同大小的锚框，这些锚框均匀分布在两个不同检测尺度的输出特征图上。选择合适的锚框可以提高预测准确性，并有助于网络训练的收敛速度。考虑到不同对象具有不同的宽高比，相应的锚框也有不同的大小。因此，本文采用 K-means 聚类算法对口罩和头盔进行处理，以获得适当大小的锚框。

标准的 K-means 聚类算法是基于欧氏距离的，并不适合 YOLOv3-tiny 选择锚框，因为 YOLOv3-tiny 希望得到能产生良好 IOU（Intersection over Union，交并比）分数的锚框，而 IOU 分数是与锚框大小无关的[37]。但是，相比于较小的锚框，基于欧氏距离的 K-means 聚类算法则会对较大的锚框产生更多的误差。因此，本文使用边界框和锚框的 IOU 作为衡量它们相似性的标准，将 K-means 的距离重新定义为：

$$d(box,centroid)=1-IOU(box,centroid) \tag{2-12}$$

其中，box 和 $centroid$ 分别表示样本框和聚类中心。由公式（2-12）可知距离与 IOU 呈反比关系，这意味着当距离减小时，IOU 将增加。边界框与锚框的交并比越高，边界的预测准确性也越高。因此，优化的目标是减小距离 $d(box,centroid)$

的值。基于此，本文中 K-means 算法的过程如下：

1. 设定 K 值作为锚框的数量，每个锚框具有初始高度 H_i^0 和宽度 W_i^0 。

2. 计算每个边界框和锚框的 IOU，并使用距离 $d(box,centroid)$ 为每个边界框匹配相应的锚框。使用边界框的高度 h_i 和宽度 w_i 来更新相应的锚框的高度 H_i 和宽度 W_i ，公式如下所示。

$$W_i = \frac{1}{N}\sum w_i \tag{2-13}$$

$$H_i = \frac{1}{N}\sum h_i \tag{2-14}$$

3. 通过重复上述过程，距离会减小，这意味着更新后的锚框的高度和宽度更适用于检测目标。

除了锚框的大小，锚框的数量也影响检测准确性。因此，需要选择适合口罩和头盔的锚框数量。本文使用 K-means 算法对具有不同数量的锚框进行聚类，并在不同数量的锚框下获得 mIOU（Mean Intersection over Union，均值交并比），如图 2-10 所示。

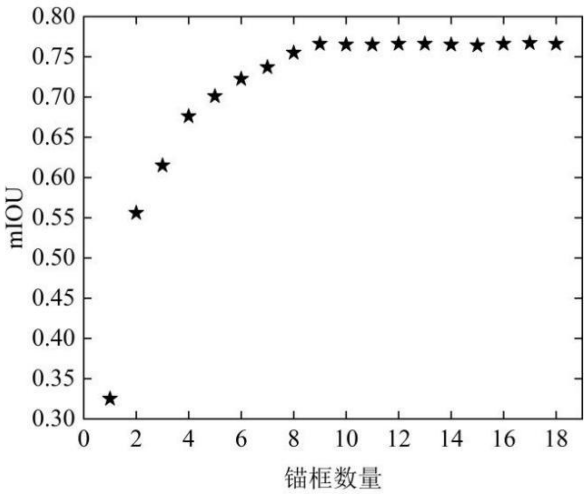


图 2-10 锚框数量与 mIOU 的关系

从图中可以看到随着锚框数量的增加，mIOU 也会增加。然而，当锚框数量大于 9 时，IOU 的增加变得较小并且几乎稳定下来。

此外，通过实验证明了锚框数量的增加对 YOLOv3-tiny 检测性能的影响，结果如表 2-1 所示。从表中可以看到锚框数量的增加对预测分类的准确性和实时性能有很小的影响。因此，使用 9 个锚框来检测口罩和头盔以在计算复杂度和召回率之间取得更好的平衡。

表 2-1 锚框数量对性能的影响

模型	锚框数量	mAP（%）	FPS
YOLOv3-tiny	7	73.45	163.7
	9	73.56	162.8
	11	73.58	161.8

（2）YOLOv3-tiny 网络结构的改进

如图 2-11 所示，YOLOv3-tiny 由卷积层、最大池化层和路由层等组成。其运行速度比 YOLOv3 快得多，但性能方面明显下降，因此本文提出了一种改进的 YOLOv3-tiny 网络结构，在保持网络速度的同时提高网络性能。改进的 YOLOv3-tiny 网络结构如图 2-12 所示。首先，在特征提取网络中添加了 3 个 3×3 的卷积层，以增强特征提取能力和非线性表达能力。增加的是 3×3 卷积核而不是 5×5 或 7×7 的卷积核，是因为使用两个或三个 3×3 卷积核可以达到与一个 5×5 或一个 7×7 卷积核相同的效果，但是计算它们的参数比例分别为 0.72 和 0.55，3×3 卷积核可以更有效压缩参数量。在深度学习网络中，网络层数越多，需要训练的参数也就越多，训练难度也就越大。因此，在增加 3×3 卷积层的同时，还会增加 1×1 卷积层，这样可以压缩数据量，使得网络的训练相对容易。

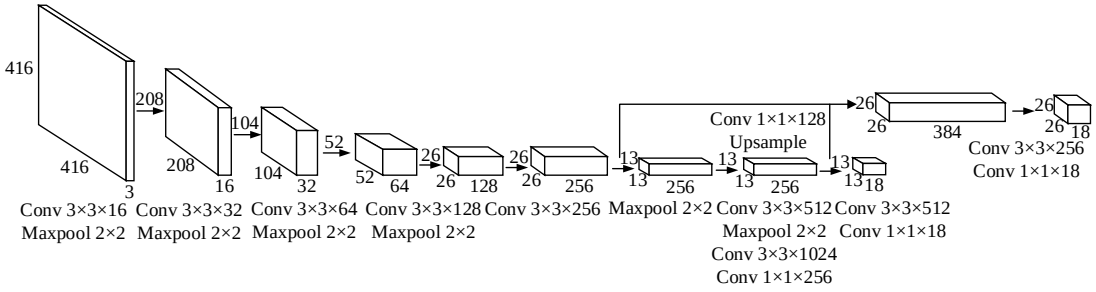


图 2-11 YOLOv3-tiny 网络结构

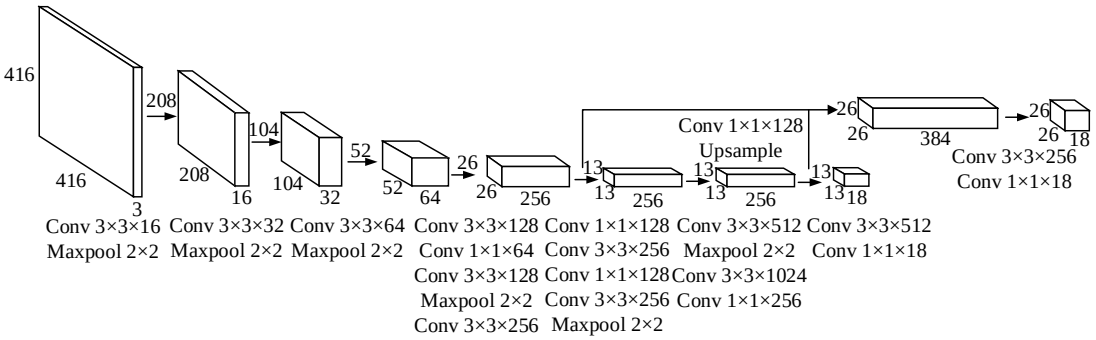
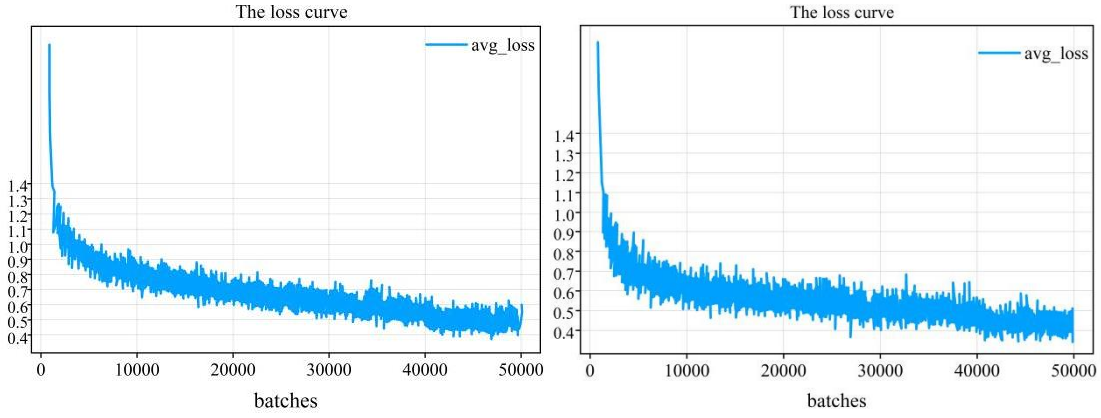


图 2-12 改进的 YOLOv3-tiny 网络结构

为了验证改进方法的有效性，本文在一台内存为 3GB 的 NVIDIA RTX 3050 GPU 的计算机上实验。如图 2-13 所示，选取网上提供的 1001 张头盔和 853 张口

罩图片作为数据集，其中 1483 张用于训练，371 张用于验证。网络总共迭代 50000 次，初始学习率为 0.001，当迭代次数达到 40000 和 45000 时，学习率除以 10，动量为 0.9，权衰减为 0.005。图 2-14 显示了训练期间的损失曲线，可以看到两条损失曲线有相似的下降趋势。然而，与 YOLOv3-tiny 相比，改进的 YOLOv3-tiny 的收敛速度更快。为了更全面地评估改进后的 YOLOv3-tiny，表 2-2 总结了平均精确度（Mean Average Precision，mAP）、精度、召回率、F1 值和检测速度等评价指标[38]。从表中可以看出，改进的方法在 mAP、精度、召回率和 F1 值上都优于 YOLOv3-tiny。在运行速度上，改进的方法比 YOLOv3-tiny 稍微慢一些，但仍能达到 117FPS 的帧率，可以满足实时性的要求。

图 2-13 头盔口罩数据集



（a）YOLOv3-tiny　　（b）改进的 YOLOv3-tiny

图 2-14 两种模型的损失曲线

表 2-2 两种模型的评价指标对比

模型	mAP	精确度	召回率	F1	检测速度
YOLOv3-tiny	72.87%	0.69	0.74	0.72	7.02ms
改进的 YOLOv3-tiny	77.56%	0.75	0.78	0.75	8.54ms

此外，图 2-15 中给出了两种方法的 PR（Precision-Recall，精度—召回率）曲线。对于 PR 曲线，曲线下方的面积越大，模型性能越好。显然，改进后的 YOLOv3-tiny 优于原始的 YOLOv3-tiny。图 2-16 展示了两种网络检测效果。从图中可看出，改进后的 YOLOv3-tiny 在保证运行速度的同时，提高了对口罩和头盔的检测性能。

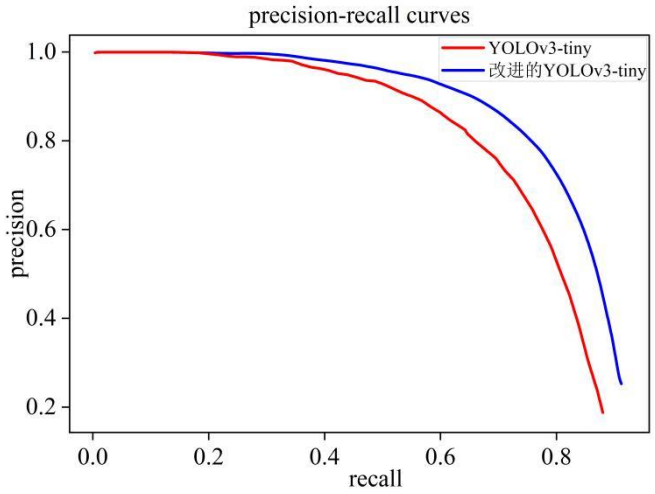


图 2-15 两种模型的 PR 曲线

（a）YOLOv3-tiny　　（b）改进的 YOLOv3-tiny

图 2-16 两种模型的检测效果

2.3.2 浮点数定点化

浮点运算会造成大量的资源消耗，不利于 FPGA 硬件加速。表 2-3 总结了不同精度的乘加运算所需的硬件资源消耗。从表中可以发现，当采用 16 位定点时，所需的资源消耗大幅度减少，所以在尽可能保证计算精度的前提下，使用定点计算在 FPGA 上进行实现更具有优势。

表 2-3 不同精度的资源消耗

	FP32	FP16	Fixed32	Fixed16	Fixed8	Fixed4
LUT	798	441	110	0	0	0
FF	1260	689	64	0	0	0
DSP	2	1	4	1	1	1

基于 INT8 的数据量化方法属于动态定点数线性非对称量化[39]，通过选择合适的量化参数，如量化比例和零点，来最小化量化误差。虽然 INT8 量化可能会引入一定的精度损失，但通过平衡模型的精度和性能，并确保训练和量化过程一致，可以实现满足应用需求的模型性能。如图 2-17 所示，本文将 Float32 模型量化为 INT8 模型，其中 Q 为量化后的整数，X 为原始浮点数，S 为缩放因子，Z 为量化后的零点。计算公式如式（2-15）和（2-16）所示。

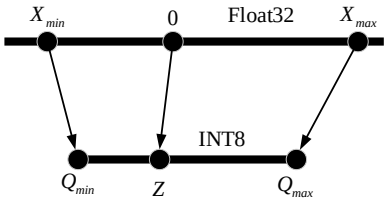


图 2-17 INT8 线性量化原理

$$S = \frac{X_{max} - X_{min}}{Q_{max} - Q_{min}} \tag{2-15}$$

$$Z = round(Q_{max} - \frac{X_{max}}{S}) \tag{2-16}$$

卷积计算的本质是乘累加运算，其表达式如下所示：

$$X_3^{i,k} = \sum_{j=1}^{N} X_1^{i,j} X_2^{j,k} \tag{2-17}$$

通过将上述表达式中的浮点数转换为量化后的整数，并适当调整得到：

$$S_3(Q_3^{j.k} - Z_3) = \sum_{j=1}^{N} S_1(Q_1^{i,j} - Z_1) S_2(Q_2^{j,k} - Z_2) \tag{2-18}$$

$$Q_3^{i.k} - Z_3 = \frac{S_1 S_2}{S_3} \sum_{j=1}^{N} (Q_1^{i,j} - Z_1)(Q_2^{j,k} - Z_2) \tag{2-19}$$

从上述表达式可知，除了 $M = \frac{S_1 S_2}{S_3}$ 是浮点数外，其他运算均为 INT8 类型。利用浮点数的存储特性，可以通过 frexp 函数将 M 拆分为尾数和指数部分：

$$M = 2^{-n} M_0 \tag{2-20}$$

因此，与 M_0 的乘法运算可被转化为定点运算，而与 2^{-n} 的乘法运算则可通过右移操作实现。通过这种方法，量化模型的推理过程完全转换为基于 INT8 的定点运算，大幅度降低了硬件资源的使用。量化训练结束后，改进的 YOLOv3-tiny 各层参数以二进制文件.bin 进行存储，这些文件将存入 SD 卡中，使得 ZYNQ 的 PS 端可以通过 SD 卡读取相应的参数文件，如图 2-18 所示。

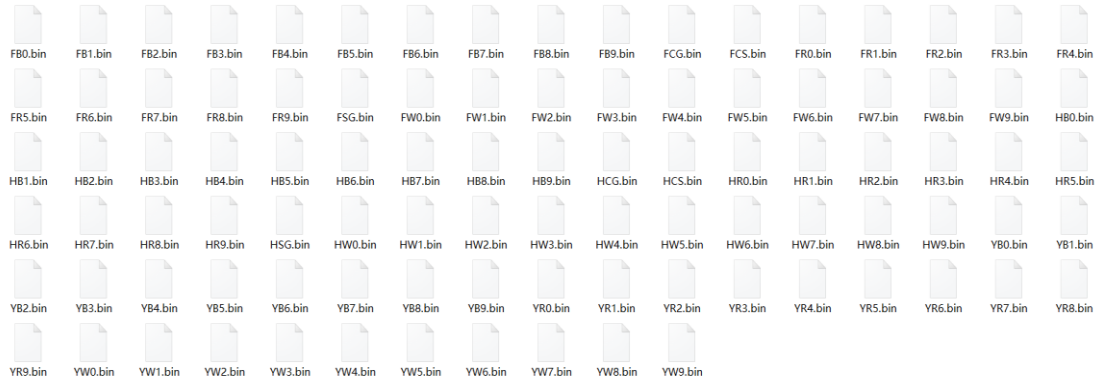


图 2-18 量化训练后的.bin 文件

2.3.3 批量归一化层融合

批量归一化层的乘除运算在 FPGA 上会占用大量资源并延长处理时间，故可将批量归一化层与卷积层融合[40]。此时，BN 层的操作如公式（2-21）所示：

$$Y_{ijk}=\frac{\gamma_i}{\sqrt{\sigma_i^2+\varepsilon}}\cdot X_i+(\beta_i-\frac{\gamma_i\cdot\mu_i}{\sqrt{\sigma_i^2+\varepsilon}}) \tag{2-21}$$

其中，X_i为卷积结果，σ_i^2为X_i方差的数学期望，μ_i为X_i均值的数学期望，β_i为偏置，γ_i为缩放因子，ε为防止分母为零的常数。训练过程中获得的参数σ_i^2，μ_i，γ_i和ε在推断时可视作常数，故$\frac{\gamma_i}{\sqrt{\sigma_i^2+\varepsilon}}$和$\frac{\gamma_i\cdot\mu_i}{\sqrt{\sigma_i^2+\varepsilon}}$可与权重和偏置合并。这样，在推断阶段批量归一化层全部融合到卷积层，使得计算量减少，推断过程加速。

2.4 本章小结

本章主要对 YOLOv3-tiny 算法和卷积加速器的相关概念进行阐述。首先，介绍了 YOLOv3-tiny 中的组成要素。其次，对典型的加速器结构进行了介绍，并分析了卷积计算和池化计算的并行性。最后，根据目标检测系统的实现要求，对 YOLOv3-tiny 网络进行了两个方面的改进：1.采用 K-means 聚类算法来重新计算锚框，使得学习更加容易；2.对 YOLOv3-tiny 网络结构进行了改进，在特征提取网络中添加了一系列的 1×1 和 3×3 卷积层，提高了检测精度。此外，还利用数据量化和层融合技术对网络进行优化，为后面目标检测系统的设计奠定了基础。

第 3 章 通用卷积神经网络加速器设计

硬件加速器是整个目标检测系统的核心。本章首先阐述了加速器的整体硬件架构；然后详细介绍了架构中的各模块，包括计算单元、缓存单元以及控制单元，其中计算单元包括卷积、池化、量化以及激活函数模块，缓存单元包括输入特征图、权重、输出以及外部缓存；接着提出了一些硬件加速优化策略；最后对加速器进行仿真测试以及资源评估。

3.1 卷积加速器总体架构

本文提出了一种基于 ARM+FPGA 的异构架构的卷积加速器用于加速 CNN。其架构如图 3-1 所示，分为 PL 和 PS 两部分。PL 端即 FPGA，包括计算单元（Processing Element，PE）、缓存单元、控制单元以及 DMA。PE 执行 CNN 中的主要计算任务。缓存单元分为输入缓存和输出缓存，为 PE 提供所需的数据并存储输出结果。控制单元接收来自 PS 端的指令，并将指令解码传递给各模块，除了 DMA 外的所有模块都在控制单元的管理下运行。DMA 负责传输 PS 端外部存储器 DDR 和 PL 端片上缓存之间的数据和指令。PS 端由通用处理器 ARM 和 DDR 组成。ARM 控制计算单元的启动和停止、数据的传输和处理以及中间结果的存储和调度。输入数据、神经网络模型参数和计算结果存储在 DDR 中[41]。

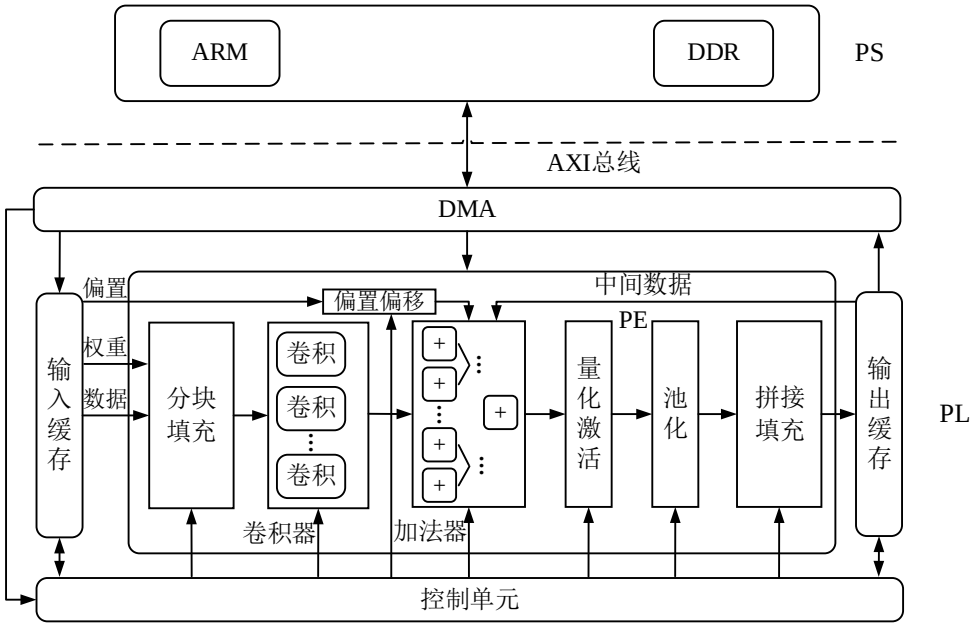


图 3-1 卷积神经网络加速器架构

原始图像数据在预处理后存储到 DDR 中。接着，DMA 通过 AXI Stream 总

线访问 DDR，获取特征图、权重和偏置，并将数据缓存在输入缓存中。计算单元处理这些数据，其中首层特征图数据需要进行分块和填充。中间结果由 PE 生成，并存储在输出缓存中。池化操作使得图像尺寸减小，因此需要通过拼接和填充使输出特征图尺寸与输入特征图匹配。之后，特征图经过填充处理后写入 DDR 中，从而完成一个 PE 的计算。重复这一过程，直到所有层的计算完成，最后将分类结果通过 DMA 传输回 ARM 处理器输出。

3.2 计算单元设计

图 3-2 展示了计算单元的硬件架构，PE 主要由卷积模块、池化模块、量化激活模块和偏置移位模块组成。卷积模块中的卷积器执行特征图数据与权重数据的乘加法运算，而加法树对卷积结果进行求和。池化模块通过行缓存对输入数据释放一个数据选择窗口，并利用比较器得到最大值。量化激活函数模块实现数据的 8bit 量化并应用非线性激活函数。偏置移位模块则是根据每层的量化结果对输入偏置进行移位。下面将介绍主要模块的硬件设计方案。

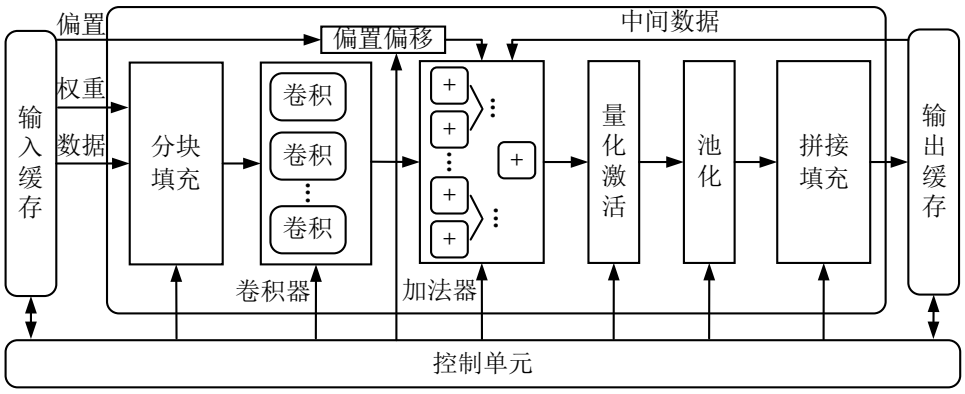


图 3-2 计算单元架构

3.2.1 卷积模块设计

为提高计算效率，卷积模块采用行缓存结构，如图 3-3 所示。其中，卷积核的尺寸为 3×3，L 为输入特征图的行长度。输入特征图数据以串行方式进入 $(2L+3)$ 个移位寄存器中，每个时钟周期，数据流向前一级寄存器，直到第一个数据输出。第一个数据加载到最后一个移位寄存器后，行缓存区在输入图像上释放一个卷积窗口。在选定的窗口中每 9 个周期得到需要卷积运算的 9 个数据，然后输出窗口数据，完成与 9 个权重的乘加法和偏置加法运算，得到第一个卷积结果。通过多次卷积窗口的数据读取和计算操作，遍历整个输入图像，最后将卷积层的计算结果经过之后的模块计算后存入到输出缓存对应的 RAM 中。

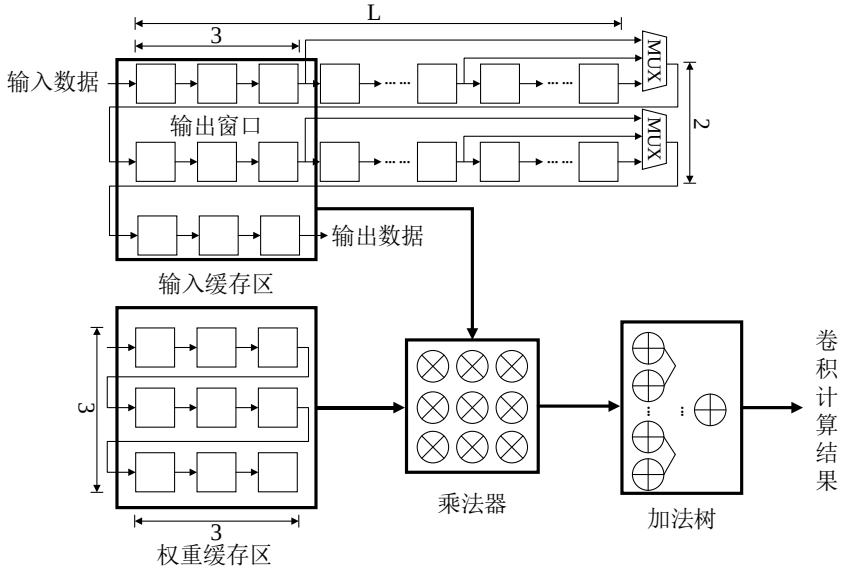


图 3-3 卷积计算硬件结构

在滑动卷积窗口时，由于部分数据会重叠，该结构实现了对数据的充分复用。在一张输入特征图进行卷积运算时，只要一次输入。一旦数据加载完成，每个时钟周期便能够执行完一个卷积窗口中所有的乘法操作。这样的设计理论上可以适应各种尺寸的卷积运算。

此外，本文的卷积模块在乘法器和加法树方面进行了资源优化。如图 3-4 所示，对于 A×B 与 B×C 而言，都有同一个因子 B。为了充分利用 DSP48 的输入位宽，将 A 和 C 这两个 INT8 类型乘数利用预加器填充至 DSP 的 27bit 输入端口两端，这两对乘法通过一个 DSP 运算在 48bit 位宽输出端口两端分别得到计算结果，从而实现将两对 INT8 乘法运算映射到一个 DSP 资源上。采用这种方法能够进行更多的乘加运算，使得加速器的吞吐率提高一倍。此外，本文使用“三叉树”三路加法树来实现多个加法对乘法结果的累加，与“二叉树”结构相比，能够降低时钟延时，并且大大减少 LUT 和 FF 的消耗。

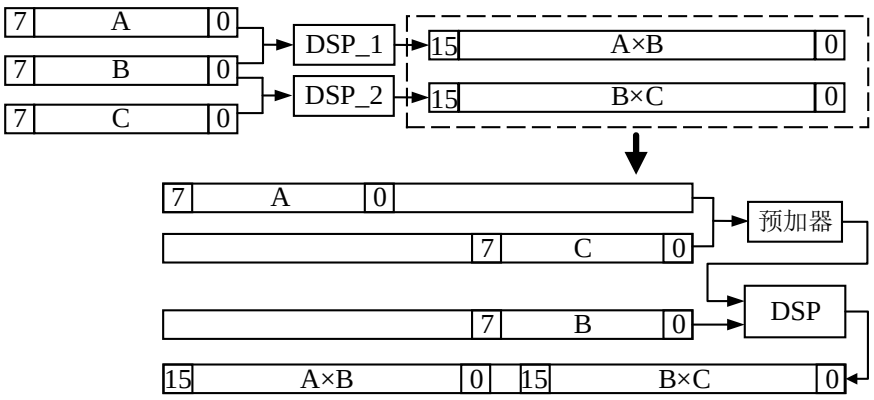


图 3-4 两个 INT8 乘法映射到一个 DSP48

3.2.2 池化模块设计

池化操作是一种常见的降采样技术，用于减少数据量并保留重要的特征。池化操作通常在卷积层之后进行，在池化窗口内计算。池化操作可分为最大池化、平均池化以及 L2 池化等[42]，其中最大池化是最常见的一种，结合硬件的实际情况，本文主要实现最大池化模块的设计。

如图 3-5 所示，3×3 最大池化模块共用到$(2l+4)$个寄存器和 2 个 3 输入比较器，其中l为卷积运算后特征图的行长度。在 3×3 最大池化中，当首个输入数据传输至第$(2l+1)$个寄存器时，移位寄存器每接收一个值，比较器 1 将比较 3 个输入，并选出最大值送至后三个移位寄存器中。随着首个最大比较值到达第$(2l+4)$个寄存器时，移位寄存器每接收两个值，比较器 2 比较一次，比较完成的结果即为最终的最大池化输出，与直接级联结构相比，使用更少的比较器数量，更节省资源。

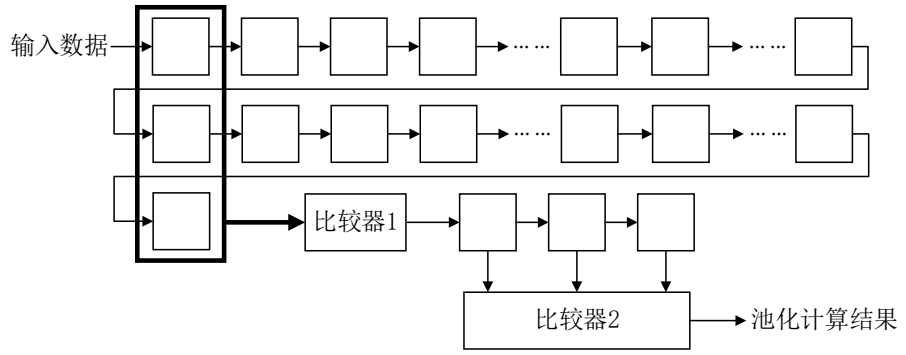


图 3-5 3×3 最大池化计算硬件结构

基于上述思路，如图 3-6 所示的 2×2 最大池化模块需要两个比较器，与直接级联的结构相比可以节省 1 个比较器资源。采用此种结构，无需等待卷积层完成所有计算后，再开始池化计算。每计算完一次卷积计算后，相应的结果即刻被送入池化模块进行比较，这样不会引起加速器整体延迟的增加。

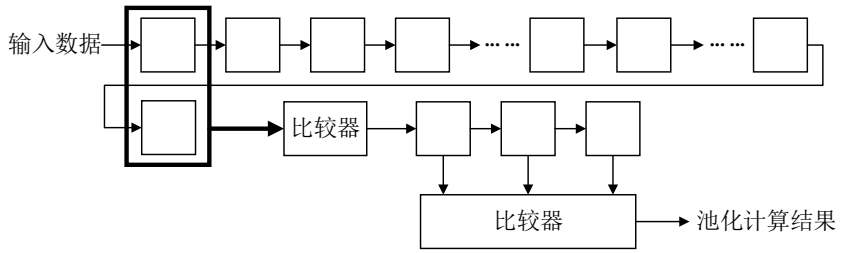


图 3-6 2×2 最大池化计算硬件结构

3.2.3 量化模块设计

由 2.3.2 节可知，卷积加速器采用的是经过 8bit 量化训练后的网络模型，并且输入特征图和权重数据都是 8bit。量化模块如图 3-7 所示，在卷积计算过程中，卷

积模块输出 16bit 的乘法结果，并将此结果进行累加，为了防止中间数据溢出，采用 18bit 进行数据存储。

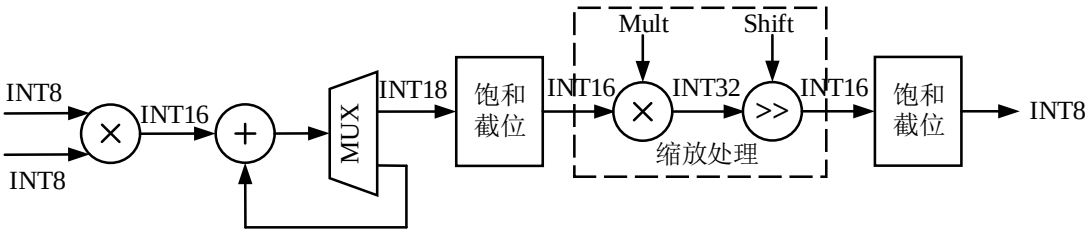


图 3-7 量化模块硬件结构

在该量化模块中，首先将 INT18 数据饱和截断为 INT16 数据，然后对 INT16 数据进行缩放处理，具体操作是与 Mult 参数进行相乘，相乘后的结果根据 Shift 参数进行右移，从而得到 INT16 数据，最后饱和处理将其截断为 INT8 形式的输出数据。其中，这里的 Mult 参数和 Shift 参数对应于公式（2-20）中的 M_0 和 n。

在 INT18 数据截断为 INT16 时，需要首先判断符号位，对于正数溢出的情况，将其截断为 INT16 的最大正数 0x71FF；对于负数溢出的情况，截断为 INT16 的最小负数 0x8000；对于其他可由 16 位有符号数表示的数据，进行简单截断。此外，n 位右移操作不是普通的向负无穷舍入的右移，而是向最近的整数舍入的右移。在这个过程中计算误差可能会逐级累加，导致结果出现较大的偏差。为了解决这个问题，在执行右移操作之前，先检查右移将要丢弃的最高位的值，即缩放因子乘积结果中第 n 位的值。若值为 1，则需要在右移后的结果加 1；若为 0，则结果保持不变。通过这种方式，右移操作由向负下舍入变为向零舍入，从而保证数值处理的精确性和一致性。

3.2.4 激活函数模块设计

本文采用线性插值查找表法实现激活函数层[43]。该方法将激活函数值域分成变化较为陡峭的区域和变化相对平缓的区域两部分。两部分的查找值在 CPU 上预先计算并根据相应的索引值，分别存放于细粒度查找表 $Table\_X$ 与粗粒度查找表 $Table\_Y$ 内。$Table\_X$ 存储坡度较为显著的区域，具体范围为 X_{min} 到 X_{max}，查找步进值为 X_{step}，索引值范围为 0 到 $X-1$。而 $Table\_Y$ 则存储坡度较为平缓的区域，其范围为 Y_{min} 到 Y_{max}，查找步进值为 Y_{step}，索引值范围为 0 到 $Y-1$。

$$X = \frac{X_{max} - X_{min}}{X_{step}} \tag{3-1}$$

$$Y=\frac{2(Y_{max}-X_{max})}{Y_{step}} \tag{3-2}$$

对于给定的查找输入值 m，首先确定其查找范围是位于 $Table\_X$ 还是 $Table\_Y$。如果查找范围落在 $Table\_X$ 内，则计算可知该值位于索引 $index(m)$ 和索引 $index(m)+1$ 之间。计算公式如下：

$$index(m)=floor(\frac{m-X_{min}}{X_{step}}) \tag{3-3}$$

$$\alpha=\frac{m-X_{min}}{X_{step}}-index(m)=\frac{m-X_{min}}{X_{step}}-floor(\frac{m-X_{min}}{X_{step}}) \tag{3-4}$$

$$a=Table\_X\left[index(m)\right] \tag{3-5}$$

$$b=Table\_X\left[index(m)+1\right] \tag{3-6}$$

其中，$index(m)$ 为 m 的索引值整数部分，α 为其小数部分。a 为 $Table\_X$ 中小于 m 且最接近 m 的查找值，b 为 $Table\_X$ 中大于 m 且最接近 m 的查找值，根据线性插值公式，m 的非线性激活函数值为：

$$f(m)=(1-\alpha)a+\alpha b \tag{3-7}$$

同理，从 $Table\_Y$ 中取值的方法与上述过程类似。

基于上述的分析，本文的激活函数模块如图 3-8 所示。首先利用比较器和 $|X_{min}|$ 判断输入索引值应该送至 $Table\_X$ 还是 $Table\_Y$。然后经过减法器与乘法器的处理，并通过最低有效位左移放大，以得到索引值整数部分 $index$ 和小数部分 α。接着，查找表内对应的 $index$ 值与 α 一起输入到线性插值器中得到临时结果。经右移缩放及数据选择器筛选合适的输出通道后，完成查找值的输出过程，从而得到非线性运算的结果。

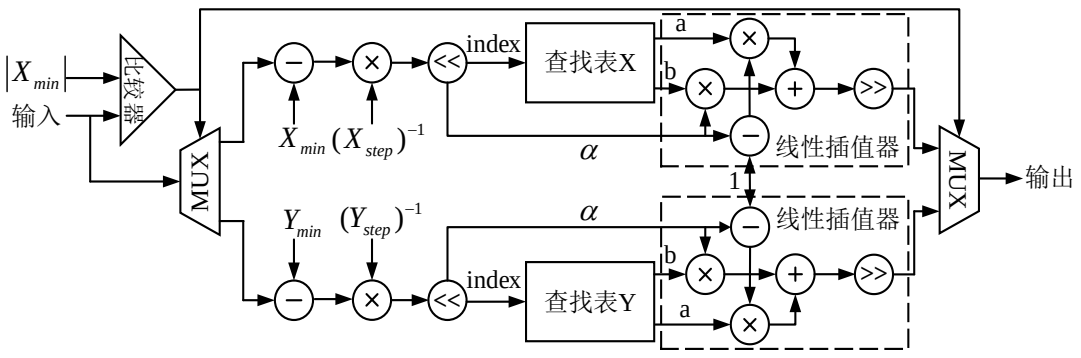


图 3-8 激活函数模块硬件结构

3.3 缓存单元设计

卷积加速器的性能与片上存储的设计密切相关，而缓存单元的设计则是其中的关键部分。缓存单元涉及特征图缓存、权重缓存、输出缓存和外部缓存，这些模块在卷积加速器中发挥着重要作用。下面将对每个模块进行分析介绍。

3.3.1 特征图缓存设计

卷积操作通过连续滑动窗口的方式进行，这导致它们之间通常会共享大量的重叠区域。基于行缓存思想，本文提出如图 3-9 所示的特征图缓存设计，减少了对外部存储带宽的需求。假设卷积核的大小为$W \times L \times H$，步长为S，输入特征图缓存区深度为$((W/S)+\theta)-1$。首先，将特征图数据从 DDR 中读取出来，并将其按行缓存在特征图缓存区中。当卷积开始时，从缓存区中的θ个并行输出端口按列读取θ个滑动窗口的像素数据，并在每个行缓存区中提前存储足够的行数，从而实现同时按列方向θ个窗口的并行卷积。最后，重用特征图缓存区中的数据依次进行卷积。

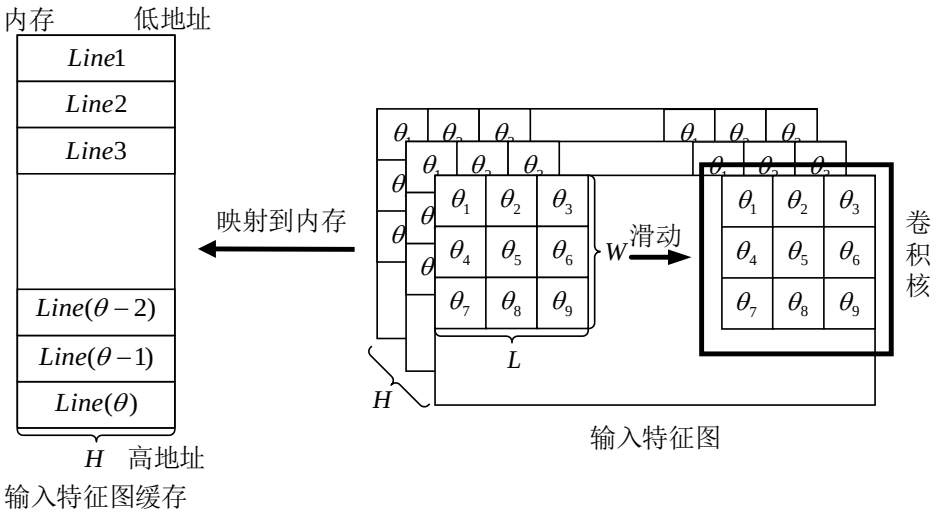


图 3-9 输入特征图缓存

3.3.2 权重缓存设计

每个卷积模块需要一个权重缓存模块来存储算法的权重参数。鉴于在卷积操作中可以避免使用大量的零权重参数，因此可以仅存储非零的权重参数。本文提出了如图 3-10 所示的内存排布方案，能够满足同时读取多个数据且低延迟的要求。假设在卷积过程中需要使用$X \times Y$大小的空间来读取K个卷积核的权重，则读取完全部权重参数需要产生K个时钟周期的延迟。通过采用三维的只读存储器架构，将$X \times Y$的存储器转换为$Z \times N \times Y$的存储器，其中$Z = X/N$，这使得可以通过同

一地址同时从 Z 个存储器中读取权重参数，进而提高权重的访存效率。

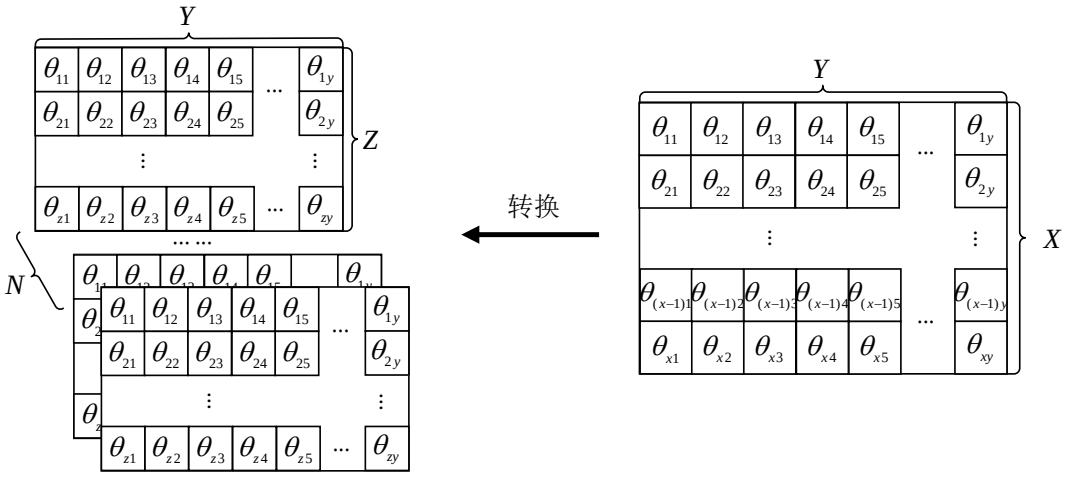


图 3-10 输入权重缓存

3.3.3 输出缓存设计

输出缓冲的主要作用是存储中间计算结果。如图 3-11 所示，本文使用 FIFO（First In First Out，先入先出）缓存对中间结果进行存储[44]，主要有四种状态。首先是等待输出数据变为有效。其次是将第一个输出通道的结果存入到 FIFO 缓存中。第三个状态是当下一通道数据到来时，从 FIFO 缓存中读取之前的数据，与将要输入的数据做累加运算，然后将结果存入 FIFO 中。在最后的状态中，随着最后一个通道的到来，从 FIFO 缓存中读取之前通道的数据，并与最后一个通道的数据做累加运算，将结果输出给下一个模块。

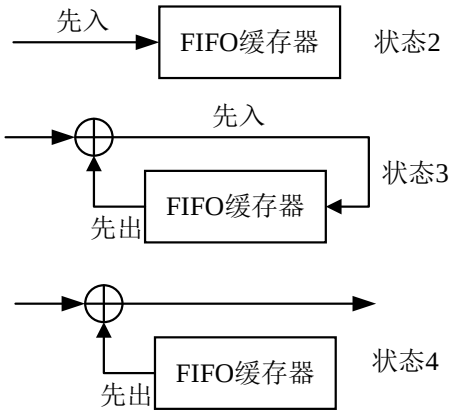


图 3-11 输出缓存模块

3.3.4 外部缓存设计

外部缓存模块用于储存特征图数据和权重。在网络训练完后，需要重新排序各个卷积层的权重，将其映射到 DDR 中不同的区域，这样能够减少延时，提高存储访问速率。图 3-12 为大小为 3×3 的卷积核权重的排列方式。

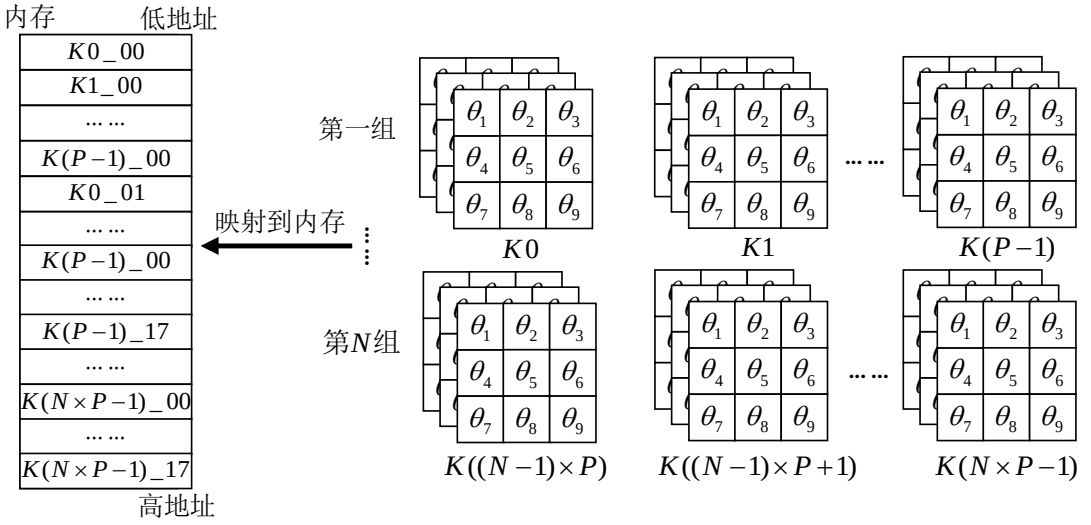


图 3-12 权重在外部存储器中的排列方式

假设特征图被划分为N组，每组P份，每份编号且大小为$1\times1\times P$，P为通道并行度。在内存排列时，首先处理第一组数据，将其中的θ_1数据放置在指定的内存地址中，然后是同组的θ_2数据也被分配到相应的内存地址，当第一组内的所有数据都被存储后，便继续按照此方法依次处理后续的分组，直至所有分组的数据都被完整地存储于相同大小的内存中。通过这种方式实现了权重参数的复用，提高系统整体的加速效率。

3.4 控制单元设计

卷积加速器依靠外部调度和控制，高效执行卷积、激活及量化等核心计算任务。为简化系统设计，避免复杂的状态机控制，采用 PS 端的设备驱动通过配置总线对状态和控制寄存器进行配置[45]。

本文的加速器涉及 7 个寄存器，借助 AXI-Lite 总线完成 PS 端对这些寄存器的读写操作，这样可有效地控制卷积加速器。表 3-1 为寄存器的配置。地址为 0x00 的寄存器控制加速器接收输入特征图、权重和偏置数据或发送输入特征图数据，控制卷积和池化运算并判断加速器是否完成本次计算。通过配置该寄存器的前 5 位控制字，实现 5 种不同类型的任务。每种任务完成后，通过 AP_DONE 信号触发中断，向 PS 端反馈任务完成情况。地址为 0x04 的寄存器配置加速器卷积计算的轮数。地址为 0x08 的寄存器用于设置量化参数中的比例因子，分为乘法操作（Mult）状态和移位操作（Shift）状态两种。地址为 0x0C 的寄存器则设置量化参数中的零点。地址为 0x10-1C 的寄存器提供加速器的地址信息。

表 3-1 7 个寄存器配置列表

寄存器地址	寄存器名	位宽	参数描述
0x00	控制寄存器	10bit	Bit[0]：输入特征图数据的接收 Bit[1]：权重数据的接收 Bit[2]：偏置数据的接收 Bit[3]：输出特征图的发送 Bit[4]：卷积运算的使能 Bit[5]：当前的卷积结果是否池化 Bit[6]：是否是第一次卷积 Bit[7]：是否是最后一次卷积 Bit[8]：乒乓缓存的指针指向的内存区域 Bit[9]：当前数据是否有效
0x04	卷积参数寄存器	31bit	Bit[0:15]：缓存多少数据进行卷积 Bit[16:27]：当前特征图一行有多少数据 Bit[28:30]：当前特征图的行缓存类型
0x08	量化参数寄存器 1	20bit	Bit[0:15]：Mult 状态的比较因子 Bit[16:19]：Shift 状态的比较因子
0x0C	量化参数寄存器 2	16bit	Bit[0:7]：输入零点的设置 Bit[8:15]：输出零点的设置
0x10-1C	3 个地址寄存器	32bit	Bit[0:31]：权重参数读地址 Bit[0:31]：偏置参数读地址 Bit[0:31]：输出特征图发送地址

当系统上电后，需要对 7 个寄存器和 DMA 相关寄存器进行复位，然后将寄存器相应的位设置成当前状态的控制指令，之后，DMA 会根据这些设置将数据传输到计算单元，计算单元接受数据后会判断当前数据是否缺失，如果缺少数据，则发送补充数据请求并生成中断，中断发生后，将进行下一组寄存器的设置，然后继续这个循环过程。

3.5 加速器优化策略

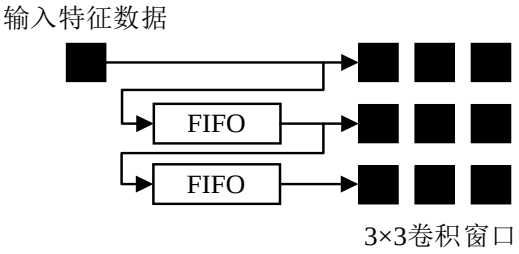
3.5.1 行缓存结构优化

（1）深度可配置行缓存结构

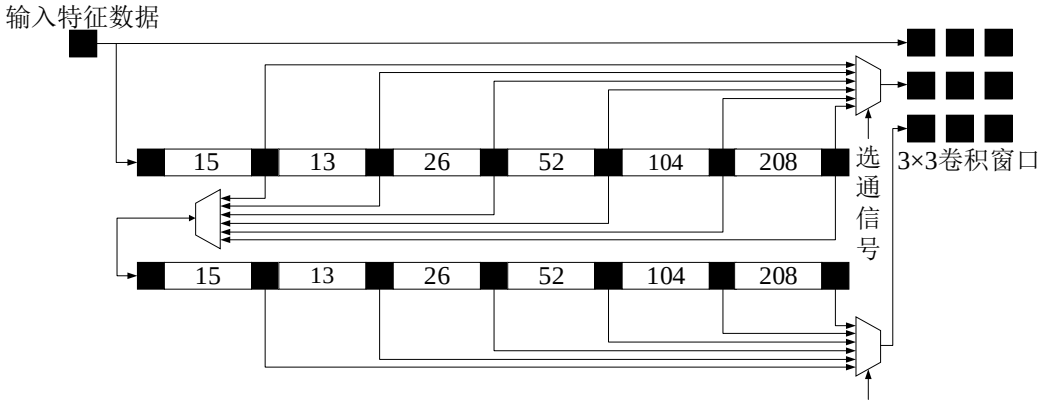
在深层神经网络的卷积运算中，特征图的大小往往多种多样。为此，本文对如图 3-13（a）的行缓冲结构进行了改进和扩展，以更好地适应卷积神经网络推理计算的要求。这一改进旨在提高卷积加速器的通用性和灵活性，使其能够处理多种大小的特征图，增强其在不同应用场景下的适用性。

根据改进的 YOLOv3-tiny 网络结构，本文对大小为[13，26，52，104，208，416]的特征图行缓存时，首先提前配置深度最长的行缓存，然后按照特征图大小

配置抽头的深度，最后通过数据选择器来选择所需要的行缓存深度，从而匹配当前的特征图的尺寸，具体结构如下图 3-13（b）所示。



（a）传统行缓存结构

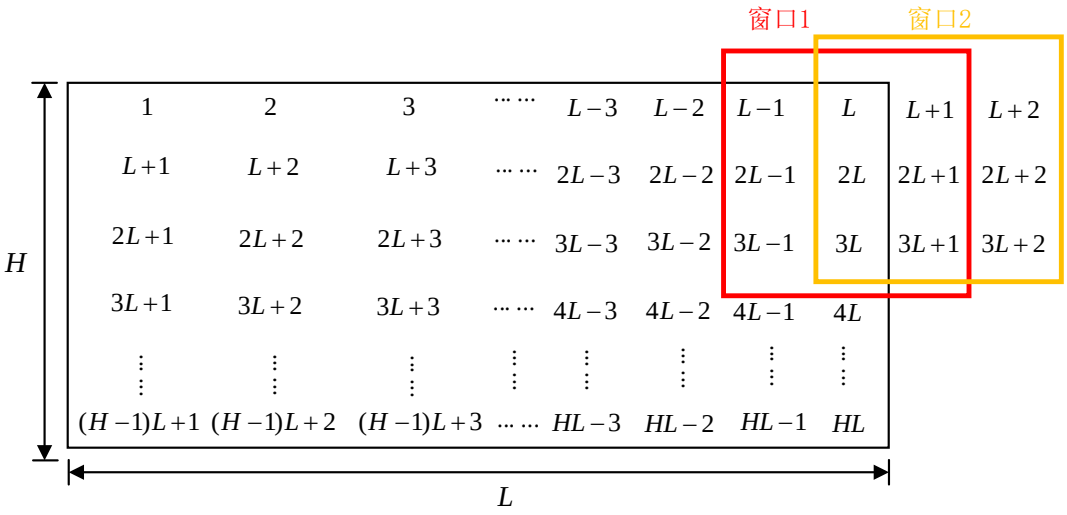


（b）深度可配置行缓存结构

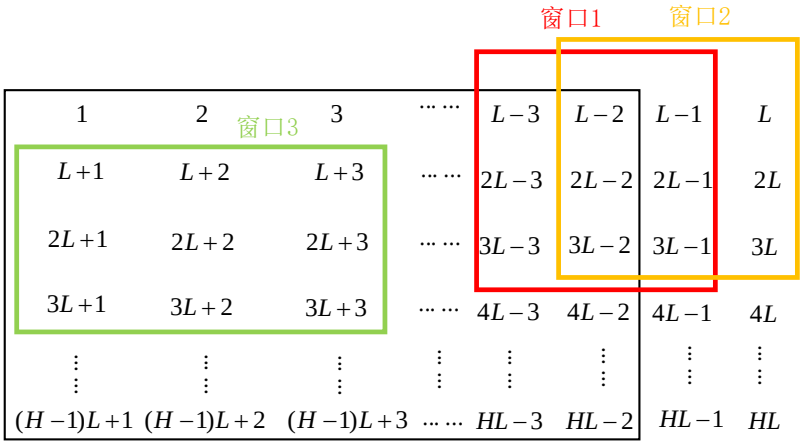
图 3-13 两种行缓存结构的区别

（2）带有列缓存功能的行缓存结构

像素值在行与行转换时，会存在无效数据传输时间。对于大小为 $H \times L$ 的图像，卷积核大小为 3×3，步长为 1 而言，无效数据传输时间为 2 个时钟周期。如图 3-14（a）所示，当卷积窗口到达窗口 1 和窗口 2 时，对于卷积计算而言，这两个窗口中的数据是不需要的，但它们依然会进入行缓存中，消耗了两个时钟周期。这会增加卷积计算的延时，导致整个 CNN 网络的运行时间延长。如图 3-14（b）所示，本文对像素值图像的第 $(L-1)$ 列和第 L 列的像素值，即 $[L-1, 2L-1, 3L-1, 4L-1, ..., HL-1]$ 和 $[L, 2L, 3L, 4L, ..., HL]$ 进行单独缓存处理。当卷积窗口移至窗口 1 时，将先前缓存的第 $(L-1)$ 列的 3 个数即 $[L-1, 2L-1, 3L-1]$ 赋给窗口 1 的第 3 列。等待一个时钟周期后，将这些数再分配到窗口 2 的第 2 列。与此同时，将缓存在第 L 列的 3 个数即 $[L, 2L, 3L]$ 赋给窗口 2 的第 3 列。那么，在下个时钟周期后，数据将在窗口 3 出现。此时，所有数据都是需要的，避免了无效数据的出现，进而减少无效数据传输时间，缩短卷积计算的延时。图 3-15 为优化后的行缓存结构。



（a）像素值图像



（b）像素值图像

图 3-14 像素值图像

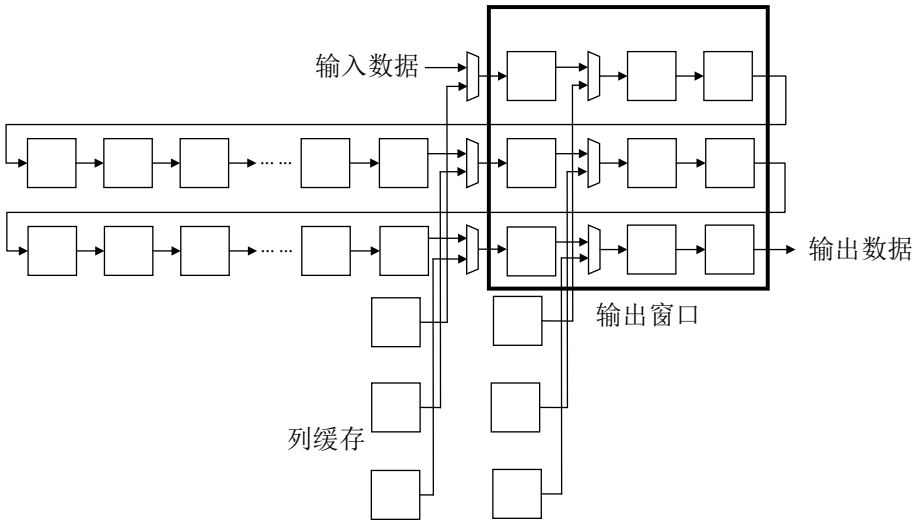


图 3-15 带有列缓存的行缓存结构

3.5.2 乒乓和全流水设计优化

为有效降低卷积计算的延时，本文设计了如图 3-16 所示的乒乓和全流水结构。

在单次计算中，考虑了行缓存预填充延时和流水线延时。预填充延时是第一个像素值从数据输入端口到输出端口所消耗的时间。流水线延时是流水线中寄存器所消耗的时间。对于行缓存预填充延时，采用了乒乓结构来使其被淹没在单次计算的总延时内。而流水线延时则是由乘法、加法等操作引入的，把计算单元设计成全流水结构使得该延时被淹没在带流水线的计算延时内。

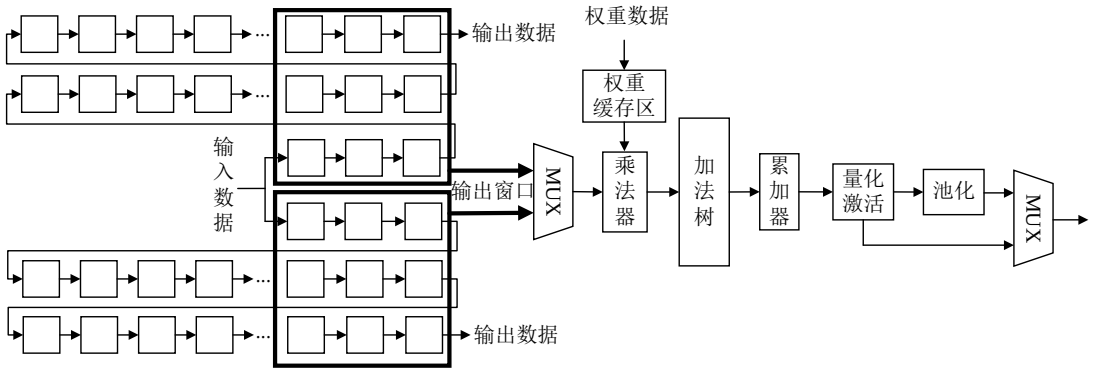


图 3-16 乒乓和全流水结构

3.5.3 并行度设计优化

在本文的卷积加速器中，通过乘加树内部的流水线计算实现卷积核内展开，并对输入和输出两个维度进行并行处理[46]。这两个维度的并行，实质上是多个相同卷积模块的并行，因此需要综合考虑输入输出数据的位宽，片上资源的消耗，存储读写带宽的占用和数据的组织形式等。

首先考虑输入输出数据的位宽。模型在量化后，卷积计算使用 8 位的数据进行，且每个通道的数据都为 8 位。本文采用 AXI DMA 来实现数据的传输，将输入输出带宽设为 64 位，这使得可以同时处理 8 组特征图。其次，考虑片上资源的消耗。本文中所使用的开发板的 DSP48 资源为 360 个，设计时需要保留 20%的余量，则剩余的 DSP 最多可支持例化 32 个卷积模块。而由 3.2.1 节可知，每个卷积模块能够完成 2 组 INT8 数据的卷积计算，故并行度设为 32×2=64，这意味着在每个时刻，能够并行执行 64 组 8 位的卷积操作。

如图 3-17 所示，64 位的权重数据通过 AXI DMA 进行传输，将其分成 8 组。通过串行转并行的方式，8 组数据被分配到 8 个 BRAM 中，从而分配完 64 组权重。对于 3×3 大小的卷积核，展开并拼接权重后得到的 72 位数据正好适配 BRAM36K 的内部架构，从而实现资源的无冗余存储。

通过这种并行设计，64 组卷积计算中每组都共享 8 路输入特征图数据。尽管输入数据相同，但由于每组卷积核具有独立的权重参数，因此在不同权重的作用

下，每组卷积运算都能产生 8 组不同的输出特征图数据。

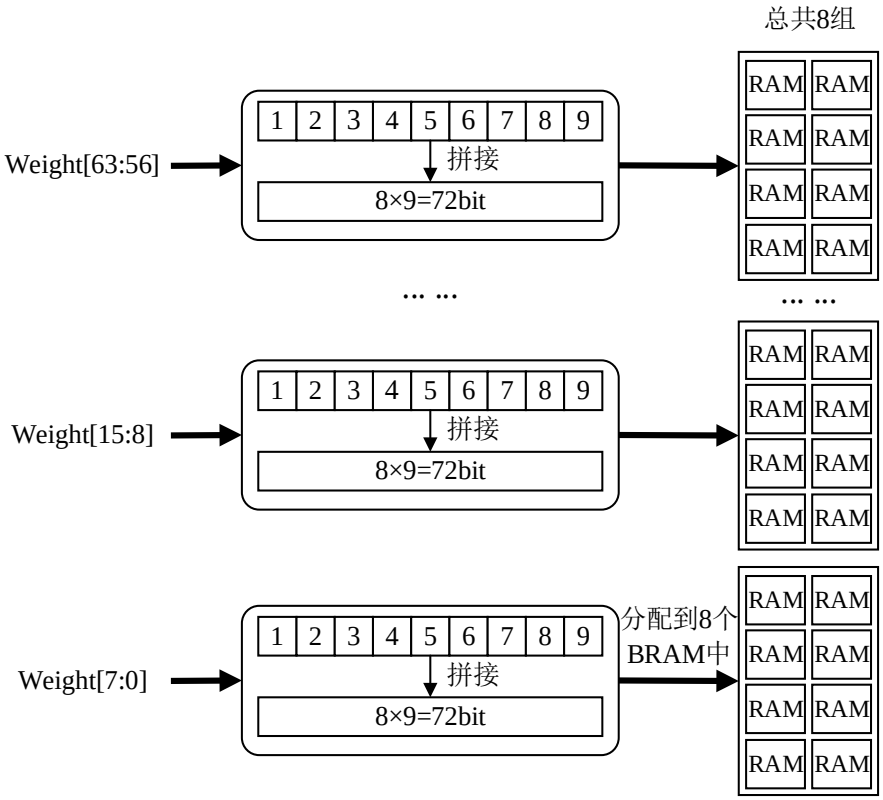


图 3-17 多通道权重存入 64 个 BRAM 中

3.6 加速器功能性验证

1. 加速器整体仿真

如图 3-18 所示，首先通过 AXI 总线发送卷积计算所需的数据，发送顺序依次为权重，偏置以及输入图像数据。输入图像大小为 16×16×8，数据为第一行 0~f，第二行 10~1f，最后到第 16 行 f0~ff。第一帧图像数据 image 寄存器值为 0，第二帧图像数据 image 寄存器值为 1，以此类推。在发送第二帧图像时，卷积神经网络加速器开始进行计算。卷积窗口中的图像数据从 222120121110020100 开始输出，并存入 RAM 中，然后从权重缓存区中取出权重数据。系统总线位宽为 64bit，在每个时钟周期内，可以读取 8 个特征图数据或权重数据（8×8bit），并进行这 8 个数的乘累加运算，其中 8 个偏置均为 1。图 3-19 中第一个卷积窗口的结果为 0038a。8 个通道相加 32'h38a×32'd8=32'h1c50，该结果需要经过量化。scale 和 shift 的值设为 007f 和 0。quant_result 为量化之后的结果。relu_out 为 ReLU 函数激活后的结果。当 pool_enable=1 时，数据开始进行 2×2 池化操作，池化结果存入 RAM 中，波形图如图 3-20。如图 3-21 所示，通过 AXI Lite 总线配置 send_enable 的值。当 send_enable=1 时，通过 AXI Stream Master 发送数据，结果由 slv_data 保存。

这个寄存器是计算的最终结果。slv_data[7:0]表示输出图像第一个通道的数据。slv_data[15:8]表示第二个通道的数据，以此类推。图中第一个数据为 0，是由于在卷积加速器中数据在 Relu 函数中有 1 个周期的延迟，但是 valid 和 addr 没有延迟，所以导致存储的数据有 1 个周期的延迟。故第一个输出数据为 1c，符合预期结果。

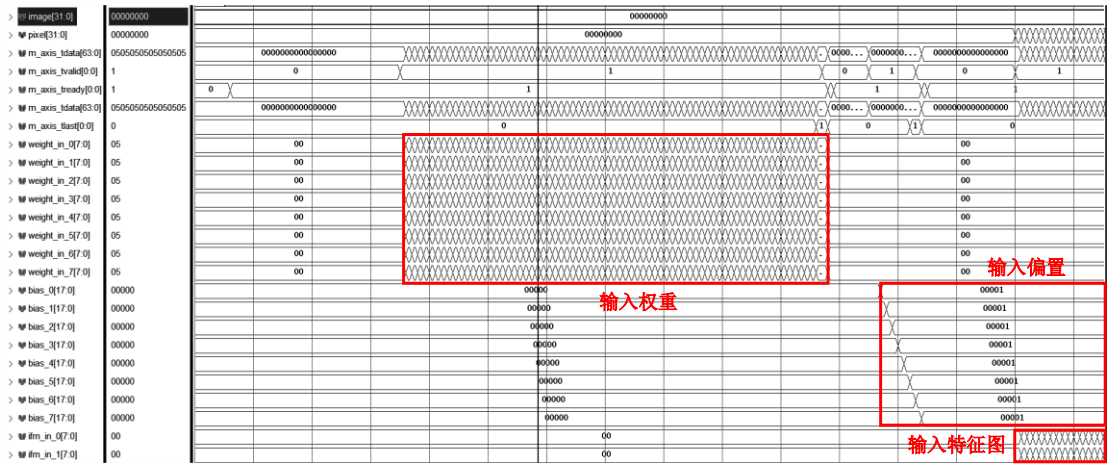


图 3-18 输入图像，权重以及偏置数据的发送

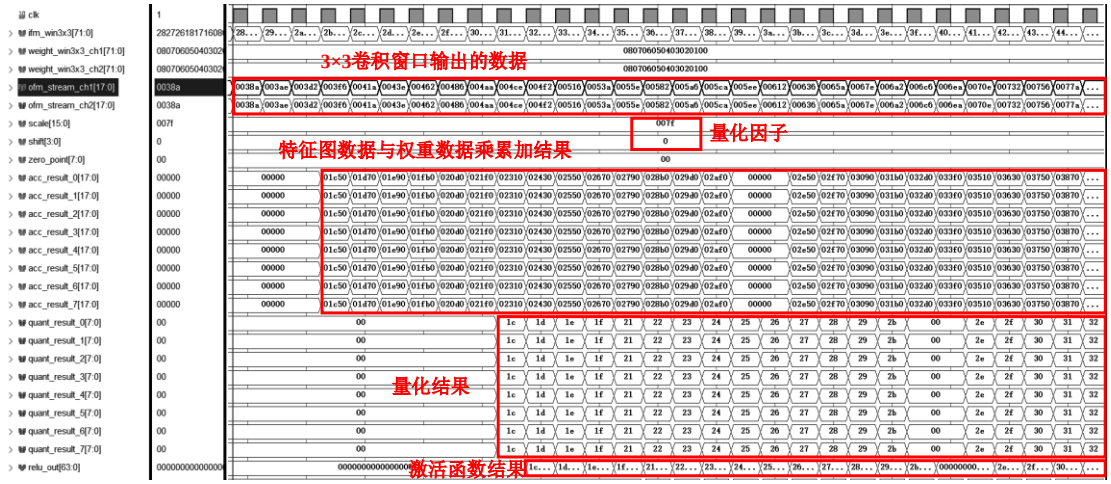


图 3-19 乘累加与量化激活计算结果

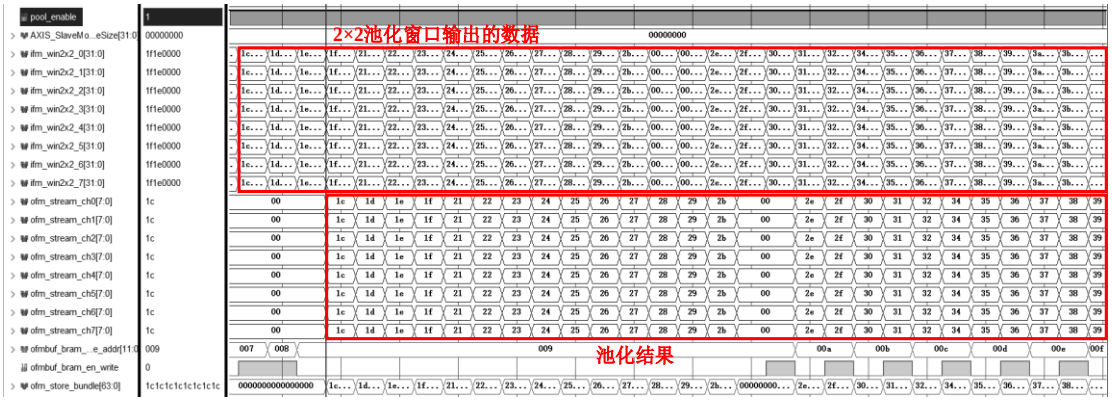


图 3-20 池化计算结果

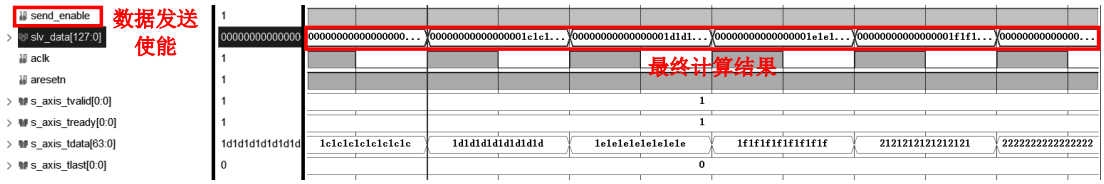


图 3-21 最终计算结果

2. 资源消耗分析

硬件设计的大小和成本与资源消耗紧密相连，本文采用的硬件平台是 Xilinx 的 XCZU3EG-SFVC-1-E 芯片。在本文中，例化了 8 个芯片资源。加速器接收 8 个输入特征并生成 8 个输出特征，同时进行 64 组卷积计算。表 3-2 展示了所使用的逻辑资源使用量，DSP 使用了 288 个，查找表 26707 个，触发器 20374 个以及 39.5 个 BRAM。与文献[23]和文献[47]中所提出的加速器相比，本文的加速器通过使用更少的 LUT、FF 和 BRAM，使用较多的 DSP，使得吞吐率（GOPS）分别是前者的 1.7 倍和 5.9 倍，这显然体现了加速器较低的资源密度。

表 3-2 加速器资源消耗

	LUT	FF	BRAM36K	DSP48	GOPS	Clock
本文	26707	20374	39.5	288	144.2	250MHz
文献[23]	29867	35489	85.5	190	84.3	214MHz
文献[47]	27548	50801	92.5	181	24.3	100MHz
比率	0.89/0.97	0.57/0.40	0.46/0.43	1.52/1.59	1.71/5.93	—

3.7 本章小结

本章首先介绍卷积加速器的整体架构，包括计算单元、缓存单元以及控制单元。计算单元设计了卷积、池化、量化以及激活函数模块。卷积模块和池化模块均采用行缓存结构进行设计；接着设计了缓存单元，提出了关于输入特征图和权重的存储方案；然后，控制单元部分主要介绍了卷积加速器 7 个寄存器的配置；并提出了一些加速器的优化策略，包括深度可配置和带有列缓存功能的行缓存结构、乒乓和全流水结构降低计算时延以及并行度设计等优化；最后，对本文的加速器进行了功能性仿真和逻辑资源对比。

第 4 章 目标检测系统实现与验证

本章首先介绍了目标检测系统的总体设计方案，分为硬件和软件两个部分；然后简要概述了硬件开发平台，并详细介绍了 PL 端和 PS 端工程的实现过程。接着开发嵌入式 Linux 环境。最后，搭建用于检测口罩和头盔的硬件平台，并对系统的运行结果进行了分析。

4.1 系统总体设计

目标检测系统总体设计结构图如图 4-1 所示。本文采用 ZYNQ 架构进行实现，PL 端主要实现图像采集模块、图像预处理模块、卷积神经网络硬件加速模块以及图像显示模块。PS 端实现外接驱动的调用、卷积神经网络加速器的控制、AXI-DMA 高速总线的配置以及 Linux 系统的开发等。

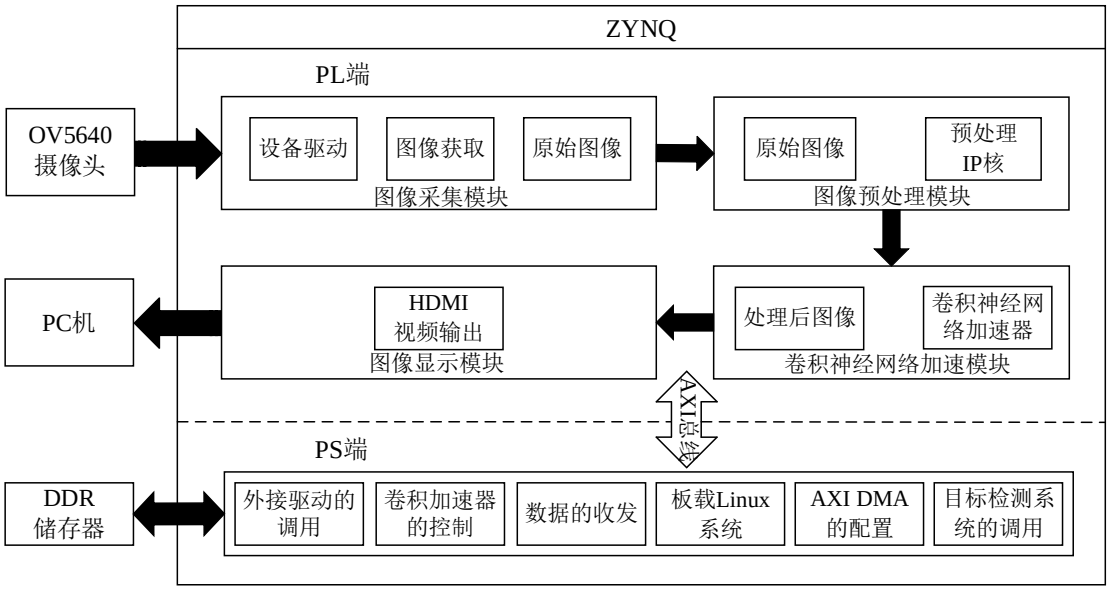


图 4-1 目标检测系统总体结构图

本文中的目标检测系统首先通过 OV5640 摄像头采集模块获取原始图像数据，然后经过图像预处理模块将 720P（1280×720）的原始图像缩放为 416×416，预处理后的图像传输到卷积加速器模块中进行目标检测模型的加速和推理，最后通过 HDMI 显示设备输出目标检测结果。

4.2 硬件开发平台

4.2.1 硬件开发板

本文选用如图 4-2 所示的 Digilent Genesys ZU 3EG 开发板作为硬件开发平台，

其搭载 Xilinx Zynq UltraScale+MPSoC ZU3EG 芯片，提供高性能的应用处理和实时控制[48]。除了强大的处理器外，还具备丰富的可编程逻辑资源，可满足各种应用需求。板载的外围设备和接口多样，包括 DDR4 内存、闪存存储、千兆以太网、USB、HDMI 等，同时配备 FMC 扩展插槽。其还享有丰富的开发和社区支持，是一个性能强大、灵活可扩展、性价比较高的开发工具。

图 4-2 Genesys ZU 3EG 开发板

4.2.2 OV5640 摄像头

本系统选用如图 4-3 所示的 OV5640 摄像头进行图像采集工作，其能够在不同分辨率下以最快的采集帧率运行[49]。OV5640 使用的是两线式 SCCB 接口总线，SCCB 控制器是一种用于连接应用处理器和图像传感器的组件，其主要功能是向图像传感器的特定寄存器写入数据，以实现对传感器的配置和控制，并监测图像处理器的工作状态。

图 4-3 OV5640 摄像头

4.3 PL 端实现部分

PL 端主要是搭建系统的硬件电路。如图 4-4 所示，本文使用 OV5640 摄像头以 60FPS 的速率采集分辨率为 720P 的视频流数据，经过预处理 IP 核后，通过 VDMA IP 核来实现数据与 DDR4 之间的高带宽直接存储访问。AXI DMA 则将存储在 DDR4 中的权重数据传输到加速器中，与视频帧缓存中的数据进行运算，并由 ZYNQ IP 核处理后，将结果传输到 DVI（Digital Visual Interface）转换 IP 核，此时，图像数据信号和时钟信号转换为最小差分信号（Transition Minimized Differential Signaling，TMDS），从而实现数字视频在 HDMI 显示屏上的显示。

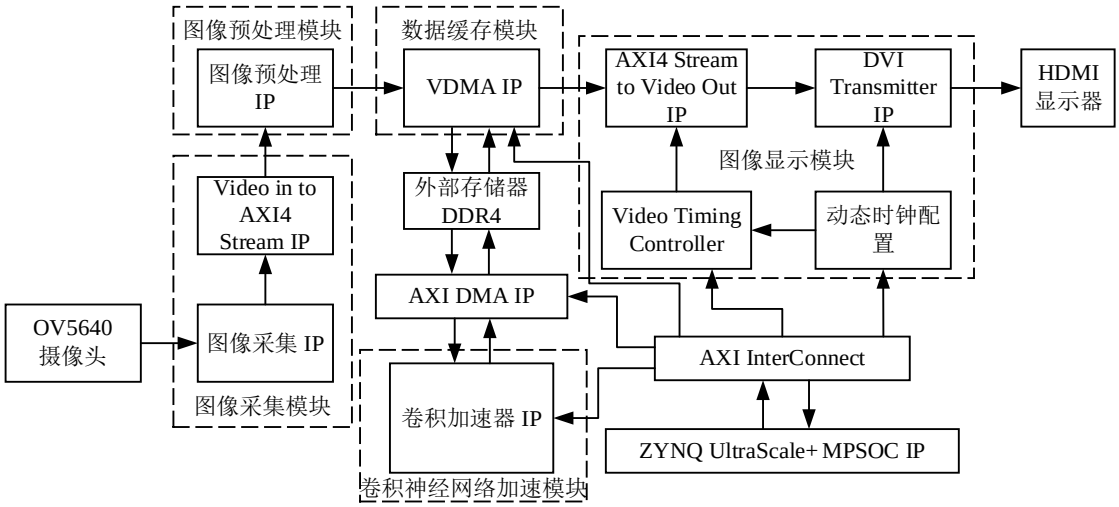
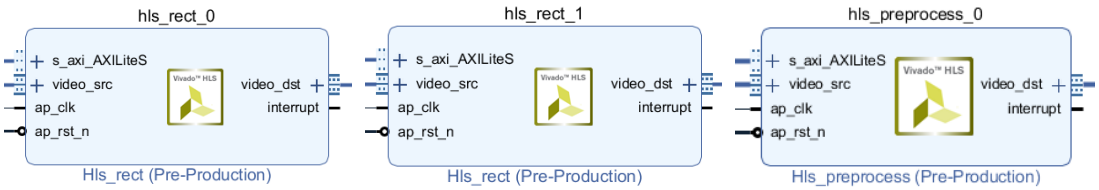


图 4-4 目标检测系统硬件设计结构图

4.3.1 图像预处理模块

如图 4-5 所示，本文采用了三个 IP 核来实现图像的预处理功能，分别是 HLS_rect_0，HLS_rect_1 和 Hls_Preprocess。Hls_preprocess 是一个自定义的图像缩放 IP 核，OV5640 摄像头采集到的图像大小为 1280×720，该 IP 核将图像缩放成 416×256。HLS_rect_0 和 HLS_rect_1 用于绘制 YOLO 检测出来的边框和类别。下面主要介绍图像缩放模块的设计。



(a) HLS_rect_0 IP 核　　(b) HLS_rect_1 IP 核　　(c) Hls_preprocess IP 核

图 4-5 图像预处理模块 IP 核

由于 CNN 的第一层输入特征图为 416×416，本文将输入图像经过横纵方向同比例缩放，生成 416×256 的 RGB 图像，然后结合像素填充最终生成 416×416 的 RGB 图像。本文选用如图 4-6 所示的双线性插值算法实现缩放[50]。

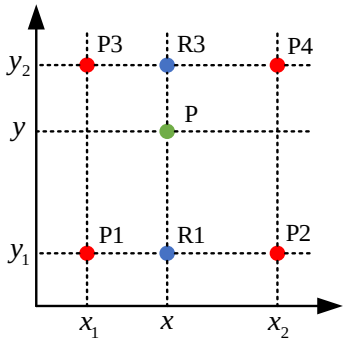


图 4-6 双线性插值原理图

双线性插值算法对水平和垂直方向进行线性插值。假设四个像素点坐标为 $P_1=(x_1,y_1)$，$P_2=(x_2,y_1)$，$P_3=(x_1,y_2)$ 以及 $P_4=(x_2,y_2)$，目标像素点的坐标为 $f(P)=f(x,y)$，整个计算公式如下。

$$f(R_1)=f(x,y_1)\approx\frac{x_2-x}{x_2-x_1}f(P_1)+\frac{x-x_1}{x_2-x_1}f(P_2) \tag{4-1}$$

$$f(R_2)=f(x,y_2)\approx\frac{x_2-x}{x_2-x_1}f(P_3)+\frac{x-x_1}{x_2-x_1}f(P_4) \tag{4-2}$$

$$f(P)=f(x,y)\approx\frac{y_2-y}{y_2-y_1}f(R_1)+\frac{y-y_1}{y_2-y_1}f(R_2) \tag{4-3}$$

如果选择一个坐标系使得四个像素点的坐标分别为(0，0)，(0，1)，(1，0)和(1，1)，那么公式可简化为

$$\begin{aligned}f(x,y)&\approx f(0,0)(1-x)(1-y)+f(1,0)x(1-y)+f(0,1)(1-x)y+f(1,1)xy\\&\approx f(0,0)\times C_1+f(1,0)\times C_2+f(0,1)\times C_3+f(1,1)\times C_4\end{aligned} \tag{4-4}$$

其中，C_1，C_2，C_3 和 C_4 表示双线性插值系数。由于 FPGA 上资源有限，所以需要将浮点形式的插值系数转换为定点形式再进行计算。计算公式如下：

$$\begin{cases}C_1=\dfrac{1}{256}((0\text{x}4000-X_{scale})\times(0\text{x}4000-Y_{scale}))\\C_2=\dfrac{1}{256}(X_{scale}\times(0\text{x}4000-Y_{scale}))\\C_3=\dfrac{1}{256}((0\text{x}4000-X_{scale})\times Y_{scale})\\C_4=\dfrac{1}{256}(X_{scale}\times Y_{scale})\end{cases} \tag{4-5}$$

其中，X_{scale} 和 Y_{scale} 是根据图像在水平和垂直方向上的缩放系数与当前输入像素的行列计数得到的。本文首先将插值系数左移一位，随后将得到的结果右移八位，从而实现双线性插值的近似计算。

如图 4-7 所示的图像缩放模块是基于双端口 BRAM 实现的，分为 RAM 写控制模块、RAM FIFO 模块、RAM 读控制模块、插值控制模块和插值运算模块。RAM FIFO 模块协同 RAM 写控制和读控制模块工作，负责图像缓存和时序控制。在图像缩放阶段，写控制模块选择需要写入的像素。完成写入后，可获得双线性插值所需的四个值。之后，读控制模块输出处理完的目标图像。

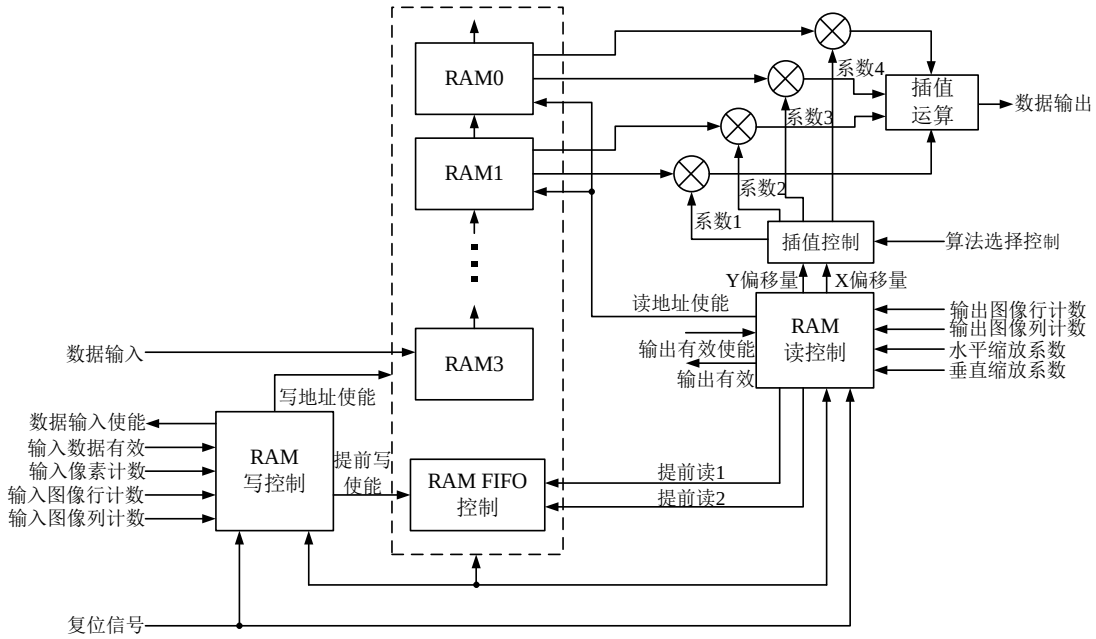


图 4-7 图像缩放模块示意图

本文使用 Vivado HLS 对图像缩放模块进行设计，如图 4-8 所示为图像缩放工作时的仿真波形图，*dOut* 表示缩放后的数据，*coeff* 为目标像素点与四个邻近像素点距离的加权系数，*xScale* 和 *yScale* 分别为水平和垂直方向的缩放比例，*inputXRes* 和 *inputYRes* 表示原始图像的水平和垂直方向像素数量，而 *outputXRes* 和 *outputYRes* 表示缩放后的图像的水平和垂直方向的像素数量。图像缩放模块资源使用情况如表 4-1 所示。

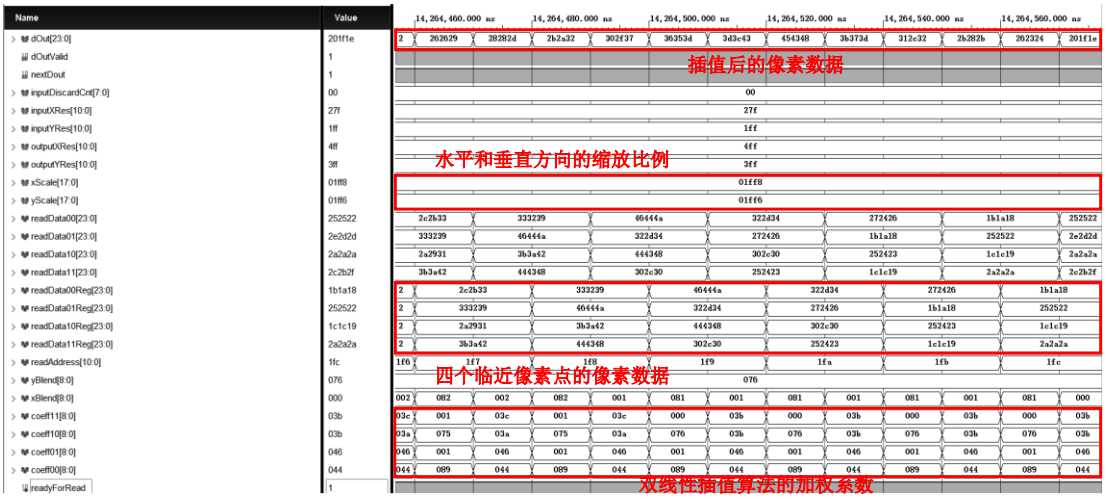


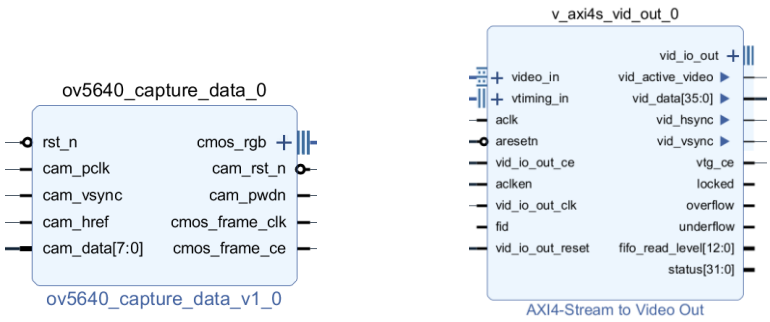
图 4-8 缩放模块仿真图

表 4-1 图像缩放模块资源使用情况

资源	LUT	FF	BRAM	DSP
使用量	1357	460	10	8

4.3.2 图像采集模块

如图 4-9 所示，图像采集模块包括用于驱动 OV5640 摄像头的图像采集 IP 核和转换数据格式的 Video In to AXI4-Stream IP 核。



(a) 图像采集 IP 核　　(b) Video In to AXI4-Stream IP 核

图 4-9 图像采集模块

图 4-10 为 OV5640 摄像头的输出时序图。其中，PCLK 为像素时钟，HREF 为行同步信号，D[9:2]为像素数据。在 OV5640 的 RGB565 模式下，只有高 8 位是有效数据。当 HREF 为高电平时，首先输出的是 RGB565 的高 8 位数据，其次是低 8 位数据。这两组数据组成完整的 16 位 RGB565 数据。为了完成从 RGB565 到 RGB888 的格式转换，需要在 RGB565 数据后添加 3、2 和 3 个零。如下图可知，要想在最稳定的时间点获取图像数据，数据采集应在像素时钟上升沿进行。

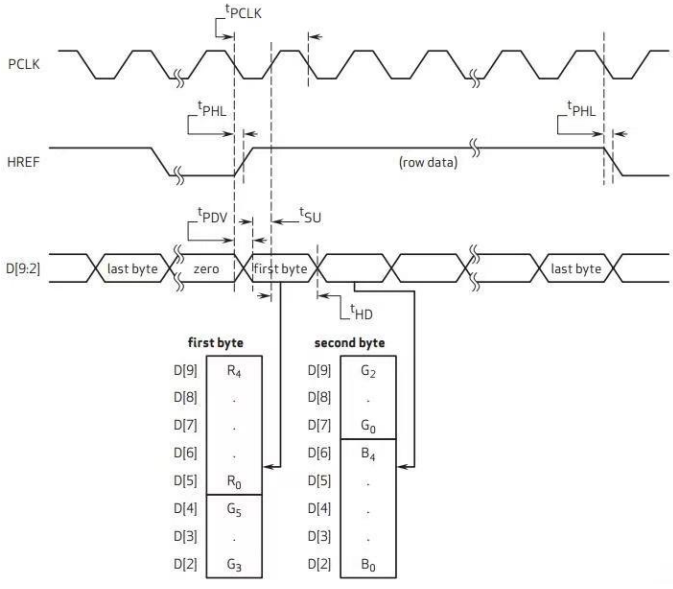


图 4-10 OV5640 输出时序图

图 4-11 为上板验证时利用在线逻辑分析仪（Integrated Logic Analyzer，ILA）抓取到的一些图像采集模块信号的波形图，图中 cmos_frame_href 为行同步信号，标志一行数据的开始和结束；cmos_frame_vsync 是场同步信号，标志一帧图像的

开始和结束；cmos_frame_data 则为输出的图像数据。

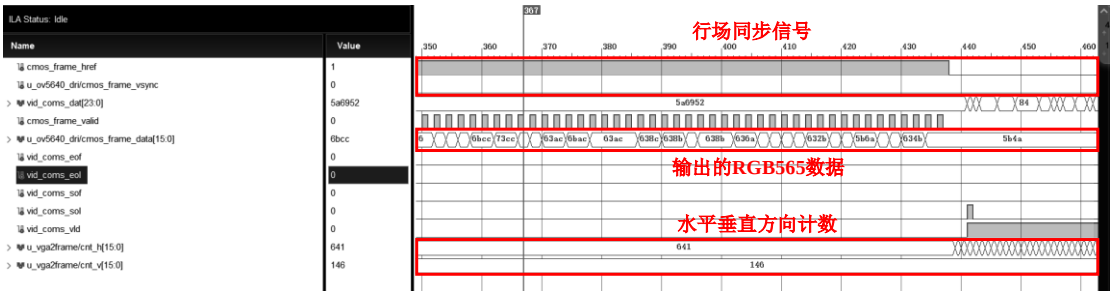


图 4-11 OV5640 仿真波形图

摄像头 IP 核以 RGB888 格式输出数据，但之后需要使用 AXI4-Stream 数据的形式进行后续图像处理和存储。为了实现这种数据格式转换，采用了 Video In to AXI4-Stream IP 核。

4.3.3 卷积加速器模块

卷积加速器模块是整个系统的核心模块，用于实现卷积神经网络的加速。其具体设计在第三章。如图 4-12 所示为封装后的卷积加速器 IP 核。

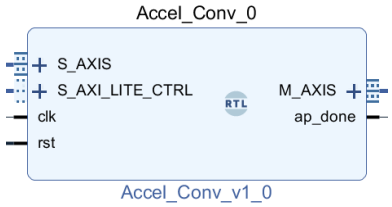


图 4-12 卷积加速器 IP 核

在目标检测系统中，ARM 硬核 CPU 通过 AXI Interconnect 与卷积加速器相连，这样可以让 ARM 控制 4 个用于接收输入数据、权重和偏置以及发送输出数据的 AXI DMA 数据流。这些数据通过 AXI SmartConnect 与 ZYNQ 的 HP 端口相连，以完成对 DDR 的读写操作。

4.3.4 数据缓存模块

数据缓存主要利用图 4-13 所示的 VDMA IP 核。VDMA 具有三个接口：AXI4 Stream 接口，其处理视频的输入输出；AXI4 Lite 接口，用于配置 VDMA 状态；AXI4 Memory Map，其负责视频数据的读写操作。本系统中使用了 VDMA_0 和 VDMA_1 两个 IP 核，分别用于缓存图像采集回显和卷积加速器推理的数据。

如图 4-14 所示，VDMA 的主要基本配置是帧缓存数量和 AXI4-Stream 数据宽度。帧缓存分配多个帧缓存区域用于数据的存储，允许系统以各自的速率进行读写操作。在本次设计中，VDMA_0 的帧缓存设置为 1。对于卷积加速器部分的

VDMA_1 帧缓冲设置为 3，这样可以有效避免因视频流数据的不停读取而造成画面撕裂的情况。根据系统的需要，本文将 AXI4-Stream 的数据宽度设置为 32 位。

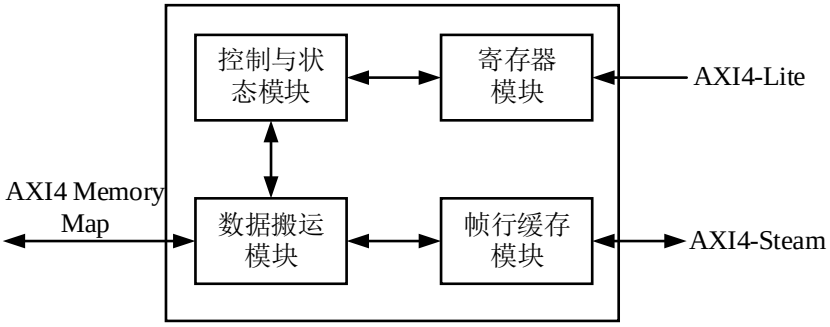
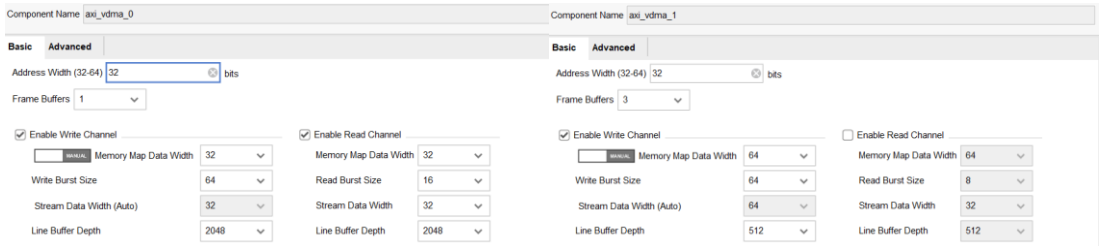


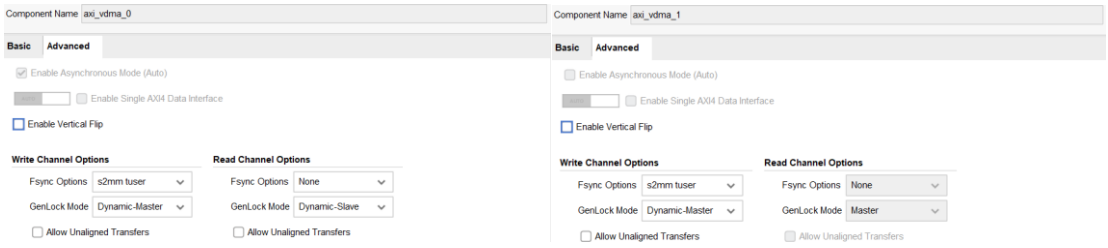
图 4-13 VDMA IP 核原理图



（a）VDMA_0 的基本配置　　（b）VDMA_1 的基本配置

图 4-14 VDMA IP 核的基本配置

如图 4-15 所示，VDMA 主要的高级配置有帧同步和同步锁相模式。对于写通道，帧同步使用 S2MM Tuser 模式以确保数据按帧精确同步；对于读通道，采用 None 模式以提高数据传输速度。这样的配置有助于优化视频处理流程，提高系统的整体性能和效率。VDMA 的 Genlock（生成锁相）特性通过匹配输入或输出视频流的帧率，以及调整相应的相位，确保了帧同步操作的准确性。本次设计中 VDMA_0 的写通道和读通道分别设置为 Dynamic-Master 和 Dynamic-Slave 模式，VDMA_1 的写通道和读通道分别设置为 Dynamic-Master 和 Master 模式。这样的配置提高了 VDMA 在视频处理应用中的灵活性和效率，确保了数据的稳定性和可靠性，同时也减少了对 CPU 的依赖。

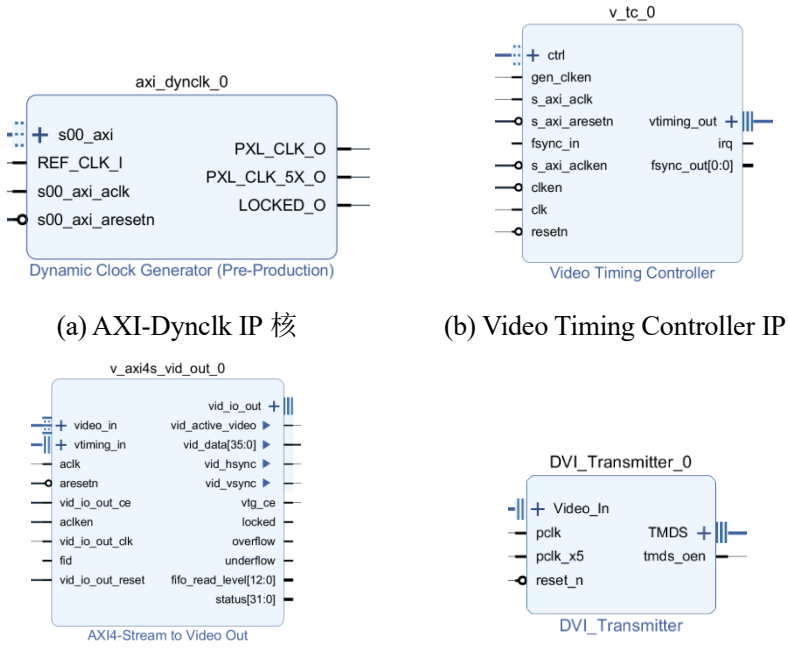


（a）VDMA_0 的高级配置　　（b）VDMA_1 的高级配置

图 4-15 VDMA IP 核的高级配置

4.3.5 图像显示模块

如图 4-16 所示，显示模块包括 AXI-Dynclk IP 核、Video Timing Controller IP 核和 AXI4-Stream to Video Out IP 以及 DVI Transmitter IP 核。



(a) AXI-Dynclk IP 核　　(b) Video Timing Controller IP 核

(c) AXI4-Stream to Video Out IP 核　　(d) DVI Transmitter IP 核

图 4-16 图像显示模块

AXI-Dynclk IP 在目标检测系统中动态调整时钟频率，从而为各个模块提供最适合的时钟条件。Video Timing Controller IP 核主要负责生成和管理视频时序信号，以确保视频数据的同步传输和准确显示。AXI4-Stream to Video Out IP 将 AXI4-Stream 格式的数据流转换成 RGB888 格式，而自定义的 DVI Transmitter IP 则是将 RGB888 格式的视频数据转换成适合传输的 TMDS，这些信号可以驱动 HDMI 接口，从而实现通过 HDMI 接口输出目标检测结果的功能。其内部结构如图 4-17 所示，输入视频信号首先经过 Encoder 模块编码后，由 Serializer 模块转换成串行格式，最后通过 OBUFDS 模块以 TMDS 信号的形式输出。其中，DVI 转换 IP 核的像素时钟（Pixel Clk）由 AXI-Dynclk IP 核生成。

图 4-18 显示了 Video Timing Controller（VTC）的参数配置。该 IP 核选择 AXI-Lite 接口，支持动态编程和参数更改；单行最大时钟数和行数设定为 4096，这定义了视频帧的最大分辨率；帧同步选择一个独立输出；启用 Generation 为 HDMI 接口提供一个同步信号，这有助于控制视频输出的时序，确保视频的流畅性和稳定性；在 Generation 配置中，启用了垂直和水平两个方向的空白输出、同步输出以及活动视频输出。

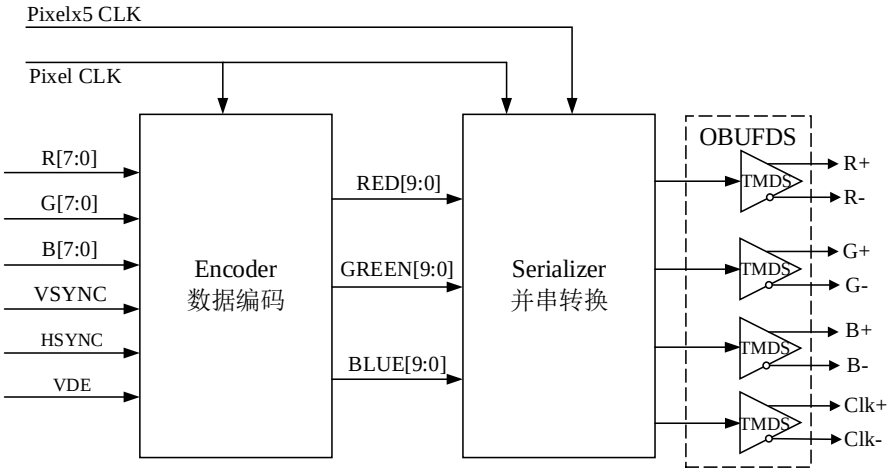


图 4-17 DVI 转换 IP 核结构图

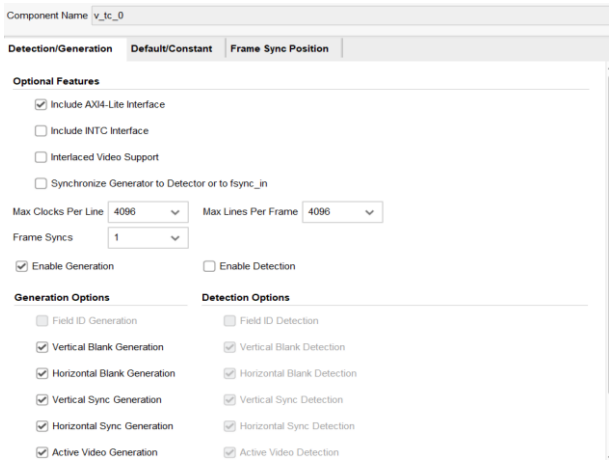


图 4-18 Video Timming Controller IP 核配置

4.3.6 整体设计

在完成上述的 IP 核的参数配置之后，还需要对 ZYNQ IP 核，AXI4-Stream Subset Converter IP 核以及 DMA IP 核等进行配置。

（1）ZYNQ IP 核配置

在 ZYNQ IP 核中添加 AXI_HP 和 AXI_GP 接口用于 PS 与 PL 中的 IP 核之间的控制和数据传输。配置 UART 用于输出打印信息，并通过 EMIO 配置摄像头寄存器，同时设置图 4-19 中的 FCLK_CLK0 为 250MHz 的时钟源。

图 4-19 时钟配置

（2）AXI4-Stream Subset Converter IP 核配置

配置如图 4-20 所示，目标检测的结果为 AXI4-Stream Subset Converter IP 核的输入，数据为 4 字节。输出至 AXI4-Stream to Video Out IP 核的数据为 24 字节。故在 IP 核的配置中需将 tdata[15:8]，tdata[23:16]和 tdata[7:0]重复一遍。

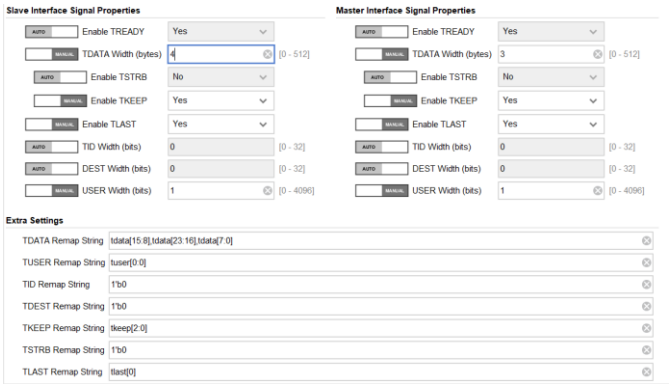


图 4-20 AXI4-Stream Subset Converter IP 配置

（3）DMA IP 核配置

AXI-DMA 用于在 DDR 和 FPGA 之间传输卷积加速器所需的计算参数。为了增强数据传输效率，采用 HP 接口，同时启用 DMA 中断功能以优化通信流程。其配置如图 4-21 所示，将 DMA 缓冲区长度寄存器的宽度设为 26 位，地址空间的宽度为 32 位。由于需要双向传输数据，启用 DMA 的读写双通道。突发传输长度设为 256 以提升传输效率，流数据及内存映射数据的宽度均设为 64 位以确保数据传输过程的一致性。

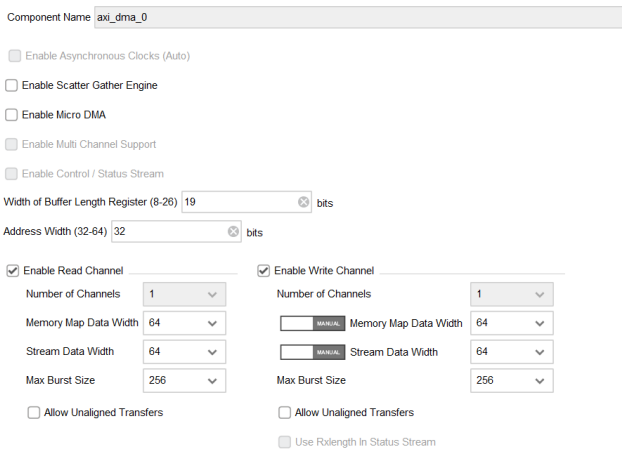


图 4-21 DMA IP 配置

完成 IP 核参数配置后，先手动连接主要信号线，然后用 Vivado 的自动连线完成其余连接，如图 4-22 所示。OV5640 摄像头采集的视频数据首先经过 Video In to AXI-Stream 转换为 AXI 流格式。随后，使用 AXI-Stream Broadcaster IP 核将这一视频流数据分成两个独立的路径进行传输，一路去进行视频回显，一路进行

缩放交给卷积加速器推理。交给视频回显的那一路，先后通过 Hls_rect_0 和 Hls_rect_1 两个 IP 核进行绘制边框和类别，然后经过 AXI VDMA_0 IP 核以多缓存的方式缓存到外部存储器 DDR，缓存出来的数据通过 AXI4-Stream to Video Out IP 核最终传输给 HDMI 接口进行输出。交给卷积加速器推理的那一路，先通过一个 AXI-Stream Data FIFO IP 核以提供缓存区来支持多个数据流的传输，再经过 Hls_preprocess_0 进行图像缩放后进入 AXI VDMA_1 缓存至 PS 端的 DDR。卷积加速器由硬核 CPU 控制，通过 AXI DMA_0 和 VDMA 分别从内存里获取权重和偏置以及输入特征图进行目标检测神经网络模型的推理，推理结果经 CPU 处理后，与 AXI VDMA_0 输出的视频流通过 Video Mixer 输出至 HDMI 显示器。其中，ZYNQ IP 核通过 AXI Interconnect 与 DMA、卷积加速器和 VDMA IP 核等连接，以实现对它们的控制，确保系统功能集成和性能优化。VDMA 输出的视频流数据受到 Video Timing Controller 的时序控制，而 Video Timing Generator 负责产生所需的输出视频时序。

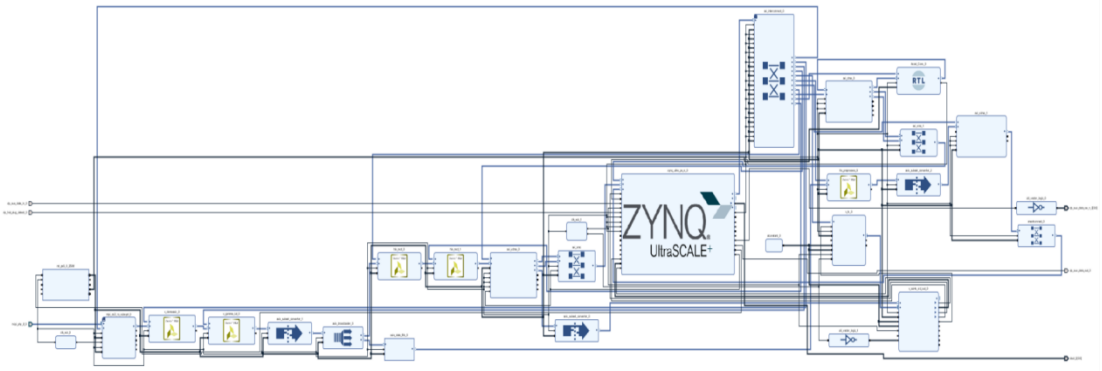


图 4-22 目标检测系统 Block Design

在后续的 PS 端实现中，必须为每个 IP 核指定独立的内存地址，以便于进行配置和管理。因此，在硬件设计阶段，分配给每个 IP 核适当的地址空间是必要的，确保它们能被系统有效识别和访问。如图 4-23 所示为 IP 地址的分配。

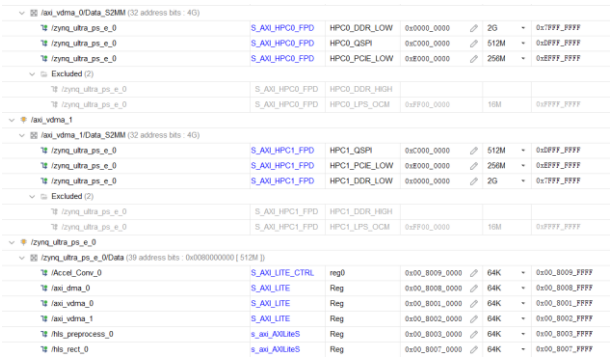


图 4-23 IP 地址分配

构建完硬件系统后，便可进行综合、实现以及生成 bit 流文件，然后导入到 Vitis 开发环境中，进行相应的软件部分开发。

4.4 PS 端实现部分

PS 端的主要内容是在 Vitis 平台编写软件驱动程序。PS 端主要实现 OV5640 摄像头的配置、VDMA 的配置以及 HDMI 接口配置。

4.4.1 OV5640 摄像头的配置

在软件系统设计中，通过编译 C 语言来对 OV5640 寄存器进行配置。对于摄像头图像采集，主要配置包括将分辨率设置为 1280×720，帧率调整为 60FPS，并指定图像的输出格式为 RGB565。摄像头的寄存器配置如表 4-2 所示。

表 4-2 OV5640 寄存器配置

地址	寄存器名称	数值	描述
0x3008	控制寄存器	0x02	Bit[7]：软件复位 Bit[6]：软件电源休眠
0x3808	水平时序寄存器	0x05	Bit[3:0]：水平宽度高 4 位
0x3809	水平时序寄存器	0x00	Bit[7:0]：水平宽度低 8 位
0x380A	垂直时序寄存器	0x02	Bit[2:0]：垂直高度高 3 位
0x380B	垂直时序寄存器	0xd0	Bit[7:0]：垂直高度低 8 位
0x380C	水平总尺寸寄存器	0x07	Bit[3:0]：总水平尺寸高 4 位
0x380D	水平总尺寸寄存器	0x64	Bit[7:0]：总水平尺寸低 8 位
0x380E	垂直总尺寸寄存器	0x02	Bit[7:0]：总垂直尺寸高 8 位
0x380F	垂直总尺寸寄存器	0xe4	Bit[7:0]：总垂直尺寸低 8 位
0x4300	数据格式寄存器	0x61	Bit[7:4]：数据输出格式 Bit[3:0]输出顺序
0x3035	PLL 控制寄存器 1	0x11	Bit[7:4]：时钟分频器 Bit[3:0]：MIPI 的比例分频器
0x3036	PLL 控制寄存器 2	0xd2	Bit[7:0]：PLL 倍频

地址为 0x3808-0x380B 的寄存器用于配置输出图像的水平和垂直方向的像素点数，1280×720 的分辨率用十六进制表示为 0500×02d0，将其高位和低位分别配置到相应的寄存器中。地址为 0x380C-0x380F 的寄存器将总水平尺寸和总垂直尺寸分别配置为 1892（0x0764）和 740（0x02e4）。地址为 0x4300 的寄存器用于配置数据的格式和输出顺序，其中的高四位表示数据输出格式，0x61=0110 0001，高四位是 6 对应的是 RGB565 格式，低四位为 1 对应的是{r[4:0]，g[5:3]}，{g[2:0]，b[4:0]}的顺序；地址为 0x3036 的寄存器用于配置锁相环乘法器，当取值范围在 4-

127 范围内的时候可取任意整数，在 128-252 范围内则只能是偶数，在这里 0xd2 转为十进制是 210，800K×210=168MHz，正好是 720P 分辨率下所需要的像素时钟大小。

在配置 OV5640 摄像头时，首先初始化 ZYNQ 的 EMIO，然后配置其 SCCB 端口为输出模式，同时启用该输出功能，从而使得 SCL 和 SDA 为高电平，以实现 SCCB 通信的基本操作。在 OV5640 上电后，需要先等待一段时间使其初始化，接下来，通过读取特定寄存器来确认设备 ID。一旦确认 ID 无误，则寄存器初始化为默认值，然后将设备从休眠调至工作模式。最后，通过编写表 4-2 的寄存器值来完成对摄像头的配置。

4.4.2 VDMA 的配置

VDMA IP 核设置完成后，还需要在 Vitis 上通过 AXI-Lite 接口配置如表 4-3 所示的寄存器，以便于对 VDMA IP 核的读写操作进行动态控制。

表 4-3 VDMA IP 内部寄存器

名称	偏移地址	描述
MM2S_VDMACR	00h	MM2S VDMA 控制寄存器
MM2S_VDMASR	04h	MM2S VDMA 状态寄存器
MM2S_REG_INDEX	14h	MM2S 寄存器索引
PARK_PRT_REG	28h	MM2S 和 S2MM Park 指针寄存器
VDMA_VERSION	2Ch	VDMA 版本寄存器
S2MM_VDMACR	30h	S2MM VDMA 控制寄存器
S2MM_VDMASR	34h	S2MM VDMA 状态寄存器
S2MM_REG_INDEX	44h	S2MM 寄存器索引
S2MM_VDMA_IRQ_MASK	3Ch	S2MM 错误中断掩码寄存器
MM2S_VSIZE	50h	MM2S 垂直方向显示大小寄存器
MM2S_HSIZE	54h	MM2S 水平方向显示大小寄存器
MM2S_FRMDLY_STRIDE	58h	MM2S 帧延迟和跨度寄存器
MM2S_START_ADDRESS(1~16)	5Ch~98h	MM2S 帧存起始地址（1~16）
S2MM_VSIZE	A0h	S2MM 垂直方向显示大小寄存器
S2MM_HSIZE	A4h	S2MM 水平方向显示大小寄存器
S2MM_FRMDLY_STRIDE	A8h	S2MM 帧延迟和跨度寄存器
S2MM_START_ADDRESS(1~16)	ACh~E8h	S2MM 帧存起始地址（1~16）

VDMA 的配置可以通过两种方法实现：其一是直接对寄存器进行操作，其二是利用库函数。鉴于前者的过程较为复杂，本文采用“run_vdma_frame_buffer”函数来配置 VDMA，包括帧缓存地址、读通道以及启动读通道等配置。在本系统中，通过该函数对 VDMA_0 和 VDMA_1 进行独立的配置。在主函数里，为 VDMA 设

置了不同的帧缓存起始地址，这些地址通过全局变量传递给函数。如图 4-24 所示为 VDMA 的配置。

```
int run_vdma_frame_buffer(XAxiVdma* InstancePtr, int DeviceId, int hsize,
        int vsize, int buf_base_addr, int number_frame_count,
        int enable_frm_cnt_intr,vdma_run_mode mode)
{
    int Status,i;
    XAxiVdma_Config *Config;
    XAxiVdma_FrameCounter FrameCfgPtr;

    /* This is one time initialization of state machine context.
     * In first call it will be done for all VDMA instances in the system.
     */
    if(context_init==0) {
        for(i=0; i < XPAR_XAXIVDMA_NUM_INSTANCES; i++) {
            vdma_context[i].InstancePtr = NULL;
            vdma_context[i].device_id = -1;
            vdma_context[i].hsize = 0;
            vdma_context[i].vsize = 0;
            vdma_context[i].init_done = 0;
            vdma_context[i].buffer_address = 0;
            vdma_context[i].enable_frm_cnt_intr = 0;
            vdma_context[i].number_of_frame_count = 0;

        }
        context_init = 1;
    }
```

图 4-24 VDMA 的配置

4.4.3 HDMI 的配置

HDMI 的 PS 端配置主要是配置 Dynamic Clock Generator IP 核以生成符合要求的时钟，从而给 DVI_Transmitter IP 核使用。其配置代码如图 4-25 所示。此外，本文通过“clk_wiz_cfg”函数对时钟 IP 核进行配置，通过“DisplayInitialize”函数对显示相关的 IP 核进行初始化；“DisplaySetMode”函数用来设置 VTC 输出的分辨率；“DisplayStart”函数用来启动 VTC 开始工作。

```
     * Enable the dynamically generated clock
    */
    ClkStop(dispPtr->dynClkAddr);
    ClkStart(dispPtr->dynClkAddr);

    /*
     * Configure the vtc core with the display mode timing parameters
     */
    vtcTiming.HActiveVideo = dispPtr->vMode.width;  /**< Horizontal Active Video Size */
    vtcTiming.HFrontPorch = dispPtr->vMode.hps - dispPtr->vMode.width;  /**< Horizontal Front Porch Size */
    vtcTiming.HSyncWidth = dispPtr->vMode.hpe - dispPtr->vMode.hps;     /**< Horizontal Sync Width */
    vtcTiming.HBackPorch = dispPtr->vMode.hmax - dispPtr->vMode.hpe + 1;        /**< Horizontal Back Porch Size */
    vtcTiming.HSyncPolarity = dispPtr->vMode.hpol;  /**< Horizontal Sync Polarity */
    vtcTiming.VActiveVideo = dispPtr->vMode.height; /**< Vertical Active Video Size */
    vtcTiming.V0FrontPorch = dispPtr->vMode.vps - dispPtr->vMode.height;    /**< Vertical Front Porch Size */
    vtcTiming.V0SyncWidth = dispPtr->vMode.vpe - dispPtr->vMode.vps;    /**< Vertical Sync Width */
    vtcTiming.V0BackPorch = dispPtr->vMode.vmax - dispPtr->vMode.vpe + 1;;  /**< Horizontal Back Porch Size */
    vtcTiming.V1FrontPorch = dispPtr->vMode.vps - dispPtr->vMode.height;    /**< Vertical Front Porch Size */
    vtcTiming.V1SyncWidth = dispPtr->vMode.vpe - dispPtr->vMode.vps;    /**< Vertical Sync Width */
    vtcTiming.V1BackPorch = dispPtr->vMode.vmax - dispPtr->vMode.vpe + 1;;  /**< Horizontal Back Porch Size */
    vtcTiming.VSyncPolarity = dispPtr->vMode.vpol;  /**< Vertical Sync Polarity */
    vtcTiming.Interlaced = 0;       /**< Interlaced / Progressive video */
```

图 4-25 HDMI 配置部分代码

4.4.4 嵌入式 Linux 系统搭建

为了最大化开发板上硬核 CPU 的使用效率，并实现对加速器的有效控制，故在 ARM 上构建了 Linux 系统。PetaLinux 是 Xilinx 提供的嵌入式 Linux 开发环境，专为其 FPGA 和 SoC 平台设计[51]。其包含预配置的 Linux 内核、根文件系统及应用示例，旨在简化 Linux 系统的配置、构建和部署过程。Petalinux 系统设计流程如图 4-26 所示。

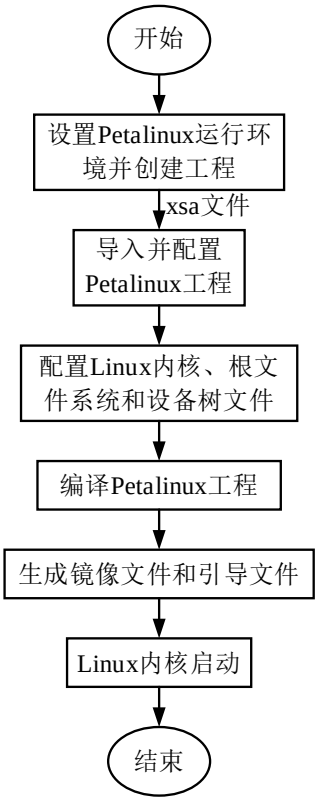


图 4-26 Petalinux 设计流程图

本文在 Ubuntu18.04 操作系统下安装了 Petalinux2019.2 开发工具来构建适合目标检测系统的嵌入式 Linux 环境，具体步骤如下：

（1）设置 Petalinux 运行环境并创建工程

首先在 Linux 终端输入命令“sptl”即可完成 Petalinux 运行环境的配置，然后再运行“petalinux-create -t project”完成工程的创建。

（2）配置 Petalinux 工程

将之前 4.3.6 节生成的 xsa 文件导入到工程中，并输入命令“petalinux-config --get-hw-description”，Petalinux 工具会解析文件并启动如图 4-27 所示的窗口。

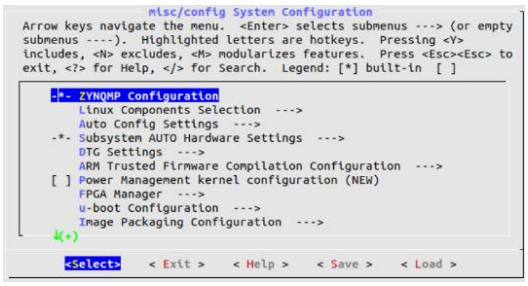


图 4-27 Petalinux 工程配置

（3）Linux 内核配置

在终端运行命令“petalinux-config -c kernel”，系统会在终端中打开如图 4-28 所

示的配置界面。

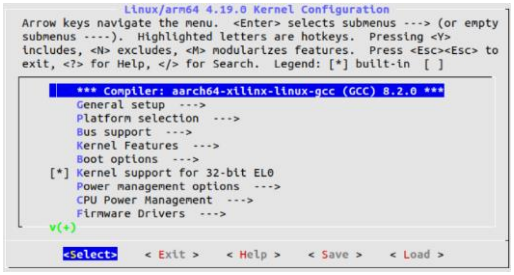


图 4-28 Linux 内核配置

（4）Linux 根文件系统和设备树配置

通过在终端执行命令“petalinux-config -c rootfs”，可以启动图 4-29 所示的根文件系统配置过程。虽然默认设置已足够满足正常需求，用户仍然可以根据特定需求进行个性化配置。本文将继续使用默认配置。

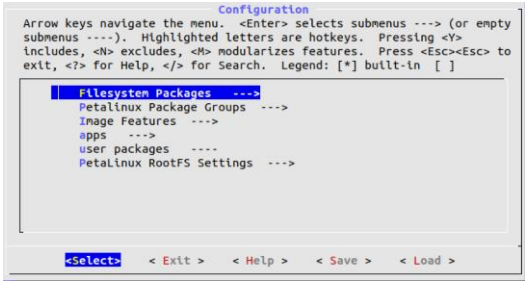


图 4-29 根文件系统配置

完成上述配置后，通过在终端执行“petalinux-build”命令，可以生成设备树 DTB 文件、fsbl 文件、U-Boot 文件，Linux 内核和根文件系统映像等。Petalinux 工程编译过程如图 4-30 所示。

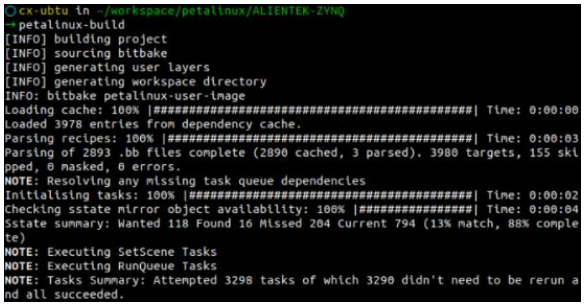


图 4-30 Petalinux 工程编译

（5）BOOT.BIN 启动文件制作

PetaLinux 中的“petalinux-package”指令允许用户将项目封装成适于部署的形式，通过执行“petalinux-package --boot”可以创建一个如图 4-31 所示的可引导的映像文件。在本文中，该文件格式被指定为 BOOT.BIN，其支持通过 SD 卡进行系统启动。

```
cx-ubtu in ~/workspace/petalinux/ALIENTEK-ZYNQ
→ petalinux-package --boot --fsbl --fpga --u-boot --force
INFO: File in BOOT BIN: "/home/cx/workspace/petalinux/ALIENTEK-ZYNQ/images/linux/zynqmp_fsbl.elf"
INFO: File in BOOT BIN: "/home/cx/workspace/petalinux/ALIENTEK-ZYNQ/images/linux/pmufw.elf"
INFO: File in BOOT BIN: "/home/cx/workspace/petalinux/ALIENTEK-ZYNQ/project-spec/hw-description/e
g_base.bit"
INFO: File in BOOT BIN: "/home/cx/workspace/petalinux/ALIENTEK-ZYNQ/images/linux/bl31.elf"
INFO: File in BOOT BIN: "/home/cx/workspace/petalinux/ALIENTEK-ZYNQ/images/linux/u-boot.elf"
INFO: Generating zynqmp binary package BOOT.BIN...


****** Xilinx Bootgen v2019.2
  **** Build date : Oct 23 2019-22:59:42
    ** Copyright 1986-2019 Xilinx, Inc. All Rights Reserved.

INFO: Binary is ready.
cx-ubtu in ~/workspace/petalinux/ALIENTEK-ZYNQ
```

图 4-31 BOOT.BIN 启动文件

生成启动文件之后，为了通过 SD 卡启动 Linux 系统，将 SD 卡空间划分为用于存储启动镜像文件的 FAT32 文件系统区和用于放置根文件系统的 EXT4 文件系统区。在文件系统挂载完后，需要将 BOOT.BIN 和 image.ub 文件复制到指定分区。最后，把 SD 卡插入到开发板的卡槽内，这样就完成了嵌入式 Linux 系统的搭建。

4.5 系统调试与结果分析

4.5.1 系统调试

系统调试流程如图 4-32 所示，本文首先在 Pytorch 框架中使用头盔和口罩的数据集对改进的 YOLOv3-tiny 模型进行 8 位量化训练，以获得训练后的权重参数。随后，对这些数据进行预处理，并将其保存至 SD 卡中。在 PS 端，通过使用内置的 xilff.h 库来实现 SD 卡的读取功能，从而加载 SD 卡上的二进制权重文件.bin，将权重存储到 DDR 内存中。当 PL 端和 PS 端的配置完成之后，便可在开发板上连接相应的外设并上电，然后在 Vitis 软件平台上对工程进行编译，将生成的 elf 文件和 bit 流文件传输至开发板。通过串口工具，执行打印操作以输出相关信息，进而对结果进行观察和分析。

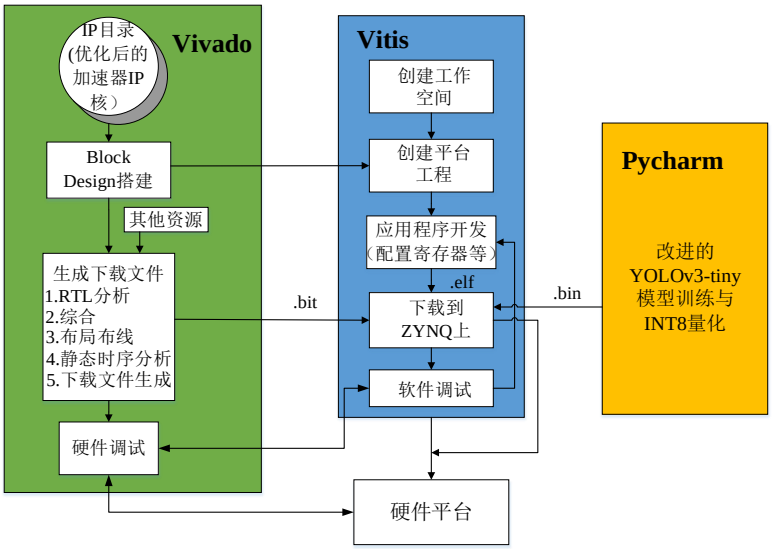
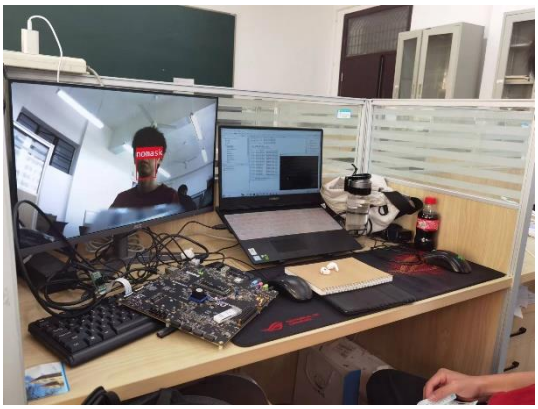
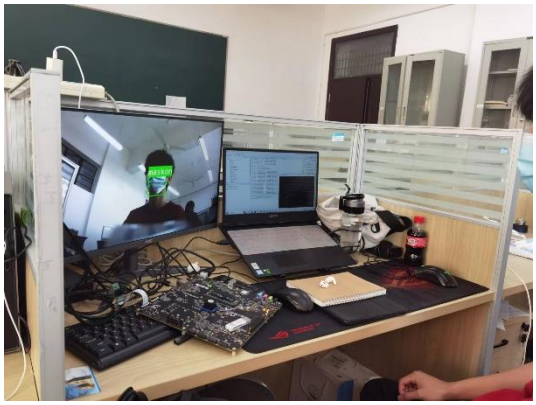


图 4-32 目标检测系统调试流程

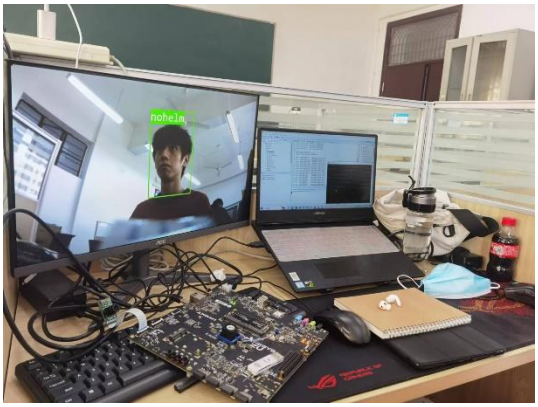
本文使用数据集中的 50 张图片对目标检测系统进行测试，统计每次推理过程的时间和精确度，并求出它们的平均值得到每张图片的处理时间和平均精度。如图 4-33 所示为目标检测系统硬件平台的检测效果图。结果表明，该系统有效实现了对口罩和头盔的识别功能。系统整体准确率达到了 70.1%，且能以 18FPS 的速度处理视频流，基本符合实时处理的需求。



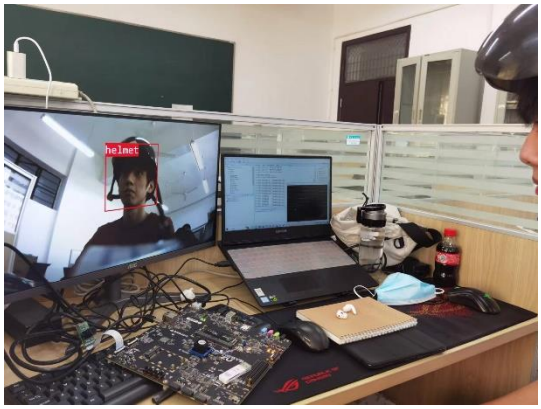
（a）未佩戴口罩时



（b）佩戴口罩时



（c）未佩戴头盔时



（d）佩戴头盔时

图 4-33 目标检测系统检测效果图

4.5.2 资源消耗分析

本文目标检测系统的资源消耗如图 4-34 所示。由于本文设计中大量使用了移位寄存器和加减移位操作，因此 LUT 占比较大，达到可用 LUT 资源的 79%，总计 55501 个。移位寄存器和普通寄存器的合成占用了 54282 个 FF 资源。乘法运算模块使用 300 个 DSP 资源。所有缓存模块通过参数配置来实现，并指定使用 BRAM 进行合成，以减少 LUT 和 FF 资源的使用，共计消耗了 92.5 个 BRAM。考虑到在 FPGA 设计时，除了需要尽可能地利用板上资源外，还应避免资源利用出现几乎满载的情况。总体来看，本文的目标检测系统设计的资源利用是较为合理的。

Resource	Utilization	Available	Utilization %
LUT	55501	70560	78.66
LUTRAM	17445	28800	60.57
FF	54282	141120	38.47
BRAM	92.50	216	42.82
DSP	300	360	83.33
IO	11	252	4.37
BUFG	7	196	3.57
MMCM	2	3	66.67
PLL	1	6	16.67

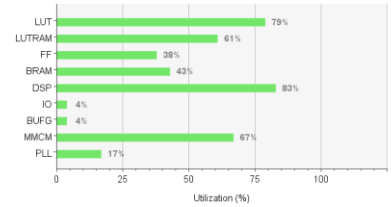


图 4-34 资源消耗情况

4.5.3 功耗分析

本文的目标检测系统功耗如图 4-35 所示。片上总功耗为 4.457W。其中，动态功耗为 4.123W，占比最高的动态功耗为 PS 部分的 61%，这可能因为 PS 端的处理器部分消耗了较大功耗。静态功耗为 0.334W，占总功耗的 7%。

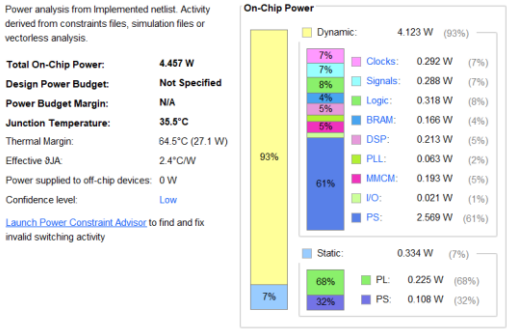


图 4-35 功耗情况

4.5.4 时序分析

图 4-36 为本系统的时序分析报告。建立时间的时序约束能确保数据在被下一级触发器捕获之前保持稳定，从而维护数据完整性和防止时序错误，其最坏路径的时序裕量为 0.041ns。保持时间的时序约束确保数据在触发器内保持足够长的时间，以正确地被下一个时钟周期捕获，其最坏路径时序裕量为 0.01ns。脉冲宽度约束确保信号保持在特定状态下的最小持续时间，从而保障电路的正确功能和性能，该约束的宽度裕量为 0.145ns。综上，所有约束均满足收敛条件，数据在电路中能被准确采样，避免了违例的情况。

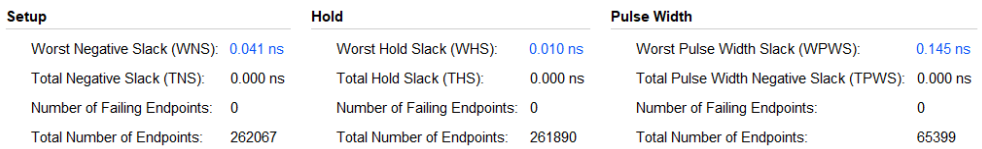


图 4-36 时序报告

4.5.5 性能对比

本文通过将卷积加速器的性能与 FPGA 上现有的工作进行对比来验证其优势。从表 4-5 中可以看出，本文的加速器的能效比和推理时间均优于[47]、[52]、[53]和[54]。这意味着本设计采用的基于列缓存的行缓存结构、乒乓和全流水结构以及并行度优化设计等优化策略大大降低了延迟并提高了吞吐率。对比其他文献，能效比依次提高了 1.5 倍、10.3 倍、1.4 倍和 4.8 倍，处理时间依次降低了 4.9 倍、1.9 倍、1.2 倍和 8.7 倍。虽然文献[47]、[53]和[54]的 DSP 使用量低于本文，但吞吐率也低于本文，通过计算可得本文的计算密度高于其他文献。与文献[47]、[53]和[54]中的计算密度相比，依次提高了 1.01 倍、1.08 倍和 3.15 倍。综上所述，本文设计的加速器具备低延迟，高吞吐率和高计算密度的特性，即便是在资源和功耗受限的情况下依然能够实现较高的处理性能，从而实现卷积加速的效果。

表 4-5 与其他 FPGA 的性能对比

	文献[47]	文献[52]	文献[53]	文献[54]	本文
硬件平台	Zedboard	ZCU102	Ultra96 V2	Zedboard	ZU3EG
网络模型	YOLOv3-tiny	YOLOv3	YOLOv3-tiny	YOLOv3-tiny	改进的 YOLOv3-tiny
输入尺寸	416×416	848×565	416×416	416×416	1280×720
频率（MHz）	100	325	250	100	250
量化精度（bit）	16	8	8	16	8
准确率	58.4%	92%	75%	33.1%	70.1%
DSP 使用量	181	—	242	160	300
功率（W）	3.36	17.78	4.26	3.36	4.49
吞吐率（GOPS）	24.32	28.48	31.50	10.45	81.11
能效比（GOPS/W）	7.23	1.60	7.40	3.11	18.06
计算密度（GOPS/DSP）	0.134	—	0.130	0.065	0.270
FPS	3.1	6.2	8.2	1.9	18.2
处理时间（s）	0.325	0.161	0.121	0.532	0.055

4.6 本章小结

本章主要是介绍基于 ZYNQ 目标检测系统的实现。首先提出了目标检测系统

的整体方案。其次简单介绍了所使用的开发板和摄像头。然后介绍了 PL 端的实现方案，包括 IP 核的封装和参数的配置，系统 Block Design 的搭建以及最后生成 bit 流文件等。接着阐述了利用 Vitis 软件来编译 C 语言来配置 OV5640 摄像头和 VDMA 等寄存器。为了充分发挥硬核 CPU 的调度功能搭建了片上 Linux 系统。最后，进行上板调试验证目标检测系统的功能，并对系统进行资源消耗、功耗、时序分析以及性能分析，与其他的 FPGA 平台的相比，本文在延时，计算密度以及计算性能上均有所提升。

第 5 章 总结与展望

5.1 全文总结

本文主要对深度学习算法在嵌入式平台上的实现进行研究。本设计采用的是FPGA+ARM 架构，并采用层数和参数量较低的 CNN，配合摄像头采集和图像回显的回路，最终构建了一个功耗低、体积小、高性能的实时目标检测系统。该系统通过在 PL 端设计卷积加速器对 CNN 相关计算进行加速，使得整个系统能够满足实际场景的需要。本文完成的工作总结如下：

（1）对国内外关于深度学习目标检测算法和卷积加速器的研究现状进行了探讨和分析，最终选择 YOLOv3-tiny 作为本文目标检测系统的核心算法，同时分析了其计算原理。此外，分析了当前卷积加速器所存在的问题和不足，在此基础上确立了本文的研究目标。

（2）为了进一步适应头盔和口罩的目标检测，本文对 YOLOv3-tiny 网络结构进行了优化，并提出了改进的算法。为加快模型的收敛速度，使用 K-means 聚类算法来获取先验框。此外，在原始的 YOLOv3-tiny 上增加了一系列 3×3 和 1×1 卷积层，以便更好地提取头盔和口罩的特征，从而提高在硬件平台上的检测精度。

（3）提出了一种低延迟、高计算密度和高吞吐率的通用卷积加速器，包括计算单元、缓存单元以及控制单元等。此外，对设计的加速器在行缓存结构、乒乓和全流水结构以及并行度设计等方面进行优化，有效降低了加速器的资源消耗和处理时间。论文完成了对加速器的整体仿真，验证了其功能的正确性。

（4）在 Vivado 中搭建目标检测系统，调用 IP 核并配置其参数以连接各个模块，完成硬件设计。然后在 Vitis 中对 OV5640 摄像头、VDMA 以及 HDMI 进行配置。接着针对硬核 CPU 的调度，搭建嵌入式 Linux 系统。当完成软件配置后进行系统调试，并对系统的功能和性能进行了分析。实验结果显示，本文设计的目标检测系统具有良好的检测性能。与其他硬件平台相比，本系统在延迟和功耗方面都取得了较好的效果。

5.2 工作展望

本文实现了一个基于 ZYNQ 的目标检测系统，系统整体延时和检测性能上相较于其他硬件平台有所提升，但是还是存在一些不足之处：

（1）由于本系统在硬件部分设计时只使用了绘制单个边框和类别的 IP 核，所以在检测头盔和口罩时一次只能识别一个类别和边框，无法同时一次性识别多个目标。在今后的研究中可以自行设计一种用于绘制多个边框和类别的 IP 核，从而实现同时检测多个目标的功能。

（2）本文的卷积加速器只能实现 1×1 和 3×3 卷积计算以及 2×2 和 3×3 的池化计算，对于后续更为复杂的神经网络模型在该加速器上还不能实现。在今后的工作中对现有结构进行优化，从而兼容更大尺寸的卷积核，在资源允许的情况下实现更复杂的网络模型。

参考文献

[1] Zou Z, Chen K, Shi Z, et al. Object Detection in 20 Years: A Survey[J]. Proceedings of the IEEE, 2023, 111(3): 257–276.

[2] Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[C]. Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 1-9.

[3] 罗会兰, 陈鸿坤. 基于深度学习的目标检测研究综述[J]. 电子学报, 2020, 48(06): 1230.

[4] 王奥. 基于 FPGA 的嵌入式目标检测系统设计与实现[D]. 西安电子科技大学, 2021.

[5] 宋亚朋. 智能目标检测系统在 FPGA 中的设计与实现[D]. 西安电子科技大学, 2022.

[6] Wu R, Guo X, Du J, et al. Accelerating neural network inference on FPGA-based platforms—A survey[J]. Electronics, 2021, 10(9): 1025.

[7] 陈科圻, 朱志亮, 邓小明, 等. 多尺度目标检测的深度学习研究综述[J]. 软件学报, 2021, 32(04): 1201-1227.

[8] Girshick R, Donahue J, Darrell T, et al. Rich feature hierarchies for accurate object detection and semantic segmentation[C]. Proceedings of the IEEE conference on computer vision and pattern recognition. 2014: 580-587.

[9] He K, Zhang X, Ren S, et al. Spatial pyramid pooling in deep convolutional networks for visual recognition[J]. IEEE transactions on pattern analysis and machine intelligence, 2015, 37(9): 1904-1916.

[10] Girshick R. Fast r-cnn[C]. Proceedings of the IEEE international conference on computer vision. 2015: 1440-1448.

[11] Ren S, He K, Girshick R, et al. Faster R-CNN: Towards real-time object detection with region proposal networks[J]. IEEE transactions on pattern analysis and machine intelligence, 2016, 39(6): 1137-1149.

[12] Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection[C]. Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 779-788.

[13] Redmon J, Farhadi A. YOLO9000: better, faster, stronger[C]. Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 7263-7271.

[14] Redmon J, Farhadi A. Yolov3: An incremental improvement[J]. arXiv preprint arXiv:1804.02767, 2018.

[15] Wang T, Gong L, Wang C, et al. Via: A novel vision-transformer accelerator based on FPGA[J]. IEEE transactions on computer-aided design of integrated circuits and systems, 2022, 41(11): 4088-4099.

[16] Li H, Fan X, Jiao L, et al. A high performance FPGA-based accelerator for large-scale convolutional neural networks[C]. 2016 26th international conference on field programmable logic and applications (FPL). IEEE, 2016: 1-9.

[17] Zhang X, Wang J, Zhu C, et al. DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs[C]. 2018 IEEE/ACM international conference on computer-aided design (ICCAD). IEEE, 2018: 1-8.

[18] Vo Q H, Linh Le N, Asim F, et al. A deep learning accelerator based on a streaming architecture for binary neural networks[J]. IEEE access, 2022, 10: 21141–21159.

[19] Nguyen D T, Nguyen T N, Kim H, et al. A high-throughput and power-efficient FPGA implementation of YOLO CNN for object detection[J]. IEEE transactions on very large scale integration (VLSI) systems, 2019, 27(8): 1861-1873.

[20] Zhang C, Li P, Sun G, et al. Optimizing FPGA-based accelerator design for deep convolutional neural networks[C]. Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays. 2015: 161-170.

[21] Chen Y H, Emer J, Sze V. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks[J]. ACM SIGARCH computer architecture news, 2016, 44(3): 367-379.

[22] Qiu J, Wang J, Yao S, et al. Going deeper with embedded FPGA platform for convolutional neural network[C]. Proceedings of the 2016 ACM/SIGDA international symposium on field-programmable gate arrays. 2016: 26-35.

[23] Guo K, Sui L, Qiu J, et al. Angel-eye: A complete design flow for mapping CNN onto embedded FPGA[J]. IEEE transactions on computer-aided design of integrated circuits and systems, 2017, 37(1): 35-47.

[24] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks[J]. Communications of the ACM, 2017, 60(6): 84-90.

[25] Glorot X, Bengio Y. Understanding the difficulty of training deep feedforward neural networks[C]. Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, 2010: 249-256.

[26] Liu F, Zhang B, Chen G, et al. A novel configurable high-precision and low-cost circuit design of sigmoid and tanh activation function[C]. 2021 IEEE international conference on integrated circuits, technologies and applications (ICTA). IEEE, 2021: 222-223.

[27] Parhi R, Nowak R D. The role of neural network activation functions[J]. IEEE signal processing letters, 2020, 27: 1779-1783.

[28] Mastromichalakis S. ALReLU: A different approach on Leaky ReLU activation function to improve neural networks performance[J]. arXiv preprint arXiv:2012.07564, 2020.

[29] Liu J J, Hou Q, Cheng M M, et al. A simple pooling-based design for real-time salient object detection[C]. Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019: 3917-3926.

[30] 张印辉, 张朋程, 何自芬, 等. 红外行人目标精细尺度嵌入轻量化实时检测[J]. 光子学报, 2022, 51(09): 266-276.

[31] Jiang B, Luo R, Mao J, et al. Acquisition of localization confidence for accurate object detection[C]. Proceedings of the European conference on computer vision (ECCV). 2018: 784-799.

[32] Ghaffari S, Sharifian S. FPGA-based convolutional neural network accelerator design using high level synthesize[C]. 2016 2nd international conference of signal processing and intelligent systems (ICSPIS). IEEE, 2016: 1-6.

[33] 张鹏飞. 基于 FPGA 的卷积神经网络异构加速器研究与实现[D]. 南昌大学, 2023.

[34] 陈辰. 基于 FPGA 的神经网络加速器的规模可伸缩性研究[D]. 江南大学, 2019.

[35] Ma Y, Cao Y, Vrudhula S, et al. Optimizing the convolution operation to accelerate deep neural networks on FPGA[J]. IEEE transactions on very large scale integration (VLSI) systems, 2018, 26(7): 1354-1367.

[36] Likas A, Vlassis N, Verbeek J J. The global k-means clustering algorithm[J]. Pattern recognition, 2003, 36(2): 451-461.

[37] 朱联祥, 徐莉娟. 基于改进 YOLOv3-tiny 的车辆目标检测[J]. 信息技术与信息化, 2022, 47(03): 9-12.

[38] Ammous D, Chabbouh A, Edhib A, et al. Improved YOLOv3-tiny for silhouette detection using regularisation techniques[J]. Int. Arab J. Inf. Technol., 2023, 20(2): 270-281.

[39] Jacob B, Kligys S, Chen B, et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference[C]. Proceedings of the IEEE conference on computer vision and pattern recognition. 2018: 2704-2713.

[40] 李岑. 基于 FPGA 的 YOLO CNN 实时目标检测系统研究与实现[D]. 上海交通大学, 2020.

[41] 李放, 曹健, 李普, 等. 基于 ARM+FPGA 异构平台的目标检测加速模块设计与实现[J]. 北京大学学报(自然科学版), 2022, 58(06): 1035-1041.

[42] Gholamalinezhad H, Khosravi H. Pooling methods in deep neural networks, a review[J]. arXiv preprint arXiv:2009.07485, 2020.

[43] Molina A, Rajamani K, Azadet K. Digital predistortion using lookup tables with linear interpolation and extrapolation: Direct least squares coefficient adaptation[J]. IEEE transactions on microwave theory and techniques, 2016, 65(3): 980-987.

[44] Ntaryamira E, Maxim C, Niyonsaba T, et al. An efficient FIFO buffer management to ensure task level and effect-chain level data properties[C]. 2020 IEEE international conference on embedded software and systems (ICESS). IEEE, 2020: 1-8.

[45] 张鑫宇. 基于 ZYNQ 的目标检测硬件加速系统研究与实现[D]. 中北大学, 2022.

[46] Bao C, Xie T, Feng W, et al. A power-efficient optimizing framework FPGA accelerator based on Winograd for YOLO[J]. IEEE access, 2020, 8: 94307-94317.

[47] Zhang H, Jiang J, Fu Y, et al. Yolov3-tiny object detection SoC based on FPGA platform[C]. 2021 6th international conference on integrated circuits and microsystems (ICICM). IEEE, 2021: 291-294.

[48] Ungurean I, Gaitan N C. Software architecture of a fog computing node for industrial internet of things[J]. Sensors, 2021, 21(11): 3715.

[49] 裴正雄, 彭安金. 基于 FPGA 和 OV5640 的图像采集和处理系统设计[J]. 机电信息, 2019, 19(32): 159-160.

[50] Yan F, Zhao S, Venegas-Andraca S E, et al. Implementing bilinear interpolation with quantum images[J]. Digital signal processing, 2021, 31(7): 103149.

[51] Almeida T B D , Pedrino E C , Fernandes M M .Complex morphological filtering for serial, parallel, GPU, SoC, PetaLinux and FPGA execution[J].IEEE Latin America transactions, 2020, 18(10): 1675-1682.

[52] Chan C W H, Leong P H W, So H K H. Vision guided crop detection in field robots

using FPGA-based reconfigurable computers[C]. 2020 IEEE international symposium on circuits and systems (ISCAS). IEEE, 2020: 1-5.

[53] Adiono T, Putra A, Sutisna N, et al. Low latency YOLOv3-tiny accelerator for low-cost FPGA using general matrix multiplication principle[J]. IEEE access, 2021, 9: 141890-141913.

[54] Yu Z, Bouganis C S. A parameterisable FPGA-tailored architecture for YOLOv3-tiny[C]. Applied reconfigurable computing. architectures, tools, and applications: 16th international symposium, ARC 2020, Toledo, Spain, April 1–3, 2020, Proceedings 16. Springer international publishing, 2020: 330-344.

攻读硕士期间取得的学术成果

发表论文：

[1] **Xianbin Zhang**, Xiaoqiang Zhang, Xinxing Zheng, et al. Design of Hardware Accelerator Architecture for Target Detection Based on ZYNQ[C]. 2023 3rd International Conference on Electronic Information Engineering and Computer Science (EIECS). IEEE, 2023: 1385-1390.

参与项目：

[1] 纵向项目，安徽省车载显示集成系统工程研究中心开放基金项目（工程类），“高性能无线视频安全传输芯片关键技术研究及产业化”，2023.12~2025.11.

科研竞赛：

“海云捷迅杯—基于 FPGA CNN 加速器 SSD_MobileNetV1 模型目标检测实现”获 2022 年全国大学生集成电路创新创业大赛华中赛区三等奖

致谢

日月如梭，岁月如歌。转眼间，三年的研究生生活即将结束，在此，我想由衷的感谢我的母校、我的父母、我的导师、我的室友、我的同门以及我的朋友。感谢我亲爱的家人对于我三年学习的默默支持和帮助；感谢我母校安徽工程大学给了我三年不断学习的机会；感谢电气工程学院的老师和同学三年来对我的帮助和关心。感谢我的室友和同门在研究生三年可以与我分享快乐和人生经验。

此外，在这段时间内，我最感谢的人当然是我的导师张肖强老师。他那种认真负责的工作态度，严谨的治学精神和深厚的理论水平使我受益匪浅。无论是在理论上还是在实践上，他都给了我巨大的帮助，使我得到了不少的提高。从论文开题到完成，张老师一直都是很悉心的给我讲解着课题中遇到的各种问题，循循善诱，严格把关，帮助我开拓思路，并不断地鼓舞着我，使我感到信心倍增，让我非常积极地投入到课题中去，不断地完成毕业论文中的一个个部分。在此，我再次感谢张老师在毕业论文上不断给予我的帮助，让我充分感受到了自己对学习的兴趣和热情，使我能够圆满地完成自己的毕业论文。

最后，我要感谢参与我论文评审和答辩的各位老师能够在百忙之中抽出宝贵的时间，对我的毕业论文进行评阅和检查。感谢他们给我所提出的建议和意见，这将会对我今后的发展提供巨大动力，他们对我的帮助是一笔无价的财富。今后，我会在学习、生活和工作上加倍努力，取得更多的成果去回报父母、母校、老师和社会!