12figure.caption.15

南京工業大學

2021 届毕业设计（论文）

题　　目:	深度神经网络硬件加速系统设计与优化
专　　业:	电子信息工程
学　　号:	1402170120
姓　　名:	王磊
指导老师:	朱艾春
起讫日期:	2020.12.01 ~ 2021.06.07

2021 年 6 月

深度神经网络硬件加速系统设计与优化

摘要

近年来，深度神经网络已经被证明在包括图像分类、目标检测和自然语言处理等任务上能够取得相当不错的效果。现如今，大量的应用程序都配备了与之相关的深度学习算法，但是对于手机、无人机等资源有限的嵌入式设备上，仅用软件方式加速深度神经网络已经不能满足日益增长的速度和功耗要求，如何利用硬件设计加速器已经成为学术领域的研究热点。

本文首先给出了深度神经网络的概述，并介绍了几种当下主流的针对深度神经网络的硬件加速系统设计方案以及实际事例。在考虑了不同加速器设计之间的相似性和差异性之后，本文将对英伟达开源的深度学习加速器框架 NVDLA 的设计思路进行概述。然后，本文基于 ZYNQ 7000 器件，在 FPGA 端实现了 NVDLA 设计，通过 AXI 4 总线协议将 NVDLA 加速器挂载到 ARM A9 处理器，并为 ARM A9 处理器移植了 Ubuntu 16.04 操作系统，将加速器的驱动程序加载到 Linux 内核。在软件设计的前端，使用神经网络编译器接受预训练的深度神经网络模型，做算子融合、内存重用等硬件无关的优化，并且结合 TensorRT 完成了深度神经网络的量化，在软件设计的后端，由 Runtime 自动调度加速器驱动程序推理深度神经网络。

关键词： 深度神经网络　FPGA　NVDLA　硬件加速

Hardware Acceleration System Design and Optimization of Deep Neural Network

Abstract

In recent years, deep neural networks have been proven to achieve quite good results in tasks including image classification, target detection, and natural language processing. Nowadays, a large number of applications are equipped with related deep learning algorithms. However, for embedded devices with limited resources such as mobile phones and drones, only using software to accelerate deep neural networks can no longer meet the increasing speed and Power consumption requirements, how to use hardware design accelerators have become a research hotspot in the academic field.

This article first gives an overview of deep neural networks, and introduces several current mainstream hardware acceleration system design schemes and practical examples for deep neural networks. After considering the similarities and differences between different accelerator designs, this article will outline the design ideas of Nvidia's open source deep learning accelerator framework NVDLA. Then, based on the ZYNQ 7000 device, this article implements the NVDLA design on the FPGA side, mounts the NVDLA accelerator to the ARM processor through the AXI4 bus protocol, and transplants the Ubuntu 16.04 operating system for the ARM processor, and mounts the accelerator driver to The Linux kernel uses the Runtime automatic scheduling accelerator to infer the deep neural network. At the front end of the software design, the neural network compiler is used to accept the model trained by Caffe to perform hardware-independent optimizations such as operator fusion and memory reuse, and combined with TensorRT to complete the quantization of the neural network.

Keywords: Deep neural network; FPGA; NVDLA; Hardware speedup

目　录

第一章　绪论

1.1　研究背景及国内外研究现状

总的来说，深度神经网络的硬件加速系统设计可以分为两个阶段，一是上世纪 50 年代第一次人工智能热潮开始到第二次人工智能热潮结束期间，针对感知机和多层感知机等浅层神经网络研发的神经网络计算机/芯片，二是针对第三次人工智能热潮中的深度神经网络设计的深度学习处理器。

1.1.1　针对感知机和多层感知机等浅层神经网络研发的神经网络计算机/芯片

在第一次人工智能热潮中，D. Hebb 提出了 Hebb 学习法则，在这之后不久的 1951 年，M. Minsky 研制出了国际上首台神经网络模拟器 SNARC；1957 年，F. Rosenblatt 提出了感知机模型，随后国际上首台基于感知机的神经网络计算机 Mark-I 就在 1960 年被研制出来，它可以连接到照相机上使用单层感知机完成简单的图像处理任务。

在第二次人工智能热潮中，著名的反向传播算法被提出使得神经网络的研究取得了一些重要突破。20 世纪 80 年代和 90 年代初，很多大公司、创业公司和研究机构都参与到了神经网络计算机/芯片的研制，包括 Intel 公司研发的 ETANN、1990 年发布的 CNAPS 、基于脉动阵列结构的 MANTRA I，以及 1997 年由中国科学院半导体研究所研发的预言神等。

然而，从当今深度学习技术发展的角度来看，这些早期的神经网络计算机/芯片有诸多缺陷，由于其只能处理很小规模的浅层神经网络算法，所以没有在工业实践中获得广泛应用。这一方面是因为浅层神经网络的应用比较局限，缺乏像当下的诸如目标检测、自然语言处理等领域的核心应用；另一方面，当时的主流集成电路工艺还是 1 微米工艺（今天的主流集成电路工艺已经达到 7 纳米），在一个芯片上只能放数量相当少的运算器，Intel 的 ETANN 芯片中仅能集成 64 个硬件神经元；最后，受限于当时的计算机体系结构技术还没有发展成熟，没有办法将大规模的算法神经元映射到少数的硬件神经元上。

而第二次人工智能热潮也随着日本的五代机计划失败而结束。基于上述原因，从 20 世纪 90 年代开始，神经网络计算机/芯片的创业公司纷纷破产，大公司也裁减掉了相关的研究部门，各个国家暂停了这方面的科研资助，但这些瓶颈在处于第三次人工智能热潮的今天都得到了解决和改善，于是深度神经网络的硬件加速系统设计进入了第二个阶段。

1.1.2 针对第三次人工智能热潮的深度学习处理器

2006 年，深度学习技术由 G. Hinton、Y. Bengio 和 Y. LeCun 等人的推动而兴起，于是有了第三次人工智能热潮。而深度学习处理器也在这种环境下重新焕发了生机，在 2008 年，中国科学院计算技术研究所的陈云霁、陈天石等人开始了人工智能和芯片设计的交叉研究，之后来自法国 Inria 的 O. Temam 也参与到可项目中。在这些人的推动下，国际上第一个深度学习处理器架构 DianNao 于 2013 年被设计出来。和第一阶段设计的神经网络计算机/芯片不同，DianNao 架构不会受到网络规模的限制，可以灵活、高效地处理上百层的深度学习神经网络，并且显得更对于通用的 CPU，DianNao 可以去的两个以上数量级的能效优势。2014 年，陈云霁等人在 DianNao 架构上改进，设计出了国际上首个多核的深度学习处理器架构 DaDianNao ，以及机器学习处理器架构 PuDianNao 。进一步，中国科学院计算技术研究所提出了国际上首个深度学习指令集 Cambricon 。在这之后，首款深度学习处理器芯片“寒武纪 1 号”问世，目前寒武纪系列处理器已经应用于超过一亿台嵌入式设备中。

1.2 研究意义及前景

深度神经网络已经被证明在包括图像分类、目标检测和自然语言处理等任务上能够取得相当不错的效果。而随着其层数和神经元数量以及突触的不断增长，CPU 和 GPU 等传统体系已经很难满足神经网络增长的速度和需求。例如 2016 年，Google 公司研发的 AlphaGo 与李世石进行围棋对弈时使用了 1202 个 CPU 以及 176 个 GPU ，在这场比赛中 AlphaGo 每盘棋需要消耗上千美元的电费，而作为人类的李世石一盘棋的功耗仅需要 20 瓦，从此可以看出传统芯片的速度和能效难以满足大规模深度学习应用的需求。如今，大量的应用程序都配备了与之相关的深度学习算法，但是对于手机、无人机等资源有限的嵌入式设备上，仅用软件方式加速深度神经网络已经不能满足日益增长的速度和功耗要求，基于深度神经网络的智能应用需要广泛普及的趋势使高性能、低功耗的深度学习处理器研发呼之欲出，如何利用硬件设计加速器已经成为学术领域的研究热点。

1.3 研究内容及结构安排

硬件设计层面，本文主要在 FPGA 平台上设计实现了 NVDLA 深度学习加速器，并将其接口封装为 AXI4 总线协议与双核 ARM A9 处理器互联。软件设计层面，在前端，使用神经网络编译器接受预训练的神经网络模型，并做硬件无关优化。在后端，将驱动程序挂

载到了 Ubuntu 操作系统上，由 Runtime 自动调度推理。最后，分析了所设计的硬件加速器的性能。

本文共有七个章节，论文结构安排如下：

第一章：主要介绍了课题的研究背景及研究意义，分析深度神经网络加速器系统设计的国内外研究现状，并提出本文的研究方向。

第二章：主要对本文研究所涉及到的相关技术背景的介绍，首先介绍了人工神经网络，并对本文所使用的算法——YOLO V2 算法进行了详细的分析，同时介绍了 RISC-V 处理器相关的背景以及在本系统中为何选用 RISC-V 处理器。

第三章：总体介绍了系统的设计思路：先对 YOLO V2 算法进行了详细的分析，结合 FPGA 与 CPU 各自的特点，对整个神经网络算法的加速进行了任务划分，并给出本设计所使用的逻辑结构。

第四章：对 CNN 加速器进行详细设计及优化，介绍了使用 HLS 设计 CNN 加速器过程中的优化方案以及具体应用情况。

第五章：由于 CNN 加速器需要配合 RISC-V 软核进行异构运算，因此本章将 CNN 加速器作为 RISC-V 处理器的外设，设计了一个通用 RISC-V SOC，介绍了该 SOC 的硬件架构以及详细设计过程。

第六章：主要计算分析了 CNN 加速器的资源消耗，结合常见的运算平台，设计了 3 组典型的对照环境，将本设计中的异构计算系统与其分别进行对比，分析了该异构计算系统的优缺点。

第七章：总结全文的研究内容，对本课题中可以进一步研究的方向进行讨论。

第二章　相关技术介绍

由于本硬件加速系统设包含了软件与硬件的协同设计，所以本章将对涉及到的关键技术，包括深度神经网络算法、ZYNQ 器件做基本的介绍。如第一章节所述，本章还会对不同类型的硬件加速体系结构进行基本概述，介绍对应的应用案例。

2.1　深度神经网络概述

2.1.1　单层感知机模型

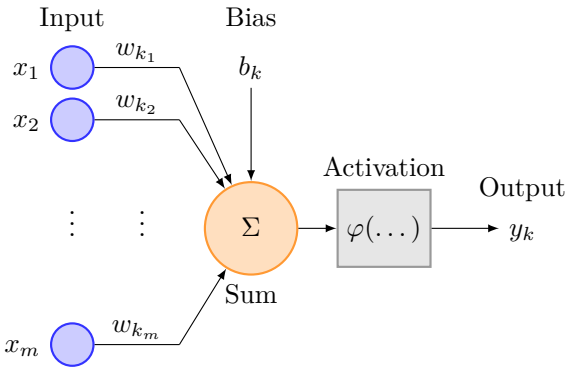


图 2-1 神经元模型

一个神经元的模型如图 2-1所示，该模型也被称为单层感知机，由输入信号、权值、偏置、加法器和激活函数共同构成，其中 w_{kj} 下标的含义为：k 表示第 k 个神经元；j 表示第 j 个输入。因此，w_{kj} 表示第 k 个神经元的第 j 个输入对应的权值。单个神经元的数学公式如公式 (2-1) 所示：

$$\begin{cases} u_k = \sum_{j=1}^{m} w_{kj} x_j \\ v_k = u_k + b_k \\ y_k = \varphi(v_k) \end{cases} \tag{2-1}$$

其中 x_j 表示第 k 个神经元的第 j 个输入，w_{kj} 表示第 k 个神经元的第 j 个输入对应的权值，b_k 表示第 k 个神经元的偏置，$\varphi(\dots)$ 为激活函数，y_k 为该神经元的输出。

感知器模型于 1958 年，由美国心理学家 Frank Rosenblatt 提出，其中激活函数采用的一般是符号函数，如公式 (2-2)所示：

$$o = sgn(x_1 w_1 + x_2 * w_2 + b) \tag{2-2}$$

进一步，为了简化问题分析本质，我们假设神经元只有两个输入: x_1 和 x_2 ，则模型的公式进一步简化为:

$$\begin{cases} 1, & x_1 w_1 + x_2 * w_2 + b >= 0 \\ -1, & x_1 w_1 + x_2 * w_2 + b < 0 \end{cases} \tag{2-3}$$

如果把 o 当作因变量、x_1 和 x_2 当作自变量，对于分界

$$x_1 w_1 + x_2 w_2 + b = 0 \tag{2-4}$$

可以抽象成三维空间里的一个分割面，能够对该面上下方的点进行分类，则该感知机能完成的任务是用简单的线性分类任务，比如可以完成逻辑 “与” 与逻辑 “或” 的分类，如表 2-1所示，在这里，第三维度只有 1 和-1 两个值，分别使用实心点和空心点来表征，这样就可以在二维平面上将问题可视化：

表 2-1 逻辑真值表

逻辑与			**逻辑或**			**逻辑异或**		
x_1	x_2	o	x_1	x_2	o	x_1	x_2	o
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

但是对于非线性问题，如异或问题，如图 2-2所示，单层感知机没有办法找到一条直线能够完成分类任务。

2.1.2　多层感知机模型

多层感知机模型是在单层感知机模型的基础上引入了一层或多层隐藏层，在图 2-3所示的多层感知机中，输入和输出的神经元个数分别为 4 和 1，中间的隐藏层包含了 5 个神经元，隐藏层中的神经元和输入层中各个输入完全连接，输出层中的神经元和隐藏层中的各个神经元也完全连接。因此，多层感知机中的隐藏层和输出层都是全连接层。

但是不难发现，即便再添加更多的隐藏层，多层感知机的设计依然只能与仅含输出层的单层神经网络等价。上述问题的根源在于全连接层只是对数据做仿射变换（affine

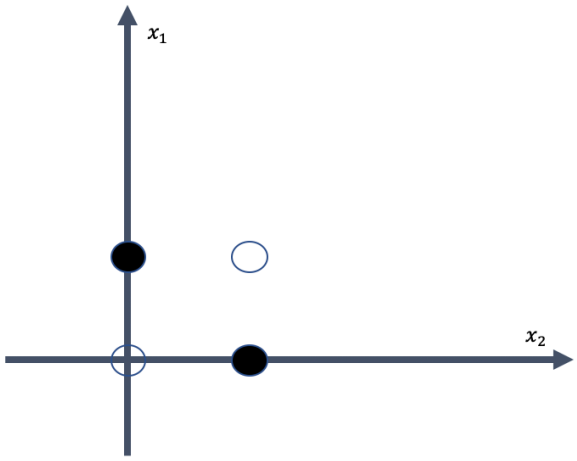


图 2-2 异或平面

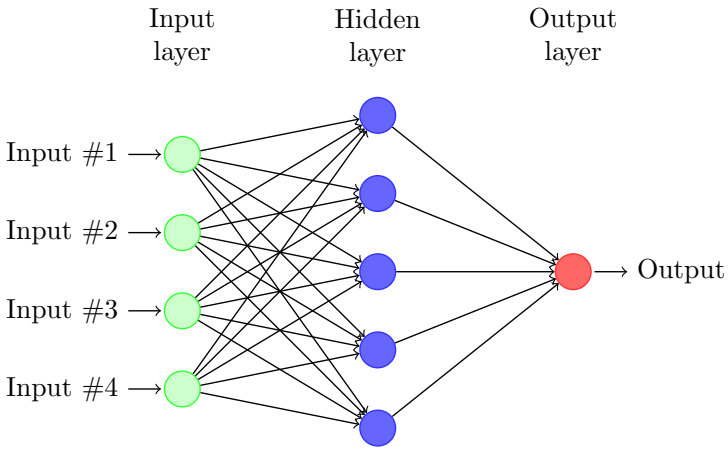


图 2-3 前向神经网络结构

transformation），而多个仿射变换的叠加仍然是一个仿射变换。解决问题的一个方法是引入非线性变换，例如对隐藏变量使用按元素运算的非线性函数进行变换，然后再作为下一个全连接层的输入。这个非线性函数被称为激活函数（activation function），常用的激活函数包括了 Sigmoid、Relu、Tanh 等。

理论上，深度神经网络，及多层感知机已经可以拟合任意的函数[?]。

2.1.3　卷积神经网络概述

卷积神经网络（Convolutional Neural Network, CNN）是目前最常用的深度神经网络，对于大型图像处理有出色表现，卷积神经网络一般由卷积层、非线性激活函数、池化层组成。

1. 卷积层（Convolution Layer）的本质是输入图像与权重矩阵的乘积累加运算 (Multiply Accumulate, MAC)，如图 2-4所示卷积核在输入特征图像上滑动，用来匹配局部的细节。一般来说，随着卷积的层数逐渐加深，卷积核的数目也会随之增加，所表征的细节也会更加抽象。

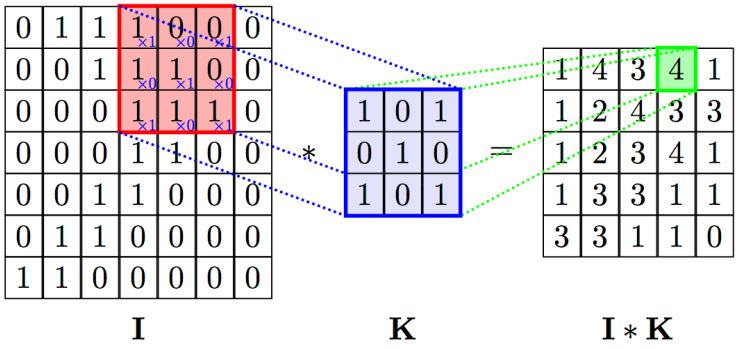


图 2-4 卷积运算

2. 激活函数（Activation Function）是非线性单元，它的存在给深度神经网络体系增加了非线性元素，是深度神经网络能够拟合任意函数的基础。比较重要且常用的激活函数有 ReLU、Sigmoid 和 Tanh 。此外，其余常用的激活函数还有 PReLU, ELU 等，他们的函数图像与取值范围如图 2-5所示：

3. 池化层（Pooling Layer）的作用是对特征图进行下采样，从而可以达到增加感受野、增加卷积神经网络运算的平移不变性、降低优化难度，减少神经网络参数的目的。典型的池化层 Max Pooling 如图 2-6所示，他会选择被池化矩阵选内最大的值作为输出得到新的特征图。Average Pooling 选取池化矩阵内的平均值，是另一种常用的池化层。

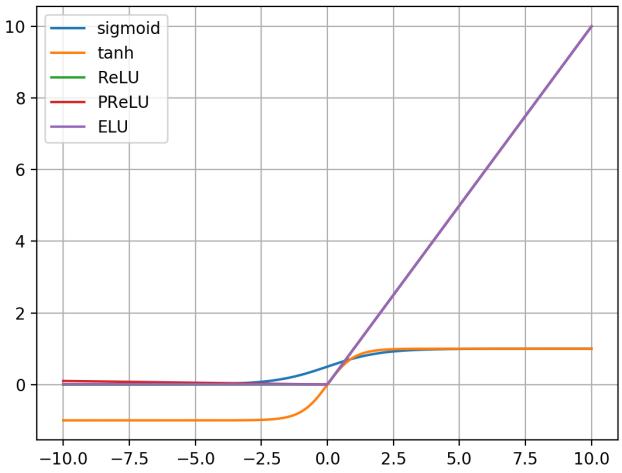


图 2-5 常见的激活函数图像

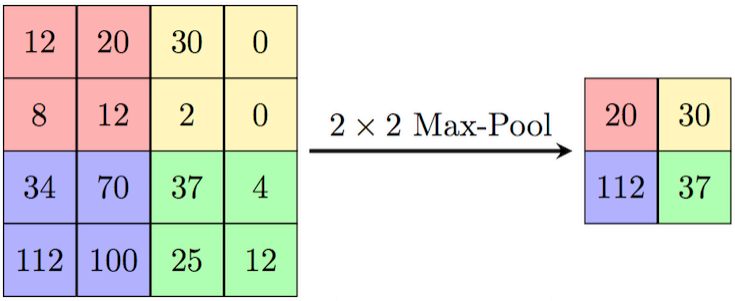


图 2-6 Max Pooling

2.1.4　LeNet5 介绍

手写字体识别模型 LeNet5 诞生于 1994 年，是最早的卷积神经网络之一。

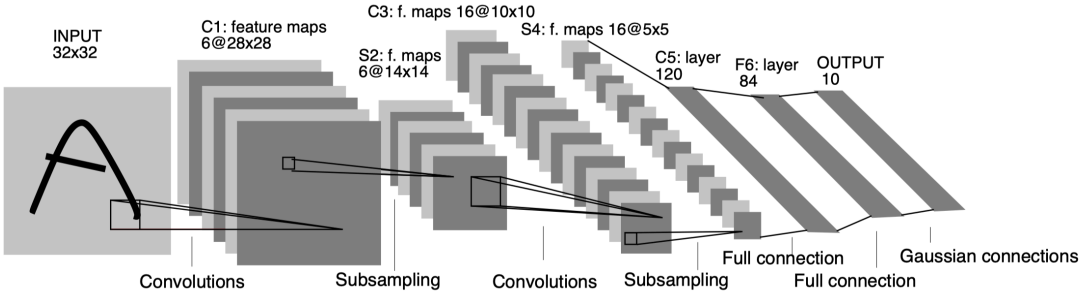


图 2-7 LeNet-5 网络结构

通过巧妙的设计，利用卷积、参数共享、池化等操作提取特征，LeNet5 避免了大量的计算成本，最后再使用全连接神经网络进行分类识别，该网络也是最近大量神经网络架构的起点，LeNet5 的网络结构如图 2-7所示。

2.1.5　Resnet 18 网络结构

纵观神经网络的发展脉络，我们可以看到研究者一直在不断尝试各种技术以加深神经网络的层数。从 8 层的 AlexNet，到十九层的 VGG，再到 22 层的 GoogLeNet，可以说，技术一直在进步。但是在突破神经网络深度的问题上真正最具有颠覆性的技术还是来自 ResNet。

为了避免随着神经网络的深度加深从而带来的退化问题，ResNet 采用了一种不同于常规卷积神经网络的基础结构，其基本单元如图 2-8所示，增加了从输入到输出的直连通道。其卷积拟合的是输出与输入的差，及残差，所以这种结构又被称为残差结构。残差网络的响应小于常规网络[?]。残差网络的优点是对数据波动更加灵敏，更容易求的最优解，能够改善深层网络的训练。

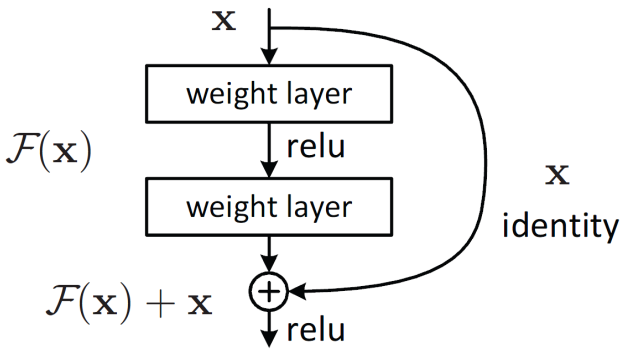


图 2-8 Resnet 残差结构

2.2 FPGA 与 ZYNQ 器件

现场可编程逻辑门阵列（Field Programmable Gate Array, FPGA）, 是一种半定制的集成电路（Integrated Circuit, IC）芯片。FPGA 包含了一组可编程逻辑门阵列（Programmable Logic Blocks, PLB）以及可重配置的互联层次结构，通过这种结构可以将 PLB 连接在一起。FPGA 可以通过重编程，实现不同逻辑特性，从而实现了可重构计算。因此理论上 FPGA 可以实现当前 CPU 上的所有算法，但是具体算法实现的效果会受制于 FPGA 的可用资源、时钟频率以及输入输出的带宽。

ZYNQ 器件是 Xilinx 公司研发的以 ARM 为主导，FPGA 为辅的嵌入式系统结构，其拥有集成度高的开发环境，支持高层次综合、Vivado 硬件逻辑设计、Simulink 协同设计、SDK 软件设计。对于一个不熟悉 FPGA 的软件工程师来说，也完全可以把 ZYNQ 器件当前简单的双核 ARM 来使用，利用 SDK 编写软件程序，如果软件调试过程中发现某些算法的速度太慢，这时候就可以使用 Verilog 硬件描述语言，或者使用高层次综合为这部分算法设计硬件加速器，ARM 处理器与加速器之间通过 AXI 标准总线协议进行通信，Xilinx 提供了若干免费的加速 IP 供用户使用，由于有了 ARM 处理器，我们甚至可以在 ZYNQ 器件上搭建 SOC 降低开发的难度。

2.3 硬件加速体系结构概述

在过去，深度神经网络算法往往被运行在通用体系结构里，如众核处理器与 GPU 设备，但是这些设备的特点是运行功耗大，并且会带来极高的内存使用率。当谈及深度神经网络，往往最让我们头疼的是训练过程，这个步骤一般在使用 GPU 完成加速，然后将预训练好的模型部署在目标平台，当平台对功耗和运算能力有要求的时候，传统的通用处理器的体系结构就不适合了，于是领域专用的，针对深度神经网络的硬件加速系统设计呼之欲出。本小节首先介绍两种典型的通用硬件加速体系结构，然后介绍三种面向深度神经网络运算的领域专用运算。

2.3.1 众核处理器

众核处理器（Manycore Processor）由成百上千的简单分立的处理器核心组成。与传统的多核处理器所不同的是，其设计牺牲了单核运行的性能来换取多核处理时的吞吐量，减少多核处理时的能量损耗。典型的使用众核处理器设计是 Kalray MPPA-256 架构。

如图 2-9所示，使用众核处理器进行深度神经网络运算时，每个处理器核心都可以贡献出一部分性能进行神经网络的运算，之后通过 Router 决定在物理总线上数据传输的方

向。

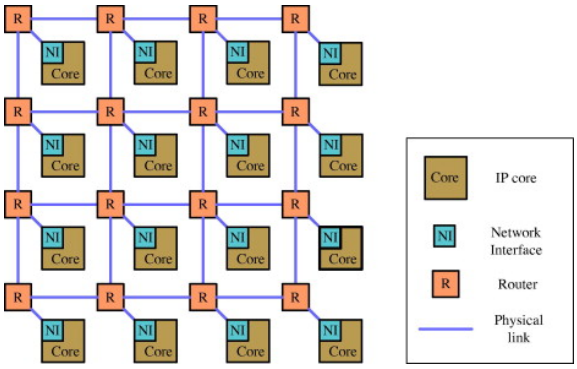


图 2-9 Net work on Manycore Processor

众核处理器是一种通用处理器，这意味着他具有一定的灵活性，除了可以运用在神经网络运算上，还可以用在各种各样的领域，尤其是那些有高并行度要求的场景，例如 HPC。但如果仅针对深度神经网络的推理，通用处理器本身便会存在一些多余、不合适的设计。

2.3.2　GPU

GPU 也可以看作是一种特殊的，针对向量运算的众核处理器。起初 GPU 被设计出来的目的是为了提高计算机对图像与视频的处理能力，但是当下，GPU 设备被广泛应用在深度学习应用的训练和推理加速中。并且对于 GPU 的调用，NVIDIA 还提供了软件支持，即 CUDA，方便了用户手动便携并行算子。

VOLTA 架构是 NVIDIA 推出的新一代的 GPU 设计架构，基于该架构的显卡被广泛运用在深度学习领域。其中，TESLA V100 是第一款被设计出来专门针对深度神经网络训练和推理的 GPU 设备（如图 2-10）。

图 2-10 TESLA V100

2.3.3 Google TPU

张量处理单元（Tensor Processing Unit）是 Google 牵头设计的一款为深度神经网络运算定制的 ASIC 器件，事实上其也取得了非常瞩目的性能，2015 年发布的 29nm 工艺的 TPUv1 版本，能够运行在 700Mhz 的频率，但功率只有 28W ~40W。

2.3.4 DianNao 系列

为了能够实现大规模深度学习网络的专用硬件执行。2014 年，中国科学院计算技术研究所研究团队提出了 DianNao 架构。DianNao 这篇文章是率先探索机器学习加速设计的先驱文章之一，开创了专用处理器实现深度学习的先河。这篇文章在 65nm 工艺，0.98Ghz 的频率，面积为 3.02mm2 的 ASIC 芯片上针对机器学习算法（DNN，CNN）实现了一个高性能的 Diannao 处理器架构，相比于 128bit 2GHz 的 4 发射 SIMD 处理器，达到了 117.87x 的加速比，21.08x 能耗比。

2.3.5 NVIDIA Deep Learning Accelerator

NVDLA 是 NVIDIA 公司开源出来的深度学习加速器框架，其结构与 DianNao 类似。DianNao 系列芯片每个周期最多能够计算 16 个神经元，NVDLA 能够计算 64 个，而 TPU 则能够计算 256 个。

与前两个领域专用的硬件加速体系结构不同，NVDLA 完全开源，其不仅提供了 Verilog 硬件描述语言、CMOD 硬件仿真程序，还为硬件加速器设计了神经网络编译器与运行时，自上而下打通了硬件栈与软件栈，基本能够实现端到端的推理。虽然 NVDLA 自 2018 年以来已经没有人维护，但其系统设计思路仍然具有学习意义和指导意义。

因此，有关 NVDLA 的内容将在下一章节具体介绍。

第三章　系统总体设计

本章将介绍系统的总体设计结构，包括对 NVDLA 的详细介绍，选用的 FPGA 开发板卡资源介绍，NVDLA 与 ARM 互联的解决方案。

3.1　NVDLA

NVDLA 是 NVIDIA 公司在 2017 年年末公布并且开源的，遗憾的是该项目自公布完成一年后便草草停止了维护。NVDLA 是一款模块化、可配置的神经网络推理加速器框架。通过核心 MAC 阵列、与查找表逻辑，NVDLA 可以支持深度神经网络算子，已经支持的有：

- 卷积运算
- 激活函数
- 池化
- 归一化

实际上，由于 NVDLA 是完全开源的，完全可以自己手动添加相关算子的硬件逻辑。

3.1.1　NVDLA 硬件结构分析

前文中提到，NVDLA 具有模块化的属性，这很有助于我们理解加速器的工作流程，如图 3-1所示，NVDLA 的各个模块可以独立、同时工作。

- 卷积引擎配置了卷积 Buffer，在使能卷积运算的时候，需要先通过寄存器配置给出 Input Image 与 Weights 在内存上的地址，然后将数据缓存到 Buffer 中来进行运算，减少访存的开销。NVDLA 支持两种卷积模式：
  - 直接卷积，及标准的卷积，可以通过 MAC 阵列并行加速。
  - Winograd 快速卷积，从算法层面通过共用权重来减少运算量。

卷积引擎的核心是 MAC 阵列，该阵列的大小是可配置的，本文使用阵列大小为 8x8 。卷积的结果不可以直接写回内存，必须经过 SDP，及激活函数层，其他引擎则没有这个限制。

- 除去卷积引擎之外，NVDLA 一共还有五个引擎，他们是负责完成激活函数的 SDP 引擎，负责完成池化操作的 PDP 引擎，负责完成 LRN 操作的 CDP 引擎，做图形 Reshape 的 RUBIK 引擎，最后，NVDLA 设计还提供了一个 BDMA 引擎，用来在 DRAM 和高速存储之间搬移数据。

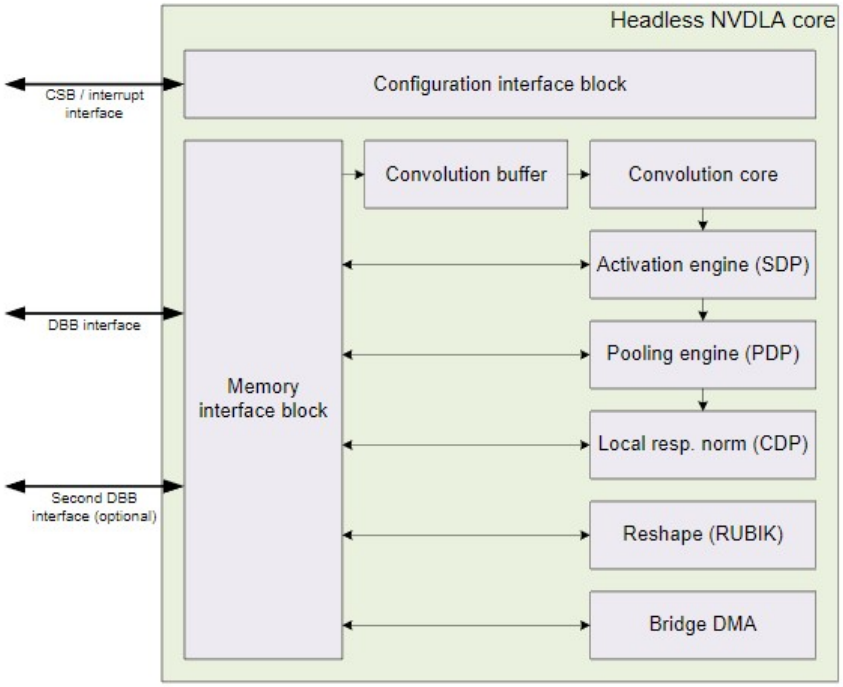


图 3-1 NVDLA 体系结构

NVDLA 在外侧暴露出四个接口，它们分别是 CSB 总线，主存储接口、高速存储接口、中断接口。

- 寄存器配置总线（Configuration Space Bus Interface，CSB），用来读写 NVDLA 的寄存器，对于 CSB 总线，我们不甚讲解，其在读写地址的时候需要做移位来压缩指令大小，NVDLA 提供了 csb2apb 转换电路，在读写的时候使用该电路包裹即可利用 APB 总线协议读写 NVDLA 的寄存器。
- 主存储接口（Data Backbone interface，DBB），用来访存，该接口使用的协议为 AXI 总线协议，可以挂载在 AXI 接口的存储控制器上，在本文中，我们将其挂载到片上的 DDR 存储，与 ARM 处理器共享内存。
- 高速存储接口（high-bandwidth interface），可以外接第二块 SRAM 作为辅存储，存储中间数据，降低访存瓶颈。
- 中断接口（Interrupt interface），NVDLA 不仅支持通过寄存器查询的方式查询任务是否完成，也支持产生硬件中断，方便我们编写驱动程序。

3.1.2　NVDLA 自定义配置

在本设计中，我们使用官方提供的 small 配置，即最精简的 NVDLA 配置，该配置有如下特点：

1. 最小化 MAC 阵列，本设计将 NVDLA 的 MAC 阵列大小配置为 8 * 8。

2. 关闭高速存储接口，只使用主存储接口。
3. 仅支持 INT8 格式的 MAC 运算。
4. 不支持权重压缩。
5. 不支持 WINOGRAD 快速卷积。
6. 不支持 Reshape 特征图。

3.1.3 NVDLA 软件栈

如图 3-2，NVDLA 的软件栈分为 Compiler 和 Runtime 两个部分。

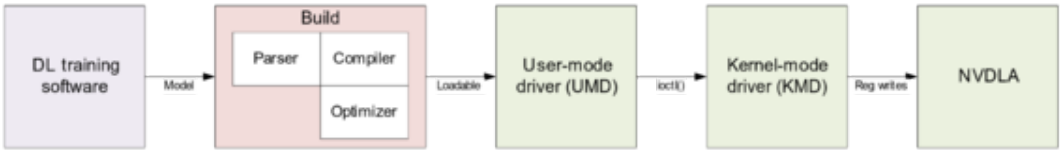


图 3-2 NVDLA 软件栈

Compiler 与硬件无关，可以在主机运行。它可以接受其他深度神经网络训练框架训练完成的模型，完成一些硬件无关的优化，例如算子融合、数据重用等，并且使用 FlatBuffers 将处理结果封装成 Loadable 数据流文件。

Runtime 与硬件紧密结合，其负责接受 Compiler 生成的 Loadable 文件，调度加速器程序，自动分配内存，自动配置寄存器，自动处理中断。而 Runtime 内部又被划分为两个部分，UMD 和 KMD 吗，分别对应了 Linux 的用户态驱动与内核驱动。

- 用户态驱动程序（USER MODE DRIVER，UMD）由 C++ 编写，负责解析 Loadable 文件，分配内存，并将上下文封装成一个 Task，最后发送给底层的 KMD 程序执行。
- 内核态驱动程序（e KERNEL MODE DRIVER）由 C 语言编写，是 NVDLA 固件的驱动程序，他是真正与 NVDLA 进行交互的部分。

软件栈设计的详细内容，我们将在软件实现这一章详细讨论。

3.2 验证平台

3.3 NVDLA 与 ARM 互联

第四章　加速器详细设计

本章分两个部分对加速器的设计过程进行说明，第一部分说明了 SIMD 架构的加速器设计的详细过程，第二部分说明了加速器的优化思路及优化过程。

4.1　SIMD 架构的卷积神经网络加速器设计

从对 Darknet 的网络结构分析可知，YOLO V2 算法除了数据输入层之外，各层都是将上一层的输出数据作为本层的输入数据，层间的数据关联性较大，需要串行处理，而层内的数据关联性较小，可以使用 FPGA 进行并行加速。本文设计的加速器运行过程如图 4-1所示，CNN 加速器每次从 DDR 内存中读取多块数据写入缓存，CNN 加速器内部的运算单元从缓存获取数据，根据相应的配置参数进行判断，将数据分别交给 CONV、POLL、REORG 模块进行运算，运算的结果数据写入 CNN 加速器的缓存中，最后 CNN 加速器再将计算结果写回 DDR 内存中，由此完成了一整个计算流程。

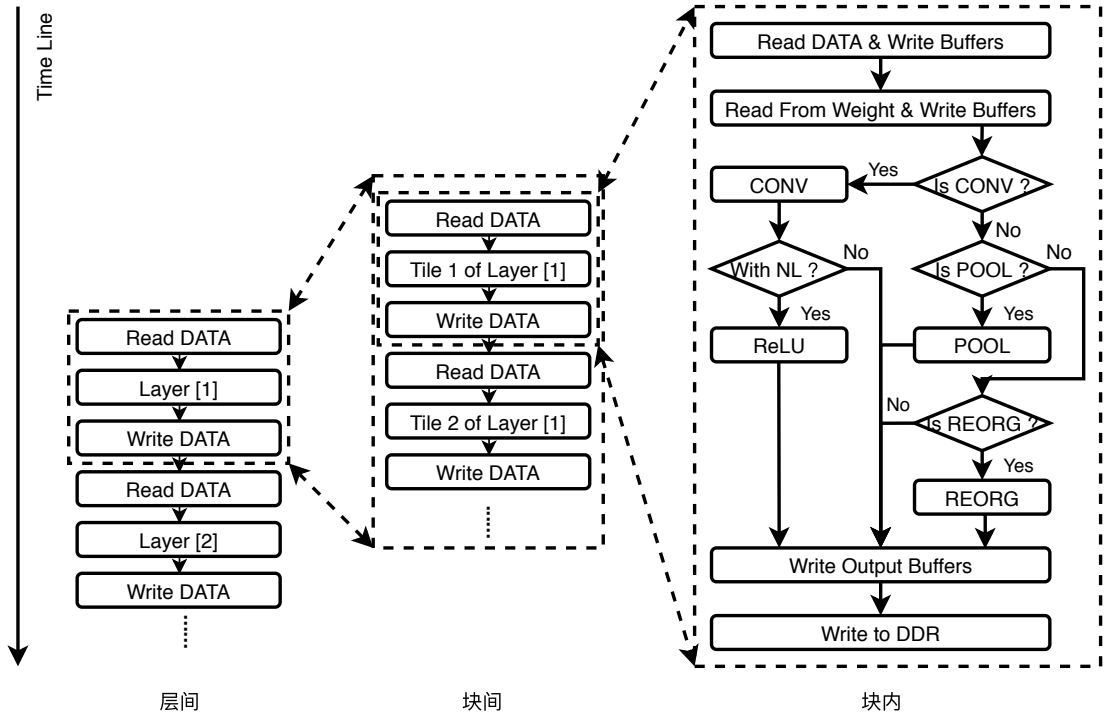


图 4-1 CNN 加速器运行流程

CNN 加速器结构如图 4-2所示，包含一个 AXI Lite 接口，以及 N 个 AXI Full 接口。其中，AXI Lite 连接在 RISC-V 处理器上，通过此接口，RISC-V 处理器可以对其进行功能的控制，选择 CNN 加速器的计算单元；一个 AXI Full 接口连接在 DDR 内存上，用于获取

权重参数；另外的 N-1 个 AXI Full 连接在 DDR 内存上，用来写入或者读取特征图数据。CNN 加速器内部有四个计算单元，分别是 CONV、POLL、REORG 和 ReLU，这四个模块从 CNN 加速器的缓存中获取数据，通过 CNN 加速器中的寄存器控制，分别完成卷积、池化、重排序和激活的计算任务。Data Gather 模块负责生成数据读取地址，并且控制 AXI Full 接口从 DDR 内存中获取数据，将其写入 Input Buffer 模块中进行缓存。Data Scatter 模块负责生成写回地址，并且控制 AXI Full 接口，将 Output Buffer 中的数据写回 DDR 内存中。Weight Buffer 模块缓存负责权重数据。

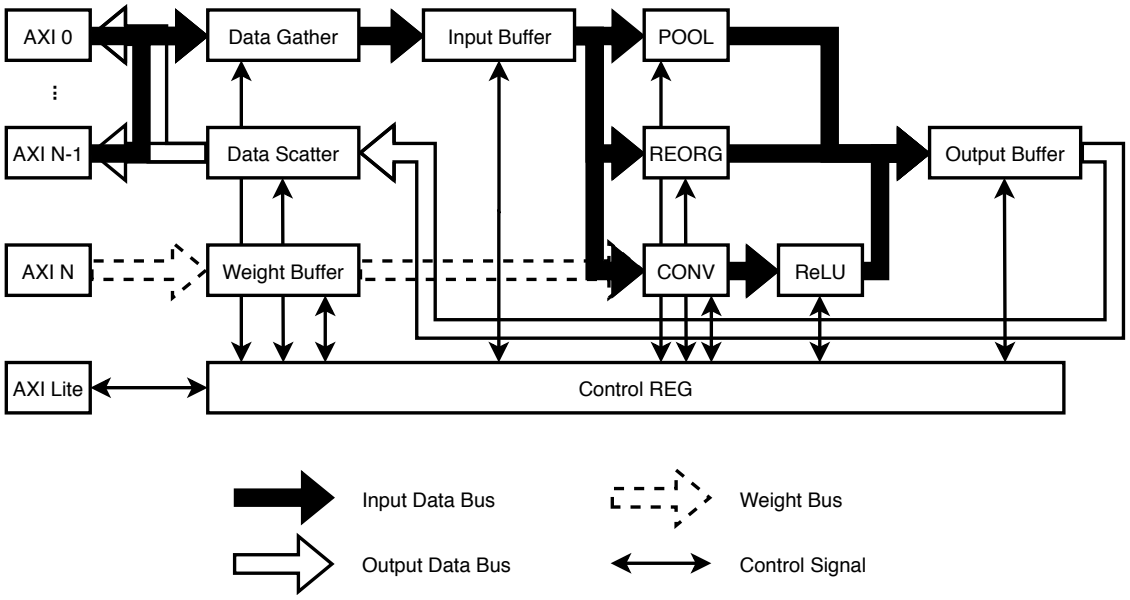


图 4-2 CNN 加速器结构

负责读写特征图数据的 AXI Full 接口总数量可以由公式 4-1计算获得：

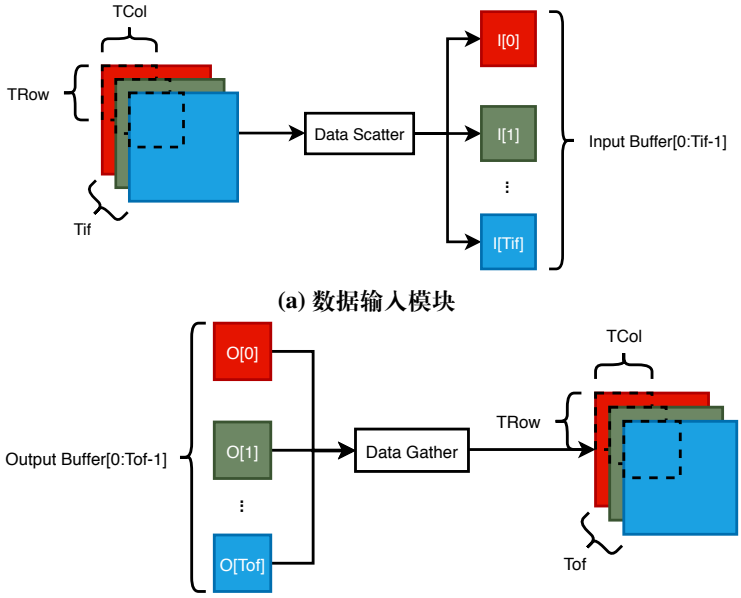
$$N_{Channel} = Max(N_{in}, N_{out}) \tag{4-1}$$

其中 $N_{Channel}$ 表示总的负责读写特征图数据的 AXI Full 接口数量，N_{in} 表示读取特征图需要的 AXI Full 接口的数量，N_{out} 表示输出特征图数据需要的 AXI Full 接口数量。由于 AXI Full 协议是一种全双工的总线协议，所以一条总线上读写操作可以同时进行，互不影响，所以实际上负责读写特征图数据的 AXI Full 接口总数量由读通道和写通道中多的一方决定。

4.1.1　数据输入输出模块

数据输入模块的硬件架构如图 4-3(a) 所示，Data Scatter 模块从 DDR 内存中获取 Tif 张 $Tix \times Tiy$ 大小的特征图数据，然后依次分发到 CNN 加速器的 Tif 个缓存模块中。数

据输出模块的硬件架构如图 4-3(b) 所示，Data Gather 模块将 CNN 加速器中 Tof 个缓存模块中的数据依次写入到 DDR 内存中。

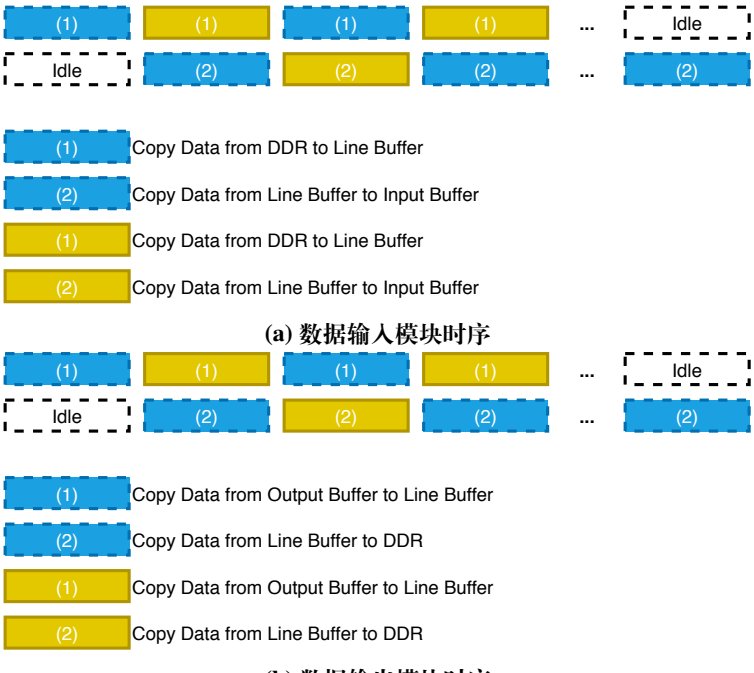


(a) 数据输入模块

(b) 数据输出模块

图 4-3 数据输入输出模块硬件架构

为了保证数据的读写效率，使数据的输入输出部分不至于成为整个加速器的瓶颈，数据输入输出模块中使用双缓存的方案交替读取或者写入数据。



(a) 数据输入模块时序

(b) 数据输出模块时序

图 4-4 数据输入输出模块双缓存时序

数据输入模块的时序如图 4-4(a) 所示，在流水线中，两个缓存模块交替完成从 DDR 内存中读取数据和将数据分发到 CNN 加速器内部的 Input Buffer 上的任务，考虑到数据读

取和数据分发所产生的时延是相同的，因此在理想情况下，使用双缓存模式后，数据输入模块产生的时延仅为原先单缓存模式下的 $\frac{1}{2}$。

数据输出模块的时序如图 4-4(b) 所示，在流水线中，两个缓存模块交替完成从 CNN 加速器内的 Output Buffer 上获取数据和将数据写入 DDR 内存的任务。和数据输入模块的时序相似，在理想情况下，使用双缓存模式后，数据输出模块产生的时延仅为原先单缓存模式下的 $\frac{1}{2}$。

4.1.2　CONV 卷积模块

由于卷积层有大量的乘加运算，占据了整个 CNN 计算的大部分时间，因此在硬件加速中，本设计将卷积层进行不同维度的展开，通过设计相应的并行乘加运算单元，来达到加速卷积计算的效果。基本的维度展开主要分以下 4 种：

1. 卷积核维度的展开

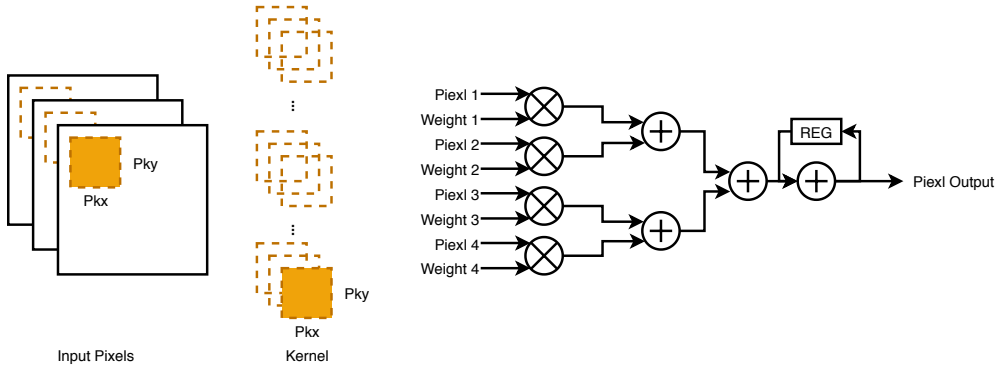


图 4-5 卷积核维度展开硬件架构

如图 4-5所示，展开卷积核的宽度 Pkx 和卷积核的高度 Pky。在每个时钟周期里，同一输入特征图的相邻的 $Pkx \times Pky$ 个相邻的像素与卷积核对应位置的权重数据进行并行的乘法运算，乘法运算的结果通过一个深度为 $log_2 Pkx \times Pky$ 的加法树相加得到中间结果，累加的结果存放在一个寄存器中。该维度展开卷积运算，消耗的资源为 $Pkx \times Pky$ 个乘法器、一个深度为 $log_2 Pkx \times Pky$ 的加法树和一个累加器。

2. 输入特征图数维度的展开

如图 4-6所示，展开 Pif 个输入特征图。在每个时钟周期里，同时从 Pif 个输入特征图的同一位置读取一个像素与卷积核对应位置的权重数据进行乘法运算，乘法运算的结果通过一个深度为 $log_2 Pif$ 的加法树相加得到中间结果，累加的结果存放在一个寄存器中。该维度展开卷积运算，消耗的资源为 Pif 个乘法器、一个深度为 $log_2 Pif$ 的加法树和一个累加器。

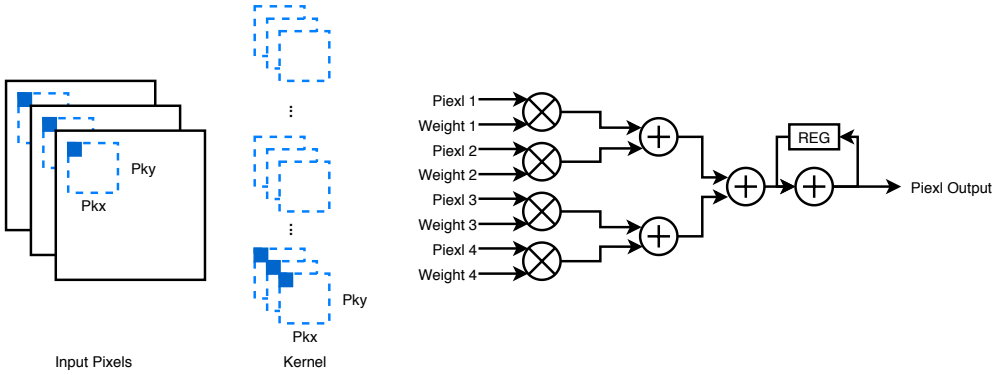


图 4-6 输入特征图数维度展开硬件架构

3. 输出特征图维度的展开

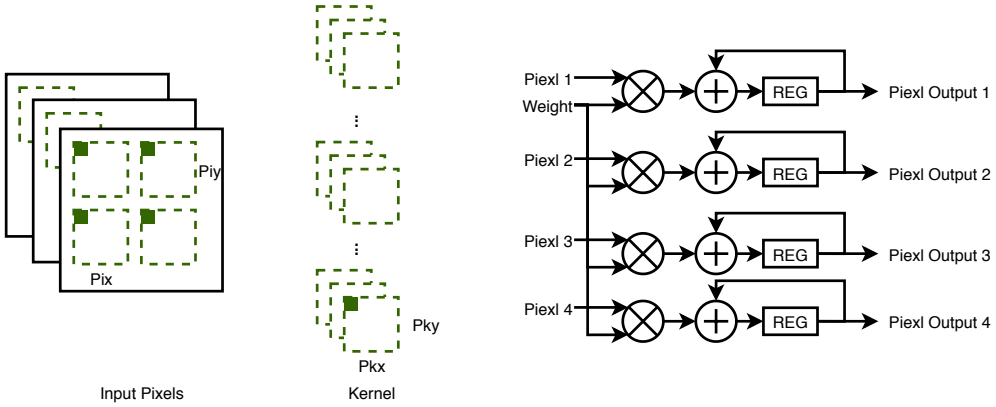


图 4-7 输出特征图维度展开硬件架构

如图 4-7所示，展开输出特征图的宽度 Pix 和卷积核的高度 Piy。在每个时钟周期里，计算同一输出特征图的相邻的 $Pix \times Piy$ 个相邻的像素的值。该过程与卷积核维度的展开的运算类似。

4. 输出特征图数维度的展开

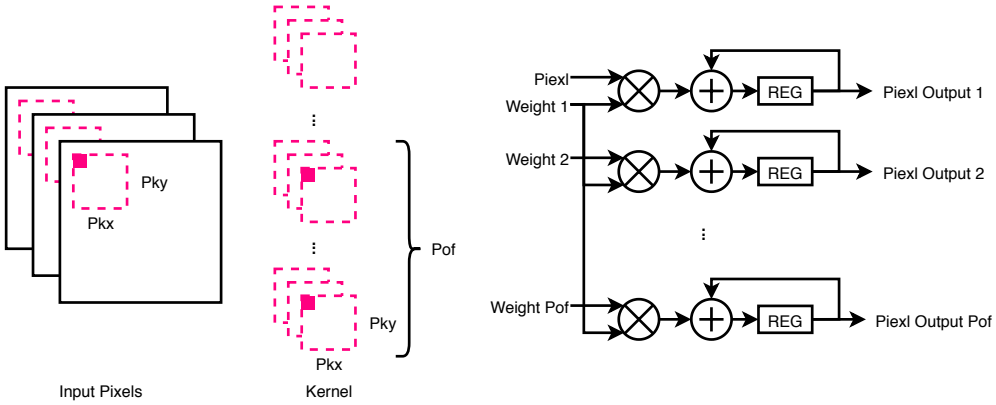


图 4-8 输出特征图数维度展开硬件架构

如图 4-8所示，展开 Pof 个输出特征图。在每个时钟周期里，同时将同一个输入特征

图的一个像素与 Pof 个卷积核对应位置的权重数据进行乘法运算，乘法运算的结果通过一个深度为 $log_2 Pof$ 的加法树相加得到中间结果，累加的结果存放在一个寄存器中。该维度展开卷积运算，消耗的资源为 Pof 个乘法器、一个深度为 $log_2 Pof$ 的加法树和一个累加器。

5. 卷积展开设计

由于在 FPGA 中，图像数据是以数据流的形式进行传输的，即在流水线中，每个时钟周期传输一个像素的数据，如果想一个时钟周期同时访问 N 个像素的数据，就需要 N 条数据总线。但由于在本设计中，特征图数据是存储在外置的 DDR 中的，同时访问 N 个像素的数据违背了 DDR 的工作原理，因此任然需要排队等待。所以这种需要同时访问多个像素数据的优化方式在访存部分会形成瓶颈，无法达到有效的加速效果。而展开方法（1）卷积核维度的展开与展开方法（3）输出特征图维度的展开正是有这种多个像素同时读取的需求，因此在本设计中使用的卷积展开方法主要是展开方法（2）输入特征图数维度的展开与展开方法（4）输出特征图数维度展开的结合使用。

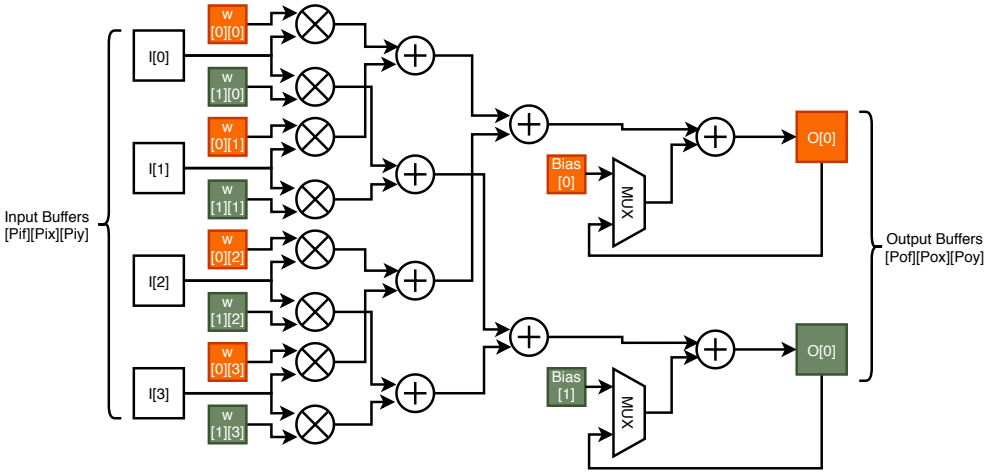


图 4-9 卷积模块硬件架构

如图 4-9所示，通过对输入特征图数维度于输出特征图数维度进行二维展开。以输入特征图数 $Pif = 4$ 和输出特征图数 $Pof = 2$ 为例，展开度为 $Pif \times Pof = 8$。在一个时钟周期内，同时从 Pif 个输入特征图的同一位置读取一个像素分别和 Pof 个卷积核对应的权重数据相乘，再将乘法的结果通过一个深度为 $log_2 Pif \times Pof$ 的加法树相加，加法树输出的结果在通过一个累加器将局部运算的结果进行累加，最终的结果存放在 CNN 加速器的寄存器中。该展开方式消耗 $Pif \times Pof$ 个乘法器，一个深度为 $log_2 Pif \times Pof$ 的加法树和一个累加器。

在内存访问速度不是瓶颈的情况下，即每个时钟周期可以获得一个输入特征图的像

素，该展开方式进行卷积计算产生的时延如公式 4-2所示。式中 $Latency_{CONV}$ 为一层卷积计算的总时延；$Const_{CONV}$ 为一个常量，代表了数据充满流水线所需要的时间；Nky 与 Nkx 分别表示输入特征图的高度和宽度；Toy 和 Tox 表示卷积核的高度和宽度。

$$Latency_{CONV} = (Const_{CONV} + Nky \times Nkx \times Toy \times Tox)/Freq \tag{4-2}$$

可以看到，在该展开方式下，卷积运算的总时延与单层输入特征图，单层输出特征图的时延相当。因此在该展开方式下，卷积计算的总时延仅为原本的 $\frac{1}{Pof \times Pif}$，因此该展开方式可以有效的对卷积运算进行加速。

4.1.3 POOL 池化模块

由于图像中相邻位置的像素有着相似的值，卷积层的输出特征图同样也会在相邻区域有值相似的像素，因此去除这些相似值的像素并不会使特征图损失很多信息，并且通过去除这些相似的像素可以极大的减少参数数量，从而降低后续卷积运算的计算量，池化层就是完成了这个任务。池化运算与卷积运算类似，YOLO V2 算法中所有的池化层都是大小为 2×2，步长 $S=2$ 的最大池化。因此池化加速模块的设计可以参考卷积模块的设计，不同的是池化模块只需要对单一的输入特征图进行运算，并且运算单元由原先卷积模块中的乘加运算单元换成了比较运算单元。

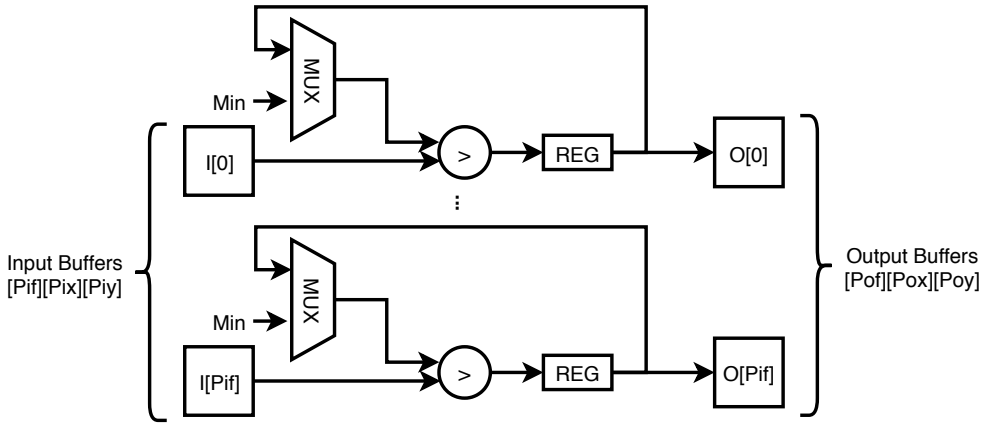


图 4-10 池化模块硬件架构

如图 4-10 所示，池化模块在运算过程中，同时从 Pif 个输入特征图的同一位置获取一个像素，并与当前的最大值进行比较，在比较完成后，将得到的最大值写入输出缓存中。该结构的池化模块消耗 Pif 个比较器和 Pif 个寄存器。该池化运算加速模块的计算时延

如公式 4-3所示：

$$Latency_{POOL} = (Const_{POOL} + Nky \times Nkx \times Toy \times Tox)/Freq \tag{4-3}$$

式中 $Latency_{POOL}$ 为一层池化计算的总时延；$Const_{POOL}$ 为一个常量，代表了数据充满流水线所需要的时间；Nky 与 Nkx 分别表示输入特征图的高度和宽度；Toy 和 Tox 表示池化核的高度和宽度。

4.1.4　REORG 重排序模块

重排序层的作用主要是对输入特征图进行一个拆分，将输入特征图相邻位置的像素分别输出到不同的特征图上。该过程的运算与池化模块的运算过程相似，不同的是池化模块将一个输入特征图输出到一个输出特征图，而重排序模块是将一个输入特征图输出到多个输出特征图。重排序模块的硬件架构如图 4-11所示。

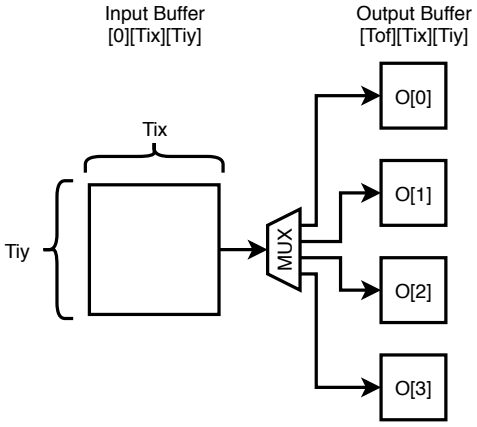


图 4-11 重排序模块硬件架构

由于 YOLO V2 中的重排序层是 $Nky = Nkx = S = 2$ 的结构，因此重排序层需要 1 个输入缓存模块和 4 个输出缓存模块，重排序模块的计算时延如公式 4-4所示：

$$Latency_{REORG} = (Const_{REORG} + Nky \times Nkx)/Freq \tag{4-4}$$

式中 $Latency_{REORG}$ 为一层重排序计算的总时延；$Const_{REORG}$ 为一个常量，代表了数据充满流水线所需要的时间；Nky 与 Nkx 分别表示输入特征图的高度和宽度。

4.2 CNN 加速器优化

4.2.1 乒乓缓冲优化

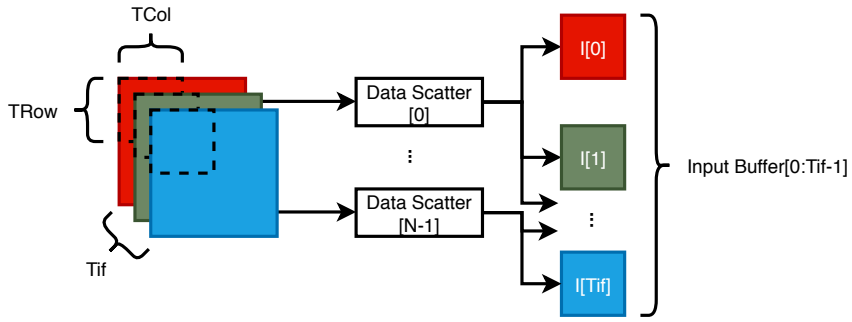
乒乓缓冲在 FPGA 设计中是一种典型的以面积换时间的做法，在本次设计的 CNN 加速器中，我们在 CNN 加速器的数据输入端口和数据输出端口使用了乒乓缓冲的结构进行优化。因此该 CNN 加速器在流水线模式下运行时可以使数据传输产生的时延与数据运算产生的时延相重叠，加速器运算的总时延的计算如公式 4-5所示：

$$Latency_{ALL} = Max(Latency_{load}, Latency_{compute}, Latency_{store}) \tag{4-5}$$

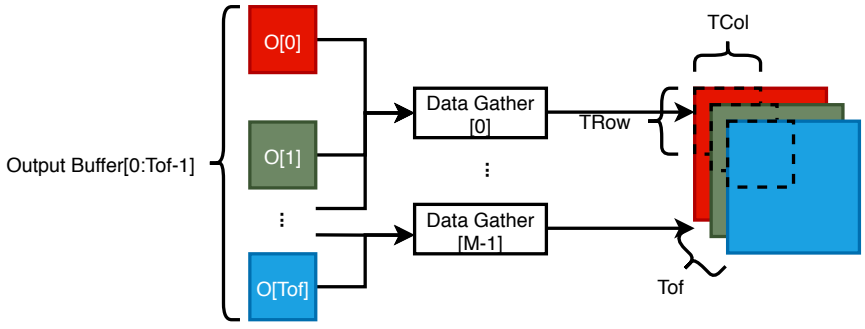
式中 $Latency_{ALL}$、$Latency_{load}$、$Latency_{compute}$、$Latency_{store}$ 分别代表 CNN 加速器运算的总时延、从 DDR 内存中读取数据所产生的时延、加速器计算所产生的时延、将数据写回 DDR 内存中的所产生的时延。

4.2.2 多数据通道传输优化

由于 DDR 内存的带宽远大于一条 AXI4 总线的带宽，为了充分利用 DDR 内存的带宽，减少数据传输过程中产生的时延，在本设计中使用了多数据通道传输的优化方式。



(a) 数据输入模块



(b) 数据输出模块

图 4-12 多数据通道传输优化硬件架构

数据输入模块的多数据通道传输优化硬件架构如图 4-12(a) 所示，模块内部有多个独立的 Data Scatter 模块，每个模块有一条独立的 AXI4 总线，每个 Data Scatter 模块负责一

部分数据缓存模块，因此多个 Data Scatter 模块可以并行地从 DDR 内存中读取特征图数据，并将数据分发到其负责的缓存模块上，从而提高了 DDR 内存的带宽利用率。

数据输出模块的多数据通道传输优化硬件架构如图 4-12(b) 所示，与数据输入模块的优化思路相似，在模块内部使用多个独立的 Data Gather 模块，每个模块有一条独立的 AXI4 总线，因此多个 Data Gather 模块可以并行的将其负责的一部分缓存模块中的数据写入到 DDR 内存中。

4.3　本章小结

本章对 CNN 加速器进行了详细的设计及优化。具体说明了四种卷积循环的展开模式以及各模式产生的时延和局限性，最终选取了输入特征图数维度的展开和输出特征图数维度的展开相结合的模式，从而在硬件资源的允许范围内，尽可能多的减少卷积运算产生的时延。同时由于硬件资源的限制，无法将 YOLO V2 模型的所有层同时进行硬件加速，因此采取了每次加速一层中一部分的做法，该方法类似于 FPGA 中状态机的思路，是一种用时间换面积的做法。由于该方法涉及大量的内存读写，为了使内存读写的时延不成为 CNN 加速器的瓶颈，加速器内部使用了双缓存和多数据通道的优化方法，保证了在流水线运行模式下，内存的读写时间可以被卷积运算的时间所掩盖，从而尽量减少内存的读写所产生的时延对加速器性能造成的影响。

第五章　RISC-V SOC 设计

由于神经网络中的卷积计算、池化计算及全连接计算等都在加速器中完成，而网络中的图像预处理、softmax 层计算等耗时较短的操作则由 RISC-V 处理器来完成。为了让 CNN 加速器和 RISC-V 处理器协同工作，在本章中我们设计了一个通用的 SOC，将 CNN 加速器作为 RISC-V 处理器的一个外设，由 RSIC-V 处理器进行流程控制和图像预处理、softmax 层计算等工作，CNN 加速器负责主要的卷积神经网络计算任务。为了提高设计效率，本设计中将 RISC-V 处理器和神经网络加速器封装成带有 AXI4 总线接口的 IP，通过 AXI4 总线与 UART、GPIO、定时器等外设连接。下面详细介绍 SOC 的设计过程。

5.1　SOC 整体设计

由于目前 RISC-V 基金会还没有发布标准的 RISC-V 调试架构文档[?]，因此目前开源的几款 RISC-V 处理器大多不带调试机制，这就给软件调试带来了不便之处，为此这里参考了 PYNQ MicroBlaze Subsystem 的设计[?]，在 ZYNQ 芯片上运行 Linux 系统与 RISC-V 处理器共享内存来完成程序的下载过程。

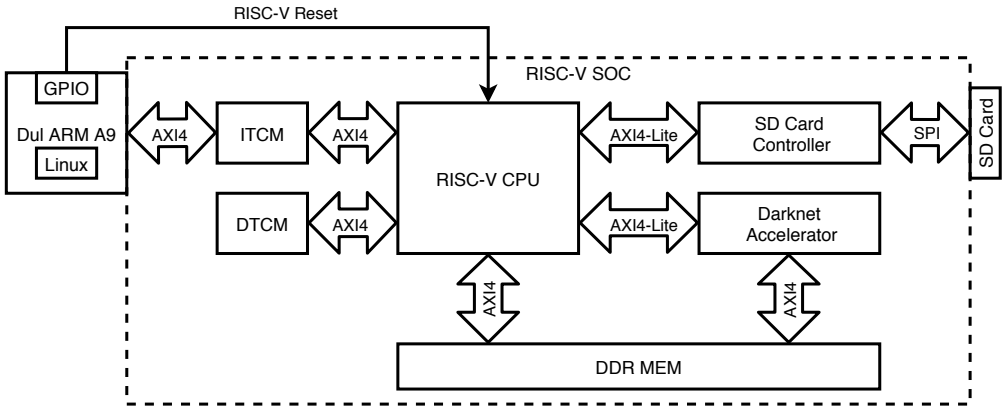


图 5-1 RISC-V SOC 架构

SOC 整体的结构设计如图 5-1所示。在本设计中，将 SOC 部分在 ZYNQ 的 PL 部分实现。RISC-V 的 ITCM 使用一个双端口 Block RAM 实现，ITCM 的一个端口与 RISC-V 处理器连接，其另一个端口通过 AXI4 总线与 ZYNQ 上的 ARM 核相连，ARM 核可以读写这块 ITCM 中的数据。在 ZYNQ 的双核 ARM A9 上运行 linux 系统，在该系统上使用交叉编译工具编译 RISC-V 的软件程序，将编译后的二进制文件放入 RISC-V 处理器的 ITCM 中，再复位 RISC-V 处理器，便可以使 RISC-V 处理器执行相应的程序。

5.2　SOC 硬件详细设计

本 SOC 主要包含 RISC-V 处理器，集成 ROM、RAM，包含了基本的外设（包括 GPIO、UART 等）以及神经网络加速器，各模块之间通过 AXI4 总线连接。

5.2.1　RISC-V 处理器接口描述

Ibex 处理器的结构如图 5-2所示，Ibex 处理器的接口主要包含 6 个部分：

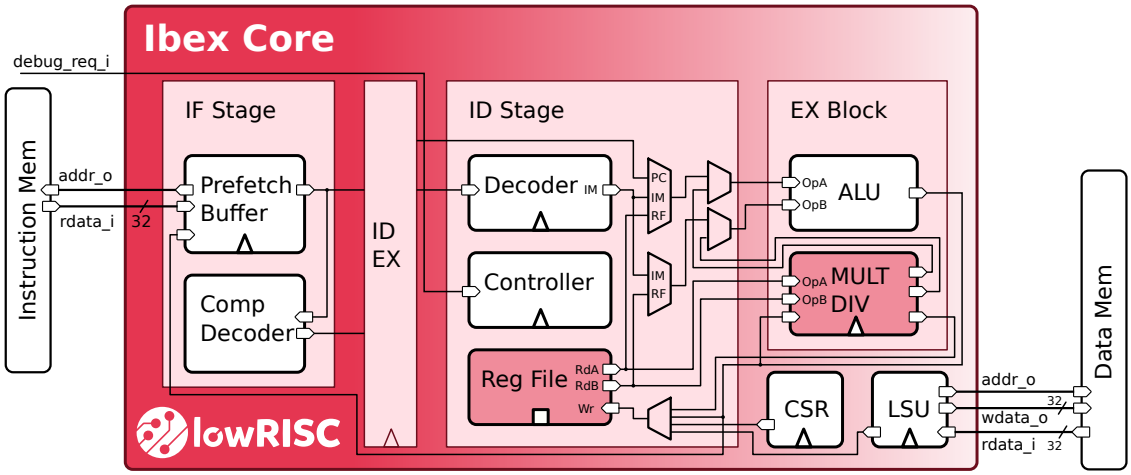


图 5-2 Ibex 处理器结构

1. 时钟和复位接口；
2. 控制接口；
3. 指令私有总线接口；
4. 数据私有总线接口；
5. 中断接口；
6. Debug 接口。

5.2.2　SOC 片上存储资源

本 SOC 中的片上存储器资源分为 ITCM 和 DTCM，其结构如图 5-3 所示。其中 ITCM（Instruction Tightly Coupled Memory）为 RISC-V 处理器内核私有的指令寄存器，其大小可以配置，默认设置为 64Kb；默认的起始地址是 0x8000_0000；由于 ITCM 是用来存放指令的，所以 RISC-V 处理器在正常运行时，这块区域是只读的，为了实现程序的掉电保存，需要将程序存放在外置的非易失性存储器上，在处理器复位后，使用 Load 和 Store 指令将程序搬运到 ITCM 中，这个阶段 ITCM 为可读写状态。因此 ITCM 使用一个双端口 Block RAM 实现，其有两个读写接口，一个读写接口直连处理器的指令总线接口，另一个接口通过一个仲裁器分别连接了处理器的数据总线接口和 Debug 模块的数据接口。

DTCM（Data Tightly Coupled Memory）为 RISC-V 处理器内核私有的数据存储器，其大小可以配置，默认为 64Kb；默认的起始地址是 0x9000_0000；DTCM 用来存放程序运行中的数据，相当于处理器的内存，因此在通常状态下，DTCM 都是可读写的，所以 DTCM 使用一个单端口 RAM 实现，其接口直接连接在处理器的数据总线接口上。

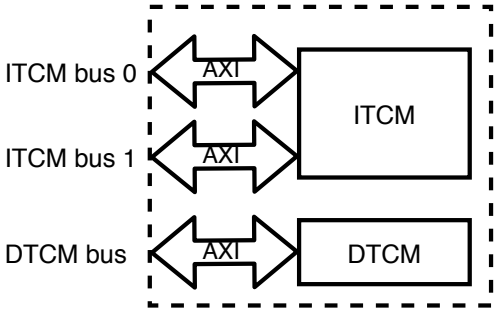


图 5-3 片上存储器资源结构框图

5.2.3 SOC 外设资源

1. UART

本 SOC 中实现了一个 URAT 外设, 采用 AXI Lite 总线接口，结构如图 5-4 所示。该 UART 在接收端使用 16 倍波特率的采样频率进行采样；数据格式为 8 位数据，没有奇偶校验位，1 位起始位，1 位停止位；波特率由内部寄存器控制，支持软件调节；支持全双工的接收和发送数据，内部带有接收缓存和发送缓存；中断产生的阈值由内部寄存器控制，支持软件调节。

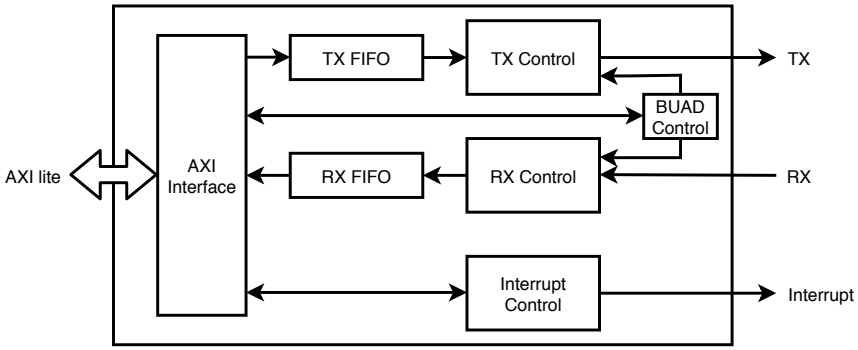


图 5-4 UART 结构框图

2. GPIO

本 SOC 中实现了一个双通道 32 位 GPIO，采用 AXI Lite 总线接口，其结构如图 5-5 所示。每个通道有 32 位的双向 I/O 口，每位 I/O 口的方向由内部寄存器控制，支持软件调节；内部带有一个中断控制器，在 I/O 口为输入时，可以产生中断信号。

3. RTC

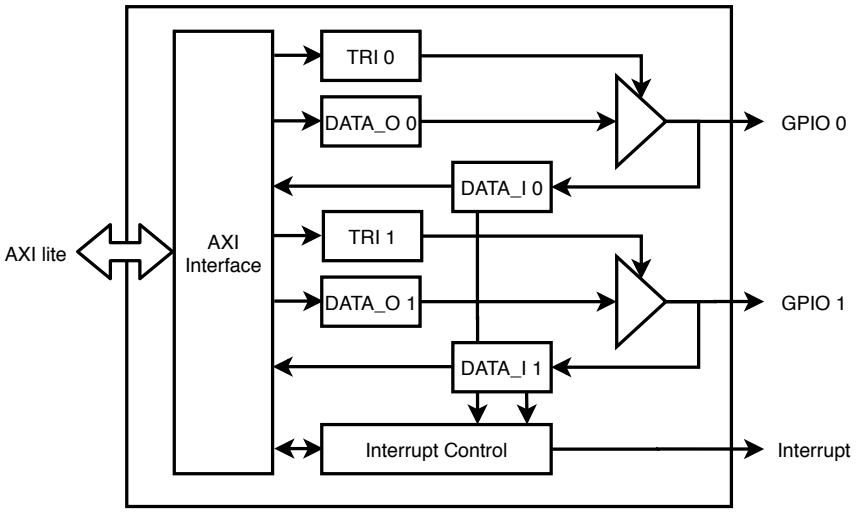


图 5-5 GPIO 结构框图

RTC（Real-Time Clock）是 MCU 中常用的模块，用于给程序运行提供精确计时的功能。本 SOC 中设计了一个 RTC 模块，其结构如图 5-6 所示。本设计中的 RTC 包含了两个独立的 32 位计数器，其触发信号为 CPU 的时钟，这两个计数器可以在其内部寄存器的控制下级联成一个 64 位的计数器。计数器的初始值可由内部寄存器控制，计数器向上计数，加满之后再加一，就会产生溢出，从而通知中断控制器产生中断。

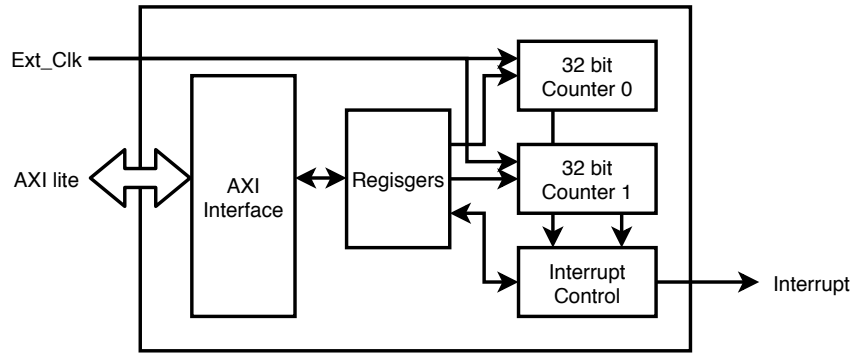


图 5-6 RTC 结构框图

5.3　本章小结

本章详细介绍了 SOC 的设计过程，采用了一种以 ZYNQ 架构为基础的通用 RISC-V SOC 设计，方便对于没有调试接口的 RISC-V 处理器进行程序调试和程序下载运行的操作，同时介绍了部分外设的设计过程。

第六章　测试与分析

6.1　CNN 加速器测试实验环境

本设计中使用的 CNN 模型是 YOLO V2 Tiny[?]；数据集为 VOC 2007 数据集[?]，该数据集包含了 20 个类；FPGA 平台为 PYNQ-Z2 开发板（Dual ARM A9 + FPGA）。

硬件设计工具使用如下：CNN 加速器使用 Vivado HLS 2018.2 工具设计，硬件工程使用 Vivado 2018.2 工具设计；软件设计工具如下：使用 riscv32-unknown-elf-gcc 交叉编译工具对 RISC-V 处理器的程序进行编译。

6.2　对比测试环境

为了直观了解 CNN 加速器的加速效果，设计了 3 组对照实验进行对比分析，分别如下：

X86 架构 CPU 平台：CPU 为 Intel i7-9700K，8 核心，频率为 4.9 GHz；32GB 运行内存；

嵌入式 CPU 平台：树莓派 3B，CPU 为四核 ARM Cortex-A53，频率为 1.2 GHz；1 GB 的运行内存；

嵌入式 GPU 平台：NVIDIA Jetson Nano，四核 ARM Cortex-A57 MPCore 处理器，频率为 1.43GHz；GPU 为 Maxwell 架构，配备 128 个 CUDA 核心；4GB 运行内存。

6.3　CNN 加速器资源消耗评估

整个异构运算系统的设计连接如图 6-1所示，其中 YOLOV2_FPGA 为基于 YOLO V2 算法的设计 CNN 加速器 IP 核。RISCV_Processor 为 RISC-V 处理器 IP 核。其余部分是 SOC 中的一些基本外设。YOLO V2 算法加速器有 1 个 AXI-Lite Slave 接口，连接在 RISC-V 处理器的数据总线接口上，负责状态寄存器的控制；还有 5 个 AXI Master 接口分别与 DDR 的 HP[0:3] 接口连接，其中有两个 AXI Master 接口连接在 HP[3] 上。CNN 加速器及其数据通路运行在 150 MHz 的时钟频率下，SOC 中其他部分运行在 100 MHz 的时钟频率下。

结合设计参数，计算得到 CNN 加速器的预估资源消耗；使用 Vivado 工具综合分别在没有加入 CNN 加速器的情况下和加入了 CNN 加速器的情况下综合硬件工程，在 Vivado 综合布线完成，根据运行报告，将两次消耗的资源相减，获得 CNN 加速器的实际资源消

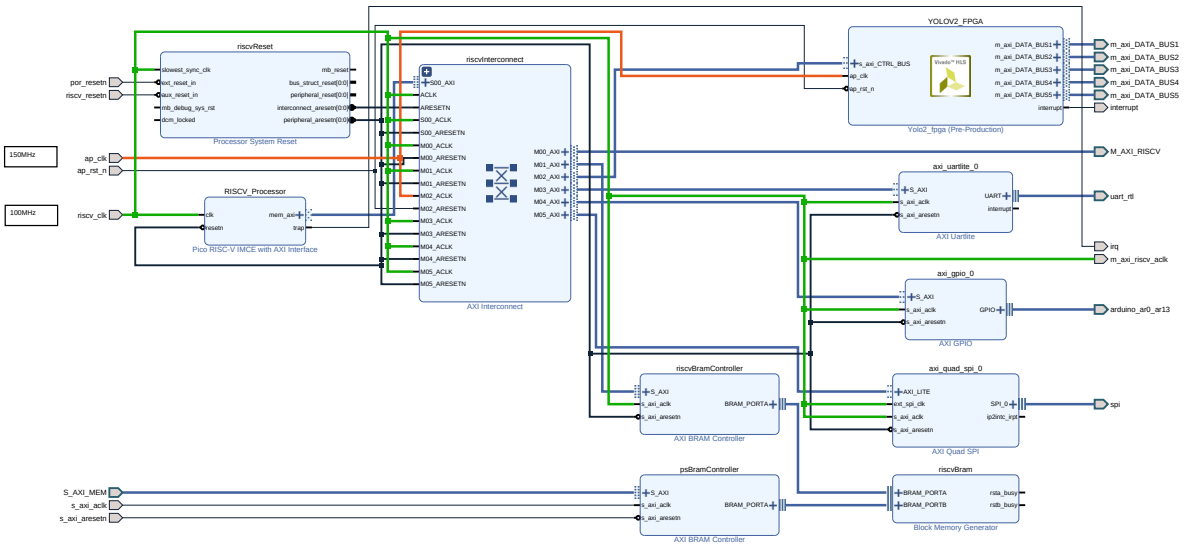


图 6-1 异构运算系统

耗。预估资源消耗与实际资源消耗对比如表 6-1所示：

表 6-1 预估资源消耗与实际资源消耗对比

	DSP	BRAM 36Kb	LUT	FF	Freq
Estimate	128(58%)	85(61%)	20736(39%)	-	150MHz
Actual	153(70%)	87(62%)	37446(70%)	44236(41%)	150MHz

DSP 资源主要用于卷积计算模块中的加法器和乘法器的实现，实际消耗的 DSP 资源和预估的相差比较大。由于 HLS 在综合的过程中会考虑到时序状况，当运行频率提高后，HLS 工具会使用更多的 DSP 资源来优化时序，因此实际设计中多出的 DSP 资源消耗主要是用于时序优化的部分。

BRAM 资源主要用于实现 CNN 加速器的缓存以及 AXI 总线接口部分的缓存，实际的 BRAM 资源消耗与预估资源消耗较为接近。

预估资源消耗中的 LUT 是权重数据的缓存所使用的数量，由于 BRAM 的最小值为 18Kb，而权重数据的数据量过小，使用 BRAM 进行缓存较为浪费，因此在本设计中使用 LUT 和 FF 来缓存权重数据。实际设计 LUT 还要用来实现加速器中的其他模块，难以预估这部分的使用量，因此才产生了 LUT 资源的预估使用量与实际使用量相差较大的现象。

6.4　CNN 加速器性能评估

分别测试本设计在 FPGA 平台上实现的 CNN 加速器的性能、CPU 平台 CNN 运算的性能和嵌入式 GPU 平台的上 CNN 运算的性能，进行对比分析。使用单位时间内处理的图片张数（FPS）作为性能对比依据；分析中的功耗为对应芯片的最大功耗。

6.4.1 与 CPU 的性能和能效对比

本文设计的加速器与 CPU 平台的对比效果如表 6-2所示，与 i7-9700K 相比，本文设计的异构计算系统的计算速度是其 6.3 倍；与四核 ARM A53 相比，本文设计的异构计算系统的速度是其 361.1 倍。由对比测试的结果分析可知，X86 架构的主机计算平台计算性能强，计算产生的时延很短，但是由于其功耗过高，导致整体上能效低下；而嵌入式 CPU 计算平台的计算性能较差，即使功耗较低，其处理图片的速度也是难以接受的；而本文设计的异构计算系统的计算速度优于 X86 架构的主机计算平台，且功耗和嵌入式 CPU 平台相近。因此相对于 CPU，本异构计算系统无论在功耗还是计算速度上都有较大的优势。

表 6-2 与 CPU 对比

Device	CPU i7-9700K	Raspberry Pi 3B	CPU+FPGA
Technology(nm)	14	40	28
Clock(Hz)	4.9G	1.2G	666.67M + 150M
Memory(MB)	32768	1024	512
Max System Power(W)	95	5.0	5.0
Latency per img(seconds)	6.82	390	1.08
FPS($seconds^{-1}$)	0.146	0.00256	0.926
Power Efficiency(FPS/W)	1.54×10^{-3}	0.522×10^{-3}	185×10^{-3}

在准确度方面，本文设计的异构计算系统的使用是动态定点 16 位数据量化得到的模型，在准确度上与原模型相近，在 VOC 2007 数据集上 mAP（mean Average Precision）达到了 49.2%。

6.4.2 与嵌入式 GPU 性能与能效对比

如表 6-3所示，与目前常用的的嵌入式 GPU 计算平台相比，由于用于测试的异构计算系统本身的资源较少，导致异构计算系统的算力较低，同时由于工艺制程相差也较大，导致功耗方面也没有优势。文献[?]中提到，在特征提取层使用二值化精度的权重数据，而之后的层使用 32 位精度的浮点数据，可以使加速器更大维度展开，从而在性能上取得更好的效果。

表 6-3 与 GPU 对比

Device	Jetson Nano	CPU+FPGA
Technology(nm)	16	28
Clock(Hz)	1.43G	666.67M + 150M
Memory(MB)	4096	512
Max System Power(W)	10.0	5.0
Latency per img(seconds)	0.262	1.08
FPS($seconds^{-1}$)	3.80	0.926
Power Efficiency(FPS/W)	0.380	0.185

6.5　本章小结

本章主要计算分析了 CNN 加速器的资源消耗，保证了在使用 HLS 工具进行设计的过程中没有出现重大的资源浪费。根据常见的运算平台，设计了 3 组典型的对照实验环境，将本设计中的异构计算系统分别与其进行对比，量化分析了该异构计算系统的性能和功耗参数。

第七章　总结与展望

目标检测算法在嵌入式领域有着广泛的应用，比如自动驾驶、无人机图像识别等，这些应用场景不仅对准确度和速度有要求，对功耗也有限制。传统的目标检测算法通常运行在高功耗的 GPU 上，并不适合应用在这些低成本低功耗的嵌入式场景中。本文设计了一种运行在 FPGA 上的异构卷积神经网络算法加速系统，将 CNN 网络进行部分展开，展开程度可以根据具体 FPGA 芯片资源的丰富程度灵活调整，同时使用了一个 RISC-V 软核来完成一些不适合在 FPGA 上完成的计算任务。从而实现了在保证准确度的情况下，使用低成本低功耗的嵌入式平台进行目标检测。

本课题对使用 FPGA 进行卷积神经网络运算加速做了一定的研究，但由于时间的限制，任然有许多可以进一步研究的地方：

1. 优化 RISC-V SOC 的设计，将加速器的调用设计成专用指令的形式，进一步提升加速器使用的通用性和便利性；

2. 使用多 FPGA 构建大规模协同运行的 CNN 加速网络，使得 CNN 加速的效果不在受限于单个 FPGA 的资源，从而实现一些更复杂的模型；

3. 使用 FPGA 的可重构技术，将 CNN 网络进行分片，相当于对局部的 CNN 网络进行了更大维度的展开，从而提升 CNN 网络的并行性，进一步优化加速器的时延。

致　谢

时光易逝，转眼间四年的本科求知时光就要结束了。随着毕业论文的定稿，电子信息专业的学习也将画上句号，我的本科生涯也要结束了，行文至此，心中满是感恩。

饮其流者怀其源,学其成时念吾师。感谢一直以来在学业方面帮助我的包亚萍老师，大学期间，包老师以她渊博的专业知识，严谨的治学态度，精益求精的工作作风影响着我。在她身上我学会了如何正确看待自己的成绩，如何树立一个正确价值观，还使我明白了许多待人接物的道理。大学期间一直跟随包老师参与竞赛和课外项目，她让我知道，竞赛一方面看专业水平，另一方面是对心智的考验。更多是一种锲而不舍的精神，一种不达目的不罢休的韧劲。同时，感谢大学期间所有的授课老师，为我们传授知识，排忧解惑。

学贵得师，亦贵得友。十分幸运能在计算机科学与技术学院遇见如此多优秀的人，在这四年里不断给我鼓励，激励我前进。感谢我志同道合的“学电子不掉发”队友刘佳诚、李杨，在电子设计的征途上是他们一直陪伴我攻克难题；感谢创新工作室的前辈们，尤其是夏一铭、苏晓东学长，他们一直是我追逐的标杆，跟着他们我掌握了更多前沿的技术，也切实的领略到电子设计的魅力；感谢辩论队的小伙伴们，感谢各位的照顾和包容，和大家在一起的欢乐时光我将终身难忘；最后感谢南工这个平台，“明德厚学，沉毅笃行”，工大以其厚德的风骨立于老山脚下，百年学府，业绩辉煌。给我的是无数的机会，教我的是宽广的心胸。

路漫漫其修远兮，吾将上下而求索。我始终相信追求知识，永远都不是徒劳的。未来的研究生涯，我必当博观遵道，推陈出新，在科研路上为自己闯出一片天地。

相逢若有期，珍重千千万。江河湖海，来日再见。