第 6 章　深度学习加速器设计

随着深度学习应用场景越来越复杂，深度学习算法的运算形式日益多样化、网络结构愈加复杂，例如 Alexnet、GoogleNet、ResNet 等。为了更高效、更灵活地支持深度学习算法，处理器设计师需要针对深度学习算法的特性进行优化加速以满足编程者越来越多样化的需求。深度学习加速器就是一类针对高效支持深度学习算法的处理器/专用芯片，其针对深度学习的通用计算进行加速，例如卷积运算、池化运算等；同时，深度学习加速器考虑深度学习算法算子多样性，提供灵活的指令集，便于程序员高效地实现算法。

在深度学习算法中，卷积运算是最核心的运算操作，卷积层包含大量的输入输出数据和权值参数，其运算量占深度学习算法总运算量的 90% 以上。处理器执行卷积运算的性能也决定了深度学习算法在处理器上的性能。在智能处理器系统中，设计出更高效地支持卷积运算的运算器，是深度学习加速器设计的关键技术之一。

6.1　实验目的

在本章节中，将根据深度学习加速器卷积运算原理，使用 Verilog HDL 编写程序设计出深度学习处理器中矩阵运算子单元——内积运算器，然后在 Mentor 公司的 Modelsim 环境下用 Verilog 进行仿真。这个内积运算器是一个专门为教学目的简化的设计。在此设计过程中，我们不但关注组成内积运算器设计的每个模块是可仿真并且可以综合成门级网表，而且关心内积运算器的性能。因此，这个设计是一个能真正通过具体物理电路实现的内积运算器。

本章节的设计仅仅是一个教学模型，设计从物理实现上来看不一定很合理，主要是为了从原理上说明深度学习处理器中矩阵运算是如何加速卷积计算。本章的内容希望达到以下目的：

（1）学习深度学习处理器矩阵运算器加速卷积层原理。

（2）基于深度学习处理器中卷积运算器的功能介绍完成内积运算器设计。

（3）通过 EDA 工具和提供的验证环境完成内积运算器的验证。

本实验预计用时：4 小时。

6.2　实验原理

本节将介绍深度学习处理器卷积运算原理，首先我们根据卷积层算法拆解卷积层计算过程，然后再结合深度学习处理器结构介绍执行卷积计算的运算单元以及卷积层在处理器上的运算过程。

卷积层由输入层、输出层和若干个卷积核（filter）组成。输入层和输出层包括若干个特征图像。每个卷积核包含 $K_x \times K_y$ 组权值，其与输入层 $K_x \times K_y$ 区域内神经元对位相乘后

累加乘法结果即可得到一个输出神经元。卷积核沿着输入特征图像的 X、Y 方向以 S_x、S_Y 的步长滑动遍历则得到一个输出特征图像，然后遍历所有卷积核得到输出层所有特征图像。

根据卷积算法，卷积计算可先采用输入层 $K_x \times K_y$ 区域内神经元和所有的卷积核计算得到不同输出特征图像上相同位置的输出神经元，然后 $K_x \times K_y$ 区域框沿着输入特征图像的 X、Y 方向滑动遍历得到输出特征图像的所有神经元。第 fo 个特征图像上 (w,h) 位置的输出神经元计算公式为：

$$Out_{w,h}^{fo} = f((\sum_{i=0}^{K_x-1}\sum_{j=0}^{K_y-1}\sum_{fi=0}^{F_i-1} W_{fi,i,j}^{fo} * In_{wS_x+i,hS_y+j}^{fi}) + \beta^{fo}) \tag{6.1}$$

其中，W、In、Out、$beta$ 分别表示权值、输入神经元、输出神经元、偏置，f 表示激活函数。

公式6.1可等效为：

$$Out_{w,h;fo} = f(\sum_{i=0}^{fi*K_x*K_y-1} \overline{W}_{fo,i} * \overline{In}_{i;w,h} + \beta^{fo}) \tag{6.2}$$

其中，$\overline{In}_{;w,h}$ 是一个包含 $f_i * K_x * K_y$ 个分量的向量，由输入层中对应计算 (w,h) 位置的输出结果的 $K_x \times K_y$ 区域内神经元组成；$\overline{W}$ 为权值矩阵，其规模为 $fo \times (f_i * K_x * K_y)$，由卷积层 $fo \times K_y \times K_x \times fi$ 规模的 4 维权值张量的低三个维度合并成一个维度转换而来。

由公式6.2可知，卷积计算可以看成先进行 $W * H$ 次 $f_i * K_x * K_y$ 分量的输入神经元向量和 $(f_i * K_x * K_y) \times fo$ 规模的权值矩阵相乘，然后进行向量加法和向量激活。

另一方面，当规模为 m 的向量 A 和规模为 $m \times n$ 的矩阵 B 相乘时，得到的乘积向量 C 的第 i 个元素可表示为：

$$c_i = \sum_{j=0}^{m-1} a_j * b_{j,i} \tag{6.3}$$

公式6.3可表示为，

$$c_i = A \cdot B_i \tag{6.4}$$

即乘积结果向量 C 的第 i 个元素为向量 A 和矩阵 B 的第 i 个列向量 B_i 的内积。

结合式6.2和式6.4可知，卷积运算有一系列向量内积运算、向量加法和向量激活组成，尤其以向量内积运算为主。

在图6.1所示由控制模块、运算模块和存储模块三大部分组成的智能计算系统中深度学习处理器架构 (Deep Learning Processor, DLP) 中，输入/输出神经元、权值分别存储于神经元存储单元（Neuron RAM, NRAM）和权值存储单元（Weight RAM, WRAM）；矩阵乘运算由矩阵运算单元（Matrix Function Unit, MFU）完成；其他向量运算由向量运算单元（Vector Function Unit, VFU）完成。控制模块则负责协调控制运算模块和存储模块完成深度学习任务。

MFU 通过图6.2所示的 H 树的互联方式将 M 个矩阵运算子单元（Processing Element，PE）组织为一个完整的矩阵运算单元，不同 PE 位于 H 树的叶节点。H 树将预处理后的输

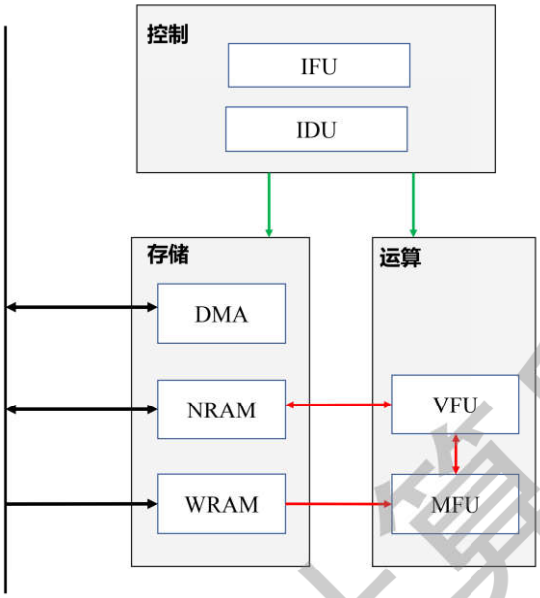

图 6.1　深度学习处理器结构图

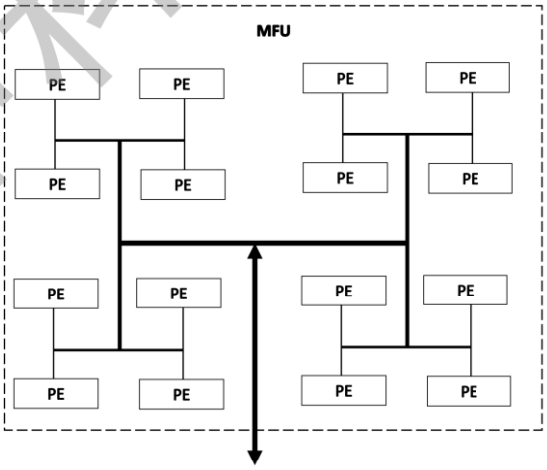

图 6.2　矩阵运算单元结构图

入神经元和控制信号广播到所有 PE，并收集不同 PE 计算的输出结果返回给 VFU。PE 单元负责进行向量的内积运算，主要由 N 个乘法器和一个 N 输入加法树组成。

类似 MFU，WRAM 也采用 H 树互联的方式将 M 个分布式的片上存储单元（Distributed Weight RAM，DWRAM）组织在一起。在深度学习处理器中，每个 DWRAM 对应一个 PE。矩阵运算时，每个 DWRAM 根据控制信号读取权值信号给 PE 进行计算。

深度学习处理器进行规模为 m 的向量 A 和规模为 $m \times n$ 的矩阵 B 相乘时，将矩阵 B 均分为 M 个子矩阵 $\overline{B}_0$、$\overline{B}_1$、……、$\overline{B}_{M-1}$。每个子矩阵规模为 $m \times \frac{n}{M}$，分别存储在 M 个 DWRAM 中。MFU 进行计算时，MFU 的 M 个 PE 利用接收 H 树广播的向量 A 的 N 个分量，并分别从 M 个 DWRAM 读取各自子矩阵的第 i 个列向量的 N 个分量进行内积运算，得到输出向量 C 的 M 个分量的部分和。处理器控制 NRAM 将向量 A 的所有分量都发送给 MFU 则可完成输出向量 C 的 M 个分量的计算。然后，处理器将前面步骤重复 $\frac{n}{M}$ 次，便可完成输出向量 C 所有分量的计算。

假设矩阵运算单元（MFU）包含 4 个矩阵运算子单元（PE），输入的神经元矩阵 A 规模为 2×8、权值矩阵 B 规模为 8×4，计算的输出神经元 C 规模是 2×4。矩阵运算单元进行 $C = A \times B$ 运算时，每个矩阵运算子单元计算 C 的列子矩阵。如图6.3所示，矩阵运算单元需 4 cycle 完成矩阵运算：

- 第 1 拍，第一行的前 4 个神经元广播给所有 PE，每个 PE 接收对应列权值的前 4 个元素，分别进行内积运算，得到第一行 4 个输出神经元结果的部分和。
- 第 2 拍，第一行的后 4 个神经元广播给所有 PE，每个 PE 接收对应列权值的后 4 个元素，分别进行内积运算，然后累加第 1 拍计算的部分和结果，得到第一行 4 个输出神经元。
- 第 3 拍，第二行的前 4 个神经元广播给所有 PE，每个 PE 接收对应列权值的前 4 个元素，分别进行内积运算，得到第二行 4 个输出神经元结果的部分和。
- 第 4 拍，第二行的后 4 个神经元广播给所有 PE，每个 PE 接收对应列权值的后 4 个元素，分别进行内积运算，然后累加第 3 拍计算的部分和结果，得到第二行 4 个输出神经元。

卷积计算时，DLP 将卷积层的所有输出特征图像（Output Feature Map，OFM）以 M 个特征图像为一组。如图6.4所示，DLP 每次采用输入神经元中 $Kx \times Ky \times Fi$ 数据子块计算一组输出特征图像中不同输出特征图像上相同位置的特征点。MFU 的 M 个矩阵运算子单元分别计算不同输出特征图像的输出神经元。然后，DLP 按照 XY 循环顺序计算特征图像上的特征点，计算一组完整的输出特征图像。

DLP 计算一组输出特征图像中不同输出特征图像上相同位置的特征点过程类似于矩阵运算。运算过程可拆分为 4 个步骤，step 1，VFU 依次将 $Kx \times Ky \times Fi$ 数据子块从 NRAM 读出，进行数据预处理后发送给 MFU。step 2，MFU 将接收的输入神经元广播给 M 个 PE 单元。step 3，PE 接收 H 树广播的输入神经元和 WRAM 读取权值进行内积运算，并将内积结果保存至部分和缓存寄存器或将内积结果累加部分和缓存器的数值。step 4，PE 收到 $Kx \times Ky \times Fi$ 数据子块的所有数据后将计算的部分和进行数据格式转换后输出。

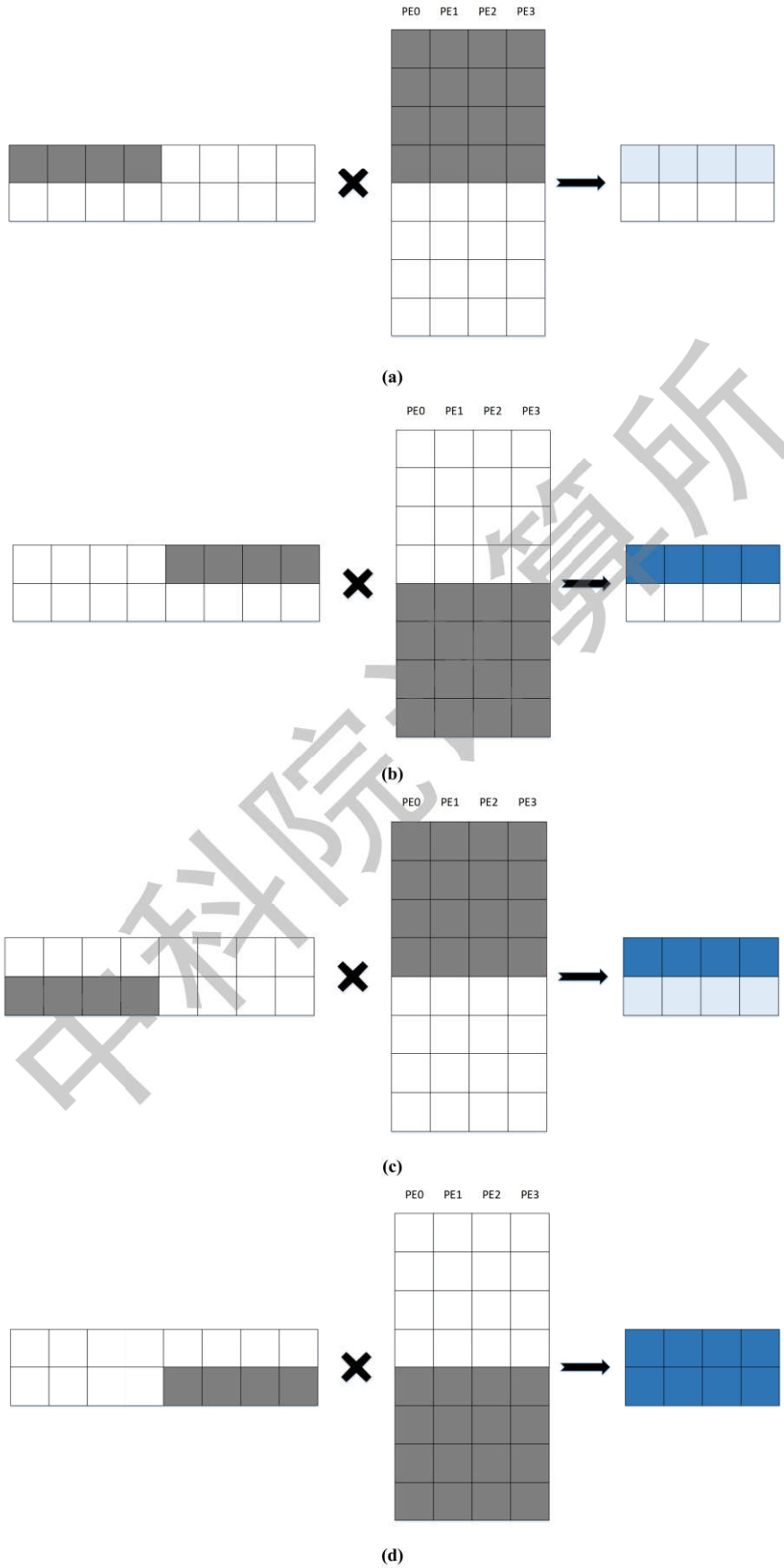

图 6.3　4PE 矩阵运算单元矩阵乘步骤示意图

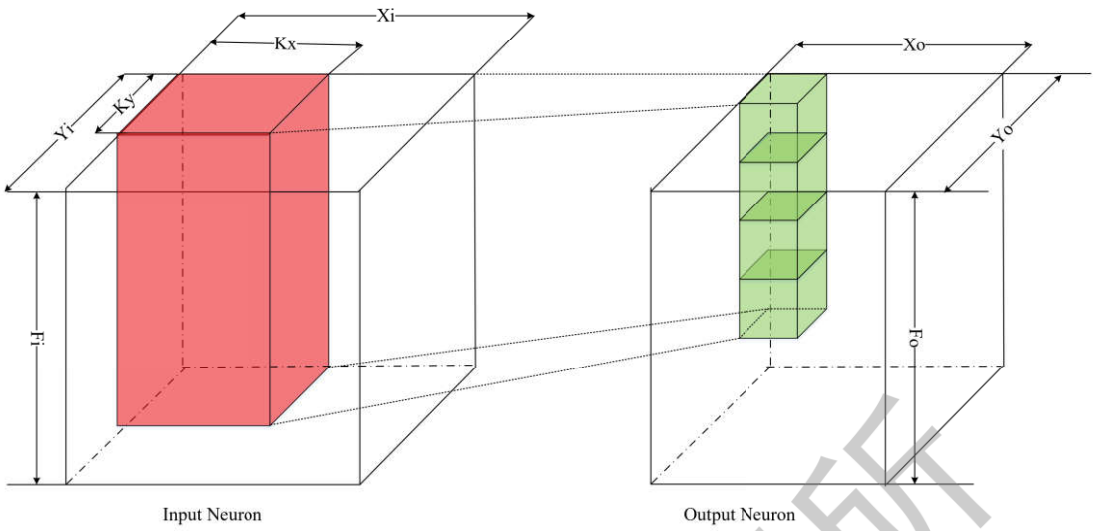

图 6.4　卷积层计算示意图

通过矩阵运算过程和卷积运算过程可知，矩阵运算子单元的内积运算是矩阵运算单元的核心。下面将详细介绍内积运算单元设计和仿真过程。

6.3　实验内容

根据第6.2节可知，矩阵运算子单元的内积运算器是深度学习处理器的核心部件。本节将通过两个实验介绍矩阵运算子单元的不同实现方法。然后通过对比两个实验效果阐述深度学习处理器加速深度学习算法内积运算的原理。

6.3.1　实验目标

矩阵运算子单元的内积运算器的功能是将长度可变的神经元向量（neuron）和权值向量（weight）进行内积运算，然后将结果输出。其功能伪代码如图6.5所示。

```
psum = 0;
for(i = 0; i < element_num; i++){
    psum = neuron[i] * weight[i];
}
output = psum;
```

图 6.5　内积运算器功能代码

在通用 CPU 中，深度学习神经元数据和权值数据一般采用单精度浮点数据表示，相应的运算单元也采用的浮点运算器。然而在深度学习处理器中，为了节省功耗、面积开销，一般采用低精度运算器代替浮点运算器。根据?? 所述，INT16 或者 INT8 已经能够满足深度学习算法的应用需求。为了简化实验内容，本章实现的内积运算所有神经元/权值数据都采

用 INT16 表示。

6.3.2 实验一：串行内积运算器

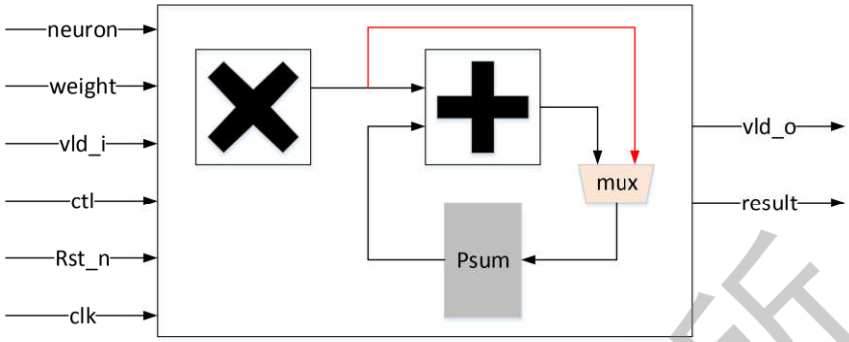

图 6.6 串行内积运算器

如图6.6所示，串行内积运算器主要包括：一个乘法器、一个加法器、一个部分和寄存器和一个数据选择器。其输入输出接口信号描述如表6.1所示。

表 6.1 串行内积运算器信号描述

域	位宽	功能描述
neuron	16	输入 INT16 神经元分量
weight	16	输入 INT16 权值分量
vld_i	1	输入数据和控制信号有效标志，高电平有效
ctl	2	输入控制信号 ctl[0]: 输入神经元/权值数据是一组神经元/权值向量的第一个元素 ctl[1]: 输入神经元/权值数据是一组神经元/权值向量的最后一个元素
rst_n	1	输入复位信号，低电平有效
clk	1	输入时钟信号
result	32	输出内积结果
vld_o	1	输出内积结果有效标志，高电平有效

模块采用异步复位方式，复位信号 rst_n 低电平有效。复位时，部分和寄存器被清零。输入 vld_i 表示输入表示神经元/权值数据和控制信号有效，当 vld_i 为高电平时，串行内积运算器接收输入的 INT16 神经元和 INT16 权值分量进行乘法运算，然后再用乘法结果更新部分和寄存器。

乘法单元输入的神经元向量和权值向量中每个每个神经元和权值数据都是有符号数据，乘法后得到的部分和也是有符号数据，部分和数据位宽为对应神经元位宽与权值位宽之和。Verilog 语法中乘法运算符默认进行无符号乘法运算，有符号数据运算需进行显示说明，如图6.7所示。

控制信号为 2bit 信号，最低位表示输入神经元/权值数据是一组神经元/权值向量的第一个元素。当 ctl[0] 有效时，乘法结果直接写入部分和寄存器；否则，乘法结果先累加部分和寄存器，然后将累加结果写入部分和寄存器。图6.6中数据选择器即用于选择写入部分和寄存器的源数据。

```
wire signed [15:0] a, b;
wire signed [31:0] c = a * b;
```

图 6.7　有符号乘法代码示例

控制信号最高位表示输入神经元/权值数据是一组神经元/权值向量的最后一个元素。当ctl[1]有效时，串行内积运算器在下一个时钟周期将部分和寄存器输出到result端口。串行内积运算器输出result值，vld_o置起1拍，表示输出内积结果有效。

6.3.3　实验二：并行内积运算器

不同于串行内积运算器每拍最多处理一个神经元和一个权值分量进行乘法运算，并行内积运算器每拍能处理多个神经元/权值分量。如图6.8所示，并行内积运算器一组（32个）乘法器、一个累加单元、一个加法器、一个部分和寄存器和一个数据选择器。

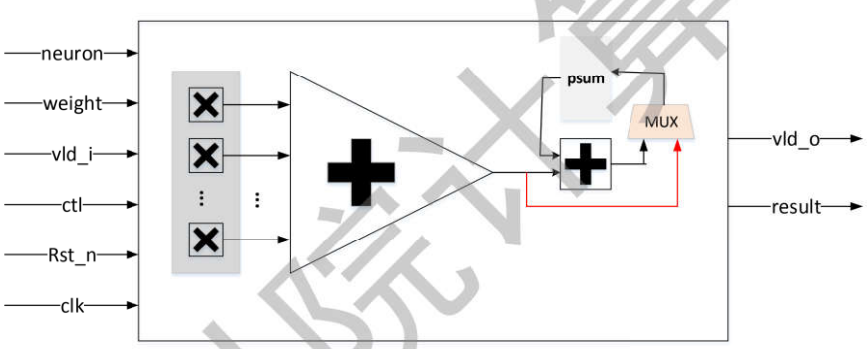

图 6.8　并行内积运算器

如表6.2所示，并行内积运算器接口信号类似于串行内积运算器接口信号。不同之处在于，并行内积运算器输入的神经元数据和权值数据为为32个INT16分量的向量，而串行内积运算器输入的神经元数据和权值数据是1个INT16分量。

表 6.2　并行内积运算器信号描述

域	位宽	功能描述
neuron	512	输入神经元数据，包括32个INT16分量
weight	512	输入权值数据，包括32个INT16分量
vld_i	1	输入数据和控制信号有效标志，高电平有效
ctl	2	输入控制信号 ctl[0]: 输入神经元/权值数据是一组神经元/权值向量的第一个元素 ctl[1]: 输入神经元/权值数据是一组神经元/权值向量的最后一个元素
rst_n	1	输入复位信号，低电平有效
clk	1	输入时钟信号
result	32	输出内积结果
vld_o	1	输出内积结果有效标志，高电平有效

并行内积运算器每次进行包含32个INT16分量的神经元向量和权值向量内积，然后

通过将连续 32 个 INT16 分量的内积结果累加来支持更长神经元向量和权值向量的内积运算。当 vld_i 为高电平时，并行内积运算器接收输入的 INT16 神经元和 INT16 权值向量，将不同的神经元分量和权值分量输入对应的乘法器进行乘法运算。然后，将 32 个乘法器的结果输出给累加单元。

累加单元将 32 个输入部分和累加成一个部分和。当 ctl[0] 有效时，累加单元结果直接写入部分和寄存器；否则，累加单元结果先通过加法器累加部分和寄存器，然后将累加结果写入部分和寄存器。图6.8中数据选择器即用于选择写入部分和寄存器的源数据。

控制信号最高位表示输入神经元/权值数据是一组神经元/权值向量的最后一个子向量。当 ctl[1] 有效时，并行内积运算器在下一个时钟周期将部分和寄存器输出到 result 端口。并行内积运算器输出 result 值，vld_o 置起 1 拍，表示输出内积结果有效。

6.3.4 实验三：矩阵运算子单元

矩阵运算子单元根据控制信号进行接收 H 树广播的神经元向量和 WRAM 读取的权值向量数据进行内积运算。如图6.9所示，矩阵运算子单元在并行内积运算器的基础上增加了控制单元（CTL）。同时，由于矩阵运算子单元和 H 树总线、WRAM 相连接，对应神经元、权值、输出结果的接口信号也变成带有反压 ready 握手的总线信号。

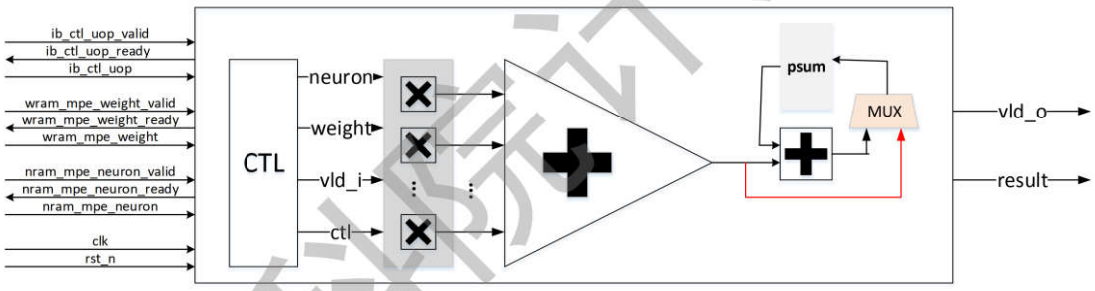

图 6.9 矩阵运算子单元

控制单元接收输入的控制信号（uop），然后根据控制信号进一步译码生成输出给乘法单元的控制信号。矩阵运算子单元接收的每条控制信号对应计算一个输出神经元。每个输出神经元采用 k 次循环控制输入神经元和权值向量的乘累加。即一条控制信号计算的输出神经元需进行 k 次乘累加。相应地，控制单元也将生成出 k 条给乘法单元的控制信号。

输入控制单元的控制信号（uop）为 10 比特位宽的信号，表示对应计算的输出神经元输入内积神经元和权值的长度（单位：64 Byte）。控制单元根据控制信号译码时，生成给乘法单元的第一条 ctl 信号中 ctl[0] 为 1，表示对应的神经元和权重内积结果直接保存至部分和寄存器，不需累加部分和寄存器；最后一条 ctl 信号中 ctl[1] 为 1，表示最后一组部分和计算完成，可以将部分和结果输出。当一条输入控制信号译码生成的所有 ctl 信号输出后，控制单元可切换下一条控制信号进行译码，输出 ib_ctl_uop_ready 信号变为高电平。

同时，控制单元接收输入的权值数据和神经元数据，与对应给乘法单元的控制信号同步后一起输出。仅当输入的神经元数据、权值数据和控制信号都有效时，控制单元才会将这三组数据发送给乘法单元，并完成输入神经元/权值数据的握手。

6.4 实验环境

本节将描述实验环境，包括工具安装和验证环境搭建。

6.4.1 工具安装

本章节所有程序都在 Mentor 公司的 Modelsim 10.4a（学生版）上运行。软件安装包可从 Mentor 官网下载页面（https://www.mentor.com/company/higher_ed/modelsim-student-edition）下载学生版软件。下载安装包后，按照如下步骤安装程序：

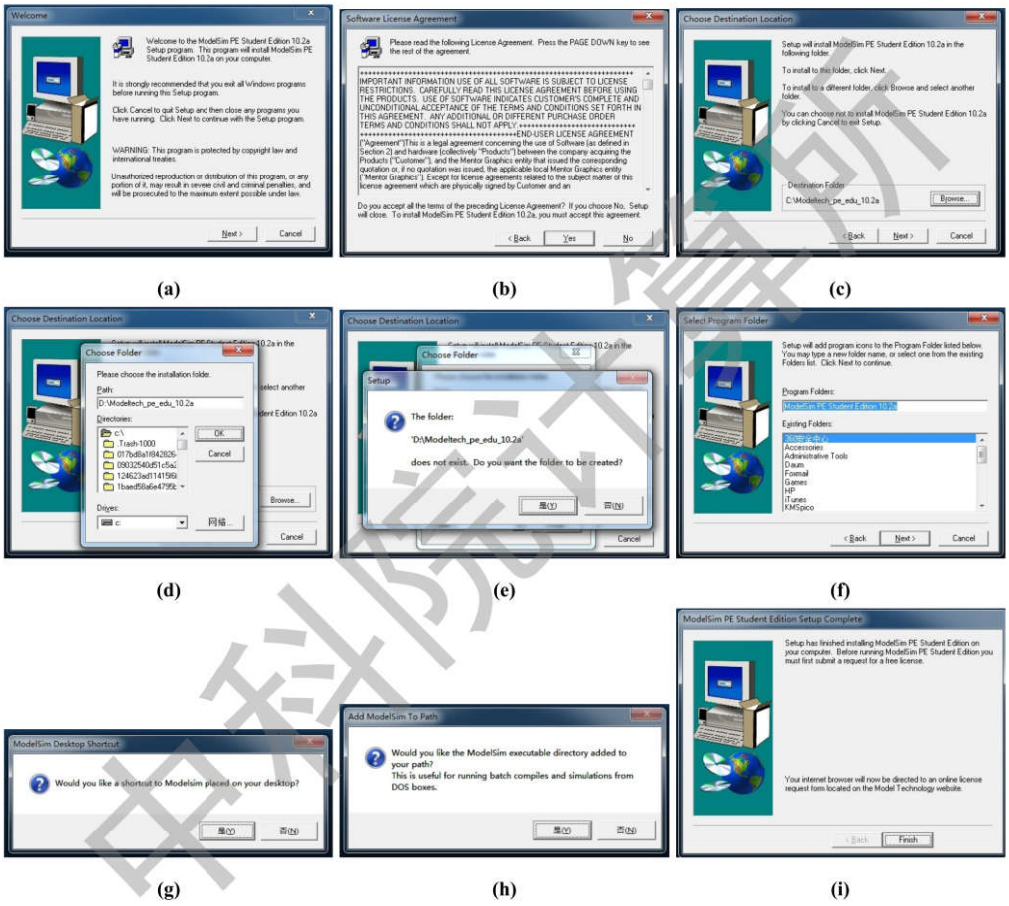

图 6.10 Modelsim 安装步骤

（1）运行安装包，进入如图6.10a所示初始界面，确认是否继续安装程序

（2）点击“Next”确认继续安装，进入如图6.10b所示 license 说明界面。

（3）点击“Yes”同意 license 说明内容，进入如图6.10c所示选择软件安装路径界面。

（4）点击“Browse”选择自定义安装路径，进入如图6.10d界面。

（5）在“Path”栏填写软件安装路径，并点击“OK”，进入图6.10e界面。

（6）第一次安装时，软件将在指定路径下创建 Modelsim 文件夹，点击“是”确认创建文件夹，进入图6.10f界面。

（7）点击“Next”确认继续安装，进入如图6.10g界面。

（8）点击“是”确认在桌面创建软件快捷方式，进入如图6.10h界面。

（9）点击“是”确认将软件安装路径加入系统环境变量中，便于在 DOS boxes 下运行批量处理脚本。安装程序进入如图6.10i界面。

（10）点击“Finish”后弹出申请 license 网页，按照提示设置 license 后，软件即安装完毕。

6.4.2 目录和代码文件

实验环境文件夹组织如下：

- 目录 src，包含编写的矩阵运算子单元 verilog 源代码。
- 目录 sim，包含仿真脚本文件和顶层文件。
  - 文件 tb_top.v，测试顶层文件，生成激励信号，例化矩阵运算子单元模块。
  - 文件 build.do，编译脚本文件，如图6.11所示。
  - 文件 compile.f，编译文件列表，如图6.12所示。
  - 文件 sim_run.do，仿真执行文件，如图6.13所示。关键参数含义为
    - +nowarnTSCALE 表示忽略没有 timescale 定义的文件，用前面的 timescale 替代。
    - -lib work 表示被仿真的库（lib）为 work。
    - -c 表示从命令行启动仿真。
    - -novopt 表示仿真时不要优化中间变量，保持最大的信号可观测性。
    - tb_top 表示仿真顶层模块名为 tb_top。
- 目录 data，包含仿真输入输出数据文件。
  - 向量规模描述文件 scale。
  - 输入神经元文件 neuron。
  - 输入权值文件 weight。
  - 输出结果文件 result。

tb_top.v 将读取控制信号文件数据输入矩阵运算子单元控制信号端口，并读取神经元数据和权值数据文件输入矩阵运算子单元的神经元端口和权值端口。Verilog 语法中，系统任务 $readmemb 和 $readmemh 用来从文件中读取数据到存储器中。对于 $readmemb 系统任务，每个数字必须使二进制数字，对于 $readmemh 系统任务，每个数字必须使十六进制数字。这两个系统函数可以在仿真的任何时刻被执行使用，使用格式共六种：

- $readmemb(“数据文件名”，存储器名)；

```
# 设置仿真环境路径
# TODO
set sim_home Path/to/simulation/Dir

# 在当前目录下创建一个叫做work的目录，在里面存放仿真数据文件
vlib ${sim_home}/work

# 将work目录下的数据文件映射为一个叫做work的仿真库
vmap work ${sim_home}/work

# 编译compile.f文件中指定的代码
vlog -f ${sim_home}/compile.f
```

图 6.11　编译脚本

```
// 源代码文件
path/to/src_dir/module_0.v
path/to/src_dir/module_1.v

// 顶层测试文件
path/to/sim_dir/tb_top.v
```

图 6.12　编译列表

```
vsim +nowarnTSCALE -lib work -c -novopt tb_top
```

图 6.13　执行脚本

- $readmemb(“数据文件名”，存储器名，起始地址);
- $readmemb(“数据文件名”，存储器名，起始地址，结束地址);
- $readmemh(“数据文件名”，存储器名);
- $readmemh(“数据文件名”，存储器名，起始地址);
- $readmemh(“数据文件名”，存储器名，起始地址，结束地址);

其中，“数据文件名”指示被读取的数据文件，包含输入文件的路径和文件名，如图6.14所示。

```
30 initial
31 begin
32   $readmemb("D:/pe_exp/data/inst", inst);
33   $readmemh("D:/pe_exp/data/neuron", neuron);
34   $readmemh("D:/pe_exp/data/weight", weight);
35 end
```

图 6.14　设置仿真数据路径

6.5　实验步骤

6.5.1　代码编写

根据6.3节介绍完成矩阵运算子单元代码。

6.5.2　编译代码

打开 ModelSim 软件，在命令行窗口输入“do path to build/build.do”命令，工具开始编译测试顶层文件和源代码。ModelSim 命令行窗口中将显示编译的文件、顶层文件已经编译错误和警告数量，如图6.15所示。

```
# ├- Compiling module tb_top
# -- Compiling module fifo
# -- Compiling module mcpu_mult
# -- Compiling module mcpu_acc
# -- Compiling module int45_to_fp_stg1
# -- Compiling module int45_to_fp_stg2
# -- Compiling module int45_to_int16
# -- Compiling module mcpu_cvt
# -- Compiling module mcpu
#
# Top level modules:
#       tb_top
# End time: 20:13:52 on Feb 14,2020, Elapsed time: 0:00:00
# Errors: 0, Warnings: 0
```

图 6.15　编译日志

6.5.3 启动仿真

在 ModelSim 命令行窗口输入“do path to build/sim_run.do”命令，启动仿真。软件将显示如图6.16所示 sim instance 面板，以及仿真模块的层次关系和模块的信号名称。

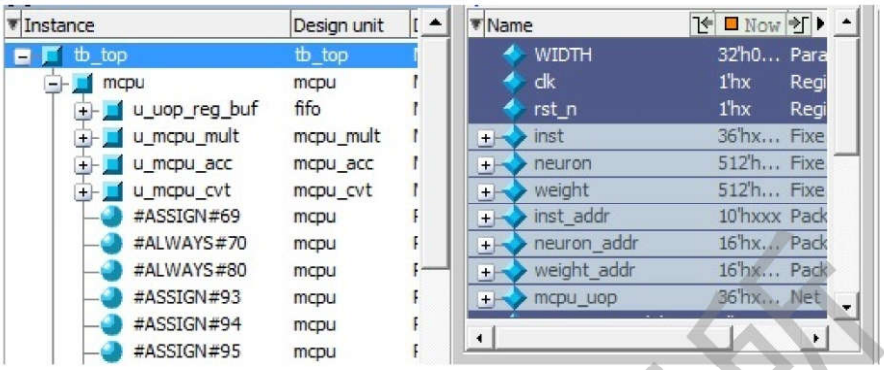

图 6.16　sim instance 面板

6.5.4 添加观测信号

在 sim instance 面板，先选择要调试的模块，然后在 Object 面板中，选择要观察的信号，单击右键“Add to -> Wave -> Selected Signals”将选中的信号添加到 Wave 窗口。

6.5.5 运行仿真

在命令行窗口输入“run all”命令，进行仿真，仿真完成后可通过波形窗口检查观察信号仿真的波形结果。

6.5.6 迭代调试

当发现仿真结果错误后，重新修改代码，执行以下步骤：

- 运行“do path to build/build.do”重新编译。
- 运行“restart”命令重新启动仿真。
- 运行“run -all”命令，执行仿真观察结果。
- 仿真完成后，使用“quit -sim”退出仿真。

6.6 实验评估

为了验证实验的正确性，选择 10 组不同规模的向量进行内积运算测试。scale 文件描述需进行内积计算的向量规模；neuron.txt 文件存储输入的神经元数据；weight.txt 文件存储输入的权值数据；result 文件存储运算结果。实验实现的串行、并行内积运算器输出结果和 result 文件中结果都能比对正确，说明实现的串行/并行内积运算器功能正确。

本次实验的评估标准设定如下：

- 60 分：完成串行内积运算器，输出结果和 result 文件比对正确。
- 90 分：完成串行和并行内积运算器，输出结果和 result 文件比对正确。
- 120 分：完成串行、并行内积运算器和矩阵运算子单元，输出结果和 result 文件比对正确。

6.7 实验思考

对比串行内积运算器和并行内积运算器，请说明深度学习处理器加速卷积计算的原理是什么？