**密级： 保密期限：**

xm 拷贝

**硕士学位论文**



**题目： 基于TTA的大型卷积神经网络**

**处理器架构设计**

**学 号：**

**姓 名：**

**专 业： 信息与通信工程**

**导 师：**

**学 院： 信息与通信工程**

**2019年12月7日**



**A Thesis for Master Degree**

**TITLE: AN PROCESSOR ARCHITECTURE DESIGN FOR LARGE-SCALE CNNS BASED ON TTA**

|  |  |
| --- | --- |
| **Student No.:** |  |
| **Author:** |  |
| **Major:** | **Information and Communi-**  **cation Engineering** |
| **Supervisor:** |  |
| **School:** | **Information and Communi-**  **cation Engineering** |

**Dec. 7th, 2018**

独创性（或创新性）声明

本人声明所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名： 日期：

关于论文使用授权的说明

本人完全了解并同意北京邮电大学有关保留、使用学位论文的规定，即：北京邮电大学拥有以下关于学位论文的无偿使用权，具体包括：学校有权保留并向国家有关部门或机构送交学位论文，有权允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，有权允许采用影印、缩印或其它复制手段保存、汇编学位论文，将学位论文的全部或部分内容编入有关数据库进行检索。（保密的学位论文在解密后遵守此规定）

本人签名： 日期：

导师签名： 日期：

基于TTA的大型卷积神经网络处理器架构设计

摘 要

卷积神经网络作为实现人工智能任务的有效算法之一，已经在各种应用场景获得广泛的应用。目前卷积神经网络的加速平台主要包括通用处理器以及专用集成电路（ASIC, Application Specific Integrated Circuit）。前者灵活性较好，但计算性能与功耗较差；后者性能较好，但灵活性很差，开发成本高昂。专用指令集处理器（ASIP, Application Specific Instruction Set Processor）针对某一领域进行优化，牺牲了一定的性能以换取灵活性的增加，从而达到平衡。传输触发架构（TTA, Transport Triggered Architecture）是ASIP中的一种架构，其将寄存器单元也作为一种特殊的基本单元，有效地减少了寄存器堆的设计压力，成为一种非常适合于专用处理器领域的架构。

本文首先针对TTA处理器设计了计算框架，使处理器能够对任意深度的卷积神经网络进行加速，同时对较为紧缺的片内存储资源提出了输出缓存优化结构与输入缓存优化结构两种优化方案，减少了近一半的存储资源使用。其次，针对传统卷积加速单元在卷积操作步长大于1时计算效率低下且消耗较多片内存储单元的问题，设计了旋转存储结构。在不增加计算与存储资源的条件下，可以解决传统卷积加速单元所存在的问题，以提高计算效率与减少片内存储使用。再者，因为卷积神经网络运算逻辑较为复杂，本文从大颗粒度上基于TCE工具链对TTA处理器的功能单元及互联网络进行了设计，以减少处理器复杂度与提升计算效率。最后，基于处理器的灵活性出发，对各运算层进行了封装，以能够实现任意参数下的加速，在软件代码中通过简单的函数调用即可对指定运算层进行加速。

本文对卷积神经网络的TTA处理器进行了设计，相比于ASIC设计方式，大大减少了硬件加速平台的开发时间与成本。同时相比通用处理器CPU，TTA处理器的计算效率更高，所需片外访存次数更少，可以达到更好的加速效果。

**关键词：**卷积神经网络 传输触发架构 专用指令集处理器 旋转存储结构

AN PROCESSOR ARCHITECTURE DESIGN FOR

LARGE-SCALE CNNS BASED ON TTA

ABSTRACT

As one of the most effective algorithms for realizing artificial intelligence tasks, CNNs (Convolutional Neural Networks) have been widely used in various scenes. So far, the platforms to accelerate the CNNs mainly include General Purpose Processor and ASIC (Application Specific Integrated Circuit). The former performs better in flexibility, but the computational efficiency and power consumption cost. While the latter performs better in computational efficiency，but it has higher development costs. The ASIP (Application Specific Instruction Set Processor) optimizes a specific application domain, compared to ASIC, it sacrifices certain performance to increase flexibility, so that a balance can been reached between performance and flexibility. TTA (Transport Triggered Architecture) is a kind of architecture in ASIP. It treats register units as a special function unit, which can effectively reduce the design complexity of the register files and make it become a superb processor for specific domain.

This paper first designs a computational framework for the TTA processor, which enables the processor to accelerate the CNNs with any depth. At the same time, input-buffer optimization structure and output-buffer optimization structure are proposed for optimizing the insufficient on-chip storage resources, reducing nearly half of the storage resources. Secondly, the computational efficiency of the traditional convolution function unit is very low when the stride parameter is greater than 1, also, it costs too much on-chip memory resources. A rotary-storage structure is designed for this to solve this problem without increasing the computing and on-chip memory resources, so that computational efficiency can be increased and usage of on-chip memory can be reduced. Furthermore, due to the complexity of computing procedure of CNNs, this paper designs the functional units and interconnection networks of TTA processors based on TCE toolchain from large scale to reduce processor complexity and increase computational efficiency. Finally, start from the flexibility of the processor, each operation layer is encapsulated, which can achieve acceleration under arbitrary parameters. The acceleration of specific layer can be achieved in the software with a simple function call.

This paper designs the TTA processor of the CNNs, which greatly reduces the development time and the cost of the hardware acceleration platform. At the same time, compared to CPU, TTA processor can achieve high computationally efficient, furthermore, less load-store times are needed, which means better performance.

KEY WORDS: Convolutional Neural Network TTA ASIP Rotary-Storage Structure

**目录**

[第一章 绪论 1](#_Toc535091501)

[1.1 课题背景 1](#_Toc535091502)

[1.2 国内外现状与研究目的 2](#_Toc535091503)

[1.3 论文主要工作 4](#_Toc535091504)

[1.4 论文组织安排 5](#_Toc535091505)

[第二章 相关技术研究 6](#_Toc535091506)

[2.1 卷积神经网络结构详解 6](#_Toc535091507)

[2.2 卷积操作并行性研究 9](#_Toc535091508)

[2.2.1 卷积窗口内并行 10](#_Toc535091509)

[2.2.2 输入特征图并行 11](#_Toc535091510)

[2.2.3 输出特征图并行 12](#_Toc535091511)

[2.2.4 并行性组合 13](#_Toc535091512)

[2.3 TTA架构及TCE开发工具集 14](#_Toc535091513)

[2.3.1 TTA架构介绍 14](#_Toc535091514)

[2.3.2 TCE工具集介绍 15](#_Toc535091515)

[2.3.3 TCE设计流程 19](#_Toc535091516)

[2.4 本章小结 21](#_Toc535091517)

[第三章 大规模卷积神经网络计算框架设计 22](#_Toc535091518)

[3.1 计算框架设计 22](#_Toc535091519)

[3.2 片内存储优化方案 25](#_Toc535091520)

[3.2.1 输出缓存优化结构 25](#_Toc535091521)

[3.2.2 输入缓存优化结构 26](#_Toc535091522)

[3.3 本章小结 27](#_Toc535091523)

[第四章 基于TTA架构的功能单元及互联网络设计 28](#_Toc535091524)

[4.1 片外存储功能单元设计 28](#_Toc535091525)

[4.2 输入缓存功能单元设计 32](#_Toc535091526)

[4.2.1 旋转存储结构 33](#_Toc535091527)

[4.2.2 顺序存储结构 36](#_Toc535091528)

[4.3 通用计算功能单元设计 37](#_Toc535091529)

[4.3.1 卷积运算操作设计 38](#_Toc535091530)

[4.3.2 池化运算操作设计 39](#_Toc535091531)

[4.3.3 激活函数操作设计 40](#_Toc535091532)

[4.3.4 批量归一化操作设计 41](#_Toc535091533)

[4.3.5 全连接层操作设计 42](#_Toc535091534)

[4.4 输出缓存功能单元设计 42](#_Toc535091535)

[4.5 互联网络设计 45](#_Toc535091536)

[4.6 本章小结 47](#_Toc535091537)

[第五章 基于TTA架构的卷积神经网络实现及性能分析 48](#_Toc535091538)

[5.1 自定义运算层实现 48](#_Toc535091539)

[5.1.1 卷积层实现 48](#_Toc535091540)

[5.1.2 池化层实现 51](#_Toc535091541)

[5.1.3 激活函数层实现 52](#_Toc535091542)

[5.1.4 批量归一化层实现 53](#_Toc535091543)

[5.1.5 全连接层实现 54](#_Toc535091544)

[5.2 灵活性与性能分析 55](#_Toc535091545)

[5.3 本章小结 59](#_Toc535091546)

[第六章 总结与展望 60](#_Toc535091547)

[6.1 工作总结 60](#_Toc535091548)

[6.2 不足与展望 61](#_Toc535091549)

[参考文献 62](#_Toc535091550)

[致谢 65](#_Toc535091551)

[攻读学位期间取得的学术论文目录 66](#_Toc535091552)

第一章 绪论

1.1 课题背景

人工智能（AI, Artificial Intelligence）[1]，是通过计算机来达到扩展人的智能的一门科学，与空间技术和能源科技并称为世界的三大顶尖科技。近年来，人工智能领域取得了极大的进展与成就，使得人工智能成为最引人注目的热点领域。其也影响着诸如制造业、出行、教育行业、金融行业、新闻行业等一系列我国的产业应用，人工智能已成为推动我国发展的新动力[2]。深度学习作为人工智能领域中的一大分支[3]，近几年在许多事件中都体现出了其强大的解决问题能力，具有很高的学术研究价值。

卷积神经网络（Convolutional Neutral Network，CNN），是深度学习领域中的一类神经网络模型。由于其具有权值共享以及局部连接的特性[4]，相比较传统深度神经网络，网络模型的复杂度以及参数数量都大大降低。该优点在网络的输入是多维图像时表现得更明显，使图像可以直接作为网络的输入, 避免了传统识别算法中复杂的特征提取和数据重建过程[5]。近年来，卷积神经网络发展迅速，在图像处理以及自然语言处理领域都有着广泛的应用[6, 7]。

针对于神经网络的使用场景，其一般都具有高实时性、低功耗的需求。传统通用处理器CPU以及ARM，虽然擅长于逻辑控制，但计算资源以及效率不足，无法满足实时性的特点。而GPU虽然具有大量的计算单元，但一般的GPU功耗都在百瓦以上，无法满足低功耗的需求。此外，GPU采用SIMD架构，在训练时可以将多幅图片同时进行计算来提升计算资源的使用效率，但在应用场景中，只需对一副图片进行计算，无法使用所有计算资源，使得GPU的计算资源的使用效率变得很低。

除了通用处理器，在应用场景中，最主流的设计方式为ASIC（Application Specific Integrated Circuit）。ASIC是一种为了专门目的而进行设计的集成电路，在设计者的能力较强的情况下，ASIC一般可以具有小体积、低功耗、高可靠性、低成本等优点，可以说是特定场景中最适用的硬件平台[8]。但由于ASIC目的性过强，因此需要设计者完成所有电路设计，所需人力成本较高，开发周期较长。同时ASIC设计方式在算法层面上灵活性较差，在对算法有少量修改时，仍需重新设计。因此ASIC适用于算法确定、开发周期较长且有大规模需求的场景。

ASIP（Application Specific Instruction Set Processor）设计是介于通用处理器以及ASIC设计的一种设计方式[9]。ASIP面向某一领域，对该领域进行分析，从而达到功耗、速度、成本、灵活性等多方面的权衡。ASIP面向某一特定领域进行特殊功能单元的设计，在通用处理器上需要多条指令完成的任务，在ASIP上可能只需要一条指令调度特殊功能单元来完成。因此相比于通用处理器，ASIP在功耗、速度上可以达到更优。而相比ASIC设计，ASIP的逻辑控制通过指令的调度完成，算法的变化只需对指令进行更改而不涉及到电路，在灵活性上具有更大的优势。ASIP更适用于需要对算法进行更新与优化的场景。

TTA（Transport Triggered Architecture）体系结构是ASIP中的一种[10]，是超长指令集VLIW（Very Long Instruction Word）体系结构的扩展。VLIW架构是一种利用指令级并行优势的架构，将数据不想关的若干条子指令打包成一条长指令执行，以提高运行速度。但由于其不规整的指令格式，使得旁路网络较为复杂，处理器复杂性较高。TTA基于数据传输触发，其指令不需要描述执行的操作，而是对操作所需的数据进行传输，操作是数据传输的一个边沿作用，降低了指令的复杂性。同时TTA架构将寄存器文件看做一种特殊的功能单元，使得处理器的互联网络复杂度大大降低，使得TTA架构成为一种非常适合于专用处理器领域的处理器架构。

FPGA（Field－Programmable Gate Array），即现场可编程门阵列，是一种可以通过软件来编程内部连接结构，完成既定设计功能的数字集成电路[11]。FPGA被称为“万能芯片”，它并不是一种设计方式，而是一种设计平台。在ASIC与ASIP的设计过程中，往往需要设计硬件电路，但这些电路被固化后无法进行修改，一旦失败则需要重新设计，导致开发周期的延长。而FPGA作为一种可编程的硬件电路，具有可编程灵活性高的特点，在FPGA平台上进行设计，可以使得设计人员在软件层面来进行硬件结构的更改，而无需替换和设计芯片，加快开发时间。

本文所研究的内容为卷积神经网络的硬件平台设计，由于神经网络算法的多样性，且深度学习领域发展迅速，算法的迭代较为快速[12, 13, 14, 15]。因此相比于ASIC设计方式，ASIP设计方式在应用场景中更为适合。同时为了减少应用中硬件的设计成本，本文选择FPGA平台来进行ASIP的设计实现，达到对卷积神经网络计算加速的效果。

1.2 国内外现状与研究目的

神经网络的实现通常以软件方式实现，谷歌、微软等公司以及一些学者也专门开发了针对于神经网络的开源软件框架，如Tensorflow、Caffe、Pytorch等。GPU近年来发展迅速，由于GPU的高计算吞吐量，同时采用SIMD架构，可以同时多个输入数据同时进行计算，使它成为加速神经网络最好的硬件平台之一。现有的软件框架也陆陆续续开始支持在GPU上进行加速，达到CPU+GPU的异构框架模式。2012年，多伦多大学的Hinton与Alex使用GPU对卷积神经网络Alexnet进行加速，拿到了当年的ImageNet图像识别比赛的冠军[16]，拉开了深度学习热潮的帷幕，也使得GPU加速神经网络也成为共识。然而在工业界的应用中，由于GPU存在高价格、高功耗、在推断阶段低效率等一些问题，在实际使用中并不是很理想的硬件平台。

神经网络的硬件实现由来已久，早在19世纪60年代，感知机和ADALINE神经网络模型被提出，当时有几家研究机构试图使用硬件来实现神经元。在实际实现中，感知机的突触强度是通过电动马达的转动来进行改变，这种方法有着体积大、不可靠等缺点。ADALINE模型的实现则使用电阻来改变突触强度。1974年，LouisGilstrap和Roger Barton开发出第一块神经元芯片。这种芯片只包含一个神经元，具有多个输入且互联强度可以自适应改变，将多个芯片插板构建在一起，就可以构建出具有多个神经元的网络。而随着硬件以及算法的发展，如今神经网络的神经元数量往往在十万甚至百万级别，发展新的硬件平台成为需要。

现有的神经网络硬件设计方案主要分为两种，一种是针对于某一种网络算法，针对神经网络算法直接进行ASIC开发，另一种是则是基于FPGA平台使用ASIC设计方式对卷积神经网络进行加速。

目前，有一些较大的公司或者机构针对神经网络进行了ASIC开发。其中较为成功的有谷歌的TPU[17]以及寒武纪的NPU[18, 19, 20, 21]。TPU的核心在于脉动阵列，原理是数据以流水的形式向下游处理单元流动，充分利用数据的重用以减少所需带宽。脉动架构结构简单，灵活性低，但却十分适合卷积操作。寒武纪的NPU有一系列的芯片，其中针对于卷积神经网络的芯片为ShiDianNao，ShiDianNao的核心优化点在于取消使用DRAM进行数据的存放，其中处理器的物理位置比较靠近图像来源，这样就避免了图像数据的存放，而卷积操作参数量较少，可以直接存放在片上SRAM中。经过这样处理，就避免了DRAM的使用，从而降低功耗。但谷歌的TPU造价高昂且并不出售，寒武纪的TPU系列至今没有提供开发平台，这使得使用这两种芯片的途径被中断。

在FPGA平台上使用ASIC设计方式对某种特定神经网络进行加速的研究成果较多。赵文来、方睿等学者首先发表了在FPGA平台加速LeNet网络的研究成果[22, 23]。LeNet网络是一种结构非常简单的卷积神经网络，由于其参数量较小，因此可以将所有参数放入到FPGA的内部缓存中，可以减少卷积神经网络中的数据吞吐。李慧敏等学者则在FPGA平台上对AlexNet网络进行了加速[24]，AlexNet网络规模较大，由于FPGA内部BRAM资源有限，因此无法将参数存入FPGA中。该论文提出了将参数放入外部存储DDR中，在FPGA内部只进行中间计算数据的存储，同时为了减少所需外部存储的带宽，在计算全连接层时，采用批处理的方式，大大减少了参数加载所需带宽。之后，姚颂等学者在FPGA平台上对大型卷积神经网络VGGNet进行加速[25]，该论文提出，卷积操作是计算密集型运算，全连接层是访存密集型计算。对于卷积操作，设计了多个可并行计算的基本单元来进行加速，对于全连接层，则设置了较大的输入带宽来保证参数加载的速度。实际上，在FPGA平台上设计加速方案的一大难点就是卷积操作的功能单元设计，现有的加速方案基本都基于输入特征图并行、输出特征图并行和卷积窗口内部并行三种并行方式[26]，其中卷积窗口内部并行使用Z型单元[27, 28, 29, 30, 31]完成，具体内容将在第二章中详细给出。

尽管已经有许多学者基于FPGA平台对一些卷积神经网络模型进行了加速，但都只是针对于特定的网络模型，而没有一种普适的解决方案。而且，在FPGA平台上进行ASIC设计方式开发，对开发者的要求较高，需要同时对软件算法以及硬件结构有一定的了解，同时对数据的输入输出以及硬件实现算法中的一些优化都需要进行深入的勘察，这使得在FPGA平台上对卷积神经网络模型进行加速的人工成本非常高昂。因此，本文基于FPGA平台上进行ASIP设计的开发，旨在提供一种在FPGA平台加速卷积神经网络的便捷方案，使用者可以通过较少的软件层次的代码编写，就可以完成卷积神经网络的加速。同时，可以根据所使用FPGA内的资源设置加速方案，从而灵活的满足需求。

1.3 论文主要工作

本文旨在基于FPGA平台进行卷积神经网络加速方案的ASIP设计，其主要工作内容包括：

（1）计算框架设计及片内存储优化方案

卷积神经网络内部包括卷积层、池化层、批量归一化层、激活函数层、全连接层等不同形式的运算层。同时随着学术界的研究，卷积神经网络模型的深度也在不断加深。基于这两点，论文首先设计出不同运算层通用的计算框架，其不会因模型深度的增加而无法适用。再者，由于FPGA内部存储有限，往往会成为加速的瓶颈，基于上述计算框架，论文提出了两种优化方案，可以减少一半的存储资源使用。

（2）特殊功能单元及数据互联网络的设计

卷积神经网络的不同运算层有不同特点，其中最主要的运算层为卷积层与全连接层。卷积运算为计算密集型，而全连接层为存储密集型，根据不同层设计不同的功能单元，从而达到较优的加速结果，是本文的重点所在。同时，针对原有的卷积运算的Z型单元所存在的一些缺点，本文提出了一种旋转存储方案，可以在减少存储资源使用的同时，有效的减少在卷积操作步长大于1的情况下的运算时间。TTA架构的数据互联网络也是重点之一。优异的数据互联网络可以使数据传输所需时间变少，同时减少硬件结构的复杂性，从而减少功耗。本文基于上述的特殊功能单元间的数据流动，设计了数据互联网络，并与传统的互联网络进行了比较。

（3）各运算层的设计实现

如何通过确定的功能单元对任意参数的运算层进行加速，也是设计重点之一。本文将从TTA处理器的灵活性出发，通过分析各运算层的参数带来的计算变化，以能够实现任意参数的运算层为目的，对运算层的实现进行封装。使用者可以从外部简单的调用函数接口从而完成运算层的加速。

1.4 论文组织安排

本文的组织架构如下：

第一章，主要介绍了神经网络硬件平台的研究背景和发展现状，阐述了论文的研究目的与主要工作，并安排了论文的组织结构。

第二章，首先介绍了卷积神经网络的整体架构与各运算层的计算过程。其次对卷积操作的并行性研究进行了介绍，包括卷积窗口内并行所使用的Z型单元以及各种并行方式的组合。最后对TCE工具集的开发流程进行了阐述。

第三章，主要介绍了适用于处理器的计算框架和两种片内存储的优化方案。计算框架的设计使得处理器能够处理任意深度的卷积神经网络，大大加强了处理器的灵活性；片内存储的优化方案则可以减少FPGA内较为稀缺的片内存储资源的使用，从而减少开发成本。

第四章，主要介绍了基于TTA架构的功能单元及互联网络设计。基于TCE工具集，对片外存储功能单元、输入缓存功能单元、通用计算功能单元以及输出缓存功能单元进行了设计实现，同时设计了新的互联网络，使处理器的复杂度大大降低。

第五章，主要介绍了自定义卷积层的实现以及处理器的灵活性与性能分析。自定义卷积层的实现从处理器的灵活性出发，分析各运算层参数所带来的计算方面的影响，使用已经制定好的功能单元对任意参数的运算层进行加速。最后通过AlexNet网络的实现，将处理器与ASIC设计方式和CPU进行了比较，对处理器的灵活性与性能进行了详细分析。

第二章 相关技术研究

2.1 卷积神经网络结构详解

卷积神经网络是一种多层神经网络，在图像识别和自然语言处理等领域都发挥着非常重要的作用。卷积神经网络一般分为两大部分[32]，第一部分为提取特征的运算层，包括卷积层与池化层，第二部分为完成分类的运算层，包括全连接层。此外，卷积神经网络中还包括引入非线性从而提升模型表达能力的激活层[33]，以及加快网络训练速度的批量归一化层[34]。卷积神经网络结构如图2-1所示。下面将详细介绍各个层的计算过程。

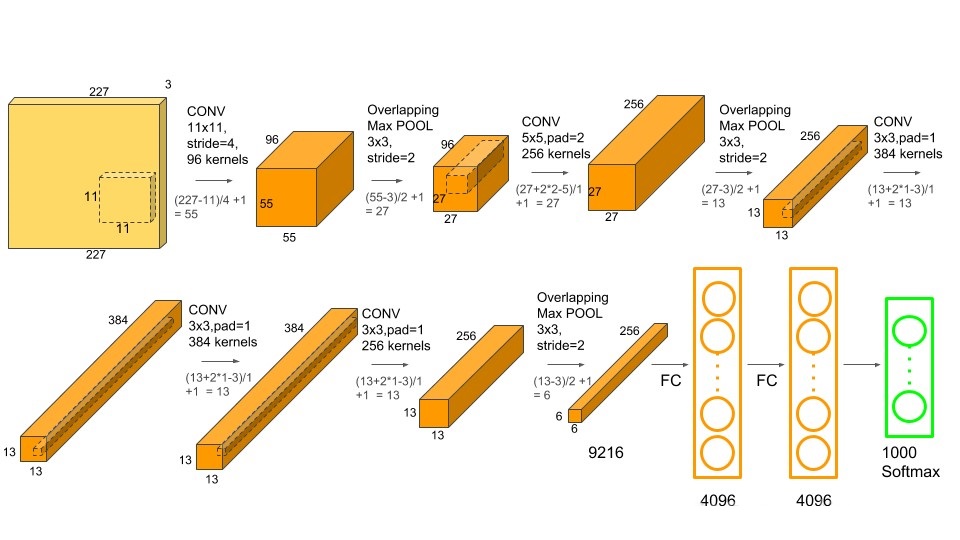


图2-1 卷积神经网络结构图

（1）卷积层

卷积运算，是将输入特征图与多个可学习的卷积核进行卷积运算，构成输出特征图的过程。一幅特征图的本质是一组离散的二维数组，对一副特征图进行卷积运算，其实际就是将一组离散的二维数据与另一组离散的二维数据进行对应位置相乘，然后进行求和。卷积运算可大概分为三个循环。最内层循环为单层输入特征图与单层卷积核之间的运算，根据卷积运算的参数，卷积窗口会不断滑动，最终产成单层输出特征图的中间结果，具体步骤如图2-2所示。第二层循环为对输入特征图的所有层进行遍历，输入特征图的所有层与单卷积核的所有层参数之间的运算，产生单层输出特征图。最外层循环为对卷积核进行遍历，最终产生所有输出特征图。卷积运算总过程如图2-3所示。



图2-2 单层输入特征图与单层卷积核的卷积运算

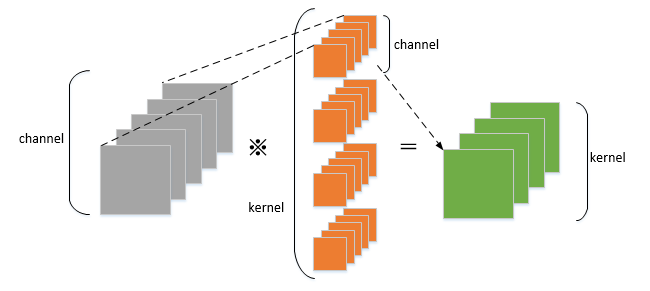


图2-3 卷积运算总过程

（2）池化层

池化层的主要作用是降低数据维度，从而大幅度减少计算量。池化操作与卷积操作较为相似，但主要不同的点在于，池化操作并不需要参数，本质上为下采样过程。而之所以可以进行池化操作的原因在于，即使减少了许多数据，但池化操作保留的是较为重要的一些数据，总体信息的损失程度较小。最大池化操作的具体计算过程如图2-4所示，根据池化操作的参数设定池化窗口与池化步长，池化窗口内比较出一个最大值作为输出，不断移动池化窗口，最终得到池化结果。池化操作种类较多，但最常用的为最大池化以及平均池化，因此本文仅对这两种池化运算进行讨论。

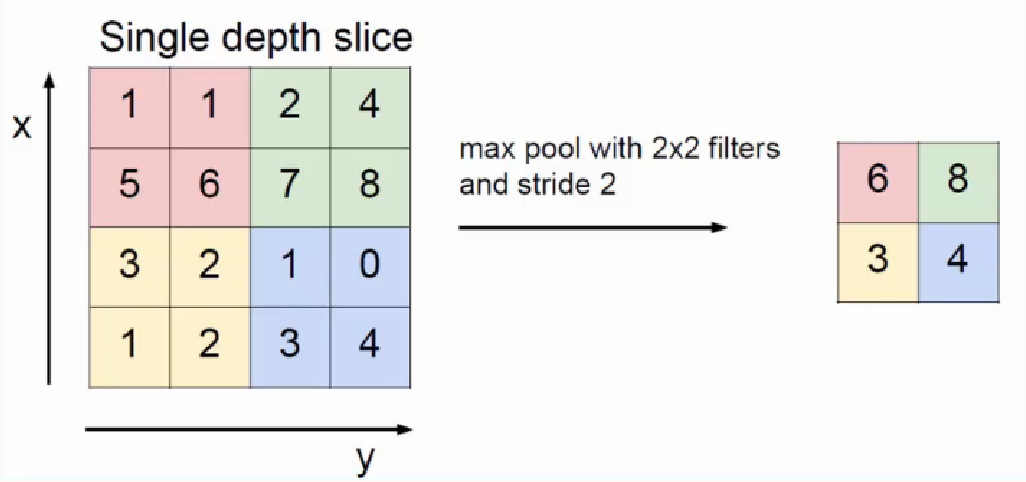


图2-4 最大化池化操作计算过程图

（3）全连接层

卷积神经网络的卷积层以及池化层会将原始输入数据映射到特征空间，而全连接层则会将网络提取的特征映射到样本的分类空间中。全连接层的特点在于参数量较大，单个全连接层的参数往往可以达到百万级别。全连接层将卷积输出的三维特征图转化为一维向量，与多个相同长度的一维向量进行点积，最终得到一个新的一维向量。在最后一层全连接层，最终输出的一维向量长度往往为样本分类数量。一个两层的全连接层如图2-5所示。



图2-5 全连接层结构图

（4）激活层

激活层的作用是为神经网络引入非线性变化，提升网络的非线性表达能力。若网络中不加入激活层，则在一定程度上可以看做单层网络，因为多层的参数矩阵相乘得到的仍为一个参数矩阵。激活层是神经网络中至关重要的一层，其也影响着网络的性能，目前常用的激活函数有tanh、sigmoid、ReLU以及后来基于ReLU提出的一系列改进函数等。激活层的实现较为简单，为单输入单输出模式，输出数据维度与输入数据维度一致。

ReLU函数是这些年应用较为广泛的激活函数，其数学公式为：

 （2-1）

其函数图像如图2-6所示：



图2-6 ReLU函数

（5）批量归一化层

归一化层的作用为对输入数据进行归一化处理，将所有数据的分布转为均值0方差1的分布，使得训练数据中数值在同样量级，从而达到加速训练的效果。批量归一化层是归一化层中的一种，因为其结构简单，加速效果显著，且有一定防止过拟合的作用，在近年来的神经网络中被广泛使用。批量归一化层指的是将一个批量即一层特征图内的数据进行归一化然后输出。其计算过程有计算均值、计算标准差、归一化以及变换重构，具体计算如下列公式表示：

 （2-2）

 （2-3）

 （2-4）

 （2-5）

2.2 卷积操作并行性研究

在卷积神经网络中，卷积操作是最为重要的一层，其操作结构较为复杂，计算复杂度最高，完成单层的卷积操作甚至需要亿级别的乘加操作。对卷积神经网络进行硬件加速，其中非常大的一部分任务是对卷积层完成硬件加速，而这需要深度挖掘卷积操作内部的并行结构，才能充分使用硬件进行并行计算。在章节2.1中曾提到，卷积操作有三层循环，其正代表着卷积操作内三种并行结构，分别为：卷积窗口内并行，输入并行图并行以及输出特征图并行。

2.2.1 卷积窗口内并行

卷积窗口内并行对应着卷积操作的最内层循环，它所要实现的是同时进行卷积窗口内的次乘法以及加法，在单个时钟内输出结果。FPGA具有天然的流水结构，通过设计乘法阵列以及加法树，可以非常轻松的实现上述功能。但单个卷积窗口的计算需要同时读取个输入数据以及个核参数，所需数据带宽较高，且控制逻辑较为复杂。而且因为FPGA内的存储资源只能读取连续数据，因此卷积窗口内的行输入数据需要个时钟进行读取，无疑会大大增加等待时间，减少计算效率。

实际上，相邻卷积运算窗口之间会有数据被重复使用，若能把这些数据存储起来，将不需要被重新加载。对于一个规模为的卷积核，设定卷积操作步长为1，则卷积窗口与其左边的卷积窗口之间会有6个数据的复用，再加上与右面、上面、下面三个方向的卷积窗口的数据复用，最终只有1个数据需要进行加载。学者们基于相邻卷积窗口内的数据重用，设计了Z型单元，以达到简化控制逻辑与减少数据、参数带宽的目的。

Z型单元的结构如图2-7所示，其主要分为两个部分：数据传输模块以及运算模块。



图2-7 Z型单元结构图

数据传输模块包括三部分：第一部分为规模为的寄存器阵列，用来存放卷积窗口内的数据；第二部分为个FIFO（First Input First Output），每个FIFO与前面一行的寄存器阵列共同存储一行数据，以保证寄存器窗口内数据的结构性；用来第三部分为参数buffer，用来存储卷积核参数。由于FIFO的深度与同行的寄存器个数相加为输入特征图的列数，因此可以保证寄存器阵列中数据为所计算的卷积窗口内的数据。在运算过程中，单个数据从寄存器阵列入口流入，寄存器阵列与FIFO中的数据依次向后移动一位，等效于卷积窗口向右移动一位，当所有数据流入完毕，得到计算结果。

运算模块包括乘法阵列以及加法树两部分，功能较简单，乘法阵列的输入为寄存器阵列以及参数buffer中的数据，将对应位置的数据与参数进行相乘。加法树的作用则是对乘法树的个结果进行相加，得到卷积结果。乘法阵列与加法树的设计一般都为流水线型，因此，在一个时钟内，就可以进行个乘法运算以及个加法运算，也就是说，在一个时钟内就可以输出一个卷积窗口的运算结果。

Z型单元相比只使用运算模块的设计有如下的好处。首先，利用数据的重用性，每次计算只需要加载一个数据，相比原先的个数据，大大减少了计算时所需要的数据带宽，同时也简化了数据加载逻辑。其次，卷积核的参数不需要被重复加载，在Z型单元中，参数可以预存在参数buffer中，每个卷积核的参数只需要加载一次，减少了所需参数带宽。

而Z型单元也有其不足之处。首先，Z型单元需要额外的片内存储来进行重用数据的存储，在使用多个Z型单元进行加速时，所消耗的片内存储资源不可忽视。其次，对于卷积步长大于1的卷积操作，其相邻卷积窗口之间的数据复用率变低，此时可能需要加载多个数据才能进行计算，而Z型单元每个时钟只输入一个数据，导致等待时间大大增加。对于的特征图、的卷积核、步长为的卷积运算，使用Z型二维卷积基本单元所花费的时间为：

 （2-6）

而实际有效运算所消耗的时间为：

 （2-7）

所以，计算资源的利用率为：

 （2-8）

可见，当卷积运算的步长大于1时，计算资源的利用率以平方的速率降低，在实际的应用中会使得计算资源的严重浪费。

2.2.2 输入特征图并行

卷积窗口内并行解决的是单层输入特征图与单层卷积核参数之间的并行，而输入特征图有多层，结构相似，其并行性也可以进行开发。输入特征图并行方式单次计算过程如图2-8所示。输入特征图并行将个输入特征图同时进行计算，也就是同时使用个卷积窗口内并行，在当前的个输入特征图计算完成后，切换到下一组个输入特征图继续计算。由于个输入特征图的计算结果都为一个输出特征图的中间结果，因此需要在个输入特征图并行结果后加入加法树结构，再将结果暂存到输出特征图中。

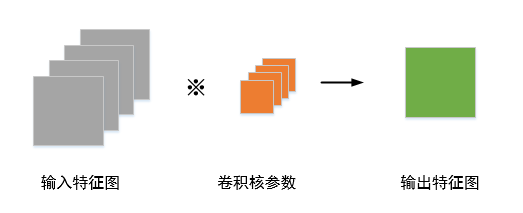


图2-8 输入特征图并行计算

输入特征图并行利用输入特征图不同层之间运算的相同性，虽然数值不同，但运算过程相同，因此，可以使用相同的控制逻辑同时对多个层同时进行计算。其具体步骤如下：

a）切换下一组输入特征图，加载卷积核该层对应的参数

b）从输入buffer中同时读取层输入特征图的数据，输入到卷积功能单元中进行计算。

c）将个卷积功能单元结果进行相加，中间结果存储到输出buffer中。

d）重复a、b、c，得到输出特征图的一层结果，切换卷积核。

e）重复d，得到输出特征图的最终结果。

2.2.3 输出特征图并行

输出特征图并行单个输入特征图与多个卷积核共同运算，生成输出特征图多层的并行方案，与输入特征图并行的思路相反。输出特征图并行单次计算过程如图2-9所示。由于输出特征图并行只需要输入特征图的一层参与运算，因此个卷积功能单元共享输入数据，个卷积功能单元的不同之处仅在于卷积核参数的不同。

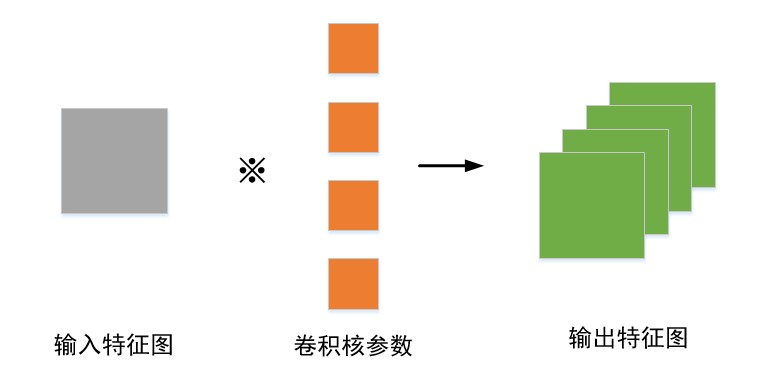


图2-9 输出特征图并行计算

输出特征图并行的一个特点是输出中间结果较多，每一层输入特征图都对应着一部分中间结果，当下一层输入特征图计算时，需要将前面层的中间结果读出来与当前层结果进行累加，在控制逻辑上可能会略显复杂。输出特征图并行的具体计算过程如下：

a）切换下一层输入特征图，加载个卷积核对应参数。

b）从输入buffer中读取该层输入特征图数，输入到卷积功能单元中进行计算。

c）将个中间结果暂存到输出buffer中。

d）重复a、b、c，得到输出特征图的中间结果。

e）重复d，得到输出特征图的最终结果。

2.2.4 并行性组合

在实际设计并行架构时，多种并行结构其实并不冲突，而且最常见的方案是设计卷积功能单元来完成卷积窗口内的并行，同时选择输入特征图并行以及输出特征图并行中的一种或者两种。但在选择两种并行方式时，仍需要确定一种主并行方式。

假设所采用的并行组合：卷积窗口内并行度为，输入特征图并行度为，输出特征图并行度为。在选择以输入特征图并行为主并行方式时，计算流程与2.2.2并没有太大的区别，只是每次运算得到的为层输出特征图。而所采用的主并行组合为输出特征图并行时，则是层输入特征图同时参与运算，得到所有层输出特征图的中间结果。可以认为，两种并行方式共用，是对输入特征图或者输出特征图并行方式的一种电路上的复制，本质上并没有变化。

但对于输入特征图并行结构，仍有一点需要考虑。在实际应用中，第一层卷积层的输入往往是真实图片，其通道数最大为3，也就是输入特征图只有三层。若此时输入特征图并行度设计过高的话，多出的电路并不会进行有效计算，浪费计算资源。

2.3 TTA架构及TCE开发工具集

2.3.1 TTA架构介绍

TTA架构由VLIW（Very Long Instruction Word）架构[35, 36]发展而来。

VLIW架构是一种利用指令级并行优势的架构，其通过编译器调度，将数据不想关的若干条子指令打包成一条长指令执行，以提高运行速度。VLIW架构具有非常高的扩展性以及灵活性，用户可以在VLWI架构中添加任意数量的可以实现任意操作的功能单元。VLIW架构示意图如图2-10所示。

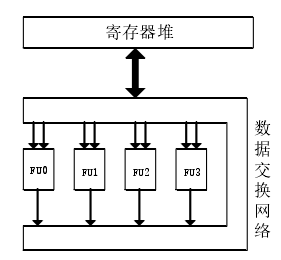


图2-10 VLIW结构示意图

在VLIW中，用于中间数据存储的称作寄存器堆，其由多个寄存器文件构成。寄存器堆与功能单元之间的数据网络以及功能单元与功能单元之间的旁路网络，统称为数据交换网络。VLIW架构主要存在两个较大的问题。首先，要考虑到最差的情况，寄存器堆要与每个功能单元进行数据交互，因此寄存器堆的输入输出接口数量将随着功能单元的数量以及功能单元接口的数量曾线性增长。其次，为了使功能单元之间的数据进行互通而不需要经过寄存器堆，VLIW在功能单元与功能单元之间设置了旁路网络，如果网络是全联通的话，复杂度将随功能单元的个数平方级增加，造成较大的性能损失。

在对VLIW的瓶颈进行考虑的基础上，荷兰Dulft大学的Corpraal教授提出了TTA架构[37]，即传输触发架构。TTA架构的核心思路之一为将寄存器文件看做一种特殊的功能单元，以减少架构的复杂性。一个典型的TTA架构如图2-11所示。

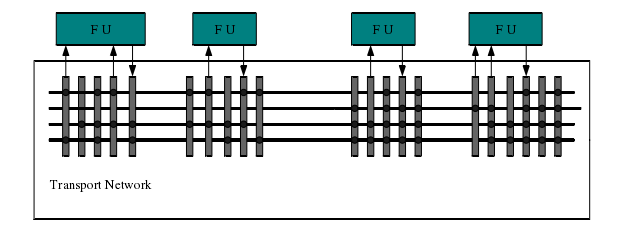


图2-11 TTA架构示意图

TTA与VLIW的另一点区别在于，传统的VLIW是基于操作触发的处理器，其指令为操作，对操作所需的数据进行传输以完成运算。而TTA则是基于数据传输触发的处理器，其指令不需要描述执行的操作，而是对操作所需的数据进行传输，操作是数据传输的一个边沿作用。TTA架构的指令单元格式非常简单，这也大大降低了译码的复杂性。

在汇编层次上来说，VLIW架构的指令需要包含三个部分：操作码、源地址、目标地址。TTA架构的指令则只需要包含两个部分：源地址、目标地址。下面是两个数相加的TTA架构指令：



第一条指令将f1功能单元的out1接口寄存器中的数据搬移到add功能单元的in2接口，第二条指令将f2的out2接口寄存器中的数据搬移到add的t接口，由于add的t接口为触发接口，因此执行加法操作。在一个时钟之后，第三条指令将add的out1接口数据搬移到f3的in1接口。

当然若总线数量足够，数据的搬移也可在一个时钟内完成，加法操作在两条总线时的指令如下：



2.3.2 TCE工具集介绍

TCE是一个设计和编程TTA传输触发架构定制处理器的工具集，它提供了一种完整的可重定向的软硬件协同设计流程。TCE支持用户使用C / C ++和OpenCL编写的程序，用户可以轻松设计新的TTA架构处理器，编译程序，分析性能，并生成设计处理器的HDL实现，下载到FPGA芯片上运行。TCE的一大优点在于软硬件协同，用户可以先在软件上进行仿真分析性能，当性能满足需要时，再进行硬件实现。下面将对TCE工具链展开介绍，主要使用的工具如图2-12所示。

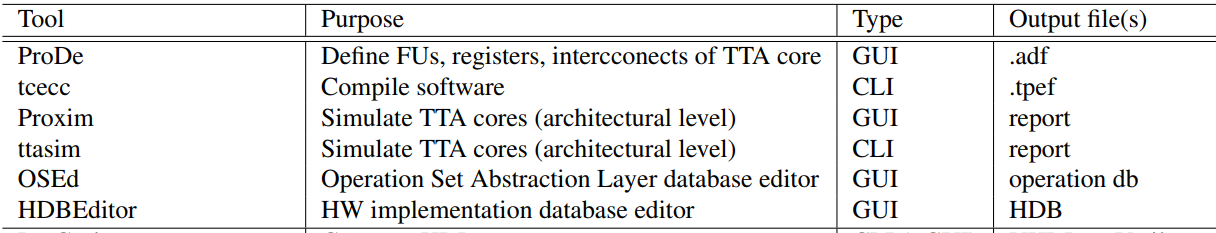


图2-12 TCE工具集

（1）ProDe

ADF（Architecture Deﬁnition File）文件是一种用于定义目标处理器体系结构的文件格式，其包含了自定义的TTA处理器的架构信息，如使用了哪些功能单元、功能单元之间的连接等。一个简单的ADF文件如图2-13所示。

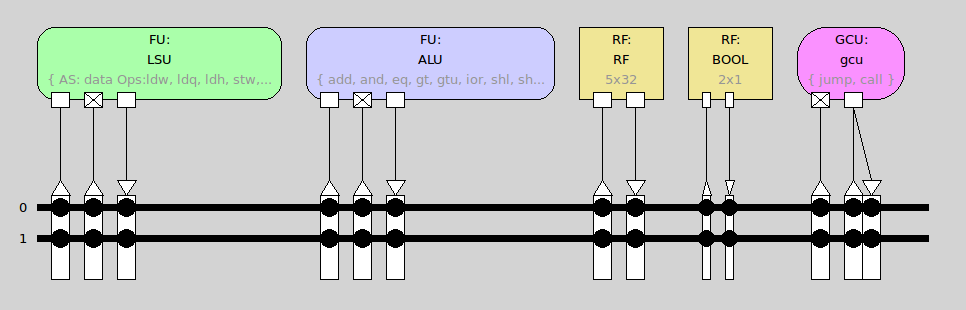


图2-13 架构文件示意图

ADF文件中主要包含了六个部分，分别为：功能单元、数据总线、输入输出接口、指令单元、立即数单元以及寄存器单元。但对于一些比较简单的处理器设计，所设计的内容主要有：

1. 特殊功能单元数量及内部操作延时

2. 功能单元输入输出接口数量

3. 总线数量及总线与输入输出接口间socket通信定义

4. 寄存器文件的大小及数量

ProDe工具的作用主要是查看和编辑ADF文件，同时也可为所使用功能单元定义HDL实现，以便整个工程可以生成二进制文件。

（2）tcecc

tcecc工具是TCE工具集中的编译工具，其可以将用户提供的高级语言源文件，编译为二进制程序或tpef文件。二进制文件是一种不针对任何架构的特定顺序程序，适用于通用处理器执行；而tpef文件则是TCE工具集中定义的文件，其为针对目标体系架构优化的特定执行文件。实际上，二进制文件仍可以转换为tpef文件，只需加入adf文件即可。用户可以将源代码编译为二进制文件，这样在对处理器架构进行修改之后，都无需再次编译源代码。

Tpef文件可以看做指定调度汇编文件的一种编码文件。在tce工具集中，用户甚至可以通过tceasm与tcedisasm工具在tpef文件以及汇编文件中进行转换，在需要对核心指令进行修改时十分有用。

（3）proxim、ttasim

Proxim与ttasim工具都为仿真工具，两者的不同之处在于，proxim使用交互界面来进行显示，ttasim则使用命令行的交互形式。仿真工具可以查看执行工程所运行时钟数、各功能单元的使用率、总线的使用情况等。因此在TTA架构的开发中，仿真处于一个比较重要的地位，通过仿真可以分析工程的待优化点，迭代进行开发，直到满足性能需求为止。在本文中，为了更方便的显示仿真结果，因此选择proxim工具进行仿真，proxim工具界面如图2-14所示。

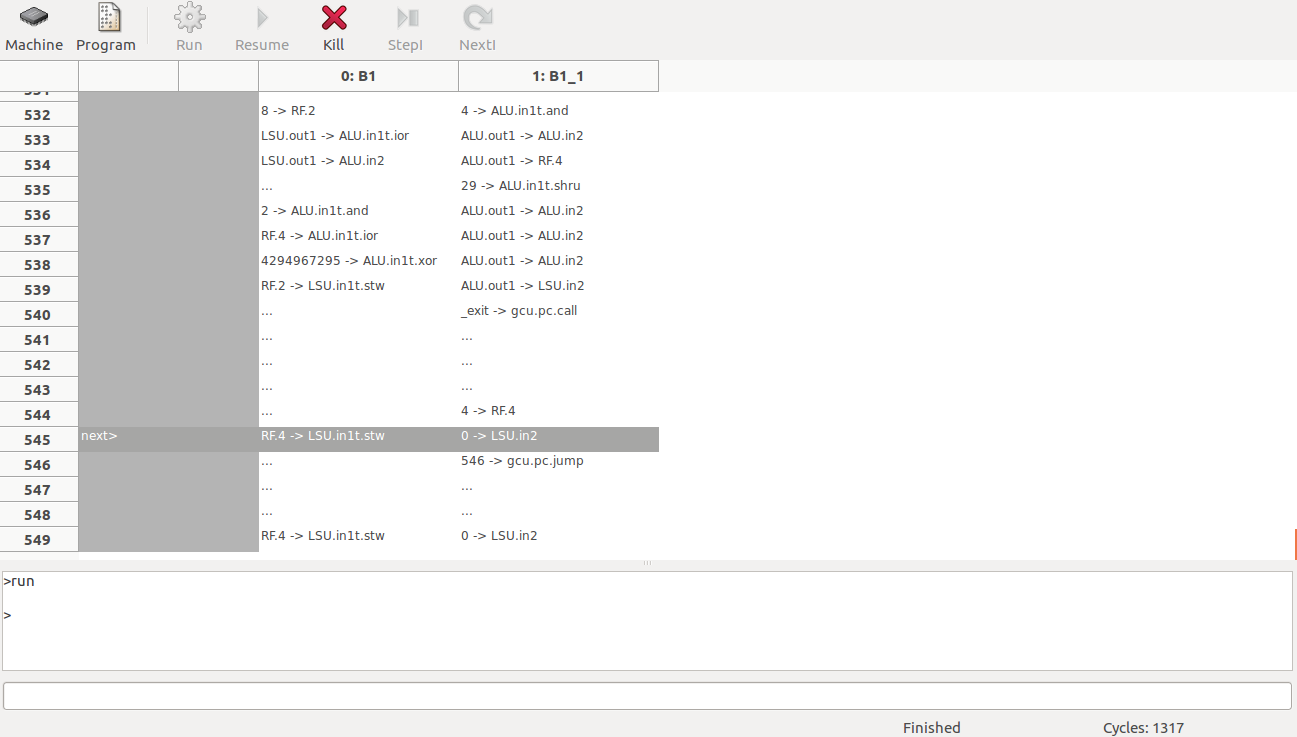


图2-14 图形化仿真界面图

由tcecc工具产生的tpef文件本质为汇编文件，控制指令的跳转，其并不包含所适应架构的内容，因此仿真工具仍需要架构文件作为输入。

（4）Osed

Osed（Operation Set Abstraction Layer database editor）是一个用户管理自定义操作的图形化界面，包括添加自定义操作的高级语言模拟代码以及静态属性。TCE工具集的一大优点在于，用户在开发出自己满意的架构前，不会涉及到硬件语言的开发。Osed使用高级语言来模拟硬件语言进行操作的定义，这样就可以在开发过程中节省许多人力成本。在仿真时，仿真工具会调用Osed中操作的定义计算结果，而不是硬件描述语言。Osed的主界面如图2-15所示，其所包含的功能主要有：

1. 添加新操作到模块或者搜索路径中。

2. 检查和编辑操作属性。

3. 对操作进行仿真。

4. 对操作进行编译。

5. 查看和编辑内存中的内容。

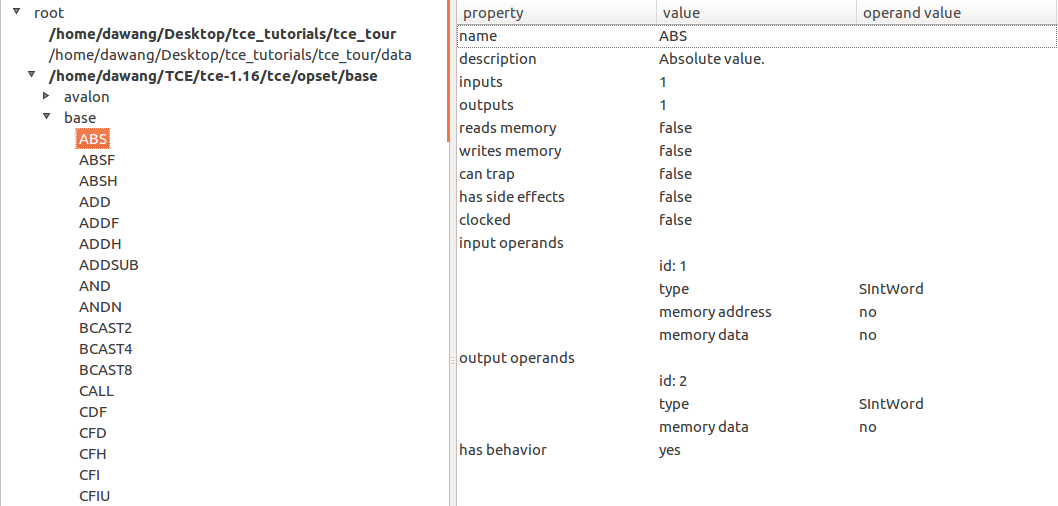


图2-15 osed界面图

（5）HDBeditor

在前面的工具中，为了易于迭代开发，一直使用高级语言来进行自定义操作的行为描述，但最终无法生成可供FPGA执行的比特文件。HDBeditor则是用来创建和修改硬件数据库的图形交互界面工具。在HDBeditor中，适用于不同平台的RF、GCU、LSU以及ALU等通用功能单元已经存在，因此用户只需要对自己定制的特殊功能单元进行HDL实现即可。在对架构文件中所有功能单元选择HDL实现后，即可通过generateprocessor工具以及generatebits工具实现整个工程的二进制文件生成。

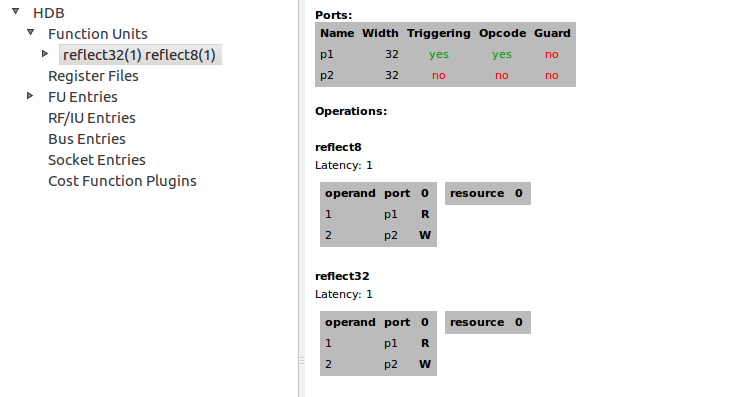


图2-16 HDBeditor界面图

HDBeditor的图形界面如图2-16所示，功能单元HDL描述中一般需要以下信息：

1. 功能单元实体命名

2. 输入参数

3. 功能单元中各自定义操作的执行码

4. 各自定义操作架构端口

2.3.3 TCE设计流程

TCE的主要设计流程如图2-17所示。

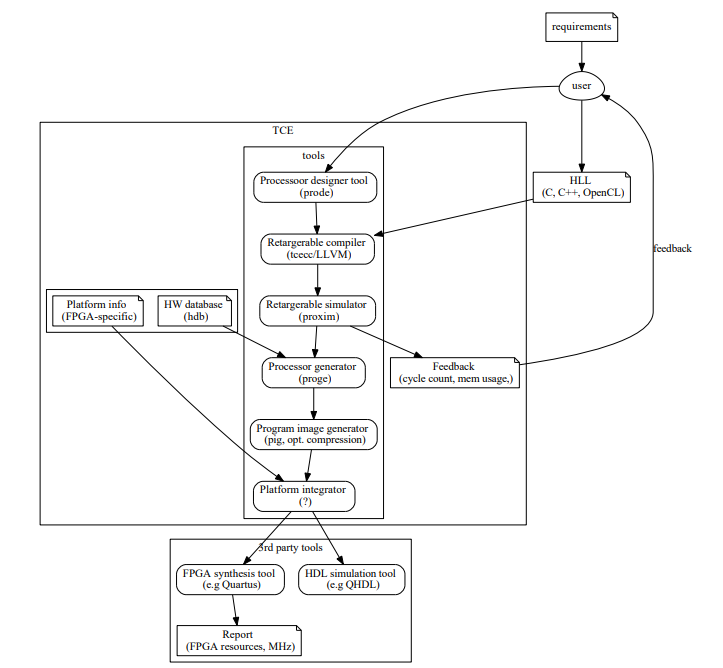


图2-17 TCE工具集设计流程

一般来说，开发一个基于TTA架构的处理器，首先要确定所使用的算法。在TCE工具集中，算法可以使用C/C++或者OpenCL高级语言进行实现，一般选择C/C++。由于TTA的指令调度是在特定的处理器架构上进行的，因此仍需要定义一个初始架构。初始架构并不需要尽可能的完善，它可以仅能完成算法所需，但之后会慢慢改进。因为所有的高级语言最终都可以编译为简单的加减乘除、取模赋值以及逻辑表达等简单的操作，在最初的处理器中，只需要包含这些简单的运算单元就可以确保算法的完整进行。同时，TCE工具集提供了一个最为简化的初始架构可供使用，架构中只包含1条总线以及5个功能单元：1个GCU（Global Control Unit）用来控制指令的读取与跳转；两个RF文件（Register Files）用来暂存数据；1个ALU（Arithmetic Logical Unit）用来进行基本运算；一个LSU（Load–Store Unit）用来与内存间进行数据交互。

在高级语言程序开发完毕以及初始架构确定之后，我们就可以知道算法的运行情况。通过tcecc工具，可以通过高级语言代码以及架构文件生成我们所需的tpef文件，之后就可以使用proxim工具进行仿真。在仿真结束后，有两个结果需要特殊关注。第一个是运行时钟数，可以直接在运行结束界面右下角直接观察，也可以通过指令来进行读取。第二个是各个功能单元以及总线等的资源使用情况，在view->Machine Window中可以直观的进行观察，图形界面如图2-18所示。其中，白色代表使用率较低，红色代表使用率较高，颜色越深时，使用率越高。

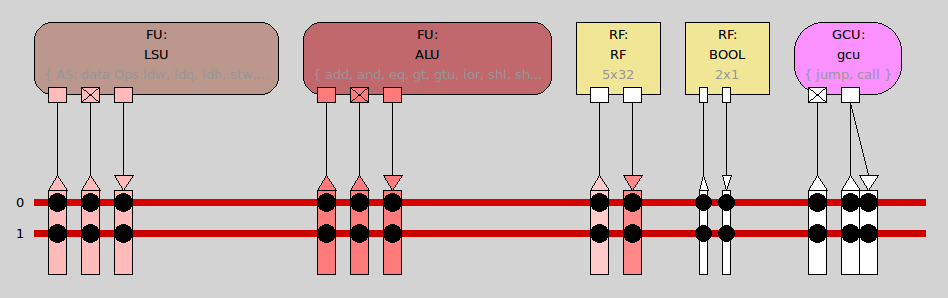


图2-18 资源使用图

在第一次仿真实现结束后，虽然我们已经可以得到算法实现的处理器架构，但其性能可能并不能满足我们的需求，因此，需要对架构进行优化。在TTA架构中，一个比较常用的优化方法就是设计自定义操作，将较为占时或者复杂的计算放入自定义操作中进行，从而简化控制逻辑进而减少时间。一个设计自定义操作的过程如下：

1.分析瓶颈代码

2.使用OSEd向操作数据库添加新模块和操作并定义新操作的行为。

3.修改高级语言源代码，利用新操作实现算法。

不同算法有不同算法的瓶颈。在仿真时，通过资源的使用情况一般可以得到一个较广的判断，如加法操作过多或者数据读取太过频繁等。前者我们可以为使用加法较为频繁的运算设计新操作，后者我们也可以将数据交换的两部分运算封装在一个操作中以减少数据与寄存器文件之间的交互，或者也可以增加总线数量增大数据传输带宽。

在确定所要设计的新操作后，我们需要使用OSED工具对新操作进行行为定义，一个简单操作的行为定义如下：

OPERATION(ADD\_4)

TRIGGER

int result = 0;

result += INT(1);

result += INT(2);

result += INT(3);

result += INT(4);

IO(5) = result;

END\_TRIGGER;

END\_OPERATION(ADD\_4)

上述操作实现的功能将输入的4个数据进行相加，输出结果。其中OPERATION中定义操作名称，INT(1)~INT(4)表示将4个输入端口中的数据按int型处理，IO(5)表示结果从5号端口输出。在高级语言代码中若需要使用新操作，则需在操作名前加\_TCE\_，以与普通函数进行区分，上述操作的调用代码为：

\_TCE\_ADD\_4(data1, data2, data3, data4, sums);

当我们加入了自定义操作并且加速效果满足需求后，此时我们就需要对自定义功能单元进行HDL描述，以生成可供FPGA执行的二进制文件。通过HDBeditor，用户可以将自定义功能单元的HDL加入到HDB中。在架构文件中，对不同功能单元可以在HDB中选择相应的HDL描述。HDBeditor对功能单元的定制有一套规范，在2.3.2中已经进行过描述。当所有功能单元的HDL描述完整之后，就可以使用ProGe工具生成最终的FPGA执行文件。

2.4 本章小结

本章的主要内容主要包括卷积神经网络结构详解、卷积操作并行性研究、TTA架构的介绍以及TCE开发工具集的说明。首先，简单介绍了卷积神经网络的结构，详细说明了卷积神经网络中常用运算层的运算形式。其次，针对卷积神经网络中运算量最为庞大的卷积层，分析了卷积窗口内并行、输入特征图并行以及输出特征图并行三种并行方式的实现设计方式，并对三种并行方式的组合进行了说明。接着，简单介绍了TTA这种新型架构，其是如何克服VLIW的局限发展而来。最后，对开发TTA架构的TCE工具集给了详细的介绍，主要包括其中主要工具的使用以及针对特定算法的开发流程。

第三章 大规模卷积神经网络计算框架设计

3.1 计算框架设计

卷积神经网络的最主要的两种运算层为卷积层以及全连接层。卷积层的计算量较为庞大，但其计算所需的数据是输入特征图以及参数的重复使用，可以使用片内存储来减少数据的访存次数，卷积运算可以看做为计算密集型运算。而全连接层计算量较小但参数较多，因此需要较大的访存带宽来支持参数的加载，全连接层可以看做为访存密集型运算。因此，在卷积神经网络的硬件加速中，存在两种瓶颈：乘法器资源以及片外存储带宽。在乘法器资源较少的情况下，乘法器资源会成为加速的瓶颈，而在乘法器资源较多的时候，片外存储带宽则会成为加速瓶颈。

由于卷积操作需要多次读取输入特征图以及多次写入输出特征图，计算框架需要设置片内缓存来减少数据带宽的使用。因此计算框架一般分为三个部分：片外存储，片内缓存以及计算单元。

为了减少框架的复杂性，在最初的计算框架设计中，往往将参数以及中间数据都放在片内存储中以方便读写。在实现小型网络如LeNet时效率较高，只需从片外存储中定期读取图片数据，在FPGA内部完成运算后将结果输出即可。但当卷积神经网络变的更加复杂时，由于全连接层的参数量非常大，能够支持片内存储参数的FPGA造价十分高昂，这种计算框架不再适用。

自然而然，将参数存储在片外存储上成为一种新的设计思路，实际上，这也是加速卷积神经网络的ASIC设计中最为广泛采用的一种计算框架。如图3-1所示，这种计算框架比较适用于运算层级别的并行，因此被称为并行计算框架。并行计算框架将大容量的参数预存在片外存储中，将容量相对较小的中间结果存储在片内存储上，以较少片内存储的使用。在实际的计算中，FPGA需要首先从片外存储中读取图片数据，再在每层的计算过程中分别加载参数，最终将结果输出。并行计算框架的一个缺点在于可扩展性不强，由于片内存储需要存储所有的中间结果，当网络层数加深时，FPGA内的存储资源并无法支持。实际上，若使用这种计算框架，一般的FPGA能支持的网络层数仅在10层左右，无法达到设计处理器的要求。

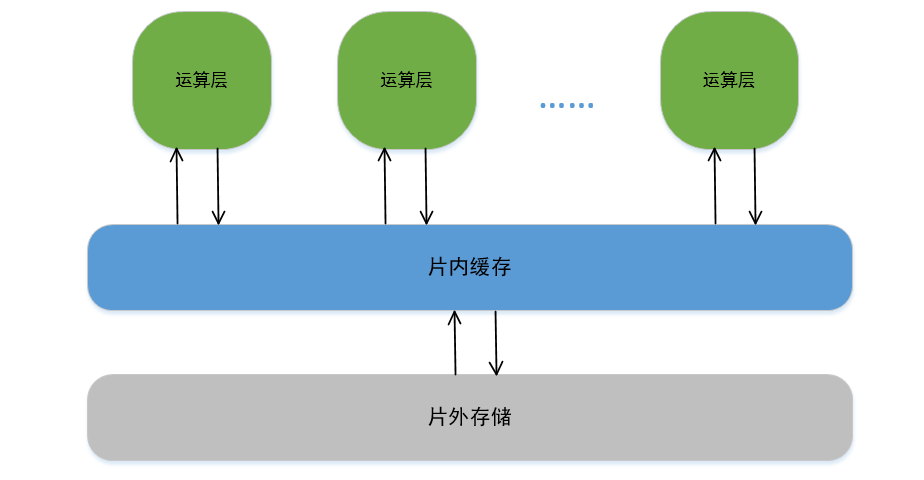


图3-1 并行计算框架图

在深度学习的领域中，一个发展的趋势是卷积神经网络变的越来越深，2012年ImageNet的冠军为8层的AlexNet网络，而2015年冠军则为152层的ResNet，到2016年，获得冠军的网络层数甚至达到了1207层。卷积神经网络越深，则其所含有的表达能力越强，所能达到的性能也就越佳。而对于处理器设计来说，如何用有限的资源来处理越来越深的卷积神经网络，即如何设计具有高扩展性的计算框架，也是一个不容忽视的重点。

满足处理器设计的计算框架应注重于单层运算层的计算，而不受其它层影响。基于这个需求出发对上述计算框架进行改进，在片内存储中并不需要存储所有中间结果，而只需要存储当前层的中间结果，即输入数据以及输出数据。如图3-2所示，这种计算框架注重于运算层之内的并行，而无法进行运算层之间的并行，称为串行计算框架。在卷积神经网络的发展中，虽然网络的深度一直在增加，但是单层所需的空间以及计算量并没有太大的变化，使得串行计算框架的扩展性变的很强，可以满足处理器加速任意深度的卷积神经网络的需求。

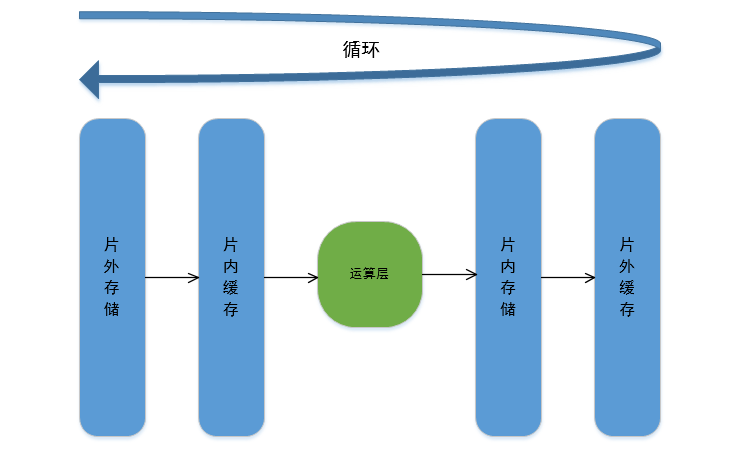


图3-2 串行计算框架图

但串行计算框架同时也带来了一些性能上的损失。在计算过程中，FPGA不仅需要从片外存储中加载参数，同时还需要加载输入数据，在计算完毕后，需要向片外存储中写入输出数据。与并行计算框架相比，串行计算框架在每一层的运算过程中需要多出一次数据读取以及一次数据写入过程。

实际上，所造成的性能损失并不会太大的影响加速效果。假设卷积神经网络的参数量为，单层的中间数据量最大为，所有层的中间数据总和为，网络层数为。在最坏的情况下考虑，即硬件平台的加速瓶颈为片外存储的带宽，串行计算框架所需的片内存储资源：

 （3-1）

并行计算框架所需的片内存储资源为：

 （3-2）

串行计算框架所需的片外存储访存次数为：

 （3-3）

并行计算框架所需的片外存储访存次数为：

 （3-4）

将串行计算框架与并行计算框架的片内存储资源进行比较：

 （3-5）

由于在卷积神经网络中，各层中间结果的数据量差别并不是很多，因此可近似等于。也就是说并行计算框架所需的片内存储资源量约为串行计算框架的倍。

将串行计算框架与并行计算框架的片外存储访存次数进行比较：

 （3-6）

在卷积神经网络中，参数的数据量与中间数据的比值往往在十倍甚至以上，所以，的值通常在1.1以内，数据带宽的需求增长并不是特别明显。

综合两种资源的使用情况，串行计算框架在增长了不到10%的片外存储带宽资源的条件下，片内存储资源的使用降低了倍。同时，串行计算框架易于扩展，可以进行任意深度的卷积神经网络加速，适用于卷积神经网络处理器的设计场景。

3.2 片内存储优化方案

相对于并行计算框架，虽然串行计算框架已经对片内存储的使用进行了优化，但FPGA的片内存储资源仍然十分紧缺，对片内存储的使用进行优化仍然是必要的。基于所加速的运算均为单层输入到单层输出、多层输入到单层输出或单层输入到多层输出的形式，本节提出了两种优化结构，在不引入额外的数据读取以及参数读取的条件下，可以将片内存储资源的使用降低一半左右。

3.2.1 输出缓存优化结构

在章节2.2中对卷积操作进行了并行性研究，针对卷积的循环展开顺序，提出了输入特征图并行以及输出特征图并行两种并行方式。输出缓存优化结构则是针对于输入特征图并行方式进行的片内存储优化。架构如图3-3所示。其核心思想在于每次计算单层输出特征图，在计算完毕后，将输出数据写入片外存储中，进行下一层输出特征图的计算。

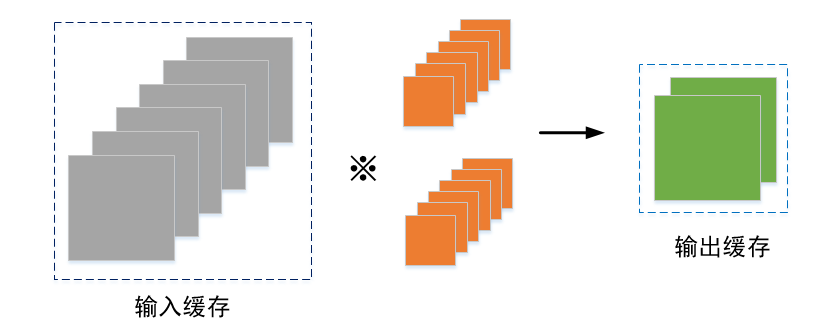


图3-3 输出缓存优结构

在这里先讨论输出缓存优化结构可以存在的确定性。对于本文所要加速的五种运算层，其中池化层、激活层、批量归一化层均为单层输入计算单层输出的运算，全连接层的输入输出可以看做只有一层，因此这些运算层并没有输出特征图之间的运算。而卷积运算通过更改循环展开的顺序，也可以看做是一种多层输入到单层输出的运算，这使得输出特征图的不同层之间存在独立性，可以依次对每层进行计算。

在输出缓存优化结构中，输入缓存中存储所有层输入特征图数据，在选择加速方案时，可以设定任意倍数的输入特征图并行度而不需要加入额外的存储空间。而如果需要增加输出特征图并行度，则需要添加额外的输出缓存空间。

设输入特征图并行度为，输出特征图并行度为，单层输入特征图面积为，输入特征图层数为，单层输出特征图面积为，输出特征图层数为，若不进行优化，则所需存储面积为：

 （3-7）

输出缓存优化结构所需存储面积为：

 （3-8）

一般来说，在输出缓存优化结构中，输出特征图并行度设置较小，所以在优化之后，存储输入特征图所占面积占绝大部分，相比未优化前需要同时存储输入特征图与输出特征图，优化后只需约一半的片内存储资源。因此，为了减少所存储的面积，在采用输出特征图优化方案是，应尽量选择高输入特征图并行度，低输出特征图并行的加速方案。此时的计算流程如下：

a）从片外存储加载所有层输入特征图到输入缓存中。

b）从片外存储加载个卷积核的层参数至计算单元中。

c）进行运算，得到层输出特征图中间结果。

d）重复步骤b与c，对个卷积核的所有层参数进行遍历，得到层输出特征图最终结果。将结果写入片外存储。

e）重复步骤a、b、c、d，得到整个输出特征图的结果。

3.2.2 输入缓存优化结构

与输出缓存优化结构相似，输入缓存优化结构是针对输出特征图并行进行的片内存储优化。其核心思想为，单次从片外存储中加载一层输入特征图，在它所参与的计算完毕后，加载下一层输入特征图，架构如图3-4所示。

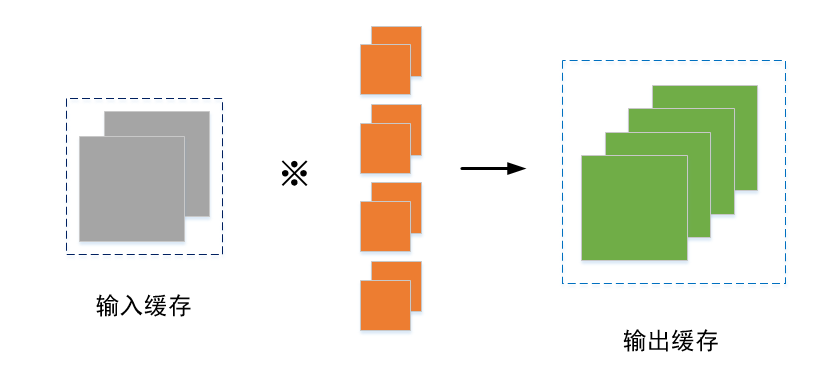


图3-4 输入缓存优结构

在输出特征图并行方式下，卷积操作的计算形式为单层输入到多层输出，输入特征图的各层之间存在独立性，可以适用输入缓存优化结构。在输入缓存优化结构中，可以设定任意倍数的输出特殊图并行度不需要加入额外的存储空间，若需要增加输入特征图并行度，则需要添加相应的输入缓存空间。输入缓存优化所需的存储面积为：

 （3-9）

与输出缓存优化结构类似，由于较小，所以输入缓存优化结构相比未优化之前，也只需约一半的存储面积。此外，值得一提的是，由于第一层卷积层的输入往往是真实图片，其通道数最大为3，也就是输入特征图只有三层。若此时输入特征图并行度设计过高的话，多出的并行电路并不会进行有效计算，浪费计算资源。此时低输入特征图并行度、高输出特征图并行度的输入缓存优化结构更具优势。

输入缓存优化结构在输入并行度为以及输出并行度为的条件下的计算流程如下：

a）从片外存储加载层输入特征图到输入缓存中。

b）从片外存储加载个卷积核的层参数至计算单元中。

c）进行运算，得到层输出特征图中间结果。

d）重复步骤b与c，对所有卷积核的层参数进行遍历，得到所有层输出特征图的中间结果。

e）重复步骤a、b、c、d，得到整个输出特征图的结果。将结果写入片外存储。

3.3 本章小结

本章的主要内容包括大规模卷积神经网络计算框架的设计以及片内存储的优化方案。首先分析了两种计算框架的设计方案，从处理器所需的高扩展性出发选择了可以对任意深度的卷积神经网络进行加速的串行计算框架作为处理器的计算框架。其相比于并行计算框架，在所需片外存储带宽不超过10%的条件下，可以将片内存储资源的使用降低倍，为网络层数。其次，针对FPGA的片内存储资源进行了优化，结合卷积操作的输入特征图并行以及输出特征图并行两种并行方式，提出了两种片内存储的优化方案。其在不引入数据及参数的读取的情况下，将片内存储资源的使用降低了大约一半。

第四章 基于TTA架构的功能单元及互联网络设计

本章将对基于TTA架构的卷积神经网络加速器设计进行详细介绍。在TTA架构中，设计的主要内容包括功能单元及互联网络两个部分。针对于卷积神经网络加速所设计的功能单元包括：DDR功能单元、输入缓存功能单元、通用计算功能单元、输出缓存功能单元。在下面的小节中将进行详细介绍，包括功能单元内的操作设计以及输入输出接口设计。此外，为了适应不同FPGA平台的资源情况，功能单元的设计将留出三个参数接口供用户来制定，以达到最优的加速效果。三个参数分别为输入特征图并行度、输出特征图并行度及全连接层并行度，具体的参数确定方案将在第五章中讨论。

4.1 片外存储功能单元设计

在第三章所设计的计算框架中，处理器需要与片外存储之间进行数据互通。在TTA架构中，虽然有着LSU（Load-Store Unit）功能单元，但其只负责与FPGA片内资源所构建的内存进行数据交互，与片外存储之间仍需要设计电路来进行数据互通。片外存储功能单元的主要功能即为处理器与片外存储之间的数据交互，其所实现的功能为：从片外存储中读取数据、向片外存储中写入数据。

不同的FPGA厂商会开发不同的设计工具，在设计工具中，厂商会开发针对不同场景所使用的IP核以方便用户的使用。Xilinx是全球最大的FPGA厂商，在其所开发的设计工具中，针对于特定FPGA或者套件中DDR的使用，也有相应的IP核的开发。在Xilinx的设计工具中，用于DDR控制的IP核简称为MIS核（Memory Interface Solution），接口设计如图4-1所示。

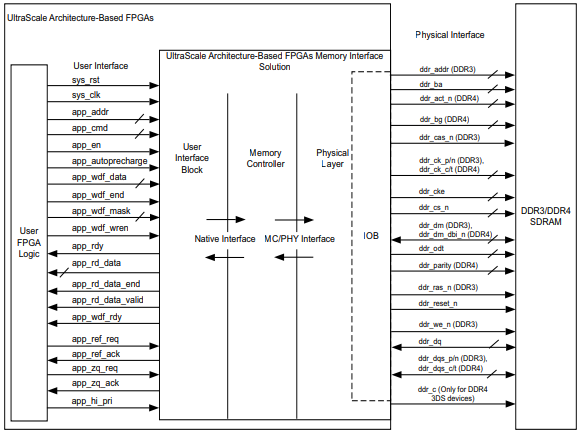


图4-1 内存控制IPcore接口图

MIS主要分为三个部分，第一部分为用户接口模块，第二部分为内存控制模块，第三部分为物理层。用户在FPGA设计中，将读写指令写入用户接口模块，用户接口模块将指令传送给内存控制模块以进行解读，之后将解读后的模块发送给物理层，物理层则将信号通过连线传送给SDRAM，完成一次操作。在用户接口模块中所定义的信号则是开发人员需要掌握的设计知识，在此对重要的信号做一个简单的介绍。系统信号有两个：sys\_clk信号为系统的时钟信号；sys\_rst信号为系统复位信号。读写操作共用信号有三个：app\_rdy信号为用户接口状态信号，当该信号为高时方可进行读写操作；app\_en为读写使能信号；app\_cmd为读写指令信号，值是000时为写操作，值是001时为读操作；app\_addr为读写地址，地址宽度为数据总线宽度。读操作接口有：app\_rd\_data为读出数据，数据宽度与总线宽度相同；app\_rd\_data\_valid为读数据使能信号；app\_rd\_data\_end为读数据结束信号，在最后一个信号时信号为高。写操作接口有：app\_wdf\_rdy为写数据FIFO状态信号，当为高时方可进行写操作；app\_wdf\_data为写入数据，宽度与总线宽度相同；app\_wdf\_wren为写使能；app\_wdf\_end为写入数据结束信号，该信号在写入数据带宽与内存中数据存储带宽不一致时使用，若一致时可一直置为高。

读数据操作流程如图4-2所示。首先是状态信号的确认，app\_en与app\_rdy都需为高才可以进行读取操作。其次配置读取指令信号，app\_cmd需要写入001指示为读操作，app\_addr为读取地址，由于MIS为流水线结构，因此每个时钟可以读取一个地址的数据。在经过不确定的时延之后，读取数据会从app\_rd\_data流出，app\_rd\_data\_valid为读数据使能信号。写数据操作流程如图4-3所示。与读操作相同，状态信号app\_en与app\_rdy需为高时才可进行操作。指令信号app\_cmd为000指示进行的为写操作，app\_addr为写地址，app\_wdf\_data为对应地址的写入数据，adf\_wdf\_en为写使能。

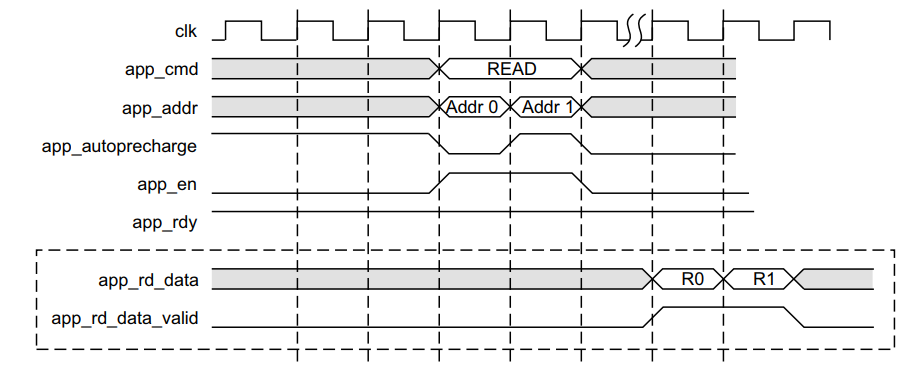


图4-2 读操作信号图

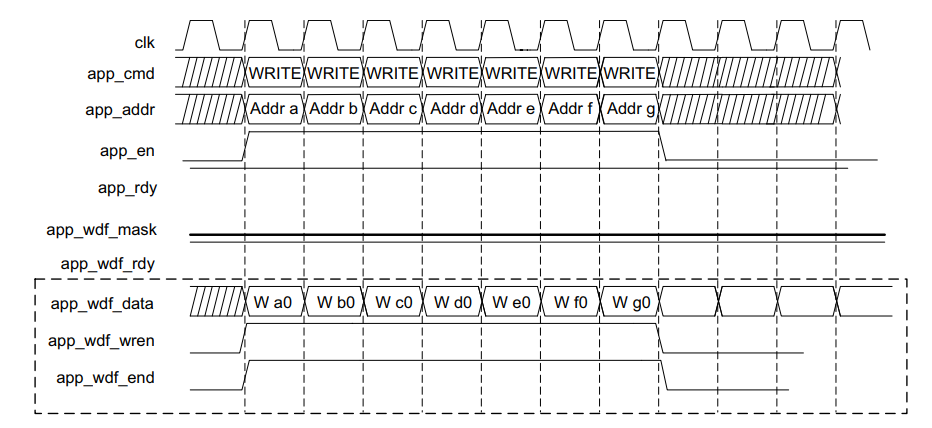


图4-3 写操作信号图

在TTA架构中应用MIS可以用来对内存进行数据的读写，但仍存在一些问题。TTA架构的自定义操作需要指定固定时延，而MIS的读取数据操作时延不定，导致TTA架构无法预先生成确定时钟的指令，此外MIS的控制逻辑相对复杂，若在软件代码中完成控制逻辑，则会相对冗余，也会需要更多的控制逻辑计算。因此，对MIS模块进行封装，在更大颗粒度上进行片外存储单元的设计成为需要。

对于读操作时延不定的问题，这里采用了后接异步FIFO的解决方案。外部读操作通过读取FIFO的状态信号，直接从FIFO中进行数据的读取，可以将读数据的延迟确定为一个时钟。同时，在实际运行中，内存的读取时钟与FPGA的运行时钟并不相同，加入异步FIFO可以解决跨时钟域的问题。

此外，功能单元对读写操作进行了简化，使TTA可以在更大颗粒度上进行数据的读写。对于读操作，输入指令由输入读地址与读使能变为输入读起始地址与读取长度，以减少读取地址的控制；对于写操作，由输入写数据、写使能和写地址变为输入写起始地址与写数据，以减少写地址的控制。读写操作的使能不需要额外进行控制的原因在于，TTA架构为数据触发结构，数据本身含有使能意义。此外，由于全连接层与卷积层的计算特性不同，全连接层需要更大的数据带宽，而卷积层的旋转存储结构需要对数据进行逐个操作，因此设计了两个不同输出宽度的FIFO以供不同计算层用来使用。

由于片外存储功能单元不同操作之间需要状态的互连，因此需要被设计为具有状态信号的功能单元，各操作均可以对状态进行控制。具体状态信号如下：

（1）fifo\_choose：fifo选择状态，选择不同FIFO，值为0或1。

（2）fifo\_empty：fifo状态信号，指示fifo是否为空，值为0或1。

（3）mis\_rd\_addr：读数据地址。

（4）mis\_rd\_depth：读取数据深度

（5）mis\_wr\_addr：写数据地址。

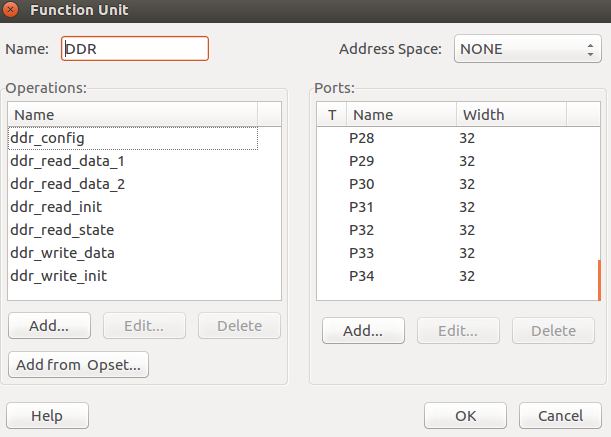


图4-4 片外存储功能单元操作与接口图

片外存储功能单元包含的操作与接口如图4-4所示，其中操作主要有：

（1）DDR\_CONFIG

该操作的主要作用为选择后接FIFO。输入接口数量为1，无输出接口。操作所需延时为1个时钟。当输入为0时，选择后接输出数据宽度为一个数据的FIFO，当输入为1时，选择后接输出数据宽度为个数据的FIFO。

（2）DDR\_READ\_INIT

该操作的主要功能是输入读指令。输入接口数量为2，无输出接口。操作所需延时为1个时钟。接口1为读操作起始地址，以初始化mis\_rd\_addr信号；接口2为读操作长度，以初始化mis\_rd\_depth信号。在初始化之后，功能单元内部将会产生连续的读取地址与读取使能，输入到MIS模块中进行数据的读取。

（3）DDR\_READ\_STATE

该操作的主要功能为读取FIFO是否可读状态信号。输入接口数量为1，输出接口数量为1。操作所需延时为1个时钟。输入接口并无意义，是由于TTA架构的操作需要数据触发而设定的。输出接口为FIFO是否可读取的状态信号，具体计算公式为1-fifo\_empty。

（4）DDR\_READ\_DATA\_1

该操作的主要功能为读取一个数据。输入接口数量为1，输出接口数量为1。操作所需延时为1个时钟。输入接口无意义，触发操作使用。输出接口为所需要读出的1个数据。在使用这个操作之前，需选择输出宽度为一个数据的FIFO，通过DDR\_CONFIG配置。

（5）DDR\_READ\_DATA\_2

该操作的主要功能为同时读取个数据。输入接口为1，输出接口为。操作所需延时为1个时钟。输入接口无意义，触发操作使用。输出接口为所读出的个数据。使用这个操作之前，同样需要通过DDR\_CONFIG配置。

（6）DDR\_WRITE\_INIT

该操作的主要功能为配置写起始地址。输入接口数量为1，输出接口数量为0。操作所需延时为1个时钟。接口1为写数据地址，以配置mis\_wr\_addr状态信号。

（7）DDR\_WRITE\_DATA

该操作的主要功能为写入数据。输入接口数量为1，输出接口数量为0。操作所需延时为1个时钟。接口1为所写入的数据。在写数据进行后，操作会对写地址进行更新。

4.2 输入缓存功能单元设计

输入缓存功能单元的作用为暂存输入计算数据。在本文的设计中，使用输入缓存优化结构以进行片内存储的优化，因此输入缓存中只需要存储幅输入特征图。此外，本节设计了一种用于卷积层与池化层通用的旋转存储结构，用于解决Z型单元在卷积或者池化操作步长大于1时计算效率低下的问题。由于大的卷积窗口可以拆分为多个的子卷积窗口进行运算，因此本节的设计为针对于窗口的卷积运算定制，在第五章时将详细对不同大小的卷积操作实现进行介绍。对于激活层、归一化层以及全连接层，也设计了较为简单的顺序存储结构以进行数据的存储。输入缓存功能单元包含的操作与接口如图4-5所示，操作的具体细节将在下面展开介绍。

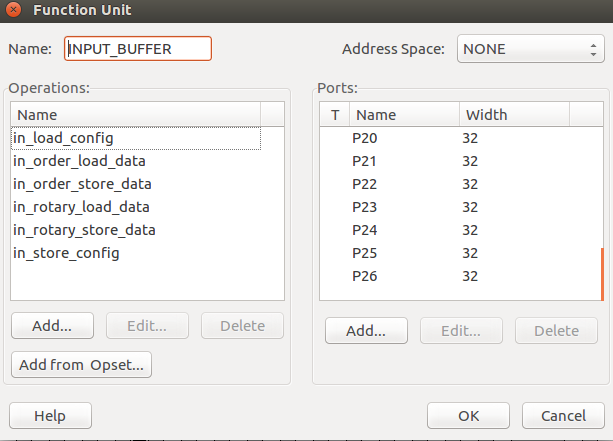


图4-5 输入缓存功能单元操作与接口图

4.2.1 旋转存储结构

在2.2.1节卷积窗口内并行中，对Z型功能单元做了详细的介绍。Z型单元利用卷积窗口间的数据复用性以优化计算数据的读取逻辑以及减少数据的读取带宽。但之后的分析也提到了Z型单元所存在的不足，Z型单元需要消耗额外的片内存储资源，同时在卷积步长大于1时，由于卷积窗口间的数据复用性变低，导致计算效率大幅度下降。因此，Z型单元只适应于卷积步长为1的卷积操作，并不特别适合作为处理器中计算单元的设计。

在输入缓存设计中，由于特征图较大，一般需要多个BRAM来对一副特征图进行存储，Z型单元每个时钟只读取一个数据，片内存储数据带宽没有被充分利用。旋转存储结构旨在充分利用片内存储资源的数据带宽，每个时钟同时读取卷积窗口内的个数据，从而避免卷积步长大于1时的数据等待。实际上，将输入特征图复制9份分别存储到不同的存储单元中即可以完成卷积窗口内9个数据的同时读取，但却会增加8倍的片内存储资源，在片内存储资源较为紧缺的条件下并不可行。因此，旋转存储的设计难点在于如何在不增加存储资源使用的前提下，确保卷积窗口内的9个数字分布在9个不同的存储单元中。旋转存储设计方案如图4-6所示。

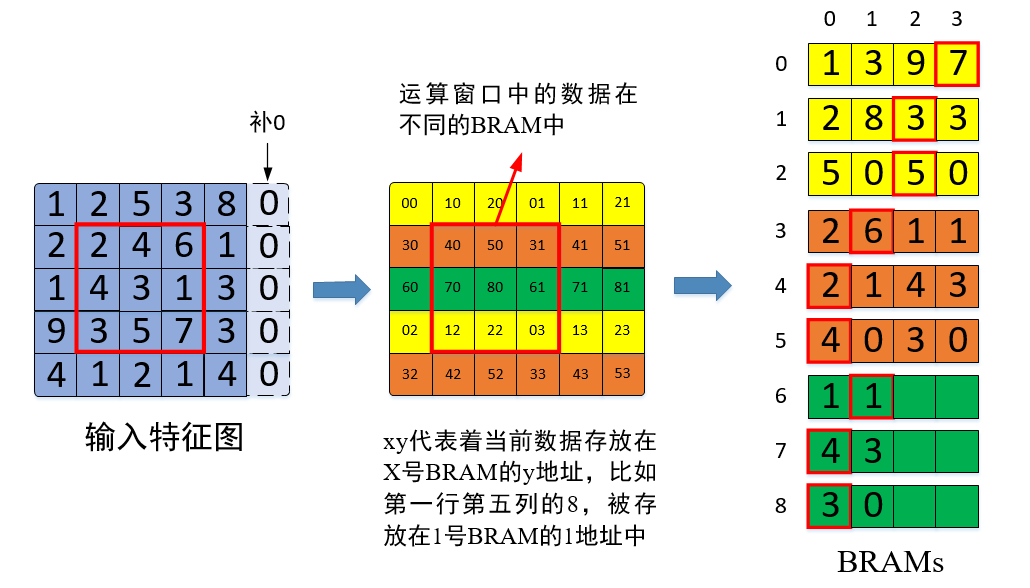


图4-6 旋转存储结构写入数据过程

旋转存储结构的基本思想为：将不同行的数据分别存储在不同的BRAM组中，将同行中的数据依次存储在BRAM组中的3个BRAM内。图中左侧为输入特征图数据，末尾补0使列数为3的整数倍以简化后续的数据读取逻辑，中间为各个数据的存储地址，<x, y>表示将该数据存储在第x个BRAM的y地址，右侧为存储后的BRAM中数据分布。这种存储方式能够确保卷积窗口内的9个数字在不同的BRAM中的逻辑在于，将不同行的数据存储在不同的BRAM组，以保证不同行的数据存储在不同的BRAM中，同行的数据依次存储在3个BRAM中，以可以保证相邻的3个数据总在不同的BRAM中。

数据的坐标<x, y>与其在BRAM中的地址之间的计算由公式4-1与4-2得出，其中row为输入特征图的列数，为计算得到的bram为所存储的BRAM编号，addr为在BRAM中的地址。由于计算公式涉及到除法以及取余操作，而这些操作在FPGA上实现较于复杂，同时流水所需时间较长。但在本公式中，由于除数为固定值3，且被除数的值即坐标值一般在1024之内，因此可以使用查找表的技巧来代替除法以及取余操作，从而简化计算量。公式的计算硬件电路如图4-7所示。

 （4-1）

 （4-2）

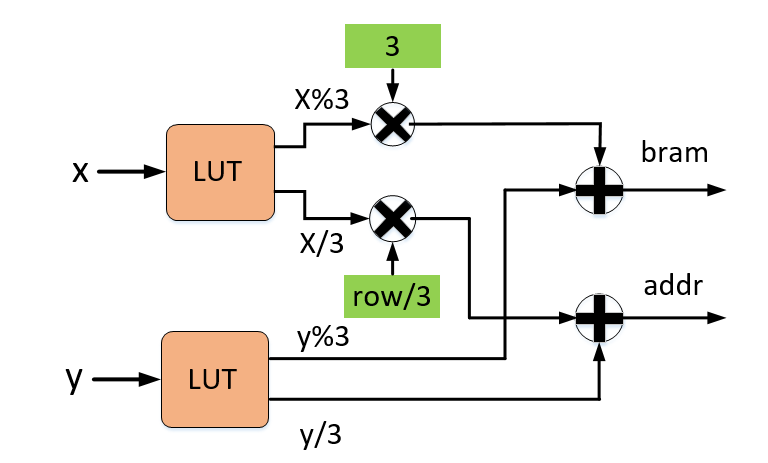


图4-7 坐标与存储位置转换电路图

基于上面的坐标与存储位置之间的计算公式与实现电路，可以完成输入缓存功能单元旋转存储部分的自定义操作。旋转存储部分所需要的操作状态有：

（1）feature\_map\_height：输入特征图高度

（2）feature\_map\_width：输入特征图宽度

（3）rotary\_store\_depth：输入数据为第几幅特征图，供输入特征图并行使用

（4）rotary\_store\_height：输入数据高度坐标

（5）rotary\_store\_width：输入数据宽度坐标

所自定义的操作有：

（1）IN\_STORE\_CONFIG

该操作的主要功能为配置输入特征图参数以及初始化数据坐标。输入接口数量为3，无输出接口。操作所需延时为1个时钟。接口1为输入特征图高度以更新feature\_map\_height，接口2为输入特征图宽度以更新feature\_map\_width，接口3为当前输入特征图深度以更新rotary\_store\_depth。此外，此操作会完成一些操作状态的0初始化，包含rotary\_store\_height、 rotary\_store\_width、order\_store\_ram、order\_store\_addr，后面两个状态为顺序存储的操作状态。

（2）IN\_ROTARY\_STORE\_DATA

该操作的主要功能为旋转存储数据。输入接口数量为1，无输出接口。操作所需延时为4个时钟。接口1为所需要存储的数据。通过状态rotary\_store\_height以及rotary\_store\_width计算存储位置后将数据进行存储，再对状态进行更新。

（3）IN\_ROTARY\_LOAD\_DATA

该操作的主要功能为旋转读取数据。输入接口数量为2，输出接口数量为9\*input\_para。操作所需延时为5个时钟。输入接口1和2为读取窗口左上角数字的坐标，输出接口为输出的9个数字。通过坐标与存储位置计算电路计算出左上角数字的存储位置后，再通过简单映射可以得到其它8个数据的存储位置，完成窗口内9个数字的读取。

4.2.2 顺序存储结构

对于归一化层、全连接层等需要多次加载数据运算且可以顺序进行数据运算的运算层，可以采用较为简单的顺序存储结构。为了节省片内存储资源的使用，顺序存储结构与旋转存储结构共用存储资源，即每个输入特征图使用9个BRAM进行存储。但由于输出特征图并行度一般设置为2的幂次，因此每个输入特征图使用8个BRAM用以存储，以优化后面计算所需的控制逻辑。对于全连接层来说，可以将其看做一层特殊的输入特征图。

顺序存储的存储逻辑如图4-8所示。在所需存储的数据到来时，控制逻辑会将数据按地址写入，即会将数据顺序存储到BRAM的当前地址中，然后对下一地址进行写入。按地址写入的原因是为了后续数据的并行读出，由于使用8个BRAM进行存储，相邻的8个数据可以同时读出进行计算，从而提升计算效率。

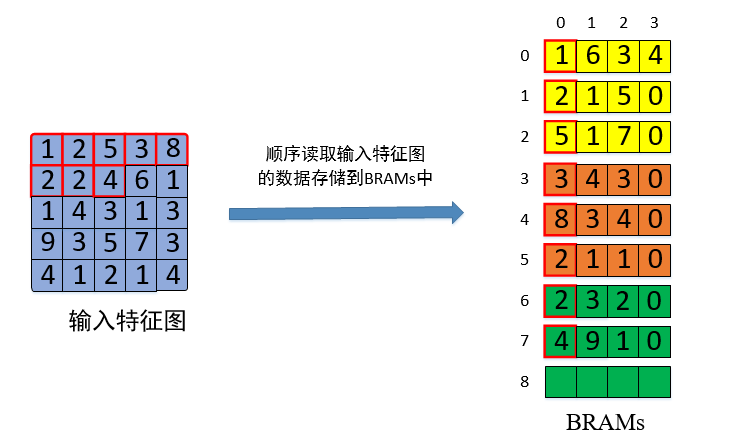


图4-8 顺序存储结构写入数据过程

实现顺序存储结构也需要使用带状态的自定义操作，具体操作状态有：

（1）order\_store\_depth：存储数据的特征图深度

（2）order\_store\_addr：存储数据的ram内地址

（3）order\_load\_addr：读取过程的ram地址

所自定义的主要操作有：

（1）IN\_ORDER\_STORE\_DATA

该操作的主要功能为对数据进行顺序存储。输入接口数量为8，无输出接口数量。操作所需延时为1个时钟。输入接口为所需存储的输入数据。操作首先根据order\_store\_depth即第几幅输入特征图选择所要存储的BRAM组，然后根据order\_store\_addr将数据顺序存储在前8个BRAM相应的地址内。

（2）IN\_LOAD\_CONFIG

该操作的主要功能为对读取参数的初始化。输入接口数量为1，无输出接口数量。操作所需延时为1个时钟。输入接口无意义，触发操作使用。该操作会将order\_load\_addr状态初始化为0，以供重新读取数据。

（3）IN\_ORDER\_LOAD\_DATA

该操作的主要功能为对存储数据进行读取。输入接口数量为1，输出接口数量为。操作所需延时为1个时钟。输入接口无意义，触发操作使用，输出接口为所读出的数据。该操作会将所有存储特征图的BRAM地址为order\_load\_addr的数据进行读出，因此输出接口数量为。由于各层所需数据均为顺序读出即可，地址的控制逻辑放在了操作内部，以减少操作外部的逻辑控制。

4.3 通用计算功能单元设计

通用计算功能单元负责对输入缓存或片外存储的输出数据进行运算，所有各运算层所需的计算。由于处理器对神经网络的加速是层间串行的，因此不同层并不会同时加速，将所有运算层的自定义操作放在同一功能单元中不会出现等待计算的状况。此外，将所有运算层操作集中在一个功能单元中，可以有效的减少架构中的输入输出接口数量，减少互联网络的复杂性。同时，由于在同一功能单元内，全连接层与卷积层可以共用乘法器阵列，从而减少乘法器的使用。通用计算功能单元包含的操作与接口如图4-9所示，操作的具体细节将在下面展开介绍。

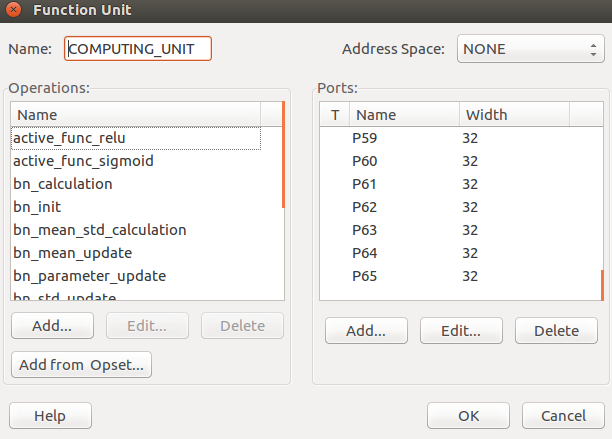


图4-9 通用计算功能单元操作与接口图

4.3.1 卷积运算操作设计

在4.2.1节中介绍了应用于卷积操作以及池化操作的旋转存储结构，使得输入缓存可以同时输出卷积窗口内的9个数字，也就是说，数字传输的控制逻辑已经在输入缓存中完成，卷积运算操作部分只需要进行参数的更新以及运算即可。参数部分的更新可以通过在操作中加入9个寄存器以寄存参数，在参数到来时，对寄存器进行更新即可。运算部分可继续延续Z型单元的运算单元，即使用乘法阵列以及加法树结构，在同一时钟内即可完成9次乘法及8次加法运算，通过流水线结构，一个时钟即可输出运算结果。卷积运算操作所完成的运算流程图如图4-10所示。

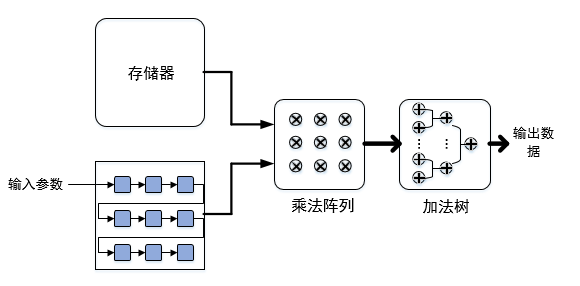


图4-10 卷积运算单元电路图

由于涉及到输入参数的更新，因此卷积运算操作仍需设计为带状态的操作。所设计的状态为：

（1）weight：卷积核参数数组，数组深度为

所设计的操作有：

（1）CONV\_PARAMETER\_UPDATE

该操作的主要功能为对计算单元内的参数进行更新。输入接口数量为，输出接口数量为0。操作所需延时为1个时钟。输入接口为所需要更新的卷积核参数，因为输入接口数量为，则每次即可完成一个卷积核内参数的更新。

（2）CONV\_CALCULATION

该操作的主要功能为完成卷积操作的乘加运算。输入接口数量为，输出接口数量为。操作所需延时为5个时钟。输入接口内的数据是从输入缓存中输出的运算窗口内的个数字，输出接口则为层输出特征图的中间结果。输入缓存输出的输入特征图卷积窗口内数据分别与个卷积核进行运算，各输入特征图运算完成的结果通过加法树进行相加，得到层输出特征图中间结果。

4.3.2 池化运算操作设计

池化运算与卷积运算结构较为相似，有着相同的数据读取逻辑，不同之处在于计算单元。与卷积运算相同，池化运算的数据读取逻辑通过输入缓存的旋转存储结构来完成，池化运算操作设计只需完成计算功能即可。此外，池化运算种类较多，但最常用的为最大池化与平均池化，具体运算过程已在2.1节中说明。若用户需要添加其它池化运算，可以通过增加自定义操作进行实现。

池化运算仅涉及到输入池化窗口内部数据间的运算，因此不需要设计带有状态的操作。具体操作设计如下：

（1）POOL\_MAX\_CALCULATION

该操作的主要功能为完成最大池化。输入接口数量为，输出接口数量为。操作所需延时为4个时钟。输入接口为层输入特征图的池化窗口内数据，输出接口是层的输出特征图中间结果。操作对输入的9个数字进行比较，取出最大值进行输出，通过层输入特征图并行运算，可以一个时钟得到个池化结果。

（2）POOL\_MEAN\_CALCULATION

该操作的主要功能为完成平均池化。输入接口数量为，输出接口数量为。操作所需延时为4个时钟。输入接口为层输入特征图的池化窗口内数据，输出接口是层的输出特征图中间结果。操作对输入的9个数字进行累加，通过查找表完成除法操作后得到平均结果，通过层输入特征图并行运算，可以一个时钟得到个池化结果。

4.3.3 激活函数操作设计

激活函数层的运算逻辑较为简单，一般的激活函数都是函数映射方式，即一个输入对应于一个输出，不涉及多个数据的组合运算。近年来，学者们提出了许多优异的激活函数，如基础的非线性函数tanh、sigmoid和relu等，之后又在这些函数基础上提出了许多诸如prelu、leaky relu等。尽管函数变化较多，但在FPGA上的实现都较为相似，对于relu等规则较为简单的函数，可以使用一般判断逻辑来完成，而对于sigmoid等形式较为复杂的函数，一般采用查找表的技巧来实现，以空间换资源的形式，避免太过于复杂的运算。使用查找表实现sin函数的一个简单例子如图4-11所示，其思路一般是将输入作为地址，表内地址所对应的数据即为输出，因此无论函数形式如何，使用查找表的技巧进行激活函数的实现都可以较为便捷。

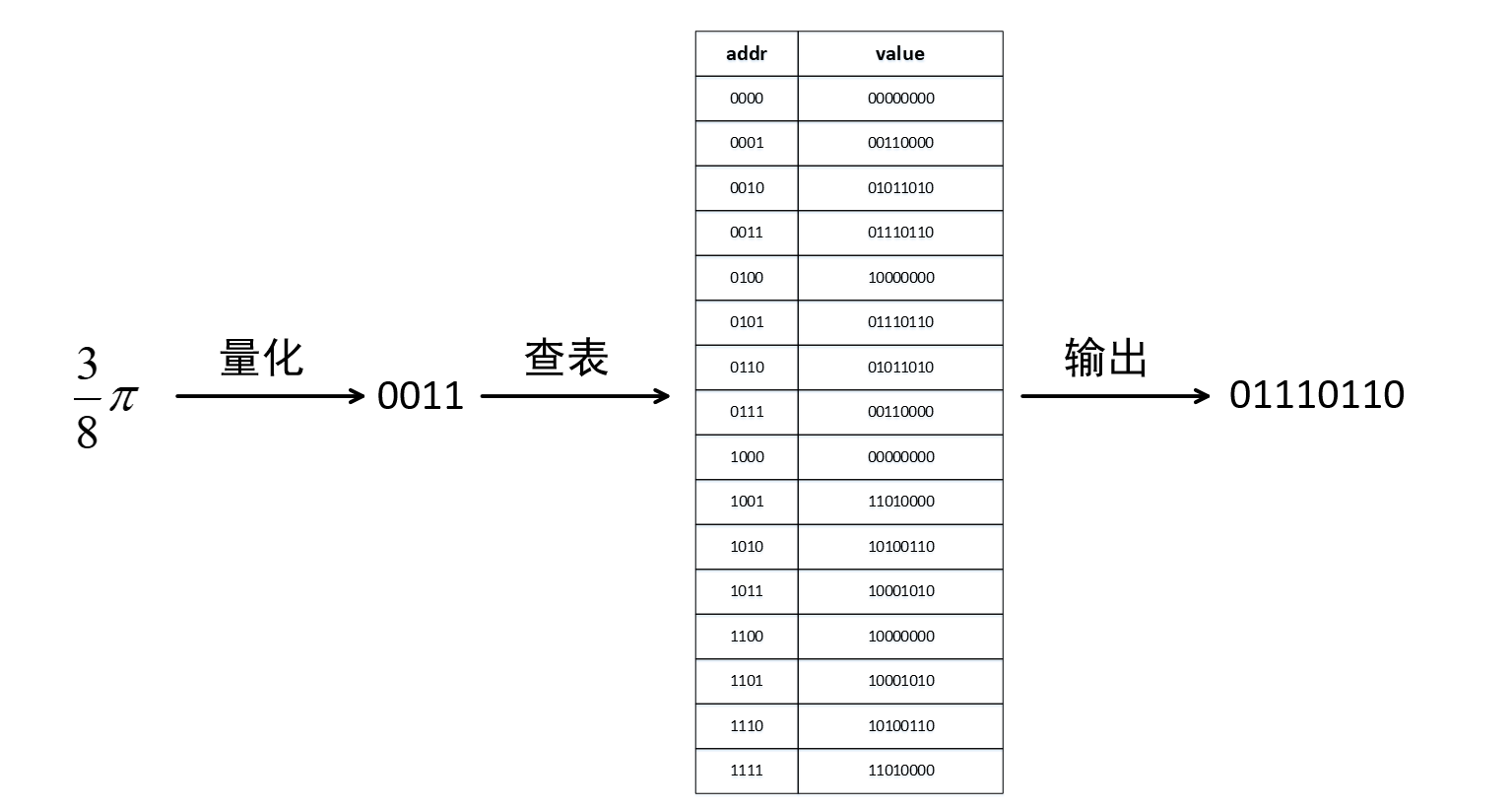


图4-11 sin函数查找表实现图

本文对最常用的激活函数sigmoid以及ReLU函数进行了实现，其中ReLU使用判断逻辑实现，sigmoid使用查找表的方式进行实现。所设计的操作主要有：

（1）ACTIVE\_FUNC\_RELU

该操作的主要功能为对输入的数据使用ReLU函数激活。输入接口数量为，输出接口数量为。输入数据为待激活数据，输出数据为已激活函数。与全连接层参数较为相似，由于输入数据只使用一次，因此从片外存储中直接读取比较高效，这也是激活操作并行度设置为的原因所在。

（2）ACTIVE\_FUNC\_SIGMOID

该操作的主要功能为对输入的数据使用sigmoid函数。输入接口数量为，输出接口数量为。输入数据为待激活数据，输出数据为已激活函数。

4.3.4 批量归一化操作设计

在2.1节中对批量归一化层做了较详细的介绍，批量归一化运算时输入特征图层内数据的运算，不涉及到层间的运算，因此在输入缓存优化结构的条件下可以顺利完成运算。根据批量归一化的运算公式，其共需读取三次数据，第一次读取数据用来计算输入特征图的均值，第二次读取数据用来计算输入特征图的方差，第三次读取数据用来计算归一化值并乘上缩放值，加上平移值得到最终的结果。

批量归一化层需要操作状态以存储及更新参数，因此需要被设计为带状态的操作，所设计的操作状态有：

（1）bn\_data\_size：单层输入特征图内数据量

（1）bn\_shift：缩放值参数，共个

（2）bn\_bias：平移值参数，共个

（3）bn\_mean：计算得到的均值，共个

（4）bn\_std：计算得到的标准差，共个

所设计的操作有：

（1）BN\_INIT

该操作的主要功能为批量归一化层参数的初始化。输入接口数量为1，无输出接口。操作所需延时为1个时钟。输入接口无意义，触发操作使用。操作会将bn\_mean以及bn\_std初始化为0，以进行后续运算。

（2）BN\_PARAMETER\_UPDATE

该操作的主要功能为对批量归一化层的参数进行更新。输入接口数量为，无输出接口数量。操作所需延时为1个时钟。接口1为该层输入特征图的缩放值，接口2为平移值，依次向后类推。

（3）BN\_MEAN\_UPDATE

该操作的主要功能是更新bn\_mean参数。输入接口数量为，无输出接口。操作所需延时为3个时钟。输入数据为层输入特征图数据。数据输入后，操作会对数据进行累加，而不马上进行均值计算。

（4）BN\_STD\_UPDATE

该操作的主要功能是更新bn\_std参数。输入接口数量为，无输出接口。操作所需延时为40个时钟。输入数据为层输入特征图数据。数据输入后，操作会计算数据与均值的误差平方和，而不马上进行标准差计算。

（5）BN\_MEAN\_STD\_CALCULATION

该操作的主要功能为计算bn\_mean及bn\_std。输入接口数量为1，无输出接口。操作所需延时为33个时钟。输入接口无意义，触发操作使用。操作会计算bn\_mean以及bn\_std，在单次批量归一化运算中需要调用两次操作才可以完成bn\_mean以及bn\_std两个参数的计算。

（6）BN\_CALCULATION

该操作的主要功能为计算输入数据的归一化值。输入接口数量为，输出接口数量为。操作所需延时为35个时钟。操作所需延时为1个时钟。输入接口为层输入特征图顺序读出的数据。输入数据通过计算得到的bn\_mean及bn\_std进行归一化，再乘以bn\_shift参数加上bn\_bias参数得到最终结果。

4.3.5 全连接层操作设计

全连接层是卷积神经网络中较为特殊的运算层，与其它运算层不同，全连接层的参数量非常庞大，因此，其加速并不受限于乘法器资源而是片外存储带宽。一般来说，将全连接层的输入数据看做为一维向量，参数看做为二维矩阵，将输入数据向量与参数矩阵进行矩阵乘法，则可以得到输出的一维向量。在本处理器的设计中，全连接层的数据加载使用输入缓存的顺序存储模式，参数则通过片外存储功能单元直接读取，根据片外存储的数据带宽来设定全连接层加速倍数，在全连接层操作中完成运算。

全连接层的运算共有两层循环，根据循环的顺序不同，加速方案也有所不同。为了减少数据的读取，处理器设计采用了数据循环在外，参数循环在内的展开顺序。具体的运算过程为：从输入缓存中加载一个数据，从片外存储中加载与该数据对应的参数向量，完成数据与参数之间的计算后，加载新的数据。全连接层所设计的操作为：

（1）FC\_CALCULATION

该操作的主要功能为完成全连接层运算。输入接口数量为，输出接口数量为。操作所需延时为1个时钟。输入接口1为加载数据，其余为参数，输出接口为运算结果。操作将加载数据与参数进行相乘后，将乘法结果直接输出到输出接口，后续的累加过程在输出缓存功能单元中完成。

4.4 输出缓存功能单元设计

在卷积层以及全连接层的运算过程中，会产生大量的中间结果，若将中间结果直接传输至片外存储，则需要大量的数据带宽对中间结果进行回读以完成运算，因此需要设计额外的缓存功能单元以进行中间结果的存储。此外，若在其它的功能单元中进行中间结果的更新计算，则会增加互联网络的数据传输负荷，使加速方案受限于互联网络的数据传输能力。因此，新的功能单元需要具备两种功能，分别为中间结果的存储及中间结果的更新，这是输出缓存功能单元的设计出发点所在。

对于卷积运算而言，其运算类型为乘加运算，中间结果的数据更新方式主要有两种。第一种是初始化更新，即第一层输入特征图的计算中间结果需要将存储器中原始数据进行覆盖，以免受到上一层运算结果的影响；第二种为累加更新，在当前层输入特征图计算完毕后，需要与前面层的中间结果进行累加。而对于池化操作而言，最大池化所需的更新方式为初始化更新以及最大化更新，平均池化所需为初始化更新及累加更新，平均计算在池化操作中完成。激活函数层与批量归一化层不需要数据更新。全连接层则需要初始化更新及累加更新。综合所有层的计算方式来看，输出缓存功能单元需要设计初始化更新、累加更新以及最大化更新三种更新方式。

此外，输出缓存功能单元的接口数量也需要进一步的确认，以防止由于接口数量不足导致的计算等待。卷积操作的输出接口数量为，池化操作的输出接口数量为，激活函数操作的输出接口数量为，批量归一化层的输出接口数量为，全连接层的输出接口数量为。因此输出缓存功能单元的接口数量如公式4-3所示。

 （4-3）

为了进一步简化卷积层与池化层中输出缓存功能单元的使用，输出缓存功能单元将地址控制逻辑，操作留有接口为缓存数据坐标，其中为缓存数据在存储器中的深度，为缓存数据所存储的起始存储器下标，与为缓存数据在输出特征图中的坐标。具体地址的计算公式如公式4-4所示。对于全连接层以及激活函数层等操作，其数据可以看做一维向量，此时只需将、、设为0，则设置为所存储的地址即可。

 （4-4）

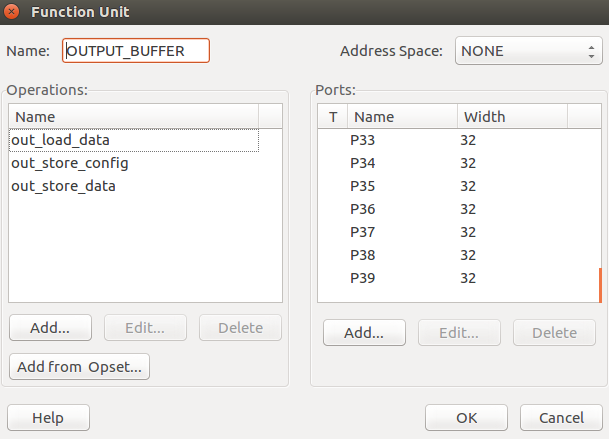


图4-12 输出缓存功能单元操作与接口图

输出缓存功能单元包含的操作与接口如图4-12所示，由于需要使用fm\_width，fm\_height两个状态来进行地址的计算，因此其被设计为具有状态的操作，状态有：

（1）fm\_width：缓存输出特征图宽度，若输入为一维向量则设置为0

（2）fm\_height：缓存输出特征图高度，若输入为一维向量则设置为0

功能单元所包含的操作有：

（1）OUT\_STORE\_CONFIG

该操作的主要功能为初始化输出特征图参数。输入接口数量为2，无输出接口。操作所需延时为1个时钟。输入接口分别为缓存输出特征图宽度以及输出特征图高度，操作根据传入值对fm\_width及fm\_height参数进行更新。

（2）OUT\_STORE\_DATA

该操作的主要功能为对输入数据进行缓存。输入接口数量为，无输出接口。操作所需延时为5个时钟。前输入接口为缓存数据，接下来4个接口为缓存数据地址。剩下的接口中，1个接口为存储模式，值为0时存储前数据，值为1时存储前数据，值为2时存储前数据。最后的接口为数据更新模式store\_mode，值为0时为初始化更新，值为1时为累加更新，值为2时为最大化更新。

（3）OUT\_LOAD\_DATA

该操作的主要功能为读出数据。输入接口数量为2，输出接口数量为1。操作所需延时为1个时钟。输入接口分别为数据存储的以及在中的地址，输出接口为所读出数据。

4.5 互联网络设计

互联网络的设计是设计TTA处理器中不可忽视的一环。在TTA处理器中，互联网络主要包括总线以及socket节点两个部分，总线负责数据的传输，而socket节点则是总线与功能单元间的接口。卷积神经网络计算量庞大，伴随而来的则是大量的数据传输，因此若互联网络设计不当，对处理器的加速效果影响颇大。互联网络设计不当主要体现在两个方面：其一是互联网络太过于简单，总线数量过少或者socket节点过少，导致数据传输的吞吐量较低，使通用计算功能单元需要进行数据传输的等待方可进行运算；其二是互联网络太过于复杂，总线数量过多或者无用socket节点过多，在导致资源浪费的同时，也会降低TTA处理器的最大运行时钟，导致加速效果降低。

在TTA处理器中，最常见的互联网络设计为全连接型设计。全连接型互联网络指的是总线与所有功能单元的所有输入输出接口之间都存在socket节点的一种互联网络，结构如图4-13所示。假设总线数量为，所有功能单元的输入输出接口数量为，则所需的socket节点数量则为。在一些较小型的处理器中，全连接型互联网络设计简单，同时耗费资源在容忍范围之内，是一种较为常用的互联网络结构。但针对于卷积神经网络的TTA处理器设计，由于其总线数量及功能单元输入输出接口数量过多，使用全连接型互联网络则过于冗余，对处理器的性能也会造成负面影响。

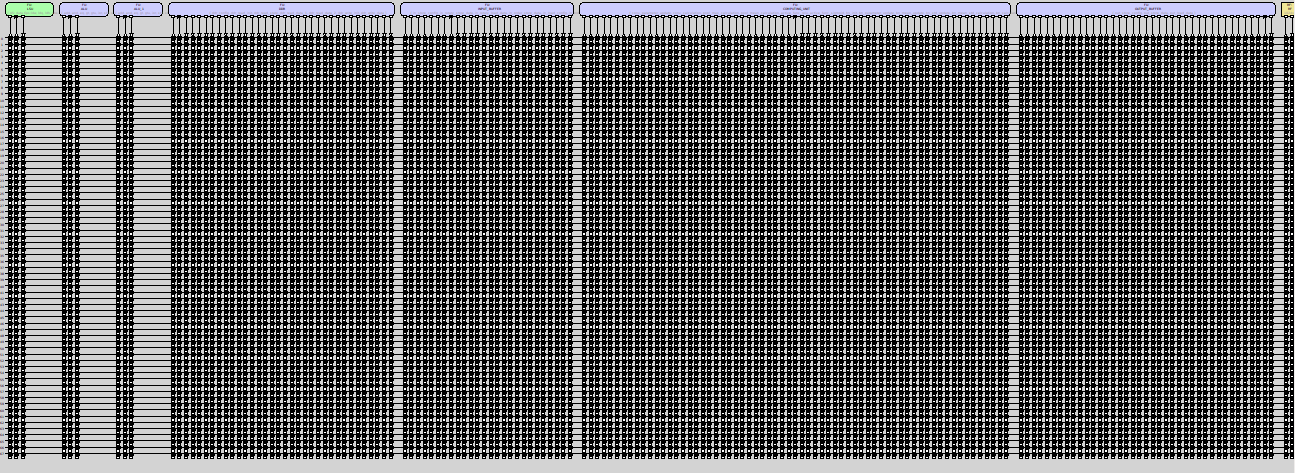


图4-13全连接型互联网络

互联网络的设计，首先从确定总线数量开始。对于输入特征图并行度为、输出特征图并行度为、全连接层并行度为的加速方案来说，卷积操作运算单个时钟所需要的输入数据量为，输出数据量为，即需要的最少总线数量为：

 （4-5）

池化操作、激活函数操作、批量归一化操作、全连接层操作所需的最少总线数量分别为：

 （4-6）

 （4-7）

 （4-8）

 （4-9）

由于不同运算层的计算是串行结构，不会同时计算，因此，总线满足其中最大需求的运算层即可。在确定加速方案后，即可以确定总线数量。

此外，socket节点的确定是互联网络设计的另一部分。在功能单元的设计中，输入缓存的输入数据来源于片外存储功能单元，通用计算功能单元的数据一部分来源于输入缓存功能单元，一部分来源于片外存储功能单元，输出缓存功能单元的输入则来源于通用计算功能单元，输出则通向片外存储功能单元。在了解数据流向之后，则可以进行socket节点的定制，在数据需要互通的功能单元的接口间设定socket节点，即可完成我们所有的需求。此外，由于有些功能单元还需要一些逻辑信号的控制，如输出缓存单元的地址及状态控制信号，需要额外设定一些状态控制总线，这些总线使用全连接型以方便数据运输，具体数量则根据各功能单元所需的状态计算即可。图4-14为在输入特征图并行度为2、输出特征图并行度为32、全连接层并行度为32的加速方案下的互联网络设计。

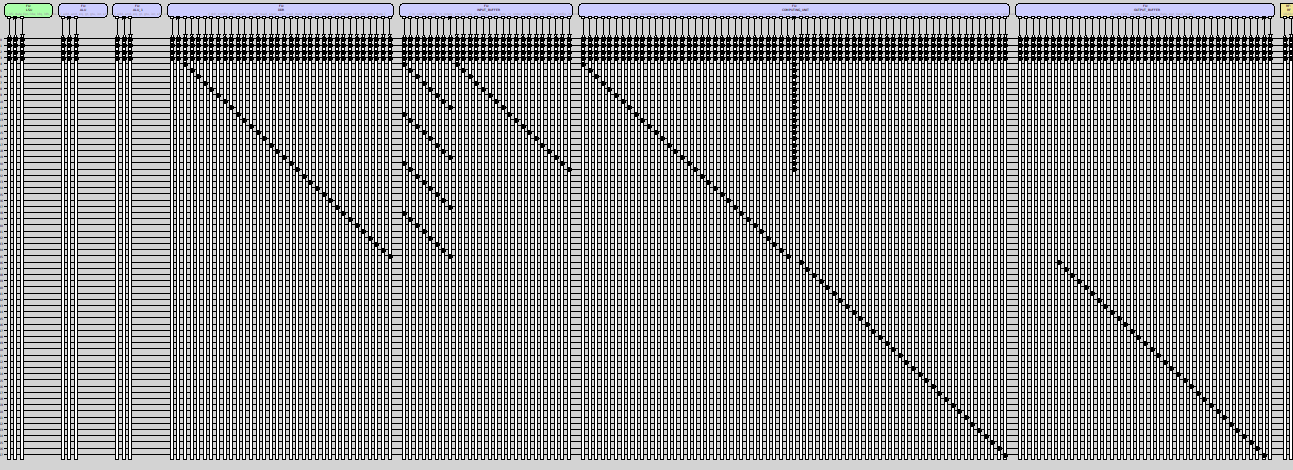


图4-14 新型互联网络设计图

相比全连接型互联网络所需的socket节点数量，图4-10所示的互联网络中socket节点为全连接型互联网络socket节点的7%。且随着加速方案的提升，互联网络socket节点数量与功能单元输入输出接口数量之间为线性增长关系，相比于全连接型互联网络的平方增长，大幅度降低了互联网络的复杂度。

4.6 本章小结

本章主要的主要内容包括TTA架构中功能单元及互联网络的设计，其中功能单元主要有片外存储功能单元、输入缓存功能单元、通用计算功能单元以及输出缓存功能单元。片外存储单元对Xilinx公司的内存接口模块进行封装，在更大颗粒度的角度上对片外存储的读写操作进行了优化。输入缓存功能单元针对卷积层及池化层的数据加载逻辑，设计了旋转存储结构以解决传统Z型卷积单元片内存储资源消耗以及在卷积操作步长大于1时计算资源浪费的问题。通用计算功能单元则包括所有运算层的实现，将所有运算层的操作放在同一功能单元内可以有效减少互联网络复杂度，同时不同运算层可以共用计算资源以减少计算资源的使用。输出缓存功能单元则主要负责中间结果的存储及更新，其中包含初始化、累加以及最大化大小三种更新操作以满足不同层的运算需求。最终对处理器的互联网络进行了设计，根据数据的流向设计了状态控制以及数据传输两种总线，分析了数据传输总线所需数量及socket节点的确定方案。相比全连接型互联网络，使互联网络的复杂度大幅降低。

第五章 基于TTA架构的卷积神经网络实现及性能分析

在第四章中对TTA处理器的功能单元设计以及互联网络设计进行了介绍，但对如何通过所设计的功能单元对任意卷积神经网络进行加速并没有详细的说明。本章将对TTA处理器的灵活性进行分析，讨论如何通过确定的功能单元对不同参数不同规模的运算层进行加速。在保证处理器灵活性的前提下，尽量降低计算效率的损失。此外，将通过对一种常见卷积神经网络的加速，对所设计的处理器的资源使用与性能进行详细分析。

5.1 自定义运算层实现

自定义运算层的实现主要包括卷积层、池化层、激活函数层、批量归一化层以及全连接层的实现。为了优化软件代码结构，在这些运算层的实现之外添加了配置文件以配置预留参数，同时添加了主函数文件以对各运算层的实现进行调用。具体文件结构如图5-1所示。下面将对各运算层的实现进行详细介绍。

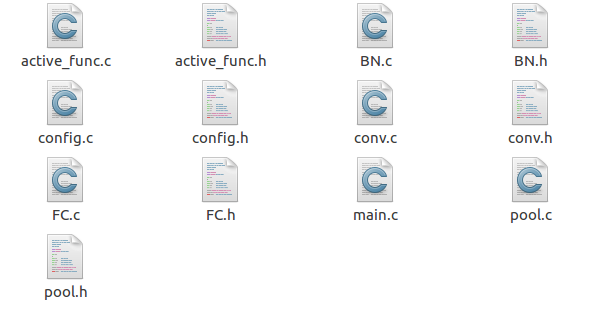


图5-1 运算层实现文件结构

5.1.1 卷积层实现

卷积层是卷积神经网络中计算逻辑最为复杂的一层，这是由于其所涉及的运算有6层循环逻辑，虽然已在旋转存储结构以及在缓存优化结构中对运算进行了优化，但通用卷积层的实现仍有着一些问题没有解决。首先，处理器所设计的旋转存储结构以及卷积运算操作都是针对于的卷积核制定的，但实际上卷积操作的卷积核大小不定，如何通过的卷积运算操作来完成卷积核大小为的卷积运算，是一个需要讨论的问题。其次，卷积运算的偏置累加计算运算量较小，在处理器中为其设计单独的运算操作显的较为冗余，且在软件代码中需要额外的逻辑控制，如何通过已有的运算操作来完成偏置的累加，也是需要思考的一点。

对于卷积核大小为的卷积操作，其可以使用拆分的方式通过的卷积操作完成，具体的拆分过程如图5-2所示。由于卷积窗口内部的运算是通过累加的形式生成输出特征图的中间结果，因此可以通过拆分后的卷积运算结果累加来生成卷积运算结果。在输出缓存功能单元的设计中，我们设计了初始化、累加、最大化三种数据更新方式，这里可以使用累加的数据更新方式来进行结果的更新。

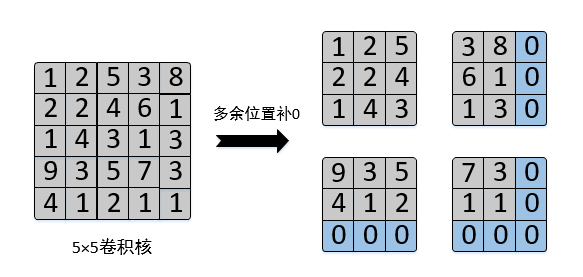


图5-2 卷积核参数拆分图

对于偏置的累加计算，也可以通过将其看做特殊的卷积层来进行运算，其中的转换如图5-3所示。偏置的累加是作用于卷积核参数对应层输的输出特征图中间结果之上的，因此其计算结果可以看做单层输出特征图的中间结果，与单层卷积核的计算结果并无不同，转换原理也基于此结论。具体的转换过程为，将偏置附近补0，使其与单层卷积核参数规模相同，将其附加到卷积核参数的最后一层之后。在实际计算过程中，如果为偏置层，那么将输入数据全部置为1，使得计算结果全部为偏置，对中间结果进行累加更新，得到当前输出特征图的最终运算结果。

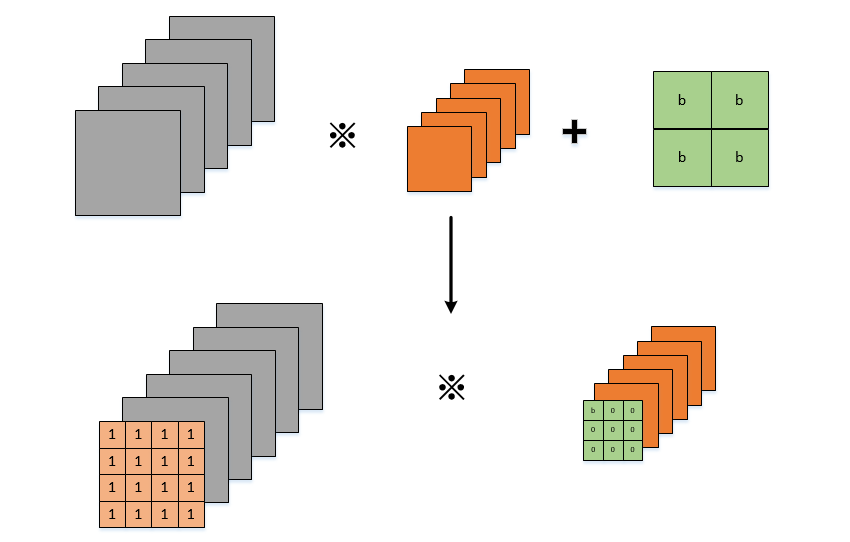


图5-3 偏置计算转换卷积计算图

在解决了任意规模卷积运算以及偏置计算的问题后，此时可以将卷积运算封装为函数以供上层调用。函数内部逻辑伪代码如图5-4所示。

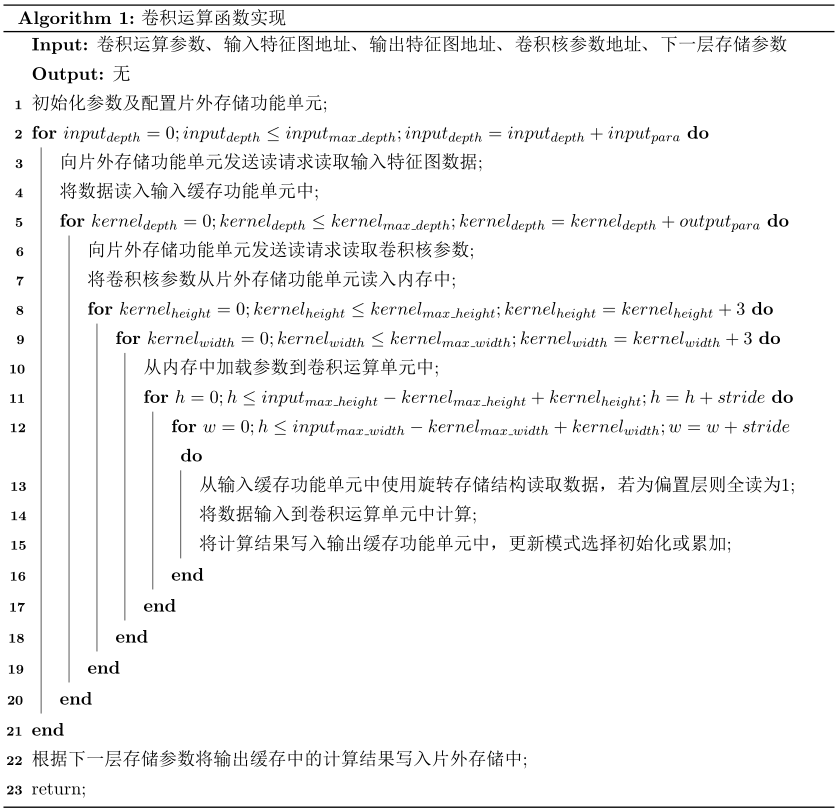


图5-4 卷积层实现伪代码

在指定参数下，卷积层所可以达到的计算并行度为：

 （5-1）

5.1.2 池化层实现

与卷积运算相似，池化运算也发生在数据窗口内，其数据的加载也需要使用输入缓存的旋转存储结构。但与卷积运算相比，池化运算的逻辑更加简洁。卷积运算在计算单层输出特征图时，需要对所有层输入特征图进行计算，池化层则为单层到单层的计算形式，即单层的输出特征图只需要单层的输入特征图参与计算。因此其计算量与计算逻辑相比卷积层都要简化很多。在池化运算中，池化窗口大小也是可制定参数，由于旋转存储结构针对输入为尺寸的数据窗口运算，因此，在实现池化层时如何用的池化运算组成的池化运算，也需要进行设计。

与卷积运算相似，的池化运算也可以进行拆分，具体拆分过程如图5-5所示。但由于所设计的卷积操作有最大化池化以及平均池化两种，因此，在输出缓存单元的使用上，需要根据具体的池化操作来选择数据更新方式。

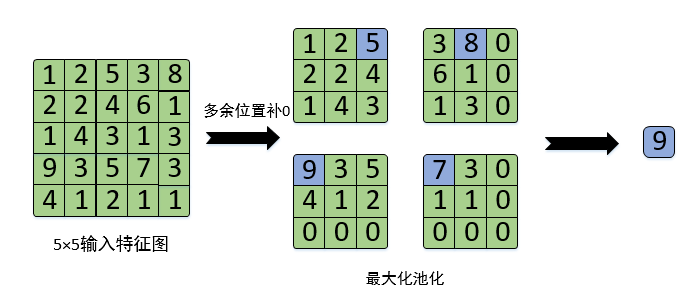


图5-5 最大化池化运算拆分

池化函数实现的伪代码如图5-6所示。

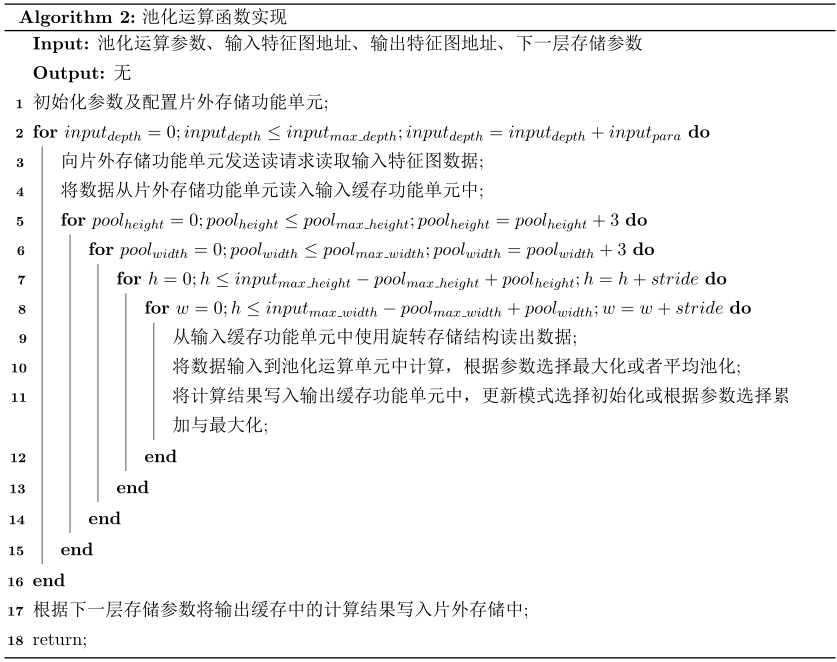


图5-6 池化层实现伪代码

在指定参数下，池化层的计算并行度可以达到：

 （5-2）

5.1.3 激活函数层实现

激活函数层逻辑最为简单，其计算逻辑为单值到单值的形式，即输出特征图中的单个数据仅对应输入特征图的单个数据。与全连接层类似，此时可以将输入数据看做一维向量，简化数据加载逻辑。另外，由于激活函数层的输入数据只使用一次，因此其并不需要额外进入到输入缓存单元中进行缓存。将数据从片外存储功能单元中进行读取，输入到计算模块，完成计算后，直接输出到输出缓存功能单元中即可。激活函数层函数实现的伪代码如图5-7所示。

在指定参数下，激活函数层的计算并行度可以达到：

 （5-3）

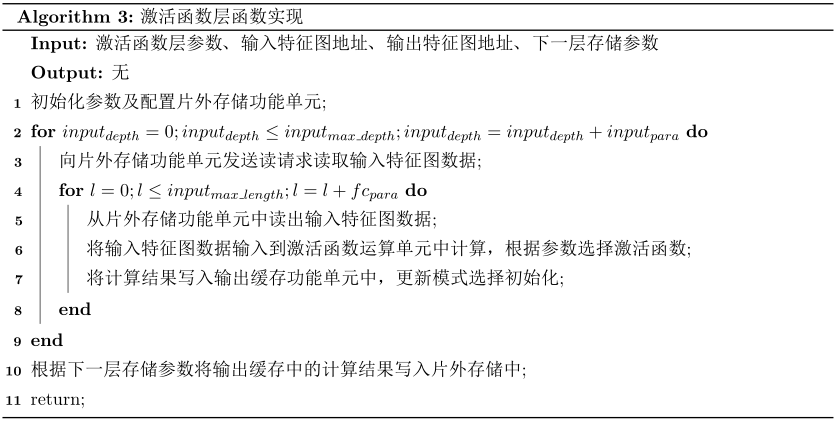


图5-7 激活函数层实现伪代码

5.1.4 批量归一化层实现

批量归一化层的作用主要是对上层输出数据进行归一化，其计算形式为单层到单层，即单层输出特征图仅需单层输入特诊图参与计算。其计算主要分为三个阶段，第一个阶段为计算当前层数据的均值，第二个阶段是计算当前层数据的标准差，第三个阶段是对当前层数据进行归一化及变换重构。由于其需要对输入数据多次读取，因此输入数据首先将进入到输入缓存单元中进行暂存。批量归一化层计算逻辑并不复杂，可以看做为对输入特征图层进行遍历，所需的计算逻辑主要为层内的数据计算。具体函数实现的伪代码如图5-8所示。

值得一提的是，尽管在计算均值与方差时，其加速速度可以达到在指定参数下，但在归一化及数据重构时，由于需要匹配输出缓存功能单元的接口数量，这里采用将归一化及数据重构计算的并行度设计为，即可使批量归一化操作的数据输出带宽与池化操作一致。批量归一化的计算并行度为：

 （5-4）

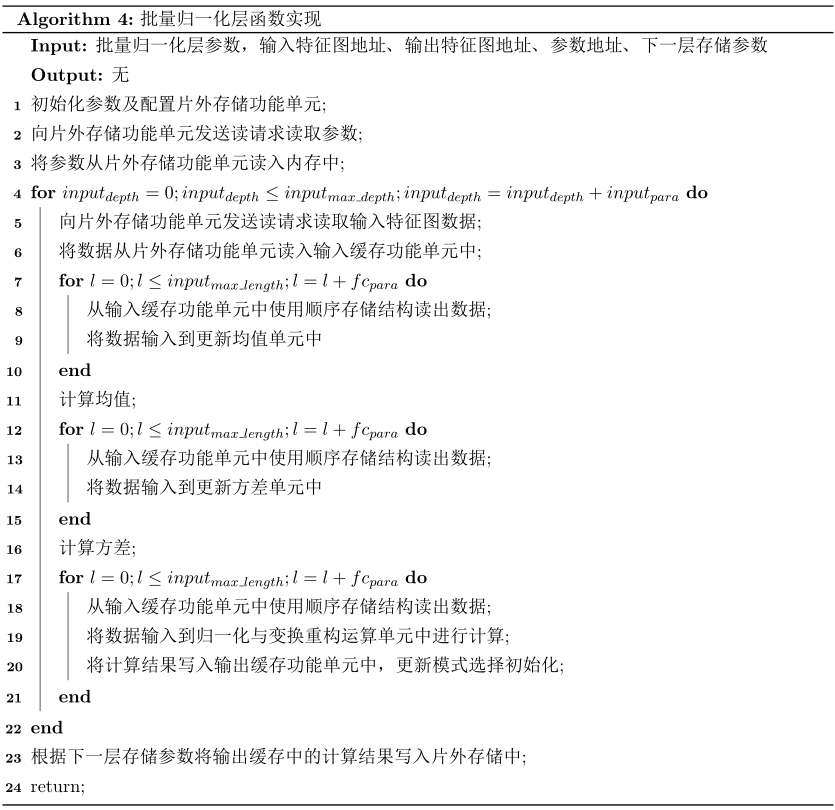


图5-8 批量归一化层实现伪代码

5.1.5 全连接层实现

全连接层的运算逻辑为输入数据与一组参数共同参与乘加运算，得到单个输出值，因此输出值的数量与参数的组数相同。但在TTA处理器的操作设计中，则将全连接层的运算看做单个输入值与所有组对应位置的参数进行运算，得到输出的中间结果，对所有输入值进行遍历得到最终结果。采用这种计算形式的原因是为了减少数据的读取带宽，在第一种计算方式中，单个时钟需要读取输入数据，而输入缓存单元的输出接口最大为，在较小 时，无法满足运算所需的数据带宽。全连接层是卷积神经网络中参数量最大的一层，单层的参数往往可达到千万级别，其计算性能的瓶颈并不是计算资源量，而是片外访存带宽，因此设计参数以供使用者根据带宽自行设定。同时由于全连接层参数只使用一次，因此其参数可从片外存储单元中直接读取，输入到计算单元中进行计算。

此外，全连接层的偏置计算也进行了通用设计。在全连接层的实现中，将偏置看做参数中的一部分，即将每组参数的偏置放在参数的最后面。与偏置进行计算时，将数据置为1即可得到偏置值，再通过输出缓存的累加更新方式，将偏置值加入到最终结果中。

全连接层的实现伪代码如图5-9所示。

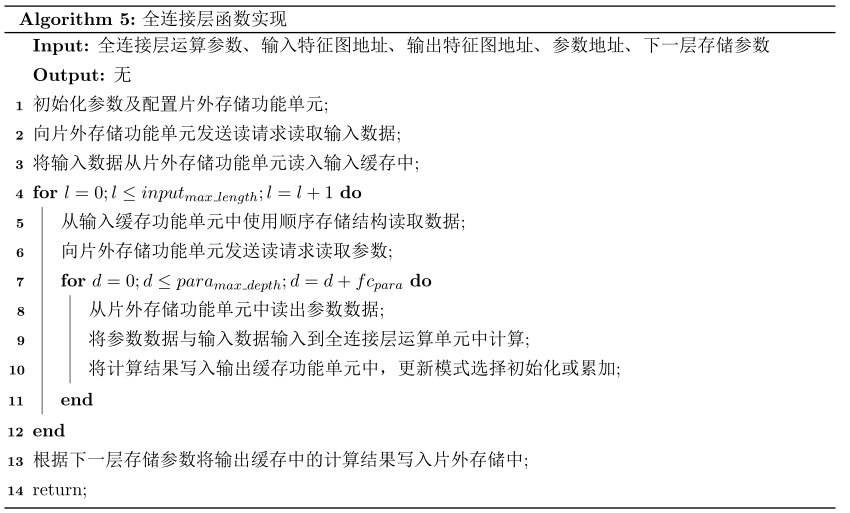


图5-9 全连接层实现伪代码

在指定参数下，全连接层的计算并行度为：

 （5-5）

5.2 灵活性与性能分析

在本文的第四章中，提到了加速方案由、与三个参数确定，但如何确定这三个参数却没有详细说明。本节将从加速方案的确定开始，对AlexNet网络进行加速。

在FPGA平台的资源中，制约加速参数的主要有两种资源：乘法器以及片外存储带宽。卷积运算为运算密集型操作，加速效率受乘法器个数限制，而全连接层运算为访存密集型运算，受片外存储带宽影响较大。与参数均为卷积层的加速参数，应依照FPGA内乘法器个数制定。实际上，保留两个参数是为了有更大的灵活性，以免在加速倍数较高时计算效率降低。比如，在输出特征图层数只有64的条件下，若只使用参数，且设为128，那么将有一半的计算资源被浪费，而如果将设为2，设为64，则可以达到全部计算资源的使用。一般来说，在输入缓存优化结构中，为了减少片内存储的使用，应该优先调高值，其次才为值。而对于参数，则需要依据外接的DDR带宽与频率，进行计算得到。

本节实现Alexnet网络所使用的FPGA平台为Xilinx公司Zynq-7000系列的Z-7045，实际上，该FPGA平台为片上系统，即包括多个芯片的电路板。FPGA外接一个时钟频率为800M带宽为64位的DDR3，此外该平台上还有一个型号为Cortex-A9的ARM芯片以进行数据的控制。因为该FPGA芯片内含有900个乘法器，所以将设置为2，值设置为32，由于卷积单元内并行度为9，所以此时可以达到的并行倍数为：

 （5-6）

而由于DDR在时钟上下升沿都工作，对于时钟频率为800M带宽为64位的DDR3，在100M的FPGA工作时钟的条件下，可以设置的最大值为：

 （5-7）

到此，加速方案已经确定：为2，为32，为32。在加速方案确定完全之后，就可以根据参数对TTA处理器架构进行修改。首先是使用OSED工具对操作的输入输出接口数量进行设置，具体接口数量在第四章中已经说明；其次是在adf文件中，对功能单元的接口数量进行修改；再者在软件代码config文件中，对参数进行修改；最后是在功能单元所对应的HDL代码中，修改预留参数。在参数修改完之后，就可以使用已确定架构的TTA处理器对任意卷积神经网络进行加速。图5-10为加速AlexNet网络的主函数代码。

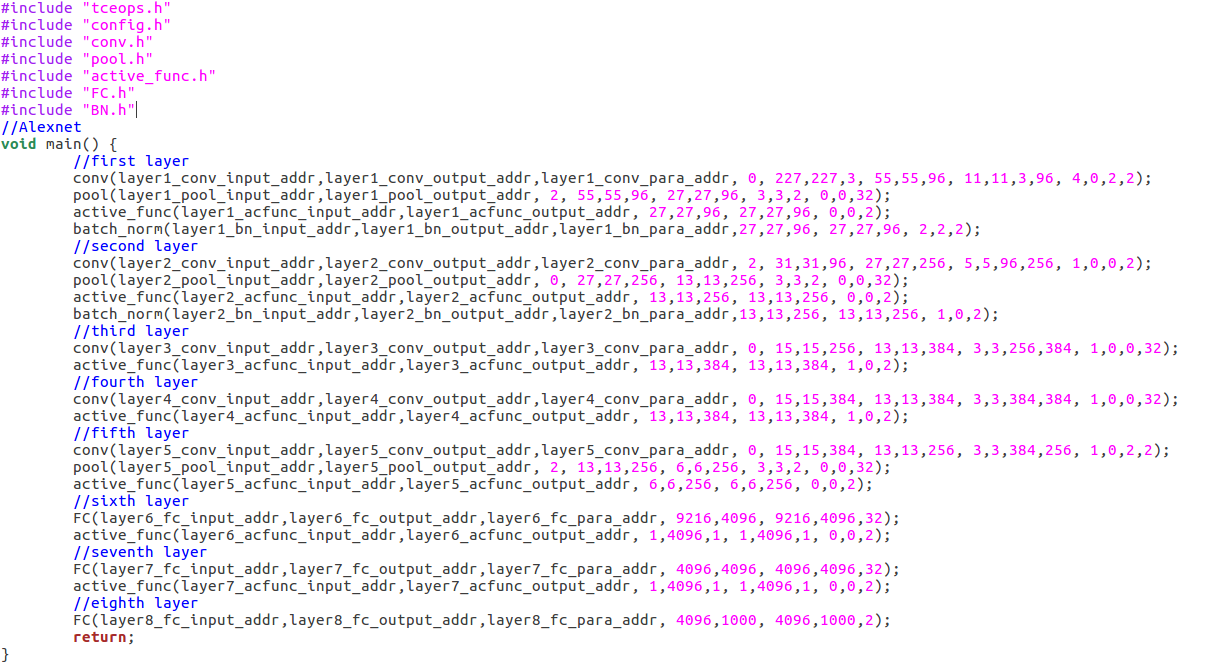


图5-10 TTA处理器AlexNet网络加速软件代码

而相比之下，基于FPGA平台的ASIC设计方式则需要更大的开发成本。图5-11为AlexNet网络的ASIC设计方式开发工程，其中共有18000+行verilog HDL代码。开发者不仅需要熟悉卷积神经网络的计算细节，还需要自行设计硬件平台的加速方案、存储结构、模块控制以及流水线设计等，对开发者的算法、硬件开发水平要求较高。同时，如果对新的卷积神经网络进行加速，还需要重新进行开发。而相比较之下，本文所提出的基于TTA架构的卷积神经网络处理器则不需要使用者有较高的硬件设计经验，只需根据FPGA资源进行加速方案的确定，根据加速方案对处理器架构进行修改后即可使用软件代码实现任意的卷积神经网络，在灵活性上达到了一个较高的水平。

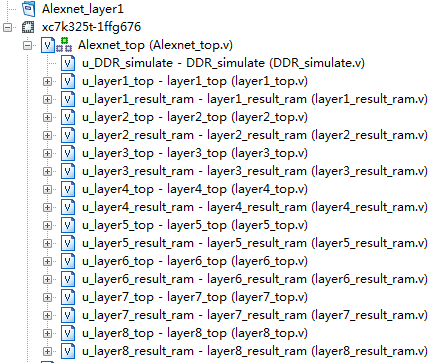


图5-11 AlexNet网络ASIC设计方式实现

在TTA处理器的性能方面，本文选择计算效率与片外访存量作为衡量指标。

假 设计算资源的工作时间为，总时间为，则计算效率的计算公式如式5-8所示。之所以选择计算效率而非计算时长作为本文的衡量指标，是由于对于处理器而言，随着其加速方案的变化，对于指定的加速任务，其所消耗的时间也是变化的。对于资源较多的FPGA平台，可以设定较高的加速参数，其所需要计算的时间也就越短，也就是说计算时间与计算资源量呈正比关系，而与处理器的性能优劣关系较小。而计算效率则是计算资源的运算时间与总时间的比值，计算效率越高，则说明在相同时间内，对于相同的计算资源，处理器所能完成的任务越多，处理器的性能也就越好。

 （5-8）

选择片外访存量作为第二衡量指标的原因在于，在卷积神经网络的计算过程中会伴随着大量的数据访存，若片外访存量需求过大，则在加速倍数较高的条件下，计算时间小于数据访存时间，此时计算单元需要等待数据的访存，从而使计算效率降低。换言之，片外访存速度限制加速倍数的上限，在全连接层的加速中，这种情况则最为明显，因此需要对片外访存量进行衡量。同理，选择片外访存次数而非片外访存速度作为衡量指标的原因在于处理器可以使用不同频率不同种类的DDR，而这些也会对片外访存速度造成较大影响。

对于所指定的卷积神经网络，使用本文所设计的TTA处理器与通用处理器CPU进行比较。在本文中，所使用的CPU为AMD Ryzen 5 1600，具有六核心十二线程。本文通过PyTorch深度学习框架，使用CPU对AlexNet网络进行加速，通过计算时长与CPU所含计算资源量即可计算出CPU的计算效率。而对于TTA处理器，分别通过仿真器查看各层的加载时间与计算时间，即可进行计算效率的计算。TTA处理器与CPU加速方案各层的计算效率如图5-12所示。同时各层的片外访存量如图5-13所示。

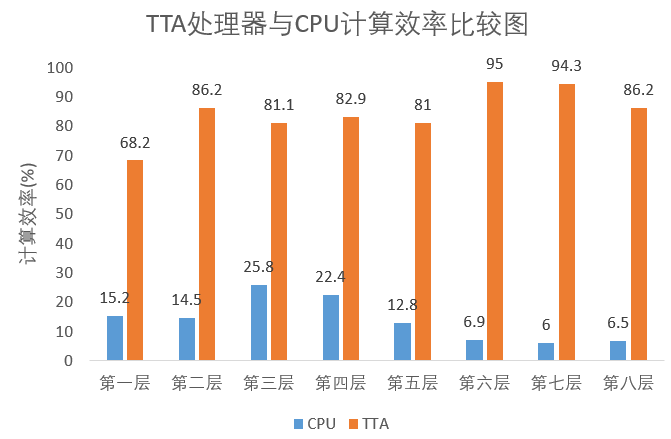


图5-12 TTA处理器与CPU计算效率

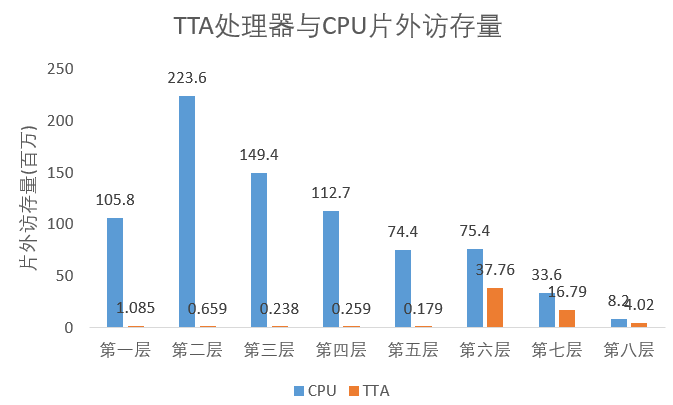


图5-13 TTA处理器与CPU片外访存量

对AlexNet网络的加速方案中，TTA处理器总计算效率为81.6%，CPU的计算效率为14.9%，可以说在卷积神经网络的加速方案中，本文所设计的TTA处理器架构在计算效率上明显高于CPU的计算效率，是CPU的5.5倍。同时TTA处理器所需总的片外访存量为61Mops，CPU所需总片外访存量为783.1Mops，TTA处理器所需片外访存量也远低于CPU所需，仅为CPU的7.8%。CPU的片外访存量较大主要是由于CPU未针对卷积层进行优化，导致参数与数据的复用率太低。因此，本文所设计的卷积神经网络TTA处理器在性能上优于传统的通用处理器CPU。

5.3 本章小结

本章的主要内容主要包括卷积神经网络中各运算层的具体实现以及卷积神经网络TTA处理器的灵活性与性能分析。各运算层的实现从TTA处理器的灵活性出发，以能够实现任意参数的运算层加速为目的进行了设计，给出了各运算层实现的伪代码。此外，对TTA处理器从加速方案的确定到指定卷积神经网络的加速进行了详细的说明，并与ASIC设计方式的开发流程进行了比较，指出本文所设计处理器在灵活性上有巨大的优势。最后，阐述了选择计算效率与片外访存量作为衡量指标的原因，并与传统通用处理器CPU进行了比较，在计算效率为CPU的5.5倍的情况下，只需7.8%的片外访存量即可完成对AlexNet网络的加速，在性能上具有巨大的优势。

第六章 总结与展望

6.1 工作总结

卷积神经网络作为人工智能发展中重要的一环，也受到人们的广泛关注。卷积神经网络计算量较大，因此在实际应用中需要依托于高性能硬件才可以达到实时的需求。FPGA由于具有高效能功耗比的特性，在卷积神经网络的加速中占有一席之地。但一般基于FPGA的项目研发都是基于ASIC的设计方式，虽然在性能上可以达到较好的表现，但需要大量的人力物力进行开发。本文旨在设计一种卷积神经网络专用处理器，在大幅度减少人力物力开发成本的前提下尽量保证加速性能。

本文基于TTA处理器架构对卷积神经网络处理器进行了设计，使用者可以通过FPGA内的计算及存储资源制定相应的加速方案。在确定加速方案后，通过TCE工具集对处理器架构进行调整，之后就可以通过简单的函数调用完成任意卷积神经网络的加速，保证了处理器的灵活性的同时具有较高的计算效率。本文的主要研究工作有：

（1）设计处理器计算框架并优化片内存储的使用。本文详细分析了串行计算框架与并行计算框架的特性及其所适应的环境。串行计算框架适用于层间串行的加速结构，只需满足单层最大的资源即可完成对所有层的加速，非常适用于加速未知层数的处理器设计。同时，在FPGA平台的资源中，片内存储资源较为稀缺，如何合理的利用片内存储资源是处理器设计的一个难点所在。串行计算框架包括输入缓存、计算单元以及输出缓存，本文针对输入缓存与输出缓存，提出了输入缓存优化结构以及输出缓存优化结构，在不影响运算的正常进行下，可以将存储资源使用量降低大约一半。

（2）针对卷积层及池化层设计输入缓存旋转存储结构。在卷积窗口内并行的实现上一般使用Z型单元，以减少所需的数据控制逻辑与数据带宽。但本文对Z型单元进行了分析，阐述了其存在消耗较多片内存储单元且在卷积操作步长大于1时计算效率低下的问题。因此，本文设计了旋转存储结构，在不增加计算与存储资源的条件下，可以解决功能单元随卷积步长增长计算效率降低的问题，同时也减少了功能单元内所需的片内存储资源。旋转存储结构同样适用于计算过程相似的池化层。

（3）基于TCE工具集的功能单元及互联网络设计。本文基于TCE工具链对卷积神经网络TTA处理器的功能单元及互联网络进行了设计，其中功能单元包括片外存储功能单元、输入缓存功能单元、通用计算功能单元以及输出缓存功能单元。本文对各个功能单元中的操作进行了详细介绍，包括其所实现的功能、输入输出接口、操作延时等。同时对TTA处理器的互联网络进行了设计，相比初始的全连接型互联网络，本文所设计互联网络的复杂度大幅度降低。

（4）自定义运算层实现。自定义运算层的实现从处理器的灵活性出发，以能够实现任意参数的运算层为目的。对于卷积层分析了如何通过3×3的操作单元实现任意大小的卷积操作，同时使用卷积操作单元进行偏置的计算，以减少控制逻辑。对于其它层也分别考虑到参数的影响进行了处理。通过简单的函数接口调用即可完成对任意规模任意层次的卷积神经网络进行加速，大大减少了开发成本。最后，对AlexNet网络进行了实现，与ASIC设计方式和CPU进行了比较，分析了其灵活性以及加速性能的优异性。

6.2 不足与展望

本文完成了基于TTA架构的大规模卷积神经网络处理器设计，但由于时间、资源以及TCE工具集本身所存在的一些问题，仍有许多地方存在不足。希望在接下来的工作中对以下所提出的问题进行更深层次的研究与完善：

（1）片外存储数据读写时间优化

在对不同运算层进行加速时，需要从片外存储中读取数据到输入缓存，在计算完成后，需要将数据从输出缓存中写入到片外存储中。由于数据读写时间固定，在加速倍数较小时，其相比于计算时间可以忽略，但随着加速倍数的提升，计算时间的不断缩短，此时数据的读写时间优化就显得更加重要。实际上，乒乓结构可以解决数据读写时间的优化。乒乓结构通过“输入数据选择控制”和“输出数据选择控制”按节拍、相互配合地在两个相同存储单元中进行来回切换，可以将缓冲的数据流没有停顿的送到后续模块。但由于TCE本身工具集的限制，无法在大颗粒度上进行功能单元的并行，因此无法将乒乓结构应用到处理器架构中。但乒乓结构思路可行，因此不失为优化的方向所在。

（2）数据精度优化

由于TCE工具集本身只含有int、double等常用的数据类型，无法自定义设置数据带宽，因此本文所设计的处理器都基于32bit定点化基础上进行的。但对于目前的神经网络FPGA加速方案，已经可以到达8bit量化而不大幅损失精度，更甚者，有学者设计1bit量化的针对于FPGA平台的特殊卷积神经网络。数据精度的优化可以带来计算资源与存储资源的减少，使得所需的FPGA平台成本降低，因此后续对数据精度进行优化显得尤为重要。

参考文献

1. Cao Z . Development and Application of Artificial Intelligence[C]// International Conference on Mechatronics Engineering & Information Technology. 2017.
2. 端木方霖.基于人工智能发展与应用的研究[J].中国商论,2018(32):27-28.
3. Lecun Y, Bengio Y, Hinton G. Deep learning[J]. Nature, 2015, 521(7553):436-444.
4. Bouvrie J. Notes on Convolutional Neural Networks[J]. Neural Nets, 2006.
5. Wang P, Li W, Liu S, et al. Large-scale Continuous Gesture Recognition Using Convolutional Neutral Networks[J]. 2016.
6. Delakis M, Garcia C. Text Detection with Convolutional Neural Networks[C]// Visapp 2008: Proceedings of the Third International Conference on Computer Vision Theory and Applications, Funchal, Madeira, Portugal, January. DBLP, 2015:290-294.
7. Yuan Z W, Zhang J. Feature extraction and image retrieval based on AlexNet[C]// Eighth International Conference on Digital Image Processing. 2016:100330E.
8. 万子平,汪琳,段国栋.浅析ASIC与PCB的联系和区别[J].电子世界,2016(16):69-71.
9. 杨君. 专用指令集处理器（ASIP）体系结构设计研究[D].中国科学技术大学,2006.
10. 刘俊, 谢憬, 王琴. 基于TTA技术的专用处理器设计[J]. 微电子学与计算机, 2009, 26(11).
11. Wolf W. FPGA-Based System Design[M]. Prentice Hall PTR, 2004.
12. Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J]. Computer Science, 2014.
13. Szegedy C, Liu W , Jia Y , et al. Going Deeper with Convolutions[J]. 2014.
14. Chollet, François. Xception: Deep Learning with Depthwise Separable Convolutions[J]. 2016.
15. He K , Zhang X , Ren S , et al. Deep Residual Learning for Image Recognition[J]. 2015.
16. A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in NIPS, 2012, pp. 1097–1105.
17. In-Datacenter Performance Analysis of a Tensor Processing Unit[J]. 2017.
18. T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam,“Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning,” in ASPLOS, vol. 49, no. 4. ACM, 2014, pp. 269–284.
19. Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun et al., “Dadiannao: A machine-learning supercomputer,” in MICRO. IEEE, 2014, pp. 609–622.
20. D. Liu, T. Chen, S. Liu, J. Zhou, S. Zhou, O. Teman, X. Feng, X. Zhou, and Y. Chen, “Pudiannao: A polyvalent machine learning accelerator,” in ASPLOS. ACM, 2015, pp. 369–381.
21. Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, “Shidiannao: shifting vision processing closer to the sensor,” in ISCA. ACM, 2015, pp. 92–104.
22. 方睿, 刘加贺, 薛志辉, et al. 卷积神经网络的FPGA并行加速方案设计[J]. 计算机工程与应用唯一官方网站, 2015, 51(8):32-36.
23. Zhao W , Fu H , Luk W , et al. F-CNN: An FPGA-based framework for training Convolutional Neural Networks[C]// IEEE International Conference on Application-specific Systems. IEEE, 2016.
24. Li H , Fan X , Jiao L , et al. A high performance FPGA-based accelerator for large-scale convolutional neural networks[C]// International Conference on Field Programmable Logic & Applications. IEEE, 2016.
25. 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. ACM, 2016.
26. 凡保磊. 卷积神经网络的并行化研究[D]. 郑州大学, 2013.
27. Mahajan A, Gill P. 2D Convolution Operation with Partial Buffering Implementation on FPGA[J]. International Journal of Image Graphics & Signal Processing, 2016, 8(12):55-61.
28. Benedetti A, Prati A, Scarabottolo N. Image convolution on FPGAs: the implementation of a multi-FPGA FIFO structure[C]// Euromicro Conference. IEEE, 1998.
29. Carlo S D, Gambardella G, Indaco M, et al. An area-efficient 2-D convolution implementation on FPGA for space applications[C]// Design and Test Workshop. IEEE, 2011:88-92.
30. Perri S, Lanuzza M, Corsonello P, et al. A high-performance fully reconfigurable FPGA-based 2D convolution processor[J]. Microprocessors & Microsystems, 2005, 29(8):381-391.
31. 朱学亮, 柴志雷, 钟传杰, et al. 基于FPGA的图像卷积IP核的设计与实现[J]. 微电子学与计算机, 2011, 28(6):188-192.
32. 侯宇昆. 卷积神经网络概述[J]. 中国新通信, 2017, 19(9):45-45.
33. Ramachandran P, Zoph B , Le Q V . Searching for Activation Functions[J]. 2017.
34. Li Y, Wang N, Shi J , et al. Revisiting Batch Normalization For Practical Domain Adaptation[J]. Pattern Recognition, 2016.
35. Philips Inc. An Introduction to Very-Long Instruction Word(VLIW) Computer[M]. pp.1-11. 1997.
36. Balakrishnan S. Very long instruction word processors[J]. Resonance, 2001, 6(12):61-68.
37. 赵学秘, 王志英, 岳虹, et al. 传输触发体系结构指导下的ASIP自动生成[J]. 计算机辅助设计与图形学学报, 2006, 18(10).

致谢

攻读学位期间取得的学术论文目录

[1]