**毕业设计项目**

**《生理信号检测卷积神经网络硬件加速器》**

**设计方案**

|  |  |
| --- | --- |
| 学 院： | 电子科学与工程学院 |
| 专 业： | 物联网专业 |
| 学 号： | 06117113 |
| 姓 名： | 吴中行 |
| 指导老师： | 刘昊 |
| 日 期： |  |
| 成 绩： |  |

目录

[第 1 章 高能效ECG检测系统功能介绍 4](#_Toc71576206)

[1. 1 ECG检测概述 4](#_Toc71576207)

[1. 2 ECG检测系统功能介绍 5](#_Toc71576208)

[1. 3 系统设计的创新点 6](#_Toc71576209)

[第 2 章 算法总体设计方案 8](#_Toc71576210)

[2. 1 数据集介绍 8](#_Toc71576211)

[2. 2 基于心电图(ECG)片段的卷积神经网络设计 9](#_Toc71576212)

[2.2.1 卷积神经网络拓扑搜寻 12](#_Toc71576213)

[2.2.1 神经网络功耗计算 13](#_Toc71576214)

[2. 3 网络量化设计 14](#_Toc71576215)

[第 3 章 卷积神经网络加速器电路总体设计方案 15](#_Toc71576216)

[3. 1 加速器总体模块 15](#_Toc71576217)

[3.1.1 加速器总体模块介绍 15](#_Toc71576218)

[3.1.2 加速器模块功能介绍 16](#_Toc71576219)

[3. 2 State Processing Unit(SPU)子模块 17](#_Toc71576220)

[3.2.1 主状态机功能介绍 17](#_Toc71576221)

[3.2.2 计算状态机功能介绍 18](#_Toc71576222)

[3.2.3 SPU主要输入输出接口 19](#_Toc71576223)

[3. 3 MemoryController子模块 20](#_Toc71576224)

[3.3.1 MemoryController功能介绍 20](#_Toc71576225)

[3.3.2 MemoryController主要输入输出接口 21](#_Toc71576226)

[3. 4 WeightBuffer子模块 23](#_Toc71576227)

[3.4.1 WeightBuffer功能介绍 23](#_Toc71576228)

[3.4.2 WeightBuffer主要输入输出接口 24](#_Toc71576229)

[3. 5 InOutBuffer子模块 25](#_Toc71576230)

[3.5.1 InOut\_Buffer功能介绍 25](#_Toc71576231)

[3.5.2 InOut\_Buffer主要输入输出接口 28](#_Toc71576232)

[3. 6 Input\_Regfile子模块 29](#_Toc71576233)

[3.6.1 Input\_Regfile介绍 29](#_Toc71576234)

[3.6.2 Input\_Regfile主要输入输出接口 31](#_Toc71576235)

[3. 7 PE\_array子模块 32](#_Toc71576236)

[3.7.1 PE\_array功能介绍 32](#_Toc71576237)

[3.7.2 PE\_array主要输入输出接口 32](#_Toc71576238)

[3. 8 Output\_Regfile子模块 34](#_Toc71576239)

[3.8.1 Output\_Regfile功能介绍 34](#_Toc71576240)

[3.8.2 Output\_Regfile主要输入输出接口 35](#_Toc71576241)

[3. 9 Relu子模块 36](#_Toc71576242)

[3.9.1 Relu模块功能介绍 36](#_Toc71576243)

[3.9.2 Relu模块主要输入输出接口 36](#_Toc71576244)

[3. 10 Pooling子模块 37](#_Toc71576245)

[3.10.1 Pooling模块功能介绍 37](#_Toc71576246)

[3.10.2 Pooling模块主要输入输出接口 37](#_Toc71576247)

[参考文献 38](#_Toc71576248)

# 高能效ECG检测系统功能介绍

## ECG检测概述

心率失常是心血管疾病的起因，中国医学科学院2018年研究数据表明：心律失常造成的心血管疾病是威胁人民健康生命的主要疾病[1]。“早预防，早发现，早治疗”是避免该现象发生的重要举措。我国每年有约54万人死于心律失常导致的猝死[2]，大部分送治不及时猝死案例发生在医院外[3]。心电图(Electrocardiography, ECG) 是心率失常的重要检测手段之一，被广泛应用于临床诊断。部分类型心律失常的早期症状由于持续时间短，不易察觉，但发病突然且症状强烈。

如图1-1所示，诸如贴片式或手环等可穿戴设备可以进行长时间的、实时的心率检测，通过云端发送给护理医师，便于记录和跟踪和诊断患者的心率变化情况。如果在患者出现心律失常时及时发出健康预警，可有效避免更多意外发生。因此，能够进行实时心律失常监测的智能可穿戴设备相关技术受到了广泛关注。



图1-1 ECG心率信号采集、检测及诊断过程

现有的自动心律诊断设备系统通常都基于模式识别的机器学习算法。而不同患者的ECG信号存在很大差异，导致这类主要依赖于特征设计水平的方法泛化能力较低，无法很好地应用于数量庞大的心律失常患者群体。

近年来不断发展的深度学习算法具有从数据中学习更高级特征的能力[4],其层次化的结构使其表现出更好的泛化能力和鲁棒性。神经网络直接从原始数据学习特征，不需要额外的数据处理和特征工程，更适合处理以图像表示心脏活动的ECG信号。因此，将深度神经网络（Deep neural network，DNN）应用于智能心律症状判决设备的研究受到越来越多的关注。

尽管DNN对于心律识别任务有诸多优势，但将DNN模型部署到计算资源及内存空间有限的可穿戴设备中仍受很多限制。

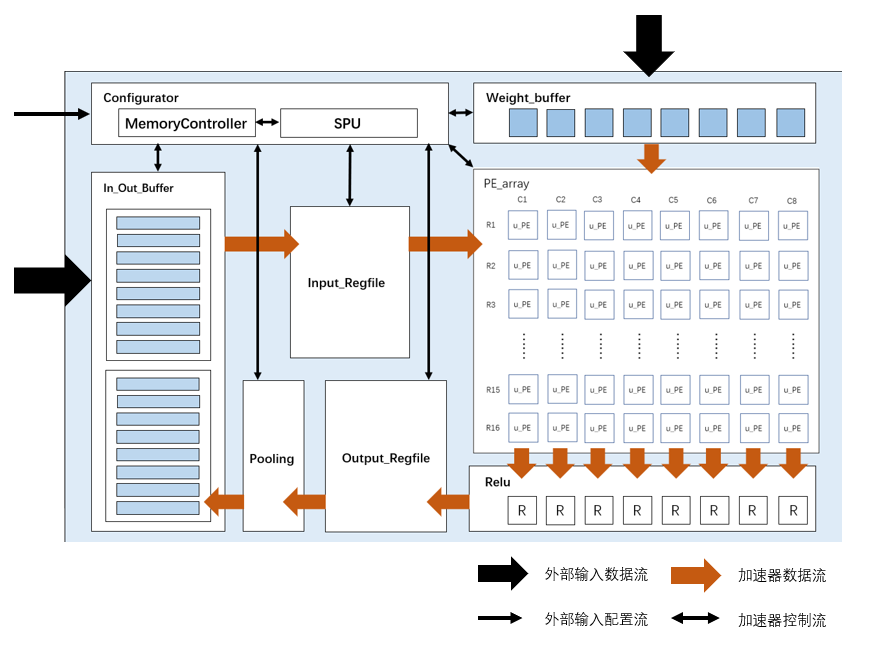
一方面在于现有DNN模型推导需要消耗大量能量。DNN作为计算密集和访存密集型模型，其推导过程包含成千上亿次计算与访存操作，这为硬件资源有限的可穿戴设备带来巨大的功耗负担[5][6]。

另一方面，保存DNN模型需要大量内存空间。DNN的设计者往往通过增加网络深度来提升网络性能[5]，这类设计方法导致DNN模型大小急剧增长[7]。将这些大深度的模型部署端到内存空间有限的可穿戴设备中将是巨大挑战[5]。同时，大规模的网络模型需要使用片外DRAM进行储存，对于片外DRAM的访存又进一步加剧了功耗问题。

因此，对更高能效的智能心率检测系统的研究具有重要价值，在临床诊断，健康状况监测等领域有很大的应用前景。

## ECG检测系统功能介绍

为了解决上述问题，我采用软硬件协同的方法。首先搭建和训练了用于ECG心率不齐检测的CNN网络。然后采用更高效的量化策略对参数进行压缩，将参数量压缩了4倍，大幅减少对计算和存储的需求，降低了设备能耗，同时17类心率不齐检测精度保持在94.4％。然后，我们设计了更为精简的参数访问策略和卷积计算脉动阵列结构来降低数据传输和处理功耗。最终在硬件设计平台上完成了仿真和性能评估。

图1-2 ECG检测系统框图

如图1-2所示为系统框图，该系统主要包括模式配置模块SPU、内存控制模块MC、脉动阵列PE Array、Relu、Pool计算模块、数据存储模块（WeightBuffer、InOutbuffer、Inputregfile和Outputregfile）。其中SPU主要控制整个加速器的状态切换和数据流配置，内存控制模块MC控制数据从外部接口传输到相应的数据存储模块。数据计算主要由PE Array、Relu和Pool模块执行，脉动阵列PE Array执行卷积计算，激活和池化由Relu和Pool模块执行。整个网络层执行完后，通过接口将最终的检测结果输出到外设。各模块详细功能和设计要点介绍在下文第三章。

## 系统设计的创新点

1）**检测方法创新性高：**本作品将心律失常检测神经网络设计与神经网络量化方法设计相结合，设计了一种轻量化的针对ECG序列心律失常检测卷积神经网络算法；

2）**检测分类多且准确率高：**本作品设计的神经网络能够实现端到端心律分类，该网络在17种ECG序列分类任务中获得94.4%准确率；

3）**软硬件协同能效高：**本作品设计了一种参数量少、计算量少的、针对ECG序列的心律失常检测网络。在保证心律检测精度的同时，极大地降低了在推理过程中所需的计算和内存需求。降低了硬件功耗，并且显著提高了硬件检测加速器运算速度，从而提升了能效；

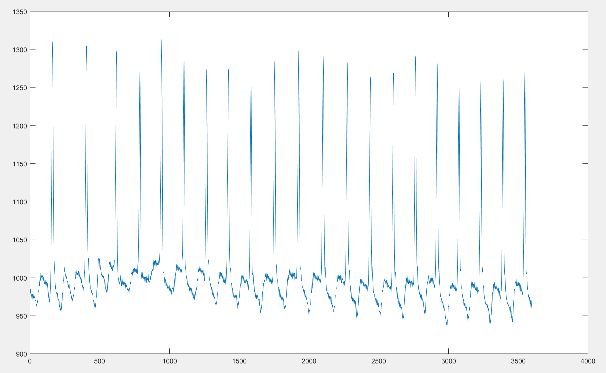
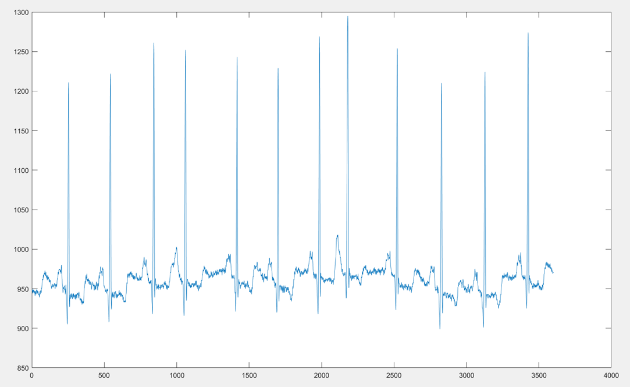
4）**可移植性强：**所需要的外设较为通用，仅需ECG传感器和简单显示外设即可，硬件移植性高，因此整个SOC移植到其他硬件平台也是十分容易；

5）**集成难度低：**采用的是高能效的设计，即整个SOC运行的功耗较低，且通用的外设较为简单，因此集成难度较低，可进一步集成为可穿戴设备芯片。

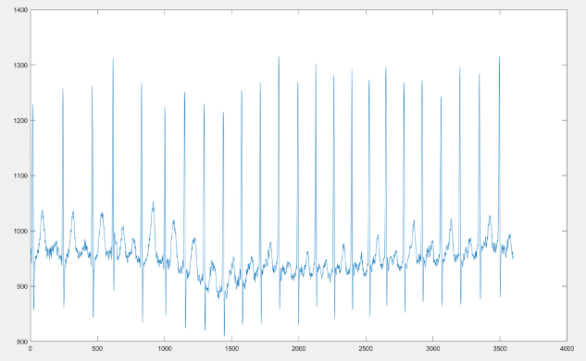
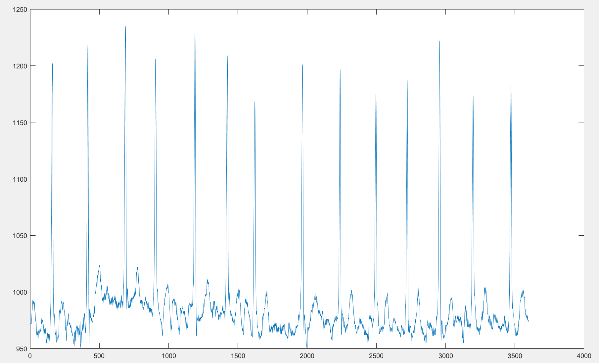
# 算法总体设计方案

## 数据集介绍

本网络用于训练和测试的ECG数据来自Physio Net的MIT-BIH Arrhythmia数据集。该数据集共包含48条长度为30分钟的双导联ECG数据，所有的QRS波都有对应的标注，所有的ECG数据经0.1-100Hz的带通滤波器滤波后，在360Hz下进行采样。如图2-1所示，列举了四种来自modified limb lead II具有代表性的心率类型。



(a) (b)

(c) (d)

图2-1（a）Atrial premature beat心率信号（b）Supraventricular tachyarrhythmia心率信号

（c）Atrial flutter心率信号 （d） Atrial fibrillation心率信号

该数据集的特点为：(1)共有1000条数据，每条数据的长度为10秒，不同样本之间互不重叠；(2)信号共来自45名患者：包括19位女性（23-89岁）和26位男性（32-89岁）；(3)信号包含17种类别：正常窦性心律，起搏器节律和15种类型的心律失常，每种类型至少包含10段信号；(4)信号均来自modified limb lead II。心律类型分布如下表所示：

表2-1 心律类型分布

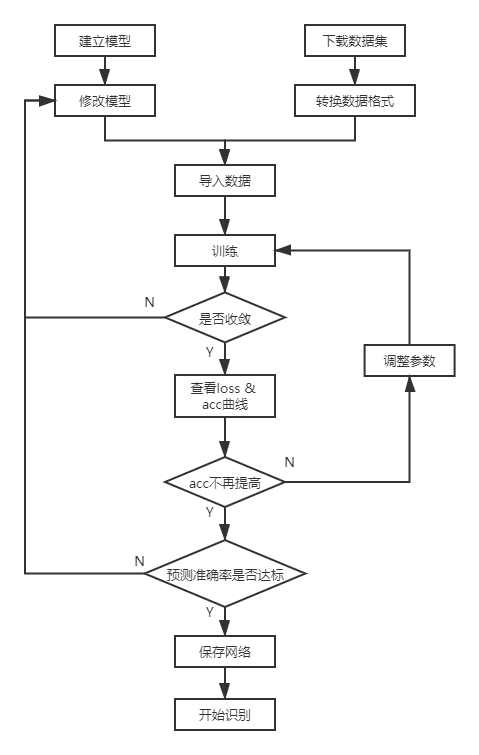
|  |  |  |
| --- | --- | --- |
| No. | Class | No. of Instances |
| 1 | Normal sinus rhythm | 283 |
| 2 | Ventricular tachycardia | 10 |
| 3 | Idioventricular rhythm | 10 |
| 4 | Ventricular flutter | 10 |
| 5 | Fusion of ventricular and normal beat | 11 |
| 6 | Left bundle branch block beat | 103 |
| 7 | Right bundle branch block beat | 62 |
| 8 | Second-degree heart block | 10 |
| 9 | Pacemaker rhythm | 45 |
| 10 | Atrial premature beat | 66 |
| 11 | Atrial flutter | 20 |
| 12 | Atrial fibrillation | 135 |
| 13 | Supraventricular tachyarrhythmia | 13 |
| 14 | Pre-excitation (WPW) | 21 |
| 15 | Premature ventricular contraction | 133 |
| 16 | Ventricular bigeminy | 55 |
| 17 | Ventricular trigeminy | 13 |
|  | Total | 1000 |

在实验阶段，我们使用70%的数据作为训练集训练网络，剩余的30%的数据作为测试集来对训练后的网络性能进行测试。考虑到数据集中不同类型的样本数量差异巨大可能会导致训练效果不佳，我们对训练集的数据进行了样本均衡，对样本数较少的种类进行过采样，使不同类型样本数量趋近一致。

## 基于心电图(ECG)片段的卷积神经网络设计

本作品设计了一种新型的、用于心律识别的面向长序列的卷积网络结构，网络训练流程如图2-2所示，网络基础结构如图2-3所示的直连型网络。

首先我们对原始数据集数据进行过采样处理，对所搭建的CNN网络进行参数训练。然后，对网络层参数量化，直到参数的压缩比例和心率检测精度均满足要求。

图2-2 基于ECG序列的卷积神经网络模型训练流程

该网络使用一段10秒的ECG序列（包含3600个样本点）作为输入。该网络可以根据输入的信号端对端地输出信号所属的心律类型。这种设计避免了在信号输入网络之前的心拍检测的阶段。尽管许多基于心拍的心率检测网络显示出优越的性能，但由于神经网络的输入通常是固定长度的序列，而这类网络的输入是一段完整的心拍信号且不同心律的持续时间差异很大，因此不仅需要在输入网络之前对输入信号执行QRS波形检测，还需要根据检测结果对数据处理。该网络避免了对心拍的检测，能够在不丢失相邻心拍特征的同时避免了心拍分割阶段的功耗，进一步简化了系统结构。

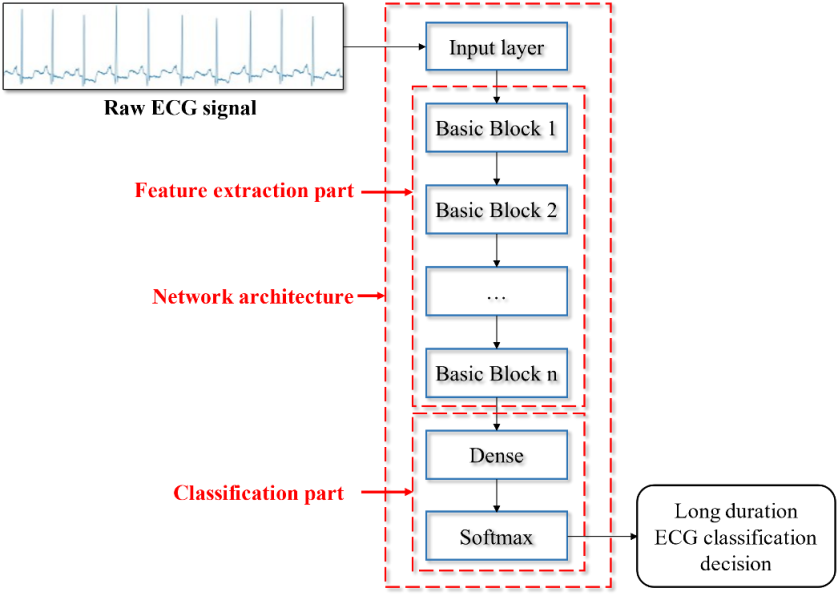


图2-3 直连型网络结构图

本作品所设计的网络结构采用直连型网络结构。该网络由下图2-4所示的基本块和两个全连接层串联组成，每个基本块受两个超参数限制：卷积核k的大小和卷积核数m。由卷积和下采样层组成的基本块用于提取ECG信号特征，全连接层用于综合全局特征进行分类，使用softmax作为激活函数包含17个输出节点，分别对应属于每种类型心律的概率。考虑到心电信号的形态特征是判断心律不齐的重要特征，本设计还使用大核卷积核对一段较长的临近采样点进行特征提取，以此获得较长时间跨度上心电图信号所包含的信息。

图2-4 网络计算过程图

此外，有研究显示网络的深度可以改善网络性能，但是网络结构太深会带来更多的计算功耗和内存。我们对该网络的性能和深度进行了探索，对具有不同个数基本块的网络结构的性能和内存进行对比，来寻找性能与内存最合适的点。我们对8种不同的深度配置进行试验，在相同实验场景下对比了不同深度配置的网络识别准确率与网络占用内存大小，实验结果显示使用6个basic block实现了最高的94.4%的识别准确率，同时参数占用的内存最小，仅为314.75KB。

### 2.2.1 卷积神经网络拓扑搜寻

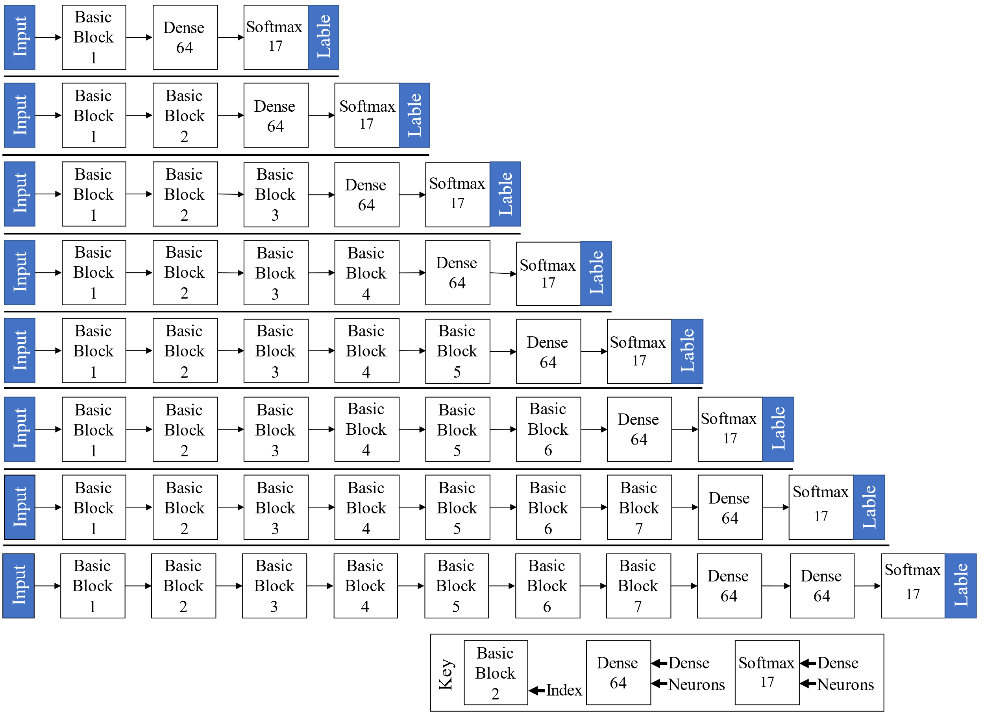
在本设计中，我们先对不同的深度的网络结构与网络识别精度和所需内存的关系进行实验，根据网络的识别准确率和内存确定最终的网络拓扑结构。图2-5为8种由基本块组成的不同深度的网络拓扑结构图。所有配置的结构都由多个基本块和全连接层串联组成，基本块用于提取ECG信号特征，全连接层用于综合全局特征进行分类。

图2-5 网络拓扑结构图

我们使用相同的训练样本对所有结构的神经网络进行3000次训练，并选取训练过程中整体准确率最高的模型。对不同拓扑结构的网络的整体准确率、网络内存大小进行统计。实验结果图2-6所示，网络识别准确率随着基本块的增加而增大，当基本块的数量为6时，识别准确率达到最高，为94.4%，之后增加基础块数量反而准确率略微下降。与识别精度曲线不同，由于受到卷积层参数和全连接层参数的影响，内存大小总体呈现先上升后下降的趋势，当基本块为2和7时，所需内存达到局部最小值，分别为341KB和315KB。

图2-6 网络精度与网络大小关系图

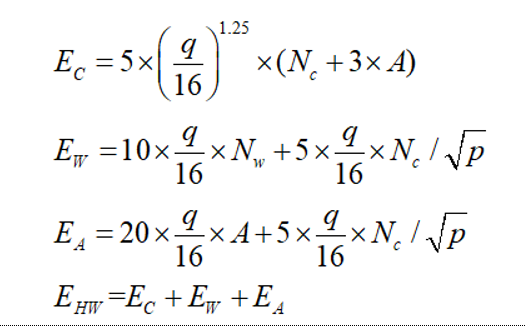
综上所述，当网络结构由6个基本块和2全连接层组成时，识别准确率最高为94.4%，同时，网络所需内存大小最低为343KB。因此，我们选择该配置的结构作为本设计最终的模型。其网络结构及参数分布如表2所示。

表2-2 网路具体结构表

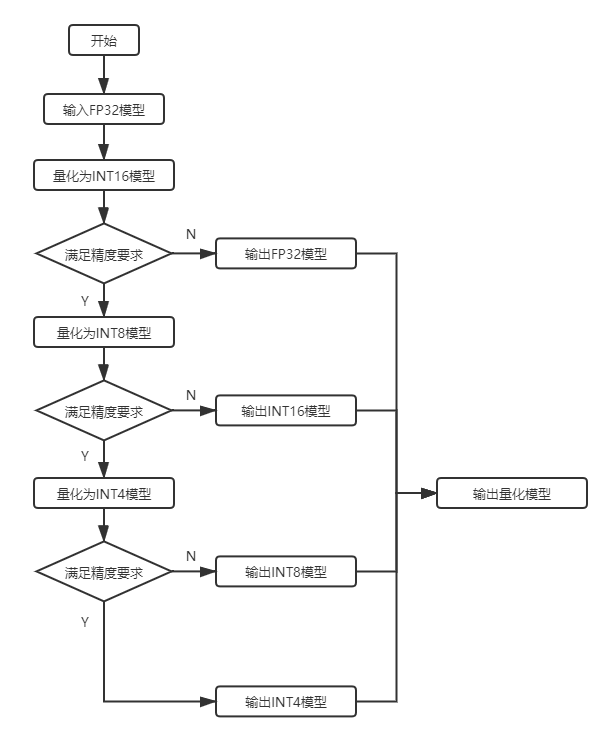
|  |  |  |  |
| --- | --- | --- | --- |
| **Layer** | **Conv** | **Pooling** | **Parameters** |
| Basic Block1 | 16×1×8 | 7×2 | 128 |
| Basic Block2 | 12×1×16 | 4×2 | 1536 |
| Basic Block3 | 8×1×32 | 5×2 | 4096 |
| Basic Block4 | 8×1×64 | 4×2 | 16384 |
| Basic Block5 | 4×1×64 | 2×2 | 16384 |
| Basic Block6 | 3×1×72 | 2×2 | 13824 |
| Global Average Pooling | - | - | 0 |
| Softmax | 72×17 | - | 1241 |

### 神经网络功耗计算

此外，考虑到该网络最终需要在硬件中进行实现。我们还对该网络的功耗进行建模计算。我们建立了网络层的功耗模型, 功耗由三部分组成，分别为计算功耗，权重访存功耗和激活值访存功耗。相关模型如下。



## 网络量化设计

对于心律失常自动诊断的可穿戴设备来说，计算和数据访存是功耗的主要来源。尽管目前已经有研究对ECG信号进行压缩处理，但是少有工作将注意力放在ECG检测神经网络参数压缩中。由于神经网络使用训练好的权重进行数据处理，同时由于网络的工作机制等原因，这些权重存在多次复用的特点，因此我们认为对神经网络权重进行压缩能有效降低硬件消耗。

尽管该网络规模相对较小，但是在网络推理阶段，大量高位宽数值计算，仍会导致该网络在进行心律失常识别过程中产生的大量的功耗和内存需求。本设计根据神经网络的特点和心律失常检测场景的特征，采用量化的方式将网络权重量化为INT8，进一步降低神经网络在使用过程中的硬件开销。

图2-7量化算法整体框架

# 卷积神经网络加速器电路总体设计方案

## 加速器总体模块

### 加速器总体模块介绍

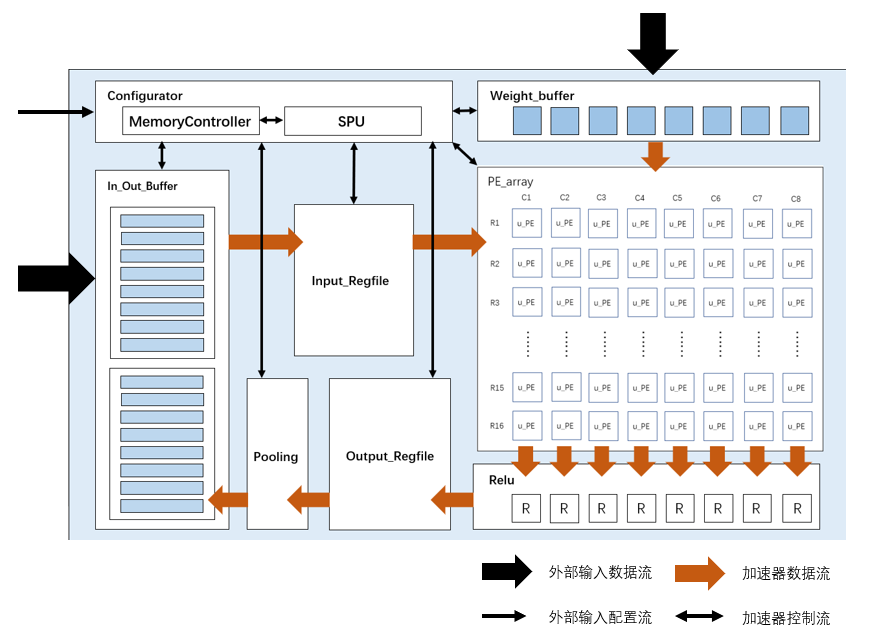
如下图所示，加速器分为Configurator, Weight\_buffer, PE\_array, In\_Out\_Buffer, Input\_Regfile, Output\_Regfile, Relu, Pooling共8个模块，有三股外部的数据流输入，分别是网络参数数据流、权重数据流和ECG信号数据流。

图3-1 硬件加速器架构图

ECG信号输入到第一块RAM中，接着按批次输出到Input Regfile，Input Regfile把数据按一定的顺序输出到PE阵列里与权重进行卷积计算。计算结束，把计算结果输出到Output Regfile，待数据整理好将其发送到Relu Pooling模块进行激活池化操作，最后池化结束将结果输出到Inout Buffer的第二块RAM中，待所有结果都进入到RAM之后，第一层运算结束。第二块RAM的数据是下一层的输入，第一块RAM即将存放下一层的输出结果，两块RAM如此乒乓操作，直到6层的卷积层计算结束。

Configurator是整个加速器的控制部分，里面包含SPU（State Processing Unit）和Memory controller。SPU是控制整个加速器工作状态的状态机，Memory controller主要工作是配置网络参数给给个模块，并控制Weight Buffer和Inout Buffer的读写功能，两个模块协同工作控制整个加速器的运行。

Inout Buffer有两块RAM，每块RAM里面有个8个bram，每个bram的深度是1024，位宽是8bits。Weight Buffer有8个bram，每个bram的深度是2048，位宽也是8bits。ECG信号输入到第一块RAM中，然后再按批次输出到PE阵列进行计算。计算一定周期之后，最终把计算结果输出到Inout Buffer的另一块RAM中，直到所有计算的结果都进入到RAM之后，第一层运作结束。第二块RAM的数据是下一层的输入，第一块RAM即将存放下一层的输出结果，两块RAM如此乒乓操作，直到6层的卷积层计算结束。

PE阵列的大小是16\*8，行数16，列数8,8列PE的权重输入对应着Weight Buffer的8个bram的输出，16行PE的特征数据输入是Input Regfile的输出。PE阵列的工作原理如下：在第一个clk的时候，第一列16个PE同时进行第一次计算，然后第二个clk的时候，第二列PE进行第一次计算，第一列PE进行第二次计算，如此从右至左地脉动，直到第八个周期，第八列PE进行第一次计算，第一列PE进行第八次计算，整个脉动阵列被激活起来，都在进行着计算工作。

当计算完一定的周期之后，PE阵列开始输出数据，每列都会有16个计算结果，每个计算结果从上往下逐个往外输出，由于每一列输入之间相隔一个周期，所以每列间的输出都是相隔一个周期。在第一个clk的时候，第一列的PE输出第一个计算结果，然后第二个clk的时候，第二列PE输出第一个计算结果，第一列PE输出第二个计算结果，如此从右至左的顺序，直到第八个周期，第八列PE输出第一个计算结果，第一列PE输出第八个计算结果，整个脉动阵列都在进行着输出工作。

### 加速器模块功能介绍

表3-1 硬件加速器模块功能介绍

|  |  |  |
| --- | --- | --- |
|  | **功能模块** | **功能** |
| 1 | ECG\_Top | 基于ECG信号的心律不齐检测加速器顶层模块。 |
| 2 | State Processing Unit(SPU) | 整个加速器的工作状态由SPU产生与控制，使加速器完成计算工作。 |
| 3 | MemoryController | 从外部接收网络配置参数，并配置其他模块，并从SPU接收加速器工作状态，并控制两个buffer的读写功能。 |
| 4 | PEArray | 高并行度卷积计算阵列，进行CNN网络模型中的卷积计算。每个PE进行一个卷积核计算，每个cycle进行一次MAC操作。 |
| 5 | WeightBuffer | 缓存weight数据，输出权重到PE阵列进行计算。 |
| 6 | InOutBuffer | 缓存输入ECG数据及各输出中间值数据，并按照网络的计算规律将数据输出到InputRegfile中。 |
| 7 | InputRegfile | 缓存待计算map数据,并配合Weight buffer将数据输出到PE阵列中。 |
| 8 | OutputRegfile | 缓存PEarray输出计算结果，并按照PE的输出规律将数据输出到ReluPool模块。 |
| 9 | Relu&Pool | 执行relu激活和Pooling池化，并按照InOut buffer的写入规律，输出到InOut buffer。 |

## State Processing Unit(SPU)子模块

SPU模块主要控制整个系统状态转换。分别有两个状态机：主状态机和计算状态机。

### 主状态机功能介绍

**主状态机**：对加速器实现一次推理的控制。状态机状态转换图如下所示。

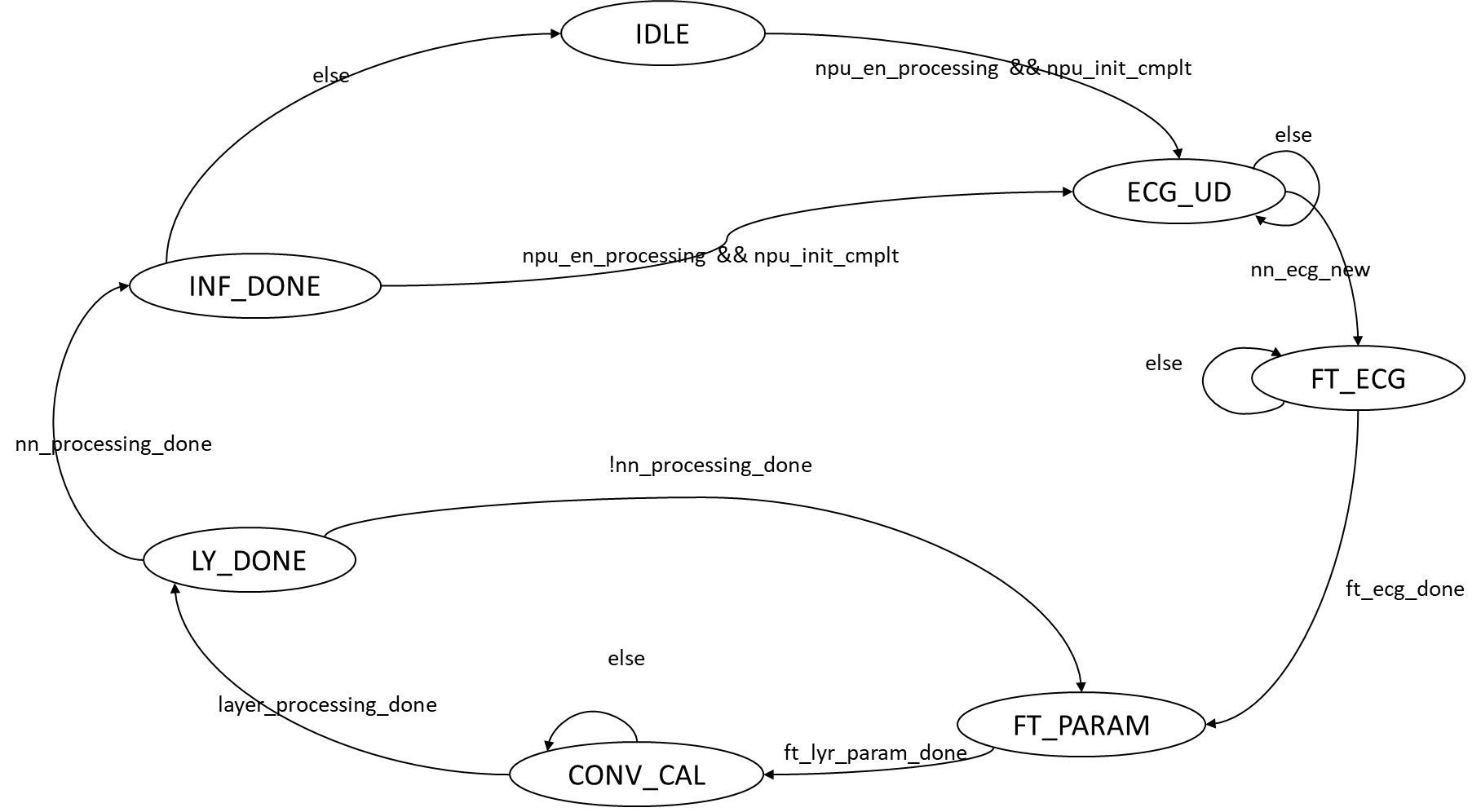


图3-2 加速器主状态机原理图

主状态机的状态分别为：

**IDLE**：空闲状态；

**ECG\_UD**：等待ECG信号采集芯片通过SPI接口将ECG信号保存在片外存储器中；

**FT\_ECG**：将保存在片外的ECG信号通过Memory\_Controller从片外存储器载入到In\_Out\_Buffer中；

**FT\_PARAM**：从片外存储器(或片内存储器)中载入当前层的网络配置参数，并将其输出给各计算模块和存储模块；

**CONV\_CAL**：卷积计算状态，控制对加速器进行当前层的数据读写和计算。

**LY\_DONE**：当前层计算完毕；

**INF\_DONE**：所有层计算完毕的状态。

### 计算状态机功能介绍

计算状态机主要是实现单次计算的控制。

计算状态机的状态分别为：

**OR\_IDLE**：空闲状态；

**OR\_FT\_WT**:加载weight值的状态，并判断ft\_wt\_done是否置1，若置1则表示权值加载完毕，进入计算状态，否则继续保持OR\_FT\_WT状态；

**OR\_CAL**:进行卷积计算的状态，并判断lyt\_cal\_done是否置1，若置1则表示卷积计算完毕，进入计算结束状态，否则继续保持OR\_CAL状态；

**OR\_DONE**:卷积计算完毕的状态，回到空闲状态。

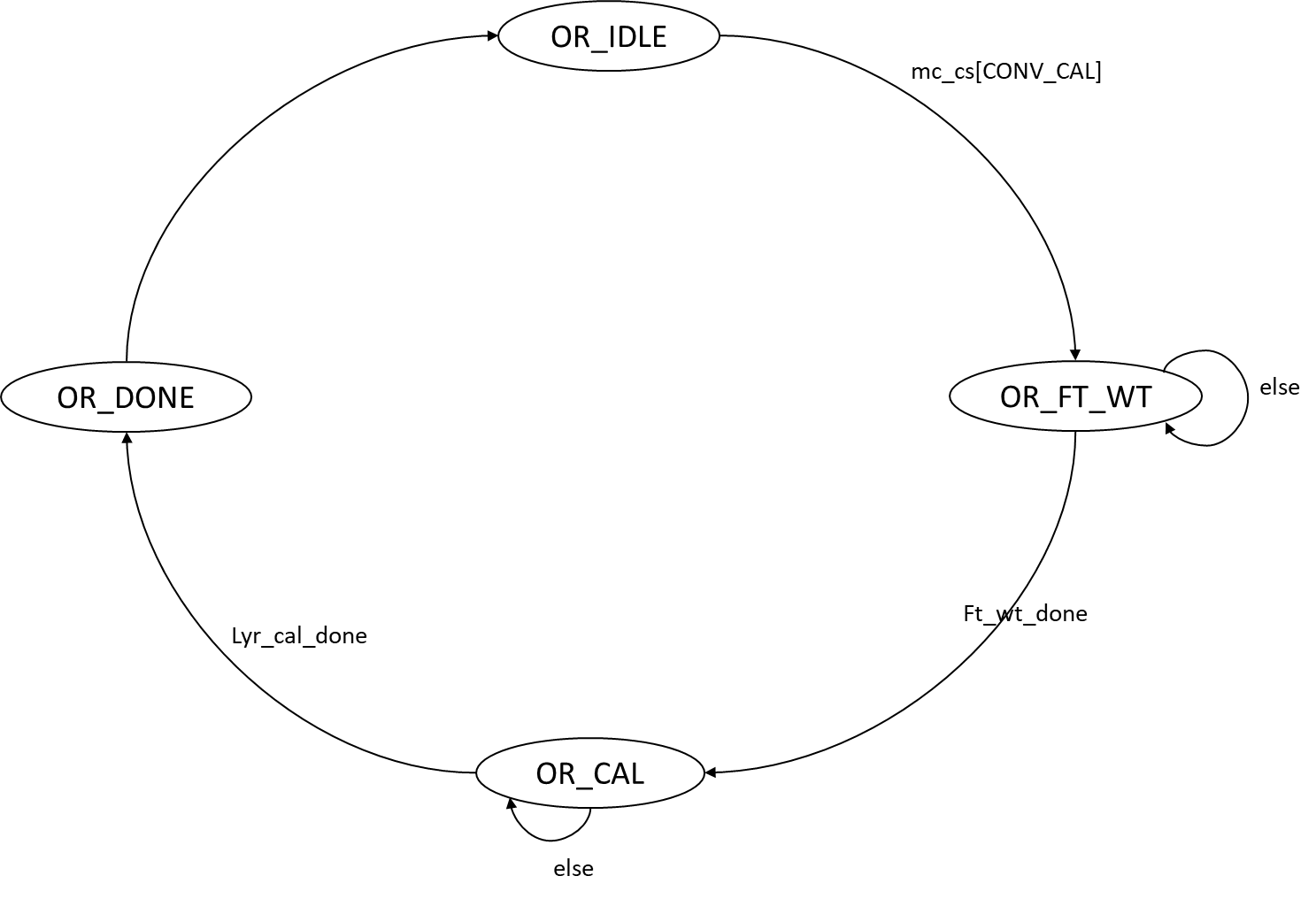


图3-3 加速器计算状态机原理图

### SPU模块主要输入输出接口

表3-2 SPU主要输入接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入接口名称 | 来源(模块—接口名) | 作用 |
| 1 | clk\_cal | 外部—clk\_ cal | 系统时钟信号 |
| 2 | rst\_cal\_n | 外部—rst\_cal\_n | 复位信号 |
| 3 | SPI\_start | 外部—SPI\_start | SPI读入数据标志 |
| 4 | ft\_lyr\_param\_done | MemCtrl—ft\_lyr\_param\_done | 取当前层参数完成标志 |
| 5 | ft\_wt\_done | MemCtrl—ft\_wt\_done | 取当前层weight完成标志 |
| 6 | ft\_ecg\_done | MemCtrl—ft\_ecg\_done | 取ecg信号完成标志 |
| 7 | memct\_init\_cmplt | MemCtrl—memct\_init\_cmplt | Memctrl初始化完成标志 |
| 8 | lyr\_cal\_done | MemCtrl—lyr\_cal\_done | 当前层计算完毕标志 |

表3-3 SPU主要输出接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输出接口名称 | 输出模块—接口 | 作用 |
| 1 | mc\_cs | MemCtrl—mc\_cs | 主状态机现在的状态 |
| 2 | mc\_ns | MemCtrl—mc\_ns | 主状态机下一个状态 |
| 3 | or\_cs | MemCtrl—or\_cs | 计算状态机现在的状态 |
| 4 | or\_ns | MemCtrl—or\_ns | 计算状态机下一个状态 |
| 5 | nn\_layer\_cnt | MemCtrl—nn\_layer\_cnt | 当前层数 |
| 6 | ecg\_len | MemCtrl—ecg\_len | 写入ECG信号的长度 |
| 7 | nn\_ecg\_saddr | MemCtrl—nn\_ecg\_saddr | ECG信号在ram里的起始地址 |
| 8 | nn\_wt\_saddr | MemCtrl—nn\_wt\_saddr | Weight权重在ram里的起始地址 |
| 9 | nn\_lyr\_param\_saddr | MemCtrl—nn\_lyr\_param\_saddr | 网络参数在ram里的起始地址 |

## MemoryController子模块

### MemoryController模块功能介绍

MC模块主要是控制并实现加速器与外部信号的数据交换，数据包括网络层配置参数，网络层权重，ECG输入信号。交换的数据类型由SPU模块状态机决定。具体功能如下：

**1、读取SPI输入ECG数据**：

从外部读取ECG信号，计算写入地址，并将地址传递给In\_Out\_Buffer，进行数据写入。对应主状态机FT\_ECG状态。

**2、读取网络层配置参数**

由主状态机控制，读取当前网络层配置参数。对应主状态机FT\_PARAM状态。

**3、读取网络层权重**：

由计算状态机控制，通过总线从外部读取当前网络层权重并写入WeightBuffer。对应计算状态机OR\_FT\_WT状态。

**4、计算Output\_Regfile输出特征图写入In\_Out\_Buffer地址：**

接收Output\_Regfile数据输出使能信号，计算写入地址，并将地址传递给In\_Out\_Buffer进行数据写入。对应计算状态机OR\_CAL状态。

**5、计算In\_Out\_Buffer写入Input\_Regfile地址**：

接收PE\_Array计算结束信号pe\_end，计算输出地址，并将地址传递给In\_Out\_Buffer进行数据读出。对应计算状态机OR\_CAL状态。

### MemoryController模块主要输入输出接口

表3-4 MemoryController主要输入接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入接口名称 | 来源(模块—接口) | 作用 |
| 1 | clk\_cal | 外部—clk\_cal | 系统时钟信号 |
| 2 | rst\_cal\_n | 外部—rst\_cal\_n | 复位信号 |
| 加速器状态机信号 | | | |
| 3 | mc\_cs | SPU—mc\_cs | 加速器主状态机当前状态 |
| 4 | mc\_ns | SPU—mc\_ns | 加速器主状态机下一时刻状态 |
| 5 | or\_cs | SPU—or\_cs | 加速器计算状态机当前状态 |
| 6 | or\_ns | SPU—or\_ns | 加速器计算状态机下一时刻状态 |
| 7 | pe\_end | PE\_Array—pe\_end | PE阵列计算状态 |
| 网络层配置参数 | | | |
| 7 | nn\_layer\_cnt | SPU—nn\_layer\_cnt | 当前处理网络层索引 |
| 8 | ecg\_len | SPU—ecg\_len | 待处理ECG信号长度 |
| 9 | nn\_lyr\_param\_saddr | SPU—nn\_lyr\_param\_saddr | 网络层配置参数起始地址 |
| 10 | nn\_wt\_saddr | SPU—nn\_wt\_saddr | 网络权重起始地址 |
| 11 | Data\_I\_vld | RP—RP \_Data\_Ovld | 特征图输入使能信号 |

表3-5 MemoryController主要输出接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输出接口名称 | 输出模块—接口 | 作用 |
| 1 | memct\_init\_cmplt | SPU—memct\_init\_cmplt | MemoryController已复位 |
| 2 | ft\_lyr\_param\_done | SPU—ft\_lyr\_param\_done | 当前层配置参数写入完成标志位 |
| 3 | ft\_ecg\_done | SPU—ft\_ecg\_done | ECG信号写入完成标志位 |
| 4 | ft\_wt\_done | SPU—ft\_wt\_done | 权重写入完成标志位 |
| 5 | lyr\_cal\_done | SPU—lyr\_cal\_done | 当前层计算结束标志位 |
| 7 | rd\_done | PE\_Array—rd\_done | 下次计算数据输出完成翻转pe\_end |
| 8 | rd\_addr | In\_Out\_Buffer—rd\_addr | IOB输出数据地址 |
| 9 | wr\_addr | In\_Out\_Buffer—wr\_addr | IOB输入数据地址 |
| 10 | Data\_O\_vld | Input\_Regfile—Data\_I\_vld | IOB数据输出至IR使能 |
| 11 | Bm\_cnt | Input\_Regfile—Bm\_cnt | 该时钟传输数据批次，计算IR地址 |
| 12 | final\_column | RP—final\_column | OR最后一次数据传输使能 |
| 13 | final\_zeros | RP—final\_zeros | OR最后一次数据补0数量 |
| 14 | wt\_I\_addr | WeightBuffer—wt\_I\_addr | 写入weight的地址 |
| 15 | wt\_I\_vld | WeightBuffer—wt\_I\_vld | 写入weight有效信号 |
| 16 | wt\_C0\_addr- wt\_C7\_addr | WeightBuffer—wt\_C0\_addr-wt\_C7\_addr | 读出weight的地址 |
| 17 | wt\_C0\_O\_vld- wt\_C7\_O\_vld | WeightBuffer—wt\_C0\_O\_vld-wt\_C7\_O\_vld | 读出weight的地址的有效信号 |
| 18 | CalCycle | PE\_array—CalCycle | 每次需要计算的周期数 |
| 19 | pass\_cnt | PE\_array—pass\_cnt | 每K次计算所需要的周期数 |
| 20 | Pool\_K | Output Regfile & RP—P | 池化大小 |
| 21 | Pool\_S | Output Regfile—S | 池化步长 |

## WeightBuffer子模块

### WeightBuffer模块功能介绍

**作用**：由8个bank组成，每个bank的大小是2048byte，8个输入输出端口用于接收和输出weight权重值，按照一定的顺序输出weight值到PE阵列里进行计算。

### WeightBuffer模块主要输入输出接口

表3-6 WeightBuffer主要输入接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入接口名称 | 输入模块—接口 | 作用 |
| 1 | clk\_cal | 外部—clk\_cal | 系统时钟信号 |
| 2 | rst\_ cal\_n | 外部—rst\_cal\_ n | 复位信号 |
| 3 | layer2weight\_cnt | SPU—nn\_layer\_cnt | 当前处理网络层索引 |
| 4 | wt\_I\_addr | MemoryController—wt\_I\_addr | 写入weight的地址 |
| 5 | wt\_I\_vld | MemoryController—wt\_I\_vld | 写入weight有效信号 |
| 6 | wt\_C0\_addr- wt\_C7\_addr | MemoryController—wt\_C0\_addr- wt\_C7\_addr | 读出weight的地址 |
| 7 | wt\_C0\_O\_vld- wt\_C7\_O\_vld | MemoryController—wt\_C0\_O\_vld- wt\_C7\_O\_vld | 读出weight的地址的有效信号 |

表3-7 WeightBuffer主要输出接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输出接口名称 | 输出模块—接口 | 作用 |
| 1 | wt\_Ovld0- wt\_Ovld7 | PE\_array—IweightVld0- IweightVld7 | 读出weight的有效信号 |
| 2 | kernel\_C0\_O- kernel\_C7\_O | PE\_array—Iweight0- Iweight7 | 读出weight接口 |

## InOutBuffer子模块

### InOut\_Buffer模块功能介绍

**作用**：

由2块4096\*8bit的SRAM组成，每块有8个bank提供8个输入输出端口用于接收原始ECG信号和OutputRegfile的输出，同时将数SRAM中的数据输出至InputRegfile。

**乒乓操作**：

两块SRAM用于乒乓操作。根据特征图所属于的网络层索引，在同一时刻，输入数据流选择单元选择其中一块SRAM，将输入数据按通道，从SRAM的第一个地址放置特征值；同时输出数据流选择单元选择另一块SRAM，将其保存的特征值输出至InputRegfile。

**数据输入**：

数据由ReLU&Pooling模块输入，**每次**写入Bm(8)个数据，每个bank写入一个数据，8个bank的写入地址相同。仅由ReLU&Pooling模块控制，写入地址由In\_Out\_Buffer产生。数据输入顺序依次为：输出Map0的第1部分，输出Map1的第1部分，输出Map2的第1部分，以此类推直到输出MapN的第1部分；然后开始输出Map的第2部分数据输入；重复上述操作直至完成所有输出Map的写入。



图3-4 输出通道展示图

Step 1：写入Omap 1的第一列；

Step 2：写入Omap 2的第一列；

...

Step N：写入Omap N的第一列；

Step N+1到Step 2N：重复上述Omap的写入第二列；

重复上述步骤，写入所有Omap。

**数据输出**：

数据输出到Input\_Regfile。由PE\_Array控制数据输出的开始，PE\_Array计算结束信号置高开始数据输出，每次进行ceil(Hu/Bm)个时钟的数据传输，每个时钟周期输出Bm个数据。

数据输出顺序依次为：(1)输出InMap0的第1部分，输出InMap1的第1部分，输出InMap2的第1部分，以此类推直到完成InMapN的第1部分输出；(2)依次输出InMap0~M的第2部分，重复**ceil(Hu/Bm)**次；(3)重复操作(4)直至完成InMap所有部分的传输。

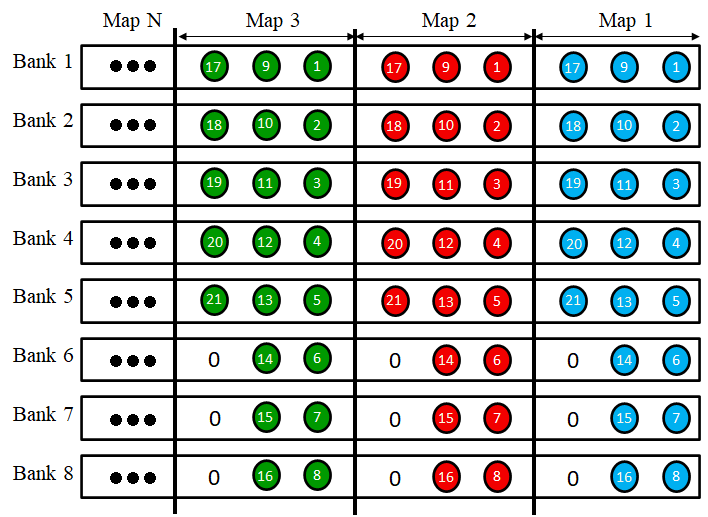


图3-5 Inout Buffer原理图

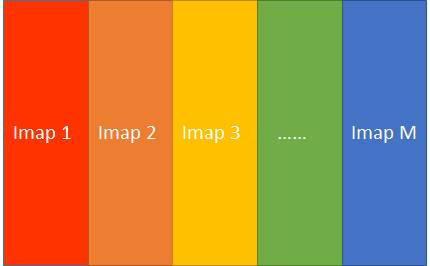


图3-6 输入通道展示图

**数据读取：**

Step 1：读取Imap1的0-Bm\_times-1列；

Step 2：读取Imap2的0-Bm\_times-1列；

...

Step M：读取ImapM的0-Bm\_times-1列；

由于PE\_Array的C≤N，所以需要重复Step1-StepM ，fetch\_N\_times次;

Step fetch\_N\_time+1：读取Imap1的Bm\_times到2\*Bm\_times-1列；

Step fetch\_N\_time+2：读取Imap2的Bm\_times到2\*Bm\_times-1列；

...

Step fetch\_times+M：读取ImapM的Bm\_times到2\*Bm\_times-1列；

重复上述操作M个Step ，fetch\_N\_times次

依此类推，直至完成所有map的所有列的fetch\_N\_times次写出。

### InOut\_Buffer模块主要输入输出接口

表3-8 InOut\_Buffer主要输入接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入接口名称 | 输入模块—接口 | 作用 |
| 1 | clk\_cal | 外部—clk\_cal | 系统时钟信号 |
| 2 | rst\_cal\_n | 外部—rst\_cal\_n | 复位信号 |
| 3 | nn\_layer\_cnt | Memory\_Controller—nn\_layer\_cnt | 当前处理的网络层索引 |
| 4 | wr\_addr | Memory\_Controller—wr\_addr | 输入数据写地址 |
| 3 | IOB\_Data\_I\_vld | RP—RP\_OVld | RP输入数据写使能 |
| 4 | IOB\_Data\_I0- IOB\_Data\_I7 | RP—RP\_Odata0- RP\_Odata7 | RP输入Bank0~Bank7的数据 |
| 5 | IOB\_Data\_I\_vld\_ECG | SPI—Data\_I\_vld | 输入ECG数据写使能 |
| 6 | IOB\_Data\_I0\_ECG- IOB\_Data\_I7\_ECG | SPI—Data\_O0- Data\_O7 | SPI输入Bank0~ Bank7的ECG数据 |
| 7 | rd\_addr | Memory\_Controller—rd\_addr | 输入数据读地址 |
| 8 | Mem\_Data\_Ivld | Memory\_Controller—Data\_O\_vld | 输出数据读使能 |
| 9 | SPI\_start | SPI-SPI\_start | SPI开始传数据有效值 |
| 10 | IOB\_FC\_vld | PE\_Array-FC\_OVld | FC层输入有效信号 |
| 11 | IOB\_FC\_I0-IOB\_FC\_I7 | PE\_Array-FC\_OMap1-FC\_Omap7 | FC层输入有效数据 |

表3-9 InOut\_Buffer主要输出接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输出接口名称 | 输出模块—接口 | 作用 |
| 1 | IOB\_Data\_O0- IOB\_Data\_O7 | Input\_Regfile—IR\_Data\_I0- IR\_Data\_I7 | Bank0- Bank7输出数据 |
| 2 | IOB\_Data\_O\_vld | Input\_Regfile—IR\_Data\_I\_vld | 输出数据的有效值 |

## Input\_Regfile子模块

### Input\_Regfile模块功能介绍

**作用**：

用于缓存PE Array在下一次计算中所使用到的输入数据。由于卷积核大小K>卷积核步长S，在PE Array的相邻两次计算过程中的输入数据存在overlapping，使用Input Regfile可以避免对In Out Buffer中的同一个地址多次访问的现象。

**内存大小**：

Input regfile用于实现输入数据复用的功能。Input regfile存放Hu×1的输入数据，其中Hu为PE Array在一次计算中所涉及的输入数据数量，Hu与当前层的卷积核大小K，卷积核步长S和PE Array的Map输入端口数量R有关，其计算公式如下：

Hu=(R-1)×S+K

本设计中，各网络层对应的Input Regfile大小如下表，最终确定256Byte。

表3-10 每层网络Input Regfile所需大小表

|  |  |  |
| --- | --- | --- |
| 网络层索引 | Hu | 对应Input Regfile大小 |
| 1(Conv1) | 46 | 46 Byte |
| 2(Conv2) | 42 | 42 Byte |
| 3(Conv3) | 23 | 23 Byte |
| 4(Conv4) | 19 | 19 Byte |
| 5(Conv5) | 18 | 18 Byte |
| 6(Conv6) | 256 | 256 Byte(max) |
| 7(Dense) | 72 | 72 Byte |

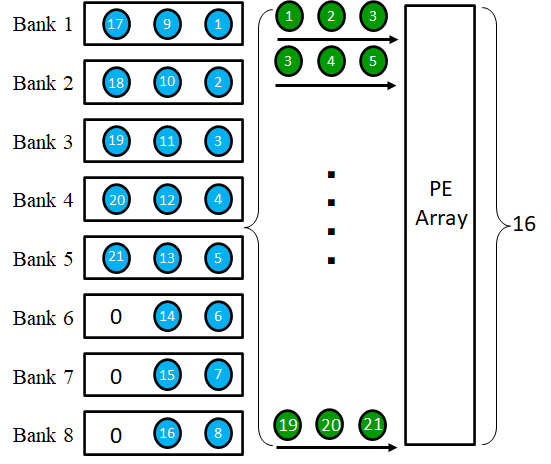
****

图3-7 Input Regfile原理图

**数据输出**：

Input Regfile接收到PE Array 结束信号为高电平时；同时接收到来自In\_Out\_Buffer的输出数据有效信号后开始数据写入操作。PE Array 结束信号为低电平时计算的信号，停止将数据写入操作。数据由In\_Out\_Buffer写入Input\_Regfile，读写时钟周期由In\_Out\_Buffer控制。

**数据输入**：

Input Regfile接收到PE Array 结束信号为低电平时；同时到来自In\_Out\_Buffer的输出数据有效信号置0后开始数据读出操作。PE Array 结束信号为高电平时计算的信号，停止将数据读出操作。数据由Input\_Regfile写入PE Array，读写时钟周期由PE Array控制。

### Input\_Regfile模块主要输入输出接口

表3-11 Input\_Regfile主要输入接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入接口名称 | 输入模块—接口 | 作用 |
| 1 | clk\_cal | 外部—clk\_cal | 系统时钟信号 |
| 2 | rst\_cal\_n | 外部—rst\_acc | 复位信号 |
| 3 | K | Memory\_Controller—K | 当前网络层卷积核大小 |
| 4 | S | Memory\_Controller—S | 当前网络层卷积核步长 |
| 5 | Bm\_cnt\_in | Memory\_Controller—Bm\_cnt | 当前传输Hu的批次数 |
| 6 | IR\_Data\_I\_vld | InOutBuffer—IOB\_Data\_O\_vld | 数据输入使能 |
| 7 | IR\_Data\_I0- IR\_Data\_I7 | InOutBuffer—IOB\_Data\_O0-IOB\_Data\_O7 | 当前传输Hu批次的第1-7个值 |
| 15 | pe\_end | PE\_Array—pe\_end | 当前计算结束信号,控制数据的读写 |
| 16 | Weight\_Data\_Ovld | Memory\_Controller—wt\_C0\_O\_vld | 第一列权重输出有效信号，控制map同时输出 |

表3-12 Input\_Regfile主要输出接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输出接口名称 | 输出模块—接口 | 作用 |
| 1 | IR\_Data\_O0- IR\_Data\_Of | PE\_Array—IMap0- IMap15 | PE\_Array第1-16行输入数据 |
| 2 | IR\_Data\_O\_vld | PE\_Array—ImapVld | 输入数据有效值 |

## PE\_array子模块

### PE\_array模块功能介绍

**作用：**

PE\_array一共含有128个PE单元（16行8列），虽然功能比较单一但是这是核心模块，只进行卷积计算，并输出结果。每一个PE单元主要有8bit的乘法器和16bit的加法器组成，乘累加K次之后输出结果。

PE阵列的工作原理如下：在第一个clk的时候，第一列16个PE同时进行第一次计算，然后第二个CLK的时候，第二列PE进行第一次计算，第一列PE进行第二次计算，如此从右至左地脉动，直到第八个周期，第八列PE进行第一次计算，第一列PE进行第八次计算，整个脉动阵列被激活起来，都在进行着计算工作。

### PE\_array模块主要输入输出接口

表3-13 PE\_array主要输入接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入接口名称 | 来源(模块—接口) | 作用 |
| 1 | clk\_cal | 外部—clk\_cal | 系统时钟信号 |
| 2 | rst\_cal\_n | 外部—rst\_cal\_n | 复位信号 |
| 3 | ImapVld | InputRegfile—IR\_Data\_O\_vld | 输入map有效信号 |
| 4 | IMap0- IMap15 | InputRegfile—IR\_Data\_O0- IR\_Data\_Of | 16行map数据输入 |
| 5 | IweightVld0- IweightVld7 | Weight\_buffer—kernel\_C0\_O- kernel\_C7\_O | 8列weight数据输入 |
| 6 | IweightVld0- IweightVld7 | Weight\_buffer—wt\_Ovld0- wt\_Ovld7 | 输入weight有效信号 |
| 7 | CalCycle | MemoryController—CalCycle | 每次需要计算的周期数 |
| 8 | pass\_cycle | MemoryController—pass\_cycle | 每K次计算所需要的周期数 |
| 9 | rd\_done | MemoryController-rd\_done | 控制pe\_end信号 |
| 10 | or\_cs | SPU-or\_cs | 控制信号在计算状态才能变化 |
| 11 | nn\_layer\_cnt | SPU-nn\_layer\_cnt | 控制7、8层的计算结果输出 |

表3-14 PE\_array主要输出接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输出接口名称 | 输出模块—接口 | 作用 |
| 1 | PE\_Omap0-PE\_Omap7 | OutputRegfile—ORegfile\_Idata0-7 | 输出计算结果 |
| 2 | PE\_OmapVld1- PE\_OmapVld7 | OutputRegfile—ORegfile\_IData\_vld0-7 | 输出有效信号 |
| 3 | PE\_end | In-Out\_buffer—PE\_end  MemoryController—PE\_end | PE计算完毕信号 |
| 4 | FC\_OMap1- FC\_Omap7 | In-Out\_buffer—IOB\_FC\_I0- IOB\_FC\_I7 | FC层的输出有效数据 |
| 5 | FC\_OVld | In-Out\_buffer—IOB\_FC\_vld | FC层的输出有效使能 |

## Output\_Regfile子模块

### Output\_Regfile模块功能介绍

**作用：**为了减少中间层的存储量，在输出计算结果到buffer之前先进行relu和pooling，这里需要考虑到输出数据的复用，因此需要有一个模块对卷积计算结果进行整理，然后有序地输出数据进行池化，这个模块就是Output\_Regfile，在PE阵列和relu\_pooling模块之间。

**内存大小：**Output Regfile用于实现输出数据复用的功能。PE阵列每一列通道一次输出16个计算结果，有8列通道，因此每一列通道对应有一个Output Regfile。由于当前输出的计算结果与上一次输出的计算结果还有一定池化联系，所以将一个Output Regfile的大小定为32 byte，可以存两次卷积计算的结果。

**数据写入：**Output Regfile的大小定为32 byte，一次卷积计算的结果为16byte，所以低地址16byte写入奇数次的计算结果，高地址16byte写入偶数次的计算结果，轮流写入即可。

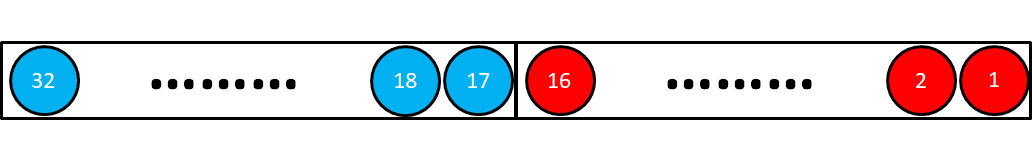


图3-8 Output Regfile输入原理图

**数据读出：**因为每一次需要池化的数量为P个，所以Output Regfile输出P个计算结果，这里有三种地址跳变的形式，这里举一种最简单的例子p=2,s=1。

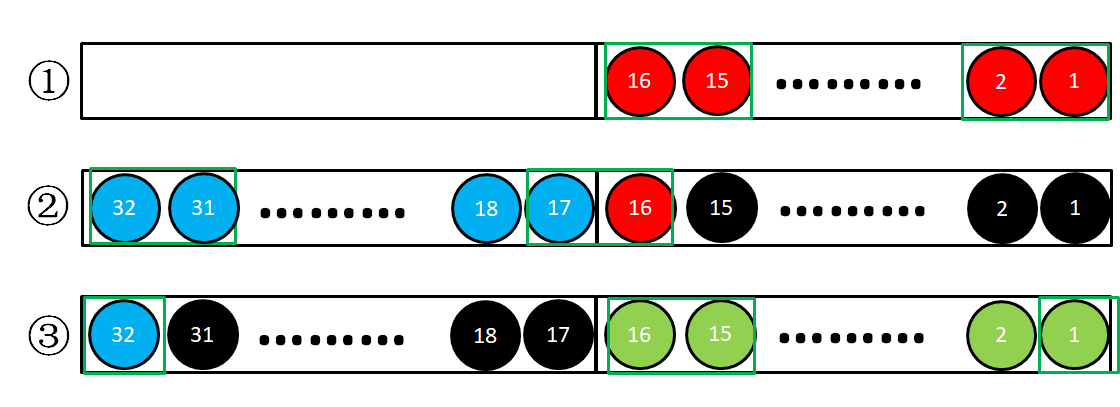


图3-9 Output Regfile输出原理图

第一种情况：红色为第一次写入的计算结果，然后按P=2的大小，间隔为s=1,读出即可；

第二种情况：前面第一次写入的数据已经没用了，所以标黑色，蓝色为第二次写入的结果，也是按P=2的大小，间隔为s=1,读出即可；

第三种情况：前面第二次写入的数据已经没用了，标黑，绿色为第三次写入的结果，蓝色与绿色转换之间需要有地址的跳变，之后也是按顺序读出，如同第一种情况，如此循环。

### Output\_Regfile模块主要输入输出接口

表3-15 Output\_Regfile主要输入接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入接口名称 | 输入模块—接口 | 作用 |
| 1 | clk\_cal | 外部—clk\_cal | 系统时钟信号 |
| 2 | rst\_cal\_n | 外部—rst\_cal\_n | 复位信号 |
| 3 | ORegfile\_IData\_vld0- ORegfile\_IData\_vld7 | PEArray—PE\_OmapVld1- PE\_OmapVld7 | 从PEArray中载入data的使能 |
| 4 | ORegfile\_Idata0- ORegfile\_Idata8 | PEArray—PE\_Omap0-PE\_Omap7 | 从PEArray中载入data |
| 5 | P | MemoryController—Pool\_K | 池化大小 |
| 6 | S | MemoryController—Pool\_S | 池化步长 |
| 7 | or\_cs | SPU-or\_cs | 控制有些信号在下一层计算之前清零 |
| 8 | ft\_N\_cnt | MemoryController—ft\_N\_cnt | 每列PE后面的某通道Output\_Regfile的MUX控制信号 |

表3-16 Output\_Regfile主要输出接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输出接口名称 | 输出模块—接口 | 作用 |
| 1 | ORegfile\_Odata0- ORegfile\_Odata7 | Pool—Pool\_Idata0-7 | 输出到Pooling的数据 |
| 2 | ORegfile\_OData\_vld0- ORegfile\_OData\_vld7 | Pool—Pool\_IData\_vld0-7 | 输出到Pool的使能 |

## Relu子模块

### Relu模块功能介绍

Relu模块的功能为根据网络的要求作为激活函数。有两种激活的模式[0:2]和[0:1]，即大于上限取上限，小于下线取下限，在区域中取本身，前两层用[0:2]模式，后四层用[0:1]模式。

### Relu模块主要输入输出接口

表3-20 Relu模块主要输入接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入接口名称 | 来源(模块—接口) | 作用 |
| 1 | clk\_cal | 外部—clk\_cal | 系统时钟信号 |
| 2 | nn\_layer\_cnt | SPU—nn\_layer\_cnt | 当前处理网络层索引 |
| 3 | Relu\_Din1-8 | PE\_array—PE\_OMap1-8 | Relu的输入数据 |
| 8 | Relu\_Din\_vld0- Relu\_Din\_vld7 | PE\_array—PE\_OMapVld1-8 | Relu写入的有效信号 |

表3-21 Relu&Pooling主要输出接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输出接口名称 | 输出模块—接口 | 作用 |
| 1 | Relu\_Dout\_vld1-8 | Output\_Regfile—ORegfile\_IData\_vld1-8 | Relu输出有效 |
| 2 | Relu\_Dout1-8 | Output\_Regfile—ORegfile\_IData1-8 | Relu输出数据 |

## Pooling子模块

### Pooling模块功能介绍

Pooling模块的功能主要为使用最大池化的方法，在P个卷积结果中取最大的结果。

### Pooling模块主要输入输出接口

表3-22 Pooling模块主要输入接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输入接口名称 | 来源(模块—接口) | 作用 |
| 1 | clk\_cal | 外部—clk\_cal | 系统时钟信号 |
| 2 | rst\_cal\_n | 外部—rst\_cal\_n | 复位信号 |
| 3 | Pool\_IData1-8 | Output\_Regfile—PE\_OMap1-8 | Pooling的输入数据 |
| 8 | Pool\_IData\_vld1-8 | Output\_Regfile—PE\_OMap\_vld1-8 | Pooling写入的有效信号 |

表3-23 Pooling模块主要输出接口

|  |  |  |  |
| --- | --- | --- | --- |
|  | 输出接口名称 | 输出模块—接口 | 作用 |
| 1 | Pool\_OData1-8 | InOutBuffer -- IOB\_Data\_I0-7 | Pooling的输出数据 |
| 2 | Pool\_OData\_vld1-8 | InOutBuffer -- IOB\_Data\_I\_vld0-7 | Pooling输出的有效信号 |

## 参考文献

[1] https://www.texasheart.org/heart-health/heart-information-center/topics/arrhythmia/

[2] Organization W H . WHO | Noncommunicable diseases country profiles 2018[J]. Scand J Soc Med, 2018, 14(1):7-14.

[3] https://www.heart.org/en/health-topics/arrhythmia/symptoms-diagnosis--monitoring-of-arrhythmia

[4] Al-Khatib SM et al 2018 2017 aha/acc/hrs guideline for management of patients with ventricular arrhythmias and the prevention of sudden cardiac death: a report of the American College of Cardiology/American Heart Association task force on clinical practice guidelines and the heart rhythm society J. Am. Coll. Cardiol. 72 e91–220

[5] S. Hong, Y. Zhou, M. Wu, J. Shang, Q. Wang, H. Li, and J. Xie, "Combining deep neural networks and engineered features for cardiac arrhythmia detection from ECG recordings," Physiological Measurement, vol. 40, p. 054009, 2019-06-04 2019.

[6] P. de Chazal, R. R. B. and O. M., "Automatic classification of heartbeats using ECG morphology and heartbeat interval features," IEEE Transactions on Biomedical Engineering, vol. 51, pp. 1196-1206, 2004.

[7] K. S. Park, B. H. Cho, D. H. Lee, S. H. Song, J. S. Lee, Y. J. Chee, I. Y. Kim, and S. I. Kim, "Hierarchical support vector machine based heartbeat classification using higher order statistics and hermite basis function," in 2008 Computers in Cardiology,2008 Computers in Cardiology, 2008, pp. 229-232.

[8] M. Llamedo and J. P. Martinez, "Heartbeat Classification Using Feature Selection Driven by Database Generalization Criteria," IEEE Transactions on Biomedical Engineering, vol. 58, pp. 616-625, 2011.

[9] P. Pławiak, "Novel methodology of cardiac health recognition based on ECG signals and evolutionary-neural system," Expert Systems with Applications, vol. 92, pp. 334-349, 2018.

[10] S. Parvaneh, J. Rubin, S. Babaeizadeh, and M. Xu-Wilson, "Cardiac arrhythmia detection using deep learning: A review," Journal of Electrocardiology, vol. 57, pp. S70-S74, 2019.

[11] S. Kiranyaz, T. Ince and M. Gabbouj, "Real-Time Patient-Specific ECG Classification by 1-D Convolutional Neural Networks," IEEE Transactions on Biomedical Engineering, vol. 63, pp. 664-675, 2016.

[12] S. S. Xu, M. Mak and C. Cheung, "Towards End-to-End ECG Classification With Raw Signal Extraction and Deep Neural Networks," IEEE Journal of Biomedical and Health Informatics, vol. 23, pp. 1574-1584, 2019.

[13] J. Pan and W. J. Tompkins, "A Real-Time QRS Detection Algorithm," IEEE Transactions on Biomedical Engineering, vol. BME-32; BME-32, pp. 230-236, 1985-01-01; 1985-01-01 1985.

[14] N. Wang, J. Zhou, G. Dai, J. Huang, and Y. Xie, "Energy-Efficient Intelligent ECG Monitoring for Wearable Devices," IEEE Transactions on Biomedical Circuits and Systems, vol. 13, pp. 1112-1121, 2019.

[15] Ö. Yıldırım, P. Pławiak, R. Tan, and U. R. Acharya, "Arrhythmia detection using deep convolutional neural network with long duration ECG signals," Computers in Biology and Medicine, vol. 102, pp. 411-420, 2018.