分类号：TN4　　单位代码：10335

密　级：公开　　学　　号：21931061

浙江大学

硕士学位论文

中文论文题目：面向语音关键词识别的低功耗神经网络处理器设计

英文论文题目：Design of Low Power Neural Network Processor for Speech Keyword Spotting

申请人姓名：杨树园

指导教师：沈海斌

专业名称：电子科学与技术

研究方向：智能系统与芯片

所在学院：信息与电子工程学院

论文提交日期　**2022 年 5 月**

面向语音关键词识别的低功耗神经网络处理器设计

论文作者签名:_____

指导教师签名:_____

论文评阅人 1:_____

评阅人 2:_____

评阅人 3:_____

评阅人 4:_____

评阅人 5:_____

答辩委员会主席:_____

委员 1:_____

委员 2:_____

委员 3:_____

委员 4:_____

委员 5:_____

答辩日期:_____

浙江大学研究生学位论文独创性声明

本人声明所呈交的学位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。除了文中特别加以标注和致谢的地方外，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 **浙江大学** 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谢意。

学位论文作者签名：　　签字日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解 **浙江大学** 有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 **浙江大学** 可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名：　　导师签名：

签字日期：　　年　　月　　日　　签字日期：　　年　　月　　日

致谢

三年的硕士研究生生涯转瞬即逝，还记得初入实验室做毕业设计时的陌生，不知不觉已经走过了三年，到了要说再见的时候。在此，感谢一路上细心指导的老师，感谢给我帮助和支持的家人和朋友们，是你们给我力量，伴我前行。

首先感谢沈海斌老师和黄科杰老师在科研工作上的悉心指导。沈老师开阔的视野，精益求精的态度、渊博的学识让我受益良多，感谢沈老师在我碰到困难时的帮助。黄老师在神经网络加速器上具有前瞻性见解，他大胆创新的精神激励和感染了我，感谢一直以来的指导和帮助。

其次，感谢朱朝阳、周资群、郑瑞沣、章烨炜等师兄的帮助，感谢同届的刘一铭、厉军城、赵智洋、陆凯晨、肖蕊、李鑫的陪伴和鼓励，以及陈杨、王文威、李永根、顾峻川等师弟的支持和配合。感谢室友胡永睿、肖晓辉、张泽琪，谢谢你们的理解和包容。很高兴与你们在浙大相识。

最后，感谢我的家人。感谢父亲和母亲对我一直以来的鼓励和关怀，你们的理解和支持是我前行路上最坚强的后盾。

摘要

人工智能物联网（Artificial Intelligence of Things，AIoT）在终端物联网设备上结合了人工智能算法，使设备更加智能，提升了用户体验。在物联网应用中，存在大量使用神经网络算法进行序列信号处理的任务，如语音关键词识别等。由于物联网设备通常由电池供电，因此对功耗有着严格的要求。此外准确度直接反映了设备的响应能力，延时会影响用户和设备之间交互的速度，因此高准确度和低延时对用户来说十分重要。存内计算技术在存储器中嵌入了计算的功能，大大减少了存储器访问的次数，具有高能效和高吞吐量的优势，非常适合用来运行神经网络算法。受限于硬件开销，存内计算技术中往往使用低位宽模数转换器（Analog-Digital Converter, ADC）进行量化，由于数据分布的不均衡性，使用低位宽 ADC 进行量化时量程利用率低，从而降低了准确度。综上所述，在较小硬件开销的前提下，如何获得较低功耗、较低延时与较高准确度，是神经网络处理器实现过程中的一大问题。针对上述问题，论文的主要研究内容和工作如下：

首先提出了面向神经网络量化分布的 ADC 动态缩放方法，该方法包括读出电流调整、参数倍增和脉宽调制 3 个步骤，可以根据输出激活的分布进行动态缩放，在充分利用低位宽 ADC 量程的前提下，提高了存内计算硬件运行神经网络推理计算的准确度。

其次设计了低功耗神经网络处理器指令集架构，以存内计算为依托，在张量运算器、控制器、通用寄存器组、配置寄存器组和地址寄存器组等框架基础上，配合配置、正常和休眠三种模式，建立了支持寄存器迭代计算的 SIMD 专用优化指令集，以实现对高能效语音关键词识别的支持。

最后设计了低功耗神经网络处理器微架构，设计了 IF、ID、EX、WB 四级流水线，围绕 S-TC-Resnet 神经网络算法在处理器上的流式推理，提升了数据重用，减小了每帧的计算量。40nm eflash 工艺下的仿真结果表明，与同类工作相比，本文设计的处理器在保证相对较高准确度的前提下，功耗得到了较大幅度的降低。

关键词：低功耗，动态缩放，流式推理，存内计算，神经网络处理器

Abstract

Artificial Intelligence of Things (AIoT) combines artificial intelligence algorithms on IoT devices to make the devices smarter and improve user experience. Neural network algorithms are widely used for sequential signal processing in AIoT applications, such as keyword spotting (KWS). IoT devices are usually battery-powered, which imposes strict requirements on power consumption. In addition, accuracy directly reflects the responsiveness of the device, and delay will affect the speed of interaction between the user and the device, so high accuracy and low delay are very crucial for users. The IMC (In-Memory Computing) technology embeds computing capabilities in the memory, which greatly reduces the number of memory accesses. IMC technology achieves high energy efficiency and high throughput, and is very suitable for running neural network algorithms. Limited by hardware overhead, low bit-width ADC(Analog-to-Digital Converter) are often used for quantization in IMC (In-Memory Computing) technology. Due to the unbalanced data distribution, the use of low bit-width ADC for quantization will cause underutilization of the ADC range and bring down the accuracy. To sum up, under the premise of less hardware overhead, how to obtain lower power consumption, lower latency and higher accuracy is a major problem in the implementation of neural network processors. In view of the above problem, the main research contents and work of this dissertation are as follows:

Firstly, a dynamic scaling method for distribution of quantized neural network is proposed. The method includes three steps: current adjustment, parameter increment and pulse width modulation, which makes full use of the range of the low bit-width ADC and improves the accuracy of inference based on the IMC core.

Then, a low-power neural network processor instruction set architecture based on in-memory computing is designed. The processor contains a tensor processing unit, a controller, a general-purpose register file, a configuration register file and an address register file. It has three modes: configuration mode, normal mode and sleep mode. A SIMD optimized instruction set supporting register iterative computation is built to

enable energy-efficient speech keyword spotting.

Finally, a low-power neural network processor microarchitecture with a four-stage pipeline of IF, ID, EX, and WB is designed. The processor improves data reuse and reduces computation per frame by streaming inference of the S-TC-Resnet. Measured results in 40nm eflash process show that compared with similar work, the power consumption of the processor designed in this dissertation has been greatly reduced under the premise of high accuracy.

Key Words: Low Power, Dynamic Scaling, Streaming Inference, In-memory Computing, Neural Network Processor

目录

图目录

表目录

缩写词表

缩写	英文名称	中文名称
KWS	keyword spotting	关键词识别
ADC	Analog-to-Digital Converter	模数转换器
CNN	Convolution Neural Network	卷积神经网络
LSTM	Long-short Term Memory	长短期记忆
GRU	Gated Recurrent Unit	门控循环单元
DSCNN	Depthwise Separable Convolution Neural Network	深度可分离卷积神经网络
TCN	Temporal Convolutional Network	时间卷积网络
TC-Resnet	Temporal Convolutional Residual Network	时间卷积残差网络
S-TC-Resnet	Streaming Temporal Convolutional Residual Network	流式时间卷积残差网络
SONOS	Silicon Oxide-Nitride-Oxide-Silicon	硅-氧化物-氮化物-氧化物-硅

第1章 绪论

1.1 课题背景与意义

人工智能物联网（Artificial Intelligence of Things,AIoT）在物联网设备上结合了人工智能技术来实现更有效的物联网操作，通过增强数据的分析过程，提升人和设备之间的交互能力。人工智能技术可以用来对原始的物联网数据进行学习训练，从而提升相关任务决策的表现。轻量级神经网络算法被广泛用于物联网设备中进行序列信号处理，如语音关键词识别[1]，心电图信号处理[2]，脑电图信号处理[3]等。其中语音关键词识别可以检测语音信号流中关键词是否出现，大量应用于支持语音控制的物联网设备中，如手机、可穿戴设备、智能家居等，语音关键词识别可以帮助人们使用自然语言实现对设备的直接控制，带来更好的用户体验。

由于神经网络算法具有参数量大和计算量大的特点，因此运行神经网络算法通常会产生较大功耗，而物联网设备通常由电池供电或者充电，一个典型的 LR44 型号纽扣电池的容量为 190mAh/1.5V[4]，为了尽量延长电池的使用时间，物联网设备执行神经网络算法的功耗需要尽可能低，同时延时影响设备与用户间的交互速度，因此低延时对好的用户体验来说十分重要。由于通用处理器运行神经网络算法在功耗、延时等方面难以满足上述要求，因此有必要设计加速器或专用的神经网络处理器。

存内计算指在存储器内和临近存储器的电路中完成计算，从而大大减少存储器访问的次数，减少数据搬移的功耗[5]。由于上述存内计算的概念与神经网络算法中权重存储和激活计算邻近的特点相一致，因此本文结合了存内计算技术来设计低功耗的神经网络处理器。

1.2 国内外研究现状

最近几年的研究工作设计了一些面向语音关键词识别的神经网络加速器或神经网络处理器[7][8][27]，两者在架构上具有相似性：（1）均以大规模的计算阵列为核心，如文献[8]中的专用处理器和文献[7]中的加速器均设置了8×8大小的计算阵列；（2）控制逻辑较简单，如文献[7]中的加速器具有简单的配置寄存器，文献[8]中的专用处理器尽管设置了指令集，但是由于神经网络推理过程的逻辑较

简单，因此并不需要设置复杂的分支跳转等指令，即专用处理器的实现方式在控制逻辑的复杂程度上与加速器相近；（3）需要一定容量的片上存储器，并往往根据不同的数据类型进行设置，如权重存储器、激活存储器等。如文献[7]的加速器中设置了 64KB 的权重存储器和 7.4KB 的激活存储器，文献[8]的专用处理器中设置了 64KB 的权重存储器和 64KB 的激活存储器。综上所述，面向神经网络的加速器和专用处理器在架构等方面具有相似性，对其进行区别讨论的意义不够显著，出于方便叙述的角度，本文统一使用神经网络处理器指代两者，下面对相关领域的研究现状进行具体介绍如下。

目前面向语音关键词识别的神经网络处理器根据采用的架构可以分为 2 类，包括全数字电路架构和结合存内计算的数模混合电路架构。

文献[6]中采用了全数字电路的架构，使用深度可分离卷积神经网络来实现语音关键词识别，其中的深度可分离卷积层相比传统卷积层计算量减少了 6 倍。该工作采用面向深度可分离卷积的硬件优化方法减少了冗余计算量和存储空间，每帧计算量仅包含 8736 次 MAC(Multiplier and Accumulation)运算，在 GSCD 数据集上的准确度为 94.6%，但由于采用的神经网络模型较小，因此只能识别 2 个关键词。该芯片工艺为 28nm，在 0.41V 和 40KHz 下测试性能，神经网络处理器部分的功耗仅为 0.17uw，但低频同时带来了较高的延时，[6]中的延时达到了 64ms。

文献[7]中基于全数字电路架构，设计了面向时间卷积残差网络（Temporal Convolutional Residual Neural Network, TC-Resnet）的神经网络处理器。[7]中采用了条件执行策略，对每个残差层结果送入退出分支中进行计算，若当前结果已经达到了退出标准，则直接输出推理结果，结束本次推理，从而减小推理的计算量。[7]采用 22nm 工艺，可推理关键词个数为 10，在 GSCD 数据集上准确度为 93.09%，虽然该工作准确度较高，但功耗和延时也较高，其中功耗为 8.2uw，每次推理延迟为 100ms。

文献[10]中采用了基于 SRAM 内计算的数模混合架构，设计了面向注意力循环网络（Recurrent Attention Model, RAM）的神经网络处理器，可识别 7 个关键词，在 GSCD 数据集上准确度为 90.38%。该工作充分利用了 SRAM 存内计算核高能效和高吞吐量优势，具有很低的动态功耗和推理延时，单次推理的能耗为

0.44uJ，单次推理的延时为 39.9us，但是该工作中片上大容量的 SRAM 会带来大的漏电功耗，而在物联网应用中，静态功耗的占比很高[6]。

此外上述提到的文献[7]具有可配置的特性，可以运行可变大小的 TC-Resnet 网络，文献[6][10]中的架构针对具体的网络设计，灵活性低。由于用于语音关键词识别的神经网络模型也可以进一步迁移到其他序列信号如心电图信号的处理[2]，因此设计具有一定灵活性的神经网络处理器架构来支持不同的序列信号处理，可以进一步拓展芯片的应用场景。综上所述，目前支持语音关键词识别的神经网络处理器，在功耗、延时、准确度和灵活性等方面还存在提升的空间。

1.3 研究内容与章节安排

本文的研究目标是结合存内计算技术来设计面向语音关键词识别的低功耗神经网络处理器，主要研究内容和工作如下：

1）介绍了神经网络模型量化相关理论，分析了存内计算核中 ADC 执行激活量化的过程，然后针对由于输出激活的动态分布而导致的 ADC 量程利用率低的问题，提出了动态缩放方法，并设计实验验证了动态缩放方法对于神经网络推理准确度的提升效果。

2）以存内计算核为核心设计了张量处理单元，配合控制器和寄存器组等设计了低功耗神经网络处理器框架，并根据实时语音关键词识别的特点设计了处理器的工作模式，在此基础上建立了 SIMD 专用优化指令集。

3）设计了处理器的 4 级流水线，并设计了处理器流式推理的执行方式来实现数据复用，大幅减小了计算量，最后在 40nm 工艺下进行了仿真实验，分析了功耗组成，并与国内外同类设计进行了对比，验证是否满足低功耗的设计要求。

本文共包含五章，各章的主要内容如下：

第一章绪论介绍了论文的背景，阐述了课题研究的意义，然后介绍了国内外面向语音关键词识别的神经网络处理器的研究现状。通过分析研究现状，提出了现有的神经网络处理器存在的问题，并给出了本文解决这些问题的思路。

第二章介绍神经网络专用处理器技术基础，从语音关键词识别总体框架、神经网络模型和神经网络处理器基本设计方法三个方面展开，并对 TC-Resnet 算法

进行流式推理优化，得到了本文的目标算法 S-TC-Resnet。

第三章为动态缩放方法设计，首先分析了存内计算核执行量化的过程，针对如何充分利用低位宽 ADC 量程提高量化准确度的问题，设计了动态缩放方法，并通过实验验证了动态缩放方法对推理准确度的提升效果。

第四章为低功耗神经网络处理器的设计实现，首先从指令集架构的角度介绍了处理器框架、工作模式和专用指令集设计，然后从微架构的角度介绍了流水线设计、存内计算核设计以及流式推理的实现，最后对处理器在语音关键词识别任务下的性能进行了测试，并和相关工作进行了对比分析。

第五章为总结与展望，首先总结本文所做的工作，然后对后续进一步的研究工作进行了展望。

第2章 神经网络专用处理器技术基础

2.1 引言

本章首先介绍基于神经网络算法的语音关键词识别总体框架，阐述语音关键词识别的相关原理，然后介绍神经网络基本概念和神经网络处理器设计方法，最后对 TC-Resnet 模型进行了流式推理优化，设计了本文的目标算法 S-TC-Resnet。

2.2 语音关键词识别总体框架

随着深度学习近年来的发展，基于神经网络算法的语音关键词识别系统展现出优异的性能[6][7][8]，其流程如图 2-1 所示，包含了 3 个阶段：信号采集、预处理和神经网络分类[27]，本文设计的专用处理器即面向该流程设计。

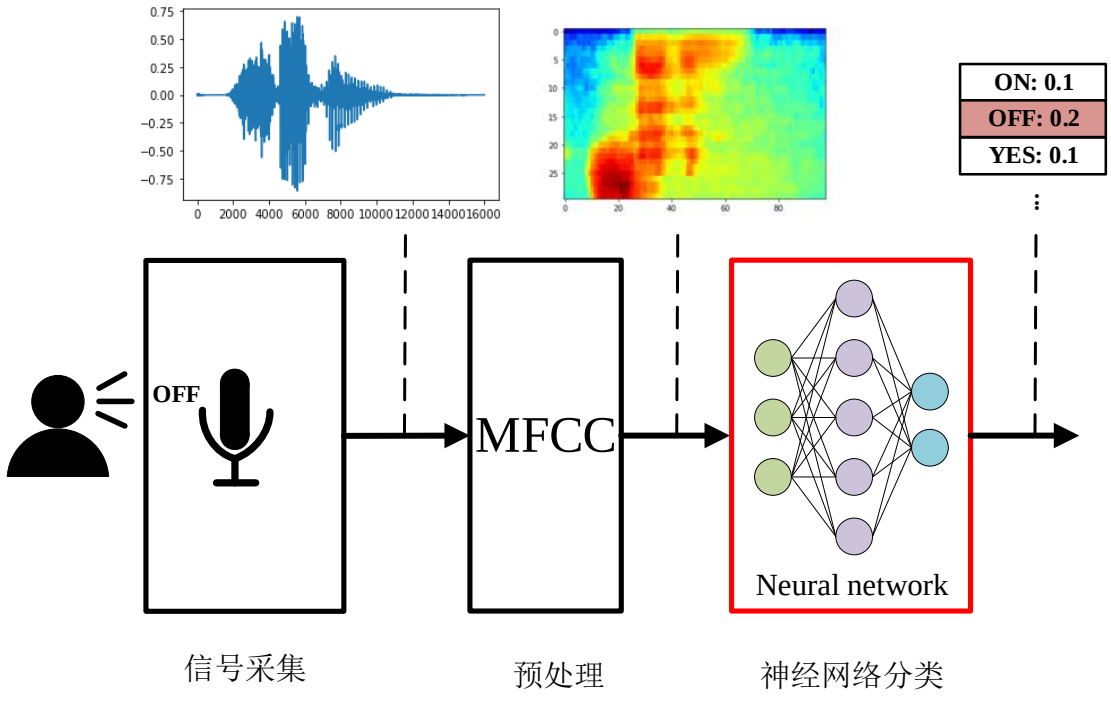


图 2-1 语音关键词识别流程图

信号采集阶段传感器采集原始的声音信号，对其进行放大和滤波等操作，然后进行模数转换并输出数字信号结果。信号采集阶段输出的数据维度与采样率有关，采样率通常设置为 8KHz 或 16KHz，在 16KHz 的情况下每秒输出 16K 个采样点。

预处理阶段对信号采集阶段输出的语音信号进行声学特征提取，降低数据维度，目前语音关键词识别中应用最广泛的预处理方式为梅尔频率倒谱系数（Mel-

Frequency Cepstral Coefficients，MFCC）。MFCC 预处理首先在时域上进行预加重、分帧和加窗操作，经过时域上的处理后，语音信号分割为语音帧，常见的语音帧宽度为 10-30ms，相邻帧之间有$\frac{1}{3}$到$\frac{1}{2}$的重叠。然后 MFCC 使用快速傅里叶变换变换将语音信号从时域转换到频域，并依次在频域上进行梅尔滤波、取对数和离散余弦变换[14]得到最终的频域特征向量，根据梅尔滤波不同的参数设置，每帧输出的特征向量维度在 10-40 之间。

神经网络分类阶段接收预处理阶段输出的特征向量，多个相邻的特征向量组成一个完整的输入特征图（对应一段语音信号经过 MFCC 预处理后的输出，可视为一个样本），神经网络对输入特征图进行进一步的特征提取，并完成分类，输出该段语音判别为每个类别的概率。类别通常分为 3 种，包括特定关键词、无关键词和未知关键词。比如一个可以检测 10 个关键词的神经网络包含 12 个输出，其中有 10 个输出分别代表判定为 10 个特定关键词出现的概率，剩下的 2 个输出分别代表判定为无关键词出现的概率，和未知关键词出现的概率。

在语音关键词识别的整个流程中，神经网络分类对于最终的识别效果有着决定性作用，并且神经网络的计算量和存储空间在整个系统中占比非常大，因此设计性能优异的神经网络模型及神经网络处理器对于整个系统来说十分重要，本文的主要工作也围绕此展开。

2.3 神经网络基本概念与模型介绍

本节对神经网络基本理论进行介绍，分析了几种神经网络的优缺点，并选择了适合存内计算硬件的神经网络模型。

早期被用来进行时间序列信号处理的机器学习算法为隐马尔可夫模型（Hidden Markov Models，HMMs）[15]，近年来随着人工智能技术的不断发展，基于深度学习的模型展现出了更好的准确度和鲁棒性[16]。

[17]中使用了基于全连接层的深度神经网络进行语音关键词识别，实现了较高的准确度，但是全连接层的形式并不能够有效建立不同输入间在时间上的联系，为了解决这个问题，一些工作提出了用于时间序列信号处理的 CNN[18]和 RNN[19]模型。其中 CNN 使用在时间维度上的卷积来提取贯穿在时间维度上的特征，而典型的 RNN 包含长短期记忆（Long-short Term Memory，LSTM）网络和门控循

环单元（Gated Recurrent Unit，GRU），LSTM 解决了普通 RNN 在长序列训练过程中的梯度消失和梯度爆炸问题，GRU 则是为了解决 LSTM 计算过于复杂而提出。最近提出的深度可分离卷积神经网络（Depthwise Separable Convolution Neural Network，DSCNN）[20]和时间卷积网络（Temporal Convolutional Network，TCN）[21]在准确度和减小计算量方面表现更加出色。下面分别介绍神经网络基本概念和上述典型模型。

基础的神经元模型包括输入信号、权重、偏置、求和部分和激活函数。每个神经元可以看作是处理多个输入信号，并产生单个输出信号的信号处理单元。其中输入信号 X 为$1 \times m$维的向量，即$X = (X_1, X_2, \ldots, X_m)$。权重为$1 \times m$维的向量，$W = (W_{k1}, W_{k2}, \ldots, W_{km})$，此处的 k 表示第 k 个神经元。$\varphi(\cdot)$表示激活函数。$Y_k$表示第 k 个神经元最终的输出。

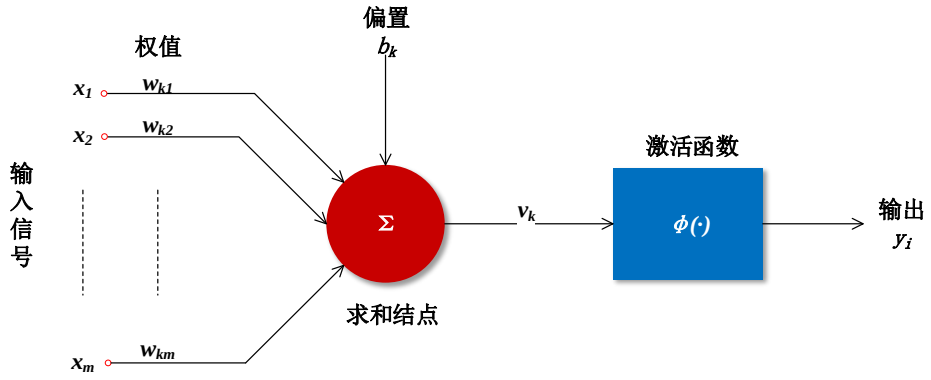


图 2-2 神经元模型

输入向量和权重向量进行向量乘法操作，并加上偏置得到v_k，v_k经过激活函数后得到最终的输出y_k，如下式所示：

$$v_k = \sum_{i=1}^{m} X_i \cdot W_{ik} + B_k \tag{2.1}$$

$$Y_k = \varphi(v_k) \tag{2.2}$$

其中，W_{ik}的大小表明了X_i对于结果的贡献程度，B_k表示第 k 个神经元的偏置。W_{ik}和B_k均为神经元的可训练参数，神经网络的训练即把W_{ik}和B_k调整到最佳，从而提升网络的拟合能力。

卷积层或者全连接层的运算形式本质上是乘加运算，属于线性计算，单纯地叠加线性计算最终得到的还是线性计算。神经网络算法中在每个线性计算层后面

均使用了激活函数来引入非线性，提高网络的非线性拟合能力。常用的激活函数有Tanh函数，Sigmoid函数和Relu函数等，Relu函数具有运算简单，收敛速度快等优点。

全连接网络包含多层，每层由多个神经元组成，并且相邻两层间的每个神经元均存在连接。全连接网络中的第一层称为输入层，中间层称为隐藏层，最后一层称为输出层，全连接网络也被称为多层感知器。如图 2-3 所示为一个典型的 3 层全连接网络，输入为$\mathrm{X}_1,\mathrm{X}_2,\ldots,\mathrm{X}_n$，a、b、c 均为神经元，输出为$\mathrm{Y}_1,\mathrm{Y}_2,\mathrm{X}_3$。每两个神经元之间的连接都对应一个权重$\mathrm{W_{ij}}$，每个神经元包含一个偏置$\mathrm{bias_i}$。

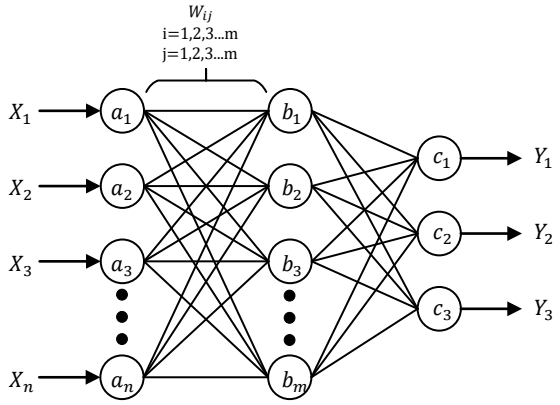


图 2-3 全连接网络

全连接网络中每层的计算形式都为一次向量矩阵乘法，如公式所示。

$$\begin{aligned} b_1 &= W_{11}\cdot a_1 + W_{12}\cdot a_2 + \cdots + W_{1n}\cdot a_n + bias_1 \\ b_2 &= W_{21}\cdot a_1 + W_{22}\cdot a_2 + \cdots + W_{2n}\cdot a_n + bias_2 \\ b_m &= W_{\mathrm{m1}}\cdot a_1 + W_{32}\cdot a_2 + \cdots + W_{\mathrm{mn}}\cdot a_n + bias_m \end{aligned} \tag{2.3}$$

上式可整理为下式所示的矩阵运算

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} W_{11} & W_{12} & W_{13} & \cdots & W_{1n} \\ W_{21} & W_{22} & W_{23} & \cdots & W_{2n} \\ W_{31} & W_{32} & W_{33} & \cdots & W_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ W_{m1} & W_{m2} & W_{m3} & \cdots & W_{mn} \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{bmatrix} + \begin{bmatrix} bias_1 \\ bias_2 \\ bias_3 \\ \vdots \\ bias_m \end{bmatrix} \tag{2.4}$$

全连接层每一层的计算可以看作为一次向量-矩阵乘法，其中向量维度为$1\times\mathrm{m}$，矩阵维度为$\mathrm{m}\times\mathrm{n}$。每个全连接层的乘加计算数量为$\mathrm{m}\times\mathrm{n}$，由于全连接层每个权重均只参与了一次计算，权重没有重用的情况，所以全连接层具有参数量大的特点。全连接层具有优异的分类性能，主要用在深度神经网络的最后几层

实现分类输出。

2.3.1 卷积神经网络

卷积神经网络的核心在于卷积层的使用，不同于全连接层中相邻两层神经元全连接的特性，卷积神经网络中每个神经元均与上一层中的部分神经元进行连接，即局部连接。每个局部连接对应了一个卷积核，卷积核对输入激活进行滑窗卷积操作，实现了权值共享，从而大大减少了卷积层中的参数量。

一个典型的卷积层如图 2-4 所示，图中各符号的意义如下表所示。

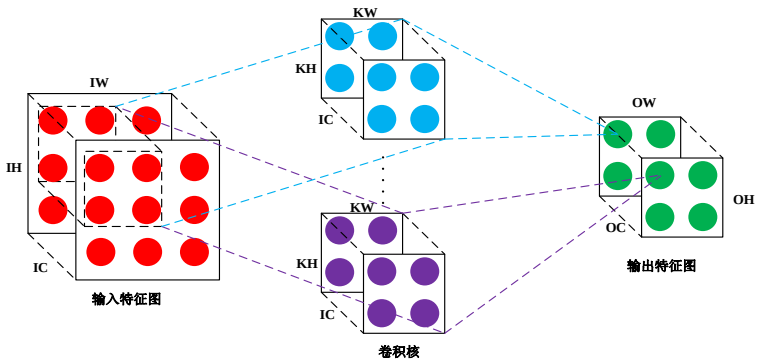


图 2-4 卷积层结构示意图

表格 2-1 卷积层各参数说明表

符号	意义	符号	意义
IW	输入特征图宽度	OW	输出特征图宽度
IH	输入特征图高度	OH	输出特征图高度
IC	输入特征图通道数	OC	输出特征图通道数
KW	卷积核/权重宽度	S	卷积步长
KH	卷积核/权重高度		

其中输入特征图为大小$IW \times IH \times IC$的三维矩阵，权重为大小$KW \times KH \times IC \times OC$的四维矩阵，对应 OC 个维度为$KW \times KH \times IC$的卷积核，输出特征图为大小$OW \times OH \times OC$的三维矩阵。以步长为 1 为例，卷积操作的流程如图 2-5 所示。

Algorithm 1 CNN

```
for OC = 0; OC < oc; oc++ do
  for ic = 0; ic < IC; ic++ do
    for oh = 0; oh < OH; oh++ do
      for ow = 0; ow < OW; ow++ do
        for kh = 0; kh < KH; kh++ do
          for kw = 0; kw < KW; kw++ do
            Y^{oc}_{ow,oh} += X^{ic}_{ow+kw,oh+kh} × w^{oc,ic}_{kw,kh} ;
          end for
        end for
      end for
    end for
  end for
end for
```

图 2-5 卷积层伪代码图

2.3.2 深度可分离卷积神经网络

DSCNN 使用深度可分离卷积层替代了传统的卷积层，每个深度可分离卷积层由一个逐通道卷积层（Depthwise Convolution）和一个逐点卷积层组成（Pointwise Convolution）。DSCNN 中网络模型各参数可由表格 2-1 描述。逐通道卷积层分通道进行卷积，即每个通道上进行二维卷积，经过逐通道卷积层后特征图的通道数不变；逐点卷积层卷积核的宽和高均为 1，对各通道之间的数据进行加权组合。相比传统的三维卷积，深度可分离卷积在运算上更加高效，大幅减少了参数量和计算量。但是由于逐通道卷积层中输入通道数 IC 为 1，不存在输入通道上的累加，即每个输出激活的计算相当于两个$1 \times (KW \times KH)$维度向量的乘法，由于 KW 和 KH 的大小一般较小，常见的 KW 和 KH 均为 3，而存内计算硬件计算的粒度一般较大，因此逐通道卷积层结构难以高效映射到存内计算硬件上。

2.3.3 时间卷积残差神经网络

时间卷积神经网络通过使用膨胀卷积的方法，在计算量没有增加的情况下，增大了感受野[21]。文献[1]中提出了时间卷积残差神经网络 TC-Resnet8，使用了沿时间维度的一维卷积，并结合了残差结构，进一步提高了网络的准确度。文献[1]中提出的 TC-Resnet8 网络结构如图 2-6 所示,其中的参数意义同表格 2-1 所

述。TC-Resnet8 中包含了 1 个一维时间卷积层 conv0，3 个残差层，1 个平均池化层和 1 个全连接层。其中每个残差层中又分为主路径和分支路径，主路径上包含了 2 个一维时间卷积层 conv_1 和 conv_2，分支路径上包含了 1 个一维时间卷积层 conv_3，主路径的结果和分支路径的结果进行元素级相加后产生残差层最终的输出。

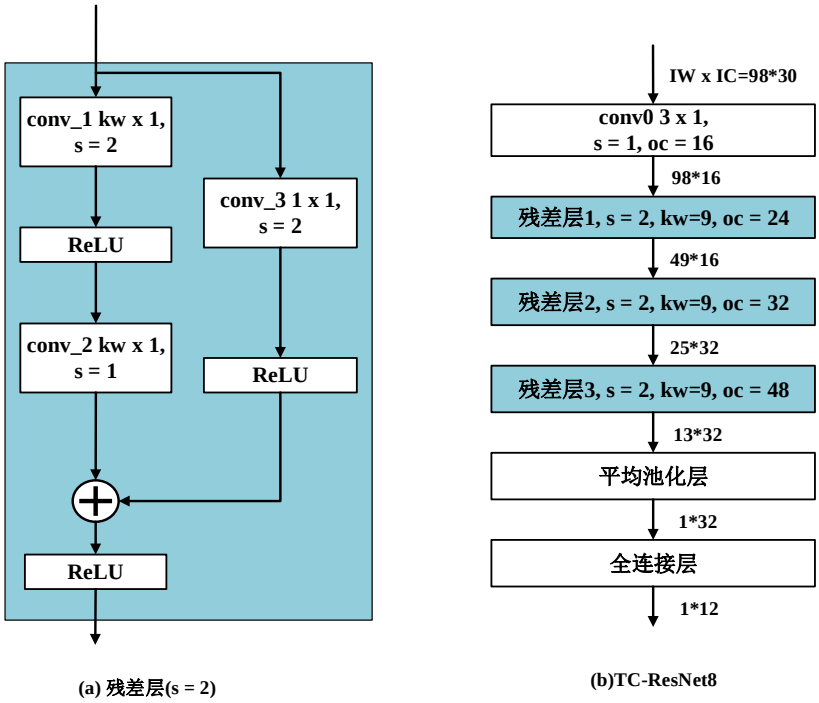


图 2-6 时间卷积残差神经网络 TC-Resnet 结构图

文献[1]中使用的一维卷积结构相比二维卷积可以大幅减少计算量。如图 2-7（a）所示，MFCC 预处理之后的特征图维度为$t \times f$，其中t表示时间维度，f表示特征维度。

传统二维卷积中，把经过 MFCC 预处理之后的特征图当作一个灰度图片来进行处理，如图 2-7 (b)中的二维卷积层包含OC个$3 \times 3 \times 1$大小的卷积核，卷积步长为 1（包含 padding 0，即在特征图边缘进行 0 填充），该二维卷积层中卷积核在时间维度和特征维度进行滑窗卷积操作，输出的特征图大小为$IW \times IC \times OC$。总的 MAC 数为$3 \times 3 \times 1 \times IW \times IC \times OC$。

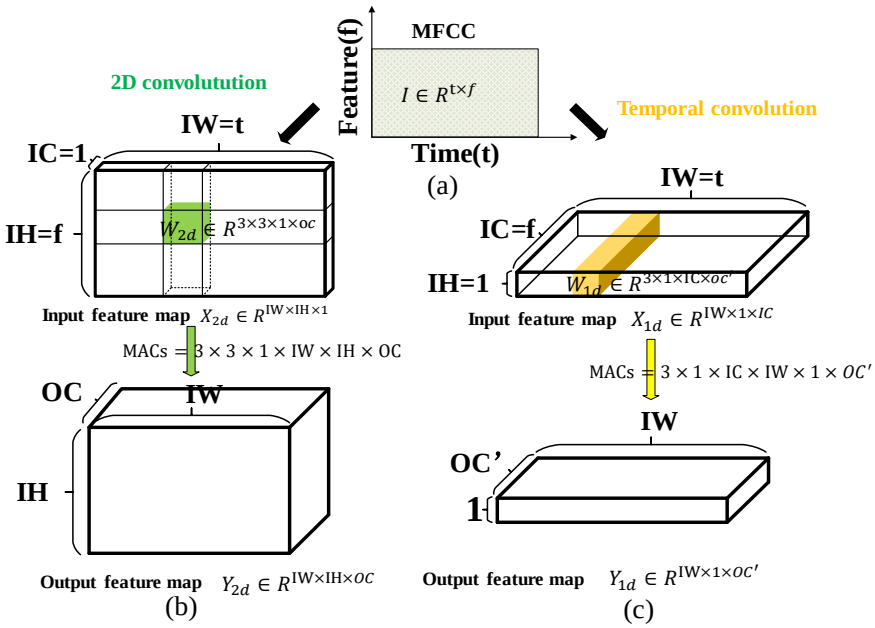


图 2-7 一维卷积二维卷积对比图

一维卷积中把 MFCC 预处理后的每帧特征向量，当作时间序列信号，如图 2-7(c)所示，即特征图分为 IW 帧，每帧的特征向量维度为$1 \times \mathrm{IC}$，整个特征图的维度为$\mathrm{IW} \times 1 \times \mathrm{IC}$。一维卷积层共包含$\mathrm{OC}'$个$3 \times 1 \times \mathrm{IC}$大小的卷积核，卷积步长为 1（包含 padding 0），该一维卷积层中卷积核只在时间维度进行滑窗操作，输出特征图大小为$\mathrm{IW} \times 1 \times \mathrm{OC}'$，总的 MAC 数为$3 \times 1 \times \mathrm{IC} \times \mathrm{IW} \times 1 \times \mathrm{OC}'$。假设二维卷积层和一维卷积层具有相同的参数量，则$\mathrm{OC}' = \frac{3\times OC}{IC}$，进一步得到二维卷积层与一维卷积层的 MAC 数的比例如下式所示

$$\frac{3 \times 3 \times 1 \times \mathrm{IW} \times \mathrm{IC} \times \mathrm{OC}}{3 \times 1 \times \mathrm{IC} \times \mathrm{IW} \times 1 \times \mathrm{OC}'} = \mathrm{IC} \tag{2.5}$$

由此可见，使用一维卷积层后在参数量相同的情况下，相比二维卷积层减少了 IC 倍的计算量。

文献[2]和文献[3]分别把 TC-Resnet 网络结构用于心电信号处理和脑电信号处理中，并取得了优异的性能，展现出 TC-Resnet 网络在序列信号处理中的巨大潜力。

由于 RNN 需要复杂的激活函数，DSCNN 中的逐通道卷积不适合使用存内计算硬件实现，而 TC-Resnet 具有准确度高，结构相对简单的特点，本文选用 TC-Resnet 网络来实现语音关键词识别任务。

2.4 神经网络处理器设计方法

2.4.1 基于存内计算的神经网络处理器设计相关进展

在传统冯诺依曼架构中，运算器和存储器分离，完成计算需要把对应的操作数从存储器中读出，然后送入运算器执行计算，该架构在执行计算的过程中需要对存储器进行频繁访问，由于存储器访问的功耗和延时往往大于计算单元，因此限制了整体的性能。存内计算技术在存储器中实现了计算的功能，大幅减少了存储器访问的次数[5][6]，进而降低了整体的功耗和延时。

在基于存内计算的神经网络处理器架构设计中，通常把存内计算核作为实现向量乘法的矢量运算器，配合数字逻辑电路实现完整的神经网络推理计算，存内计算技术虽然具有高能效和高吞吐量的优势，但是由于单个存内计算核的计算粒度大，并且具有权重固定的特性，因此基于存内计算设计的神经网络处理器往往不够灵活。如文献[9]和文献[10]中基于存内计算设计了专用的语音关键词识别芯片，使用了多个分立的存内计算核，每个存内计算核负责执行神经网络的一层计算，该架构仅适用于运行一个固定的神经网络模型。最近一些工作对提高存内计算灵活性的问题进行了研究，文献[30]和文献[31]中设计了支持存内计算的专用指令集，通过指令把大部分向量乘法运算映射到存内计算核上执行，并配合使用通用标量处理器来执行算法中逻辑相对复杂的部分。文献[32]和文献[33]中结合了片上网络技术（Network on Chip，NOC），其设计中包含了大规模的存内计算核阵列，每个存内计算核位于 NOC 的一个节点上，不同存算核之间的数据流动通过 NOC 中的路由实现，具有很高的灵活性。文献[30][31][32][33]主要出于通用性的角度进行设计，配合通用处理器或者 NOC 方案或者对于追求超低功耗的物联网应用来说存在功耗太高的问题。

在具体的存内计算核设计上，与通常情况一样，根据存内计算所采用的存储器类型可以分为 2 类，即基于易失性存储器的存内计算方案和基于非易失性存储器的存内计算方案。易失性存储器在掉电后数据会丢失，如动态随机存取存储器（Dynamic Random Access Memory，DRAM）和静态随机存取存储器（Static Random-Access Memory，SRAM）等，易失性存储器静态漏电功耗较高，采用该类存储器实现的存内计算方案整体计算能效偏低。非易失性存储器在掉电后数据

不会丢失，如闪存（Flash Memory）、可变电阻式存储器（Resistive Random Access Memory，RRAM）、相变存储器（Phase Change Memory，PCM）等，非易失性存储器漏电功耗低，采用该类存储器实现的存内计算方案适合用于需要长待机的物联网设备中[5]。其中 Flash 存储器具有工艺成熟度高、稳定性好等优点，本文研究的专用处理器中嵌入的存内计算核以此作为存储介质。

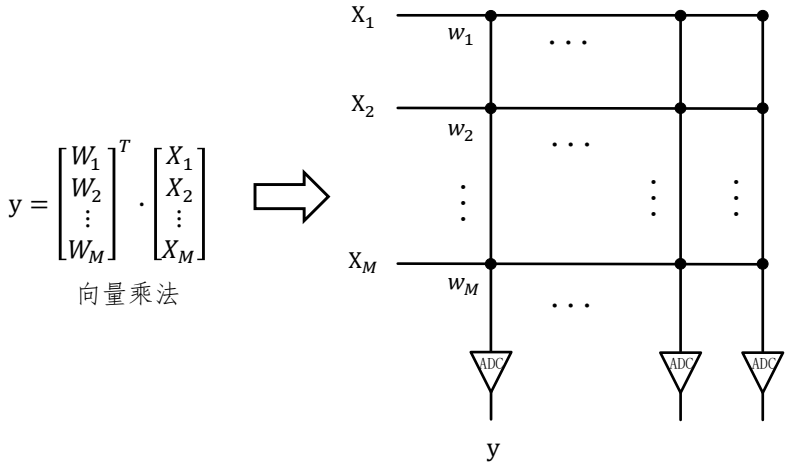


图 2-8 多值存内计算方案

目前基于 Flash 的存内计算技术[37][38][39]大多采用了多值存内计算方案，该方案的特点是单个存储单元存储多个 bit，计算全部在存储器内部实现。典型的电路结构如图 2-8 所示，图中单列存储器支持向量乘法计算，即$y = \sum_{m=0}^{M-1} x_m w_m$，其中向量 W 中的元素以电导的形式存储在 Flash 的存储单元中，向量 X 的元素以电压的形式从字线输入，利用欧姆定律与存储单元中的电导相乘产生电流，位线上每个单元的电流进一步根据基尔霍夫电流定律进行累加产生最终结果。如文献[37]和[38]中设计的 Flash 阵列支持 4bit 的向量乘法，实现了卷积神经网络的运行，文献[39]提出了一种新型 Flash 存储器件，单个存储单元可实现 7bit 数据存储，并实现了多层全连接网络的运行。多值存内计算方案由于单个存储单元可以存储多 bit，因此消耗的 Flash 存储资源较少，但该方案同时存在一些问题，由于每一根字线需要有较复杂的输入转换电路把多 bit 数据转换为电压输入（如多 bit DAC），且每一根位线需要一个 ADC 进行模数转换，使得整体的功耗和面积大幅增加，此外使用单个存储单元存储多 bit 数据时，映射误差较大，影响最终的计算精度。

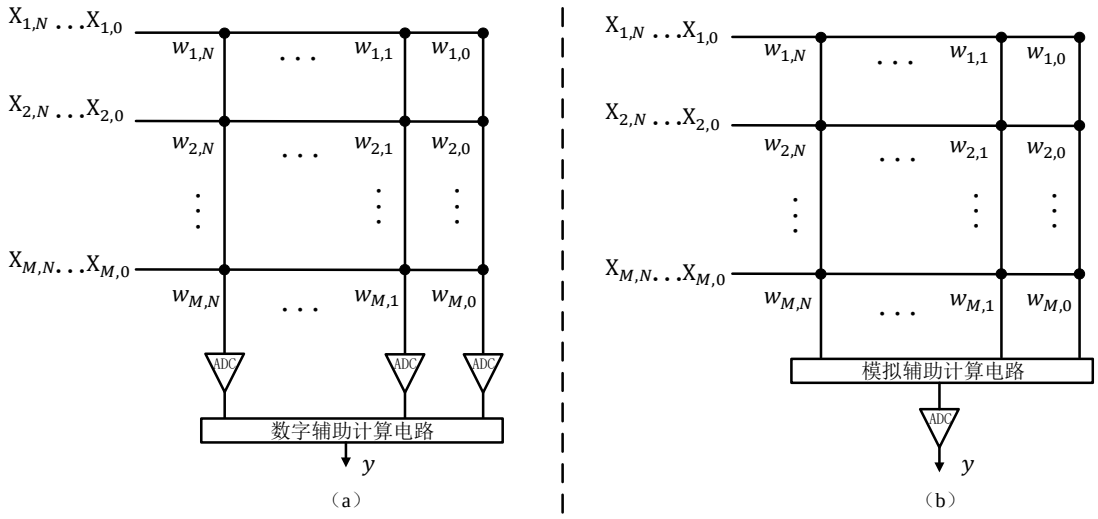


图 2-9 二值存内计算方案

除多值存内计算方案外，一些工作提出了二值存内计算的方案[5][28][40][41]，即单个存储单元存储 1bit，减少了多值存内计算方案中的高映射误差问题。目前该方案大多采用了 RRAM 等存储介质，可以进一步迁移到 Flash 等存储介质上。以完成向量乘法$\mathrm{y}=\sum_{m=0}^{M-1}x_m w_m$为例，其中$x_m$和$w_m$均为 Nbit 无符号整数，$\mathrm{x}_m$的二进制编码为$\mathrm{x_{m,N-1}x_{m,N-2}}\ldots\mathrm{x_{m,0}}(\mathrm{x_{m,i}}\in\{0,1\},\mathrm{i}\in\{0,\ldots,\mathrm{N}-1\})$，$w_m$的二进制编码为$w_{\mathrm{m,N-1}}w_{\mathrm{m,N-2}}\ldots w_{\mathrm{m,0}}(\mathrm{w_{m,j}}\in\{0,1\},\mathrm{j}\in\{0,\ldots,\mathrm{N}-1\})$，对该向量乘法进行格式转换得$\sum_{m=0}^{M-1}x_m w_m=\sum_{\mathrm{i}=0}^{N-1}2^{\mathrm{i}}\sum_{\mathrm{j}=0}^{N-1}2^{j}\sum_{m=0}^{M-1}x_{m,i}\,w_{m,j}$。二值存内计算方案中每个存储单元存储$w_m$中的 1bit，同时每根字线上$x_m$也进行串行输入，每个存储单元实现$x_{m,i}w_{m,j}$的逻辑与运算，即单比特乘法，然后每根位线上电流累加完成部分和$psum$的计算，即$psum=\sum_{m=0}^{M-1}x_{m,i}\,w_{m,j}$。部分和需要进行移位加来实现最终的结果即$\sum_{\mathrm{i}=0}^{N-1}2^{\mathrm{i}}\sum_{\mathrm{j}=0}^{N-1}2^{j}psum$，因此在存储阵列外部需要搭建辅助计算电路。根据辅助计算电路的实现方式又可以分为两种，分别为数字电路[40][41]实现和模拟电路[5][28]实现。数字电路实现方式如图 2-9(a)所示，先对位线上的部分和进行模数转换，然后把转换后的部分和送入数字辅助计算电路完成后续移位加计算。模拟电路实现方式如图 2-9(b)所示，位线上的部分和直接在模拟辅助计算电路中完成移位加计算，然后再进行模数转换，其中模拟辅助计算电路可采用电容间电荷重分配等方式实现[5][28]，该方式中模拟辅助计算电路通常由无源电容阵列等构成，功耗相比数字辅助计算电路更低，本文使用的存内计算核也采用了该方案。

2.4.2 流式推理与分批推理

分批推理和流式推理[27][34][35]为硬件执行神经网络推理计算实现实时序列信号处理的两种不同方式。

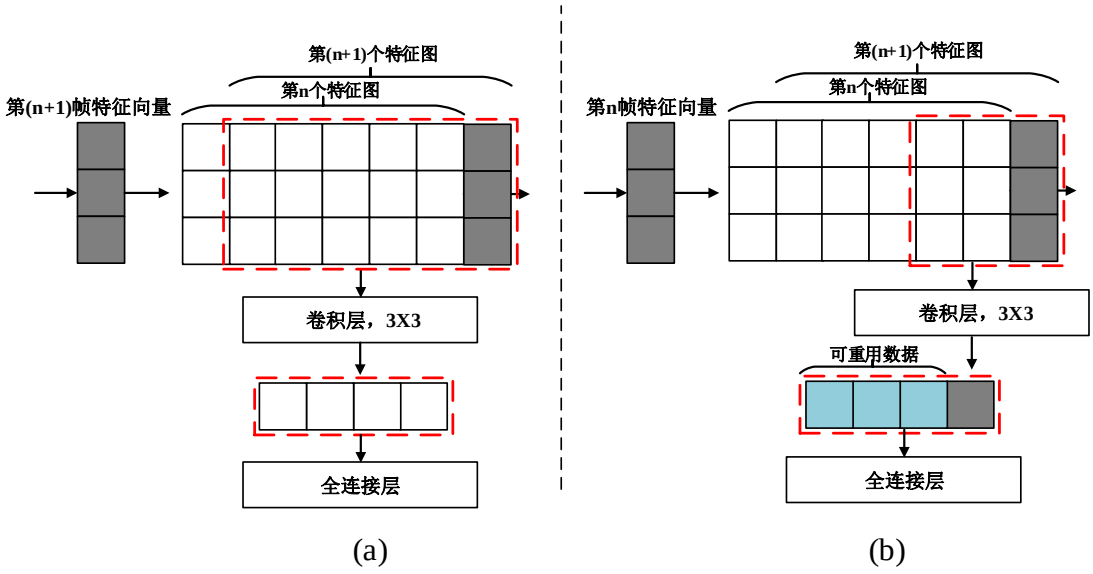


图 2-10 分批推理与流式推理

分批推理指硬件执行神经网络推理计算时，每帧对由多帧特征向量构成的完整特征图进行处理[27][34]。下面以卷积神经网络为例进行说明，如图 2-10(a)所示，该卷积神经网络包含一个卷积层和一个全连接层，其中卷积层为 1 维卷积，输入特征图大小为6×3，卷积核大小为3×3，卷积步长为 1，输出特征图大小为 4×1，该网络每帧都会接收到 1 帧大小为1×3的特征向量。在分批推理中硬件每帧对整个特征图进行计算，即对整个6×3大小的输入特征图进行滑窗卷积，共需计算 4 个卷积窗口，相应产生 4 个输出激活，然后送入全连接层。由于无法提前预知序列信号流中关键信息出现的位置，因此在实时序列信号处理过程中，分批推理每帧需要对包含重叠信息的特征图进行处理，如图 2-10（a）中第 n 个特征图和第（n+1）个特征图重叠部分的大小为 5×3。分批推理是执行神经网络推理计算的传统方式，主流框架如 Pytorch、Tensorflow 等均采用了分批推理的方式执行神经网络推理计算，此外很多支持语音关键词识别的神经网络处理器中采用了分批推理的方式[7][9][11]，如文献[7]实现了对 TC-Resnet 网络的分批推理，文献[9]实现了对 RNN 网络的分批推理，文献[11]实现了对 CNN 网络的分批推理，分批推理的逻辑相对简单，容易实现，但由于没有重用相邻特征图中的重复信息，因此计

算存在冗余。

流式推理指硬件执行神经网络推理计算时，每帧重用之前帧的计算结果，仅对更新的一帧特征向量邻近的部分特征向量进行处理[27][34]。如图 2-10(b)所示，之前帧可重用的数据被缓存下来（如图 2-10(b)中蓝色方块所示），每帧卷积层只进行一个卷积窗口的计算，产生 1 个输出激活，并同之前缓存的 3 个输出激活一起被送入全连接层中进行计算，进而产生推理结果。文献[6]和[8]设计的支持语音关键词识别的神经网络处理器中，均采用了流式推理的方式。其中文献[6]实现了 DSCNN 的流式推理，相比分批推理每帧的计算量减少了 94.3%，得益于每帧计算量减少，在满足实时语音关键词识别的前提下，文献[6]降低了频率和电压，从而进一步降低了功耗。文献[8]中实现了 TCN 的流式推理，由于流式推理显著减少了计算量，因此每帧的计算可以在短时间内快速完成，文献[8]中对每帧剩下的大部分时间使用电源门控技术来降低静态功耗。与分批推理的方法相比，流式推理充分实现了数据重用，大幅减少了实时序列信号处理中神经网络每帧的计算量，进而大幅减小了硬件整体的功耗和每帧推理的延时。

2.5 流式推理优化的 S-TC-Resnet 算法

2.5.1 基本原理

如 2.4.2 节所述，相比分批推理，流式推理大幅减少了每帧的计算量，进而减少了硬件执行神经网络推理计算的延时和功耗，但并不是所有的神经网络模型都适合进行流式推理，如文献[1]中提出的原始的 TC-Resnet 模型，本节对原始 TC-Resnet 模型中阻碍流式推理的结构进行了分析，对其进行了流式推理优化，得到了适用于流式推理的 S-TC-Resnet 模型（下文各符号含义同表格 2-1）。

首先对原 TC-Resnet 模型进行去除了 padding 0。在包含卷积结构的神经网络模型设计中，大量使用了 padding 0 的结构，padding 0 指在输入特征图的边缘进行 0 填充，使用 padding 0 可以提高神经网络捕捉特征图边缘信息的能力，并且使得神经网络中特征图维度的变化更有规律，有利于深层网络的设计。但在流式推理中，包含 padding 0 的卷积窗口的结果不能被重用。如图 2-11 TC-Resnet8 的卷积层 CONV0 中，输入特征图的大小为98×30，包含 98 帧1×30 维度的特

征向量，假设第 n 帧对编号为 1-98 帧的特征向量组成的特征图进行卷积，第(n+1)帧对编号为 2-99 帧的特征向量组成的特征图进行卷积，在存在 padding 0 的情况下，第 n 帧需要计算包含 padding 0 的卷积窗口为（0，1，2）和（97，98，0），第（n+1）帧需要计算包含 padding 0 的卷积窗口为（0，2，3）和（98，99，0），这 2 个卷积窗口的结果均不能重用，而且第(n+1)帧还需要对新的卷积窗口(97，98，99）进行卷积，因此第（n+1）帧进行增量计算需要执行 3 个卷积窗口。此外硬件不仅需要缓存第 97 帧和 98 帧特征向量，还需要缓存第 2 帧和第 3 帧，从而增加了缓存容量，并且需要复杂的控制逻辑，可见 padding 0 不利于流式推理的执行。考虑到 padding 0 结构可以从卷积层中去除，如[6]中设计了不含 padding 0 的卷积神经网络，[45]中设计了不含 padding 0 的时间卷积神经网络，因此本文设计的流式推理优化方法首先去掉了模型中的 padding 0。在去掉 padding 0 后，CONV0 层只需计算（97，98，99）一个卷积窗口，只需缓存第 97 帧和第 98 帧特征向量，从而减少了缓存空间和计算量，简化了控制逻辑。

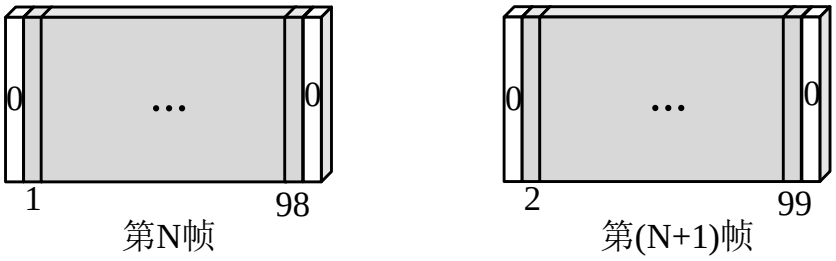


图 2-11 Padding 0 对数据重用的影响图

然后在去除 padding 0 的基础上，进行了卷积核宽度的调整。去掉 padding 0 后，特征图每经过一个卷积核卷积层，特征图的宽度都可能发生变化，即$\mathrm{OW}=\frac{\mathrm{IW}-\mathrm{KW}}{\mathrm{S}}+1$，由于残差层主路径上往往包含 2 个卷积层，且 KW>1，而分支路径上只包含 1 个卷积层，且 KW=1，因此主路径和分支路径的输出特征图维度不同，无法完成元素级相加，如图 2-12 所示。考虑到原始 TC-Resnet 模型并没有对卷积核宽度进行限制[1]，且已有相关工作对 TC-Resnet 模型卷积核宽度的调整进行了探索[44]，因此为了解决维度不同无法相加的问题，本文对卷积核的宽度进行调整，使得残差层主路径和分支路径的输出特征图维度相同，如图 2-12 所示。最终通过去掉 padding 0，并且对卷积核的宽度进行调整，实现了对原始 TC-Resnet 模型的流式推理优化。

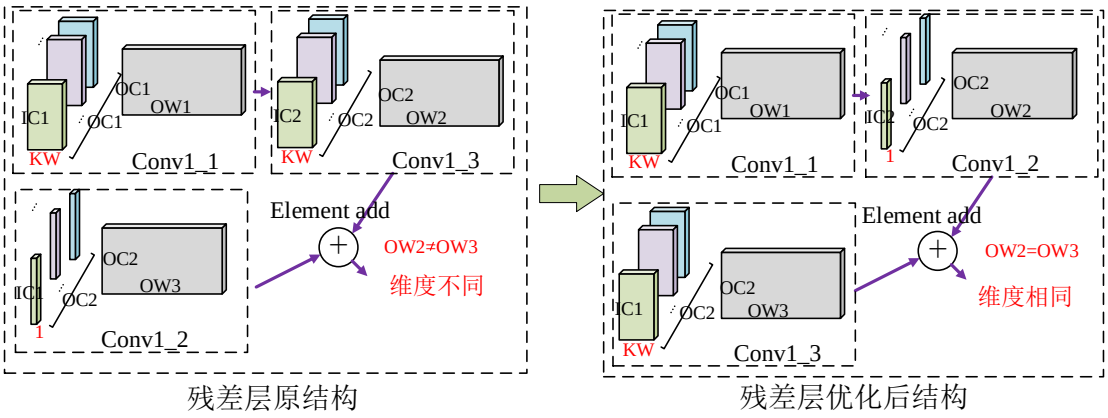


图 2-12 残差层优化前后结构对比图

2.5.2 设计实现

通过使用模型流式推理优化方法，本文在文献[1]中提出的 TC-Resnet 结构的基础上，重新设计了支持流式推理的 S-TC-Resnet 模型。图 2-13(a)、(b)分别为本文所设计的卷积步长为 1 和 2 的两种残差层结构。

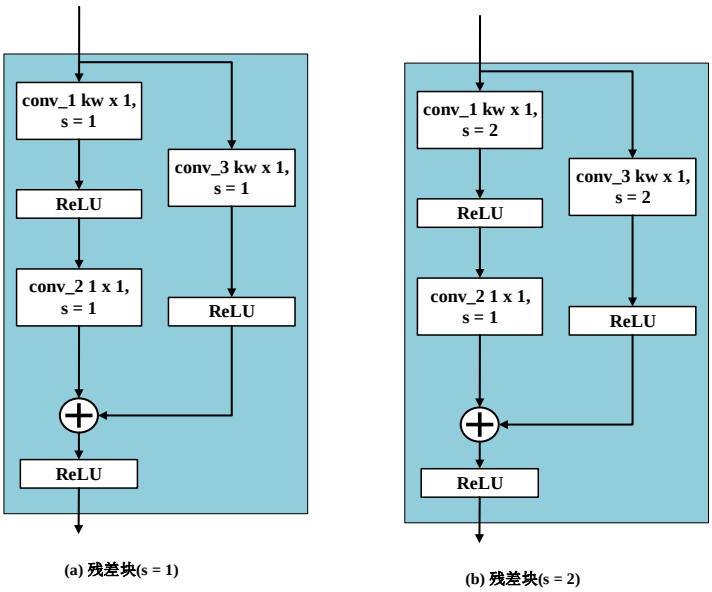


图 2-13 残差层结构

基于上述残差层结构所设计的 S-TC-Resnet 模型结构如下图 2-14 所示（图中输出通道数使用oc_0, oc_1, oc_2, oc_3表示）。图(a)表示 S-TC-Resnet-1 模型，其中卷积层步长全为 1；图(b)表示 S-TC-Resnet-2 模型，其每个残差层中 conv_1 和 conv_3 的卷积步长均为 2，其余卷积层步长为 1。

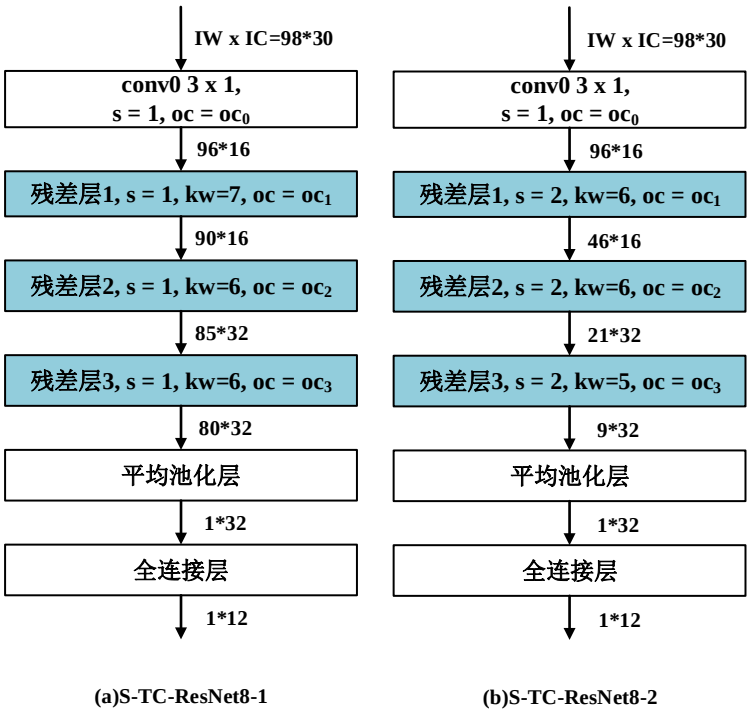


图 2-14 本文设计的支持流式推理的 TC-Resnet 结构图

S-TC-Resnet8-1 模型中卷积步长均为 1，因此可以每帧实现一次推理，而 S-TC-Resnet8-2 主路径上存在 3 个步长为 2 的卷积层，由于每个步长为 2 的卷积层都需要接收前一层 2 帧的结果后才可以完成一次计算，因此步长叠加后每8帧（即 $2\times2\times2$）执行一次推理，推理的间隔更大，计算量减小，但实时语音关键词识别的识别效果会下滑。

2.5.3 实验分析

实验选用了语音关键词识别数据集 GSCD[12]进行测试，该数据集由 Google 在 2017 年公开，包含约 65000 个样本，样本由 Go，Yes，No 等 30 个单词的录音以及一些纯背景噪声组成，每个样本为 1 秒长度的音频，音频的内容为某个单词的录音，该音频由 16KHz 的频率采样得到。本文采用的模型训练和测试方案参考了文献[1]并进行了部分重新设置，代码基于 Python 脚本语言，并使用了 Pytorch、Numpy 等软件库。网络训练相关设置为：（1）网络实现对 12 个类别的分类，包含 10 个关键词“Yes”，“No”，“Up”，“Down”，“Left”，“Right”，“On”，“Off”，“Stop”，“Go”以及无关键词（即静音）和未知关键词；（2）训练集、验证集和测试集的比例为 8：1：1；（3）单个音频样本长度为 1s，对原始音频样本进行 MFCC 预处理，其中帧长为 30ms，帧移为 10ms，MFCC 输出特征维度为

30，即单个音频样本经 MFCC 预处理后输出大小为98×30的特征图，该特征图作为神经网络模型的输入；（4）在预处理过程中随机对 80%的样本添加了噪声，噪声幅度在 0-0.1 之间随机均匀分布；（5）训练的损失函数为交叉熵损失函数，初始学习率为 0.1，训练过程中若累计出现 10 次经过 1 个 epoch 训练后损失函数值无明显下降的情况，则学习率减半。

本文对 2.5.2 中设计的网络模型进行了训练，训练过程中借鉴文献[1]中对 TC-Resnet8 通道数进行调整的思想，对通道数进行了裁剪来减小参数量和计算量，具体设置了 3 个档位的通道数（即oc_0, oc_1, oc_2, oc_3的值）：（1）和原 TC-Resnet8 通道数一致的（16,24,32,48）；（2）出于便于硬件实现的角度，将原通道数裁剪为最邻近的 16 的倍数，即（16,16,32,32）；（3）将通道数进一步全部裁剪到 16，即（16,16,16,16）。

训练过程中网络的 loss 曲线和准确率曲线分别如下图 2-15 和图 2-16 所示，两图中横坐标均为训练的 epoch 数，图 2-15 纵坐标 Loss 代表损失函数值，图 2-16 纵坐标 Accuracy 代表准确率，由图可知经过 130 个 epoch 的训练后，Loss 和 Accuracy 基本稳定，结合学习率最终下降到了 0.003125 左右可得网络收敛。

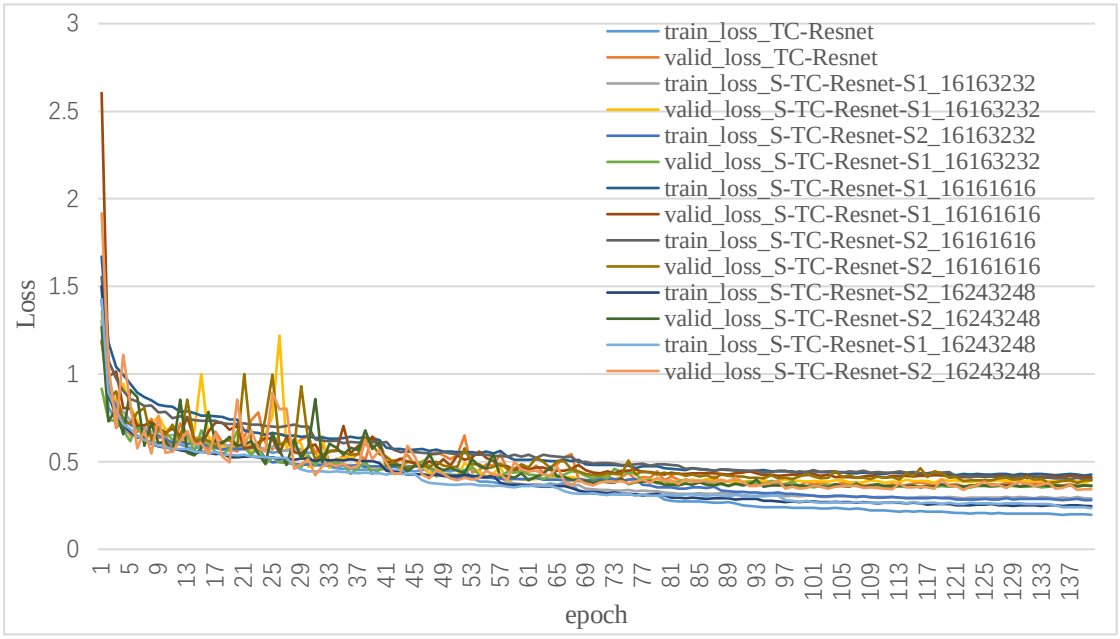


图 2-15 网络训练过程中的 Loss 曲线

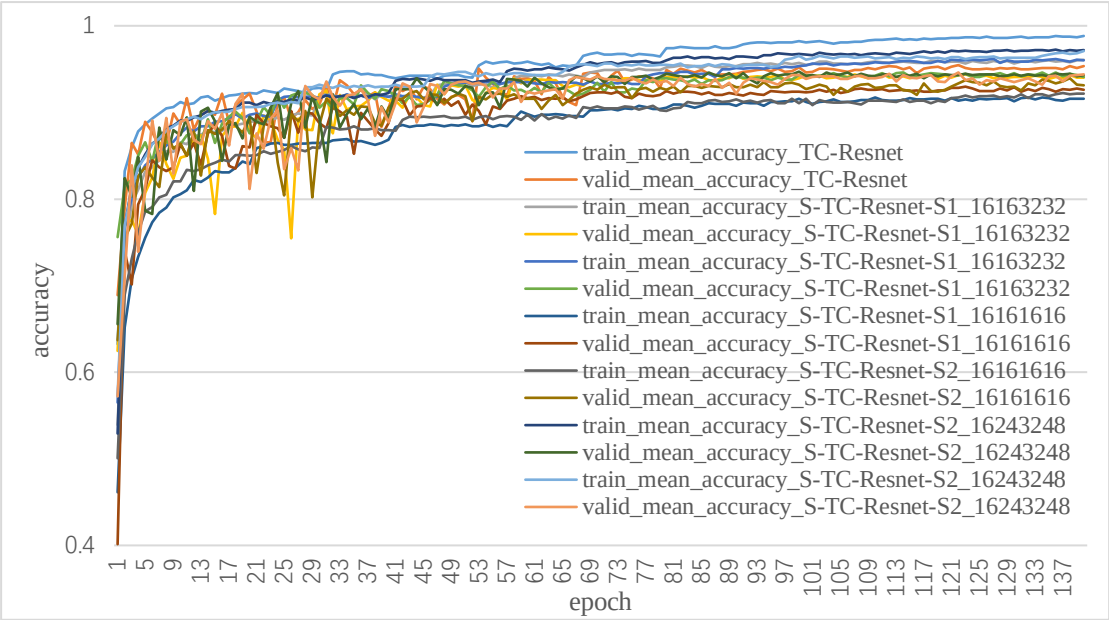


图 2-16 网络训练过程中的 Accuracy 曲线

实验对参数量、计算量和准确度等指标进行了测试，其中参数量表示神经网络模型中参数的个数，参数量越小模型需要的硬件存储资源越少。计算量表示神经网络每次推理计算中需要执行的乘累加计算个数，一次乘累加表示形如$(a \times b + c)$的计算。准确度为语音关键词识别任务中衡量神经网络模型分类能力最常用的指标，表示神经网络模型对测试样本正确分类的比例，假设测试样本总数为 T，其中模型分类正确的个数为 A，则准确度$=\frac{A}{T}$，准确度越高，神经网络模型的分类能力越强[27]。实验结果如下表所示，表中同时列出了原始 TC-Resnet8 进行对比。

表格 2-2 本文设计的 S-TC-Resnet 准确度

网络模型	通道数 oc_1, oc_2, oc_3, oc_4	参数量	分批推理计算量	流式推理计算量	浮点模型准确度
原 TC-Resnet8	16,24,32,48	66k	1.48M	-	95.5%
本文 S-TC-Resnet8-1	16,24,32,48	39K	3.21M	39K	94.8%
	16,16,32,32	26.2K	2.16M	26.2K	94.5%
	16,16,16,16	12.2K	1.04M	12.2K	93.1%
本文 S-TC-Resnet8-2	16,24,32,48	35.2K	0.75M	8.88K	95.2%
	16,16,32,32	23.6k	0.54M	6.36K	94.8%
	16,16,16,16	11.1K	0.39M	4.32K	93.5%

由上表可知，相比原 TC-Resnet，本文设计的 S-TC-Resnet8-1 和 S-TC-Resnet8-

2 模型尽管准确率有所下降，但可以支持流式推理，计算量大幅减少。此外 S-TC-Resnet8-1 和 S-TC-Resnet8-2 在经过通道裁剪后，与原通道数为（16,24,32,48）相比，计算量均明显下降，同时准确度略微下降，由于相关工作[6][7][8]中模型准确度均在 93%以上，为了匹配相关工作的准确度并考虑到后续的量化操作会使得网络准确度略微下降，本文最终选择了通道数为（16,16,32,32）的 S-TC-Resnet8-1 和 S-TC-Resnet8-2 作为本文的目标算法。

2.6 本章小结

本章首先介绍了基于神经网络算法的语音关键词识别总体框架，其次介绍了神经网络算法概念，并对原 TC-Resnet 模型进行了详细描述，然后介绍了神经网络处理器设计方法为下文处理器设计中流式推理实现等做铺垫，最后对原始 TC-Resnet 模型进行了流式推理优化得到 S-TC-Resnet 模型，该模型支持流式推理，通过数据重用大幅减少了计算量。

第3章 面向神经网络量化分布的 ADC 动态缩放

3.1 引言

神经网络量化可以将浮点数模型转换为定点数模型，进而减少神经网络处理器的硬件开销。使用量化后的定点数模型执行神经网络推理计算的过程中，模型每层的原始输出激活需要进行量化后输入下一层，该量化操作需要设计相应的硬件。基于存内计算实现神经网络推理的硬件设计中，往往在模拟信号域内完成原始输出激活的计算，然后使用 ADC 执行量化，出于减小功耗和面积的角度，通常使用较低位宽的 ADC[42][43]。由于输出激活在一定范围内动态分布，该范围超出了低位宽 ADC 的量程，因此低位宽 ADC 进行量化时量程利用率较低，从而产生了量化误差。本章将围绕如何充分利用低位宽 ADC 的量程，在保证量化准确度的前提下，降低 ADC 的硬件开销展开，首先介绍了神经网络模型量化和基于 Flash 的存内计算的基本原理，然后对 ADC 实现量化的过程进行了分析，并设计了动态缩放方法来提高 ADC 量程利用率，最后以 S-TC-Resnet 模型为测试例进行了实验验证。

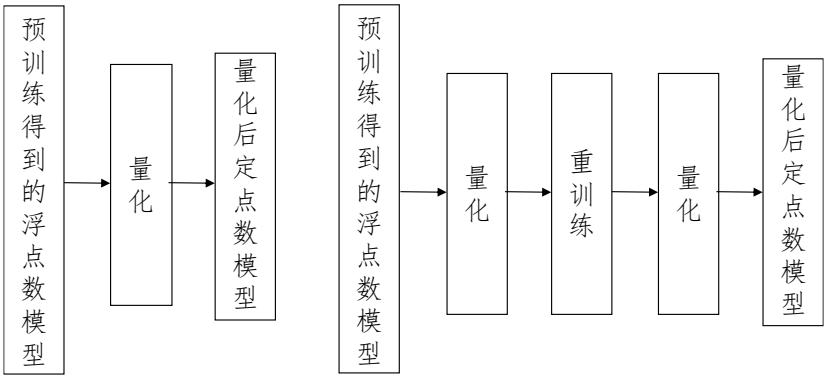
3.2 基本原理

3.2.1 神经网络模型量化

通常在主流框架上直接训练得到的神经网络模型的数据类型为 32bit 单精度浮点数，完成单精度浮点数运算需要消耗大量的存储和计算资源，限制了神经网络处理器的性能。神经网络量化可以在保证网络准确度几乎不变的前提下，把浮点数表示的激活和权重量化为低位宽定点数（如 8bit 或更低）[22][23][26]，从而降低神经网络模型需要的存储空间，减小计算功耗，大幅提升硬件执行神经网络推理计算的性能。

神经网络模型量化的方式可以分为训练后量化（Post Trainning Quantization，PTQ）和量化感知训练（Quantization Aware Training，QAT），两种方式的区别在于量化后是否进行重训练[7]，两种方式的过程如下图所示。训练后量化直接对预训练得到的浮点模型进行量化，由于量化会产生信息损失，因此该方法会使得网络准确度明显下降。量化感知训练在量化后进行了重训练，使网络参数可以更好

地适应量化带来的信息损失，量化后网络的准确度可以保持基本不变或者轻微下降，为了尽量减少量化带来的准确度损失，本文采用了量化感知训练的方式。



(a) 训练后量化 (b) 量化感知训练

图 3-1 训练后量化与量化感知训练流程

量化感知训练中具体的量化方法又可以分为非线性量化和线性量化[24]，由于非线性量化通常需要特殊的硬件支持[25]，因此本文选择了更通用的线性量化，其过程可以用下式表示：

$$q = clamp(Round\big(S(f - Z)\big), min, \max) \tag{3.1}$$

其中f为量化前的浮点数，Z 为浮点数 0 量化后的值，Z 为 0 时称为对称线性量化，Z 不为 0 时称为非对称线性量化[24]，q为量化后 n bit 位宽的定点数。S为缩放因子，S与$(f - Z)$相乘表示对$(f - Z)$进行放大或缩小操作，$Round(\cdot)$为舍入取整函数，$clamp(\cdot)$为截断函数，表示把输入值截断到[min,max]区间内，min 表示 n bit 定点数可表示的最小值，max 表示 n bit 定点数可表示的最大值，当量化到 nbit 有符号数时，$\min = -2^{n-1}$,$\max = 2^{n-1} - 1$；当量化到 nbit 无符号数时，$\min = 0$，$\max = 2^{n} - 1$。对于线性量化，量化后的比特数越低，网络的准确度下降幅度越大，通常保证网络准确度没有明显下降的最低位宽要求在 8 比特附近[25]。量化过程中的缩放因子 S 由量化前数据 f 的范围和量化后数据 q 的范围决定。若量化前数据 f 范围为$[\min_f, \max_f]$，量化后数据 q 的范围为$[\min_q, \max_q]$，则理论上$S = (\max_q - \min_q)/(\max_f - \min_f)$，由该式可见数据的分布情况决定了缩放因子的大小，同时缩放因子表征了数据的分布情况。

模型完成量化后，会得到量化后的低位宽参数和每层激活的缩放因子S_q，如图 3-2 所示。使用量化后定点数模型执行推理计算的过程中，参数和输入激活以

低位宽参与神经网络某层的计算，在计算过程中由于乘累加产生进位，因此该层的原始输出激活为高位宽，该高位宽原始输出激活需要根据缩放因子从高位宽量化到低位宽，然后输入下一层，对应公式(3.1)。出于便于硬件实现的角度考虑，每层输出激活的缩放因子S_q限制为 2 的幂次，即$S_q = 2^{-n}$，其中 n 为自然数。由于神经网络模型各层输出激活的数据分布存在差异，即各层的缩放因子在一定范围内动态分布，因此需要设计支持动态缩放的硬件来实现输出激活的量化。

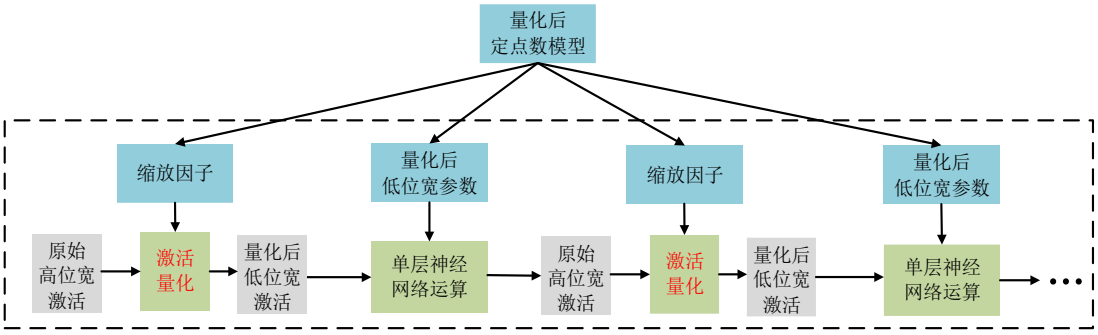


图 3-2 量化后神经网络模型推理计算

3.2.2 基于 Flash 的存内计算

论文采用的存内计算核中的 Flash 存储器由上海华力微电子有限公司以 IP 的形式提供，类型为 40nm SONOS 工艺的 NOR Flash 存储器（大小为256×2048，即 64KB），该 IP 包括 Flash 存储器阵列、列译码器和行译码器等。在该 IP 的基础上由课题组其他成员对其进行了修改，改动包括引出了所有的字线，使得可以并行访问所有字线，并在位线端口增加了接口电路和辅助计算电路，配合存储器阵列实现存内计算功能。下面首先介绍存内计算核执行单比特乘法和向量乘法的原理，在此基础上阐述存内计算核执行神经网络算法的原理。

（1）存储单元结构及执行单比特乘法原理。SONOS 工艺下的 NOR Flash 单个存储单元包含浮栅管和选通管 2 个晶体管，如图 3-3 所示。每个存储单元包含 4 个端口，分别为控制栅端口（Control Gate，CG），字线端口（Word Line，WL），位线端口（Bit Line，BL），源线端口（Source Line，SL）。

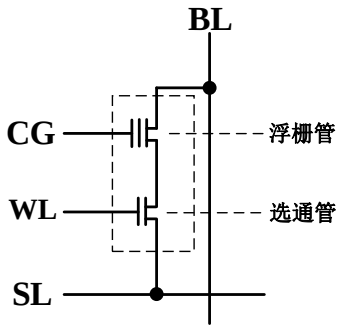


图 3-3 SONOS 工艺下 Flash 单个存储单元结构图

单个存储单元有三种操作方式：擦除、编程和读取。擦除模式时，存储单元的 CG 端接负高压，WL 接正压，BL 和 SL 接正高压，擦除后浮栅管中浮栅上的电子进入衬底，阈值电压减小，等效电阻为低阻，等效电导较大，读出电流变大，相当于存储单比特“1”。编程模式下，选中的存储单元的 CG 接正高压，WL、BL 和 CSL 接负高压，编程后浮栅管中浮栅上注入了电子，阈值电压增大，等效电阻为高阻，等效电导较小，读出电流减小，相当于存储单比特逻辑“0”。在读取模式下，SL 接地，BL 接正压，CG 接正压，WL 输入高电平时，选通管导通，若浮栅管为低阻状态（对应存储逻辑“1”），则整个存储单元导通，位线端口有电流通过；当浮栅管为高阻状态（对应存储逻辑“0”），则存储单元关断，电流几乎为 0，通过测量位线上的电流可以读取存储单元中的值。读取模式下把 WL 输入的高/低电平作为一个单比特乘数，把存储单元中存储的值作为另一个单比特乘数，则每个存储单元可视为完成了单比特乘法操作。

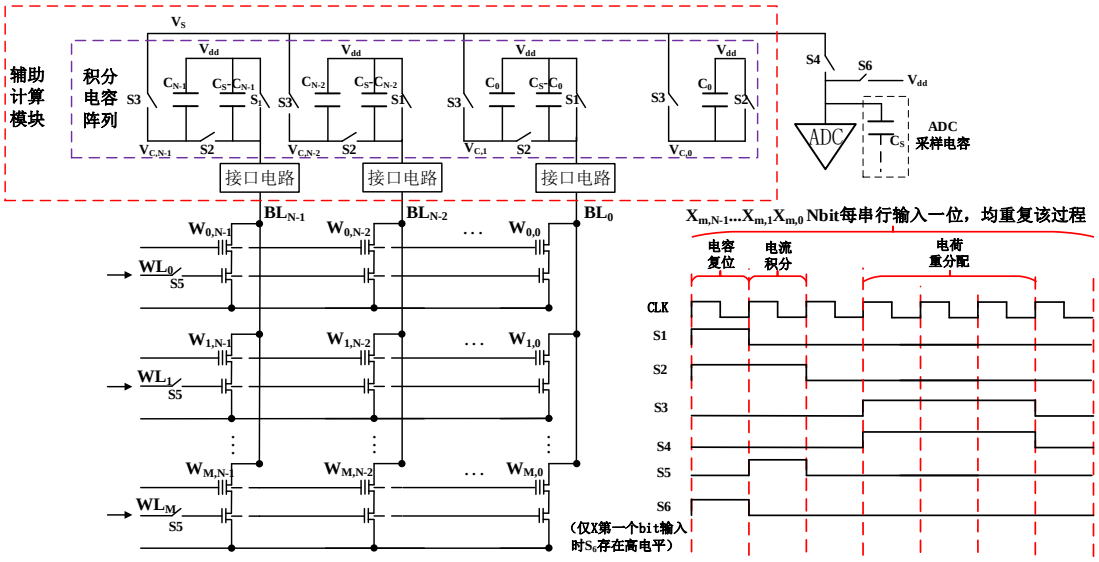


图 3-4 单通道存内计算核电路图及控制波形图

（2）执行向量乘法原理。存内计算核电路基于电流积分和电荷重分配的原理实现向量乘法，一个单通道存内计算核电路如图 3-4 所示，包含 Flash 存储器阵列、辅助计算模块和 1 个逐次逼近式模数转换器（Successive-approximation register ADC，SAR ADC）。Flash 存储器阵列大小为$\mathrm{M}\times\mathrm{N}$，共 M 根字线和 N 根位线。辅助计算模块包含 N 个接口电路、积分电容阵列、传输门开关 S1-S6。积分电容阵列包含多个积分电容，图 3-4 中每根位线经接口电路与 2 个积分电容相连，两电容之和为C_f，如图 3-4 中位线BL_{N-1}所连的积分电路中两个电容大小分别为C_{N-1}和$(\mathrm{C}_f-\mathrm{C}_{N-1})$，积分电容阵列中的电容之比$\mathrm{C}_{N-1}:\mathrm{C}_{N-2}:\ldots:\mathrm{C}_0=2^{N-1}:2^{N-2}:\ldots:1$，其中$\mathrm{C}_0=2^{-N}C_f$，$\mathrm{C}_{N-1}+\mathrm{C}_{N-2}+\cdots+\mathrm{C}_0+\mathrm{C}_0=\mathrm{C}_f$。图中$\mathrm{C}_s$为 ADC 的采样电容（该电容实际在 ADC 内部），且$\mathrm{C}_s=\mathrm{C}_f$。

单通道存内计算核支持向量乘法计算，如下式所示，其中 X 和 W 均为$1\times\mathrm{M}$维向量，x_m和w_m为 Nbit 无符号数，x_m的二进制编码为$\mathrm{x}_{\mathrm{m,N-1}}\mathrm{x}_{\mathrm{m,N-2}}\cdots\mathrm{x}_{\mathrm{m,0}}(\mathrm{x}_{\mathrm{m,i}}\in\{0,1\},\mathrm{i}\in\{0,\ldots,\mathrm{N}-1\})$，$W_m$的二进制编码为$w_{\mathrm{m,N-1}}w_{\mathrm{m,N-2}}\cdots w_{\mathrm{m,0}}(\mathrm{w}_{\mathrm{m,j}}\in\{0,1\},\mathrm{j}\in\{0,\ldots,\mathrm{N}-1\})$，$\mathrm{Q}(\cdot)$为量化函数；$\mathrm{y}'$为原始输出，y 为经过 ADC 量化后的最终输出。

$$\begin{aligned}y=Q(y')=Q(X\times W)&=\mathrm{Q}(\textstyle\sum_{m=0}^{M-1}x_m w_m)\\&=\mathrm{Q}\big(\textstyle\sum_{m=0}^{M-1}\sum_{\mathrm{i}=0}^{N-1}2^i x_{m,i}\sum_{\mathrm{j}=0}^{N-1}2^j w_{m,j}\big)\\&=\mathrm{Q}\big(\textstyle\sum_{\mathrm{i}=0}^{N-1}2^{\mathrm{i}}\sum_{\mathrm{j}=0}^{N-1}2^j\sum_{m=0}^{M-1}x_{m,i}\,w_{m,j}\big)\end{aligned}\tag{3.2}$$

使用存内计算核执行向量乘法计算首先需要通过擦除和编程操作将向量 W 存储在阵列中，存储方式为向量 W 的 M 个元素W_m分别存储在 Flash 阵列的 M 行中，每行的 N 个存储单元分别存储W_m二进制编码(Nbit)中的一位，对应高阻状态表示数字“0”，低阻状态表示数字“1”。向量 X 的 M 个元素X_m分别从 M 行字线上并行输入阵列，每一行字线上 Nbit 输入数据X_m从最低有效位（Least Significant Bit，LSB）到最高有效位（Most Significant Bit，MSB）依次串行输入，每 bit 串行输入过程中对应的开关 S1-S6 波形如图 3-4 所示，共包含电容复位、电流积分和电荷重分配 3 个阶段，下面分别对 3 个阶段展开介绍。

在电容复位阶段 S1、S2 和 S6 闭合，其余开关断开，如图 3-4 所示，积分电

容阵列中的电容和 ADC 采样电容两端电压复位为$\mathrm{V_{dd}}$，为后续计算做准备。

在电流积分阶段实现的运算逻辑为单比特乘累加$\mathrm{psum}=\sum_{m=0}^{M-1}x_{m,i}\,w_{m,j}$，其中每个存储单元完成单比特乘法$x_{m,i}w_{m,j}$，通过位线上的电流积分进一步实现$\sum_{m=0}^{M-1}x_{m,i}\,w_{m,j}$。具体的过程为在电流积分阶段 S2 和 S5 闭合，其余开关断开，如图 3-4 所示，以第 j 列字线上的电容$\mathrm{C_j}$为例，积分后该电容电压如公式(3.3)所示，其中$\mathrm{V_{dd}}$为积分前电容$\mathrm{C_j}$上的电压，$x_{m,i}$为第 m 行字线的第 i bit 输入，M为存储阵列字线数量，T为积分时间（对应图 3-4 中 S5 高电平持续的时间），I_{DS}为单个存储单元的读出电流，若$w_{m,j}$为逻辑 1，则该单元阈值电压较小，I_{DS}较大，若$w_{m,j}$为逻辑 0，则该单元阈值电压较大，I_{DS}几乎为 0，即I_{DS}反映了$w_{m,j}$的值。

$$V_{c,\mathrm{j}}=\mathrm{V_{dd}}-\frac{I_{DS}\sum_{\mathrm{m=0}}^{\mathrm{M-1}}x_{m,i}\,T}{C_f} \tag{3.3}$$

在电荷重分配阶段，对电流积分后产生的单比特乘累加结果进行移位加，即$\sum_{\mathrm{i=0}}^{N-1}2^{\mathrm{i}}\sum_{\mathrm{j=0}}^{N-1}2^{\mathrm{j}}\mathrm{psum}$，该计算可以进一步拆分为 2 种形式的移位加，一种为w_m不同比特的权重2^{j}产生的移位加（即$\sum_{\mathrm{j=0}}^{N-1}2^{\mathrm{j}}\ldots$），另一种为$x_m$不同比特的权重$2^{\mathrm{i}}$产生的移位加(即$\sum_{\mathrm{i=0}}^{N-1}2^{\mathrm{i}}\ldots$)。电荷重分配阶段开关 S3、S4 闭合，其余开关断开，出于更清晰描述计算原理的角度，假设 S3 先闭合，然后 S4 再闭合。如图 3-4 所示，开关 S3 先闭合后，各位线上的积分电容并联，并联电容总大小为$C_{\mathrm{N-1}}+C_{\mathrm{N-2}}+\cdots+2C_0=C_f$，S3 闭合前后积分电容电荷重分配，积分电压$V_S$的初始值为$\mathrm{V_{dd}}$，在完成第 Nbit 输入的积分和积分电容电荷重分配后电压V_S为：

$$\begin{aligned}V_{S,i}&=\frac{V_{c,\mathrm{N-1}}C_{\mathrm{N-1}}+V_{c,\mathrm{N-2}}C_{\mathrm{N-2}}+\cdots+V_{c,0}C_0+\mathrm{V_{dd}}C_0}{C_{\mathrm{N-1}}+C_{\mathrm{N-2}}+\cdots+2C_0}\\&=\mathrm{V_{dd}}-\frac{I_{DS}T}{C_f}\left(2^{-1}\sum_{\mathrm{m=0}}^{\mathrm{M-1}}x_{\mathrm{m,N-1}}+2^{-2}\sum_{\mathrm{m=0}}^{\mathrm{M-1}}x_{\mathrm{m,N-2}}+\cdots+2^{-\mathrm{N}}x_{\mathrm{m,0}}\right)\end{aligned} \tag{3.4}$$

积分电容电荷重分配前后电压V_S的变化值$\triangle V_{S,i}$如下式所示，由前文可知$\mathrm{I_{DS}}$即反映了$w_{m,j}$的值，所以下式可视为$\sum_{\mathrm{j=0}}^{N-1}2^{j}\sum_{m=0}^{M-1}x_{m,i}\,w_{m,j}$的运算结果。

$$\triangle V_{S,i}=\mathrm{V_{dd}}-V_{S,i}=2^{-\mathrm{N}}\sum_{j=0}^{N-1}2^{n}\sum_{m=0}^{M-1}x_{m,i}\frac{I_{DS}T}{C_f}\propto\sum_{\mathrm{j=0}}^{N-1}2^{j}\sum_{m=0}^{M-1}x_{m,i}\,w_{m,j} \tag{3.5}$$

在开关 S4 闭合后，SAR ADC 的采样电容C_s与积分电容阵列并联，并进行电荷重分配，实现进一步的移位加。由于$\mathrm{C}_s=\mathrm{C}_f=\mathrm{C}_{N-1}+\mathrm{C}_{N-2}+\cdots+\mathrm{C}_0+\mathrm{C}_0$，所

以进行一次电荷重分配后新的电容电压V_{out}为：

$$V_{out} = 2^{-1}(V_S + V_{out}^{-}) \tag{3.6}$$

其中V_{out}^{-}表示采样电容C_s每次电荷重分配前的电压。在第一次电荷重分配前，C_s的初始电压为V_{dd}，字线上 Nbit x_m分 N 次串行输入，每次均经过一次电荷重分配，经过 N 次电荷重分配后$V_{out} = \left(2^{-N}V_{dd} + 2^{-N}V_{S,0} + \cdots + 2^{-1}V_{S,N-1}\right)$,电压变化值$\triangle V_{out}$如下公式所示：

$$\begin{aligned} \triangle V_{out} &= V_{init} - \left(2^{-N}V_{dd} + 2^{-N}V_{S,0} + \cdots + 2^{-1}V_{S,N-1}\right) \\ &= 2^{-N}\left[\left(V_{init} - V_{S,0}\right) + \cdots + 2^{N-1}\left(V_{init} - V_{S,N-1}\right)\right] \\ &= 2^{-N}\sum_{i=0}^{N-1} 2^{i} \triangle V_{S,n} \end{aligned} \tag{3.7}$$

根据公式 3.5，$\triangle V_{S,n} \propto \sum_{j=0}^{N-1} 2^{j} \sum_{m=0}^{M-1} x_{m,n} w_{m,j}$，因此$\triangle V_{out} \propto \sum_{i=0}^{N-1} 2^{i} \sum_{j=0}^{N-1} 2^{j} \sum_{m=0}^{M-1} x_{m,i} w_{m,j} = \sum_{m=0}^{M-1} x_m w_m$，从而实现了向量 X 和 W 的乘法操作，得到了原始输出y'，进一步使用 ADC 进行模数转换，完成模拟信号的量化，从而输出最终的数字值 y。

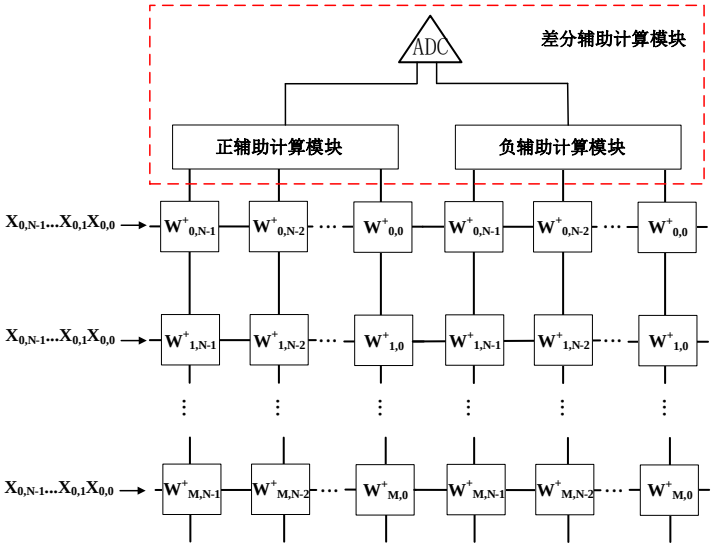


图 3-5 支持有符号数向量乘法的存内计算核

上文介绍的向量乘法中x_m和w_m为无符号数，当w_m为有符号数时，可以对w_m使用差分编码，即正编码为$w_{m,N-1}^{+} w^{+}_{m,N-2} \cdots w_{m,0}^{+}$，负编码为$w_{m,N-1}^{-} w_{m,N-2}^{-} \cdots w_{m,0}^{-} (w_{m,j}^{+} \in \{0,1\}, w_{m,j}^{-} \in \{0,1\}, j \in \{0, \ldots, N-1\})$，$w_m = w_m^{+} - w_m^{-} = \sum_{j=0}^{N-1} 2^{j} w_{m,j}^{+} - \sum_{j=0}^{N-1} 2^{j} w_{m,j}^{-}$，并将图 3-4 中的存储阵列和辅助计算模块复

制一份，如图 3-5 所示，其中正辅助计算模块和负辅助计算模块分别用于计算 $\sum_{m=0}^{M-1} x_m w_m^+$ 和 $\sum_{m=0}^{M-1} x_m w_m^-$，并进一步通过差分 ADC 实现 $\sum_{m=0}^{M-1} x_m w_m = \sum_{m=0}^{M-1} x_m w_m^+ - \sum_{m=0}^{M-1} x_m w_m^-$。

接下来介绍使用存内计算核运行神经网络推理计算的原理，如图 3-2 所示，在量化后的神经网络执行推理计算过程中，主要包含单层神经网络运算和激活量化两种操作，其中单层神经网络运算中大多数计算可以分解为输入激活向量 X 和权重向量 W 的向量乘法[25][28]，权重向量 W 中的元素为有符号数，激活向量 X 中的元素为无符号数，可以将该运算映射到支持有符号数向量乘法的存内计算核上执行，同时原始输出激活的量化由存内计算核中的 ADC 实现。

3.3 设计实现

3.3.1 S-TC-Resnet 模型量化

采用 3.2.1 节中所述的量化感知训练方式，对 2.5 节预训练得到单精度浮点数模型 S-TC-Resnet-1 和 S-TC-Resnet-2 进行对称线性量化，受限于代工厂提供的 Flash 存储器阵列大小，本文对网络进行了 6bit 量化，量化结果如下表所示：

表格 3-1 量化前后模型准确度

网络	数据集	量化前准确度	量化后准确度
S-TC-Resnet8-1	GSCD	94.5%	94.2%
S-TC-Resnet8-2	GSCD	94.8%	94.0%

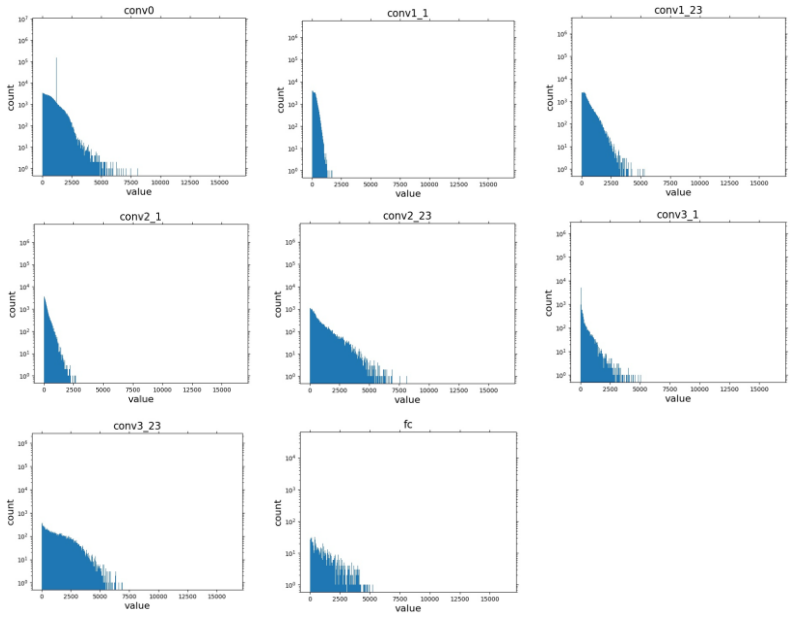


图 3-6 S-TC-Resnet-2 各层输出激活分布图

以S-TC-Resnet-2为例，该网络每层原始输出激活分布如上图所示，上图中横轴为原始输出激活的值，纵轴为出现的频次，由上图可见不同层输出激活的分布情况存在差异。S-TC-Resnet-1和S-TC-Resnet-2模型各层激活具体的缩放因子S_q在一定范围内分布，$S_q \in [2^{-7}, 2^{-4}]$，如下表所示：

表格 3-2 S-TC-Resnet-2 各层量化因子

层名称	缩放因子S_q	
	S-TC-Resnet-1	S-TC-Resnet-2
conv0	2^{-7}	2^{-7}
conv1_1	2^{-4}	2^{-4}
conv1_23	2^{-7}	2^{-6}
conv2_1	2^{-4}	2^{-4}
conv2_23	2^{-6}	2^{-7}
conv3_1	2^{-5}	2^{-4}
conv3_23	2^{-7}	2^{-7}
fc	2^{-6}	2^{-6}

3.3.2 动态缩放可行性分析

本节以存内计算核包含256行字线，并对神经网络采用6bit低位宽量化为例对ADC的量化过程进行分析。激活向量X和权重向量W的乘法过程中，需要执行256组2个6bit数的乘法操作得到256个乘积，然后对256个乘积进行累加，因此原始输出激活y'的最大位宽理论上为19bit，计算如下公式所示(减1表示由于采用了Relu激活函数，所以输出结果的值全为非负数，减去1比特符号位占用的空间)：

$$6 + 6 + \log_2 256 - 1 = 19\text{bit} \tag{3.8}$$

原始的19bit高位宽输出激活y'需要重新量化为6bit输出激活y，然后作为下一层的输入。根据缩放因子$S_q = 2^{-n}$，量化过程为对19bit的原始输出激活取中间的6bit，即对低 nbit进行四舍五入，对高$(13-n)$bit进行截断操作，如图 3-7所示。截断操作是指若高$(13-n)$bit中存在1，表示要量化的值超出了6bit所能表示的最大范围，则最终的量化结果取6bit能表示的最大值，即$6'b111111$。

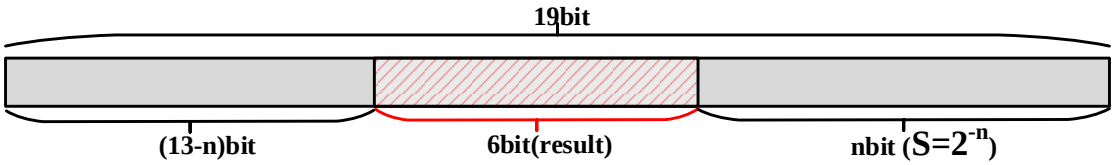


图 3-7 神经网络激活量化示意图

SAR ADC 的位数越高，量程越大，量化的准确度越高，但同时 ADC 的功耗和面积越大。SAR ADC 的单次模数转换的能耗如下式所示，其中 n 表示 ADC 的位数，C_0表示单位电容，V_{ref}表示参考电压[36]。

$$E = \sum_{i=1}^{n-1}\left(2^{n-2-i}\right)CV_{ref}^2 = \sum_{i=1}^{n-1}\left(2^{n-2-i}\right)E_0 \tag{3.9}$$

根据上式，随着 SAR ADC 的位宽升高，功耗成指数增加，如图 3-8 所示。

SAR ADC 面积和位宽的关系如下式所示，其中 n 表示 ADC 的位数，A_0表示单位电容C_0的面积，根据下式，随着位宽升高，SAR ADC 的面积成指数增加，如图 3-8 所示。此外如 3.2.2 节所述，在本文采用的存内计算核中，电容阵列的总电容和 ADC 采样电容间需要满足一定的比例，当 ADC 的电容大小增加后，电容阵列中的电容大小也需要进行同比例增加，而电容面积在整个存内计算核中占主导低位，因此 ADC 位数增加后会显著增加整体的面积。

$$A = \left(1 + \sum_{i=1}^{n-1}\left(2^{n-1-i}\right)\right)A_0 \tag{3.10}$$

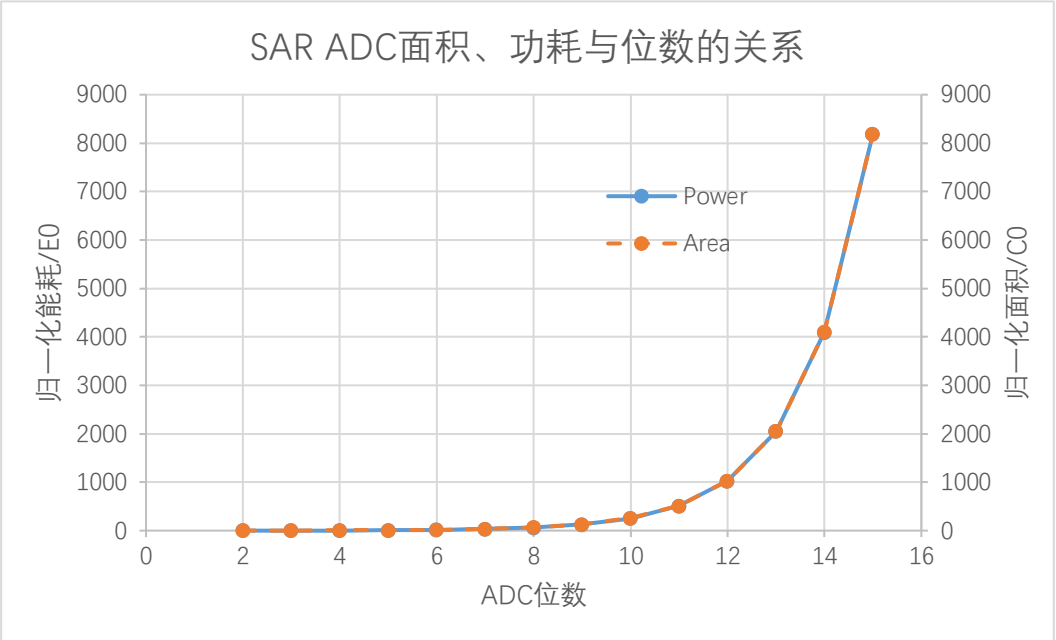


图 3-8 SAR ADC 能耗、面积与位数关系图

由于 SAR ADC 的功耗和面积随着 ADC 位数的增加成指数增加，因此在保

证 ADC 量化精度的前提下，需要尽可能减少 ADC 的位数。本工作中模型量化为 6bit，理论上至少需要采用 6bit ADC，即对 19bit 结果的高 6bit 进行量化，提供的缩放因子$S_1 = 2^{-n_1} = 2^{-13}$。

由 3.2.1 节所述，神经网络各层的缩放因子S_q在一定范围内动态分布，当S_q的值较大时，使用 6bit ADC 进行量化时，ADC 的量程利用率比较低，如图 3-9 所示，即 ADC 对高位进行量化，但结果的有效位数分布在中间部分，因此 ADC 的量程利用率低，量化的准确度低。由于模型量化后各层输出激活的分布已知，因此理论上可以根据分布情况对结果进行动态缩放，从而提高低位宽 ADC 量程的利用率。

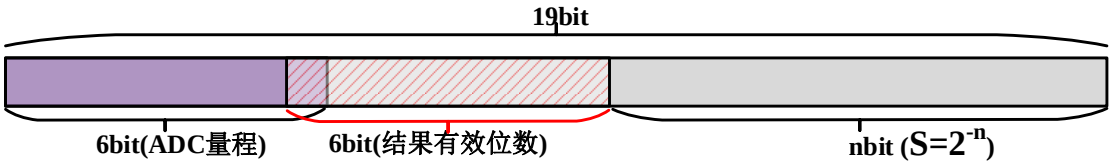


图 3-9 ADC 低量程利用率示意图

3.3.3 动态缩放的实现

针对上述低位宽 ADC 量化准确度低的问题，最直接的方法可以通过增大 ADC 的位宽来提高量化的准确度，该方法使用高位宽 ADC 进行结果的量化，配合使用多路选择器 MUX 从 ADC 的高位宽结果中选出 6bit，并进行四舍五入和截断操作，使用该方法在 6bit ADC 的基础上把 ADC 位宽增加A bit，可以提供一个$S_2 = 2^{n_2} \in \{2^0, \dots, 2^A\}$的缩放因子，但由于 ADC 功耗和面积随位宽呈指数增加，因此直接使用该方法会显著增加整体的功耗和面积。此外文献[42][43]中提出了通过调整 ADC 的参考电压V_{ref}来调整 ADC 量程的方法，即在输出激活较小时（对应缩放因子较大）将 ADC 量程调小，在输出激活较大时（对应缩放因子较小）将量程调大，但V_{ref}减小会使得 ADC 的信噪比提高，从而使得模数转换的性能下降[48]。本文设计了一种动态缩放方法，可保证 ADC 高量程利用率来实现每层激活的量化，提高了低位宽 ADC 量化的准确性，该动态缩放因子调整方法包含 3 个可选步骤，分别为：

（1）读出电流调整。如 3.2.2 节所述，存内计算核中 ADC 对输出电压$\triangle V_{out}$进行模数转换实现原始输出激活的量化，$\triangle V_{out}$与存储逻辑“1”的单元导通后的

读出电流I_{DS}成正比，当$\triangle V_{out}$较小（ADC 量程利用率较低时），可以通过增大读出电流I_{DS}，以实现对$\triangle V_{out}$的放大。根据代工厂提供的资料，擦除/编程时间(Time)、阈值电压(V_t)和读出电流(I_{ds})的关系如图 3-10 所示，图中红色线条表示编程时阈值电压V_t与时间Time关系，蓝色线条表示擦除时阈值电压V_t与时间Time关系，绿色线条表示读出电流I_{ds}与阈值电压V_t的关系，根据该图得可以通过调整擦除时间，来调整逻辑“1”的阈值电压，进而实现读出电流I_{ds}的调整。根据下图，考虑擦除时间在中间位置调整（如图中蓝色箭头所示），使其左右均有裕量，同时由于仅增大逻辑“1”的读出电流，逻辑“0”的读出电流不变（逻辑“0”的编程时间对应图中红色箭头），因此两者读出电流间的差距变大，整体的可靠性、抗干扰性和耐久性均有所提升[49][50]。考虑到外围电路的适配，I_{DS}最小设置为0.125uA，令I_{DS}以$\{2^1, 2^2, 2^3\}$倍数进行放大，可以提供大小为$S_3 = 2^{n_3}$的缩放因子，如下式所示。

$$S_3 = 2^{n_3} \in \{2^0, \dots, 2^3\} \tag{3.11}$$

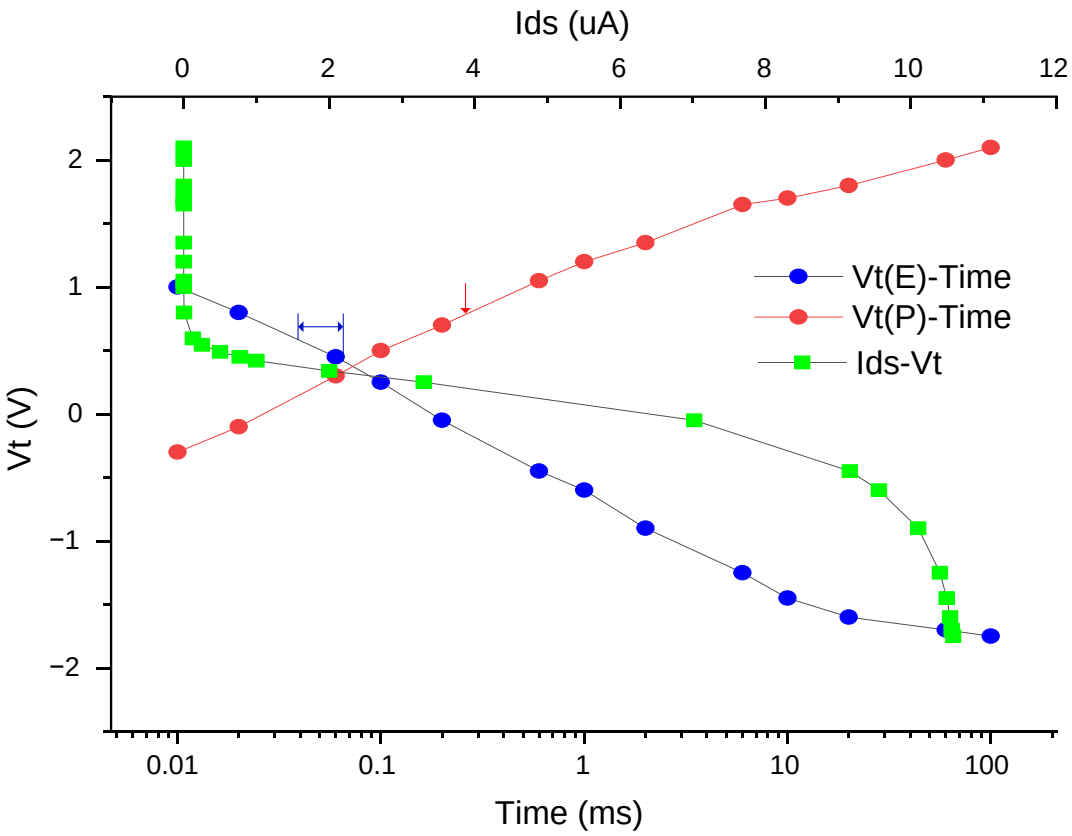


图 3-10 编程/擦除时间(Time)、阈值电压(V_t)和读出电流(I_{ds})的关系图

（2）参数倍增。在存内计算核的参数（即权重）存储设计上，本工作对 6bit

有符号权重使用差分编码的方式存储在 Flash 存储器的多个 cell 中，考虑到 6bit 有符号权重的范围为$[-32,31]$，因此需要占用了 10 个 cell（$r_4^+ - r_0^+$和$r_4^- - r_0^-$），如图 3-11 所示。若使用 12 个 cell 进行存储，则会空余 2 个 cell（r_5^+和r_5^-），提供了权重倍增的空间，如图 3-11 所示，该 12 个 cell 中原来存储的权重值为 32，经过权重倍增后，存储的权重变为 64。在模型量化后已经确定了每层输出激活的缩放因子，当该层缩放因子太大的情况下，可以在对 Flash 存储器进行编程操作的时候写入倍增后的权重，相当于把输出激活的有效位数向高位移动。选择是否倍增存储的权重可以提供的缩放因子S_4如下公式所示。使用更多的 cell 来存储 6bit 有符号权重可以使S_4有更大的调整范围，但也会使得 Flash 利用率下降，导致 Flash 可容纳的参数个数下降，受限于 Flash IP 的大小，本文中选用了 12 个 cell 来存储差分编码的 6bit 有符号权重，即提供了权重值增大 1 倍的空间。

$$S_4 = 2^{n_4} \in \{2^0, 2^1\} \tag{3.12}$$

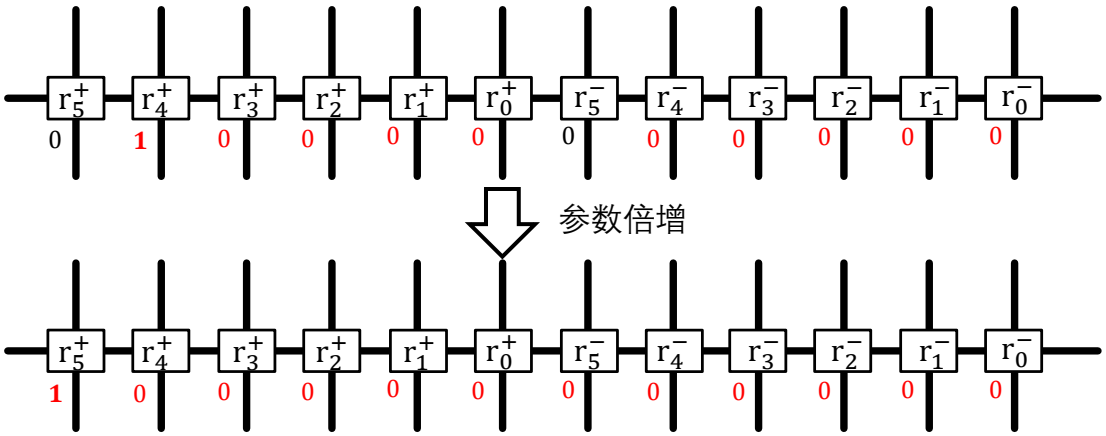


图 3-11 权重倍增图

（3）脉宽调制，动态调整存内计算核的积分时间。如 3.2.2 节所述，ADC 的输入电压值$\triangle V_{out}$和积分时间成正比，积分时间越长，ADC 的输入电压越大。存内计算核控制模块产生存内计算核的控制波形，本文设计了可配置的存内计算核控制模块，如图 3-12 所示，该控制逻辑包括嵌套的模 10 计数器和模 8 计数器、状态机 FSM 和波形生成器。波形生成器接收两个计数器的计数值，并接受 FSM 输入的配置信息，产生存内计算核的控制波形。通过配置输入信号 integral_time 可以对积分时间进行调整，该信号可以在存内计算核运行过程中实时配置，从而实现动态调整存内计算核每次计算过程中的积分时间。

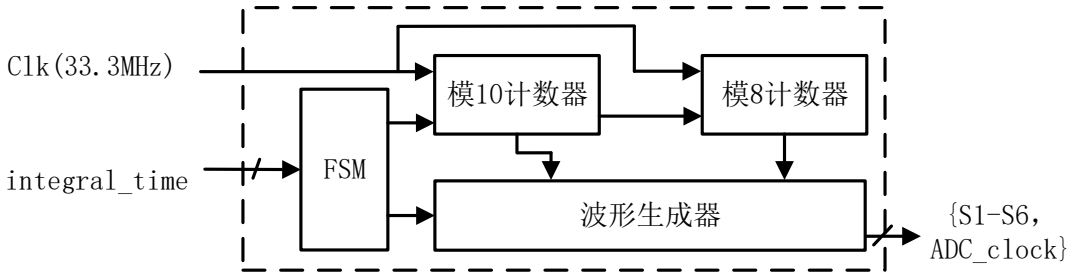


图 3-12 存内计算核控制模块

存内计算核控制模块输出控制波形 S1-S6 和 ADC 时钟信号 ADC_clock，其中 S1-S6 的波形如图 3-13 所示，积分时间即 S2 和 S5 同时为高电平的时间，通过控制 S2 和 S5 下降沿的位置，可以实现积分时间的控制。积分时间过长，会使得存内计算核计算的时间显著增大，因此在本文中积分时间设置了 1 档扩大空间，即可以从 30ns 增大到 60ns，从而实现对$\triangle V_{out}$一倍的放大。脉宽调制可以提供的缩放因子S_5如下公式所示。

$$S_5 = 2^{n_5} \in \{2^0, 2^1\} \tag{3.13}$$

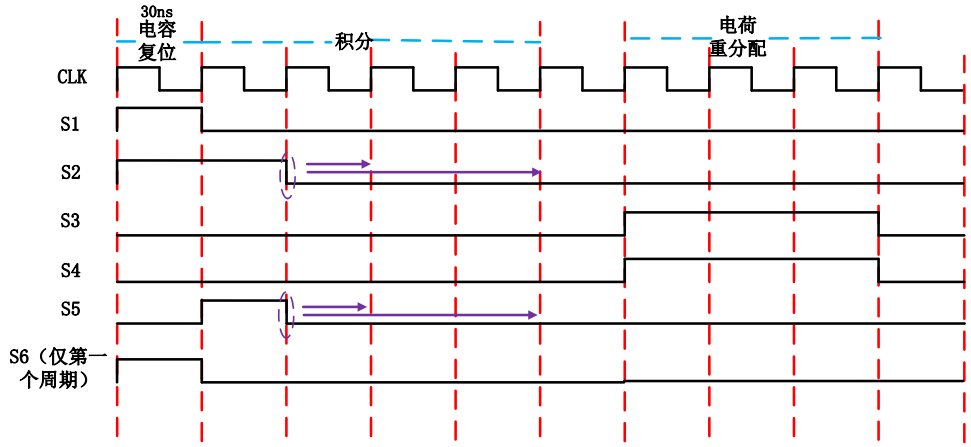


图 3-13 脉宽调制波形图

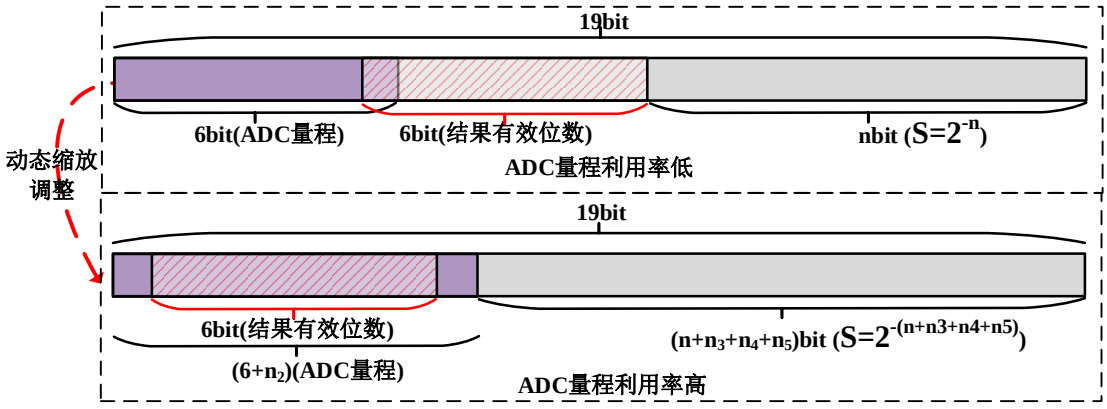


图 3-14 动态缩放前后 ADC 量程利用率对比图

通过上述 3 个步骤，最终可达到的缩放因子调整范围为$S = S_1 \cdot S_2 \cdot S_3 \cdot S_4 \cdot S_5 \in [2^{-13}, \ldots, 2^{-8+A}]$，从而实现了宽范围缩放因子的调整，提高低位宽 ADC 量程的利用率，如图 3-14 所示。

3.4 实验分析

本实验采用的实验环境与 2.5.3 节相同，均基于 Python 脚本语言在语音关键词识别数据集 GSCD[12]上进行实验，首先建立存内计算模型，然后使用 3.3.1 节得到的量化后模型作为测试例测试动态缩放方法对推理准确度的提升效果。

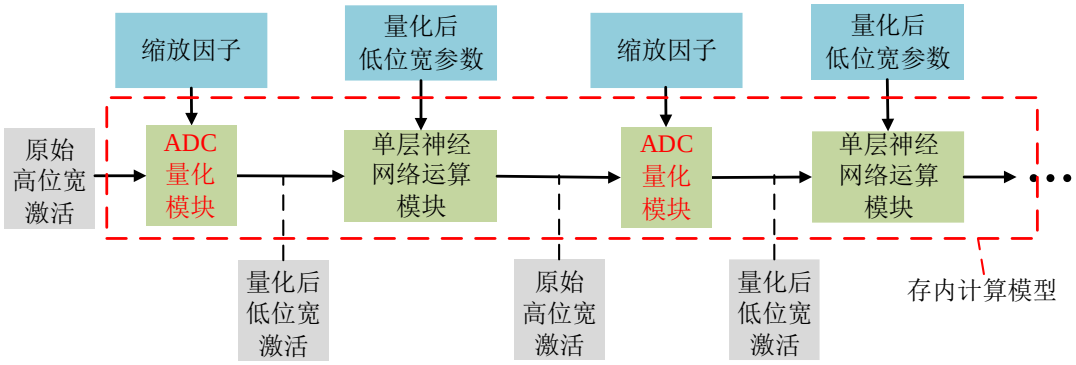


图 3-15 存内计算模型图示

存内计算模型中主要考虑由于 ADC 量程利用率不足带来的量化误差，忽略 ADC 有效位数和存储器单元阻值偏差带来的影响。存内计算模型如图 3-15 所示，该模型由 Python 代码实现，主要包含单层神经网络运算模块和 ADC 量化模块，其中单层神经网络运算模块执行单层神经网络（如卷积层或者全连接层）的计算并输出原始高位宽激活，ADC 量化模块模拟存内计算核中 ADC 的量化过程，对原始高位宽激活执行量化并输出低位宽激活。常规量化方法中直接使用网络模型量化后产生的的缩放因子S_q进行量化，即取中间 6bit，对低位进行四舍五入，对高位进行截断操作。ADC 量化模块方法模拟了使用 ADC 进行量化时，不同量程利用率产生的影响，即在 ADC 量程利用率低时，只对高位进行量化，低位补零，如图 3-16 所示。

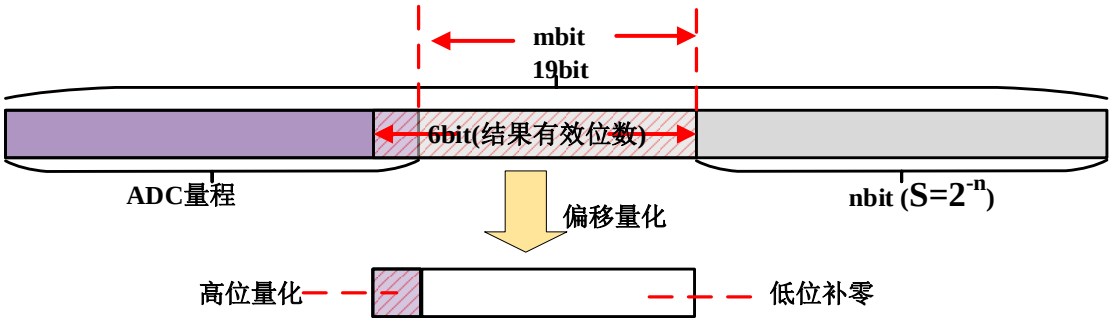


图 3-16 ADC 量化图

ADC 量化模块执行 3 个步骤：（1）根据使用的 ADC 位宽和是否使用缩放因子调整方法，计算offset，如图 3-16 所示，$\text{offset} = 2^{m} = S_1 \cdot S_2 \cdot S_3 \cdot S_4 \cdot S_5 / S_q$，然后根据 offset 产生新的缩放因子$S'$，$S' = S \cdot \text{offset}$，使用$S'$对结果进行常规量化（根据实际使用的 ADC 位数进行截断操作）。（2）对量化后的结果乘 offset，即对低 m 位补零。（3）对乘 offset 之后的结果取低 6 比特，高位进行截断操作。offset 反应了量化过程中 ADC 量程的利用率，当 offset 为 1 时，ADC 量程的利用率为 100%，随着 offset 升高，ADC 量程利用率逐渐下降。

前文介绍了存内计算模型的建立，接下来阐述测试过程和测试方案。测试过程首先对 GSCD 测试集中数据进行预处理得到神经网络模型的输入，然后送入存内计算模型执行神经网络推理准确度测试。在测试方案的设计上，对本文设计的动态缩放方法中的不同步骤进行了对比分析，相应设计了 4 种对比方案：（1）小读出电流缩放方法，即常规 ADC 量化，不采用动态缩放方法；（2）大读出电流缩放方法，即采用动态缩放方法步骤 1;（3）参数倍增缩放方法，即采用动态缩放方法步骤 1、2;（4）本文方法，即采用动态缩放方法步骤 1、2、3。每种测试方案下，分别设置存内计算模型中的 offset 参数以及 ADC 的位数，然后在测试集上进行准确度测试，S-TC-Resnet-1 和 S-TC-Resnet-2 网络的实验结果分别如图 3-17 和图 3-18 所示。

GSCD 数据集为 12 分类，且每一类样本在总样本中均匀分布。当 offset 过大，ADC 量程利用率为 0 进行量化时，ADC 的量化结果基本失效，输出为全 0，此时模型的推理结果固定判别为 12 分类中的某一类，所以在 ADC 位数很低，量化完全失效的情况下，存内计算模型准确度接近$100\%/12 = 8.3\%$。

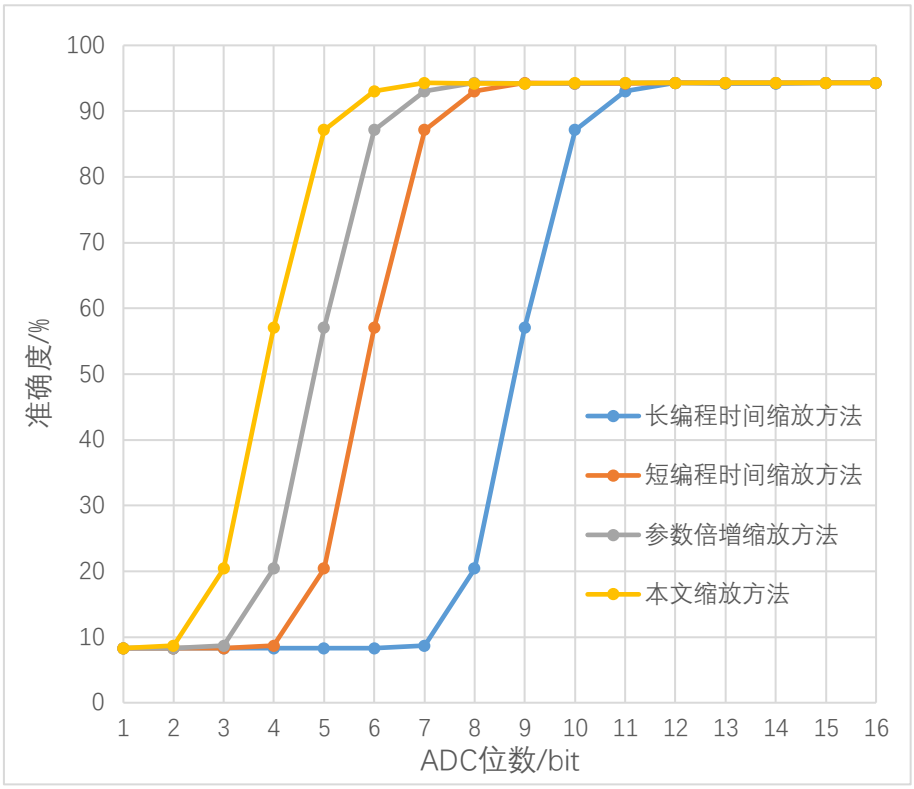


图 3-17 S-TC-Resnet-1 不同缩放方法准确度测试结果

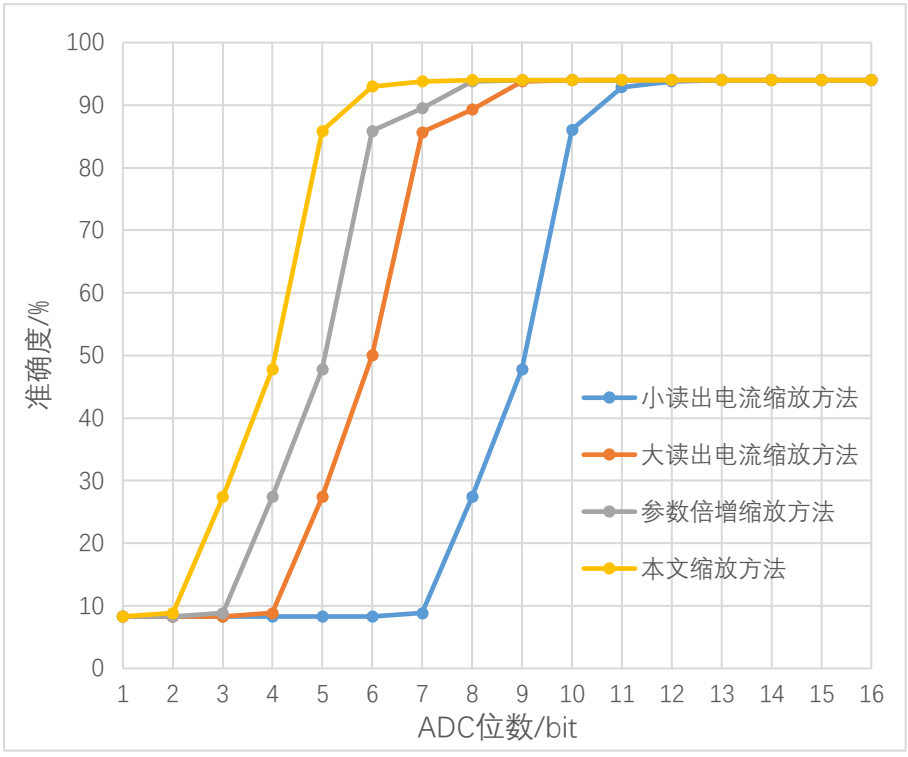


图 3-18 S-TC-Resnet-2 不同缩放方法准确度测试结果

在使用“小读出电流缩放方法”的情况下，当 ADC 位宽小于等于 7 时，ADC 的量化基本完全失效，此时准确度接近 8.3%；在 ADC 位宽在 7 到 11 之间时，

随着 ADC 位宽增大，ADC 的量程利用率逐渐升高，模型准确率显著提升；当 ADC 位数超过 11 后，模型准确度基本稳定。

在使用“本文缩放方法”的情况下，当 ADC 位宽小于等于 2 时，ADC 的量化结果完全失效，因此准确度均为 8.3%，在 ADC 位宽在 3 到 6 之间时，随着 ADC 位宽增大，ADC 的量程利用率逐渐升高，模型准确率显著提升；当 ADC 位数超过 6 后，模型准确度基本稳定。

由于 ADC 的功耗和面积随着位数的增加成指数增加，特别是本文设计的专用处理器中 ADC 的面积占比较大，增加 ADC 位数会带来整体面积显著增大，因此需要在 ADC 位数尽可能低的情况下达到较高的准确度。在使用“本文缩放方法”的情况下，ADC 位数为 6 时准确度为 93%，已匹配同类工作的准确度表现[7][8]，因此考虑将 ADC 位数设置为 6，相比于“小读出电流缩放方法”下 ADC 位数为 11 时才可以实现较高准确度，大大降低了 ADC 的硬件开销。

3.5 本章小结

本章首先介绍了神经网络模型量化的基础知识，对量化后模型执行推理过程中的激活量化进行了说明，然后分析了由于神经网络模型量化分布不均衡带来的低位宽 ADC 量程利用率低的问题，并提出了动态缩放方法，最后进行了软件仿真，实验结果表明，通过使用动态缩放方法，可以大幅提高低位宽 ADC 量程的利用率，在 ADC 开销尽可能低的情况下实现了较高的准确度。

第4章 低功耗神经网络处理器设计

4.1 引言

本章首先从指令集架构和微架构两个方面展开，指令集架构中首先介绍了处理器框架及不同模式，然后介绍了专用指令集设计，微架构中对流水线设计、低功耗存内计算核和处理器流式推理的实现依次进行了说明，最后进行了整体电路实验，并同相关工作进行了对比分析。

4.2 专用处理器指令集架构

4.2.1 专用处理器框架及工作模式

神经网络处理器架构框图如图 4-1 所示，包括（1）控制器；（2）张量运算器，包括向量-矩阵乘法器（即存内计算核）、向量常数除法器和向量加法器；（3）片上寄存器组，包括用来存储数据的通用寄存器组（GRF）、用来存储存内计算核相关配置信息的配置寄存器组 CRF，以及用来存储通用寄存器组 GRF 访问地址的地址寄存器组 ARF。

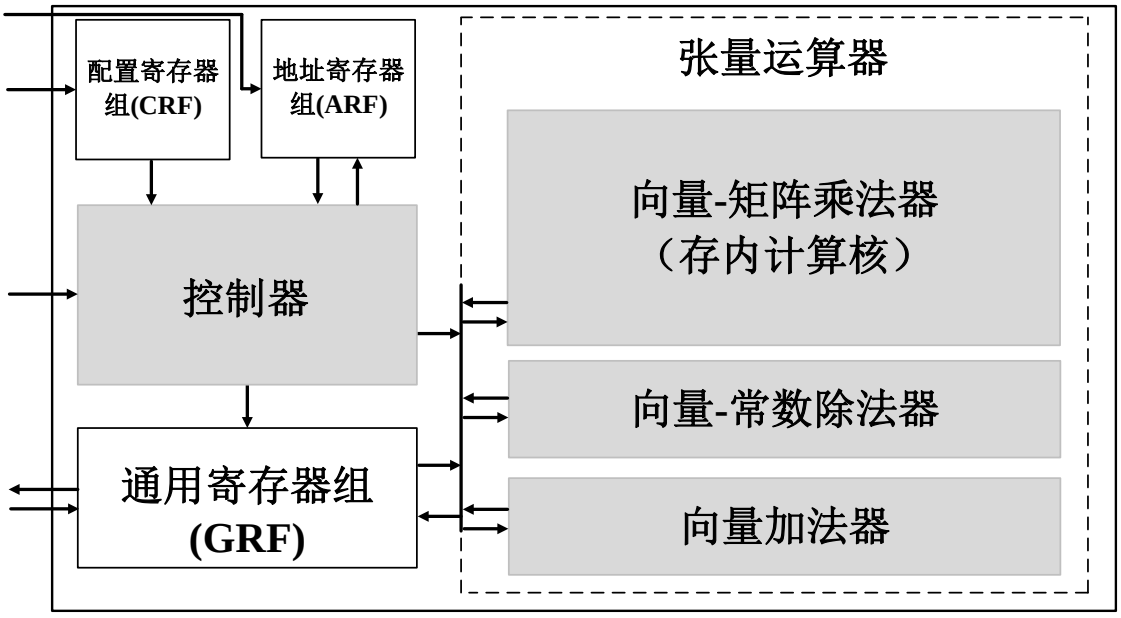


图 4-1 处理器整体架构图

各存储器类型和大小如下表所示(表中包含了用于存放指令的指令存储器 IRF，以及存内计算核内部的存储器)：

表格 4-1 处理器存储器类型、容量表

名称	存储器类型	存储数据类型	存储器容量
通用寄存器组 GRF	Regfile	神经网络激活	208 × 96b
指令存储器 IRF	Regfile	指令	256 × 16b
配置寄存器组 CRF	Flip-flop	存内计算核相关的配置信息	16 × 48b
地址寄存器组 ARF	Flip-flop	指令读取通用寄存器组 DRF 的地址	16 × 24b
存内计算核内的 Flash 阵列	Flash	神经网络参数	256 × 2048b
存内计算核内的输入数据寄存器 DRF	Flip-flop	向量-矩阵乘法中的输入向量，对应神经网络输入激活	1536b

针对语音关键词识别任务中分帧处理的特点，设计了处理器的 3 种模式，分别为配置模式、正常模式和休眠模式，各模式转换关系如图 4-2 所示。

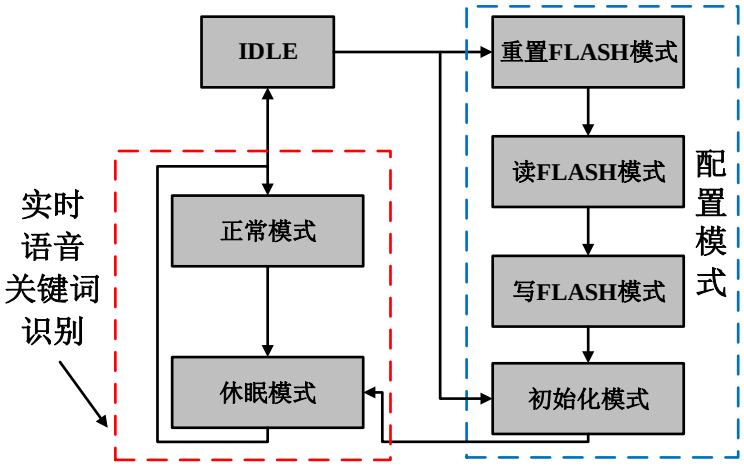


图 4-2 处理器模式转换图

如上图 4-2 所示，配置模式包括了 4 个子模式，分别为重置 Flash 模式、读 Flash 模式、写 Flash 模式和初始化模式，其中重置 Flash 模式实现对 Flash 存储器的全局擦除操作，读 Flash 模式读取 Flash 存储器中每个单元的阻值，写 Flash 模式完成对 Flash 存储器所有单元的编程操作（对应把神经网络参数写入 Flash 存储器），初始化模式完成片上寄存器组 IRF、CRF、ARF 和 GRF 中数据的初始化。需要指出的是代工厂提供的 IP 本身支持重置 FLASH 模式和写 FLASH 模式，因此这 2 个模式可以依据代工厂提供的资料从外围对 IP 进行控制实现，而读 FLASH 模式则基于 3.2.2 节中 FLASH 存内计算的原理实现，实现过程为通过控制图 3-4 中的 S3 开关来选通某一列位线上的单元，并通过仅在特定字线上输

入高电平其余字线上输入低电平来选通某一行的单元，从而实现单个单元的选通，然后按照图 3-4 中的波形进行控制，进行电容复位、电流积分（对单个单元导通电流I_{ds}的积分）和电荷重分配，得到$\triangle V_{out}$，对其进行模数转换得到$\triangle V_{out}$的数字值，该数字值即反映了阻值大小。正常模式顺序执行 3 个步骤：（1）语音特征传入；（2）神经网络推理计算；（3）推理结果传出。3 个步骤中仅神经网络推理计算通过控制器中的流水线实现，剩余 2 个步骤由控制器的其它通路实现。休眠模式中处理器流水线停止运行，控制器对存内计算核进行 power gating 操作，并将 IRF 和 GRF 寄存器组置入保持（Retention）模式，从而尽可能减少处理器的漏电功耗。

在完成配置模式后，处理器可以开始执行实时语音关键词识别，在实时语音关键词识别过程中，处理器不断在正常模式和休眠模式之间切换，如图 4-2 所示。由于处理器的吞吐量很大，因此每帧神经网络的推理计算可以很快完成，即每帧正常模式持续的时间很短，剩下的绝大部分时间处理器从正常模式切换到休眠模式，直到下一帧特征向量的输入后再切回正常模式，如图 4-3 所示。

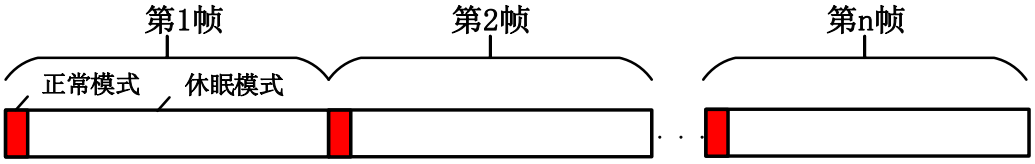


图 4-3 处理器运行实时语音信号处理下的模式示意图

4.2.2 专用处理器指令集设计

本节结合处理器框架设计了专用指令集，通过单指令多数据（Single Instruction Multiple Data，SIMD）的设计，实现对神经网络模型中相邻通道的数据并行计算/存储，考虑到神经网络模型中通道数一般为 2 的整数次幂，如本文采用的 S-TC-Resnet 网络的通道数有 16 和 32 两种情况，为了在增加并行度的同时提高硬件资源的利用率，处理器单条指令对 16 个通道的数据进行并行操作，且本文对 S-TC-Resnet 进行了 6bit 低位宽量化，因此处理器最终采用了 96 比特的数据位宽，表示一个1×16维度的向量，向量中每个元素的位宽都是 6 比特。

处理器每帧正常模式执行的操作相同，理论上每帧可以执行相同的指令流，从而减小需要存储的指令流长度，但由于相邻帧之间存在数据重用，因此相邻帧

的指令访问通用寄存器组 GRF 的地址有一定的偏移，使得处理器每帧不能直接执行完全相同的指令流。为了解决上述问题，本文在指令集设计中增加了对于寄存器迭代计算的支持，具体实现方式为把地址的偏移信息保存在地址寄存器组 ARF 中，并在指令执行过程中通过计数器等逻辑对地址寄存器 ARF 中的偏移信息进行更新，相应在指令中设置了地址寄存器组 ARF 地址字段，指令通过读取地址寄存器组 ARF 获取当前帧访问通用寄存器组 GRF 的地址，使得处理器每帧可以迭代执行相同的指令，减小了需要存储的指令流长度。

处理器的指令集如下图所示，指令宽度为 14bit，可分为 4 个字段，其中 13-10 比特表示指令的操作码；9-6 比特表示指令访问地址寄存器组 ARF 的地址；5-2 比特表示指令访问 CRF/DRF 寄存器组的地址，或者除法指令的除数 divisor（对原始的除数取 2 的对数，如原始除数为 64，则指令字段中存储 6）；1-0 比特表示指令的类型，指令共包含 3 种类型，分别为头指令、体指令和尾指令，其中 2′b00表示头指令，2′b01表示体指令，2′b10 表示尾指令。

<table>
<tr><th>Bits
inst name</th><th>13-10</th><th>9-6</th><th>5-2</th><th>1-0</th></tr>
<tr><td>vector-matrix mul</td><td>opcode=0</td><td rowspan="6">Addr of ARF</td><td>Addr of CRF</td><td rowspan="11">Inst type</td></tr>
<tr><td>vector add</td><td>opcode=1</td><td rowspan="2">0</td></tr>
<tr><td>vector sub</td><td>opcode=2</td></tr>
<tr><td>vector div</td><td>opcode=3</td><td>divisor</td></tr>
<tr><td>input to GRF</td><td>opcode=4</td><td>0</td></tr>
<tr><td>from GRF to DRF</td><td>opcode=5</td><td rowspan="2">Addr of DRF</td></tr>
<tr><td>write 0 to DRF</td><td>opcode=6</td><td rowspan="4">0</td></tr>
<tr><td>reset DRF</td><td>opcode=7</td><td rowspan="3">0</td></tr>
<tr><td>do-nothing</td><td>opcode=8</td></tr>
<tr><td>stand-by</td><td>opcode=9</td></tr>
</table>

图 4-4 指令集

根据指令功能可以分为 3 类：（1）数据运算类指令，负责执行向量的运算操作，包括 vector-matrix mul 指令、Vector add 指令、Vector sub 指令和 Vector div 指令；（2）数据传输类指令，负责数据在通用寄存器组、特殊寄存器之间的转移，包括 Input to GRF 指令、From GRF to DRF 指令、Write 0 to

DRF 指令和 Reset DRF 指令；（3）控制类指令，包括空指令和休眠指令。各指令执行的具体操作如下所述：

（1） vector-matrix mul 指令，向量-矩阵乘法指令，使用存内计算核完成一次向量-矩阵乘法操作，结果存入通用寄存器组 GRF 指定地址。vector-matrix mul 指令的执行需要配合大量的控制信息，包括：动态缩放方法中积分时间的控制信息、Flash 存储器块的选择信息和偏置选择信息，14bit 无法容纳 vector-matrix mul 指令需要的所有信息，因此增加了配置寄存器组 CRF 来配合存储这些信息。vector-matrix mul 指令中包含配置寄存器组 CRF 的地址字段，在 vector-matrix mul 指令运行过程中会读取配置寄存器组 CRF 对应地址来得到所有的控制信息。

（2） Vector add 指令，向量加法指令，调用向量加/减法器完成 2 个向量的加法操作，结果暂存在 sum 寄存器中。

（3） Vector sub 指令，向量减法指令，调用向量加/减法器完成 2 个向量的减法操作，结果暂存在 sum 寄存器中。

（4） Vector div 指令，向量-常数除法指令，调用向量除法器完成向量-常数除法操作，结果写入通用寄存器组 GRF 指定地址。

（5） Input to GRF 指令，将输入语音特征写入通用寄存器组 GRF 指定地址。

（6） From GRF to DRF 指令，从通用寄存器组 GRF 指定地址读出，写入 DRF 指定地址中。

（7） Write 0 to DRF，向 DRF 指定地址写入 0 向量。

（8） Reset DRF，把 DRF 全部重置为 0。

（9） do-nothing，空指令。

（10）stand-by，休眠指令，控制处理器进入休眠状态。

4.3 专用处理器微架构

前文介绍了处理器指令集架构，本节对处理器微架构进行阐述，首先介绍了处理器流水线设计，对流水线不同阶段的执行过程进行了阐述，然后介绍了低功耗存内计算核设计与流式推理，处理器中把神经网络中绝大多数计算映射到存内计算核上执行，大大减少了存储器访问的次数，此外通过流式推理方法大幅减少

了每帧的计算量，相应降低了整体的功耗和延时。

4.3.1 流水线设计

本节在 4.2.2 节 SIMD 指令集的基础上设计了流水线，来进一步提高处理器的吞吐量，提升各部件的利用率。流水线仅在处理器正常模式执行推理计算时启动，由于推理计算中只需要对寄存器组进行读写，不需要对其它存储器如 Flash 进行读写（Flash 的编程和擦除操作在配置模式已经完成），因此本文在经典五级流水线的基础上，省去了“存储器访问（MEM）”阶段，并对其它阶段进行了一些调整，最终设计了 4 级流水线，分别为“取指(IF)、译码(ID)、执行(EX)、写回(WB)”，如下图所示。下文对流水线不同级执行的功能进行阐述。

指令数	流水线阶段						
1	IF	ID	EX	WB			
2		IF	ID	EX	WB		
3			IF	ID	EX	WB	
4				IF	ID	EX	WB
时钟周期数	1	2	3	4	5	6	7

2400ns

图 4-5 处理器 4 级流水线图示

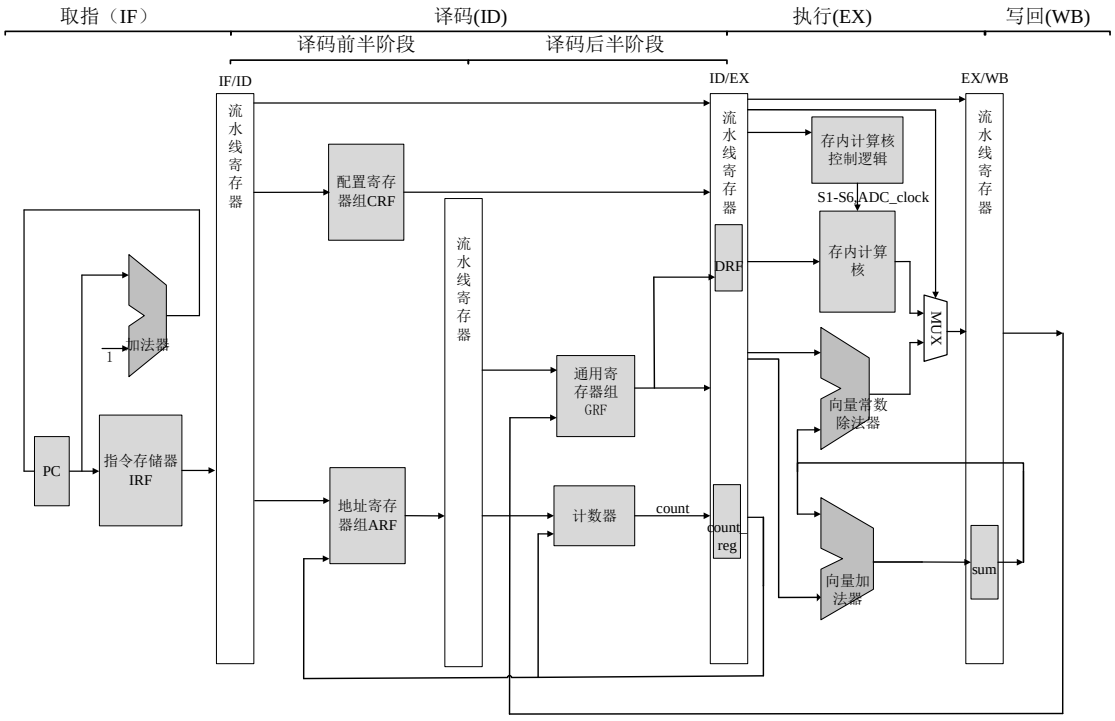


图 4-6 处理器流水线数据通路和控制通路

取指阶段实现了指令的读取，即使用程序计数器 PC 产生的地址对指令存储

器 IRF 进行读取，输出指令，同时程序计数器 PC 计数值递增，为下个周期取指做准备。

译码阶段控制器对指令进行译码操作，产生各模块的控制信号，并实现寄存器组的读取。如前文 4.2.1 节所述，本文设计的专用处理器中包含配置寄存器组 CRF、地址寄存器组 ARF 和通用寄存器组 GRF，并且通过访问地址寄存器组 ARF 实现对通用寄存器组 GRF 的间接读取，因此译码阶段共包含对 3 个寄存器组的访问。译码阶段又可分为前后两个子阶段（由频率为流水线频率 2 倍的高频时钟控制），前半阶段对配置寄存器组 CRF 和地址寄存器组 ARF 进行读取操作，从配置寄存器组 CRF 中获取存内计算核相关的配置信息，为执行阶段可能执行的向量-矩阵乘法做准备，从地址寄存器组 ARF 获取激活在通用寄存器组 GRF 中的地址，为后半阶段读取通用寄存器组 GRF 中的激活做准备。后半阶段使用从地址寄存器组 ARF 中读到的 GRF 地址，对通用寄存器组 GRF 进行读取，得到参与后续执行阶段计算的激活，同时计数器进行计数得到后续指令读取通用寄存器组 GRF 的地址信息，并将结果写入寄存器 count_reg 中，以待后续指令使用。

执行阶段中，数据运算类指令启动张量运算器，完成向量矩阵乘法、向量加/减法和向量-常数除法三种操作中的一种；其中向量加/减法器和向量-常数除法器的控制逻辑较简单，存内计算核的控制逻辑相对复杂，因此具有独立的存内计算核控制模块，该控制模块在执行阶段负责产生存内计算核的控制波形，即 3.2.2 节所述的信号 S1-S6。数据传输类指令完成寄存器组 GRF/DRF 的写入操作，把寄存器的写入操作安排到执行阶段，一方面是由于数据传输指令要写入的数据不依赖执行阶段的结果，另一方面将写入操作从写回阶段提前到执行阶段，可以使数据尽快完成写入，从而减少由于后续指令可能存在的数据依赖而产生的阻塞。

写回阶段将数据运算类指令执行阶段的结果写回通用寄存器组 GRF 或 sum 寄存器中，由于译码阶段后半阶段进行了 GRF 读取操作，为了避免写回阶段和译码阶段同时访问 GRF 产生冲突，写回阶段实际上在高频时钟的控制下在前半阶段完成对 GRF 的写入操作。

4.3.2 存内计算核设计

本节在 3.2.2 节单通道存内计算核的基础上扩展到了 16 通道，增加了外围的

寄存器 DRF 和 MUX2，执行的计算形式从向量乘法扩展到了向量-矩阵乘法，并设计了存内计算核的不同模式和分块策略。

存内计算核的结构如图 4-7 所示，由 5 个部分组成，包括：（1）1536 比特的触发器 DRF；（2）256 个 6 选 1 MUX2；（3）192 个 10 选 1 MUX3；（4）64KB Flash 存储器阵列；(5)16 个差分辅助计算模块(每个辅助计算模块的细节如 3.2.2 节所述)。

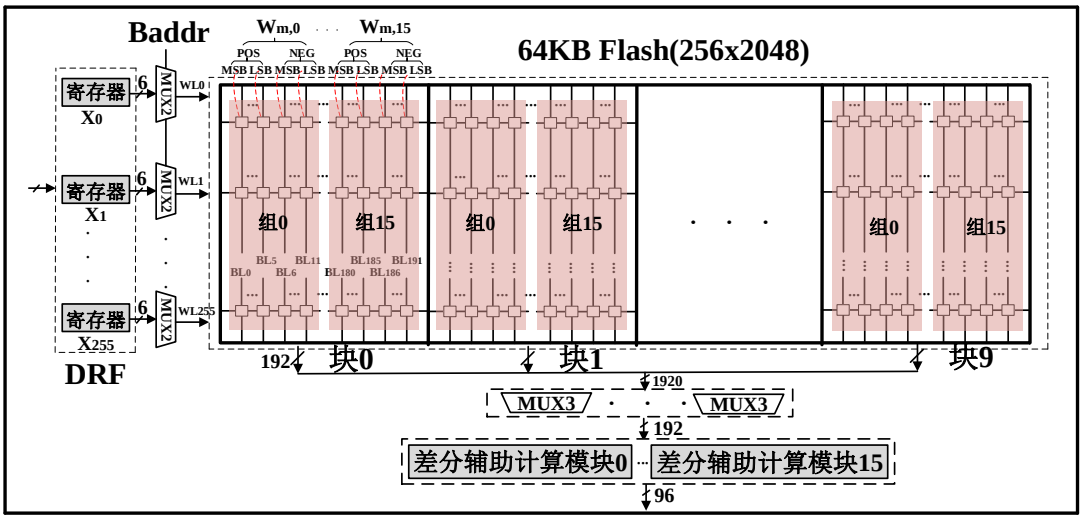


图 4-7 支持向量-矩阵乘的存内计算核结构图

DRF 中的 1536 个单比特触发器分为 256 组，每组包含 6 个触发器，每组触发器的输出端口和 1 个 6 选 1 MUX2 连接。256 个 6 选 1 MUX2 的输出分别连接到 Flash 存储器阵列的 256 根字线上。64KB Flash 存储器阵列的尺寸为 256×2048，包含 256 根字线和 2048 根位线。Flash 存储器阵列在位线维度上分为了 10 块，每块的大小为 256×192，所有的块共享辅助计算模块。每块 Flash 存储器阵列在位线维度上又分为了 16 组，每组包含 12 列。同一根位线上的一组字线对应了 12 个 Flash 存储单元，一个 6 比特有符号权重使用差分编码的方式存储在 12 个 Flash 存储单元中。192 个 10 选 1 MUX3 的 1920 个输入端口分别连接 10 块存储阵列包含的 1920 根字线，用来从 10 块 Flash 存储阵列中选出其中的一块，即从 1920 根位线中选出 192 根位线，然后接入辅助计算模块。

存内计算核共包含 4 种模式，分别为擦除模式，读模式，写模式和计算模式，各模式说明如下表所示。为了减小处理器片上资源，存内计算核部分控制电路在片外 FPGA 上实现，片外 FPGA 和处理器通过 SPI 协议进行通信。

表格 4-2 基于 Flash 的存内计算核各模式说明

模式名称	功能
计算模式	执行向量-矩阵乘法
擦除模式	Flash 整块擦除为低阻态
读模式	读出 Flash 指定地址单元的阻值
写模式	对 Flash 指定地址进行编程操作

在计算模式中，存内计算核支持的计算形式为向量-矩阵乘法，如下公式所示。

$$Y = Q(Y') = Q\big(Relu(X \times W)\big) \tag{4.1}$$

其中向量 X 为存内计算核单次计算的输入向量，$X \subseteq R^{1\times256}, X_i \in [0,2^6-1], i \in [1,256]$，向量 X 对应神经网络算法每层的输入激活；矩阵 W 对应神经网络算法中的权重，$W \subseteq R^{256\times16}, W_{m,n} \in [-2^5, 2^5-1], m \in [1,256], n \in [1,16]$，矩阵 W 存储在 10 块 Flash 存储器阵列的其中一块中；向量Y'表示向量 X 和矩阵 W 相乘并经过 Relu 操作后的原始输出向量，$Y' \subseteq R^{1\times16}, Y'_j \in [0,2^{19}-1], j \in [1,16]$。向量 Y 表示对向量$Y'$中每个元素进行量化后的最终输出结果，$Y \subseteq R^{1\times16}, Y_j \in [0,2^6-1], j \in [1,16]$，对应神经网络算法中的输出激活；Relu(·)为线性整流激活函数；Q(·)为量化函数。

Flash 存储器阵列共分为 10 块，通过 MUX3 的选通作用，每次只选择其中一块来完成向量-矩阵乘。存内计算核每完成一次向量-矩阵乘法共包含 7 个阶段，如图 4-8 所示，每个阶段时长为 300ns。DRF 中的 256 个 6 比特触发器组依次存储向量 X 中每个元素X_i。经过 256 个 MUX2 的选通作用，X_i的最低有效位（Least Significant Bit，LSB）到最高有效位（Most Significant Bit，MSB）分 6 个阶段依次输入到 Flash 存储器阵列的字线上，对应下图中的 BIT0-BIT5 阶段，在每个阶段都会进行电容重置、电流积分和电荷重分配。第 7 个阶段执行模数转换，对应下图 ADC 阶段，16 个 ADC 并行完成模数转换，每个 ADC 输出 6bit 的转换结果，对应存内计算核单次向量-矩阵乘的最终输出向量 Y。

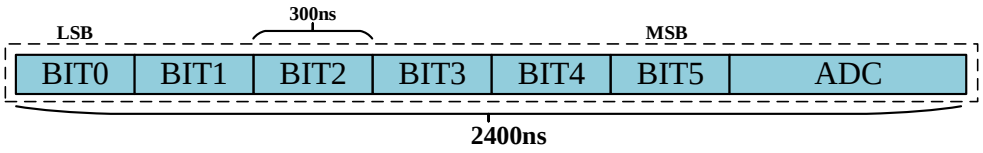


图 4-8 存内计算核单次计算各阶段示意图

在调用存内计算核完成一次向量-矩阵乘法操作前需要完成数据准备，即把输入向量 X 先写入 DRF 中，把权重 W 先写入 Flash 存储器阵列某一块中，其中输入向量 X 在处理器正常模式中实时更新，权重 W 在神经网络模型推理过程中一般不会发生变化，因此可以在处理器的写 Flash 模式中预先写入 Flash 存储器中。完成数据准备后，存内计算核控制模块产生存内计算核需要的控制信号，执行该次向量-矩阵乘操作。

每次对 Flash 存储器存储的权重进行更新，都需要依次经过擦除模式，读模式和写模式。其中读模式可以把 Flash 存储器每个单元的阻值读出来，根据读模式读出的每个单元的阻值，可以对权重的差分编码进行优化，即根据读模式读出的阻值来优化写模式要写入的值，减少由于 Flash 阻值分布带来的误差，提高存内计算核计算的准确度[28]。

4.3.3 流式推理的处理器实现概述

本节介绍处理器以流式推理的方式运行 S-TC-Resnet 算法的细节，如前文 4.2.1 节所述，处理器进入正常模式执行推理计算前，会先在配置模式下把 S-TC-Resnet 所有的参数值通过擦除和编程预先写入 Flash 存储器内。以目标算法 S-TC-Resnet-1 网络为例，其参数写入 Flash 存储器后资源使用情况如图 4-9 所示，其中 Flash 存储器共分为了 10 块，阴影部分表示写入后用到的部分，白色部分表示空闲部分，阴影部分的字母标号表示该部分权重所属的层，各层由于权重数量不同，因此对存储器的占用情况不同，对于权重数量较少的层可以容纳在一块内，如 conv0 层等，对于权重数量较多的层需要使用多块，如 conv2_1 层等。

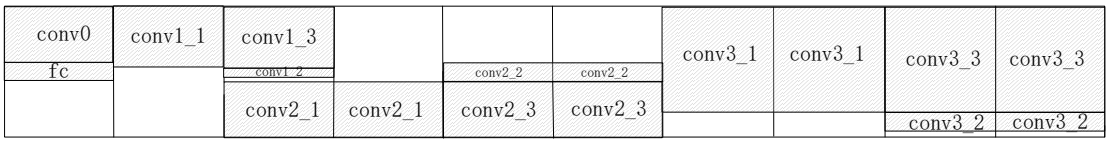


图 4-9 S-TC-Resnet-1 权重映射图

处理器执行流式推理时，每帧依次从第一层开始执行，到最后一层结束，执

行到某层时，若该层为卷积层或全连接层，则将该层的输入激活从通用寄存器组GRF搬移到DRF中以便从Flash字线输入，然后选中存储该层权重的Flash块，调用存内计算核执行计算，若该层为平均池化层，则调用向量加法器和向量-常数除法器执行计算，下面分别从不同层在处理器上的流式推理执行过程展开（本节中各符号意义见表格 2-1）。

（1）卷积层的流式推理。如图 4-10 所示，卷积层中每个卷积窗口的计算都可以转换为一次向量矩阵乘法。每个卷积窗口的输入激活转换为一个维度为$1 \times (IC \times KW)$的向量X，OC个卷积核转换为一个$(IC \times KW) \times OC$维度的矩阵W。此外每个卷积窗口的计算需要在输入激活和卷积核卷积结果（$X \times W$）上加入偏置b，如下公式(4.3)所示。由于偏置b的位宽一般大于激活和权重的位宽，为了方便进行统一化的存储和计算，在模型量化中把每个偏置b分解为$b_x \times b_w$，其中b_x和b_w的位宽和量化后的激活和权重一致。进一步把激活和权重的向量矩阵乘法与偏置b_x与b_w的乘法进行整合，则每个卷积窗口的计算最终转换为一次向量-矩阵乘法。流式推理中，在步长为1的情况下，卷积层缓存之前帧输入的（KW-1）列输入激活进行复用，同时卷积层每帧会得到一列新的输入激活，如图 4-10 中输入特征图中红色填充的第（IW+1）列所示，该列和缓存的（KW-1）列组成一个新的卷积窗口，如图 4-10 中输入特征图上的红色虚线框所示。卷积层每帧对该新的卷积窗口进行计算，通过转换为向量-矩阵乘法映射到存内计算核上执行，其中向量映射到DRF中，矩阵映射到Flash阵列的某个块中，然后存内计算核启动计算得到每帧的输出，如图 4-10 中输出特征图中红色填充的第（OW+1）列所示。不考虑偏置的情况下，分批推理下卷积层每帧的计算量为$OC \times IC \times KW \times OW$，流式推理下卷积层每帧的计算量为$OC \times IC \times KW$，相比分批推理，流式推理下计算量减少了OW倍。

$$Y = X \times W + b = X \times W + b_x \times b_w = \begin{bmatrix} X & b_x \end{bmatrix} \begin{bmatrix} W \\ b_w \end{bmatrix} \tag{4.3}$$

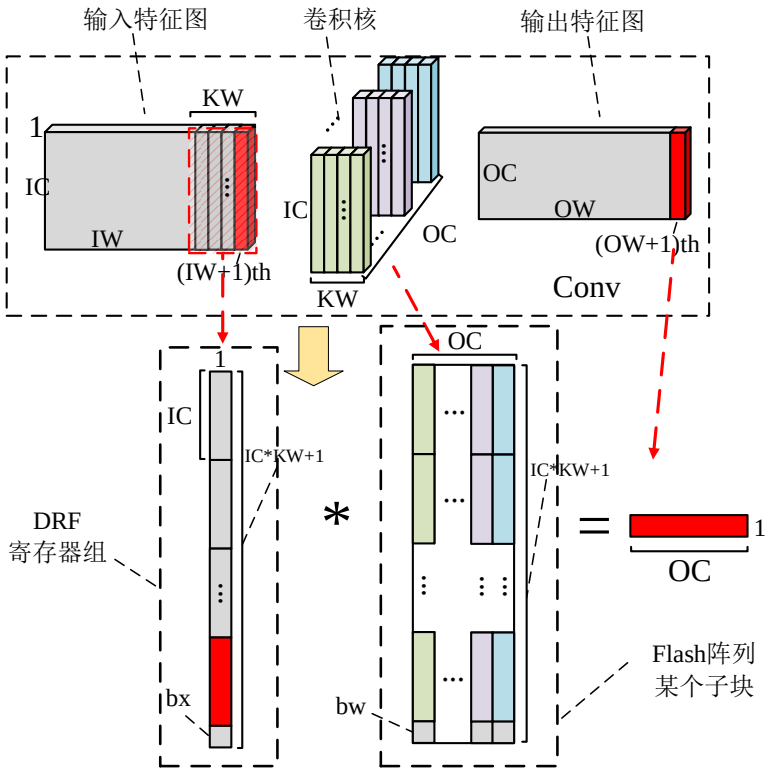


图 4-10 一维卷积层流式推理执行图

残差层中包含了 3 个一维卷积层，即主路径上的 conv_1 和 conv_2，以及分支路径上的 conv_3，每个一维卷积层均单独执行流式推理，每帧主路径和分支路径分别会产生一列输出向量，对两个输出向量进行元素级相加，从而产生残差层每帧流式推理的结果。本文设计的专用处理器把每帧 conv_2 和 conv_3 各自的向量-矩阵乘计算合并为单次向量-矩阵乘，进而通过调用一次存内计算核完成 conv_2 和 conv_3 的计算。在流式推理下，残差层中每个一维卷积层均减少了 OW 倍的计算量。

（2）全连接层的流式推理。全连接层本身激活和权重的计算形式为向量-矩阵乘，如图 4-11 所示，在此基础上对偏置的计算也进行整合，如上公式所示，从而把全连接层一次完整的计算也转换为一次向量矩阵乘。全连接层每帧会输入新的输入特征图，每帧均转换为向量-矩阵乘映射到存内计算核上执行，映射方式和卷积层相同，从而每帧都产生新的输出特征图，即每帧生成推理结果。不考虑偏置的情况下，分批推理和流式推理下全连接层每帧的计算量均为$OC \times IC$。

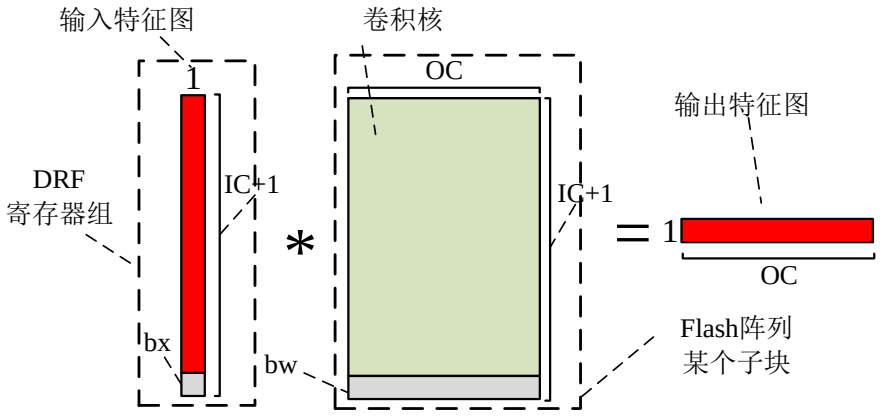


图 4-11 全连接层流式推理执行图

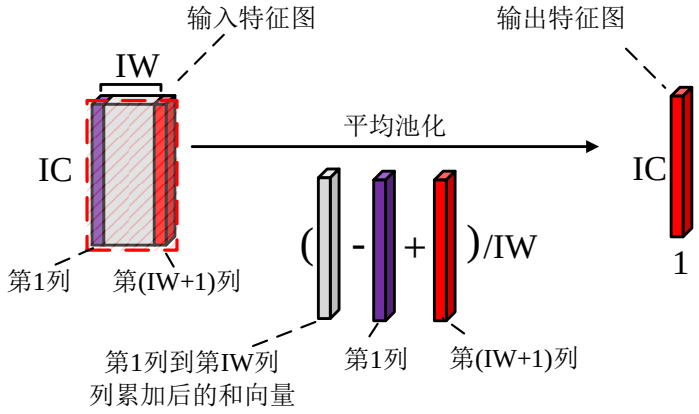


图 4-12 平均池化层流式推理执行图

（3）平均池化层的流式推理。常规的分批推理中，平均池化层对整个输入特征图在 IW 维度上进行累加得到一个$1 \times IC$维度的和向量，然后该向量和常数 IW 进行向量-常数除法操作得到池化结果。在流式推理中，对和向量以及之前 IW 帧的输入激活进行缓存和复用，每帧会得到一列新的输入激活，如图 4-12 输入特征图中的红色填充的第（IW+1）列所示，每帧在和向量的基础上减去保存的第 1 列的输入激活，然后加上第（IW+1）列的输入激活，再进行向量-常数除法操作，得到当前帧的池化结果。池化层的计算由向量加法器和向量-常数除法器完成。分批推理下平均池化层每帧的加/减法数为$IW \times IC$，流式推理下池化层每帧的加/减法数为$2 \times IC$，加法/减法数量减少了 IW/2 倍，除法数保持不变。

（4）S-TC-Resnet 的流式推理。下图 4-13 以 S-TC-Resnet-1 为例，展示了完整的网络流式推理的方法，下图中每层分别按前文所述的流式推理方法执行，即：卷积层每帧执行红色虚线框中的单个卷积窗口，通过把该卷积窗口转换为向量-矩阵乘然后调用存内计算核实现；平均池化层每帧执行和向量的更新然后执行除

法，通过调用向量加/减法器和向量-常数除法器实现；全连接层每帧的计算转换为向量-矩阵乘然后调用存内计算核实现。

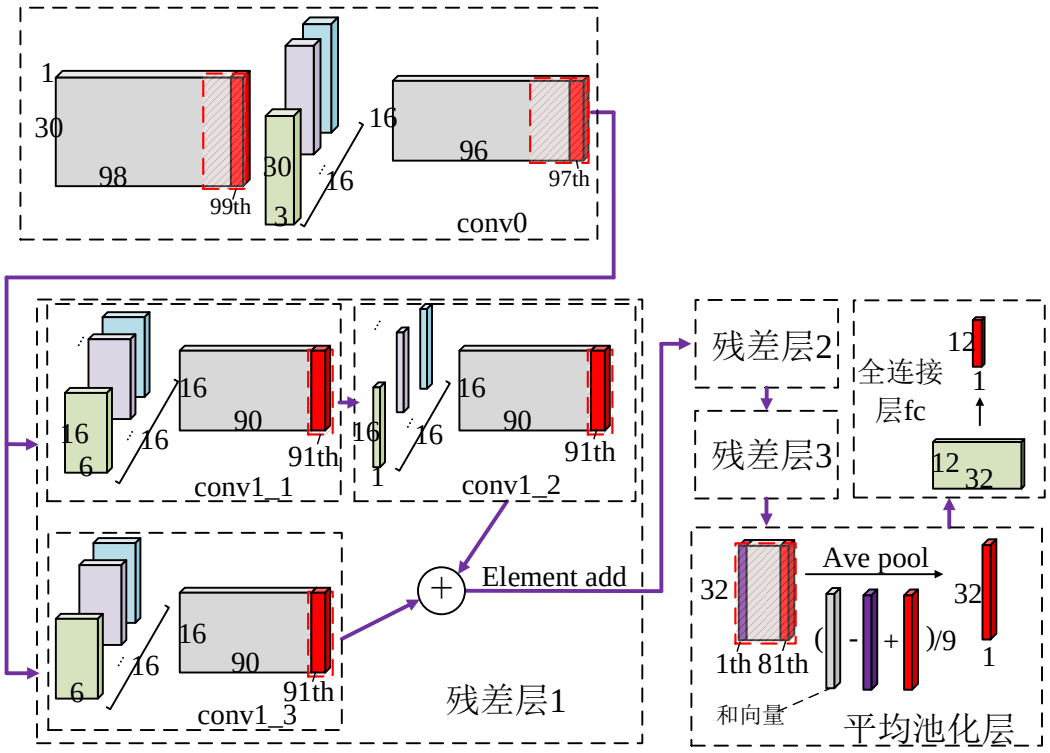


图 4-13 S-TC-Resnet 流式推理执行图

根据上文对计算量的分析，相比分批推理，通过使用流式推理的执行方式处理器运行 S-TC-Resnet 的计算量大幅减少，如下表所示。

表格 4-3 分批推理与流式推理对比

	分批推理 MAC 数	流式推理 MAC 数	减少比例
S-TC-Resnet-1	2.16M	26.2K	98.8%
S-TC-Resnet-2	0.54M	6.36K	98.8%

4.3.4 流式推理的指令流实现

上文介绍了流式推理的处理器实现概述，本节对更具体的指令实现进行阐述。为了实现流式推理中的数据重用，处理器相邻帧的指令访问通用寄存器组 GRF 的地址需要产生一定的偏移，因此不同帧不能直接执行相同的指令，如 4.2.2 节所述，针对该问题本文在指令集设计中增加了对寄存器迭代计算的支持，使得处理器每帧可以迭代执行相同的指令流来实现流式推理。寄存器迭代计算具体通过指令组配合计数器和地址寄存器组 ARF 实现，在本文设计的专用处理器中把神经网络每一层相关的多条指令合并为一个指令组，即把前一层对 GRF 进行写操

作的指令和下一层对 GRF 进行读操作的指令划分为一个指令组，每个指令组中包含 3 种指令：头指令、体指令和尾指令。因为流式推理的过程从第一层开始依次计算到最后一层，因此同一指令组中的指令是连续的，并且不同指令组之间不存在交叠。

如图 4-6 所示，执行头指令时，在流水线译码阶段的前半段对 ARF 进行读取，在后半阶段使用从地址寄存器组 ARF 中读出的地址对 GRF 进行读取，并使用计数器对该地址加 1，写入寄存器 count_reg 中；在执行体指令时，在译码阶段的后半段使用寄存器 count_reg 中记录的地址去访问 GRF，并使用计数器对该地址加 1，然后写入寄存器 count_reg 中；在执行尾指令时，在译码阶段的后半段使用寄存器 count_reg 中记录的地址读 GRF，并使用计数器对该地址加 1，然后在流水线执行阶段写回 ARF 的指定地址中（写 ARF 的地址从尾指令的 ARF 地址字段中获得）。通过上述执行过程，相邻帧间的地址偏移信息可以在指令执行过程中自动记录下来，并保存在地址寄存器组 ARF 中，从而保证处理器每帧可以对相同的指令流进行迭代执行，减小了指令流的存储空间。

下面以 S-TC-Resnet-1 模型中的 CONV0 层为例，结合指令，阐述指令的执行过程。Conv0 层的映射如图 4-14 所示，其中 conv0 层在通用寄存器组 GRF 中分配的地址个数为 6，对应地址 0 到地址 5。

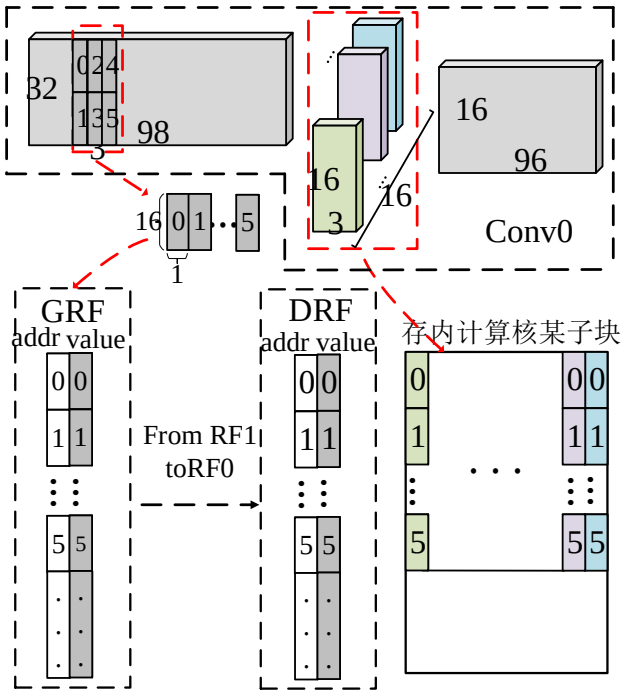


图 4-14 TC-Resnet8 CONV0 层执行示意图

如下表所示，CONV0 层经过编译后，生成 7 条指令，即下表中的指令 3-9。

表格 4-4 S-TC-Resnet-1 CONV0 层指令流

layer	Instruction	Num	Type	Nth	(N+1)th
Init	Input to GRF	1	头指令	0	2
	Input to GRF	2	体指令	1	3
Conv0	From GRF to DRF	3	体指令	2	4
	From GRF to DRF	4	体指令	3	5
	From GRF to DRF	5	体指令	4	0
	From GRF to DRF	6	体指令	5	1
	From GRF to DRF	7	体指令	0	2
	From GRF to DRF	8	尾指令	1	3
	vector-matrix mul	9	头指令	6	7
…	…	…	…	…	…

表中 init 层表示每帧把 conv0 层的输入特征向量写入通用寄存器组 GRF 的过程，init 层的前 2 条指令与 conv0 层的前 6 条指令形成一个指令组，即表中前 8 条指令。以第 n 帧和第（n+1）帧为例，该指令组数据流动如图 4-15 所示，其中 GRF 中的蓝色填充部分表示相邻帧间重用的数据。

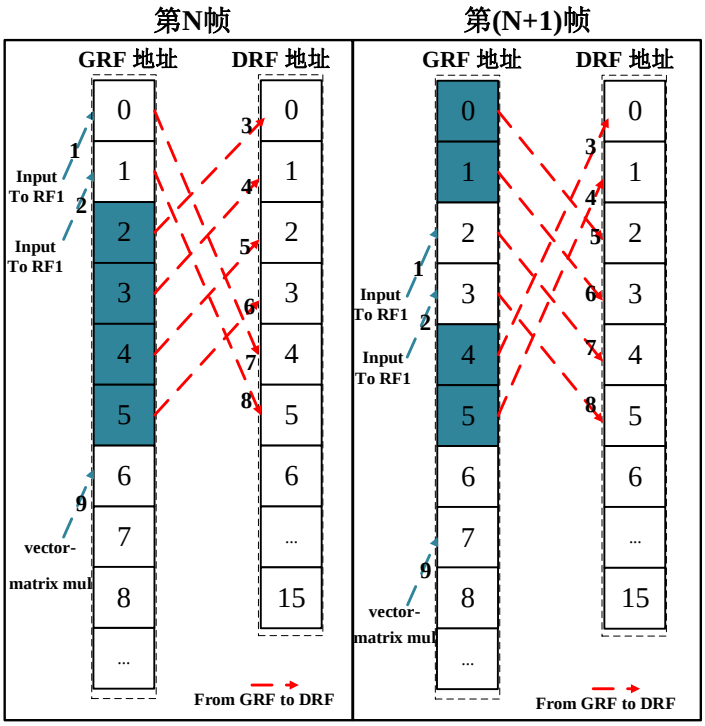


图 4-15 S-TC-Resnet-1 CONV0 相邻帧间指令执行情况及数据重用示意图

第 n 帧该指令组的执行过程中，头指令“Input to GRF”首先从 ARF 中读出

当前输入向量要写入 GRF 的地址，即地址 0，然后通过计数器对该地址加 1 变为 2 写入寄存器 count_reg 中；init 阶段的第二条“Input to GRF”为体指令，该指令把寄存器 count_reg 中存储的地址 1 作为输入向量要写入 GRF 的地址，然后通过计数器对该地址加 1 变为 2 写入寄存器 count_reg 中；conv0 层的第 1 条“from GRF to DRF”指令把寄存器 count_reg 中的地址 2 作为读取 GRF 的地址，然后通过计数器对该地址加 1 变为 3 写入寄存器 count_reg 中；conv0 层的第 2-5 条“from GRF to DRF”指令均为体指令，其操作过程与第 1 条“from GRF to DRF”指令类似；conv0 层的第 6 条“from GRF to DRF”指令为尾指令，该指令把寄存器 count_reg 中的地址 1 作为读取 GRF 的地址，然后通过计数器对该地址加 1 变为 2 写回 ARF 中。第(n+1)帧该指令组的执行过程中，头指令“Input to GRF”首先从 ARF 中读出当前输入向量要写入 GRF 的地址，即地址 2，后续过程与第 n 帧的执行过程类似，由此可见处理器每帧通过迭代执行相同的指令最终实现了对流式推理的控制。

4.4 实验分析

4.4.1 实验环境及方法

本文设计的低功耗神经网络处理器采用 40nm 工艺实现。本文设计的专用处理器为数模混合电路，本文主要集中在对数字电路部分的设计和实现，并对处理器整体性能进行测试，测试的输入为 GSCD 数据集中 10 个长度为 1s 的语音样本拼接后的连续语音，对其使用 Python 进行预处理后送入神经网络处理器执行推理计算。实验对功耗和延时等指标进行了测量，其中功耗为神经网络处理器执行语音关键词识别的平均功耗，延时为处理器从接收到当前帧的语音特征到完成当前帧语音关键词识别之间的时长。

对于数字电路部分的性能测试，首先进行数字电路部分的 RTL 设计，并建立模拟电路部分的 RTL 模型，然后使用功能仿真工具（VCS）进行整体 RTL 功能仿真，仿真结果与 python 参考模型结果进行对比，验证 RTL 功能正确。接下来使用综合工具（Design Compiler，DC）进行门级网表的综合（综合过程中不包含模拟电路部分的 RTL 模型），并使用形式验证工具（Formality）对综合的正确性进行验证；接下来使用 VCS 对门级网表进行仿真（模拟电路部分仍以 RTL 模

型参与仿真），得到了描述门级网表翻转率的波形文件，并从波形文件中测得延时数据。最后使用功耗分析工具（Prime Time PX，PTPX）结合波形文件对门级网表进行功耗分析（TT、25℃、1.1V 工艺角），从而得到数字电路部分的功耗。

模拟电路部分首先使用 virtuoso spectre 工具单独仿真验证了其功能的正确性，然后使用 virtuoso AMS 工具进行数模混合仿真验证了数模接口的正确性（其中 Flash 存储器为华力提供的 IP，包含了晶体管级电路模型和版图，Flash 存储器的擦除模式和编程模式需要高电压 4.44V，在读模式和计算模式下电压为 1V）。在进行功耗分析时（TT、25℃、1V 工艺角），由于存内计算核包含了 64KB 的 Flash 阵列和规模较大的外围模拟电路，并且服务器平台算力有限，因此难以对存内计算核进行神经网络模型系统级仿真。本文中对存内计算核进行了单独仿真，存内计算核的动态功耗$\mathrm{p_{cim}}$由阵列功耗P_{array}、ADC 功耗P_{ADC}和接口电路功耗P_{regu}三部分构成，其中阵列功耗P_{array}和每次计算中的数据分布相关，在 Flash 某个单元为低阻，且对应字线输入为高电平的情况下，该单元导通且产生较大电流，其它情况下该单元未导通，电流极低可以忽略，因此阵列功耗P_{array}和 Flash 存储器中导通单元的个数成正比。本工作首先测试了存内计算核所有单元均导通的峰值功耗，然后使用该峰值功耗折算得到实际功耗，具体步骤为：（1）使用 virtuoso spectre 测试了存内计算核在所有单元均导通下的峰值动态功耗，即把 Flash 中所有单元置为低阻，且 256 根字线全为高电平的情况下测试单次运算的峰值动态能耗$\mathrm{E_{peak}}$。（2）统计神经网络模型每层计算过程中 Flash 存储单元的导通比例P，首先统计字线输入为高电平的比例$\mathrm{P_a}$，即神经网络每层输入激活中“1”的平均比例，然后统计模型每层参数中“1”的平均比例$\mathrm{P_w}$，由于采用了差分编码的方式存储（12 个单元存储一个 6bit 有符号参数），因此 Flash 存储器中低阻单元的比例需要在$\mathrm{P_w}$的基础上除 2，最后统计每层网络在存内计算核中的占用率$\mathrm{P_{used}}$（即把该层网络参数映射到存内计算核某块上后，实际占用该块的比例，如图 4-9 所示），从而计算得到存内计算核运行某层神经网络时单元的导通比例$\mathrm{P} = \mathrm{P_a} \cdot P_w/2 \cdot P_{used}$。最终阵列功耗$P_{array}$计算如下公式，$N^i$表示神经网络第 i 层在 1s 内调用存内计算核的次数，P^i表示该层数据的稀疏度，I 表示网络的层数。进一步在该动态功耗的基础上加入 ADC 功耗P_{ADC}、接口电路功耗P_{regu}和静态功耗即

可得到模拟电路总功耗。

$$P_{array} = E_{peak} \cdot \sum_{i=1}^{I} N^i \cdot P^i = E_{peak} \cdot \sum_{i}^{N} N^i \cdot \mathrm{P}_{\mathrm{a}}^i \cdot \frac{P_w^i}{2} \cdot P_{used}^i \tag{4.4}$$

最终对数字电路功耗和模拟电路功耗进行叠加，从而得到处理器整体功耗。

4.4.2 专用处理器特性评估

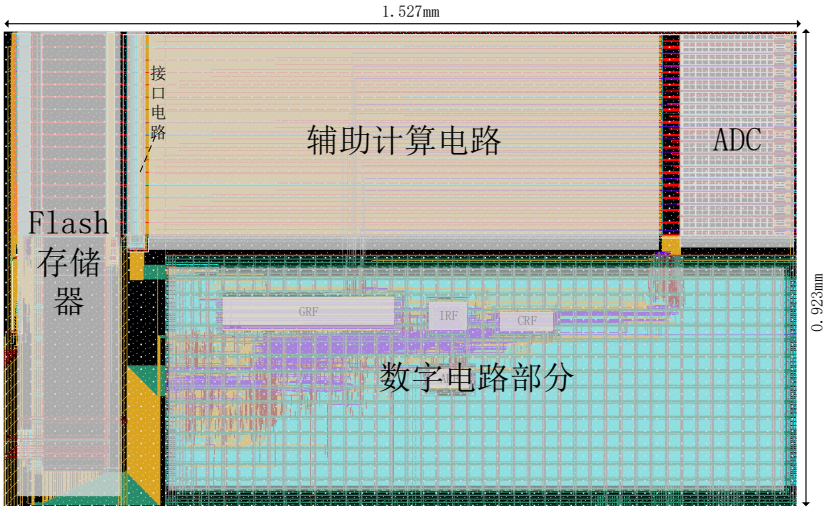


图 4-16 芯片版图

芯片初步的版图如图 4-16 所示，面积为 1.41mm^2，该芯片为处理器的测试芯片，在处理器的基础上增加了指令存储器 IRF，以及用于片内外通信的串行外围设备接口模块。

当运行语音关键词识别任务时，本文测试了 2 种模式：每帧推理模式和每 8 帧推理模式。每帧推理模式下运行的网络模型为 S-TC-Resnet-1（卷积步长全为 1），运行该模型处理器每帧（10ms）均产生一次推理结果；每 8 帧推理模式下运行的网络模型为 S-TC-Resnet-2（包含 3 个卷积步长为 2 的卷积层），由于步长的累积，运行该模型处理器每 8 帧（$2 \times 2 \times 2 = 80$ms）产生一次推理结果。每帧推理模式和每 8 帧推理模式可分别用于对推理频率需求不同的应用场合。

在每帧推理模式下，每帧 S-TC-Resnet-1 模型中每一层均运行一次，S-TC-Resnet-1 模型每层激活和权重的稀疏情况，以及阵列的占用情况如图 4-17 所示（图中 conv_23 表示 conv_2 和 conv_3 合并计算，fc 表示全连接层），整体的功耗分布如表格 4-5 和图 4-18 所示，总功耗为 4.01uw。

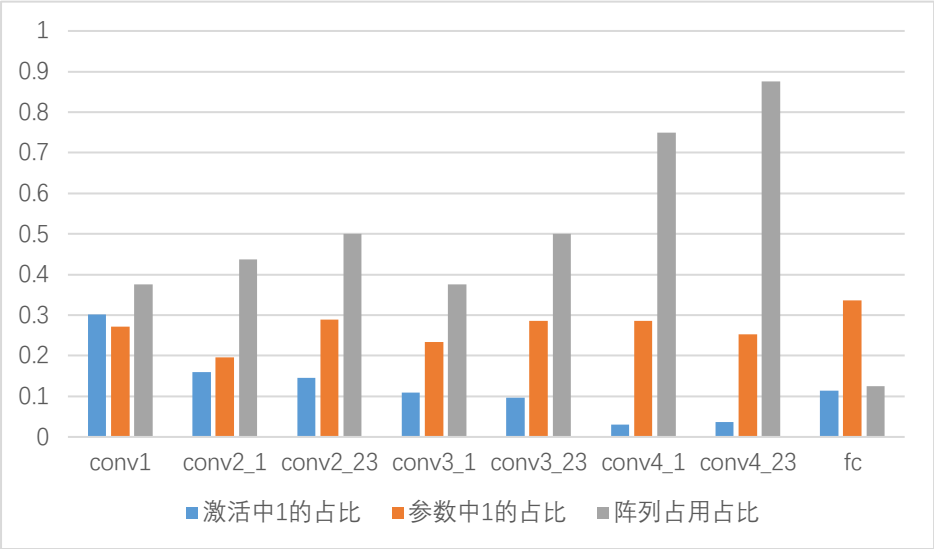
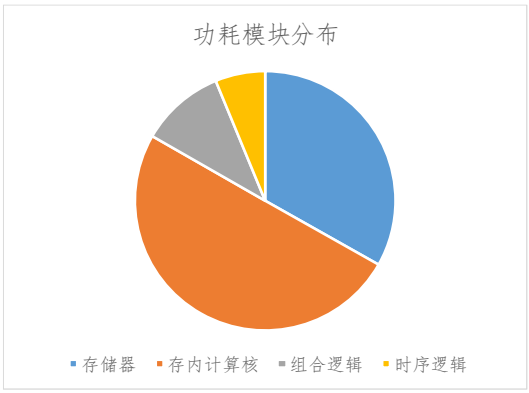


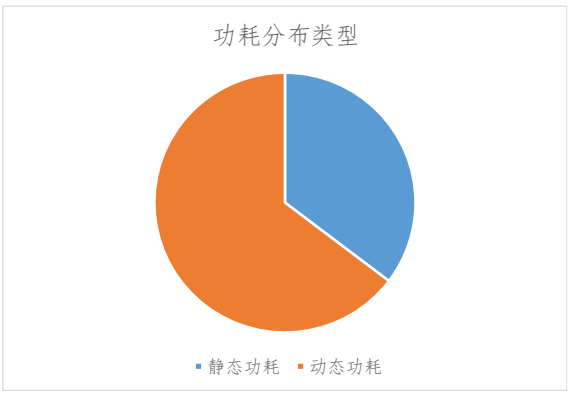
图 4-17 S-TC-Resnet-1 模型数据稀疏情况

表格 4-5 每帧推理模式各模块功耗组成

电路类型	功耗/uw			占比
	动态功耗	静态功耗	总计	
组合逻辑	0.327	0.093	0.420	18.2%
时序逻辑	0.030	0.218	0.248	10.7%
存内计算核	1.881	0.130	2.011	13.5%
存储器	0.357	0.974	1.331	57.6%
整体	2.595	1.415	4.010	100.0%



(a)



(b)

图 4-18 每帧推理模式下功耗分布图

每 8 帧推理模式运行的网络模型为包含卷积步长为 2 的 S-TC-Resnet-2。每 8 帧推理模式下，S-TC-Resnet-2 中由于存在步长为 2 的卷积层，由于步长的累加，各层之间运行的频率会出现差异，层数越靠后的层运行的频率越低。CONV0 层每帧运行 1 次，残差层 1 每 2 帧运行一次，残差层 2 每 4 帧运行一次，残差层 3、平均池化层和全连接层每 8 帧运行一次。S-TC-Resnet-2 中每层激活和权重的

稀疏情况（图中 conv2_23 表示 conv_2 和 conv_3 合并计算，fc 表示全连接层），以及阵列的占用情况如图 4-19 所示，整体功耗分布如表格 4-6 和图 4-20 所示，总功耗为 2.127uw。

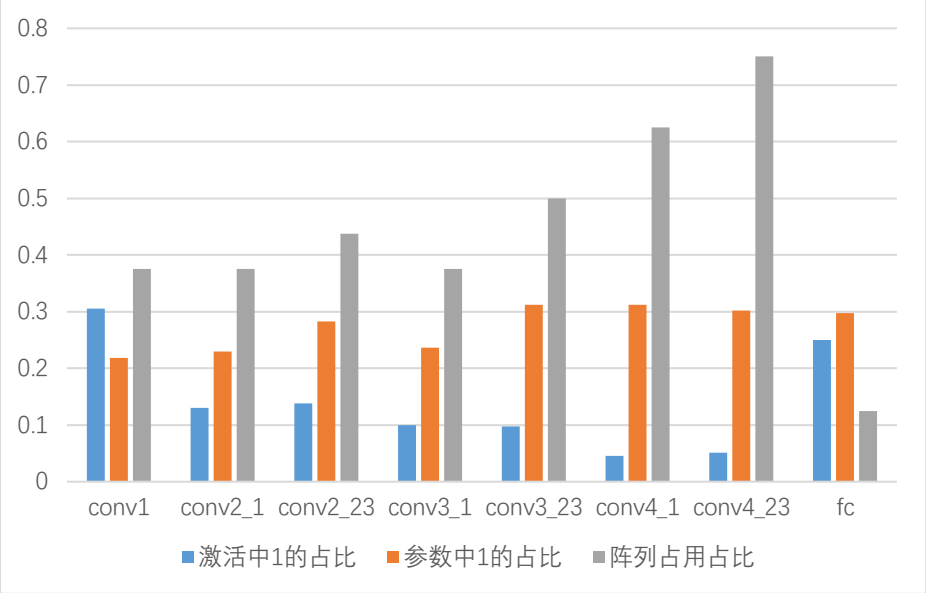
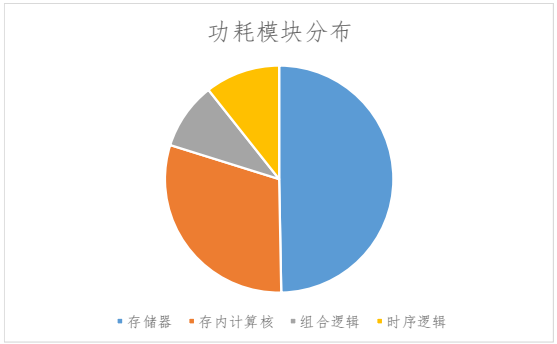


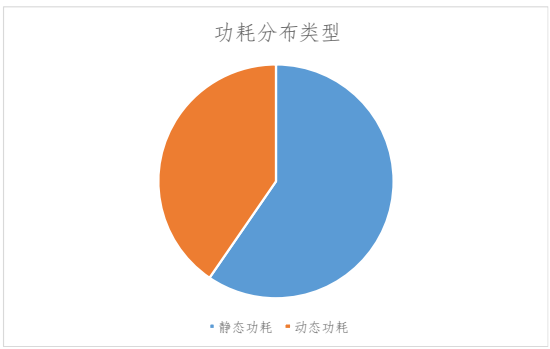
图 4-19 S-TC-Resnet-2 模型数据稀疏情况

表格 4-6 每 8 帧推理模式各模块功耗组成

电路类型	功耗/uw			占比
	动态功耗	静态功耗	总计	
组合逻辑	0.111	0.092	0.203	12.7%
时序逻辑	0.010	0.216	0.226	14.1%
存内计算核	0.597	0.044	0.641	7.2%
存储器	0.142	0.915	1.057	66.0%
整体	0.860	1.267	2.127	100.0%



(a)



(b)

图 4-20 每 8 帧推理模式下功耗分布图

与每帧推理模式相比，每 8 帧推理模式中每层运行的频率更低，因此动态功耗大幅降低，同时正常模式持续的时间更短，休眠模式持续的时间更长，因此静

态功耗也有下降。但是由于每 8 帧推理模式中推理的频率更低，因此在实时关键词检测过程中，会出现识别效果下降的问题，需要综合应用场景的需求选择不同的模式。出于增加实时语音关键词识别效果的考虑，相关工作中大多采用了每帧推理模式[12]，下文的对比分析中也使用每帧推理模式下的性能参与对比。

4.4.3 对比分析

神经网络处理器的性能与可识别的关键词个数有关，通常可识别的关键词个数越多，处理器需要执行的计算量越大，功耗越高[47]，目前相关工作中较好的水平实现了 10 个关键词的识别[27]，下表选择满足该条件并且功耗较低的最新工作进行对比。

表格 4-7 性能对比表

	TCAD 2020 [7]	TCAII 2021 [4]	TVLSI 2021 [8]	本文设计的专用处理器
工艺(nm)	22	40	65	40
算法	TC-Resnet	LSTM	TCN	S-TC-Resnet
数据集	GSCD			
识别关键词个数	10	10	10	10
准确度	93.09%	90.60%	93.30%	93.00%
位宽	8/4bit 混合	8bit	8bit	6bit
片上存储器容量	74.125KB SRAM	9.28KB SRAM	128KB SRAM	64KB Flash 2.9KB regfile
频率(KHz)	250	400	27-31	420
电压(V)	0.8	0.6	1.2	1.1
面积(mm^2)	0.2	0.16	-	1.41
归一化面积(mm^2)	0.59	0.16	-	
延时(ms)	100	-	3	0.18
归一化延时(ms)	127.49	-	6.36	
功耗(uw)	8.2	1.84	20.9	4.01
归一化功耗(uw)	37.8	6.10	9.5	
可配置性	TC-Resnet	No	TCN	S-TC-Resnet

上表中采用文献[46]中不同工艺和电压的归一化方法，将其它工作的性能归一化到了与本文设计的专用处理器相同的 40nm 和 1.1V（本文采用了 LP 低功耗工艺，文献[46]中的归一化方法对 65nm 以下工艺区分了 LP 低功耗工艺和 HP 高性能工艺，上表对比文献[7]和文献[4]均采用了 65nm 以下工艺，其中[7]中指明

了采用 LP 工艺，因此文献[7]以 LP 工艺参与归一化，文献[4]中未指明工艺类型，通过比较文献[4]在 LP 和 HP 工艺下的归一化功耗发现以 LP 工艺参与归一化时功耗更低，因此文献[4]以 LP 工艺参与归一化）。本文使用每帧推理模式下的性能和同类工作进行对比（文献[4]实现了神经网络处理器和预处理模块，上表中功耗和延时指标中减去了预处理部分的贡献）。

在归一化功耗的对比上，与文献[7][4][8]相比，本文设计的专用处理器功耗分别减少了 89.4%、34.3%和 57.8%，实现了较低的功耗。在归一化延时的对比上，本文设计的专用处理器的延时低于上表其它工作，满足实时的要求。在归一化面积对比上，虽然同表中其它工作相比本文设计的专用处理器面积较大，但参考文献[27]中对相关领域的综述，本文设计的专用处理器的面积在领域中处于中等水平，同时随着工艺的发展，面积的影响将越来越小。综上所述，本文设计的专用处理器实现了低功耗的设计目标。

4.5 本章小节

本章首先从专用处理器指令集架构展开，然后介绍了处理器框架及工作模式，并对指令集进行了阐述，专用指令集的设计使得处理器可以灵活运行 S-TC-Resnet 网络，实现语音关键词识别。其次介绍了处理器流水线设计，对流水线各个阶段的执行情况进行了说明。然后阐述了低功耗存内计算核设计与处理器流式推理执行细节，其中流式推理显著减少了推理过程中的计算量，相应地带来处理器功耗和延时上的降低。最后对处理器进行了整体的电路仿真，分析了各个模块的功耗分布，并与其它支持语音关键词识别的神经网络处理器进行了对比。实验结果显示，在每帧推理模式下，本芯片可识别 10 个关键词，识别准确度为 93.0%，功耗为 4.01uw，延时 0.18ms，与同类工作相比，本工作功耗降低了 34.3%以上。

第5章 总结与展望

5.1 总结

神经网络算法广泛应用于物联网场景中，物联网设备受限于电池供电等因素，对功耗有着严格的要求，此外低延时和高准确度对用户来说十分重要。本文设计了一种低功耗神经网络处理器，主要工作包括：

（1）提出了面向神经网络量化分布的 ADC 动态缩放方法。以神经网络的量化为基础，依据输出激活的分布，通过读出电流调整、参数倍增和脉宽调制三步的不同因子，实现了 ADC 取值范围的动态缩放，能较好地提升低位宽 ADC 量程的利用率，以提高存内计算硬件运行神经网络推理的准确度。

（2）设计了面向语音关键词识别的神经网络处理器指令集架构。以存内计算为依托，在配置寄存器、地址寄存器、通用寄存器组、张量运算器和控制器等框架基础上，配合配置、正常和休眠工作模式，建立了支持寄存器迭代计算的 SIMD 专用优化指令集，以实现对高能效的语音关键词识别的支持。

（3）设计了面向语音关键词识别的神经网络处理器微架构。设计了专用处理器的 IF、ID、EX、WB 四级流水线，围绕 S-TC-Resnet 神经网络算法在处理器上的流式推理，提升了数据的重用与计算的性能，测试芯片的仿真验证表明，与同类工作相比，本文的专用处理器实现了较低的功耗。

受条件所限，本文仍存在一些可以探讨的空间，如研究工作以处理器架构设计、逻辑综合和仿真实现为主，而相关的定制电路由他方提供，目前尚未流片，因此仍需进一步加强物理特性验证。

5.2 展望

神经网络算法及其理论发展为本文研究的专用处理器提供了新的契机。神经网络算法及其理论是本文研究的专用处理器的前提，它从多角度提示了神经网络算法实现的本质问题。伴随神经网络算法的效率与认识深度的进一步发展，通过原创机理的神经网络的研究，以及神经网络在样本特征、框架结构、激活函数、损失函数、权值更新、批次归一等方面研究，能够减小模型容量、提升计算效率与准确率，从而为本文研究的专用处理器的未来发展添加助推动力。

硬件与计算能力的发展为本文研究的专用处理器提供更强有力的依托。硬件与计算能力是实现本文研究的专用处理器的物质基础，它为更多更优能力集成打开了大门，比如：集成传感器前端、样本预处理，集成能力更多更优的推理识别、在线学习、端到端学习，集成网络通信、系统控制、其它后处理。而这些能力集成的要求对专用处理器提出了更高的要求，推动专用处理器的研究向更高更深认识层次发展，从而成为本文研究的专用处理器的未来发展的直接动力。

设计工具及其自动化的发展为本文研究的专用处理器提供了额外的支持。设计工具及其自动化是实现本文研究的专用处理器的效率基础，它能够使得设计者与应用者集中在核心问题上，比如：不再羁绊于常规处理器架构而专注于特殊的扩展架构，不再羁绊于常规编译器软件架构而专注于应用软件优化架构，不再羁绊于常规硬件描述而专注于高层计算描述，不再羁绊于数据描述而专注于应用描述，不再羁绊于设计者而专注于应用者。同时设计工具及其自动化也反向推动专用处理器研究更规范化，并成为本文研究的专用处理器未来发展的催化剂。

参考文献

[1] Choi S, Seo S, Shin B, et al. Temporal convolution for real-time keyword spotting on mobile devices[J]. arXiv preprint arXiv:1904.03814, 2019.

[2] Ingolfsson T M, Hersche X W M, Burrello A, et al. Ecg-tcn: Wearable cardiac arrhythmia detection with a temporal convolutional network[C]//2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS). IEEE, 2021: 1-4.

[3] Ingolfsson T M, Hersche M, Wang X, et al. EEG-TCNet: An Accurate Temporal Convolutional Network for Embedded Motor-Imagery Brain–Machine Interfaces[C]//2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, 2020: 2958-2965.

[4] Chong Y S, Goh W L, Nambiar V P, et al. A 2.5 uW KWS Engine with Pruned LSTM and Embedded MFCC for IoT Applications[J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2021.

[5] Zhang S, Huang K, Shen H. A robust 8-bit non-volatile computing-in-memory core for low-power parallel MAC operations[J]. IEEE Transactions on Circuits and Systems I: Regular Papers, 2020, 67(6): 1867-1880.

[6] Shan W, Yang M, Xu J, et al. 14.1 A 510nW 0.41 V low-memory low-computation keyword-spotting chip using serial FFT-based MFCC and binarized depthwise separable convolutional neural network in 28nm CMOS[C]//2020 IEEE International Solid-State Circuits Conference-(ISSCC). IEEE, 2020: 230-232.

[7] Bernardo P P, Gerum C, Frischknecht A, et al. Ultratrail: A configurable ultralow-power tc-resnet ai accelerator for efficient keyword spotting[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2020, 39(11): 4240-4251.

[8] Giraldo J S P, Jain V, Verhelst M. Efficient Execution of Temporal Convolutional Networks for Embedded Keyword Spotting[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2021.

[9] Guo R, Liu Y, Zheng S, et al. A 5.1 pJ/neuron 127.3 us/inference RNN-based speech recognition processor using 16 computing-in-memory SRAM macros in 65nm CMOS[C]//2019 Symposium on VLSI Circuits. IEEE, 2019: C120-C121.

[10] Dbouk H, Gonugondla S K, Sakr C, et al. A 0.44-μJ/dec, 39.9-μs/dec, Recurrent Attention In-Memory Processor for Keyword Spotting[J]. IEEE Journal of Solid-State Circuits, 2020.

[11] Zhou C, Redondo F G, Büchel J, et al. AnalogNets: ML-HW Co-Design of Noise-robust TinyML Models and Always-On Analog Compute-in-Memory Accelerator[J]. arXiv preprint arXiv:2111.06503, 2021.

[12] Giraldo J S P, Verhelst M. Laika: A 5uW programmable LSTM accelerator for always-on keyword spotting in 65nm CMOS[C]//ESSCIRC 2018-IEEE 44th European Solid State Circuits Conference (ESSCIRC). IEEE, 2018: 166-169.

[13] Pete Warden. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. arXiv:1804.03209.Retrieved from https://arxiv.org/abs/1804.03209.

[14] Mittermaier S, Kurzinger L, Waschneck B, et al. Small-footprint keyword spotting on raw audio data with sinc-convolutions[C]//ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2020: 7454-7458.

[15] Rose R C, Paul D B. A hidden Markov model based keyword recognition system[C]//International Conference on Acoustics, Speech, and Signal Processing. IEEE, 1990: 129-132.

[16] Baljekar P, Lehman J F, Singh R. Online word-spotting in continuous speech with recurrent neural networks[C]//2014 IEEE Spoken Language Technology Workshop (SLT). IEEE, 2014: 536-541.

[17] Chen G, Parada C, Heigold G. Small-footprint keyword spotting using deep neural networks[C]//2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2014: 4087-4091.

[18] Sainath T, Parada C. Convolutional neural networks for small-footprint keyword

spotting[J]. 2015.

[19] Fernández S, Graves A, Schmidhuber J. An application of recurrent neural networks to discriminative keyword spotting[C]//International Conference on Artificial Neural Networks. Springer, Berlin, Heidelberg, 2007: 220-229.

[20] Zhang Y, Suda N, Lai L, et al. Hello edge: Keyword spotting on microcontrollers[J]. arXiv preprint arXiv:1711.07128, 2017.

[21] Coucke A, Chlieh M, Gisselbrecht T, et al. Efficient keyword spotting using dilated convolutions and gating[C]//ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2019: 6351-6355.

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

[23] Zhou A, Yao A, Guo Y, et al. Incremental network quantization: Towards lossless cnns with low-precision weights[J]. arXiv preprint arXiv:1702.03044, 2017.

[24] Szymon Migacz. 8-bit Inference with TensorRT. http://on-demand.gputechconf.com/gtc/2017/presentation/s7310-8-bit- inference-with-tensorrt.pdf

[25] VIVIENNE SZE, Yu-Hsin Chen, Tien-Ju Yang and Joel S, Efficient Processing pf Deep Neural Networks: A Tutorial and Survey[C], in Proceedings of the IEEE, 2017: 2295-2329.

[26] Qin H, Gong R, Liu X, et al. Binary neural networks: A survey[J]. Pattern Recognition, 2020, 105: 107281.

[27] Giraldo, J. S. P., and Marian Verhelst. "Hardware acceleration for embedded keyword spotting: Tutorial and survey." ACM Transactions on Embedded Computing Systems (TECS) 20.6 (2021): 1-25.

[28] Zhang Y, Huang K, Xiao R, et al. An 8-Bit in Resistive Memory Computing Core With Regulated Passive Neuron and Bitline Weight Mapping[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2022.

[29] Xiao R, Huang K, Zhang Y, et al. A Low Power In-Memory Multiplication andAccumulation Array with Modified Radix-4 Inputand Canonical Signed Digit Weights[J]. arXiv preprint arXiv:2101.02419, 2021.

[30] Wang J, Wang X, Eckert C, et al. A 28-nm compute SRAM with bit-serial logic/arithmetic operations for programmable in-memory vector computing[J]. IEEE Journal of Solid-State Circuits, 2019, 55(1): 76-86.

[31] Noel J P, Pezzin M, Gauchi R, et al. A 35.6 TOPS/W/mm^2 3-stage pipelined computational SRAM with adjustable form factor for highly data-centric applications[J]. IEEE Solid-State Circuits Letters, 2020, 3: 286-289.

[32] Zhou K, He Y, Xiao R, et al. A Customized NoC Architecture to Enable Highly Localized Computing-On-the-Move DNN Dataflow[J]. IEEE Transactions on Circuits and Systems II: Express Briefs, 2021.

[33] Jia H, Ozatay M, Tang Y, et al. 15.1 A Programmable Neural-Network Inference Accelerator Based on Scalable In-Memory Computing[C]//2021 IEEE International Solid-State Circuits Conference (ISSCC). IEEE, 2021, 64: 236-238.

[34] Rybakov O, Kononenko N, Subrahmanya N, et al. Streaming keyword spotting on mobile devices[J]. arXiv preprint arXiv:2005.06720, 2020.

[35] Li Y, Tagliasacchi M, Rybakov O, et al. Real-Time Speech Frequency Bandwidth Extension[C]//ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2021: 691-695.

[36] Liu C C, Chang S J, Huang G Y, et al. A 10-bit 50-MS/s SAR ADC with a monotonic capacitor switching procedure[J]. IEEE Journal of Solid-State Circuits, 2010, 45(4): 731-740.

[37] Zhang Y, Zeng S, Zhu Z, et al. A 40nm 1Mb 35.6 TOPS/W MLC NOR-Flash based Computation-in-Memory Structure for Machine Learning[C]//2021 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2021: 1-5.

[38] Zeng S, Zhang Y, Zhu Z, et al. MLFlash-CIM: Embedded Multi-Level NOR-Flash Cell based Computing in Memory Architecture for Edge AI Devices[C]//2021

IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS). IEEE, 2021: 1-4.

[39] Gao S, Yang G, Qiu X, et al. Programmable linear RAM: A new flash memory-based memristor for artificial synapses and its application to speech recognition system[C]//2019 IEEE International Electron Devices Meeting (IEDM). IEEE, 2019: 14.1. 1-14.1. 4.

[40] Li W, Huang S, Sun X, et al. Secure-RRAM: A 40nm 16kb compute-in-memory macro with reconfigurability, sparsity control, and embedded security[C]//2021 IEEE Custom Integrated Circuits Conference (CICC). IEEE, 2021: 1-2.

[41] Lu A, Peng X, Li W, et al. NeuroSim validation with 40nm RRAM compute-in-memory macro[C]//2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems (AICAS). IEEE, 2021: 1-4.

[42] Azamat A, Asim F, Lee J. Quarry: Quantization-based ADC Reduction for ReRAM-based Deep Neural Network Accelerators[C]//2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD). IEEE, 2021: 1-7.

[43] Wang Q, Wang X, Lee S H, et al. A deep neural network accelerator based on tiled RRAM architecture[C]//2019 IEEE international electron devices meeting (IEDM). IEEE, 2019: 14.4. 1-14.4. 4.

[44] Zhang B, Li W, Li Q, et al. Autokws: Keyword spotting with differentiable architecture search[C]//ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2021: 2830-2834.

[45] Steinmetz C J, Reiss J D. Efficient neural networks for real-time analog audio effect modeling[J]. arXiv preprint arXiv:2102.06200, 2021.

[46] Stillmaker A, Baas B. Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm[J]. Integration, 2017, 58: 74-81.

[47] Liu B, Shen Z, Huang L, et al. A 1D-CRNN Inspired Reconfigurable Processor for Noise-robust Low-power Keywords Recognition[C]//2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2021: 495-500.

[48] Wu W L, Zhu Y, Ding L, et al. A 0.6 V 8b 100MS/s SAR ADC with minimized DAC capacitance and switching energy in 65nm CMOS[C]//2013 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2013: 2239-2242.

[49] W. J. Tsai et al., "Data retention behavior of a SONOS type two-bit storage flash memory cell," International Electron Devices Meeting. Technical Digest (Cat. No.01CH37224), 2001, pp. 32.6.1-32.6.4, doi: 10.1109/IEDM.2001.979614.

[50] Woo C, Lee M, Kim S, et al. Modeling of charge loss mechanisms during the short term retention operation in 3-D NAND flash memories[C]//2019 Symposium on VLSI Technology. IEEE, 2019: T214-T215.

攻读学位期间取得的科研成果

期刊论文

1. Zhu C, Huang K, Yang S, et al. An efficient hardware accelerator for structured sparse convolutional neural networks on FPGAs[J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 2020, 28(9): 1953-1965.

已受理的发明专利

1. 黄科杰，杨树园，沈海斌，一种低功耗存储器内计算处理器架构（发明专利，申请号：202110265116.8）
2. 黄科杰，杨树园，陆凯晨，沈海斌，一种语音关键词检测专用芯片（发明专利，申请号：202110111358.1）