北京大学

本科学位论文

题目：**SoCaffe: 基于 Zynq SoC 平台的高性能深度学习框架**

姓　　名：赵睿哲

学　　号：1200012778

院　　系：信息科学技术学院

专　　业：计算机科学与技术

研究方向：计算机系统结构

导　　师：梁云

2016 年 5 月 27 日

版权声明

摘要

pkuthss 文档模版最常见问题：

`\cite`、`\parencite` 和 `\supercite` 三个命令分别产生未格式化的、带方括号的和上标且带方括号的引用标记：1，[5]、[1,5]。

若要避免章末空白页，请在调用 pkuthss 文档类时加入 openany 选项。

如果编译时不出参考文献，请参考 `texdoc pkuthss`“问题及其解决”一章“其它可能存在的问题”一节中关于 biber 的说明。

关键词：Zynq, 深度学习, SDSoC, FPGA, 高层次综合, Caffe

SoCaffe: High-Performance Deep Learning Framework on Zynq SoC

Ruizhe Zhao (Computer Science and Technology)
Directed by Prof. Yun Liang

ABSTRACT

Test of the English abstract.

KEYWORDS: Zynq, Deep Learning, SDSoC, FPGA, High-Level Synthesis, Caffe

目录

序言

pkuthss 文档模版最常见问题：

\cite、\parencite 和 \supercite 三个命令分别产生未格式化的、带方括号的和上标且带方括号的引用标记：1，[5]、[1,5]。

若要避免章末空白页，请在调用 pkuthss 文档类时加入 openany 选项。

如果编译时不出参考文献，请参考 texdoc pkuthss“问题及其解决”一章“其它可能存在的问题”一节中关于 biber 的说明。

第一章　基本原理

本研究的主要内容是基于 Zynq SoC 平台实现高性能的深度学习框架，因此涉及到三个领域的基本内容：深度学习与神经网络，系统芯片（System-on-Chip，简称 SoC），以及可编程逻辑门阵列（Field Programmable Gate Array，简称 FPGA）和高层次综合（High-Level Synthesis，简称 HLS）。本章分别介绍上述三个领域的基本原理，并选择与本研究密切相关的细分方向进行具体分析。

1.1　深度学习与神经网络

深度学习属于机器学习领域的一个新兴分支。机器学习是实现人工智能的一种方法，简单而言，机器学习往往从大规模的历史数据中学习规律，进而对新的样本进行分类（classification）和预测（prediction）。传统的机器学习算法需要人工指定学习规则，比如从样本中提取哪类特征（feature）、用哪种统计模型进行训练等等。此类方法虽然直观，但需要大量的工程和领域（domain）相关的知识才能达到不错的效果。面对人类认知范围之外的模型时，往往特征提取和模型训练的效果都不尽如人意，有时甚至完全提取不出合适的特征。

深度学习系统的设计是革命性的：特征不再是由专家设计，而是交给通用的算法来自动学习。具体而言，深度学习将特征组织成从低到高的多层堆叠的特征层，使用反向传播算法（Back-propagation，简称 BP）提取特征。深度学习与人脑的工作原理密切相关，堆叠连接的特征层本质上是对神经元的模拟。从这个角度看，深度学习与神经网络是密不可分的：深度学习中的特征层和反向传播算法都来自神经网络。本节接下来首先介绍深度学习的基本概念与基本流程，之后介绍神经网络的几种网络层的特性，其中着重介绍卷积神经网络。

1.1.1　深度学习

2015 年 5 月，深度学习领域的三位代表人物 Yann LeCun①，Yoshua Bengio②与 Geoffrey Hinton③在《自然》杂志发表封面文章《Deep Learning》[4]，介绍深度学习的基本概念与发展现状。文章将深度学习归纳为表征学习（Representation Learning）[2] 的一类方法。表征学习即学习如何从原始数据中提取出可以用来训练与学习的特征的过程，

① Yann LeCun 目前为纽约大学教授，Facebook 人工智能研究院负责人
② Yoshua Bengio 为加拿大蒙特利尔大学教授
③ Geoffrey Hinton 为加拿大多伦多大学教授，并且任职与 Google

深度学习在此之上使用多层表征，每层的特征与上层相比更抽象。此外，每一层都获取上层的输出特征作为输入，进行非线性的转换之后输出到下一层。每层的转换方程都有特定的权值，显著的权值可以放大某些输入特征的“影响”，相应地不显著的权值则“隐藏”了无用的输入特征。深度学习应用经过训练之后，得到的权值可以使得每层提取出最合理的特征。由此看出，深度学习几乎不依赖人类的先验知识，给定特征层的结构和训练数据集之后便可以学习到最合适的特征形式。接下来简单介绍深度学习的训练过程。

所谓训练，即通过历史数据修正深度学习各层中的权值。深度学习的训练有三个基本概念：训练集，损失函数（Loss function）与随机梯度下降（Stochastic Gradient Descendent，简称 SGD）。训练集由大规模的输入数据与预期的输出结果组成，比如图像识别应用往往使用标记好类型的图片集作为训练集。训练集的好坏很大程度上决定了训练结果的好坏。损失函数衡量预期结果与当前系统预测结果的区别，衡量了当前训练阶段的成果，给下一步对权值的调整方向以反馈信息。随机梯度下降根据损失函数相对于权值在小范围内的梯度对权值进行调整。训练集、损失函数与随机梯度下降构成了完整的深度学习训练过程。使用训练集并进行训练的深度学习也称之为有监督学习（Supervised Learning）。

1.1.2 神经网络

神经网络与深度学习不是一类概念，深度学习是一种机器学习的方法，而神经网络则是从生物神经元的构成和连接启发的一种计算过程或组织形式。深度学习侧重方法，神经网络偏重结构。深度学习定义了分层的表征学习结构，而神经网络提供了更具体的计算细节。神经网络起源于 19 世纪 60 年代的感知器（Perceptron）模型，它抽象了生物的神经元：神经元接受一组信号的输入，通过权值加权、激活函数等等的计算得到结果。神经网络将神经元组织成大规模的网络结构，相同类型的神经元组成神经网络层，提供特殊形式的计算。这种分层的神经网络结构与深度学习不谋而合。

深度学习中提到的随机梯度下降计算利用了反向传播算法。反向传播算法于 19 世纪 80 年代左右发明，主要用来推导多层网络结构中每一层的梯度，进而实现随机梯度下降计算。反向传播算法利用复合函数求导的“链式法则”（chain rule），结合网络的分层结构，可以根据网络顶部的输出结果变化反向传播得到每层输入权值的梯度。反向传播算法也因此成为深度学习训练的基础算法。与反向传播相反，前向传播（Forward propagation）是从神经网络的底层逐层计算到顶层输出结果，反向传播用来求梯度，前向传播则用来预测。

神经网络有多种网络层，有些网络层由激活函数构成：比如常用的线性整流层（Rec-

tified Linear Unit，简称 ReLU，$f(x) = max(x, 0)$），tanh 层（$f(x) = tanh(x)$）、Sigmoid 层（$f(x) = \frac{1}{1-exp(x)}$）等等；有些网络层主要是计算，比如卷积层（Convolution Layer）等等。卷积层使用一系列矩阵块（也称为卷积核）对输入值进行卷积计算。神经网络根据组成的网络层以及它们的堆叠方式不同而分类为不同的神经网络类型，卷积神经网络（Convolutional Neural Network，简称 ConvNet 或 CNN）是其中最常用的网络结构之一。

卷积神经网络的设计来源于生物的视网膜工作原理，网络结构主要由卷积层、池化层（pooling layer）和线性整流层构成：卷积层进行卷积计算提取特征（feature map），池化层按照划分好的区域过滤降维，一般求最大值（max pooling）或均值（mean pooling）。卷积神经网络广泛使用于图像识别、视频分析以至于围棋比赛中，并取得了很好的效果。除了卷积神经网络以外，递归神经网络（Reccurrent Neural Network，简称 RNN）在需要“记忆”和上下文信息的学习过程中也起到很大作用，比如文本生成和自然语言处理等。

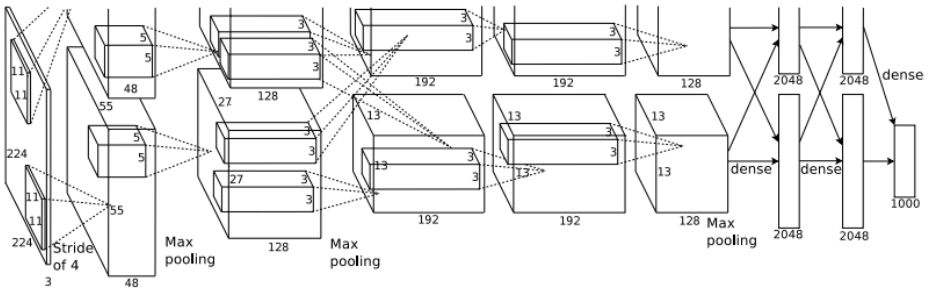

图 1.1　ImageNet LSVRC-2012 比赛中使用的大规模卷积神经网络结构[3]

综上所述，深度学习是机器学习的全新分支，挑战了传统的机器学习方法论。神经网络通过仿生的神经元与网络设计，具象化了深度学习的特征层与堆叠方式。此外，深度学习涉及到的计算，比如卷积神经网络中的卷积层等等，都十分耗费硬件的计算资源和存储资源。本研究的主要目标即为将传统的、部署于大规模 CPU 和 GPU 的深度学习应用，运行于硬件资源和计算能力都十分有限的嵌入式平台上。接下来会对本研究涉及的硬件系统进行详细的介绍。

1.2　系统芯片（SoC）

系统芯片（System-on-Chip，简称 SoC）是一类集成电路，其上包含了处理器芯片和其他电子系统，以及丰富的外部接口和设备等等。SoC 主要被电子工程师用来进行系统开发和验证：SoC 的处理器性能足够强大，一般足以运行常见的操作系统；SoC 一

般也搭载了可编程的 FPGA 硬件系统，可以非常方便修改硬件系统的设计。因此 SoC 可以明显缩短电子产品的开发周期。

SoC 的设计分为软件与硬件两部分。软件部分的开发主要涉及偏底层的硬件驱动、数据传输、硬件控制与基于操作系统的高层次软件等等。硬件系统的设计则更加复杂：工程师一方面开发基于 FPGA 的硬件逻辑，定制系统所要求的特殊功能；另一方面也尽可能基于预定义的、标准的 IP 核（全称知识产权核，Intellectual Property core）进行基础平台的搭建，提高开发效率。此外，硬件与软件不同，在开发完成之后需要进行各种系统级、行为级的功能验证才能正式完成。正因为 SoC 开发难度大，目前在开发工具上和平台设计上都有相应的进展，本研究所使用的 Xilinx Zynq SoC 平台与 SDSoC 开发工具便是例证。

本研究的目标是将深度学习框架部署于嵌入式平台中。嵌入式设备指的是一类针对特定应用场景开发的硬件设备，这类场景往往对实时性要求高，同时也对功耗与设备尺寸进行限制。SoC 作为嵌入式设备的一种实现方式，完整的功能和较低的功耗比兼具，因此非常适合作为本研究的基础开发平台。第二章对本研究使用的 Zynq SoC 平台与 SDSoC 开发工具进行详细的介绍。

1.3 现场可编程逻辑门阵列（FPGA）

FPGA（Field Programmable Gate Array，现场可编程逻辑门阵列）是一种特殊的半导体设备，特点在于可以对硬件逻辑进行重复编程。与 ASIC（Application-Specific Integrated Circuit，专用集成电路）相比，FPGA 比 ASIC 速度要慢，而且占用更多空间。但 ASIC 在设计和实现完成之后就不能修改，而 FPGA 可以不断修改其硬件配置，因此能大大提高硬件开发效率。与 CPU 相比，FPGA 针对特定形式的计算速度更快，而且功耗很低，因此在嵌入式设备和 SoC 中往往将 FPGA 视为重要的组成部分。

FPGA 的基础组件为 CLB（Configurable Logic Block，可配置逻辑块），其中包含 LUT（Look-Up Table，查找表）、全加器和触发器（Flip-Flop）等等基本元素。CLB 十分灵活，既可以用来实现时序与组合逻辑，也可以用来实现 RAM。CLB 之间则通过可编程的线路进行连接。FPGA 的可编程性主要取决于两个部分：一个是查找表的配置，查找表中包含 SRAM 与多路选择器，SRAM 的值随着硬件逻辑设计不同而不同，进而改变 CLB 的计算结果；另一个是连接 CLB 的线路，线路的交叉点也是可以配置进而改变 CLB 之间互联模式的。除此以外，不同的 FPGA 版本具有不同的资源配置，比如 BRAM（Blocked RAM，块内存）与 DSP（Digital Signal Processing，数字信号处理单元）等等在不同的 FPGA 上具有不同的数量，性能也不尽相同。

传统的 FPGA 开发使用的是 VHDL，Verilog，SystemC 等硬件描述语言，这类语言

用于表述硬件的寄存器传输级别（Register Transfer Level，简称 RTL）的抽象。使用这类语言完成设计并仿真验证之后，便可以使用不同 FPGA 厂商提供的工具进行 FPGA 的综合（Sythesis）、实现（Implementation）和比特流生成（Bitstream Generation）。综合把电路的高级抽象转换为底层的逻辑门之间的连接，建立逻辑门网表。实现则是针对具体的 FPGA 硬件，进行逻辑网表的布局（Placement）、布线（Routing）以及满足资源的限制等等。最后的生成可以编程到 FPGA 上的比特流，比特流用来配置 FPGA 上的各类硬件资源。与软件编译相比，FPGA 硬件的综合、实现与比特流生成往往需要花费几十分钟到数小时，依 FPGA 的资源和硬件设计大小而定。

综上所述，FPGA 具有可重构的硬件逻辑、优异的低功耗以及成熟的开发流程，并且广泛应用在各种计算平台中。FPGA 也有其缺点，一方面是开发门槛比较高，没有数字电路基础的软件开发者很难写出性能优秀的 FPGA 应用；另一方面，FPGA 只适用部分的计算过程，涉及到复杂的条件判断、循环等计算过程更适合部署于 CPU 执行。因此，本研究将 FPGA 作为硬件加速器，即部署计算量大但逻辑简单的计算过程于 FPGA 上加速。同时本研究也是用高层次综合工具提高硬件逻辑的开发效率。接下来进一步介绍高层次综合的基本概念和使用方式。

1.4　高层次综合

高层次综合（High-Level Synthesis，简称 HLS）是一种将行为级的、算法级的代码转换为硬件逻辑的过程。高层次综合的输入不再是传统的硬件描述语言，取而代之的是 C/C++ 等高级程序语言。高层次综合使得硬件逻辑开发可以从软件代码开始：传统的开发方式需要等到 RTL 实现完成之后才能验证，迭代周期很长；而高层次综合工具可以令开发者先实现好软件系统，之后直接把需要在 FPGA 上加速的代码进行高层次综合即可。通过高层次综合工具，软件工程师可以不用学习 RTL 级别的描述语言就可以实现硬件逻辑。根据 Xilinx 的调查显示，基于高层次综合的设计开发方法与传统方法相比时间加快 4 倍，结果质量提高 0.7 到 1.2 倍。

一般的高层次综合流程包含如下几步：控制流和数据流图（Control Data Flow Graph）分析，资源分配（Resource Allocation），调度（Scheduling），绑定（Binding）等等。高层次综合的流程本质上是求解在资源和时间的约束下，能达到最优的资源分配和调度策略。这种最优解往往依赖启发式的算法，因此高层次综合工具分析的结果往往比较保守，在资源配置和调度上不一定能取得最佳的结果，因此需要通过开发者手动指定生成硬件的选项。比如指定某一运算需要使用 DSP 实现，某一循环需要流水线化等等。

综上，高层次综合工具提供了一种生产力更高的开发 FPGA 应用的方式，为了取得更高的性能也需要开发者对应用进行分析、手动生成限制条件。本研究使用的 SDSoC

工具中整合了 Xilinx Vivado HLS 工具，在后续的硬件加速器的设计中主要用它进行硬件逻辑的实现和优化。

1.5 总结

第二章　基本设计

本章阐述本研究的整体设计。整体设计部分在基本原理之上，给出更具体的实现方案设计。从本研究的目标出发，为了提供可以运行于 FPGA SoC 平台的深度学习框架，首先应确定用什么 FPGA SoC 平台和使用什么样的开发工具。其次，如果自行从头构建深度学习框架，一方面工作量过于庞大、需要考虑很多优化问题；另一方面也缺乏足够的受众，研究人员更熟悉流行的深度学习框架。因此应选用成熟的流行的深度学习框架作为基础，在其上进行针对特定嵌入式平台的适配与优化。

基于上述的考虑，本章首先介绍 Xilinx Zynq 平台的系统架构、搭载的 ARM 与 FPGA 的硬件特点、以及本研究使用的基于 Zynq 的开发板 Zedboard 的具体特性等等。之后给出 Xilinx SDSoC 工具的基本情况介绍，包括其包含的工具以及支持的开发流程。随后介绍本研究选用的基础深度学习框架 Caffe，主要涉及与后续开发密切相关的软件架构和实现方式。最后，综述整体设计方案，即本研究的最终结果是如何有机结合 ZYNQ，SDSoC 和 Caffe。

2.1　Zynq 平台

Zynq 平台的全称是“全可编程 SoC”（All Programmable SoC），由双核的 ARM Cortex-A9 处理系统（Processing System，简称 PS）与 Xilinx 可编程逻辑（Programmable Logic，简称 PL）组成。除此以外，还包含片上内存（On-chip memory），外部内存接口（External Memory Interface），各类外围设备输入输出接口（I/O Peripherals and Interfaces）以及连接 PS 与 PL 的高速 AXI 总线等等。

本研究基于 Zynq 平台的最新系列 Zynq-7000 进行系统设计与实现。

2.1.1　Zynq 系统架构

Zynq 平台的系统架构如图 2.1 所示。接下来分块介绍 Zynq 的几个组成部分。

ARM 处理系统

Zynq SoC 系统上搭载了双核 ARM Cortex-A9 MPCore 处理器，该处理器可以运行到最高 1GHz，拥有指令和数据的 32KB L1 缓存与 512KB L2 缓存，以及 NEON 媒体处理引擎（media-processing engine）和浮点数向量处理单元（Vector Floating Point Unit，简称 VFPU）等等。Cortex-A9 的性能十分优异，足以满足多种情况下的计算需求。

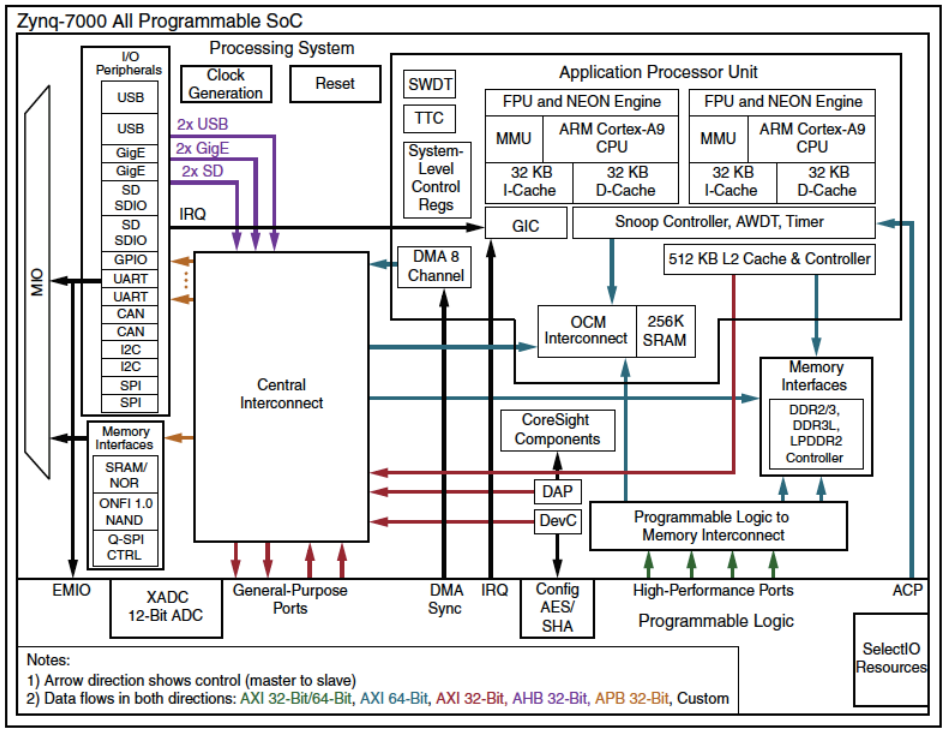

图 2.1　Zynq-7000 平台的系统架构

本研究中依赖该 ARM 处理器提供的 CPU 端环境进行大部分的计算。有关 ARM 处理器与内存系统和 PL 的交互接口在 2.1.4 节介绍。

可编程逻辑

Zynq 的可编程逻辑（PL）与一般的 FPGA 设计一致，主要包含 CLB，BRAM 以及 DSP 处理单元。相关指标如下表所示：

Programmable Logic Cells	LUTs	Flip-Flops	BRAMs	DSP Slices
85K	53,200	106,400	560 KB	220

表 2.1　Zynq-7000 XC7Z020 可编程逻辑上资源容量

本研究使用的 XC7Z020 系列 Zynq 的可编程逻辑的配置与性能大致上与 Artix-7 FPGA 相似，硬件资源比较有限。因此想要取得高效的性能必须做精细的资源分配和优化。相关的优化策略和技巧在第四章详述。

数据通路 AXI 与存储系统

存储系统与数据通路负责连接处理系统与可编程逻辑，以及处理和其他外围设备通信。Zynq 平台的存储系统主要由三个部分组成：处理系统中的缓存（L1 与 L2 cache）和

片上内存（On-chip Memory）；可编程逻辑上的 BRAM；以及大容量的外部存储（External Memory）。Zynq 上的几种数据通路可以完成这三个组成部分之间的不同形式的交互。

Zynq 上的数据通路的基础是 ARM 的 AXI（Advanced eXtensible Interface）总线协议，属于 AMBA（Advanced Microcontroller Bus Architecture）协议的一部分。AXI 的最新版本是 AXI4 协议，Xilinx 设计并实现了基于 AXI4 的 IP 核，使得 AXI 变得更加灵活、高效和便捷。AXI4 协议主要有三类：针对大数据量传输的 AXI4，小数据量的 AXI4-Lite，以及针对流数据的 AXI4-Stream。Zynq 上的数据通路正是基于 Xilinx 提供的 AXI IP 核实现的。

Zynq 基于 AXI 总线协议连接处理系统和可编程逻辑，主要实现两类接口：高性能 AXI 端口（High-Performance AXI ports）与 ACP（Accelerator Coherency Port）接口，二者区别关键在内存数据一致性上。高性能 AXI 端口可以实现处理系统和可编程逻辑高速的数据传输，但不保证 CPU 缓存中的数据与内存读写一致。ACP 接口则在硬件机制上保证缓存中的数据在传输之前被写回内存、内存更新之后缓存同步更新。

在设计 Zynq 应用时具体使用哪种数据通路取决于系统的特性。本研究在第三章给出对数据通路选择的原则，在第四章会具体讨论两种数据通路的性能对比。

2.1.2　Linux 运行模式

基于 Zynq 平台开发的应用有两种运行模式：Standalone 与 Linux 模式。前者代表应用的运行不用启动操作系统，后者则要求先启动 Linux 操作系统再运行应用。考虑到深度学习应用并不是简单的计算，还需要各种数据处理的操作，并且科研人员更熟悉基于操作系统的运行模式，本研究只在 Linux 模式下进行开发。

Linux 模式首先启动预先加载与 ROM 中的引导程序，该程序从 SD 卡中读取第一阶段启动程序（First Stage Boot Loader, 简称 FSBL），FSBL 负责处理可编程逻辑的比特流（bitstream）和 Linux 引导程序（u-boot）的加载与执行。引导程序之后分别加载 Linux 的镜像文件（uImage）、设备树文件（Device Tree）和根系统文件（Rootfs）。Linux 镜像主要包含 Linux 内核，设备树文件定义了硬件设备环境供引导程序动态加载，根系统文件则包含了系统的根目录和必要的程序与库。准备就绪之后即可通过 UART 接口或者 SSH 连接到 Zynq 平台，运行程序或进行开发。

Xilinx 提供了 bootgen 工具来创建 SD 卡引导程序，以及一系列预生成的文件，例如 Linux 镜像，根系统文件等等。SDSoC 工具中包含了 bootgen，以及更多跟硬件平台相关的预生成文件，可以更简便地生成 SD 卡中需要的内容。在 2.3 节中会进一步介绍。

2.1.3 Zedboard

本研究选用的是 Zedboard 作为 Zynq 开发平台。Zedboard 是基于 Zynq-7000 系列的扩展式处理平台，Zedboard 上除了满足 Zynq-7000 所要求的 ARM Cortex-A9 MPCore 处理器与 Xilinx FPGA 之外，还增加了如下配置：512MB DDR3 内存、256MB QSPI 闪存和一系列输入输出接口（HDMI、VGA、OLED 与音频接口）等等。除此以外，Zedboard 成本低廉，还有 Xilinx SDSoC 的特别支持，尤其是预先搭载好的输入输出接口对深度学习应用的开发（计算机视觉等应用）非常方便。因此本研究使用 Zedboard 作为最终深度学习框架所运行的平台。

2.2 SDSoC 开发环境

Xilinx 于 2015 年 3 月推出了 SDSoC 开发环境，提供类似于嵌入式 C/C++ 开发体验。SD 即“软件定义”（Software Defined）：SDSoC 的目标是降低 FPGA 应用开发门槛，让更多的软件开发者，而不是专业的 FPGA 开发人员，可以通过 C/C++ 代码直接生成 Zynq SoC 系统。此外，SDSoC 综合了 Vivado HLS，Xilinx ARM GNU 编译工具链，以及一系列系统生成工具（bootgen, Xillybus 等等），令开发效率得到很大提升。但是 SDSoC 的最大优势在于其对传统 SoC 开发流程的改进。接下来主要介绍 SDSoC 的开发流程与基本使用方式。

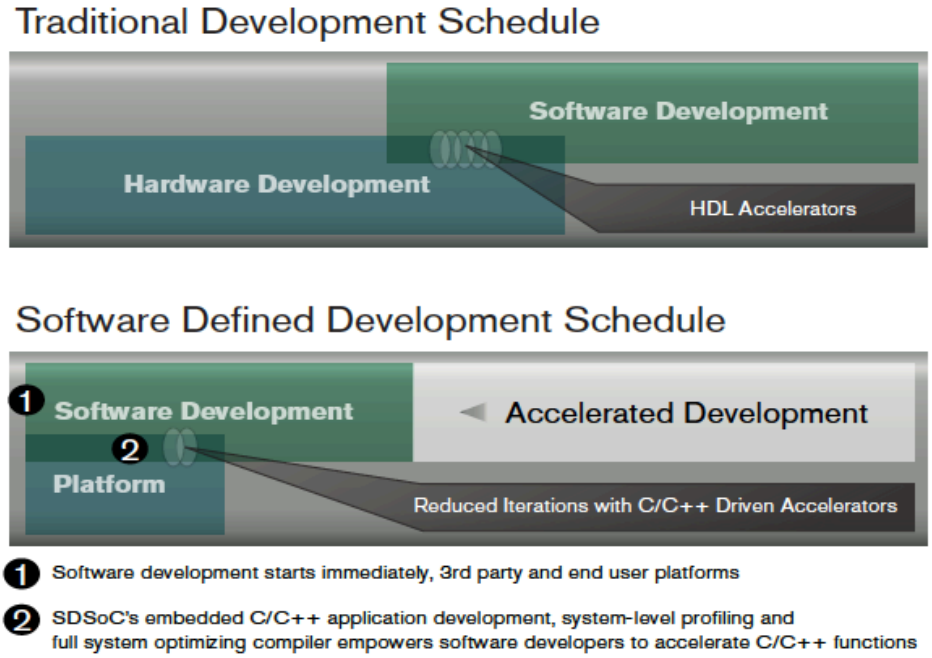

图 2.2 传统的开发流程（Traditional Development Schedule）与软件定义的开发流程（Software Defined Development Schedule）的区别

2.2.1　软件定义的开发流程

传统的 SoC 平台的系统开发流程首先需要完成硬件部分的设计，之后软件开发者才能通过已经综合实现好的硬件接口进行进一步开发。这种开发流程迭代周期非常长，软件部分的开发在硬件设计过程中处于停滞状态，硬件设计只能等软件部分完成才能获得反馈。基于 SDSoC 的开发基于软件定义的开发流程：先用 C/C++ 代码完成整个系统的设计，之后以函数为单元部署于可编程逻辑，进行硬件加速。SDSoC 甚至提供了硬件加速性能预测工具，可以令开发人员在长时间的综合实现完成之前就能大概了解目前的优化策略是否正确。图 2.2 是对两种开发流程的对比。

2.2.2　基本概念与使用方式

数据移动网络

SDSoC 开发环境关键概念是数据移动网络（data motion network），表示 PS 与 PL 之间的数据通路模式：SDSoC 使用预先设计好的数据通路 IP 核来负责 PS 与 PL 之间的数据传输，这些 IP 核称为数据移动单元（data mover）。比如 AXIDMA_SG 就表示通过 AXI 总线用 Scatter-Getter 的方式通过 DMA 传输数据。数据移动单元与数据类型密切相关，标量数据只能用 AXI_LITE 传输，而数组与更复杂的数据类型可以用 AXI_DMA_SG，AXI_DMA_SIMPLE，AXI_DMA_2D 等数据传输单元传输。一般而言 SDSoC 可以根据数据类型，内存分配方式等分析出最适合使用的数据传输单元，但开发者也可以使用预编译指令（pragma）进行优化。

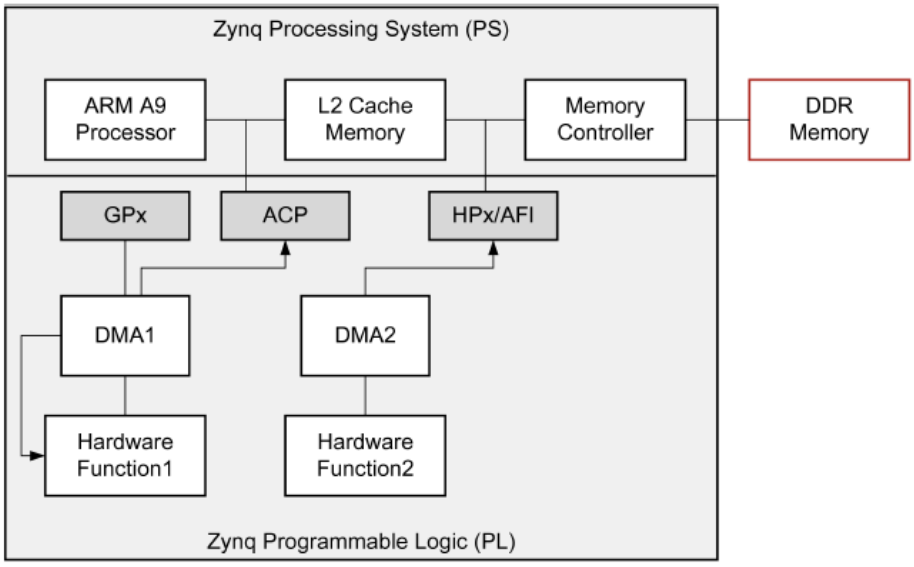

图 2.3　Zynq AXI 总线协议的几种端口

使用方式

SDSoC 的可以通过其 IDE 使用，也可以基于纯粹的 Linux 命令行使用。由于 IDE 的限制更多，本研究完全使用命令行环境开发。SDSoC 的最关键工具是 sdscc/sds++。这两个程序本质上为 C/C++ 编译器，可以完全兼容 gcc/g++ 的命令行选项。但与一般编译器不同，sdscc/sds++ 可以通过命令行选项指定：1）需要 HLS 的函数：包含函数名和要求的时钟频率；2）数据移动网络的时钟频率；3）目标硬件平台：比如 zed（针对 Zedboard）或 zybo（针对 ZYBO 开发板）。因此，通过 sdscc/sds++ 工具可以轻易与原有的 Linux 项目进行兼容。有时候也需要更细粒度的操作，可以使用 arm-xilinx-linux-gnueabi 的 GNU 工具链对 C/C++ 代码进行交叉编译，使用 sdslib 封装 IP 核等等，在第三章对 Caffe 和第三方库的移植中会详细介绍。

本研究的应用完全依赖 SDSoC 进行开发，利用 Vivado HLS 进行硬件逻辑编写和优化，SDSoC 自带的 IP 核进行数据通路的自动生成，以及 Xilinx ARM GNU 工具链编译 CPU 端代码并进行链接。最后 SDSoC 把项目全部整合到 SD 卡中。

2.3 深度学习框架 Caffe

Caffe 是加州伯克利大学计算机视觉组开发的深度学习框架，基于 C++ 与 CUDA 开发。Caffe 因其速度快，数据模版清晰易懂，并拥有开源的、高度模块化的代码而广受开发者欢迎。

本研究选用 Caffe 作为移植到 Zynq 上进行移植和加速的对象，主要是因为其清晰的软件架构，非常便于修改和优化。除此以外，与其他深度学习框架（如基于 LuaJIT 的 Torch，基于 Python 的 Theano）相比，Caffe 的 CPU 端代码完全基于 C++ 实现，与 SDSoC 可以无缝衔接。因此本研究最终选用 Caffe 作为优化的基础。

Caffe 的标准用法非常简单，可以通过编译出的 caffe 命令行工具直接训练 Caffe 模型，其中 Caffe 模型与训练数据需要提前定义好。除此以外，也可以使用 Caffe 动态链接库（libcaffe.so）在其他的 C++ 应用中使用 Caffe 提供的接口。两种方法都十分方便。接下来着重介绍与移植相关的 Caffe 软件架构和神经网络层的实现。

2.3.1 软件架构与第三方库依赖

Caffe 的软件架构主要包含如下几个部分，依赖关系具有清晰的分层次的结构：

1. 基础架构：数据类型的定义，比如 blob.cpp 定义了 Blob 数据结构，net.cpp 与 layer.cpp 分别定义了网络类与网络层类。同时常用数学函数 math_functions.cpp 也十分重要，Caffe 的各种大运算量的计算基本都会调用该文件所定义的函数。

2. 网络层的实现：在 layers/目录下定义了常见的神经网络层的实现，比如卷积神经网络 conv_layer.cpp，ReLU 层 relu_layer.cpp 等等。相关 GPU 代码也包含其中。
3. 求解器实现：在 solvers/目录下定义了多种求解器。
4. 其他：比如 Caffe 使用的 Protobuf 文件等等。

Caffe 所依赖的第三方库不多，可以简单分为如下几类：

1. 数据存储：LMDB 与 LevelDB
2. 数据格式和接口：HDF5 与 Protobuf
3. 数学计算：BLAS 库比如 ATLAS 或 OpenBLAS
4. 其他：比如日志 Google glog，基础组件 Boost 等等

总之，Caffe 的分层设计和简单的第三方库依赖都非常便于优化和修改。

2.4　综合设计方案

结论

pkuthss 文档模版最常见问题：

\cite、\parencite 和 \supercite 三个命令分别产生未格式化的、带方括号的和上标且带方括号的引用标记：1，[5]、[1,5]。

若要避免章末空白页，请在调用 pkuthss 文档类时加入 openany 选项。

如果编译时不出参考文献，请参考 texdoc pkuthss“问题及其解决”一章“其它可能存在的问题”一节中关于 biber 的说明。

参考文献

[1] Author. “*Title*” [J]. *Journal*, 2014-04-01.

[2] Yoshua Bengio, Aaron Courville and Pierre Vincent. “*Representation learning: A review and new perspectives*”. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **2013**, *35*(8): 1798–1828.

[3] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. “*Imagenet classification with deep convolutional neural networks*”. In: *Advances in neural information processing systems*, **2012**: 1097–1105.

[4] Yann LeCun, Yoshua Bengio and Geoffrey Hinton. “*Deep learning*”. *Nature*, **2015**, *521*(7553): 436–444.

[5] 作者。“标题”[J]。期刊，2014-04-01。

附录 A　附件

pkuthss 文档模版最常见问题：

\cite、\parencite 和 \supercite 三个命令分别产生未格式化的、带方括号的和上标且带方括号的引用标记：1，[5]、[1,5]。

若要避免章末空白页，请在调用 pkuthss 文档类时加入 openany 选项。

如果编译时不出参考文献，请参考 texdoc pkuthss“问题及其解决”一章“其它可能存在的问题”一节中关于 biber 的说明。

致谢

pkuthss 文档模版最常见问题：

\cite、\parencite 和 \supercite 三个命令分别产生未格式化的、带方括号的和上标且带方括号的引用标记：1，[5]、[1,5]。

若要避免章末空白页，请在调用 pkuthss 文档类时加入 openany 选项。

如果编译时不出参考文献，请参考 texdoc pkuthss“问题及其解决”一章“其它可能存在的问题”一节中关于 biber 的说明。

北京大学学位论文原创性声明和使用授权说明

原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不含任何其他个人或集体已经发表或撰写过的作品或成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律结果由本人承担。

论文作者签名：　　日期：　　年　　月　　日

学位论文使用授权说明

（必须装订在提交学校图书馆的印刷本）

本人完全了解北京大学关于收集、保存、使用学位论文的规定，即：

- 按照学校要求提交学位论文的印刷本和电子版本；
- 学校有权保存学位论文的印刷本和电子版，并提供目录检索与阅览服务，在校园网上提供服务；
- 学校可以采用影印、缩印、数字化或其它复制手段保存论文；
- 因某种特殊原因需要延迟发布学位论文电子版，授权学校在□一年/□两年/□三年以后在校园网上全文发布。

（保密论文在解密后遵守此规定）

论文作者签名：　　导师签名：　　日期：　　年　　月　　日