硕 士 学 位 论 文

MASTER'S DISSERTATION

（专业学位）

论文题目　基于 ZYNQ 的可配置卷积神经网络加速器的设计与实现

作者姓名　杨红光

学科专业　电子信息

指导教师　张立国　副教授

2023 年 5 月

中图分类号：TG166　　学校代码：10216
UDC：621.3　　密级：公开

专业学位硕士学位论文
（应用研究型）

基于 ZYNQ 的可配置卷积神经网络加速器的设计与实现

硕 士 研 究 生：杨红光
导　　师：张立国 副教授
副　导　师：陈毅强 教授级高工
申 请 学 位：电子信息硕士
学 科 专 业：电子信息
所 属 学 院：电气工程学院
答 辩 日 期：2023 年 5 月
授予学位单位：燕山大学

A Dissertation in Electronic and Information Engineering

DESIGN AND IMPLEMENTATION OF A CONFIGURABLE CONVOLUTIONAL NEURAL NETWORK ACCELERATOR BASED ON ZYNQ

By Yang Hongguang

Supervisor: Zhang Liguo

Yanshan University

May, 2023

燕山大学硕士学位论文原创性声明

本人郑重声明：此处所提交的硕士学位论文《基于 ZYNQ 的可配置卷积神经网络加速器的设计与实现》，是本人在导师指导下，在燕山大学攻读硕士学位期间独立进行研究工作所取得的成果。论文中除已注明部分外不包含他人已发表或撰写过的研究成果。对本文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。本声明的法律结果将完全由本人承担。

作者签字：杨红光 日期： 2023 年 5 月 31 日

燕山大学硕士学位论文使用授权书

《基于 ZYNQ 的可配置卷积神经网络加速器的设计与实现》系本人在燕山大学攻读硕士学位期间在导师指导下完成的硕士学位论文。本论文的研究成果归燕山大学所有，本论文的研究内容不得以其它单位的名义发表。本人完全了解燕山大学关于保存、使用学位论文的规定，同意学校保留并向有关部门送交论文的复印件和电子版本，允许论文被查阅和借阅。本人授权燕山大学，可以采用影印、缩印或其它复制手段保存论文，可以公布论文的全部或部分内容。

保密□，在 年解密后适用本授权书。

本学位论文属于

不保密☑。

(请在以上相应方框内打“√”)

作者签名：杨红光 日期：2023 年 5 月 31 日

导师签名：张庆国 日期：2023 年 5 月 31 日

摘 要

针对现阶段卷积神经网络加速器只能应用在单一 FPGA 平台，不支持硬件平台升级迭代的问题，本文提出了一种基于 ZYNQ 的可配置卷积神经网络加速器，既能够搭建多种卷积神经网络，完成边缘推理的任务，又能够通过修改配置参数，调整该加速器硬件资源占用量，实现对当前硬件平台资源的较好利用，主要完成的工作如下：

(1)针对卷积神经网络中数据参数量庞大，无法全部加载到片上缓存空间的问题，该加速器融合了多种数据分块方式，并在数据分块后对比分析不同复用方式下的总传输参数量，提出了一种自适应数据复用策略，以减少数据传输的总参数量。

(2)针对卷积神经网络的快速搭建需求，对卷积神经网络所需参数进行封装，并定义了专用的卷积神经网络指令集，通过调用其指令集，用户可快速地搭建出多种卷积神经网络模型。

(3)为便于在不同硬件平台中应用该加速器，提出了一种软硬件协同配置的方案。在设计过程中将数据位宽、乘累加阵列并行度和中间缓存空间大小作为一种可配置参数，通过选用不同配置方式可调整整体资源使用量，以适配 FPGA 硬件平台。

(4)为使得该加速器在同等吞吐量下具备较低的功耗，提出了一种划分时钟域的解决方案，采用高频时钟+低频时钟结合的方式，将核心运算模块工作在高频时钟域，非核心模块工作在低频时钟域，进一步优化了电路时序。

最终在 Xilinx ZCU104 板卡上进行实验验证，由实验结果可得，当乘累加阵列并行度选择 1024、数据位宽选择 8、核心加速引擎工作在 180 MHz 时，该加速器峰值吞吐量为 180 GOPS，功耗为 3.752 W，能效比达到 47.97 GOPS/W，对于 VGG16 网络，其卷积层的平均乘累加利用率达到 84.37%。这表明该加速器具有较好的可配置性和高效性能，在边缘计算和嵌入式设备中具有广泛的应用前景。

关键词：卷积神经网络；FPGA；硬件加速；软硬件可配置

Abstract

This paper proposes a configurable convolutional neural network (CNN) accelerator based on ZYNQ, which can not only build various CNN models to perform edge inference tasks but also adjust the accelerator's hardware resource usage by modifying configuration parameters to make better use of the current hardware platform. The main contributions of this work are as follows:

(1) To address the issue of large data parameter volume in CNNs that cannot be fully loaded into on-chip cache space, this accelerator integrates multiple data partitioning methods and proposes an adaptive data reuse strategy that reduces the total parameter volume of data transmission by comparing and analyzing different reuse methods.

(2) To meet the demand for fast construction of CNNs, this work encapsulates the required parameters of CNNs and defines a dedicated CNN instruction set. Users can quickly build multiple CNN models by calling this instruction set.

(3) To facilitate the application of this accelerator on different hardware platforms, this paper proposes a software-hardware co-configuration scheme. Data bit width, MAC array parallelism, and intermediate cache space size are considered as configurable parameters, and different configuration methods can be selected to adjust the overall resource usage to adapt to the FPGA hardware platform.

(4) To achieve lower power consumption for the same throughput, a clock domain partitioning solution is proposed. The core computing module works in a high-frequency clock domain, while the non-core module works in a low-frequency clock domain, which further optimizes circuit timing.

Experimental validation is conducted on the Xilinx ZCU104 board. The experimental results show that when the MAC array parallelism is set to 1024, the data bit width is set to 8, and the core acceleration engine operates at 180 MHz, the peak throughput of the accelerator is 180 GOPS, the power consumption is 3.752 W, and the energy efficiency ratio is 47.97 GOPS/W. For the VGG16 network, the average MAC utilization rate of the convolutional layers reaches 84.37%. These results demonstrate that the proposed

accelerator has good configurability and high efficiency performance and has a wide range of applications in edge computing and embedded devices.

Keywords: Convolutional neural network; FPGA; Hardware Acceleration; software-hardware co-configuration

目 录

摘 要 I
Abstract III
第 1 章 绪 论 1
1.1 研究背景及意义 1
1.2 国内外研究现状 3
1.2.1 卷积神经网络的研究现状 3
1.2.2 硬件加速器的研究现状 4
1.3 研究内容及组织结构 5
1.3.1 本文的研究内容 5
1.3.2 本文的组织结构 6
第 2 章 相关理论基础 7
2.1 卷积神经网络基础 7
2.1.1 卷积算子 7
2.1.2 池化算子 9
2.1.3 激活算子 9
2.1.4 全连接算子 11
2.2 基于 ZYNQ 的开发流程 11
2.2.1 FPGA 基础知识 11
2.2.2 ZYNQ 结构 12
2.2.3 FPGA 开发流程 13
2.3 低位宽量化方法 15
2.4 本章小结 16
第 3 章 CNN 加速器关键技术 17
3.1 可配置定点量化 17
3.1.1 量化方法对比与选择 17
3.1.2 量化过程 18
3.1.3 可配置数据位宽设计 19
3.2 硬件电路时序优化 19
3.2.1 内外双重缓存空间 20
3.2.2 核心时钟频率优化 22
3.3 数据分块和数据复用 23
3.3.1 多方向数据分块方法 23

3.3.2 自适应数据复用策略 26
3.4 软硬件协同配置 29
3.5 本章小结 30
第 4 章 CNN 加速器架构设计与仿真验证 31
4.1 CNN 加速器整体架构 31
4.2 软件层驱动设计 32
4.2.1 数据加载 32
4.2.2 CNN 指令集配置 33
4.2.3 运算模块驱动引擎 33
4.3 硬件层电路设计 35
4.3.1 卷积+激活模块电路设计 35
4.3.2 池化模块电路设计 47
4.3.3 数据交互通路设计 48
4.4 硬件层电路仿真 49
4.4.1 卷积+激活模块电路仿真 49
4.4.2 池化模块电路仿真 54
4.4.3 多配置方式的电路仿真 56
4.5 本章小结 57
第 5 章 CNN 加速器板级验证 58
5.1 实验环境 58
5.2 实验过程 59
5.3 板级实验与分析对比 62
5.3.1 资源占用情况分析 62
5.3.2 推理精度对比分析 64
5.3.3 不同配置方式下性能分析 64
5.3.4 乘累加阵列实验分析 65
5.3.5 加速器性能对比分析 66
5.4 本章小结 67
结 论 68
参考文献 70
攻读硕士学位期间承担的科研任务与主要成果 75
致 谢 76

第 1 章 绪　论

1.1 研究背景及意义

随着人工智能和大数据技术的不断进步，卷积神经网络(Convolutional Neural Networks, CNN)在多个应用领域中获得了突破性进展。在语音识别领域，卷积神经网络可将声音信号转换成文本，从而广泛应用于智能语音助手、语音识别输入法等场景；在图像处理领域，卷积神经网络在图像识别、目标检测和人脸识别等方面已经超越了人类，成为计算机视觉领域的重要技术；在视频处理领域，卷积神经网络可实现视频分析、物体跟踪、行为分析等任务，有助于提高视频处理的自动化水平。总的来说，卷积神经网络在各个领域中的应用前景广阔，为实现智能化提供了重要的支持和保障。

然而，随着神经网络模型的复杂性和数据参数量的不断增加，对硬件的计算能力、内存和数据存储等方面的要求也越来越高。因此，需要一种计算能力强、可并行加速、数据吞吐量高的高性能硬件平台来对卷积神经网络进行训练推理。面对各种实际需求和越来越复杂的神经网络结构，多样化的卷积神经网络硬件平台也在不断发展，其中包括通用性芯片、半定制化芯片和全定制化芯片等。

通用性芯片，被设计用来执行各种不同的任务，这些芯片可以执行多种计算任务，如数据处理、图形处理和人工智能等，通用性芯片具有高度的灵活性和可编程性，但它们的性能和功耗可能不如全定制化芯片，常用的通用性芯片包括中央处理器(Central Processing Unit, CPU)和图形处理器(Graphics Processing Unit, GPU)。半定制化芯片介于通用性芯片和全定制化芯片之间，可用来被编程来执行特定的任务，相比于通用性芯片，它们的性能更高、功耗更低，这种芯片在设计完成之后可以重新编程，因此具有一定的灵活性，如可编程逻辑门阵列(Field Programmable Gate Array, FPGA)。全定制化芯片，是一种专用芯片，其被设计用来执行特定的任务，由于这种芯片是对特定的任务进行的优化，所以能够获取高效的性能，但由于它们需要进行精确的设计和制造，所以设计的成本和时间要比通用性芯片和半定制化芯片更高，如专用集成电路(Application-Specific Integrated Circuit, ASIC)。

由于 FPGA 具备低功耗、高灵活性和高并行度的特点，更适于应用在移动场景领域，所以可将神经网络模型移植到 FPGA 平台中，完成卷积神经网络的推理工作。考虑到 FPGA 平台并非一成不变的，随着 FPGA 硬件计算性能的提升，其硬件平台也呈现多样的发展趋势，如 Xilinx 公司旗下的 FPGA 芯片有 ZNYQ-7000、ZYNQ-7020、XCZU3EG 和 XCZU7EG 等。现阶段，面对多样化的 FPGA 芯片及多种 FPGA 硬件平台的性能升级，以往设计的卷积神经网络加速器开始出现了时效性的问题，即随着加速器性能需求的增强，当将以往设计的卷积神经网络加速器应用到新平台时，还需对内部结构重新设计，消耗较多的时间和精力。针对这一问题，需设计一种可跨平台应用的通用加速架构，以利于后期加速器的升级换代，从而紧跟卷积神经网络模型的快速发展。

近年来基于 FPGA 的卷积神经网络加速器的架构设计，按加速灵活性和加速性能可分为三类：专用加速架构、可配置加速架构和异构加速架构。以往研究方向主要集中于专用加速架构，这种方式能够得到最高的计算性能，但彻底牺牲了通用能力；可配置加速架构是在牺牲部分性能的基础上得到了网络模型的可配置能力，从而支持搭建多种卷积神经网络模型，异构加速架构与可配置加速架构相似，不同的是异构加速采用 ARM+FPGA 协同加速的方案，将卷积神经网络模型的搭建和网络层的配置过程放在了 ARM 端，只需修改 ARM 端网络层就能实现各类卷积神经网络的推理工作。

对比分析三种加速架构，可以看出专用加速架构不具备通用能力，而可配置加速架构虽然在卷积神经网络配置过程中具备较高的通用能力，但由于网络层配置过程放在了 FPGA 逻辑端，而 FPGA 逻辑端的电路综合、布局布线等操作过程需要消耗较长的时间。针对可配置架构的缺陷，设计了异构加速的方式，其通过将配置过程放置在 ARM 端，从而解决了 FPGA 逻辑端配置过程时间长的问题，这是因为在 ARM 端只需编译就能生成 FPGA 可执行的指令语言，从而能够快速的调整卷积神经网络结构。但不管异构加速架构还是可配置加速架构都存在支持硬件平台单一的问题，当面对跨平台应用需求，即对老旧的 FPGA 硬件平台升级时，还需对内部结构重新设计，消耗较多的精力。针对这一问题，本文提出了一种可配置卷积神经网络加速器，围绕如何设计一种高度通用性、资源可配置性和高能效比进行展开，最终实现多平台应用的目的。

1.2 国内外研究现状

1.2.1 卷积神经网络的研究现状

LeNet-5[1]是由 Yann LeCun 在 1998 年提出的卷积神经网络模型，该模型最早被应用于数字识别，并在美国银行中得到成功推广，该神经网络的实用化推动了卷积神经网络的井喷式发展。2012 年，Krizhevsky 等人提出的 AlexNet[2]模型在 ImageNet 大规模视觉识别挑战赛(ILSVRC)中获得了 84.6%的识别精度，超过了第二名 10.9 个百分点，成为当时最先进的图像分类模型，AlexNet 采用了小卷积叠加的方式替代了传统的大卷积，首次使用 ReLU 激活函数作为神经网络中的非线性函数，这解决了使用 Sigmoid 激活函数时遇到的梯度消失问题，引发了对神经网络的广泛研究兴趣。

随着人工神经网络的发展，网络结构也在不断演化。为了提高卷积神经网络的识别精度，研究人员主要从以下四个方面展开探索：首先是增加网络的深度，例如 2014 年提出的 VGG 网络[3]，它与 AlexNet 的网络结构类似，但是采用更深的网络层数；其次是减小网络中卷积核大小并增强卷积核功能，例如，NIN[4]模型使用全局平均池化层来代替全连接层，从而大大减小了网络参数数量；SqueezeNet[5]模型在保证准确率的前提下，通过使用 1x1 卷积核来减小通道数，进而将 AlexNet 模型的参数数量减少了 50 倍；MobileNet[6]模型采用了深度可分离卷积来减小计算量和参数数量；ShuffleNet[7]模型主要采用了组卷积和通道随机重排的方法来减小模型参数数量和计算量。第三个突破方向是将前两种突破方向结合在一起并引入残差网络进行收敛加速。这方面的代表模型是 ResNet[8]，它通过引入残差网络块结构，避免了梯度消失、梯度爆炸和退化问题。最后一个突破方向是将卷积神经网络的任务扩展为监测任务或增加新的功能单元，以实现更多样化和复杂化的应用。这方面的代表模型包括 R-CNN[9]、FCN[10]、LRCN[11]等，通过对卷积神经网络的功能进行扩展，如目标检测、图像分割、视频分析等，扩大了卷积神经网络在计算机视觉中的应用领域。

短短几年，神经网络的应用呈现出爆炸式增长，目前已广泛应用于各种人工智能应用，如计算机视觉、语音识别和机器人技术等。在一些领域，神经网络甚至已经取得了比人类更高的准确性。然而，这种出色的准确性是以计算复杂度为代价的。因此，一种能够提高能源效率和吞吐量，且不牺牲准确性的人工智能硬件平台，对于支撑神经网络持续飞跃式发展至关重要。

1.2.2 硬件加速器的研究现状

2010 年，法国科学家 Temam 在计算机体系结构顶级会议 ISCA 上发表主题演讲 The rebirth of neural networks[12]，阐述了神经网络在通用计算和专用计算的各个领域可能给硬件带来的重大影响，自此人工智能芯片[13,14]的设计进入了高速发展期，成为计算机体系结构和集成电路设计领域的一个重要研究方向。人工智能芯片架构通常采用高度并行的计算架构，主要包括时域计算机架构和空域计算机架构。

时域计算架构通常是指采用专门针对神经网络算法定制指令集的专用处理器架构，主要集中于神经网络处理器发展的早期，学者更多关注的是人工智能硬件在功能上是否支持目标应用场景。随着时间的推进，学者也开始关注时域计算架构本身的进一步优化，有两个典型的代表性设计。其一是通过利用内核分区技术减少数据复用[15-17]，解决了神经网络处理过程中因在不同设备间搬移数据而导致的能效受限问题；另一典型代表是中国科学院计算技术研究所的 DianNao[18-20]系列神经网络加速器，该系列的首款芯片-DianNao 于 2014 年设计完成，支持大规模 CNN 等卷积神经网络的加速处理。虽然时域计算架构的计算单元配置非常灵活，但是每一步操作都需要精确的指令来控制存储器访问和具体计算单元的操作类型，而且集中式存储设计导致架构与片外存储交互频繁，因此性能和能效受限。

空域计算架构设计中每个计算资源算数逻辑单元(Arithmetic Logic Unit, ALU)都具有独立的控制逻辑，并可以带本地存储器，整个架构采用数据流控制。其中较为典型的是 MIT 设计的 Eyeriss[21]，在计算过程中，Eyeriss 采用一种新提出的行固定(Row Stationary, PS)的数据流来大幅度提升片内数据复用率，同时实现了权重数据和输出数据的重用，与单一类型的数据复用模式相比，能够更好的利用低成本存储访问来提高能效。另一个空域计算架构的典型代表作是比利时鲁汶大学于 2017 设计的 ENVISION[22]架构，这是一种用于低功耗卷积神经网络、精度可扩展的计算架构，并在硬件上提供稀疏化的计算支持，提高了神经网络的硬件执行效率。谷歌公司于 2016 年推出的 ASIC 型云端神经网络加速器[23](Tensor Processing Unit, TPU)也是空域计算架构的典型代表之一，同年击败李世石的围棋机器人 AlphaGo[24]就在谷歌云上使用了 50 个 TPU 进行计算，谷歌在 TPU 设计上取得的巨大成功，使得人工智能芯片的研究开始受到工业界的重视，成为各大公司产品路线中的重要一环，微软[25]、英特尔[26]、NVIDIA[27]、赛灵思[28]等科技巨头都陆续推出相关产品，人工智能芯片产品

从此逐渐走向实用化。空域计算架构能够有效提高计算任务的执行性能，并控制能耗，但缺少时间域上的可变性，使其灵活性不足，应用范围受限。

除了专用加速器之外，基于 FPGA 的加速器的由于其低功耗、并行度高的优势，成为近年来主流卷积神经网络硬件加速平台之一，对于基于 FPGA 的卷积神经网络加速器的研究主要集中在三个方向。第一个研究方向是提升乘累加(Multiply Accumulate，MAC)阵列并行度，如 Li 设计的高效卷积计算引擎[29]中通过三种并行策略提高了运算单元的利用率；，第二个研究方向是数据传输和复用，如 Tian[30]等提出用于连续片外内存访问的非重叠平铺方法，并通过利用不同的循环排序策略探索片上数据重用机会，减少数据传输的时间。第三个研究方向是对于卷积神经网络模型的优化，主要分为稀疏网络结构[31-33]、网络剪枝[34,35]和网络压缩[36-38]三类，通过减少卷积神经网络操作数和模型大小从而间接提升卷积神经网络加速器的推理效率。

除此之外提出了多种新型智能加速架构[39-47]，但普遍存在以下问题，第一，通常设计为固定的计算模式，电路设计不可调整，即只能适用于单种硬件平台，无法做到加速器平台的升级。第二：存储系统通常采用固定的设计，无法按照不同网络层的特点调整访问模式和数据流支持，即对于数据传输过程中出现的数据复用问题，缺少灵活的复用策略。

1.3 研究内容及组织结构

1.3.1 本文的研究内容

针对以往卷积神经网络加速器只能应用于单平台，不便于加速器平台升级，即跨平台应用难的问题，本文设计了一种基于 ZYNQ 的可配置卷积神经网络加速器，并在 Xilinx 旗下的 FPGA 板卡上进行了实际应用，具体研究内容如下：

(1) 本文对比分析了不同数据复用方式下总传输数据参数量的差异，提出了一种自适应数据复用的策略，可以在每层卷积运算中对比不同复用方式的总参数量，灵活选取最优复用方式，从而减少数据传输的总参数；

(2)本文提出了一种软硬件协同配置的方法解决跨平台应用问题。在硬件层，针对不同硬件平台资源存在差异的问题，设计了一种可配置加速器，即在电路设计中将该加速器数据位宽、中间缓存空间大小和乘累加阵列并行度等作为一种可选配置

参数，并将相关配置参数添加到顶层配置文件，通过修改配置文件实现对占用资源的调整，以便于适应多种硬件平台；同理，在软件层，也将相关配置参数添加到顶层配置文件，从而实现跨平台应用的目的。

(3)为同等加速能力基础下设计较低功耗的电路，本文设计了一种划分工作域的电路架构，即在电路设计中将核心运算模块放置在高频时钟域，其余部分放置在低频时钟域，通过降低非核心模块时钟频率达到功耗降低的目的。

1.3.2 本文的组织结构

本文组织结构如下：

第 1 章，绪论，本章首先介绍了所研究课题的目的和意义，然后详细介绍了卷积神经网络及硬件加速器的发展脉络及研究现状，最后展示了本文的研究内容和组织结构。

第 2 章，相关理论基础，本章简要介绍了 CNN 加速器设计过程中涉及到的相关技术和开发流程，包括卷积神经网络的基础知识、ZYNQ 的开发流程和常见的低位宽量化方法。

第 3 章，CNN 加速器关键技术，本章主要介绍 CNN 加速器设计过程中的四项关键技术。一方面为了满足不同场景的精度需求，将量化后数据位宽设为一种可选项，可根据应用场景选择数据位宽大小；另一方面通过新增片上缓存、改进时钟结构来优化加速器电路时序；其次提出了一种自适应选择最佳数据复用的策略，以减少数据传输的总参数量；最后提出了一种软硬件协同配置的方案解决了 CNN 加速器在跨平台应用过程中面临的资源差异的难题。

第 4 章，CNN 加速器架构设计和仿真验证，本章主要介绍了 CNN 加速器的架构设计和硬件仿真。架构设计中分为软件层驱动设计和硬件层电路设计。在软件层分为数据加载、CNN 指令集配置和运算模块驱动引擎。在硬件层电路设计这部分主要介绍了运算模块的设计过程，并通过仿真验证了其逻辑功能的正确性。

第 5 章，CNN 加速器板级验证，本章对实验环境、实验过程和上板验证过程进行介绍，并将本文设计的 CNN 加速器与其他同类产品做比较，经实验可得，本文设计的 CNN 加速器在具备可配置能力的前提下，还有着较高的能效比。

第 2 章 相关理论基础

本章主要介绍卷积神经网络加速器设计过程中涉及到的相关理论基础，首先介绍卷积神经网络中的相关算子的运算方式，包括卷积、池化、激活和全连接，然后介绍 FPGA 的基础知识、硬件平台及其开发流程，最后简要介绍了现阶段常用的低位宽量化方法。

2.1 卷积神经网络基础

随着计算机视觉技术的蓬勃发展，对一些图像的识别效果，机器的准确率已经超过了人类，而这一切都要归功于卷积神经网络。对于卷积神经网络，我们熟知的有卷积层、池化层和全连接层等，另外还有反卷积层、激活层、批量归一化层、Softmax 层和 Shortcut 层等，这些层中丰富而全面的层算子使卷积神经网络在解决图像处理以及其他复杂问题上更快速、更高效。本节对卷积神经网络的基础算子进行简单介绍。

2.1.1 卷积算子

卷积算子是卷积神经网络中的核心算子，其在深度学习领域发挥着重要的作用，在计算机视觉方面引领了一波又一波的学习热潮。卷积算子中包括五个基本操作参数：输入特征图尺寸 I，卷积核尺寸 K (Kernel)，滑动步长 S (Stride)，填充像素数 P (Padding)和输出特征图 O。在卷积神经网络中，输出特征图 O 的计算方式如下式(2-1)所示：

$$O=[\frac{I-K+2P}{S}]+1 \tag{2-1}$$

其中输入特征图尺寸参数是指输入特征图的长、宽和输入通道数信息；卷积核是由一组权重参数组成的小矩阵，通常用于从输入数据中提取特征，也称滤波器或卷积滤波器，现阶段卷积神经网络中卷积核大小一般为 3x3 或 5x5；Stride 即在卷积运算中，遍历输入特征时每次移动的像素数，如当步长为 1 是每次移动 1 个像素，当步长为 2 时每次移动 2 个像素；Padding 是指在卷积运算中对输入特征图边界处理的方式，为了不丢弃原图信息，让更深层的输入依旧保持有足够大的信息量，一般

会对输入特征图边界进行补零，再执行卷积操作，从而使得输出特征图尺寸不会太小，或者与输入特征图的尺寸一致；输出特征图尺寸是指输出特征图的长、宽、输出通道数参数信息。如下图 2-1 所示为单通道时卷积运算过程。

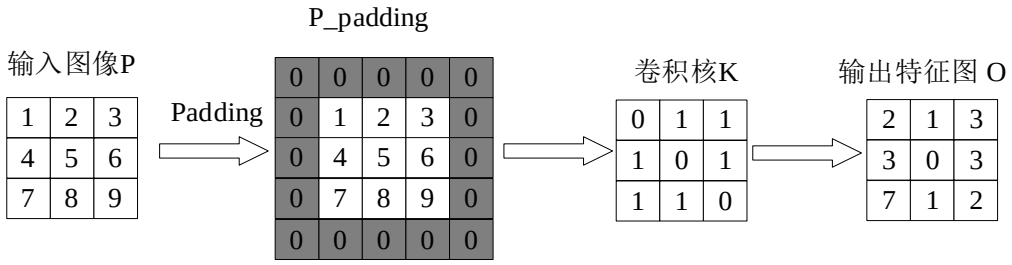


图 2-1 卷积运算示意图

在卷积层中，特征图一般都是三维的形式，当采用 M 个尺寸为 k×k×N 的三维卷积核，从尺寸为 h×l×N 的输入三维特征图中提取参数时，得到尺寸为 r×c×M 的输出特征图。其计算过程可以理解为：每个三维卷积核在输入特征图上方向上以 S 为步长，分别与滤波器窗口内数据构成的三维输入数据子向量进行卷积运算，得到一张尺寸为 r×c 的输出二维特征图，M 个卷积核计算完成以后得到 M 张尺寸为 r×c 的输出二维特征图，构成尺寸为 r×c×M 的输出三维特征图。该计算过程如下图 2-2 所示。

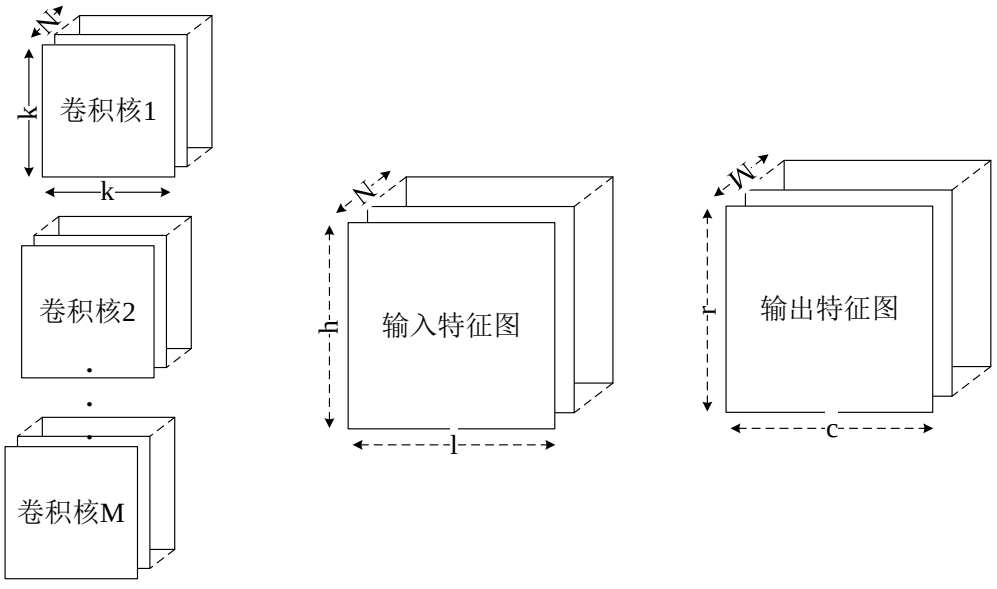


图 2-2 卷积层运算示意图

2.1.2 池化算子

池化算子是神经网络中的池化层算子，池化层算子也被称为下采样算子，其池化类型包括最大池化(MAX POOL)和平均池化(AVG POOL)两种，池化层通过在感受野进行相应的池化操作来对图像进行下采样。其中，最大池化是将待处理的特征图中感受野内池化算子范围内的值选取最大值作为输出值，经过这种运算后的特征图像能够有效减少旋转、平移或者缩放影响。平均池化是将池化算子范围内像素的平均值作为输出值。如下图 2-3 所示为池化核为 2x2 时的池化运算示意图。

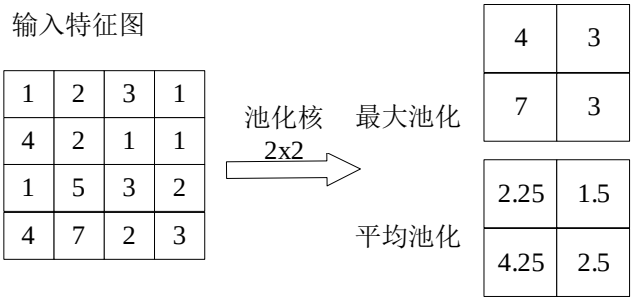


图 2-3 池化运算方式示意图

2.1.3 激活算子

激活函数是神经网络中的重要组成部分，其定义了上一层神经元输出与下一层神经元输入之间的函数关系，通常用于卷积层或全连接层后，对输入信号进行非线性映射。而最原始的感知机由没有激活函数的神经元连接而成，只能处理线性问题，这一缺陷导致了第一次人工神经网络研究的衰退。因此，激活函数是神经元中非常重要的组成部分，它极大地增强了网络的学习和表达能力。

激活算子通常使用特定函数表示，在卷积神经网络中，常用的激活函数包括 Sigmoid、Tanh、ReLU 和 Leaky_ReLU 函数。每个函数都有其独特的数学属性，可以影响网络的学习效果和训练速度。其常用的激活函数如下表 2-1 所示。

Sigmoid 激活函数又称为 Logistic 函数，通常用于二分类模型。它可以将变量映射到值域为(0,1)的区间中，并在输入值接近两端时抑制输入并趋向饱和。然而，Sigmoid 函数存在三个缺点。首先，它的输出值始终为正值，不对称于零点，导致权值只能向一个方向更新，从而使收敛速度变慢。其次，Sigmoid 函数需要进行指数计算，非常耗费计算资源。最后，其饱和性质导致梯度消失，使得网络变得难以学习。

Tanh 函数是 Sigmoid 函数的改进版，和 Sigmoid 函数有异曲同工之妙，这两个

都是在早期研究工作中被广泛适用的激活函数，它由双曲正弦 sinh 和双曲余弦 cosh 相除得到，将变量映射到[-1,1]的区间范围内，从而克服了 Sigmoid 函数非 0 均值的缺点，不容易出现 loss 值晃动，收敛速度快。但是它和 Sigmoid 函数一样，都是饱和型函数，会出现梯度消失的问题，而且这两个函数的计算量级都是指数级的，计算相当复杂。

ReLU(Rectified Linear Unit, ReLU)激活函数是一种常用的激活函数，在深度学习中得到了广泛的应用。它是一种线性且不饱和的激活函数，当输入值大于 0 时，输出值等于输入值，当输入值小于等于 0 时，输出值等于 0。ReLU 的优点在于它不需要进行指数运算，计算速度比 Sigmoid 函数和 Tanh 函数快很多，同时也具有很好的稀疏性，能够显著减少计算量。此外，当输入值大于 0 时，导数为 1，从而可以在一定程度上缓解神经网络中的梯度消失问题，有利于加速梯度下降的收敛速度。然而，ReLU 函数也有一些缺点。当输入值小于等于 0 时，导数为 0，因此在训练过程中，这些神经元的权重将永远无法更新，也就是说，这些神经元会“死亡”，从而导致网络性能下降。

表 2-1 常用的激活函数表

激活函数名称	激活函数计算公式
Sigmoid 函数	$S(x)=\frac{1}{1+e^{x}}$
Tanh 函数	$Tanh(x)=\frac{1-e^{-2x}}{1+e^{-2x}}$
ReLU 函数	$f(x)=\max(x,0)$
Leaky_ReLU 函数	$y=\max(x,leak\times x)$

Leaky ReLU 函数是一种改进版的 ReLU 函数，它通过在零点左侧引入一个小的斜率，来避免 ReLU 函数在这一区域出现的死亡神经元问题。具体地说，当输入小于 0 时，Leaky ReLU 函数会返回一个小的斜率值乘以输入值，而当输入大于等于 0 时，它仍然返回输入值本身。这个小斜率通常是一个非常小的数，比如 0.01。与 ReLU 函数相比，Leaky ReLU 函数在一定程度上减少了死亡神经元的问题，并且仍然保留了 ReLU 函数的线性和稀疏性质。在实践中，Leaky ReLU 函数已经被证明是一种比较有效的激活函数，尤其是在卷积神经网络中的使用效果更佳。

2.1.4 全连接算子

卷积神经网络(Convolutional Neural Network, CNN)通常由卷积层和池化层两种类型的层组成。然而，在某些情况下，为了将卷积层和池化层提取的特征传递给最终的分类器，CNN 还包含全连接层。全连接层主要作用是将卷积层和池化层的输出映射到指定的输出大小，以生成最终的分类或预测结果。通常，全连接层位于卷积层和池化层的后面，用于对卷积特征进行分类。

全连接层的结构与普通神经网络相似，包含输入层、输出层和一个或多个中间层。不同之处在于，输入层通常是卷积层或池化层的输出结果。在全连接层中，每个神经元都与前一层的所有神经元相连，权重矩阵的大小为前一层神经元的数量乘以当前层神经元的数量。因此，在输入数据维度非常高的情况下，全连接层的参数量非常庞大。为了避免过度拟合和参数量过多的问题，在卷积神经网络中，全连接层通常只用于最后几层。此外，全连接层的输出通常需要经过 softmax 激活函数，以输出预测类别的概率分布。

2.2 基于 ZYNQ 的开发流程

2.2.1 FPGA 基础知识

FPGA 是一种可编程逻辑门阵列，用户可以根据需要进行编程配置，实现各种逻辑功能。由于其灵活性高、可重配置性强、可实现高速运算等优点，使其广泛应用于数字信号处理、通信、图像处理、控制系统等多个领域。下面是关于 FPGA 的基础知识。

FPGA 的结构：FPGA 由可编程逻辑单元(Configurable Logic Block, CLB)、输入输出块(Input/Output Block, IOB)、时钟管理单元(Buffered Universal Clock Gate, BUFG)、数字信号处理器乘法器(Digital Signal Processor, DSP)等组成，这些单元通过可编程的互连资源进行连接。

FPGA 的编程方式：FPGA 的编程方式有两种，一种是使用硬件描述语言(Hardware Description Language, HDL)进行编程，如 Verilog 和 VHDL；另一种是使用高级综合工具 HLS 进行编程，如 VIVADO HLS。

FPGA 的时序分析：由于 FPGA 中逻辑单元和互连资源的可编程性，时序分析在

FPGA 设计中是非常重要的一环，需要进行时序约束的设置和时序分析的验证。

FPGA 的设计流程：FPGA 的设计流程通常包括设计、综合、实现和验证等环节，其中综合和实现是 FPGA 设计中的核心环节。

综上所述，FPGA 是一种可编程逻辑器件，是数字电路设计中的重要工具之一，目前在多个领域得到广泛应用。

2.2.2 ZYNQ 结构

ZYNQ UltraScale+是赛灵思(Xilinx)公司推出的一款高度集成的可编程 SoC(系统级芯片)，它融合了赛灵思的最新一代 UltraScale+ FPGA(现场可编程门阵列)和 ARM Cortex-A53/Cortex-R5 多核处理器，其采用了 TSMC 16nm FinFET+工艺，具有出色的性能和低功耗特性。它支持多种不同类型的 I/O 标准，包括 PCI Express Gen3、USB 3.0、Gigabit Ethernet、DisplayPort 等，同时还集成了高速存储器控制器、视频编解码器、数字信号处理器以及各种外设接口，为用户提供了丰富的硬件资源和接口。ZYNQ UltraScale+ MPSOC 还配备了可编程逻辑和硬件调试器，用户可以使用 VIVADO 开发套件进行设计、仿真和调试。此外，赛灵思还提供了丰富的开发工具和软件库，帮助用户快速开发出高效、可靠的系统。目前 Xilinx 推出的搭载此类型芯片的产品包括 Ultra 96 v2、ZCU102、ZCU104 和 ZCU106 等，主要应用于机器视觉、监控、汽车自动驾驶等场景。如下图 2-4 所示展示了 ZCU104 的架构图。

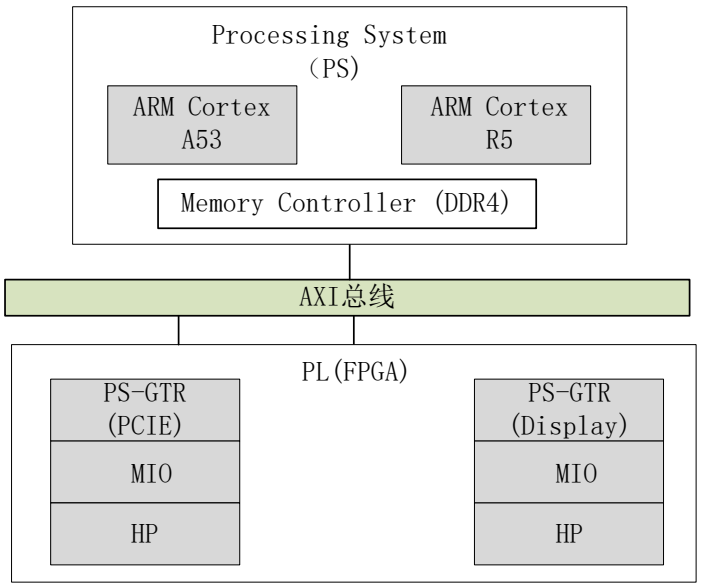


图 2-4 ZCU104 的架构示意图

该架构图显示了ZCU104开发板上的一个四核的ARM Cortex-A53和一个双核的ARM Cortex-R5，以及DDR4内存控制器。此外，该开发板还包括一个可编程逻辑区域FPGA和多个高速串行连接口，包括PCI Express和DisplayPort。MIO和HP是处理器子系统中的多个输入/输出端口，它们可用于与外部设备进行通信。

总的来说，ZYNQ UltraScale+结构集成了高性能的FPGA和多核处理器，为各种应用场景提供了强大的计算和通信能力，是一款非常优秀的可编程SoC芯片。

2.2.3 FPGA 开发流程

三种基于 FPGA 硬件电路开发方式中，RTL 级开发过程最为繁琐，开发周期最长，一般来说基于 RTL 级开发的完整流程，包括需求分析、方案评估、芯片选择、详细方案设计、RTL 代码编写、功能仿真、综合优化、布局布线、静态时序分析、上板验证等环节，以下是 RTL 设计流程中各环节具体内容。

(1)需求分析

在 FPGA 项目开始之前，必须充分分析项目要求的功能和各方面性能指标，从而根据项目要求的功能和性能指标，大致分析项目预计资源使用量、设计程度、数据带宽等关键内容，并给出需求分析报告。

(2)方案评估

根据需求分析报告，对需求分析相关因素总行综合权衡，选择可满足要求的FPGA器件和较为合适的设计方案，并给出方案评估报告，分析方案的可行性。通常，FPGA方案评估时应考虑到FPGA芯片选择、FPGA开发环境、FPGA整体设计架构、数据流传输、时钟和复位信息等。其中，FPGA选型尤为重要，要充分考虑到将消耗的I/0资源、时钟资源、逻辑资源、RAM资源、DSP资源、高速接口数量等，并结合成本和关键资源瓶颈，选择合适型号板卡。

(3)详细方案设计

方案评估报告通过后，即可进行详细电路设计了。一般采用从整体到部分的设计思路，将整个大系统分为多个小模块，即将电路模块化，以便于各模块功能独立设计并验证。 详细方案一般包括开发环境、FPGA 整体架构设计、FPGA 通信接口定义、FPGA时钟和复位系统、子模块结构设计、子模块接口定义、软硬件通信方式等。模块化设计的方式一方面有助于观察和思考各个模块划分的合理性，使得整体

设计脉络分明。另一方面可以为不同设计者之间系统工作提供帮助。

(4)RTL 级代码编写

将详尽方案中各模块所设计的电路以 Verilog 或者 VHDL 的语言进行描述，并将各模块连接成一个整体。需要注意的是，描述的目标对象必须是综合、可映射为电路的结构，并且在 EDA 工具为可直接综合成网表文件，已便于布局布线到 FPGA 芯片中。在电路编程过程中需要反思两个问题，一是描述的电路是否满足功能要求，二是描述的电路在 FPGA 中用什么资源实现，其是否为最佳方式。

(5)功能仿真

功能仿真也叫做前仿真，一般用来对所描述的电路进行逻辑功能验证，不考虑硬件的线路延时情况，仅对初步的功能进行验证。

(6)综合优化

综合就是将高层次的描述语言转化为对硬件的直接描述，优化综合就是将器件约束和目标对象匹配，已实现约束要求。一般来说通过综合可以将输入编译成由与门、非门、或门、触发器、RAM 等基本逻辑单元组成的逻辑连接网表。

(7)布局布线

布局布线是将综合生成的逻辑连接网表映射到 FPGA 器件的芯片上的过程。它包括布局阶段和布线阶段。在布局阶段，底层单元和硬件原语被放置到芯片内部固有的硬件上，以实现面积最优或速度最优的选择。而在布线阶段，会根据布局的拓扑机构，合理利用芯片内部的各种连线资源，以正确连接各个元件。整个布局布线过程对于 FPGA 设计至关重要，能够显著影响电路的性能和功耗，因此需要进行严密的设计和优化。

(8)静态时序分析

静态时序分析是一种检测电路时序违规的方法，通过计算电路的延时来判断是否满足时序约束条件和器件固有的时序规则(如建立时间和保持时间等)，而无需模拟电路。时序优化则是通过约束优化、设计修改(例如插入寄存器、并行结构、逻辑展开、寄存器平衡和路径重组)以及手工修改布局布线等方式反复优化综合布局布线，以逐渐收敛时序，以满足设计要求的过程。

(9)上板验证

上板验证是 FPGA 设计的最后一步，通常需要将设计生成的比特流文件通过编

程器烧写到 FPGA 芯片中，然后通过一系列的测试和调试来验证设计方案的正确性和性能指标是否符合预期。在上板验证过程中，工程师需要对设计进行逐个模块的测试，并检查各个模块之间的接口是否正常，同时还需要进行时序分析、功耗分析等测试，以保证 FPGA 设计的性能和稳定性。此外，在上板验证过程中，还需要进行一系列的调试和优化工作，例如时钟域转换、布局布线优化、时序约束调整等，以满足设计的性能指标和约束要求。总之，上板验证是 FPGA 设计过程中不可或缺的重要步骤，对于保证设计的正确性和性能指标具有至关重要的作用。

2.3 低位宽量化方法

近年来，为了获取较高的网络模型预测精度，卷积神经网络模型的层数和复杂度不断增加，使得模型变得越来越大，带来了巨大的计算和存储代价。这种情况特别突出在一些计算能力相对较弱的边缘设备上，导致神经网络的应用受到了严重阻碍。为了在有限的资源和硬件设备上实现智能计算加速，我们需要对网络模型进行优化和压缩，其中低位宽量化技术就扮演了重要的角色。通过减少网络参数的位宽，我们可以将神经网络压缩到更小的规模，降低计算和存储的代价，并在边缘设备上实现更快速、更高效的计算加速。

神经网络的低位宽量化技术是一种常用的深度学习技术，旨在将神经网络的权重和激活值从高位宽的浮点数转换为低位宽的整数或定点数。这种技术的主要目的是在保证神经网络性能的同时，降低神经网络的计算资源和存储资源消耗。低位宽量化技术通常可以分为对称量化和非对称量化两种方式。对称量化是指将权重和激活值统一量化为有符号整数，通常采用二进制表示，例如二值网络和三值网络等。而非对称量化则是将权重和激活值分别量化为有符号整数和无符号整数，通常采用定点表示，例如 8 位整数和 16 位整数等。低位宽量化技术可以有效降低神经网络的存储需求和计算复杂度，并且不影响神经网络的分类性能。因此，在实际应用中，低位宽量化技术被广泛应用于各种嵌入式设备、移动设备等资源受限的场景中。

Bengio[48]等人最早提出了二元连接的概念，使用带符号的 1 或-1 代替权重的实数值，以将乘法操作转换为加法操作。这种方法的改进版本是二值神经网络(BNN)，它将权重和激活函数都表示为二进制形式，并将乘累加运算变换为同或运算[49]。尽管这种方法可能会导致网络精度损失较大，但研究者已经提出了多种方法来降低精

度损失，例如同或网络(XNOR-Net)、二值权重网络(BWN)和低精度量化神经网络(Quantized Neural Network)。近期还出现了三值权重网络[50](Ternary Weight Net，TWN)，它通过给二值权重增加一个“0”值，即使用三个值“-1，0，1”来保存权重。尽管三值化取得了一定的效果，但无论是二值化还是三值化，都很容易在网络训练过程中出现过拟合的问题，从而导致经训练后测试精度较差，无法满足预期要求。为了解决这个问题，研究者提出了 8 位整型量化[51-53]和 16 位整型量化[54]，这两种方法在避免过拟合的同时，可以有效提高网络精度。虽然 8 位整型量化的资源消耗较少，但 16 位整型量化能够获得更高的精度。

2.4 本章小结

本章主要介绍了卷积神经网络(CNN)的基础知识、硬件开发流程和量化方法。首先，介绍了卷积算子、池化算子、激活算子和全连接算子等几种算法，并简要介绍了它们的运算方式，为硬件电路中运算模块设计提供理论基础。其次，介绍硬件电路设计过程涉及到的操作流程，包括 FPGA 基础知识、ZYNQ 结构和 FPGA 开发流程。最后，介绍了 CNN 的低位宽量化技术，包括二值化、三值化、8 位整型量化和 16 位整型量化等方法，并简要介绍了这些方法的特点。

第 3 章 CNN 加速器关键技术

为设计一种可跨平台的高效卷积神经网络加速器，需要解决三个关键问题，首先需要考虑跨平台应用的问题，其次还需要提高加速器本身的性能，除此之外还需保持较高的推理精度。针对这些问题，本章分别从量化方法、电路时序优化、数据复用和软硬件配置方面提出了相关解决方案。

3.1 可配置定点量化

3.1.1 量化方法对比与选择

卷积神经网络的量化是硬件推理工作中较为重要的一环，直接影响模型推理的精度和资源消耗。量化环节一般在 PC 端完成，通常采用静态量化的方式将 32 位权重数据量化为低位宽数据，从而便于在 FPGA 端进行运算。低位宽量化方法有二值化，三值化，8 位定点量化和 16 位定点量化，如下表所示。

表 3-1 低位宽量化方法

量化方法	数据位宽	表示范围	优缺点
二值量化	Bit-2	[+1, -1]	速度最快，资源最少，精度损失最多
三值量化	Bit-2	[+1, 0, -1]	
8 位整型量化	Bit-8	[-128, 127]	精度损失较低，资源较好较高
16 位整型量化	Bit-16	[-65536, 65535]	精度损失最低，但资源消耗最多

二值化量化和三值化量化的优点是大量减少 FPGA 运算复杂度，运算速度更快，但相应的在 PC 端训练过程中会出现过拟合问题，导致模型推理精度很差；8 位整型量化中数据运算比 16 位数据运算消耗更少的资源，但推理精度低于 16 位量化方法。

经分析可得，二值和三值量化的方法不适用于通用卷积神经网络加速器，仅限于专门的卷积神经网络加速器，而 16 位整型量化方法和 8 位整型量化都可应用于卷积神经网络加速器中。考虑到 8 位整型量化和 16 位整型量化方法各有利弊，分别对应不同的应用场景，应将选择权交给使用者，所以本文在设计中将数据位宽作为一种可选项，可根据场景需求选择数据位宽为 8 或者 16，即当默认 8 位整型量化方法

下推理精度不满足要求时，可将数据位宽调整为 16 位整型数据，以牺牲资源换取较高的推理精度。

3.1.2 量化过程

在 PC 端通过训练后静态量化的方法，将数据通过线性量化的方式转换成指定位宽的整型数据。量化计算公式如下式(3-3)所示：

$$\text{scale} = \frac{f_{\max} - f_{\min}}{IR} \tag{3-1}$$

$$IR = 2^{DN} \tag{3-2}$$

$$q = round(\frac{f}{scale}) + zero \tag{3-3}$$

式中，f 为输入浮点数据，$scale$ 为伸缩因子，q 为量化后的整型数据，$zero$ 为量化后零点的位置，IR 为量化方式取值范围，DN 为量化后数据位宽。

量化后卷积神经网络模型的卷积运算过程，将在以下部分进行推导，首选，卷积运算作为一种乘累加计算，其浮点运算过程可以用式(3-4)进行表示。

$$f_3^{i,k} = \sum_{j=1}^{N} f_1^{i,j} f_2^{j,k} \tag{3-4}$$

式中，f_1 表示输入特征数据，f_2 表示输入权重数据，f_3 表示输出特征数据，i 表示输出所在行列位置，j 表示输入通道，k 表示输出通道。将上述式中的浮点型数据替换为量化后的整型数据，可以得到以下计算式。

$$scale_3 \times (q_3^{i,k} - zero_3) = \sum_{j=1}^{N} scale_1 \times (q_1^{i,j} - zero_1) \times scale_2 \times (q_2^{j,k} - zero_2) \tag{3-5}$$

式中，q_1 表示输入特征量化，q_2 表示输入权重量化，q_3 表示输出数据量化，$zero_1$、$zero_2$、$zero_3$ 分别表示量化后其零点位置，将上式稍加变换，可以得到：

$$q_3^{i,k} - zero_3 = P \times \sum_{j=1}^{N} (q_1^{i,j} - zero_1) \times (q_2^{i,k} - zero_2) \tag{3-6}$$

$$P = \frac{scale_1 \times scale_2}{scale_3} \tag{3-7}$$

为简化运算方式，本文选用了对称量化的方式，即将式中 $zero$ 值取为 0，则上式可以表示为：

$$q_3^{i,k} = P \times \sum_{j=1}^{N} q_1^{i,j} \times q_2^{i,k} \tag{3-8}$$

可见上述式子中除了 P 为浮点数之外，其余的运算都是整型数据的运算，根据浮点型数据的存储方式，将 P 分解成尾数和指数，如下式所示：

$$P = 2^{-n} P_0 \tag{3-9}$$

式中 2^{-n} 为指数，P_0 为尾数，与 P_0 的乘积为定点运算，与 2^{-n} 的乘积可以转换成右移运算。

经上述推导后，最终将量化后卷积运算转换成了定点运算，并通过公式(3-9)将乘法运算转换为移位操作，从而减少了乘法器资源的使用。

3.1.3 可配置数据位宽设计

为了卷积神经网络加速器能够根据推理精度需求选取合适数据位宽，本文在电路设计过程中，贯彻了通用可配置的思想，将输入数据位宽作为一种可调选项，由使用者根据场景需求选择默认 8 位数据位宽或者更高精度的 16 位数据位宽，其一方面能够适应精度需求，另一方面还能够调整该加速器中 DSP 资源使用量，这是由于在电路综合优化的过程中借助了 FPGA 板卡本身的特性，将一个输入位宽 16 位的 DSP 拆分位两个位宽 8 位的 DSP 来使用，从而使得同等计算能力的情况下，数据位宽为 16 时消耗的 DSP 数量大约为数据位宽为 8 时的 2 倍。通过数据位宽可配置的方法，使得本文设计的 CNN 加速器具备较为广泛的应用能力，能够满足大多数场景中精度需求。

3.2 硬件电路时序优化

随着卷积神经网络模型推理精度提升，其复杂程度和模型总参数量也越来越大，需要设计一种性能较强的卷积神经网络加速器做算力支撑，为设计性能较强的加速器需要时刻考虑关键指标：能效比。

能效比计算表达式如下式(3-10)所示，式中 EER 为加速器能效比，G 为吞吐量，$power$ 为运行功耗。由计算式可知，能效比与吞吐量成正比，与功耗成反比。

$$EER = \frac{G}{power} \tag{3-10}$$

峰值吞吐量是计算平台的性能上限，在本文中可以理解为每秒完成的乘累加运

算量，单位是 GOPS，其计算式可表示为：

$$CAP = f \times C_{\_TK} \tag{3-11}$$

式中，CAP 是加速器峰值吞吐量，f 是时钟运行频率，$C_{\_TK}$ 是乘累加阵列的并行度。由公式可知，要想提高卷积神经网络加速器算力，其一是提升时钟运行频率，其二是提高乘累加阵列的并行度。由于乘累加阵列并行度与 FPGA 芯片中乘法器资源量有关，无法通过电路设计提升，为此只能提高时钟频率以获取较高的能效比，然而时钟频率越高，其逻辑电路功耗也越高，从而间接影响能效比。

针对上述情况，本节提出了一种划分时钟域的方法优化了电路时序，一方面提升了加速器吞吐量，另一方面使得同等吞吐量下降低了逻辑电路的运行功耗，从而获得较高的能效比。除此之外，本节设计了一种内外层双重缓存的结构，优化了数据传输的方式，从而提高了数据从外部存储器(Synchronous Dynamic Random Access Memory, DDR4)到片上存储空间的传输效率。

3.2.1 内外双重缓存空间

本文设计了一种内外层双重缓存的结构，在外部存储器中加入了乒乓缓存，在片上存储空间中引入了中间缓存，具体设计如下图 3-1 所示。

(1)外部存储中乒乓缓存

首先，在外部存储空间中开辟出两块缓存空间用来存放输入特征图像，第一块用于当前推理运算，第二块用于下次推理运算，通过乒乓存储的方式，减少了新特征图加载的等待时间。

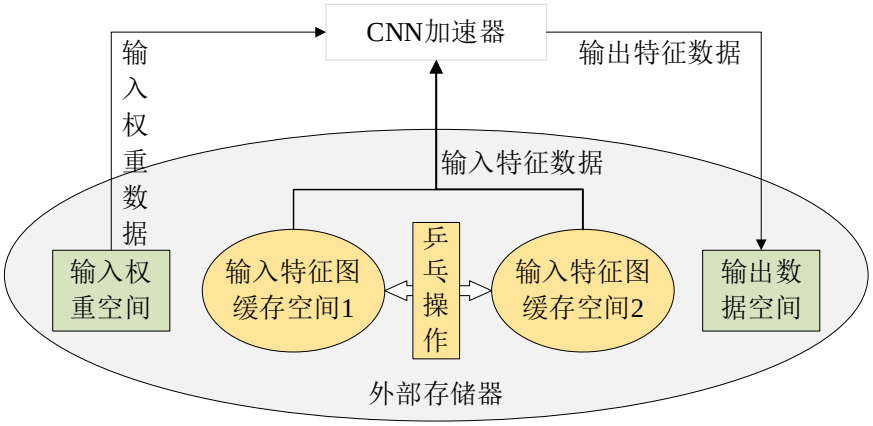


图 3-1 外部存储器中缓存空间

(2)片上中间缓存

卷积运算过程是指先输出特征数据矩阵上各值映射到原特征矩阵中的起始数据位置，然后遍历卷积核范围内的特征数据，并与权重数据做乘累加运算。当输入特征为 x，其通道为 ch_in；卷积核宽度为 kx，高度为 ky，行步进为 Sx，列步进为 Sy；方向补零为 Pad_left，列方向补零为 Pad_up；输出特征为 out，其宽度为 w_out，高度为 h_out，输出通道为 ch_out。则卷积运算伪代码计算过程如下图 3-2 所示。

```
for(k=0;k<ch_out;k=k+1)
   for(h=0;h<h_out;h=h+1)
     for(w=0;w<w_out;w=w+1)
       {
         sum=0;
        for(c=0;c<ch_in;c=c+1)
          for(r=0;r<ky;r=r+1)
           for(s=0;s<kx;s=s+1)
             {
             data_now= x[h*Sy-Pad_up+r][w*Sx-Pad_left+s][c];
             wt_now=wt[r][s][c][k];
             sum=sum+data_now*wt_now;
             }
      out[h][w][k]=sum;
       }
```

图 3-2 常规卷积运算伪代码

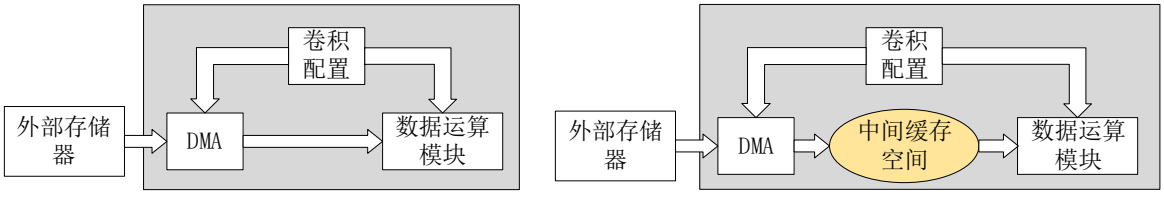
从伪代码中可以看出，正常卷积运算中乘法运算量 $\mathrm{mac\_}sum$ 为：

$$\mathrm{mac\_}sum = kx \times ky \times ch\_in \times ch\_out \times w\_out \times h\_out \tag{3-12}$$

由于每次乘法运算需要一个特征数据和一个权重数据，所以传统卷积运算方式中，总数据传输参数量 $\mathrm{size\_}sum$ 为乘累加运算次数的两倍，如下所示：

$$\mathrm{size\_}sum = mac\_sum \times 2 \tag{3-13}$$

在 FPGA 中执行卷积层运算时，由于数据参数量庞大，导致数据从外部存储器到数据运算模块需要消耗较长时间。当数据带宽无法满足运算模块需求时，会使得运算模块暂时空闲，从而无法获得较好的加速器性能。针对这一问题，本文在硬件设计时优化了电路结构，在外部存储器和运算模块间加入了中间缓存空间，如下图 3-3 所示，其中图 a)为直接传输方式，图 b)为优化后数据传输方式。



a) 直接数据传输方式　　b) 优化后数据传输方式

图 3-3 数据传输方式的优化

由于卷积运算中需要重复使用权重数据和特征数据，所以中间缓存空间选用片上存储器(Block Random Access Memory, BRAM)，当卷积神经网络加速器开始工作后，先将权重数据和特征数据从外部存储器中读取并搬运到中间缓存空间中，再从中间缓存空间中直接读取数据进行乘累加运算。

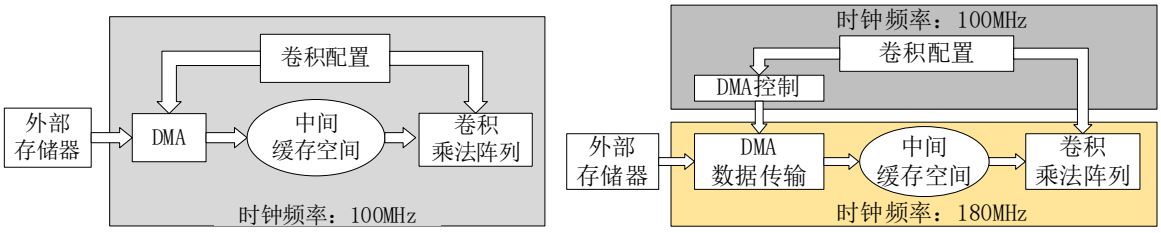
相比于直接从外部存储器中读取数据，加入中间缓存空间后，由于不需要数据运算模块的响应，可直接将外部存储器中的数据搬移到中间缓存空间，从而减少了传输总线的等待时间。除此之外，由于数据在片内的传输速度更快，能够满足数据运算模块的需求，从而减少了运算模块的空闲时间。

3.2.2 核心时钟频率优化

在 FPGA 中，想要系统工作在较高的时钟频率下，需要优化电路时序。一方面可通过综合工具上对时序进行约束；另一方面可以通过优化 RTL 代码结构，降低信号线的延时。如减少组合逻辑、插入中间寄存器、减少信号扇出等。

但在实验过程中发现经上述优化后，仍无法达到时序要求。这是由于当逻辑计算式较长时，将无法在单时钟周期内完成运算，从而出现上述时序问题。常规解决方法是将长计算式分解成多个子计算式分别运算，然后再将其相加计算，其优点是通过分解计算式、分批运算的方式优化了时序，使其工作在较高时钟频率。但同时也引来了两个新的问题，一方面，由于输出一次计算结果需要多个时钟周期，在这个过程中后续环节无法进行，从而加长了等待时间；另一方面，考虑到时钟频率提升，其运行功耗将相应升高，由式 3-10 可知，这将降低该加速器能效比。

针对上述问题，本文优化了电路结构，提出了一种划分工作时钟域的解决方案，将整个硬件电路划分为两个时钟区：高频时钟工作区和低频时钟工作区，如下图 3-4 所示，其中图 a)为优化前电路结构，图 b)为优化后电路结构。相比于优化前电路结构，经优化后其低频时钟保持不变，将高频时钟提升到 180MHz。



a)优化前电路结构　　b)优化后电路结构

图 3-4 时钟频率优化

由于加速器中参数配置部分和直接内存访问(Direct Memory Access, DMA)参数控制部分对时钟频率要求低，且不会影响高频时钟域中核心计算模块，所以将这一部分工作放置在低频时钟工作区域内，而核心计算和数据传输模块工作在高频时钟区域内。

通过划分工作时钟域的方法，一方面解决了逻辑函数计算式较长无法工作在高频时钟的问题，另一方面相比于常规解决方法，其削减了非核心模块的工作频率，从而降低了逻辑电路的运行功耗。

3.3 数据分块和数据复用

通过加入了中间缓存空间，能够减少数据从外部存储器到片上存储空间的等待时间。但由于 FPGA 片上存储资源有限，当卷积层输入参数量较大，无法一次性将所需输入权重数据和输入特征数据存放到片上缓存空间时，需要将权重数据和特征数据进行分块，即将一个大卷积拆分成多个小卷积。

针对上述问题，本节融合现有分块方法，确定了一种多方向分块方法，并对比分析该分块方式下不同数据复用方式的总参数量大小，提出了一种自适应数据复用的策略，从而减少数据传输总参数量。

3.3.1 多方向数据分块方法

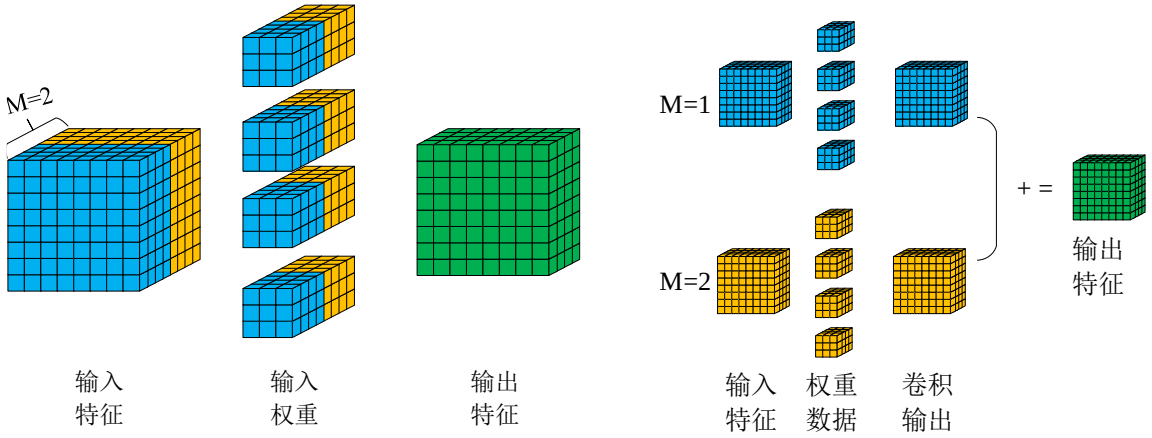
卷积运算中数据分块按方向可以分为 4 种：按输入通道方向分块、按输出通道方向分块、按输入特征高度方向分块和按输入特征宽度方向分块。如下表 3-2 所示。

表 3-2 数据分块方式

分块方式	描述
方式一	按输入通道方向分块
方式二	按输出通道方向分块
方式三	按输入特征高度方向分块
方式四	按输入特征宽度方向分块

分块方式一，如下图 3-5 中 a)图所示，将输入特征数据按通道方向分为 M 份，由于在卷积运算中特征数据和权重数据是相互对应的，所以相应的权重数据也随之

分块。这种方式的优点经分块后，将卷积层分为 M 个小卷积，在单次卷积运算中所需的特征数据、权重数据均减少，以便于将数据一次性存进中间缓存空间，但由于在卷积运算中需要将上一次小卷积运算结果与本次小卷积运算结果相加，所以需要缓存上一次计算结果，如下图 3-5 中 b)图所示，这个过程会增加过多的缓存带宽。



a) 分块方式　　b) 分块运算

图 3-5 分块方式一示意图

分块方式二，如下图 3-6 所示，按照输出通道方向将权重数据分为 N 份，由于权重和输出通道是相互对应，所以权重数据也相应范围 N 份，这种做法的好处是将权重数据分块后，单次卷积运算中所需的权重数据减少，从而便于一次性将本次卷积运算中所需权重数据存放到中间缓存空间，但从下图中可以看中输入特征数据没有分块，所以这种方式的是当特征数据较为庞大时，受 FPGA 片上缓存空间的限制，无法一次性将所需数据存放到中间缓存空间。

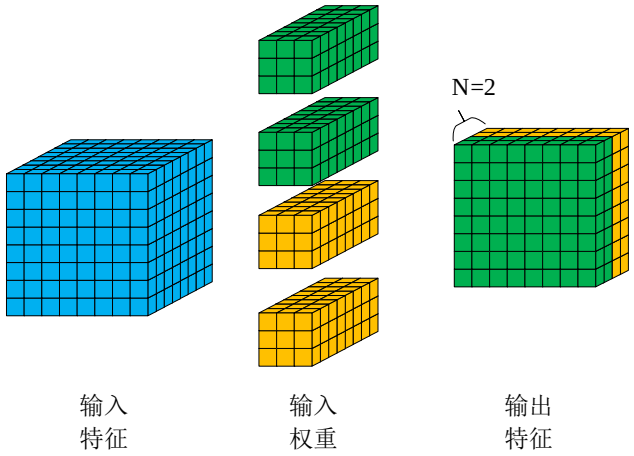
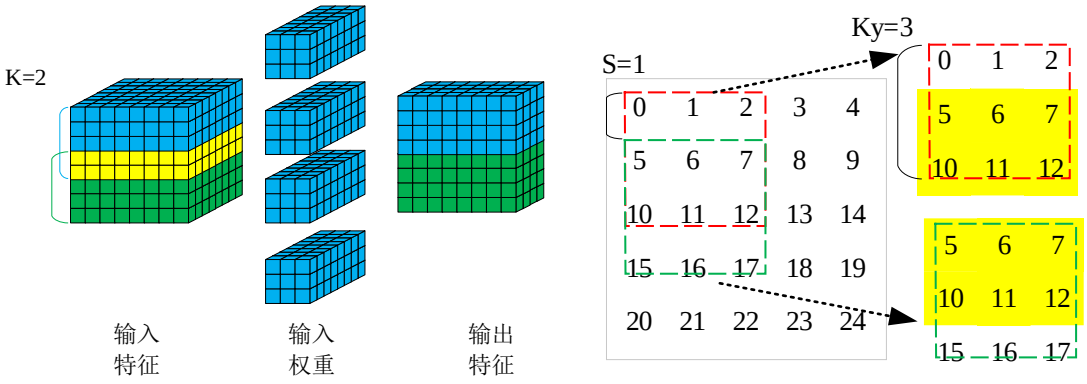


图 3-6 分块方式二示意图

分块方式三将输入特征图按高度方向分为 K 份，如下图 3-7 中 a)图所示。这种划分优点是将特征图分块后，卷积层运算将分为多个小卷积，在单次卷积运算中所需的特征数据会减少，从而便于一次性将所需特征数据存进中间缓存空间。其不足之处是在分块过程中，相邻输入特征块之间会存在重叠区域，如下图 b)图所示，当卷积核尺寸为 Ky、步进值为 S 时，在高度方向上会存在 Ky-S 的重叠数据。



a) 分块方式　　b) 数据重复使用

图 3-7 分块方式三示意图

分块方式四是将输入特征图按宽度方向分块，如下图 3-8 所示。与方式三相似，这种划分优点是经分块后，单次卷积运算中所需特征数据会减少，但相邻输入特征块之间会存在重叠区域。

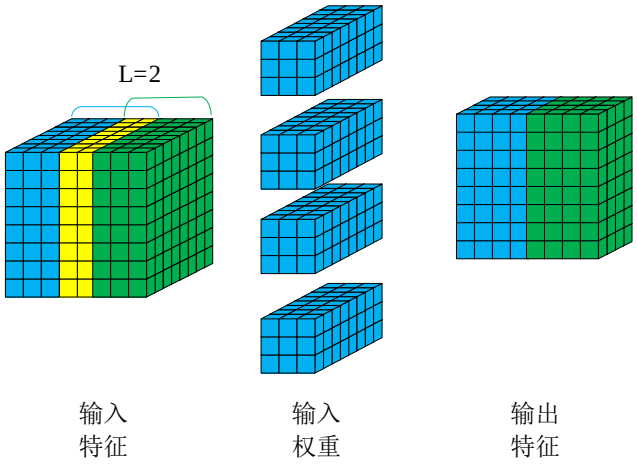


图 3-8 分块方式四示意图

对比分析四种分块方法，由于分块方式一将产生过多的额外缓存带宽，造成较多的资源占用，而其他分块方式又无法同时对特征数据和权重数据分块。针对这一

情况，本文将多种分块方式融合，同时在多个方向分割。考虑到方式三和方式四类似，所以选用一种即可，最终本文的分块方法如下图 3-9 所示，在输入高度方向上分为 K 份、输出通道上分为 N 份，这种分块方式的好处是同时对权重数据和特征数据进行分块，从而便于将数据从外部存储器中读出并暂存到中间缓存空间中。

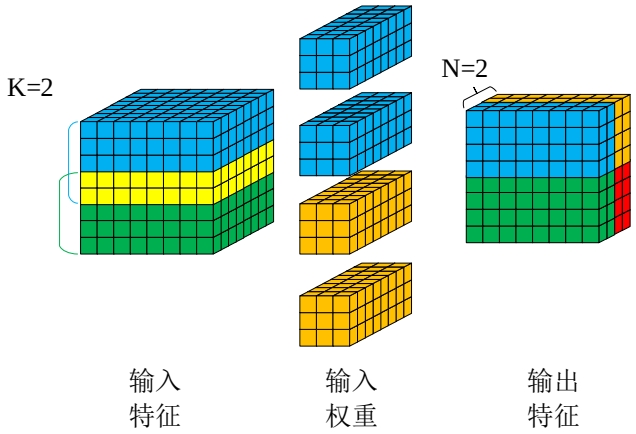


图 3-9 本文分块示意图

3.3.2 自适应数据复用策略

经上小节最终分块方式后，可以将一个大卷积分成 K×N 次小卷积。但是这种分块方式下伴随着数据的重复使用，如下图 3-10 所示，从图中可以看出在 4 次小卷积运算中存在特征数据块和权重数据块被重复使用的情况。

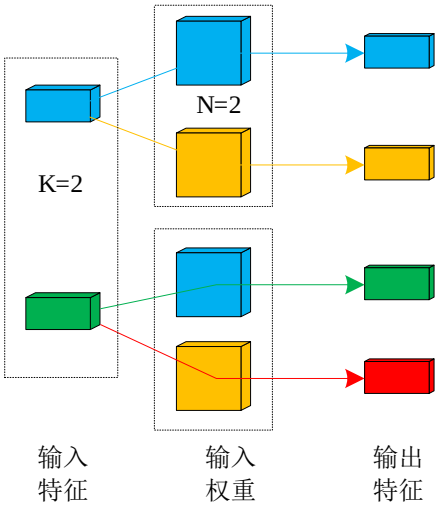


图 3-10 分块卷积示意图

针对该情况，可采用数据复用的优化方式，包括权重数据复用和特征数据复用。

权重数据复用是在卷积运算中先保持权重数据不变，遍历输入特征数据；特征数据复用是在卷积运算中保持输入特征数据不变，遍历权重数据。本节对比分析这两种数据复用方式，提出了一种自适应数据复用策略，以减少数据传输总参数量。

3.3.2.1 权重数据复用

如下图所示，卷积层的输入和输出特征图被分割成尺寸为 Tm×Th_i×Tw_i 和 Tn×Th_o×Tw_o，其中输出特征中Th_i 和Tw_i 由输入尺寸计算求得：

$$T\text{h\_i} = (Th\_o - 1) \times S + K \tag{3-14}$$

$$Tw\_i = (Tw\_o - 1) \times S + K \tag{3-15}$$

式中，S 表示扫描步长，K 表示卷积核的尺寸。

权重数据复用过程示意图如下图 3-11 所示，首先把 Tm 个通道的输入特征数据和权重数据写入到卷积运算核心的输入寄存器中，然后运算核心充分复用权重数据，遍历输入特征数据，并输出部分卷积运算结果，最后重复上面步骤，遍历输出特征图，将输出特征全部计算更新。

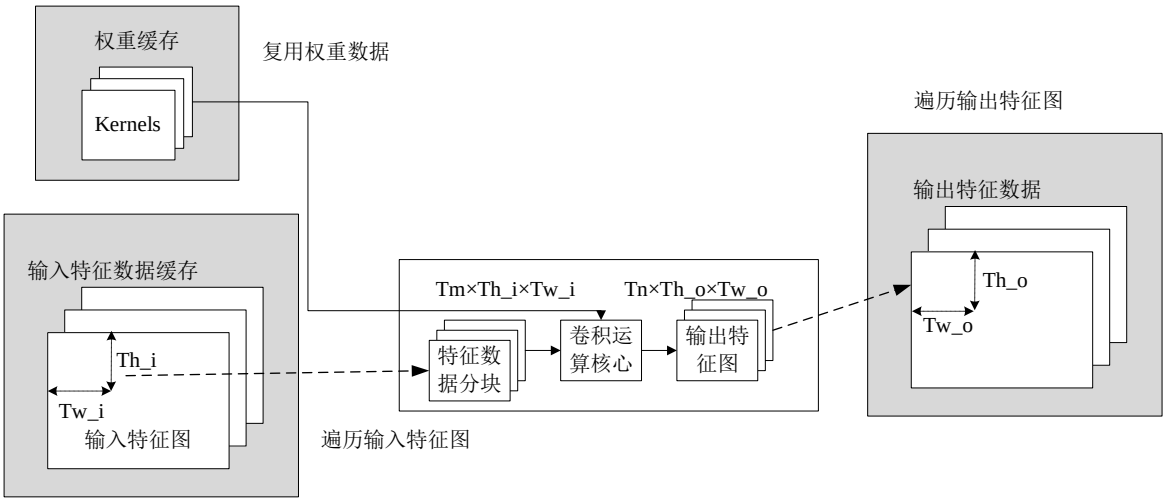


图 3-11 权重数据复用方式示意图

输入权重数据复用时，对中间缓存空间的参数总访问量$Size_1$计算公式如下(3-16)所示。

$$Size_1 = [A + overlap] \times N + B \tag{3-16}$$

式中，A 为输入特征总大小，B 为输入权重总大小，$overlap$ 为单块卷积运算中特征数据重叠部分，如上图 3-7 所示，则在小卷积运算中，被重复使用的特征数据总参数量$overlap$ 可用以下计算式表示：

$$overlap = \frac{A}{Hin} \times (\text{ky-S}) \times (K-1) \tag{3-17}$$

式中，*Hin* 为输入特征高度，ky 为卷积运算中卷积核尺寸，S 为步进值。

3.3.2.2 特征数据复用

特征数据复用过程示意图如下图 3-12 所示，首先卷积计算核心把单块输入特征数据读入局部的寄存器，然后运算核心充分复用特征数据，遍历权重数据，输出局部运算结果，最后遍历输出特征图，将输出特征全部计算更新。

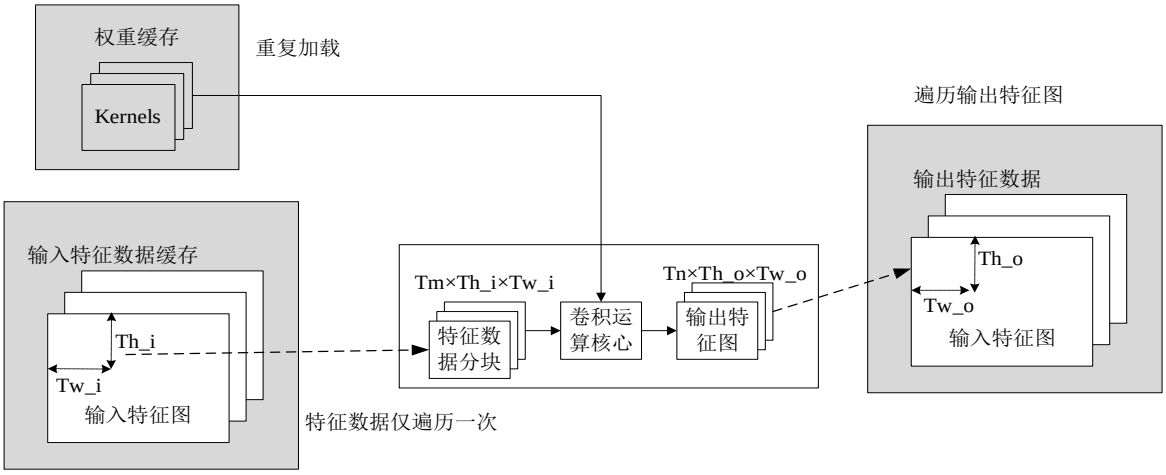


图 3-12 特征数据复用方式示意图

输入特征数据复用时，对中间缓存空间的参数总访问量 $Size_2$ 可由以下公式求得：

$$Size_2 = K \times B + A + overlap \tag{3-18}$$

3.3.2.3 自适应数据复用策略

为分析不同复用方式的差异，计算了当卷积核尺寸为 3，步进为 1 时，不同复用方式下总传输数据量，如下表 3-3 所示。

表 3-3 不同复用方式下总传输数据量

卷积层	一	二	三
输入尺寸	(224, 224)	(112, 112)	(56, 56)
输入通道	3	64	128
输出通道	64	128	256
(K, N)	(2, 2)	(4,1)	(3.1)
传统方式	86704128	924844032	924844032

权重复用	305472	919552	724992
特征复用	155328	1140736	1314816
本文	155328	919552	724992

从表中可以看出权重复用方式和特征复用方式下总传输数据量不同，且不同分块方式最优复用方式不同。针对这种情况，本文提出了一种自适应数据复用的策略，如下图 3-13 所示，在每层卷积运算前对比两种复用方式，自适应选择最优方式，以减少卷积运算中数据传输总参数量。

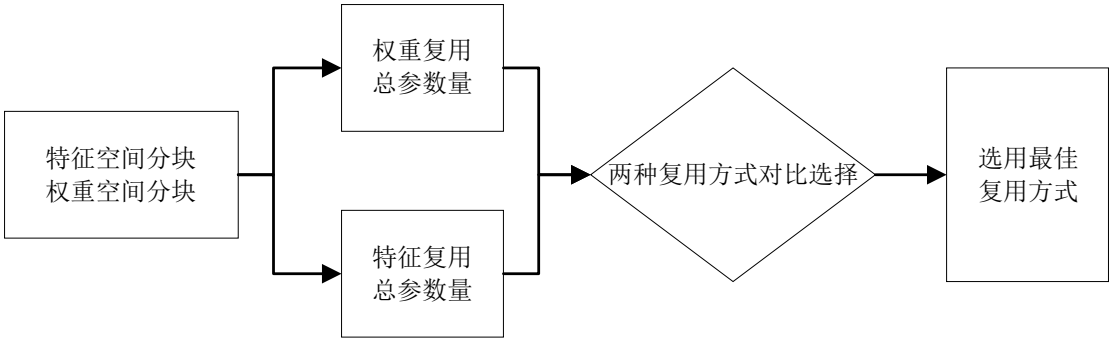


图 3-13 自适应数据复用方式

3.4 软硬件协同配置

为便于 CNN 加速器在 FPGA 硬件间的移植，以适应不同 FPGA 的资源，本节设计了软件配置+硬件配置的架构，如下图 3-14 所示。该加速器将量化后数据位宽、中间缓存空间大小、乘累加阵列并行度作为一种可选配置参数，并将其添加到配置文件中，通过选用不同配置方式，即可调整 CNN 加速器的资源使用量，从而适配不同 FPGA 平台。如下表 3-4 所示。

对于不同应用场景，可以从量化精度、FPGA 硬件资源等多方面考虑，选取合适配置参数，以满足多种 FPGA 硬件平台的需要。

图 3-14 软硬件协同配置

表 3-4 硬件电路中可配置参数表

配置参数	含义
Tk	数据排序时每组内包含数据的数目($Tk=2^n$，n 为正数)
Tk×Tk	乘累加阵列并行度(Tk×Tk)
DN	数据位宽(16 或 8)
BRAM_DATA	特征数据所在的 BRAM 空间地址深度
BRAM_WT	权重数据所在的 BRAM 空间地址深度

3.5 本章小结

本章对 CNN 加速器设计过程中的关键技术进行说明，一方面通过数据分块和软硬件协同配置的方式解决 CNN 加速器在跨平台应用中面临的难题。另一方面又通过新增片上缓存、改进时钟结构和自适应选择最佳数据复用的策略优化了电路设计，提高了加速器计算性能。最后为了满足不同场景的精度需求，将量化后数据位宽设为可选项。

第 4 章 CNN 加速器架构设计与仿真验证

为设计具有高度通用能力的卷积神经网络加速器，以满足跨平台应用的需求，采用了软硬件协同加速的方案。本章将首先介绍加速器的整体架构，以及ARM+FPGA 协同工作的方式，然后对软件层驱动设计和硬件层电路设计进行详细说明，最后对该加速器中主要功能模块进行仿真，以验证逻辑功能的正确性。

4.1 CNN 加速器整体架构

本文设计的CNN加速器采用软硬件协同的方式，将整体架构设计分为两大部分，软件层(Processing System, PS)和硬件层(Programmable Logic, PL)。其中 PS 端处理器为 ARM Cortex-A53，包含了四个处理核心，主要用于调度 PL 部分；PL 部分由各种硬件逻辑资源构成，包含了查找表(Look-Up Table, LUT)、触发器(Flip-Flop, FF)、乘法器 DSP、缓存空间 BRAM 等，卷积运算核心在 PL 部分完成设计，用于对 CNN 加速器的推理运算。

整体设计架构如下图 4-1 所示。PS 层由指令集、数据复用和硬件驱动三部分组成，主要完成卷积网络模型搭建、PL 调度和加速器状态检测的工作；PL 层由数据交互通路、卷积+激活模块、池化模块三部分组成，主要完成 CNN 加速器的推理运算。该加速器架构中数据总线为 AXI4，控制总线为 AXI4 LITE。

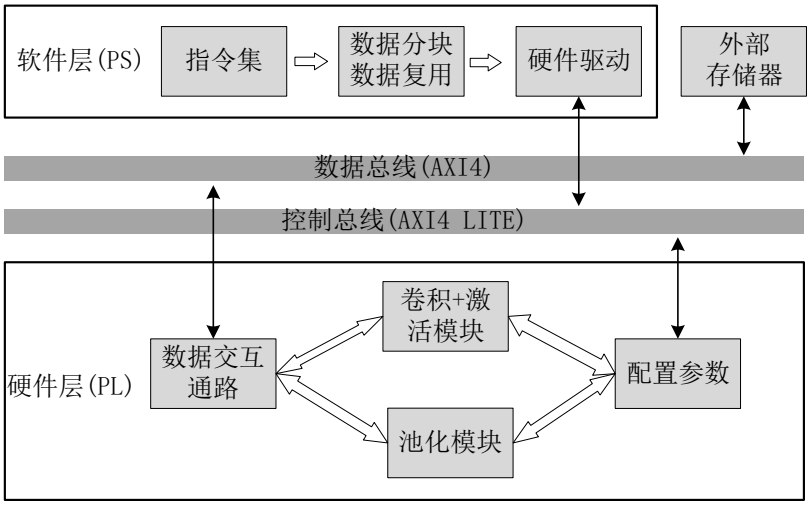


图 4-1 CNN 加速器整体架构设计

该加速器数据流由上图可看出，首先在 PS 层使用 CNN 指令集搭建卷积神经网络模型，然后计算单层卷积所需数据参数量，当所需数据总参数量大于片上缓存空间时，将该层卷积运算分为多个子卷积，其数据按本文提出的多方向分块方式进行切割，并通过自适应数据复用策略动态选择最佳复用方式，最后通过控制总线将相关配置参数发送给硬件层运算模块，驱动运算模块开始工作。在 PL 层，当收到 PS 部分发出的卷积层开始信号时，表示卷积层所需数据已经按照特定排列方式存放在外部存储器中，开始启动 CNN 加速器。该加速器运算过程如下，首先通过数据交互通路从外部存储器中读取相关数据并搬运到片上缓存空间，然后驱动卷积+激活模块或池化模块开始运算，最后通过数据交互通路将本次卷积或池化的运算结果返回到外部存储器指定空间。

4.2 软件层驱动设计

软件层主要由 CNN 指令集配置、数据分块和数据复用方式、驱动引擎三部分组成。在准备阶段，首先将所需数据以 BIN 文件的形式存储在 SD 卡中，当软件程序开始运行后，再将所需数据从 SD 卡中读出，并通过数据总线将其存储到外部存储器中，等数据加载完成后，通过控制总线对硬件层中运算模块进行配置，以驱动运算模块完成相应计算。本节主要对数据加载、CNN 指令集和驱动引擎三部分进行阐述。

4.2.1 数据加载

为便于外部存储器空间管理，将其空间划分为三大部分：输入权重数据空间、输入特征数据空间和输出特征数据空间，如下表 4-1 所示。在本文设计中默认外部存储器空间为 2GB，如下所示为外部存储器中默认地址划分区间，为便于不同配置不同卷积神经网络模型，其相关划分区间可在配置文件中调整。

表 4-1 外部存储空间地址规划

数据类型	地址区间	空间大小
输入权重数据	0x10000000-0x30000000	512MB
输入特征数据	0x30000000-0x38000000	128MB
输出特征数据	0x38000000-0x40000000	128MB

在准备阶段，现将重组后的权重数据文件和特征数据文件存放在 SD 卡中，待加

速器开始工作后，首先从 SD 卡中读取数据文件，并通过 AXI4 总线将相关数据存储到外部存储器指定空间，然后再执行后续指令集配置。

4.2.2 CNN 指令集配置

本文在软件层将相关配置参数封装到一起，定义了一套专用 CNN 指令集，以便于对快速搭建卷积神经网络模型，其指令集参数说明如下表 4-2 所示，其中功能扩展接口用于后续扩展和升级，以便于后期 CNN 加速器的功能升级。

表 4-2 CNN 加速器指令集

序号	参数	说明
0	Input_features	输入特征图长、宽、输入通道
1	Output_features	输出特征图长、宽、输出通道
2	Conv_inform	卷积层有效、卷积核尺寸
3	Active_inform	激活函数有效、激活方式
4	Pool_inform	池化层有效、池化核尺寸、池化方式
5	Panding	补零
6	Conv_Stride	步进值
7	Input_featur_data	输入特征空间大小
8	Input_featur_data	输出特征空间大小
9	Quant_inform	量化方式
10-15	Function_Extend0	功能扩展接口

一条指令集表示一层网络，相当于封装成一个卷积-池化函数，通过反复配置该 CNN 指令集即可搭建不同卷积神经网络模型。以 VGG16 为例，需要配置 21 次指令集来搭建网络模型，其中 16 层卷积层和 5 层池化层。

4.2.3 运算模块驱动引擎

软件层定义了卷积运算模块和池化运算模块的驱动引擎，该引擎通过控制总线将相关配置参数发送给运算模块，从而驱动其完成卷积+激活运算和池化运算。

(1)控制总线

本文加速器架构中控制总线采用 AXI4 LITE(Advanced Extensible Interface 4

LITE)协议，这是一种简单而灵活的总线协议，特别适合于连接处理器和低带宽外设。它是 SoC 中常用的总线协议之一，被广泛应用于各种嵌入式系统。

通过控制总线读写操作如下图 4-2 所示。当 master 向 Slave 发送数据时，Master 先发送地址信息到 Slave，以指示它将要写入哪个寄存器，待 Slave 接收后，再将要写入的数据发送给 Slave，最后 Slave 将接收状态作为响应信号发送给 Master。当 master 从 Slave 读取数据时，同样先发送地址信息到 Slave，以指示它将要读取哪个寄存器，待 Slave 获取读地址后，再将该数据返给 Master。

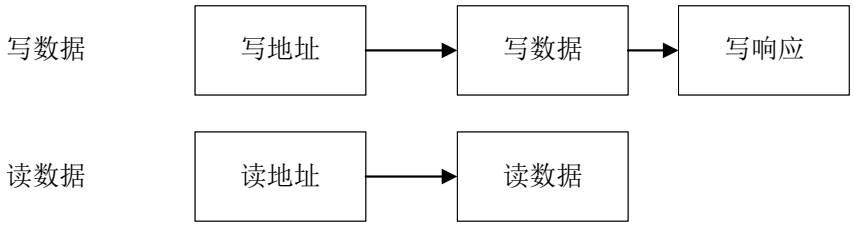


图 4-2 控制总线读写数据流程图

(2)卷积电路驱动引擎

为便于软硬件数据交互，卷积运算模块外部配置接口选用 AXI4 LITE，如下表 4-3 所示为该配置寄存器中参数信息，PS 端通过 AXI4 LITE 向卷积模块配置寄存器中写入相关卷积层配置信息，以驱动卷积模块完成相应运算。

表 4-3 卷积模块配置寄存器

地址	名称	含义	地址	名称	含义
0	start	开始	11	relu	激活
1	done	结束	12	ch_out	输出通道
2	error	出现错误	13	wt_size	权重大小
3	empty	空闲状态	14	dat_base_addr	输入特征起始地址
4	w_in	输入特征宽度	15	wt_base_addr	输入权重起始地址
5	h_in	输入特征高度	16	out_base_addr	输出特征起始地址
6	ch_in	输入通道	17	dat_reuuse	数据复用
7	pad	补零	18	wt_reuuse	权重复用
8	S	步进	19	dat_performace	运算时间
9	Kx	卷积核宽度尺寸	20	quant_truncate	反量化
10	Ky	卷积核高度尺寸			

为便于对卷积模块进行调试，在配置寄存器中定义了四项工作状态寄存器，如上表 4-3 中地址 0 至地址 3，通过访问相应寄存器地址，可以获取卷积模块工作状态。除此之外，还定义了时间计数器，以获取卷积层运算时间，如上表 4-3 中地址 19。

(3)池化电路驱动引擎

下表 4-4 所示为池化运算模块配置寄存器中参数信息，PS 端通过 AXI4 LITE 向该配置寄存器中写入相关池化层配置信息，以驱动池化模块完成相应推理运算。

表 4-4 池化模块配置寄存器

地址	名称	含义	地址	名称	含义
0	start	开始	8	Kx	池化核尺寸
1	done	结束	9	pool_method	池化方式
2	error	出现错误	10	h_out	输出特征高度
3	empty	空闲状态	11	w_out	输出特征宽度
4	w_in	输入特征宽度	12	ch_out	输出通道
5	h_in	输入特征高度	13	dat_base_addr	输入特征起始地址
6	ch_in	输入通道	14	out_base_addr	输出特征起始地址
7	S	步进	15	dat_performace	运算时间

为便于对池化模块进行调试，与卷积电路驱动引擎相似，同样在配置寄存器中定义了四项工作状态寄存器和一项时间计数器，如上表 4-4 中地址 0 至地址 3 为工作状态，地址 15 为池化层运算时间。

4.3 硬件层电路设计

硬件层主要由卷积运算模块、池化运算模块和数据交互通路组成，本节分别对这三部分电路设计过程进行说明。

4.3.1 卷积+激活模块电路设计

在卷积神经网络中卷积计算部分约占整体运算量的 90%以上，对于 CNN 加速器的设计与实现，卷积运算模块是最为重要的环节。在本文卷积运算模块设计中，采用了一种可配置并行度的卷积运算阵列，在硬件配置文件中将卷积运算阵列的并行度(Tk×Tk)作为一种可选项，以便于适应不同 FPGA 芯片中 DSP 资源差异，具备较高

的硬件通用性。

卷积运算电路如下图 4-3 所示，包括参数配置，DMA1 数据传输、中间缓存空间、卷积运算阵列、激活函数和 DMA2 数据传输。通过 DMA1 将权重和特征数据从外部存储器中读取存放到中间缓存空间，当中间缓存空间中数据和权重满足一次乘法运算所需数据时，便开始将数据传输到可配置卷积运算阵列，无需等待全部数据加载到中间缓存空间中，从而减少数据传输过程中的等待时间。

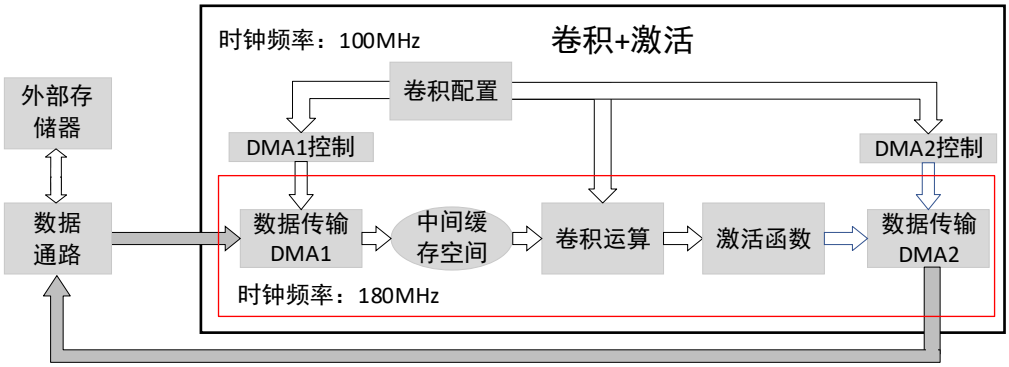


图 4-3 卷积+激活电路设计结构图

为实现上述卷积运算，控制数据流方向，本文设计了四个子模块，包括数据加载子模块、数据控制子模块、数据运算子模块和数据存储子模块。在开始卷积运算之前，权重数据和特征数据按特定排序方式存储在外部存储器中。待数据准备完成后，首先数据加载子模块负责从外部存储器中读取所需权重数据和特征数据到中间缓存空间，如上图中的 DMA1 部分；然后数据控制子模块控制数据从中间缓存空间到卷积运算阵列，如上图中的中间缓存部分；接着数据运算子模块负责卷积乘累加运算，即上图中的卷积+激活；最后数据存储子模块将输出特征数据返回到外部存储器的指定空间，如上图的 DMA2。

4.3.1.1 数据存储方式

为便于卷积运算阵列的并行展开，本文采用了数据分组的思想，将输入特征数据和权重数据按特定的排列方式存放在外部存储器指定空间中。

(1)特征数据在外部存储器中排列顺序

如下图 4-4 所示为输入特征数据在外部存储器中的存储方式，在输入通道方向上将 Tk 个特征数据分为一组。设特征数据的尺寸为 Th_i×Tw_i×2Tk，则经重新排序后，其平面跨步大小为 Tk×Tw_i×Th_i，行跨步为 Tk×Tw_i。

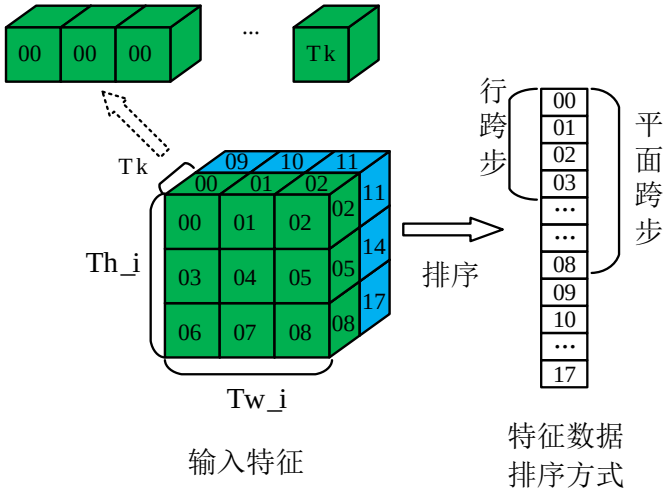


图 4-4 输入特征数据在外部存储空间的排列顺序

(2)权重数据在外部存储器中排列顺序

由于卷积运算中输入权重数据和输入特征数据相互对应，所以同样将输入权重数据按照特定的排序方式存储在外部存储器中。权重数据的存储方式如下图 4-5 所示，在输入通道上将 Tk 个权重数据分为一组，同时为了匹配卷积运算中乘累加阵列的计算方式，将相邻 Tk 组输出通道的权重数据组合到一块，从而便于 Tk 组权重数据同时与一组特征数据进行乘法运算，即卷积运算中乘累加阵列的并行度为 Tk×Tk。

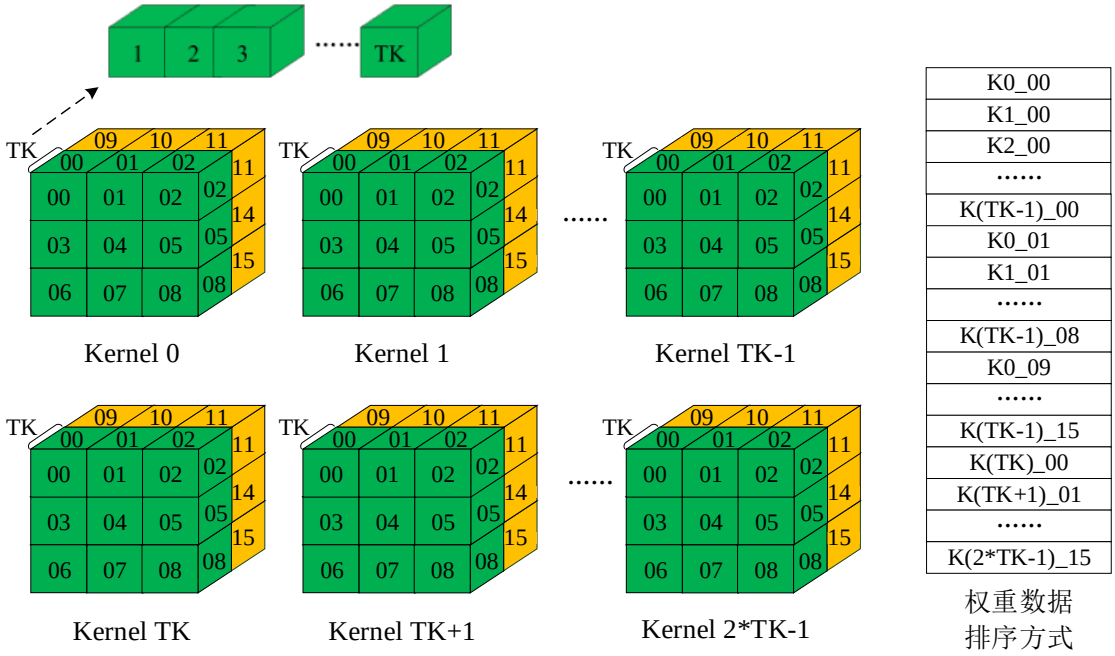


图 4-5 权重数据在外部存储空间的排列顺序

4.3.1.2 数据加载子模块

数据加载子模块由两部分组成：输入权重数据加载到中间缓存空间和输入特征数据加载到中间缓存空间，即下图 4-6 中的 DMA1 部分。

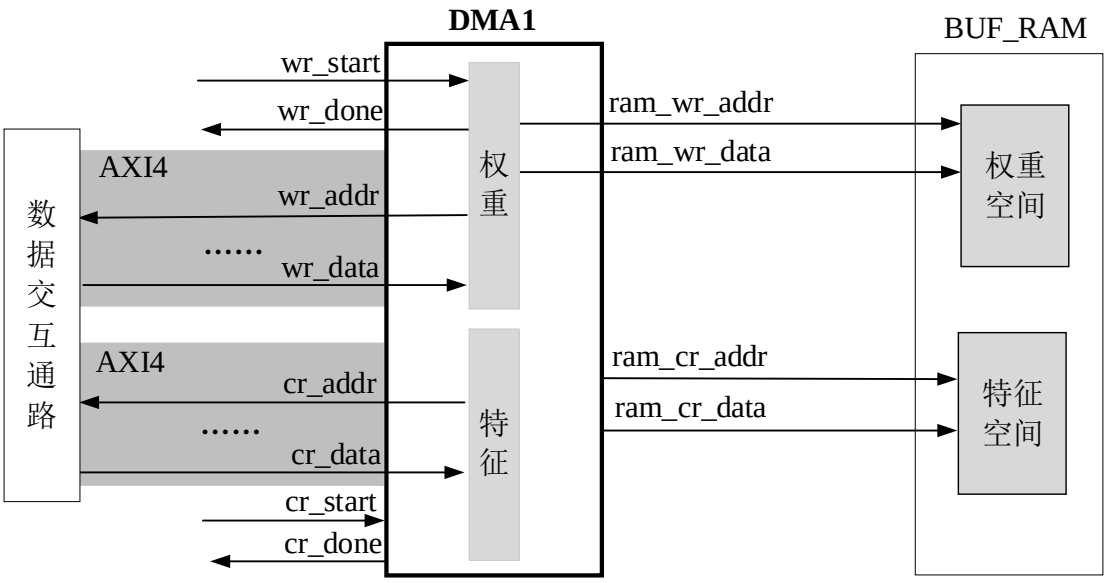


图 4-6 数据加载子模块电路结构

权重数据加载过程如下图 4-7 所示，加速器开始运行后，首先权重数据加载子模块检测 wr_start 信号，待其信号有效后，对 DMA1 进行配置，并将权重数据从外部存储器中搬运到片上的中间缓存空间，待搬运完成后，输出 wr_done 信号。

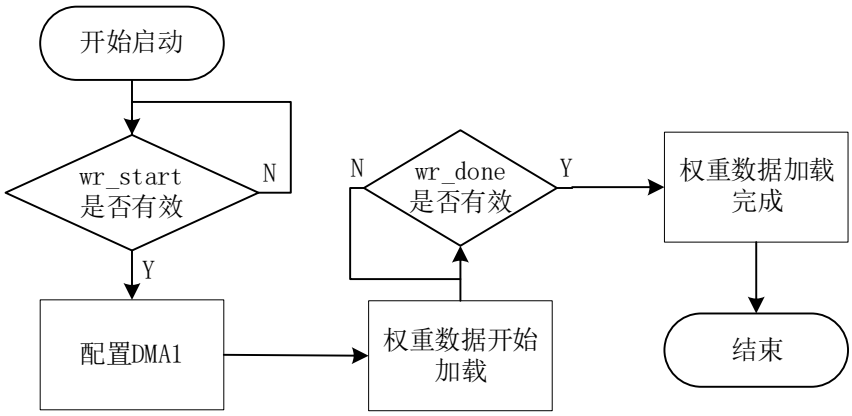


图 4-7 权重数据加载过程流程图

其中数据传输通过数据总线 AXI4，该总线协议包括五个独立通道：写地址通道、写数据通道、写响应通道、读地址通道、读数据通道。考虑到数据加载为数据读取过程，只涉及读地址通道和读数据通道，所以数据加载配置方式如下所述。首先 DMA1 配置读地址通道，并将相关配置参数发送给数据交互通路，待双方握手相应

后，再从外部存储器中读出权重数据并将其搬移到片上的中间缓存空间。

如下图 4-8 所示为读地址伪代码计算公式，其中 wt_size 本次卷积运算权重总大小，则 wt_entries 表示将 Tk 个数据分为一组后，本次权重数据所分总组数；wr_addr 表示读地址通道中地址参数，wr_burst_len 表示读地址通道中突发传输长度；AXI_MAX_BURST_LEN 表示本 AXI4 数据传输过程中最大突发长度；wr_base_addr 表示本次权重空间起始地址。

```
wt_entries = wt_size/(TK*QN/8);

for(k=0;k<wt_entries;k=k+AXI_MAX_BURST_LEN)
        wr_addr = wt_base_addr + k *TK *QN/8；
        if(k+AXI_MAX_BURST_LEN<=wt_entries)
            wr_burst_len = AXI_MAX_BURST_LEN;
        else
            wr_burst_len = wt_entries-AXI_MAX_BURST_LEN;
```

图 4-8 权重数据加载中读地址通道伪代码计算式

从伪代码中可知读地址计算式较长，在高频时钟下将出现不满足建立时间要求的情况。针对此问题，本文提出了划分工作时钟域的解决方案，将该计算式置于低频时钟工作域，其工作时钟频率为 100MHz，如 3.2.1 章节所示。同理，为提升加速器性能，将数据传输过程置于高频时钟域，其工作时钟频率为 180 MHz。

特征数据加载过程和权重数据加载类似，如下图 4-9 为特征数据加载过程中读地址通道的伪代码计算公式。其输入特征图的尺寸为 ch_in×w_in×h_in；cr_addr 为读地址信息；cr_burst_len 为突发长度信息；data_base_addr 为数据起始地址；surface_stride_in 为平面跨步长度；line_stride_in 为行跨步长度。

```
i_ovfl = h_in;
ch_in_div_Tk = (ch_in-1)/TK+1;
k_ovfl = (w_in-1)/AXI_MAX_BURST_LEN+1;

for(i=0;i<i_ovfl;i++)
    for(j=0;j<ch_in_div_Tk;j++)
        for(k=0;k<w_in/AXI_MAX_BURST_LEN;k++)
            cr_addr = data_base_addr+j*surface_stride_in+i*line_stride_in+(k<<(`log2AXI_MAX_BURST_LEN+1))*`Tk;
            if(k<k_ovfl-1)
                cr_burst_len = AXI_MAX_BURST_LEN-1;
            else
                cr_burst_len = w_in-k*AXI_MAX_BURST_LEN-1;
```

图 4-9 特征数据加载中读地址通道伪代码计算式

4.3.1.3 数据控制子模块

数据控制子模块负责控制权重数据和特征数据从中间缓存空间取出，并传输到卷积阵列参与运算，该控制子模块主要通过两个控制信号和一个后反馈信号来控制数据传输：允许权重数据传输信号 wt_go；允许特征数据传输信号 dat_go；后反馈信号 feedback_go。如下图 4-10 所示。

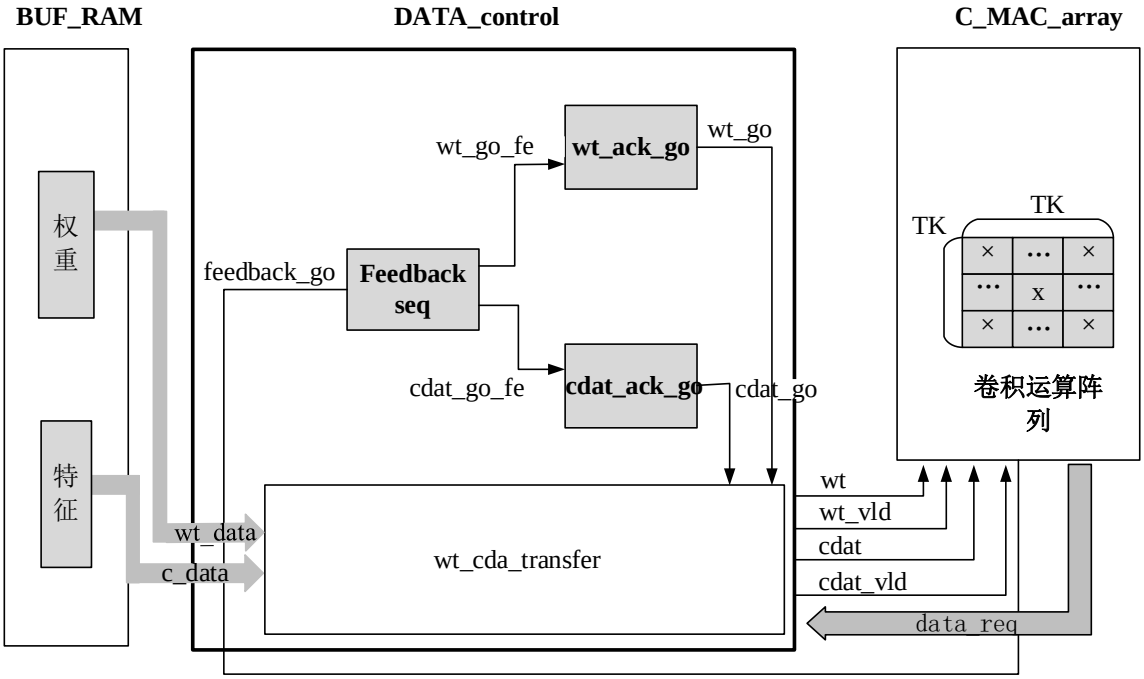


图 4-10 数据控制子模块电路结构

上图中显示了三部分：片上数据缓存空间 BUF_RAM、数据控制中心 DATA_control、卷积运算阵列 C_MAC_array。其中 DATA_control 负责控制将权重数据和特征数据传输到卷积运算阵列，本文电路设计中将 DATA_control 部分划分为两个子模块：数据控制子模块和数据传输子模块 wt_cda_transfer。其中数据控制子模块又划分为更小的三个子模块：权重数据传输控制 wt_ack_go、特征数据传输控制 cdat_ack_go、后反馈控制 feedback_seq。

(1)wt_ack_go 子模块设计

本文乘累加阵列运算方式如下图 4-11 所示，在一个时钟周期内，将一组输入特征数据与 Tk 组输入权重数据同时做乘累加运算，得到一组输出特征数据。

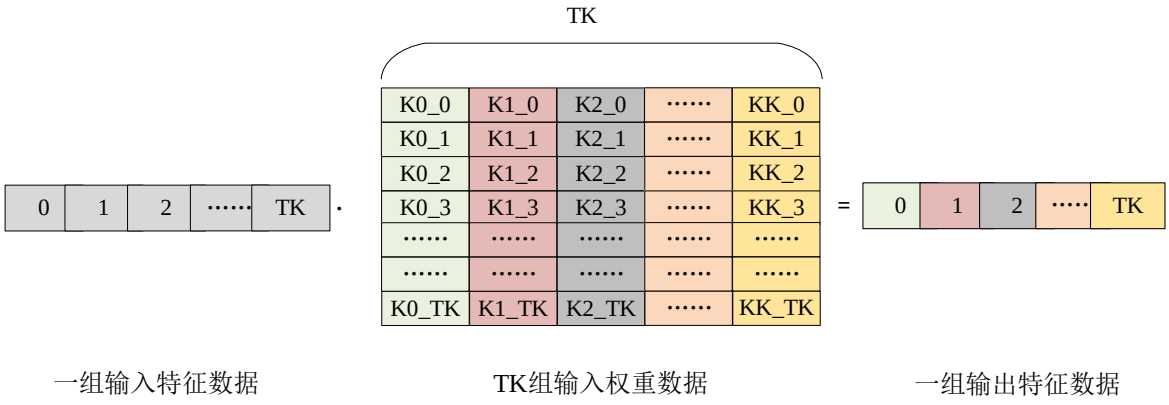


图 4-11 乘累加阵列运算方式

为实现上述卷积运算，需要在一个时钟周期内准备一组输入特征数据和 Tk 组输出特征数据。然而经数据分组后，由于中间缓存空间中每个地址中包含一组数据，所以单时钟周期内仅能读取一组数据，无法同时准备 Tk 组权重数据。针对此问题，采用了数据流水化的方式，即新增了两组寄存器缓存，每组寄存器内包含 Tk 组权重数据，当一组权重数据使用完后直接换下一组缓存寄存器内数据。如下图 4-12 所示。

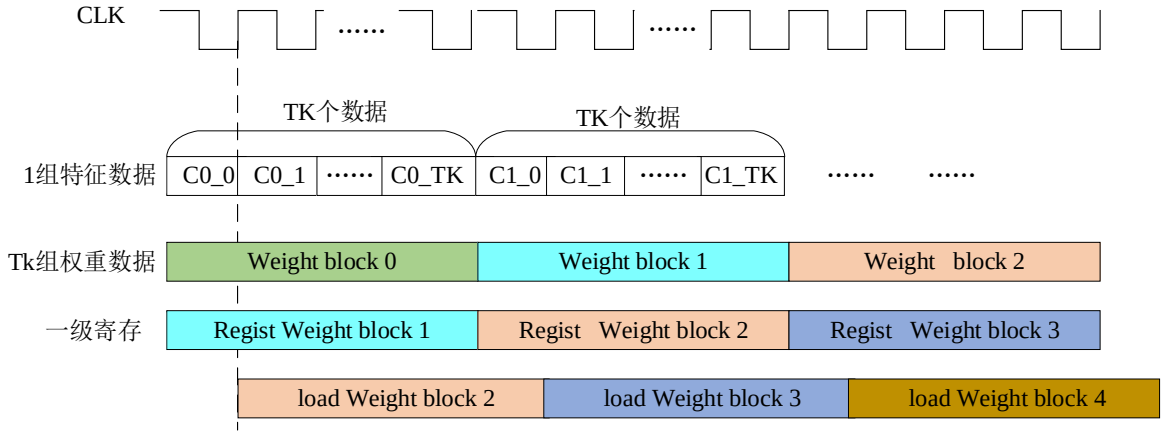


图 4-12 数据传输时序图

上图中 weight block 是由 Tk 组权重数据组成的，从时序图中可看出消耗一个 weight block，需要用掉 Tk 组特征数据，这是因为在输出特征图宽度方向上进行了数据分条，将 Tk 个输出特征数据分为一条，卷积运算中其输出特征遍历方向先按照输出宽度方向，从而使得复用 Tk 组权重数据的时间内遍历 Tk 组特征数据，优化了电路时序，从而解决了上述问题。

由于卷积运算中权重数据需要与特征数据相匹配，为实现权重数据按需传输，设计了 wt_go 信号对权重数据传输进行控制，具体控制过程如下图 4-13 所示。

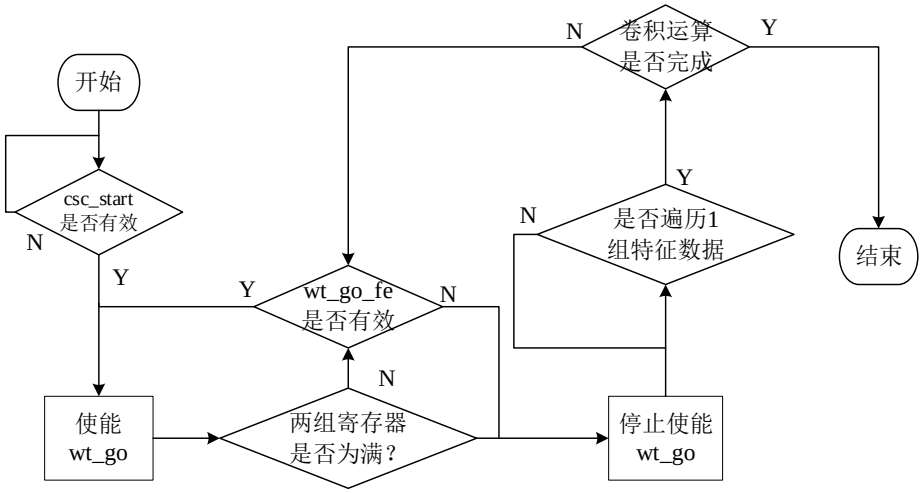


图 4-13 卷积运算中权重数据控制流程图

加速器开始运行后，首先检测 csc_start 信号，待其信号有效后，开始使能 wt_go 信号，允许从中间缓存空间读取权重数据并将 Tk 组权重数据合并在一起缓存到二级寄存器，待两组权重寄存器都有数据后，停止使能 wt_go 信号，然后遍历 1 组特征数据，待遍历完成后，重新使能 wt_go，并将一级寄存器数据作为新的 Tk 组权重数据，在这个过程中还需检测后反馈信号 wt_go_fe，当 wt_go_fe 无效时，表示此时不允许从中间缓存空间读取权重数据，则将 wt_go 信号置为低电平，重复上述操作，等待卷积运算结束。

(2)cdat_ack_go 子模块设计

由上图 4-13 可得，只有两组权重寄存器处于非全空状态时，才允许从中间缓存空间读取特征数据，否则停止特征数据读取。然而特征数据的读取除受权重寄存器限制外，还受后反馈信号 dat_go_fe 影响，与 wt_go 信号相似，当 dat_go_fe 信号无效时，停止特征数读取。其卷积运算中特征数据控制流程图如下图 4-14 所示。

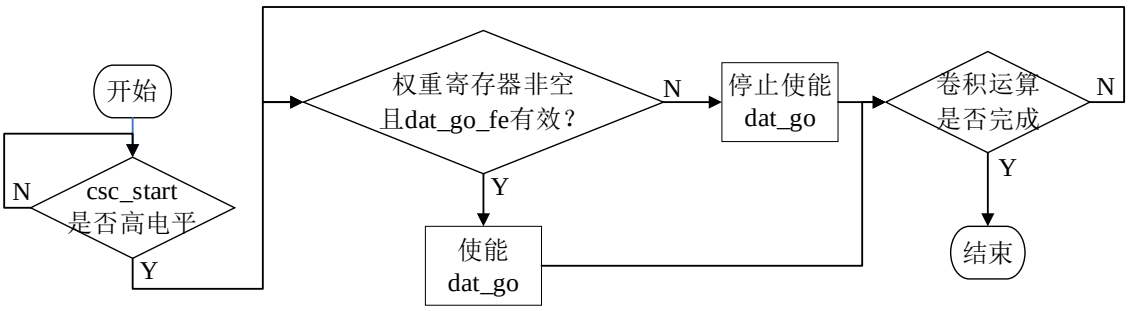


图 4-14 卷积运算中特征数据控制流程图

(3)feedback_seq 子模块设计

由于在电路设计中为充分利用 AXI4 通道，减少资源使用量，将所有对外部存储器读写的操作都集中到数据交互通路，共用一条 AXI4 总线，所以需要将输出特征数据暂存到 FIFO 空间，然后发送写数据请求到数据交互通路，待请求被仲裁后，再通过数据交互通路将 FIFO 空间数据存储到外部存储器指定空间。然而 FIFO 空间深度是固定的，当 FIFO 中数据存满时需要及时停止输入特征数据和输入权重数据读取，以中断卷积运算。为此，将 FIFO 的空满状态作为后反馈信号 feedback_go 返回到数据控制中心，其流程图如下图 4-15 所示。

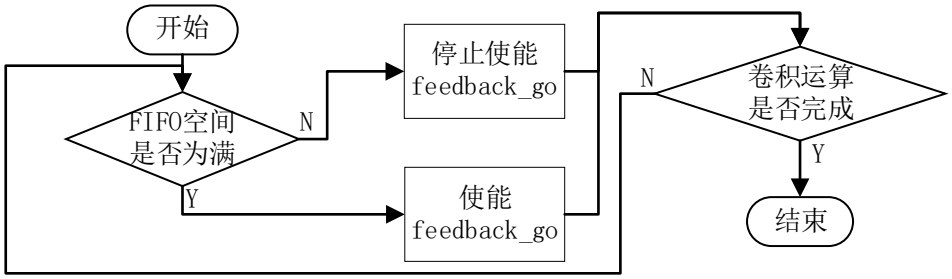


图 4-15 卷积运算中后反馈控制信号控制流程图

4.3.1.4 数据运算子模块

数据运算模块的核心是乘累加阵列，如下图 4-16 所示为数据运算子模块电路结构，其包含四个子模块：数据请求子模块(CSC)、乘累加阵列子模块、激活函数子模块、输出特征缓存子模块(FIFO_COUT)。

其中，CSC 子模块负责发送读取特征数据的请求信号到片上缓存，读取特征数据请求中包含下一时刻特征数据所在的行、列、通道、分组地址等信息，输入权重请求包含下一时刻权重数据所对应的输入通道、输出通道、分组地址等信息；乘累加阵列子模块将读取的一组输入特征数据和 Tk 组权重数据做矩阵乘法，其中一组数据内包含 Tk 个数据，所以乘累加阵列的并行度为 Tk×Tk，即一个时钟内可以做 Tk×Tk 次乘累加运算；激活函数子模块负责对最终乘累加结果进行非线性激活，在该模块中只设计了 RELU 激活方式，即当输出结果为负数时取零值，为正数和零时数值不变。FIFO_COUT 子模块负责临时存储输出特征的数据，并通过后续数据交互模块将 FIFO 中数据存储到外部存储器指定输出特征空间，然而数据交互模块未响应到写数据请求时，可能出现 FIFO 存满的现象，为避免出现 FIFO 空间存满时乘累加阵列一

直工作导致输出特征数据丢失，本文设计中将 FIFO 将满状态作为后反馈信号传递到数据控制子模块，通过停止数据传输的方法及时中断乘累加运算，保证输出特征数据不丢失。

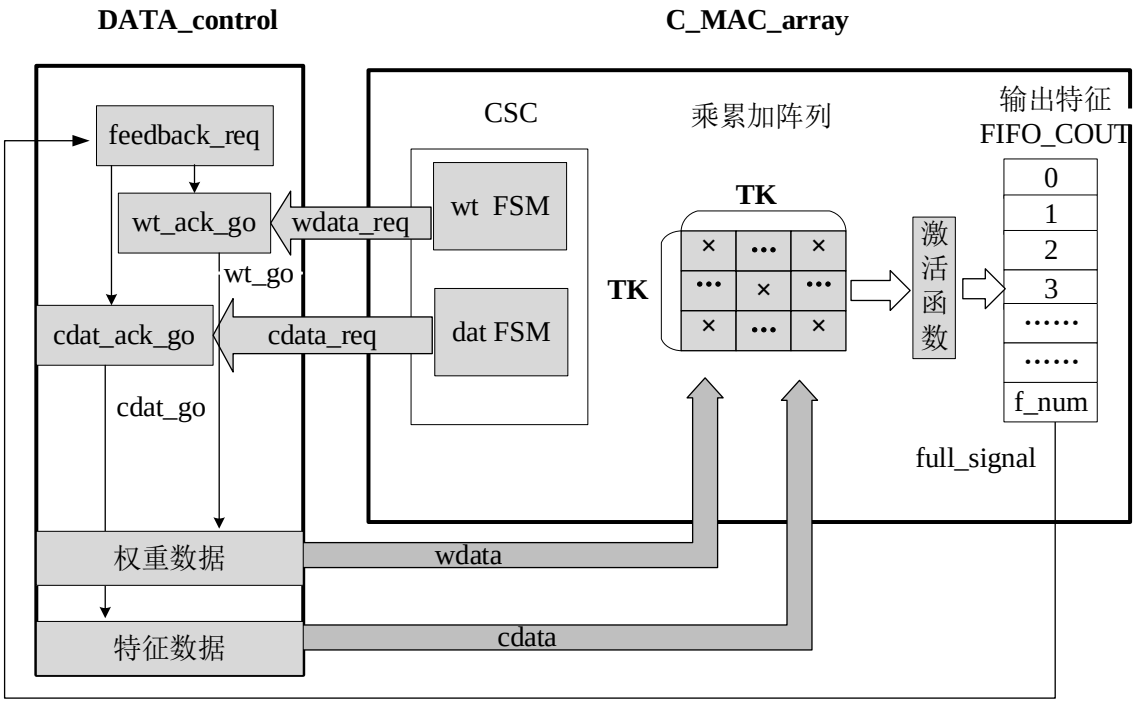


图 4-16 数据运算子模块电路结构

以上四个子模块中，由于激活函数采用 RELU 方式，只是简单与零做对比取值，不做详细展开。其次，FIFO 采用同步时钟，也不进行详细展开，仅对数据请求子模块 CSC 和乘累加阵列子模块详细展开，其具体计算过程如下所示。

(1)CSC 子模块

CSC 子模块中分为权重数据请求和特征数据请求，在 CSC 子运算运行过程中模块，首先计算下一时刻所需数据在 BRAM 中所在地址，然后从 BRAM 中读取相应数据，并将该数据传输给乘累加阵列参与运算。

如下图 4-17 为下一时刻所需的输入特征数据在 BRAM 中所处地址的伪代码计算公式。式中 pixel_in 为平面跨步长度，h 为目标数据在特征矩阵中所处行值，w 为目标数据在特征矩阵中所处列值，buf-data_addr 为目标数据在中间缓存空间中的所处地址。

```
pixel_in=h_in*w_in;

for(k=0;k<ch_out;k=k+Tk)
   for(pp=0;pp<h_out*w_out;pp=pp+Tk)
       for(c=0;c<ch_in;c=c+Tk)
           for(r=0;r<ky;r=r+1)
              for(s=0;s<kx;s=s+1)
                  if (pp<h_out*w_out)
                   {
                     h = pp/w_out;
                     w = pp %w_out;
                     buf_dat_addr=pixel_in*c/TK+h*w_in+w;
                    }
```

图 4-17 输入特征数据在 BRAM 中所处地址的伪代码计算式

权重数据请求方式与特征数据类似，都是先计算目标数据在 BRAM 存储空间的地址，再从 BRAM 中读取相应数据。下图 4-18 为下一时刻所需权重数据在 BRAM 中所处地址的伪代码计算式，其中 buf_w_addr 为下一时刻卷积运算所需权重数据在 BRAM 中的地址。

```
pixel_in=h_in*w_in;
pixel_out=h_out*w_out;
w_size_in=kx*ky*ch_in;

for(m=0;m<pixel_out;m=m+TK)
   buf_w_addr=0;
   for(kk=0;kk<ch_out;kk=kk+Tk)
       for(w_level=0;w_level<w_size_in;w_level+TK)
           for(n=0;n<TK;n=n+1  )
               buf_w_addr=buf_w_addr+1;
```

图 4-18 输入权重数据在 BRAM 中所处地址的伪代码计算式

(2)乘累加阵列子模块

常规卷积运算方式在上文已经说明，由于常规运算中不涉及并行运算，无法发挥 FPGA 并行计算能力，为最大化利用硬件资源，本文设计了一种可配置乘累加阵列。首先在通道方向上将输入特征通道、输出特征通道进行分组，每组内各包含 Tk 个数据，然后在一个时钟周期内将一组特征数据分别与 Tk 组权重数据相乘，即乘累加阵列并行度为 Tk×Tk。

由于拼接 Tk 组权重数据需要 Tk 个时钟，为充分利用这段时间，在输出特征宽度方向上进行分块，将相邻 Tk 组输出数据分为一块，在这 Tk 个时钟内按顺序计算这 Tk 个输出数据。本文卷积运算伪代码如下图 4-19 所示。

```
for(k=0;k<ch_out;k=k+Tk)
   for(pp=0;pp<h_out*w_out;pp=pp+Tk)
      {
        sum=0;
        for(c=0;c<ch_in;c=c+Tk)
           for(r=0;r<ky;r=r+1)
              for(s=0;s<kx;s=s+1)
                 {
                   if (pp<h_out*w_out)
                    {
                      h = pp/w_out;
                      w = pp %w_out;
                      for(kk=k,kk<k+Tk,kk=kk+1)
                        for(cc=c,cc<c+Tk,cc=cc+1)
                           sum=sum+x[h*Sy-Pad_up+r][w*Sx-Pad_left+s][cc]*wt[r][s][c][kk];
                    }
                 }
           out[h][w][kk]=sum;
      }
```

图 4-19 本文卷积运算伪代码

4.3.1.5 数据存储子模块

数据存储子模块负责将输出特征数据搬运到外部存储器指定空间中，如下图 4-20 中的 DMA2 子模块。首先，DMA2 检测 FIF0 空满信号 f_full 和 f_empty，当 FIFO 非空时将开始配置 AXI4 写通道参数，并将写请求发送到数据交互模块。然后，等待数据交互模块对输入请求进行仲裁，待得到数据交互通路相应后，DMA2 发送读使能 f_rd_en 到 FIFO 模块，从 FIFO 中读取输出特征数据 f_rd_data，并将其通过数据交互通路搬移到外部存储器指定空间。

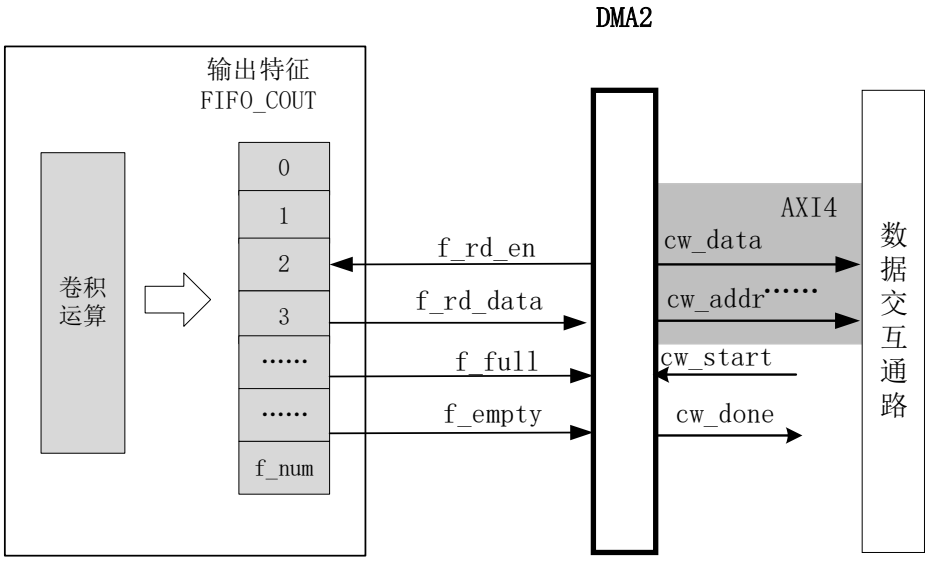


图 4-20 数据存储子模块的电路结构

4.3.2 池化模块电路设计

如下图 4-21 为池化模块电路设计架构图。其数据流向与卷积运算模块相似，区别在于片上缓存类型，将存储类型由 BRAM 调整为 FIFO，这是由于池化运算中输入特征数据只使用一次，不涉及数据复用，而 FIFO 中数据先入先出，正好与之匹配。

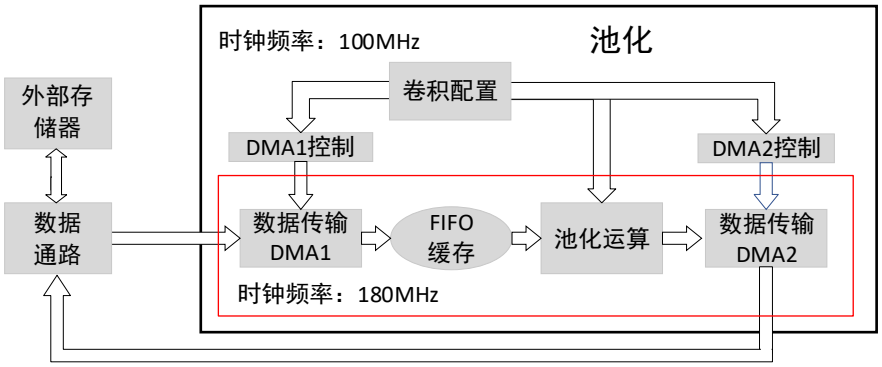


图 4-21 池化模块电路设计结构图

池化运算具体实现过程如下图 4-22 所示，其可分为四部分：行池化、行数据缓存 FIFO1、列池化和输出数据缓存 FIFO2。先对行数据进行池化运算，并将结果暂存在 FIFO1 中，当 FIFO1 中存满一行时，再将当前行池化输出结果与 FIFO1 中的数据进行列池化运算，最终输出结果暂存在 FIFO2 中。

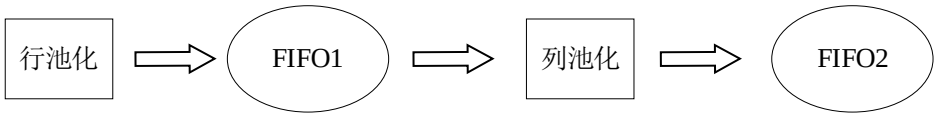


图 4-22 池化运算

如下图 4-23 所示为池化核尺寸为 2×2、池化类型为最大池化、输入特征尺寸为 8×8 时的池化模块计算示意图。

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	24	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

行池化

1	3	5	7
9	11	13	15
17	19	24	23
25	27	29	31
33	35	37	39
41	43	45	47
49	51	53	55
57	59	61	63

列池化

9	11	13	15
25	27	29	31
41	43	45	47
57	59	61	63

图 4-23 池化模块运算示意图

由于列池化运算时需要用上一行的数据与本行做运算，所以需要将上一行数据

寄存到 FIFO1 中，如上述输入尺寸为 8×8 时，池化核大小选择 2×2 时，行池化后需要缓存 4 组数据，即 FIFO1 的地址深度为 4，地址宽度为 DN×Tk。当输入特征尺寸较小时，FIFO1 所用资源较小，然而当输入尺寸较大时，必须用较大 FIFO1 来缓存数据，这将消耗较多的资源。

针对上述问题，采用了列分块的方法，其分块方式如下图 4-24 所示，按输出特征数据宽度方向进行分块，每块宽度大小为 RK。当输入特征尺寸为 224×224，池化核尺寸选择 2×2 时，RK 选取 16 时，则将其分为 7 块，每次只需缓存 16 组数据，与改进之前每次缓存 112 组数据相比，FIFO1 所用资源降低了 7 倍。

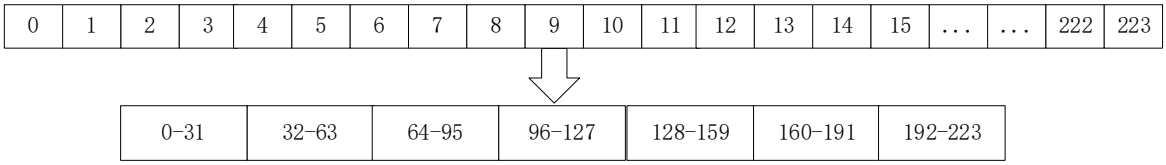
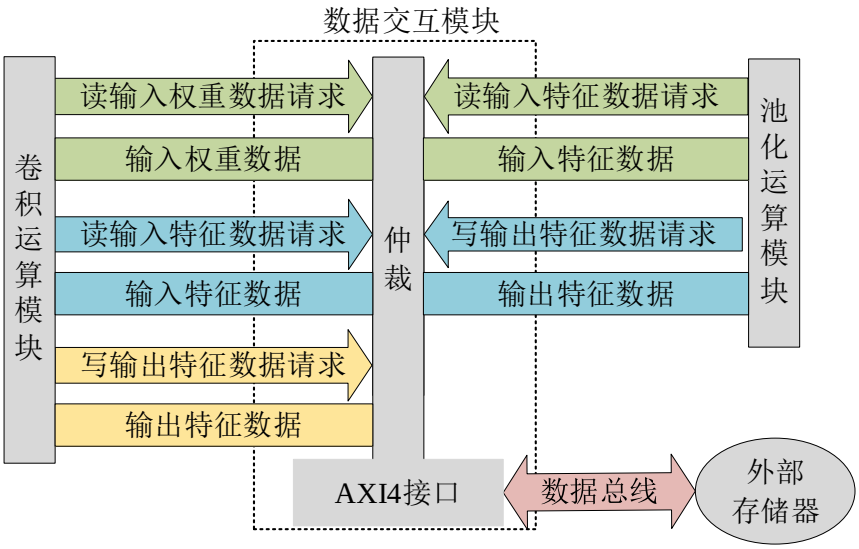


图 4-24 池化模块列分块方式示意图

4.3.3 数据交互通路设计

本设计中数据交互模块由输入仲裁和输出 AXI4 接口组成，如下图 4-25 所示，对来自卷积运算模块的读权重请求信号、读特征请求信号、写特征请求信号和来自池化运算模块的读输入特征请求信号、写输出特征数据请求信号进行仲裁，最终仲裁目标通过数据总线实现与外部存储器的数据交互。



4-25 数据交互通路电路设计结构图

4.4 硬件层电路仿真

本文基于 RTL 级描述语言搭建了通用 CNN 加速器，并在 VIVADO 2021.1 上完成了逻辑功能仿真。由于卷积神经网络中主要运算模块为卷积和池化，所以本节主要对这两个功能模块进行验证。

4.4.1 卷积+激活模块电路仿真

由于 VIVADO 自带的仿真工具性能有限，在电路仿真过程中，无法加载大尺寸输入特征图，所以仿真过程中选用小尺寸输入特征图。除此之外，考虑到仿真只为验证逻辑功能正确性，不需要真实输入图片，所以在仿真过程中模拟了尺寸为 60×60×32 的输入特征图，并将卷积运算中的输出通道设为 64，即验证当输入尺寸为 60×60×32、输出尺寸为 60×60×64 时，卷积+激活运算模块逻辑功能的正确性。本节仿真过程中硬件配置参数 Tk 选取 32，数据位宽选取 8。

由于卷积电路设计过程中采取了模块化的思想，按数据流向将卷积+运算模块分为四个子模块：数据加载子模块(DMA1)、数据控制子模块(DATA_CONTROL)、数据运算子模块(C_MAC_ARRAY)和数据存储子模块(DMA2)，所以为验证整体逻辑功能的正确性，需要先对这四个子模块进行测试，其次还需验证外部配置接口 AXI4 LITE 电路的正确性。待子模块均无误后，再对整体进行仿真，即将最终仿真输出结果与真实卷积结果对比，并将对比结果打印出来，通过打印窗口信息可以验证卷积运算模块逻辑功能的正确与否。

4.4.1.1 AXI4 LITE 配置接口仿真验证

本文 CNN 加速器采用软硬件结合的架构，PS 端通过 AXI4 LITE 控制总线将配置信息发送给运算模块，如下图 4-26 为卷积模块中对配置接口的仿真波形图。

从仿真图中可以看出 AXI4 LITE 的写通道(aw)发出数据后，波形图下方的 lite_reg 中配置寄存器的值陆续发生改变，经检测从机端接受数据正常且数值无误，如图中代表输入特征图宽度的寄存器(w_in_r)值为 O3C，将 16 进制转化为 10 进制为 60，与本文选择的特征图宽度 60 相符合。通过波形变化和 lite_reg 中数值变化可以验证 AXI4 LITE 接口的正确性。

正常传输时，当写通道的数据传输到卷积运算模块后，待主从机握手交互后，配置寄存器 lite_reg 所代表的值会在下时刻发生改变，而图中从 AXI4 LITE 写通道发

出数据到配置寄存器发生改变的期间经历 3 个时钟，这是因为文本在电路设计时为优化电路时序，将输入数据寄存了两拍，以保证从机接收的数据无误。

图 4-26 AXI4 LITE 寄存器仿真波形图

当卷积运算模块所需参数均被配置完成后，PS 端才会发送卷积运算开始信号，如下图 4-27 所示为卷积运算模块工作状态信号。当接收到开始信号(start_signal)后，四个子模块的工作状态(working_signal)会在下一时钟由低电平变为高电平，表明当前开始工作；当子模块完成工作后，会发出结束信号(done_signal)到 AXI4 LITE 接口，在检测子模块的结束信号后，相应子模块的 working_signal 信号会由高电平转为低电平，并将配置寄存器中的 done_signal 信号置为高电平，由表明当前子模块运算完成；当四个子模块均运算完成后，则本层卷积运算完成。

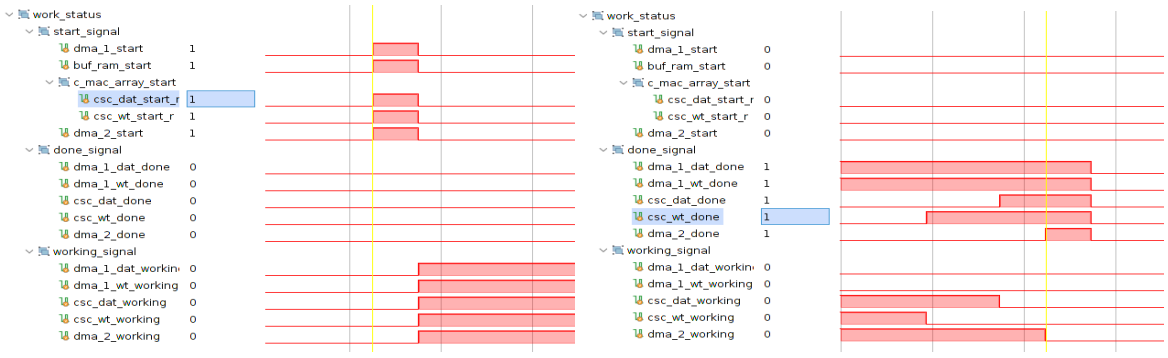


图 4-27 卷积模块工作状态

通过分析上述仿真波形可以看出 AXI4 LITE 接口数据传输和工作状态控制寄存器无误，验证了 AXI4 LITE 接口电路逻辑功能的正确性

4.4.1.2 数据加载子模块仿真验证

数据加载子模块负责将特征数据和权重数据从外部存储器中读出并搬移到片上缓存空间，即卷积模块电路架构图中的 DMA1。由于数据加载过程只用到 DMA1 的读数据通道，所以在本模块最重要的是对 DMA1 中 AXI4 读地址通道的配置，如下图 4-28 为输入特征数据加载中读地址配置(dma1_rd_config)波形图，其中地址计算(for_calculate_addr)部分通过三层循环计算所需输入特征在外部存储器中的地址(cmd_addr)。

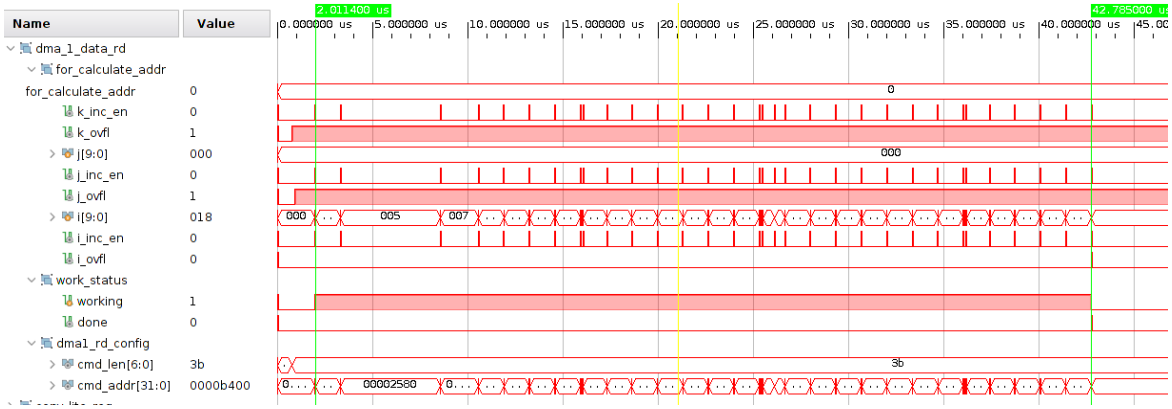


图 4-28 特征数据加载子模块核心信号仿真波形图

可以看出上图中宽度方向计满标志信号 k_ovfl、输入通道方向计满标志信号 j_ovfl 在工作期间和恒处于高电平状态，这是由于卷积运算模块仿真时模拟的输入特征图尺寸为 60×60×32，根据 AXI4 发送一次读地址后，最大可以连续读取 4KB 的数据，所以当选取分组数 Tk 为 32、数据位宽为 8 时，突发长度最大为 4096/32=128，128 大于输入特征图尺寸 60，所以在 DMA1 中每次突发长度设为 60-1=59，即发送一次读地址，可以搬运 1×60×32=1920 个数据，相邻读地址间的跨度为 1920。将上述中理论突发长度和地址跨度转化为 16 进制后与上图相符，这解释了 k_ovfl、j_ovfl 在工作期间一直处于高电平的原因，同时也间接验证了 DMA1 中逻辑功能的正确性。

4.4.1.3 数据控制子模块仿真验证

数据控制子模块负责响应卷积运算阵列数据请求，并控制权重数据和特征数据从片上缓存空间中读取。在数据控制子模块中，定义了三个控制信号：允许权重数据从片上缓存空间读取信号 wt_go、允许特征数据从片上缓存空间读取信号 cdat_go 和后反馈信号 feedback_go。三个信号的控制波形图如下图 4-29 所示。

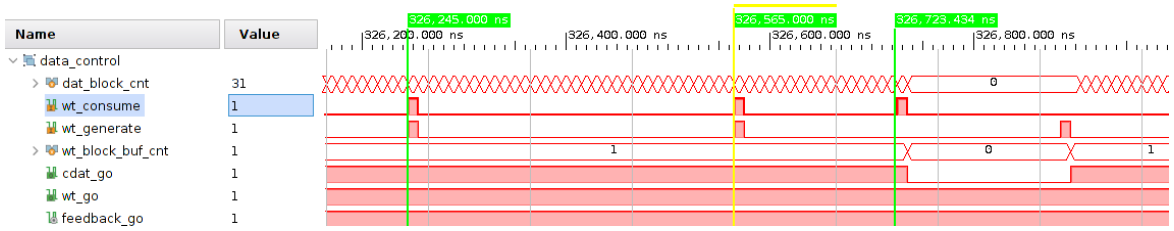


图 4-29 数据控制子模块核心信号仿真波形图

从图中可以看出，正常工作时，消耗一块权重数据的同时需要消耗 32 块特征数据，即相邻 wt_consume 脉冲信号期间，特征块消耗值 dat_block_cnt 从 0 计数到 31，这是由于 Tk 选取为 32。而 wt_block_buf_cnt 在上述波形图前半部分保持不变，后半部分从 1 减为 0，这是由于产生一块权重块最少需要 Tk 个时钟周期，在波形图前半部分权重块消耗速度与产生速度一致，则 wt_block_buf_cnt 保持不变，而波形图后半部分，当输入特征最后剩余数据小于 Tk 时，会导致消耗一组权重块的时间内无法产生一组新的权重数据，只能等待新的权重块产生后再允许特征数据块读取，当权重寄存空间为空时，导致 cdata_go 从高电平变为低电平。这很好的解释了图中波形变化，同时也验证了数据控制子模块电路控制逻辑的正确性。

4.4.1.4 数据运算子模块

由于在卷积运算中将 Tk 组权重数据拼接在一个寄存器中，数据位宽太大，导致无法直观的从图 4-31 中验证数据运算子模块的正确性，针对这一问题，从数据位宽的角度侧面验证数据运算子模块逻辑功能的正确性。

如下图 4-30 所示，当 Tk 为 32、数据位宽为 8 时，将一组特征数据拼接后位宽为 32×8=256，Tk 组权重数据拼接在一起后数据位宽为 256×32=8192，则一组特征数据与 Tk 组权重数据做矩阵相乘，则产生 Tk 个输出结果，每个输出结果位宽为 8+8+log2Tk=21，将 Tk 个输出结果拼接在一起后数据位宽为 21×32=672。上述理论数据位宽与图中所示一致，间接的验证了数据运算子模块逻辑功能的正确性。

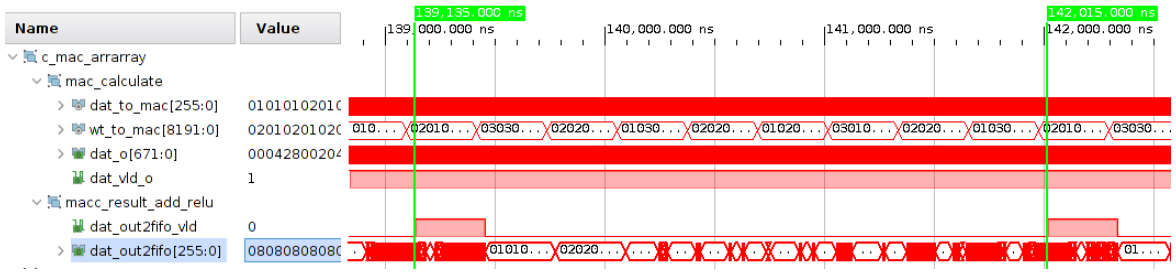


图 4-30 数据运算子模块核心信号仿真波形图

4.4.1.5 数据存储子模块

数据存储子模块负责将卷积运算的输出特征数据搬移到外部存储器指定空间中，即卷积模块电路设计架构中的 DMA2。由于 DMA2 只负责将数据写入到外部存储器指定空间，所以只需配置 DMA2 中 AXI4 的写数据通道，在这个过程中最为重要的是对写地址的计算。

如下图 4-31 所示，当检测到 dma_2_start 脉冲信号后，DMA2 开始工作，外部存储器中输出特征空间的起始地址为 0X40000000，两次相邻 AXI4 写地址跨度大小为 0X780，这是因为在输出特征图尺寸为 60×60×64，当 Tk 为 32、数据位宽为 8 时，由于受 AXI4 不可跨越 4K 边界的限制，所以最大突发传输长度为 4096/32=128，而 128 大于输出特征图宽度 60，所以将 AXI4 突发传输长度 dma_2_wr_length 设为 60-1=59，转化为 16 进制为 0X3b，与图中波形相符。与此同时，当突发传输长度为 60 时，AXI4 相邻两次写地址间跨度为 60×32=1920，转化为 16 进制为 0X780，与下图中相符，从而验证了数据存储子模块逻辑功能的正确性。

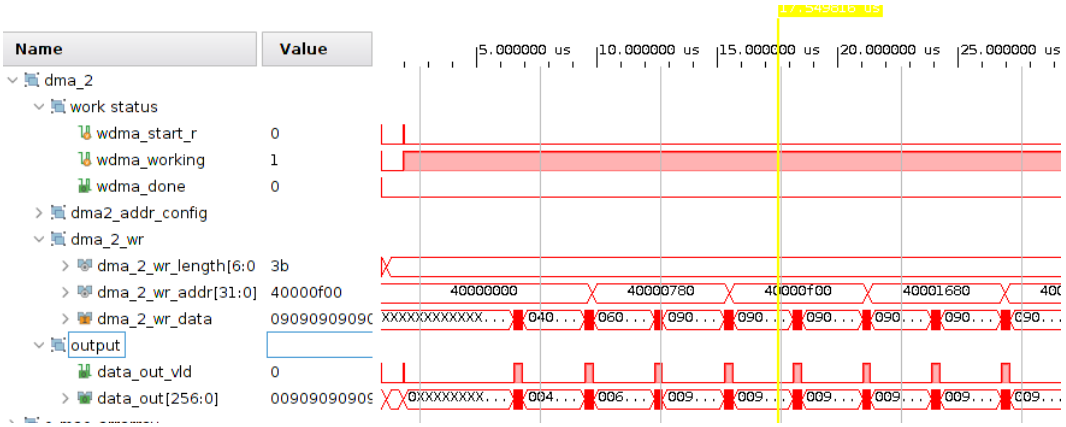


图 4-31 数据存储子模块核心信号仿真波形图

4.4.1.6 最终仿真结果

首先通过仿真验证上述子模块的正确性，待验证无误后，再对整体卷积模块逻辑功能进行仿真验证，如下图 4-32 所示为最终打印结果，打印信息中显示每次硬件仿真输出结果与电脑端理论输出结果，其中 data out 为硬件仿真结果，data out soft 为真实结果值，首先将真实值与仿真输出结果对比，当所有输出结果完全一致值时，打印“result match”，否则打印“result dismatch”。

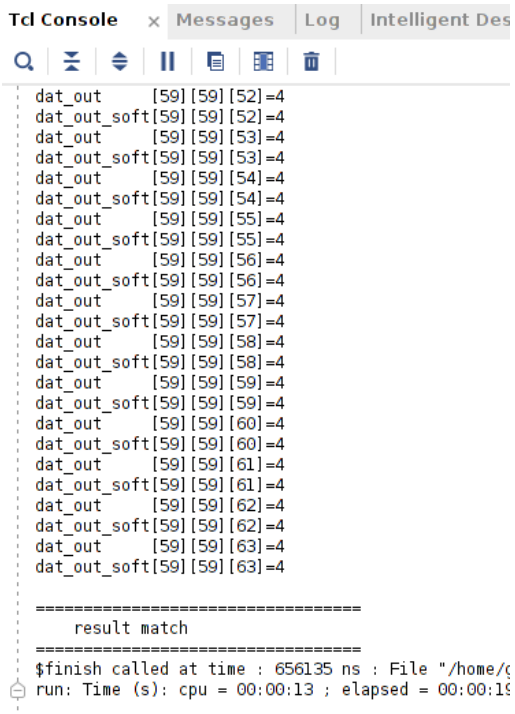


图 4-32 卷积运算模块最终打印信息

从图中可以看出输出特征数据与真实值结果一致，最终对比结果为“result match”，这验证了卷积+激活模块逻辑功能的正确性。

4.4.2 池化模块电路仿真

池化运算模块的电路设计架构与卷积运算模块相似，相同部分是都使用了数据加载子模块和数据控制子模块，不同的是池化模块中核心运算单元是行池化和列池化，而卷积运算模块中核心运算单元为卷积运算阵列。为验证池化模块逻辑功能的正确性，需要对其核心运算子模块进行验证。

池化按类型可分为最大池化、最小池化和平均池化，最大池化即选取池化核范围内的最大值，由于其计算方式简单，且不需要加载权重数据，所以 VIVADO 自带的仿真工具能够对大尺寸特征图进行仿真。考虑到本文最终上板实验中选取的卷积神经网络为 VGG16，所以在池化中对 VGG16 的第一个池化层进行模拟仿真，即模拟输入尺寸为 224×224×64 时的池化运算。硬件配置参数 Tk 选取为 32、数据位宽为 8。以下将分别验证行池化子模块和列池化子模块，待无误后，再对整体池化模块逻辑功能验证，将仿真输出结果与真实值进行对比，并将对比结果通过显示窗口打印出来。

(1)行池化子模块

行池化是指先对每行中相邻池化核宽度范围内数据做池化操作，并将结果输出到下一级，如下图 4-33 所示为池化核为 2×2、池化类型为最大池化时行池化波形图。

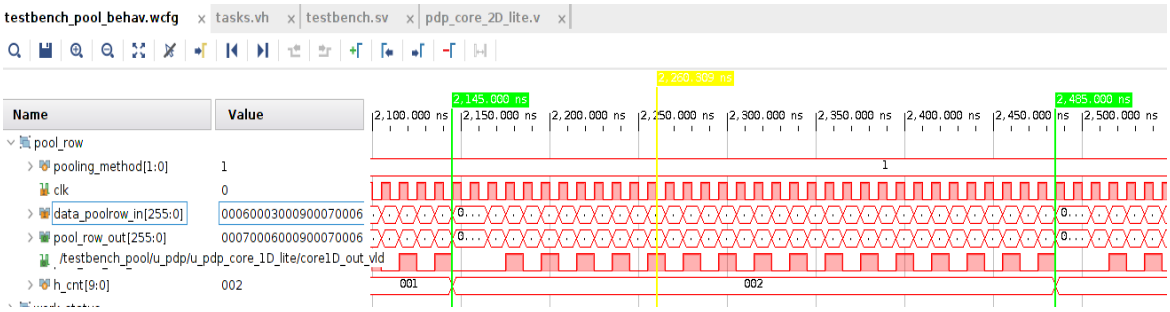


图 4-33 行池化子模块核心信号仿真波形图

从图中可以看出当前 value 区当前行池化输出数据 pool_row_out 大于当前输入数据，这是由于上一次暂存的数据大于当前数据，从而将上一次数据作为最大值输出，验证了行池化子模块逻辑功能的正确性。

除此之外，从图中还可以看出行状态 h_cnt 为 2 期间，一共产生了 16 个输出结果，这是由于从行池化结果输出到列池化参与运算的过程中，需要暂时缓存一行数据，为减少数据缓存空间，文本在池化模块设计时按输出特征图宽度方向分块，每块宽度尺寸 RK 选取为 16，这与图中 h_cnt 为 2 期间产生 16 个输出结果相符，验证了电路设计中分块逻辑的正确性。

(2)列池化子模块

列池化运算与行池化类似，如下图 4-34 为列池化子模块核心信号仿真波形图，分析方式与行池化相同，从图中可以看出总共缓存 16 个行池化运算结果，将当前输入数据 pool_row_out 与缓存数据 op_value_reg 对比，并将最大值作为输出值 pool_col_out，从图中 value 区中 pool_col_out 大于 pool_row_out，是由于上一行数据大于本行数据，所以将上一行数据作为最大值输出，其验证了列池化子模块逻辑功能的正确性。

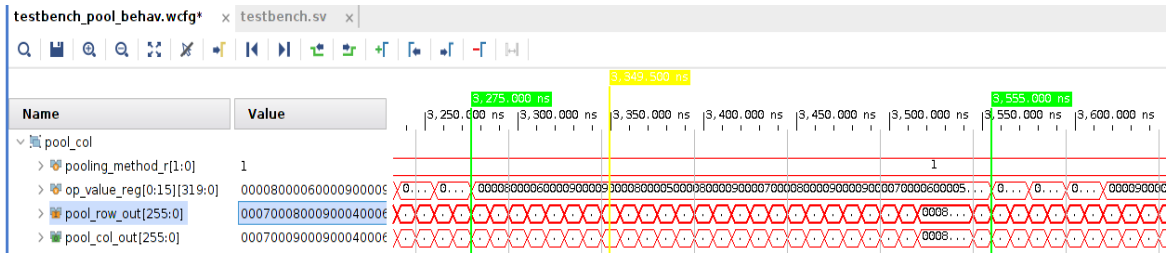


图 4-34 列池化子模块核心信号仿真波形图

(3)整体仿真结果

首先验证行池化和列池化子模块逻辑功能的正确性，待仿真验证无误后，对整体池化模块逻辑功能进行仿真验证，如下图 4-35 所示为池化模块最终仿真打印信息，从图中打印结果“result match”可知，本文设计的池化运算模块逻辑电路无误。

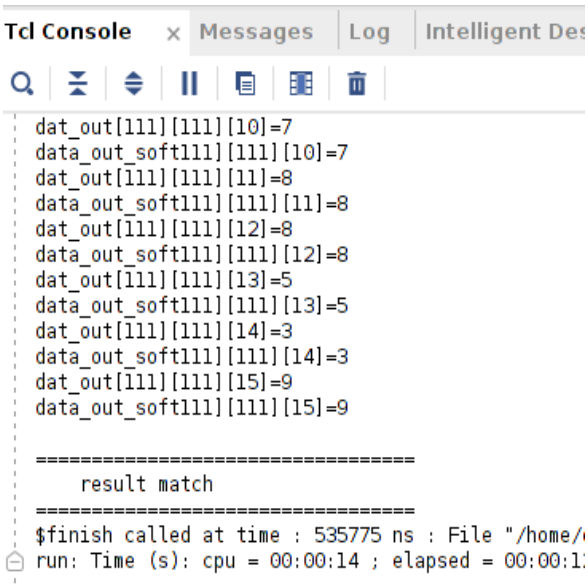


图 4-35 配置方式一时池化运算模块最终打印信息

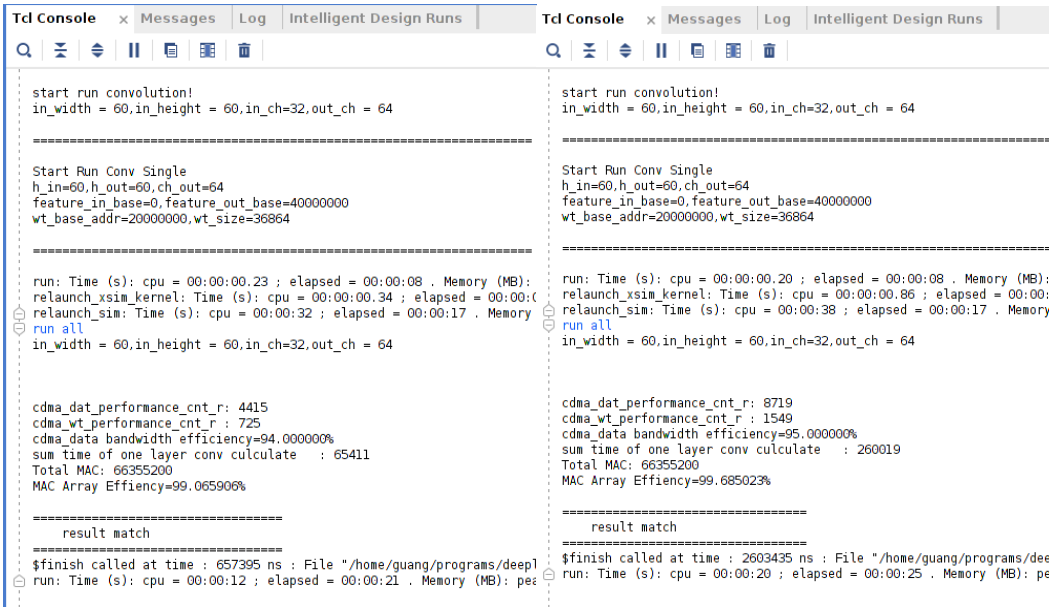
4.4.3 多配置方式的电路仿真

为实现 CNN 加速器资源使用量可调，设计了可配置电路，可通过在配置文件中修改乘累加阵列并行度、数据位宽等参数调整加速器资源使用量。为验证可配置电路的正确性，本文选取其中三种硬件配置方式，如下表 4-5 所示，三种配置方式差异在于是乘累加阵列并行度和输入数据位宽，这是由于 CNN 加速器中核心是乘累加阵列和数据位宽，乘累加阵列并行度将影响 DSP 的使用量，数据位宽将影响卷积神经网络推理精度，以下将对不同配置方式下的卷积运算模块逻辑功能进行测试。

表 4-5 三种配置方式

配置方式	Tk	乘累加阵列并行度	数据位宽	特征数据地址深度	权重数据地址深度
1	32	32×32	8	8192	8192
2	32	32×32	16	8192	8192
3	16	16×16	16	8192	8192

如下图 4-36 为两种配置方式下卷积运算模块仿真最终打印信息，从打印信息可以看出，两种配置方式下都是“result match”，这验证了本设计中卷积运算模块可配置电路逻辑功能的正确性。



a) 配置方式二　　b) 配置方式三

图 4-36 两种配置方式下最终打印信息

4.5 本章小结

本章主要介绍了 CNN 加速器的架构设计和硬件仿真。其架构设计中分为软件层驱动设计和硬件层电路设计。在软件层设计部分，首先介绍了数据加载的方式，并对外部存储器空间进行规划，接着为便于快速搭建不同的卷积神经网络，定义了一套专门 CNN 指令集，最后介绍了硬件运算模块驱动引擎。在硬件层电路设计与仿真验证部分主要介绍了三个模块的电路设计架构，卷积+激活模块、池化模块和数据交互模块，并通过仿真验证了运算模块在不同配置方式下逻辑功能的正确性。

第 5 章 CNN 加速器板级验证

本章主要对 CNN 加速器的板级验证过程进行阐述，首先对实验环境和实验过程进行介绍，其次分析了不同配置方式下，该 CNN 加速器在资源占用、功耗、吞吐量和能效比等方面的差异，最后将本文设计的 CNN 加速器与其他同类产品进行综合对比。

5.1 实验环境

本文实验过程中所用的硬件配置为 R9 3900 CPU，64G DDR4 内存，RTX 2070 GPU，系统环境为 Ubuntu 18.04 LTS，所用软件工具包括 Pycharm、MATLAB 和 VIVADO 系列工具，如下表 5-1 所示，其中 Pycharm 所选配置为 CUDA 10.1、CUDNN 7.4.3、Python 3.8 和 PyTorch 1.4。

表 5-1 环境配置

类别	配置
系统	Ubuntu 18.04 LTS
GPU	2070
CPU	R9 3900
运行内存	64G
Python	3.7.5
PyTorch	1.4
CUDA	10.1
CUDNN	7.4.3
MATLAB	2020
VIVADO 系列工具	2021.1

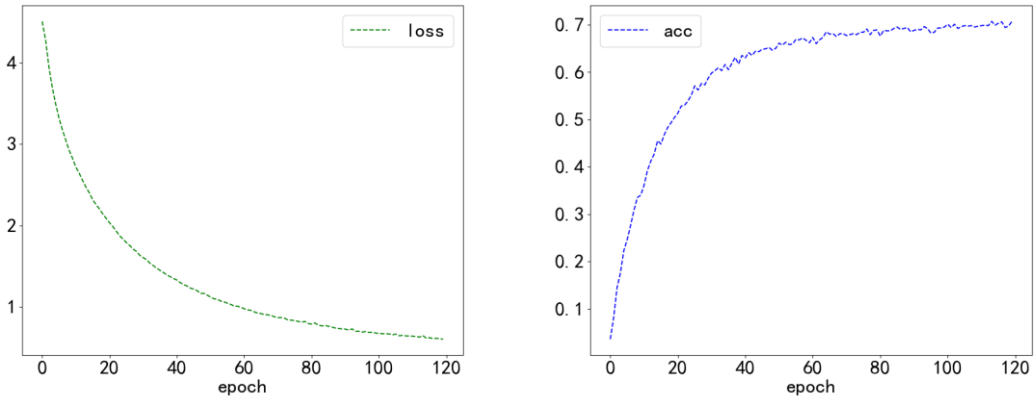
板级验证选用 Xilinx 公司的 ZCU104 评估套件，ZCU104 板卡所用芯片为 XCZU7EG，片上包含查找表 LUT 资源 230400 个，触发器 FF 资源 460800 个，乘法器 DSP 资源 1728 个，缓存空间 BRAM 资源 316 个，DDR4 资源 4GB。

5.2 实验过程

首先搭建卷积神经网络，本文选用了 VGG16 网络，其由 13 层卷积层、5 层池化层和 3 层全连接层组成，由于全连接层也可用卷积层实现，所以本文在网络训练过程中将最后三层全连接层中的矩阵运算替换为卷积运算；然后加载数据集，本文数据集选用 MiniImageNet，其由 100 类组成，如下表 5-2 所示；最后进行模型训练，如下图 5-1 所示为训练结果，其中图 a)为训练损失变化，图 b)为训练精度变化。

表 5-2 训练数据集

数据集	类别	训练集	测试集
MiniImageNet	100	48000	12000



a)训练损失变化图　　b)训练精度变化图

图 5-1 VGG16 训练结果

待训练完成后将生成的权重文件导出到 MATLAB 中进行量化和排序重组，当选择配置方式一时，其量化后各层参数如下表 5-3 所示：

表 5-3 VGG16 网络层参数

类型	输入尺寸	输出尺寸	特征量/KB	权重量/KB
CONV1	(224,224,3)	(224,224,64)	1568	18
CONV2	(224,224,64)	(224,224,64)	3136	36
POOL	(224,224,64)	(112,112,64)	3136	—
CONV3	(112,112,64)	(112,112,128)	784	72
CONV4	(112,112,128)	(112,112,128)	1568	144
POOL	(112,112,128)	(56,56,128)	1568	—

表 5-3（续表）

类型	输入尺寸	输出尺寸	特征量/KB	权重量/KB
CONV5	(56,56,128)	(56,56,256)	392	288
CONV6	(56,56,256)	(56,56,256)	784	576
CONV7	(56,56,256)	(56,56,256)	784	576
POOL	(56,56,256)	(28,28,256)	784	—
CONV8	(28,28,256)	(28,28,512)	196	1152
CONV9	(28,28,512)	(28,28,512)	392	2304
CONV10	(28,28,512)	(28,28,512)	392	2304
POOL	(28,28,512)	(14,14,512)	392	—
CONV11	(14,14,512)	(14,14,512)	98	2304
CONV12	(14,14,512)	(14,14,512)	98	2304
CONV13	(14,14,512)	(14,14,512)	98	2304
POOL	(14,14,512)	(7,7,512)	98	—
FC1	(7,7,512)	(1,1,4096)	24.5	100352
FC2	(1,1,4096)	(1,1,4096)	4	16384
FC3	(1,1,4096)	(1,1,100)	4	512
总计	—	—	16300.5	131630

待量化完成后，开始 CNN 加速器的电路设计。首先通过 RTL 级描述语言搭建 CNN 加速器各级子模块，再对其进行仿真验证、综合、布局布线、生成比特流文件，最终上板验证。如下图 5-2 为利用 VIVAD 的 Block Design 生成的电路连接图。

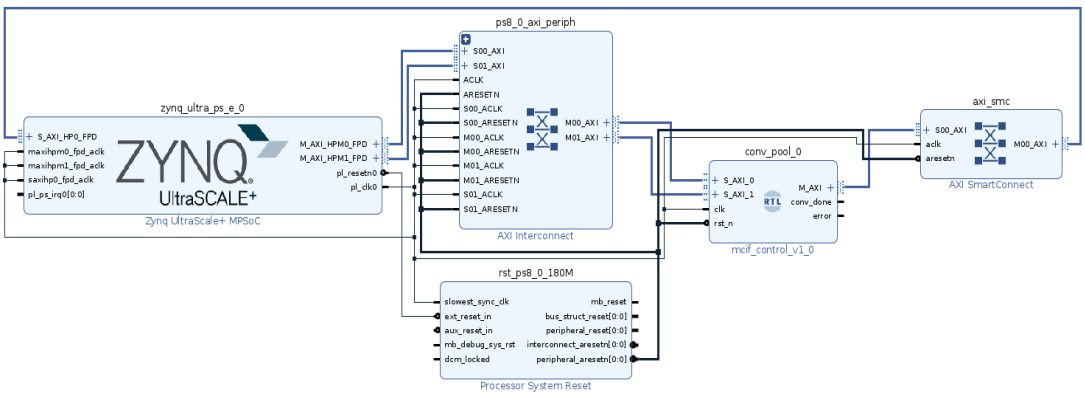


图 5-2 CNN 加速器的 Block Design 图

该加速器采用了软硬件协同运算的方式，上图中 ZYNQ UltraSCALE+为 ZCU104 的片上 ARM 处理器，conv_pool 模块为本文设计的 CNN 加速器核心运算模块，其将卷积+激活模块和池化模块封装成一个知识产权(Intelligent Property, IP)核。

待设计完成后，开始上板实验，下图 5-3 为实验中功率使用情况，图 5-4 为上板实验图，图 5-5 中 a)图为 VGG16 中各卷积层运算结果，b)图为板上推理结果。从下图可看出当选用配置方式一时，该加速器功耗为 3.752W，对于一张图片的推理时间为 138.24ms，平均 MAC 利用率为 84.37%，平均推理精度为 68.6%。

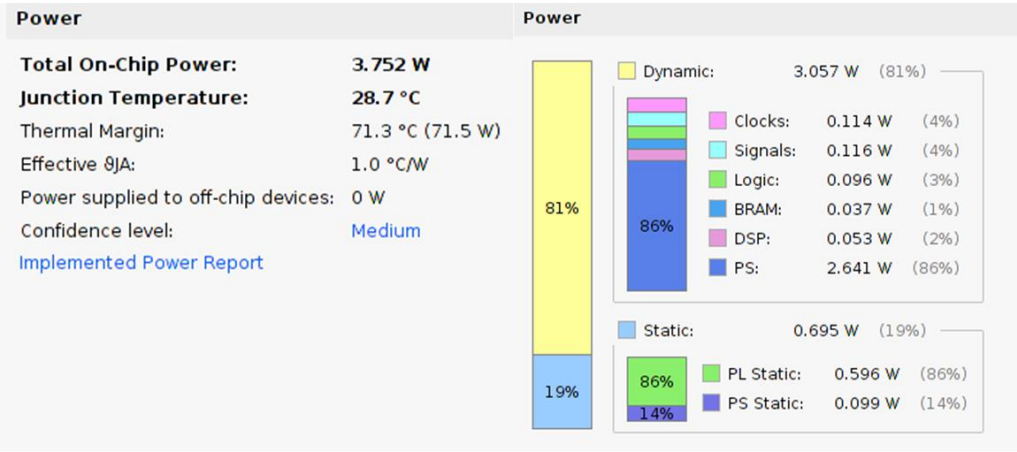


图 5-3 配置方式一时加速器功率使用情况

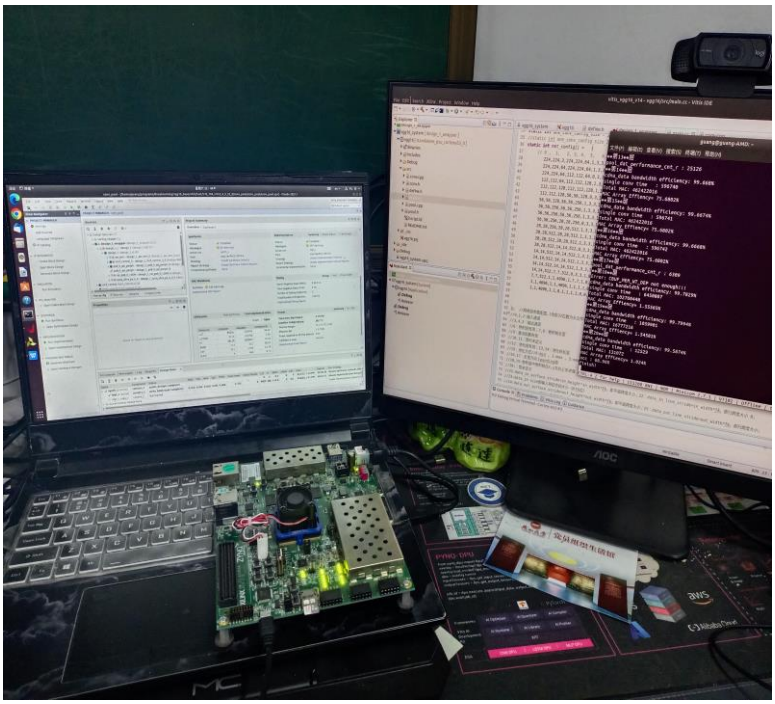
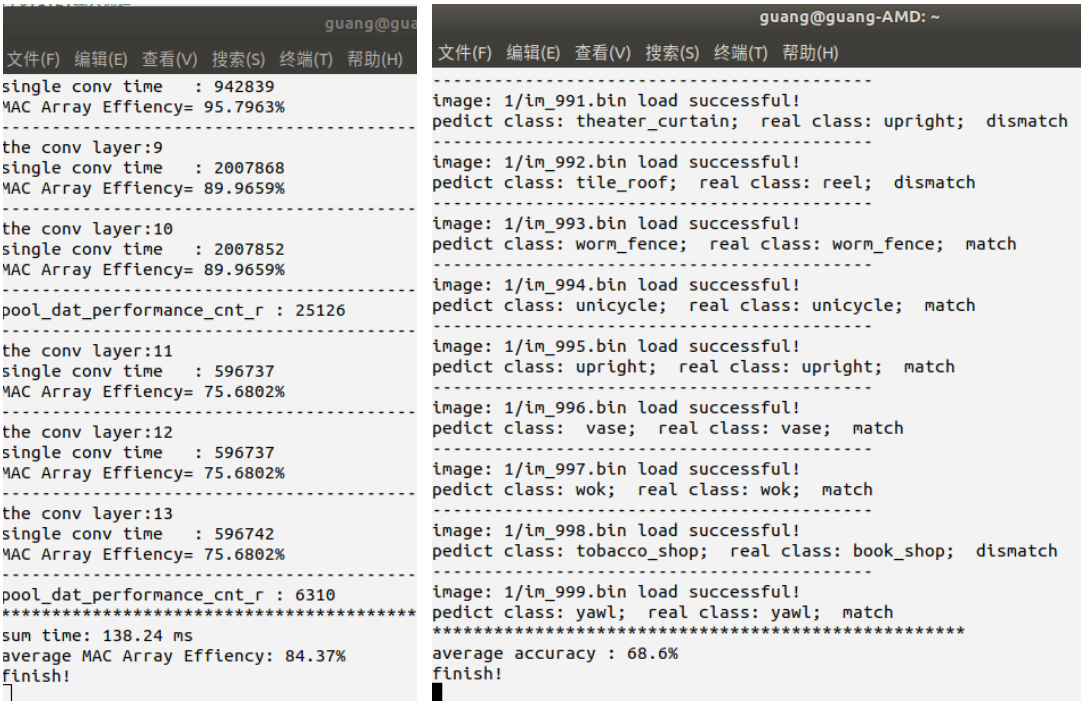


图 5-4 上板实验图



a) 各卷积运算时间及 MAC 利用率　　b) 板上推理结果

图 5-5 上板验证

5.3 板级实验与分析对比

5.3.1 资源占用情况分析

当选择配置方式一，即乘累加阵列并行度为 32×32，数据位宽为 8 时，经电路综合、布局布线后资源使用情况如下表 5-4 所示。从表中可以看出 DSP 资源使用量为 411，BRAM 资源使用量为 114，FF 资源使用量为 48130。

表 5-4 配置方式一时资源使用情况

资源使用情况	正使用	总资源	占有率/%
LUT	88291	230400	38.320747
LUTRAM	3114	101760	3.0601416
FF	48130	460800	10.444879
BRAM	114	312	36.53846
DSP	411	1728	23.784723

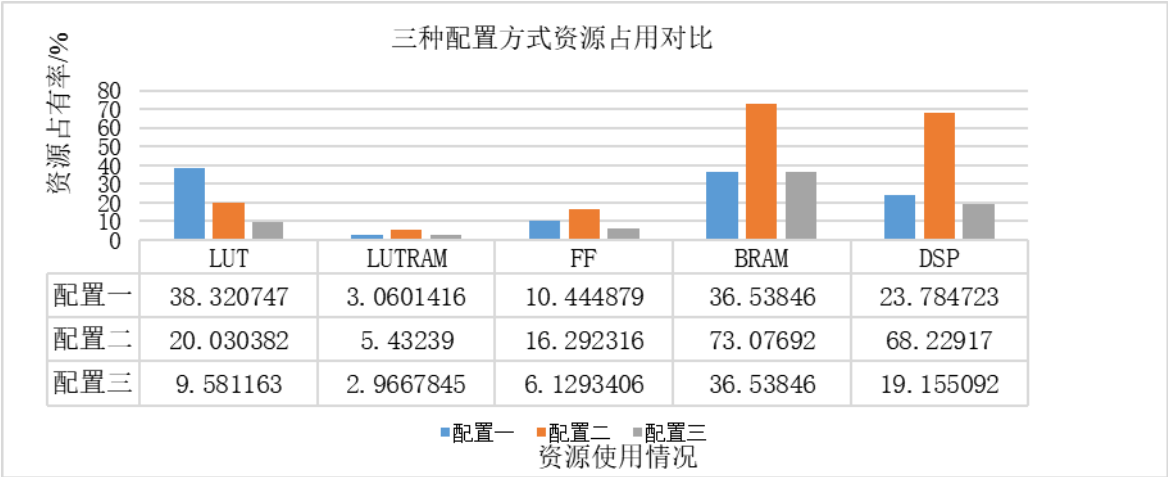
从理论角度分析，当采用 32×32 的乘累加阵列并行度时，由于每个时钟周期内

可以做 1024 次乘累加运算，所以需要 1024 个 DSP 同时工作。考虑到当数据位宽为 8 时，在电路综合过程中，对 DSP 进行优化，将一个 DSP 运算拆分成两个位宽为 8 的乘累加运算，所以理论上最少需要 512 个 DSP，这与实验结果有所差异。

为分析导致这一差异的原因，同时从资源占用角度验证可配置 CNN 加速器电路设计的正确性，本文还测试了其他两种配置。下图 5-6 为三种配置方式下资源占有率使用情况。配置方式一中乘累加阵列并行度为 32×32，数据位宽为 8；配置方式二中乘累加阵列并行度为 32×32，数据位宽为 16；配置方式三中乘累加阵列并行度为 16×16，数据位宽为 16。

从下图中可以看出配置方式二下 CNN 加速器所用 DSP 资源约为配置方式一下 DSP 资源占用的 3 倍，而配置二下 LUT 使用数量却明显少于配置方式一，这点明了上述配置一中 DSP 小于 512 的原因，即 VIVADO 工具在电路综合过程中将部分 8 位乘法运算综合成 LUT 类型，所以出现了以上结果。

而图中配置方式二下 LUT、LUTRAM、BRAM 资源占用率约为配置三下资源占用率的二倍，这是由于本文设计的 CNN 加速器采用了数据分组的思想，每组数据内包含 Tk 个数据，乘累加阵列并行度为 Tk×Tk，所以当 Tk 扩大二倍时，乘累加阵列并行度扩大四倍，即在电路设计中与之相应的 LUT、LUTRAM、BRAM 资源占用会翻倍，而 DSP 资源占用会扩大四倍，这与实验结果相符，侧面验证了可配置 CNN 加速器电路设计的正确性。



	LUT	LUTRAM	FF	BRAM	DSP
配置一	38.320747	3.0601416	10.444879	36.53846	23.784723
配置二	20.030382	5.43239	16.292316	73.07692	68.22917
配置三	9.581163	2.9667845	6.1293406	36.53846	19.155092

图 5-6 三种配置方式的资源占用率

5.3.2 推理精度对比分析

PC 端和 CNN 加速器推理结果如下表 5-5 所示，表中 CNN 加速器推理结果与 PC 端推理结果的误差主要是由于定点量化引起的，当数据位宽为 8 时其精度损失为 2%，当数据位宽为 16 时其精度损失为 1.2%，可以看出 16 位量化方式可以获得更高的推理精度。

表 5-5 VGG 16 网络模型推理结果

网络结构	数据集	PC 端推理结果	INT16 推理结果	INT8 推理结果
VGG16	MiniImageNet	70.6%	69.4%	68.6%

5.3.3 不同配置方式下性能分析

由上图 5-3 可知，乘累加阵列并行度越大，资源使用越多，但性能也得到了相应的提升，下表 5-6 列出了三种配置方式下，本文设计的 CNN 加速器的功耗、吞吐量和能效比等性能情况。

从表中可知，乘累加阵列并行度越大，吞吐量和能效比越高，但配置方式的选择需要综合考虑多方面因素，如不同硬件板卡资源有所差异，当所使用的硬件平台资源充足时，可以选择较高的乘累加阵列并行度，以获取最好的网络推理速度，然而当硬件板卡资源受限时，则可以降低乘累加阵列并行度，牺牲吞吐量以获取较低的功耗。同理数据位宽的选择也需要考虑多方面因素，当要求较高精度时时，选择 INT16，相反则选择 INT8。默认配置为 INT8，既能降低资源使用量又能获取较低的功耗和能效比。

表 5-6 三种配置方式 CNN 加速器的性能

配置方式	频率 /MHz	并行度	数据位宽	功耗 /W	吞吐量 /(GOP/S)	能效比 /(GOPS/W)
配置方式一	180	1024	Bit 8	3.752	180	47.97
配置方式二	180	1024	Bit 16	4.022	180	44.75
配置方式三	180	256	Bit 16	3.633	45	12.39

综上分析可得，本文设计的 CNN 加速器可通过修改配置参数，以调整整体资源使用量，从而便于适配不同 FPGA 硬件板卡。

5.3.4 乘累加阵列实验分析

如下表 5-7 为当选用配置方式一时，VGG16 各层网络的工作效率，主要包括运算时间和 MAC 利用率，运算时间指在加速器中单层网络运算所用时间，MAC 利用率是指从卷积运算开始到结束这一段时间内，该加速器实际吞吐量占峰值吞吐量的比例。

表 5-7 配置方式一时加速器工作效率

类型	运算量(乘加)	运算时间/ms	MAC 利用率
CONV	86704128	5.068	9.28218%
CONV	1849688064	10.241	97.9947%
POOL	—	1.115	—
CONV	924844032	5.061	99.1403%
CONV	1849688064	10.172	98.6638%
POOL	—	0.558	—
CONV	924844032	5.142	97.587%
CONV	1849688064	10.490	95.6644%
CONV	1849688064	10.490	95.6644%
POOL	—	0.279	—
CONV	924844032	5.238	95.7963%
CONV	1849688064	11.155	89.9659%
CONV	1849688064	11.155	89.9659%
POOL	—	0.140	—
CONV	462422016	3.315	75.6802%
CONV	462422016	3.315	75.6802%
CONV	462422016	3.315	75.6802%
POOL	—	0.035	—
FC	102760448	35.839	1.5556%
FC	16777216	5.888	1.54595%
FC	131072	0.229	1.2427%
总计	15466299392	138.24	—

从表中最后一栏可以看出对于 VGG16 网络，推理一张特征图耗时 138.24ms，对于前 13 层卷积运算层，其平均 MAC 利用率为 84.37%，而最后三层 MAC 利用率较低，这是由于全连接层参数量庞大，从数据加载到数据运算需要等待较长上的时间，无法满足乘累加阵列的需求，从而致使 MAC 利用率降低。

总体分析可得，VGG16 网络在硬件推理过程中，其前 13 层卷积层平均 MAC 利用率为 84.37%，这验证了本文设计的 CNN 加速器架构能够较大化的利用硬件资源。

5.3.5 加速器性能对比分析

CNN 加速器硬件设计的性能指标主要为吞吐量、功耗和能效比。当选用 ZCU104 硬件平台时，本文设计的 CNN 加速器的核心时钟频率为 180 MHz，吞吐量为 180 GOPS，功耗为 3.752 W，能效比为 47.97 GOPS/W。

将本文设计的 CNN 加速器与其他基于 FPGA 的神经网络加速器进行综合对比，如下表 5-8 所示。与文献[55]相比，在吞吐量和资源占用相近的情况下，本文通过划分时钟域的方法使得功耗降低了 2.15 倍，能效比提升了 2.07 倍；与文献[56]相比，虽然吞吐量处于劣势，但由于功耗降低了 4.1 倍，所以能效比提升了 1.49 倍；与文献[57]相比，其吞吐量提升了 2.13 倍，能效比提升了 1.99 倍。

综上分析可得，本文设计的 CNN 加速器具有高能效比、低功耗的优势，相比同产品具有较为明显的优势。

表 5-8 加速器性能对比

	文献[55]	文献[56]	文献[57]	本文
网络	VGG-16	VGG-16	VGG-16	VGG-16
FPGA	VC707	ZCU102	XC7Z020	ZCU104
频率(MHz)	100	200	214	180
量化策略	16-fixed	16-fixed	8-fixed	8-fiixed
DSP	471	1352	190	441
吞吐量(GOPS)	188.4	495.4	84.3	180.0
功耗(W)	8.15	15.4	3.5	3.752
能效比(GOPS/W)	23.1	32.16	24.1	47.97

5.4 本章小结

本章主要介绍了 CNN 加速器的板级验证，首先分析不同配置方式对资源占用的影响；其次分析了不同配置方式对加速器的功耗、吞吐量和能效比的影响，针对不同场景可选用不同配置方式；然后将文本设计的加速器与其他同类产品做比较，最终实验可得，当选择配置方式一时，卷积运算阵列峰值吞吐量为 180 GOPS，功耗为 3.752 W，能效比达到 47.97 GOPS/W，对于 VGG16 网络，其卷积层的平均 MAC 利用率达到 84.37%，相比于同类产品具有较明显的优势，这表明该加速器具有较高的可配置性和高效性能。

结　论

针对现阶段神经网络加速器主要应用在单一 FPGA 硬件，跨硬件平台使用困难的问题，本文提出了一种基于 FPGA 的较为通用的可配置卷积神经网络加速器，既能够满足多种卷积神经网络模型、实现边缘加速的效果，又能够根据不同硬件平台资源差异，调整卷积运算中乘累加阵列并行度，实现对当前硬件平台资源的较好利用。

为设计通用可配置卷积神经网络加速器，便于 CNN 加速器跨平台应用，本文采用软硬件协同运算的方式，分为软件驱动层和硬件设计层，在软件驱动层上，该加速器定义了专用的卷积神经网络指令集，通过在软件中搭建网络、编译网络生成控制指令，调度硬件完成各类卷积神经网络模型的推理工作，在硬件设计层上负责卷积运算和池化运算。本文的主要工作和贡献如下：

(1)为获得较高的通用性，本文提出了软硬件协同配置的方案。将量化后数据位宽、中间缓存空间大小、乘累加阵列并行度作为一种可选配置参数。对于不同应用场景和应用平台，可以从推理精度要求、FPGA 硬件资源等多方面考虑，选取恰当配置参数，以满足多种 FPGA 硬件平台的需要。

(2)为获得较高的能效比，本文采用高频时钟+低频时钟结合的方式，其中控制部分和参数配置部分采用低频时钟，核心运算和数据传输部分采用高频时钟，通过多时钟域的方式，进一步优化硬件时序，使得核心运算模块工作在较高频率，降低功耗的同时提高 CNN 加速器算力。

(3)针对卷积神经网络中数据参数量庞大，无法全部加载到片上缓存空间的问题，本文优化了数据分块方式。并在数据分块后对比权重数据复用和特征数据复用的总传输参数量，提出了一种自适应数据复用的策略，减少了数据传输的总参数量。

(4)针对卷积神经网络的快速搭建需求，对卷积神经网络所需的参数进行封装，并定义了专用的卷积神经网络指令集，通过使用其指令集，用户可以快速地搭建出多种卷积神经网络，该加速器具备广泛的应用能力。

本文在 Xilinx ZCU104 板卡上进行了实验验证，实验结果可得，当量化位宽选择 8 位整型数据、乘累加阵列并行度选择 32×32、核心时钟频率工作在 180 MHz 时，该

加速器峰值吞吐量为 180 GOPS，功耗为 3.752 W，能效比达到 47.97 GOPS/W，对于 VGG16 网络，其卷积层平均 MAC 利用率为 84.37%，相对于同类产品具有明显的优势，但是在功能运算模块和量化方面还有提升空间，后续的研究方向如下：

(1)本文设计的 CNN 加速器只有卷积运算模块、池化运算模块，虽然能够应用在大多数分类网络中，但由于缺乏残差块运算模块，所以在无法应用在目标检测领域应用。为此，后续会设计多种运算模块，使其支持多种函数功能，从而适应最新的卷积神经网络。

(2)本文为提高 CNN 加速器性能优化了电路设计架构，采用了训练后量化的方式，量化精度方面还有提升空间，未来工作可以通过采用感知量化训练的方法提升量化后模型的精度。

参考文献

[1] Lecun Y, Boser B, Denker J, et al. Handwritten digit recognition with a back-propagation network [C]//Proceedings of the 2nd International Conference on Neural Information Processing Systems, 1989: 396-404..

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

[3] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition[J]. arXiv preprint arXiv:1409.1556, 2014.

[4] Lin M, Chen Q, Yan S. Network in network[J]. arXiv preprint arXiv:1312.4400, 2013.

[5] Iandola F N, Han S, Moskewicz M W, et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and< 0.5 MB model size[J]. arXiv preprint arXiv:1602.07360, 2016.

[6] Howard A G, Zhu M, Chen B, et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications[J]. arXiv preprint arXiv:1704.04861, 2017.

[7] Zhang X, Zhou X, Lin M, et al. Shufflenet: An extremely efficient convolutional neural network for mobile devices[C]//Proceedings of the IEEE conference on computer vision and pattern recognition, 2018: 6848-6856.

[8] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]// Proceedings of the IEEE conference on computer vision and pattern recognition, 2016: 770-778.

[9] 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 & Machine Intelligence, 2017, 39(06): 1137-1149..

[10] Long J, Shelhamer E, Darrell T. Fully convolutional networks for semantic segmentation[C] //Proceedings of the IEEE conference on computer vision and pattern recognition, 2015: 3431-3440.

[11] Donahue J, Anne Hendricks L, Guadarrama S, et al. Long-term recurrent convolutional networks for visual recognition and description[C]//Proceedings of the IEEE conference on computer vision and pattern recognition, 2015: 2625-2634.

[12] Temam O. The rebirth of neural networks[C]//Proceedings of the 37th annual international symposium on Computer architecture, 2010: 349-349

[13] 葛悦涛, 任彦. 2020 年人工智能芯片技术发展综述[J]. 无人系统技术, 2021, 4(02): 14-19.

[14] 施羽暇. 人工智能芯片技术体系研究综述[J]. 电信科学, 2019, 35(04): 114-119.

[15] Li J, Un K-F, Yu W-H, et al. An FPGA-based energy-efficient reconfigurable convolutional neural network accelerator for object recognition applications[J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2021, 68(9): 3143-3147.

[16] Sankaradas M, Jakkula V, Cadambi S, et al. A massively parallel coprocessor for convolutional neural networks[C]//2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors, 2009: 53-60.

[17] 陈煌, 祝永新, 田犁, 等. 基于 FPGA 的卷积神经网络卷积层并行加速结构设计[J]. 微电子学与计算机, 2018, 35(10): 85-88.

[18] Chen Y, Chen T, Xu Z, et al. DianNao family: energy-efficient hardware accelerators for machine learning[J]. Communications of the ACM, 2016, 59(11): 105-112.

[19] Chen T, Du Z, Sun N, et al. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning[J]. ACM SIGARCH Computer Architecture News, 2014, 42(1): 269-284.

[20] Chen T. Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning[J]. ACM Sigplan Notices, 2014, 49(4): 269-284.

[21] Chen Y-H, Krishna T, Emer J S, et al. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks[J]. IEEE journal of solid-state circuits, 2016, 52(1): 127-138.

[22] Moons B, Uytterhoeven R, Dehaene W, et al. 14.5 envision: A 0.26-to-10tops/w subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm fdsoi[C]. 2017 IEEE International Solid-State Circuits Conference (ISSCC), 2017: 246-247.

[23] Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit[C]//Proceedings of the 44th annual international symposium on computer architecture, 2017: 1-12.

[24] Granter S R, Beck A H, Papke Jr D J. AlphaGo, deep learning, and the future of the human microscopist[J]. Archives of pathology & laboratory medicine, 2017, 141(5): 619-621.

[25] Firestone D, Putnam A, Mundkur S, et al. Azure accelerated networking: Smartnics in the public cloud[C]//Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation, 2018: 51-64.

[26] Ionica M H, Gregg D. The movidius myriad architecture's potential for scientific computing[J]. IEEE Micro, 2015, 35(1): 6-14.

[27] Markidis S, Der Chien S W, Laure E, et al. Nvidia tensor core programmability, performance & precision[C]//2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW), 2018: 522-531.

[28] Kathail V. Xilinx vitis unified software platform[C]//Proceedings of the 2020 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2020: 173-174.

[29] Li X, Huang H, Chen T, et al. A hardware-efficient computing engine for FPGA-based deep convolutional neural network accelerator[J]. Microelectronics Journal, 2022, 128: 105547.

[30] Tian T, Jin X, Zhao L, et al. Exploration of memory access optimization for FPGA-based 3D CNN accelerator[C]//2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2020: 1650-1655.

[31] Jaderberg M, Vedaldi A, Zisserman A. Speeding up convolutional neural networks with low rank expansions[J]. arXiv preprint arXiv:1405.3866, 2014.

[32] Lebedev V, Lempitsky V. Speeding-up convolutional neural networks: A survey[J]. Bulletin of the Polish Academy of Sciences. Technical Sciences, 2018, 66(6): 799-811.

[33] Albericio J, Judd P, Hetherington T, et al. Cnvlutin: Ineffectual-neuron-free deep neural network computing[J]. ACM SIGARCH Computer Architecture News, 2016, 44(3): 1-13.

[34] Wen L, Zhang X, Bai H, et al. Structured pruning of recurrent neural networks through neuron selection[J]. Neural Networks, 2020, 123: 134-141.

[35] Anwar S, Hwang K, Sung W. Structured pruning of deep convolutional neural networks[J]. ACM Journal on Emerging Technologies in Computing Systems (JETC), 2017, 13(3): 1-18.

[36] Tan M, Le Q. Efficientnet: Rethinking model scaling for convolutional neural networks[C]//International conference on machine learning, 2019: 6105-6114.

[37] Chollet F. Xception: Deep learning with depthwise separable convolutions[C]// Proceedings of the IEEE conference on computer vision and pattern recognition, 2017: 1251-1258.

[38] Sheng T, Feng C, Zhuo S, et al. A quantization-friendly separable convolution for mobilenets[C]//2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2), 2018: 14-18.

[39] Shin D, Lee J, Lee J, et al. 14.2 DNPU: An 8.1 TOPS/W reconfigurable CNN-RNN processor for general-purpose deep neural networks[C]//2017 IEEE International Solid-State Circuits Conference (ISSCC), 2017: 240-241.

[40] Desoli G, Chawla N, Boesch T, et al. 14.1 A 2.9 TOPS/W deep convolutional neural network SoC in FD-SOI 28nm for intelligent embedded systems[C]//2017 IEEE International Solid-State Circuits Conference (ISSCC), 2017: 238-239.

[41] Park S, Choi S, Lee J, et al. 14.1 A 126.1 mW real-time natural UI/UX processor with embedded deep-learning core for low-power smart glasses[C]//2016 IEEE International Solid-State Circuits Conference (ISSCC), 2016: 254-255.

[42] Sim J, Park J-S, Kim M, et al. 14.6 a 1.42 tops/w deep convolutional neural network recognition processor for intelligent ioe systems[C]//2016 IEEE International Solid-State Circuits Conference (ISSCC), 2016: 264-265.

[43] Chen Y, Luo T, Liu S, et al. Dadiannao: A machine-learning supercomputer[C]//2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, 2014: 609-622.

[44] Du Z, Fasthuber R, Chen T, et al. ShiDianNao: Shifting vision processing closer to the sensor[C]//Proceedings of the 42nd Annual International Symposium on Computer Architecture, 2015: 92-104.

[45] Liu D, Chen T, Liu S, et al. Pudiannao: A polyvalent machine learning accelerator[J]. ACM SIGARCH Computer Architecture News, 2015, 43(1): 369-381.

[46] 余子健, 马德, 严晓浪, 等. 基于 FPGA 的卷积神经网络加速器[J]. 计算机工程, 2017, 43(01): 109-114+119.

[47] 李沙沙, 李夏禹, 刘珊珊,等. 一种基于 FPGA 的通用卷积神经网络加速器的设计与实现[J]. 复旦学报(自然科学版), 2022, 61(01): 69-76+84.

[48] Courbariaux M, Bengio Y, David J-P. Binaryconnect: Training deep neural networks with binary weights during propagations[C]//Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2, 2015: 3123-3131.

[49] Rastegari M, Ordonez V, Redmon J, et al. Xnor-net: Imagenet classification using binary convolutional neural networks[C]//Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV, 2016: 525-542.

[50] Li F, Zhang B, Liu B. Ternary weight networks[J]. arXiv preprint arXiv:1605.04711, 2016.

[51] Hubara I, Courbariaux M, Soudry D, et al. Quantized neural networks: Training neural networks with low precision weights and activations[J]. The Journal of Machine Learning Research, 2017, 18(1): 6869-6898.

[52] Wu J, Leng C, Wang Y, et al. Quantized convolutional neural networks for mobile devices[C]//Proceedings of the IEEE conference on computer vision and pattern recognition, 2016: 4820-4828.

[53] 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.

[54] Sun M, Li Z, Lu A, et al. FILM-QNN: Efficient FPGA acceleration of deep neural networks with intra-layer, mixed-precision quantization[C]//Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, 2022: 134-145.

[55] 吴成路. 基于分组剪枝的 CNN 加速器设计与 FPGA 验证[D].南京 东南大学, 2020.

[56] Lu L, Xie J, Huang R, et al. An efficient hardware accelerator for sparse convolutional neural networks on FPGAs[C]//2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2019: 17-25.

[57] 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.

攻读硕士学位期间承担的科研任务与主要成果

(一) 参与的科研课题

[1] 河北省中央引导地方专项(199477141G)：视觉防跌倒看护系统在骨科医院、康养医院中的示范应用

(二) 发表的学术论文

[1] 张立国,杨红光,金梅,申前.一种基于 SOC 的可配置 CNN 加速器的设计与实现[J].高技术通讯

(三) 申请及已获得的专利

[1] 张立国,杨红光,金梅,申前,耿星硕,王磊,李佳庆,黄文汉,张升. 一种基于云端的监控视频下多目标跟踪的方法[P]. 河北省：CN113674321A,2021-11-19

致　谢

白驹过隙，不经意间就到了分别的时刻，三年的时间很短，但也足够我在此汲取丰富的知识，创造不一样的精彩，认识诸多师友。

在此首先感谢张立国老师对我的培养，高山仰止，在张老师团队学习和工作期间是我进步最快的一段时光，每当面对困难时张老师总是能从特有的角度指出关键，帮助我鼓励我，充分的锻炼了我独立思考勇于面对困难的能力，同时张老师的工作态度和处事风格也深深的感染我，使我在日常的熏陶中受益匪浅。其次特别感谢金梅老师不辞辛劳的帮我修改论文，春风化雨，在科研上提出了许多宝贵的建议，让我能在科研写作时不断完善，努力提升。

感谢师兄刘强、李福昆、李义辉、马子荐，李翔宇，在相关科研方向给予了我极大的帮助，让我能在科研阶段快速入门、快速实践；感谢室友许崇轩、许波、辛旺、徐栋、徐香院在日常生活中的帮助；感谢同学申前、孟子杰、黄文汉、李佳庆、耿星硕、王磊、张升、张玉鹏、秦芊、薛静芳在日常学习中带给我的帮助。

最后感谢三年来所有在生活、工作以及所有在我最需要的时候帮助过我的家人、老师和同学。