第 6 章 * 深度学习处理器运算器设计

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，需要设计加速深度学习算法的处理器，同时支持越来越多样化的编程需求。深度学习处理器（DLP，Deep Learning Processor）就是一类高效支持深度学习算法的处理器，其针对深度学习的通用计算进行加速，包括卷积运算、池化运算等；同时，深度学习处理器考虑深度学习算法的多样性，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重参数，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决定了深度学习算法在处理器上的性能。在智能计算系统中，设计出能高效支持卷积运算的运算器，是深度学习处理器设计的关键技术之一。

本章首先分析卷积层的算法特征，介绍面向卷积运算的 DLP 架构及其处理矩阵/卷积的过程；其次，介绍实验环境；然后介绍面向矩阵和卷积处理的 DLP 运算器的设计方法，包括串行内积运算器、并行内积运算器、矩阵运算子单元，以及如何使用 EDA 工具进行仿真调试。需要说明的是，本章的实验设计仅仅是一个教学模型，从物理实现上来看结构设计不一定很合理，主要是为了从原理上说明深度学习处理器是如何加速卷积计算。

6.1 实验目的

掌握深度学习处理器中运算器的设计原理，能够使用 Verilog 语言实现内积运算器及矩阵运算子单元的设计并进行功能仿真。具体包括：

1) 理解深度学习处理器加速卷积运算的原理，理解本实验和深度学习处理器基本模块间的关系。

2) 利用 Verilog 语言实现串行内积运算器，理解内积运算器的基本组成单元。

3) 在串行内积运算器基础上，利用 Verilog 语言实现并行内积运算器，加深对深度学习处理器加速卷积计算原理的理解。

4) 在并行内积运算器基础上，利用 Verilog 语言实现矩阵运算子单元，加深对矩阵运算子单元的理解。

实验工作量：约 300 行代码，约需 4 个小时。

6.2 背景介绍

本节首先介绍分析深度学习算法中卷积层的算法特征，然后介绍面向卷积运算的深度学习处理器架构，接下来介绍矩阵运算以及卷积层在深度学习处理器上的处理过程，最后介绍本实验环境，包括工具安装和验证环境。

6.2.1 卷积层算法特征

卷积层由输入 $\boldsymbol{X}$、输出 $\boldsymbol{Y}$ 和权重 $\boldsymbol{W}$（即卷积核）组成。假设输入 $\boldsymbol{X}$ 包含 C_i 个特征图，输出 $\boldsymbol{Y}$ 包含 C_o 个特征图，卷积核张量的形状为 $[C_o, C_i, K_w, K_h]$，其中 K_w 和 K_h 分别是卷积核的宽和高。卷积计算时，通常用所有输入特征图与 $C_i \times K_h \times K_w$ 大小的卷积核做卷积得到一个输出特征图[①]，然后输入特征图依次与其余的卷积核做卷积运算来得到所有的输出特征图。

为了提高卷积计算过程中数据的复用性，可以采用下述实现：输入特征图上 $K_w \times K_h$ 区域内的神经元和所有卷积核做卷积运算得到所有输出特征图上同一位置的神经元，然后沿着输入特征图的水平和垂直方向分别以 s_w 和 s_h 步长滑动做卷积，得到所有输出特征图的所有神经元。其中，第 c_o 个输出特征图上 (w, h) 位置的神经元的计算公式为

$$\boldsymbol{Y}(h, w, c_o) = G\left(\left(\sum_{k_w=0}^{K_w-1}\sum_{k_h=0}^{K_h-1}\sum_{c_i=0}^{C_i-1}\boldsymbol{X}(c_i, h\times s_h + k_h, w\times s_w + k_w)\right)\times \boldsymbol{W}(c_i, k_h, k_w, c_o) + \boldsymbol{b}(c_o)\right) \tag{6.1}$$

其中，$\boldsymbol{b}$ 表示偏置，G 表示激活函数。

公式(6.1)可等效为

$$\boldsymbol{Y}(h, w, c_o) = G\left(\sum_{p=0}^{C_i\times K_w\times K_h-1}\overline{\boldsymbol{X}}(h, w, p)\times \overline{\boldsymbol{W}}(p, c_o) + \boldsymbol{b}(c_o)\right) \tag{6.2}$$

其中，$\overline{\boldsymbol{X}}(h, w, :)$ 是一个 $(C_i \times K_h \times K_w)$ 维的行向量，由计算输出特征图 (w, h) 位置所需的所有输入特征图中 $K_x \times K_y$ 区域内的神经元组成；$\overline{\boldsymbol{W}}$ 为 $C_o \times (C_i \times K_h \times K_w)$ 卷积系数矩阵，由维度为 $C_i \times K_h \times K_w \times C_o$ 的卷积核张量做维度转换得到。

由公式(6.2)可知，卷积计算可以看成先做维度为 $(C_i \times K_h \times K_w)$ 的输入神经元向量和维度为 $(C_i \times K_h \times K_w) \times C_o$ 的权重矩阵相乘，然后做向量加法和向量激活。

另一方面，当 M 维的行向量 $\boldsymbol{a}$ 和 $M \times N$ 的矩阵 $\boldsymbol{B}$ 相乘时，得到的乘积向量 $\boldsymbol{c}$ 的第 i 个元素可表示为

$$\boldsymbol{c}(i) = \sum_{j=0}^{M-1}\boldsymbol{a}(j)\times \boldsymbol{B}(j, i) \tag{6.3}$$

公式(6.3)可表示为

$$\boldsymbol{c}(i) = \boldsymbol{a}\cdot \boldsymbol{B}(:, i) \tag{6.4}$$

即结果向量 $\boldsymbol{c}$ 的第 i 个元素为向量 $\boldsymbol{a}$ 和矩阵 $\boldsymbol{B}$ 的第 i 个列向量 $\boldsymbol{B}(:, i)$ 的内积。

结合公式(6.2)和公式(6.4)可知，卷积运算由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。

[①]为便于说明，本节仅考虑对所有输入特征图做卷积得到一个输出特征图的情况，实际中可能会用部分输入特征图做卷积来得到一个输出特征图。

6.2.2 面向卷积运算的 DLP 架构

深度学习处理器主要由控制模块、存储模块和运算模块三大部分组成，如图6.1所示。其中，控制模块负责协调控制运算模块和存储模块完成深度学习任务，包括取指单元（Instruction Fetch Unit，IFU）和指令译码单元（Instruction Decode Unit，IDU）。存储模块包含神经元存储单元和权重存储单元以及 DMA 单元（Direct Memory Access，直接内存存取），其中神经元存储单元（记作 NRAM）用于存储输入/输出神经元，权重存储单元（记作 WRAM）用于存储权重。运算单元包括矩阵运算单元和向量运算单元，其中矩阵运算单元（Matrix Function Unit, MFU）完成矩阵乘运算，向量运算单元（Vector Function Unit, VFU）完成其他向量运算。

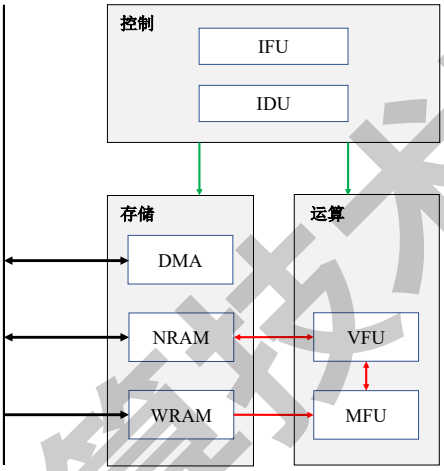

图 6.1 深度学习处理器总体架构

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 输入的加法树组成。

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 WRAM，记作 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重给 PE 进行计算。

6.2.3 DLP 上矩阵及卷积处理过程

深度学习处理器进行维度为 M 的向量 $\boldsymbol{a}$ 和维度为 $M \times N$ 的矩阵 $\boldsymbol{B}$ 相乘时，首先将矩阵 $\boldsymbol{B}$ 均分为 n 个子矩阵。每个子矩阵的维度为 $M \times \frac{N}{n}$，分别存储在 n 个 DWRAM 中。MFU 进行计算时，MFU 的 n 个 PE 接收 H 树广播的向量 $\boldsymbol{a}$ 的 m 个分量，并分别从 n 个 DWRAM 读取各自子矩阵的第 i 个列向量的 m 个分量进行内积运算，得到输出向量 $\boldsymbol{c}$ 的 n 个分量的部分和。处理器控制 NRAM 将向量 $\boldsymbol{a}$ 的所有分量都发送给 MFU 则可完成输出向量 $\boldsymbol{c}$ 的 n 个分量的计算。然后，处理器将前面步骤重复 $\frac{N}{n}$ 次，便可完成输出向量 $\boldsymbol{c}$ 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），输入神经元矩阵 $\boldsymbol{A}$ 的维

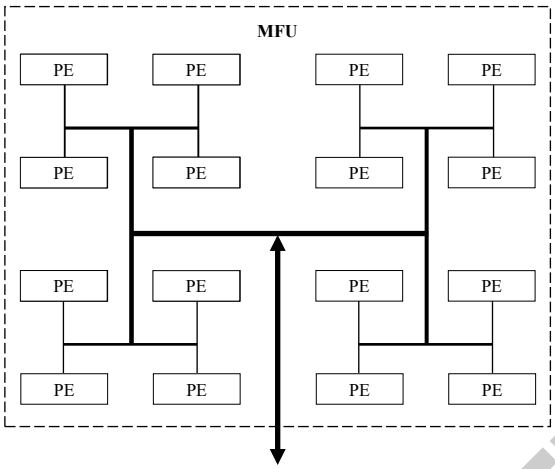

图 6.2　矩阵运算单元结构

度为 2×8，权重矩阵 $\boldsymbol{B}$ 的维度为 8×4，输出神经元 $\boldsymbol{C}$ 的维度是 2×4。矩阵运算单元进行 $\boldsymbol{C}=\boldsymbol{A}\times\boldsymbol{B}$ 运算时，每个矩阵运算子单元计算 $\boldsymbol{C}$ 的列子矩阵。如图6.3所示，矩阵运算单元需 4 拍完成矩阵运算：

- 第 1 拍，第一行的前 4 个神经元广播给所有 PE，每个 PE 接收对应列权重的前 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 接收对应列权重的后 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 接收对应列权重的前 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 接收对应列权重的后 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以 n 个特征图为一组。如图6.4所示，DLP 每次使用输入特征图中维度为 $C_i\times K_h\times K_w$ 的数据子块计算一组输出特征图中不同输出特征图上相同位置的神经元。MFU 的 n 个 PE 分别计算不同输出特征图的神经元。然后，DLP 沿特征图的水平、垂直方向循环顺序计算特征图上的神经元，得到一组完整的输出特征图。

DLP 计算一组输出特征图中不同输出特征图上相同位置的神经元过程类似于矩阵运算。运算过程可拆分为 4 个步骤：

- 步骤一：VFU 依次将维度为 $C_i\times K_h\times K_w$ 的数据子块从 NRAM 读出，进行数据预处理后发送给 MFU。
- 步骤二：MFU 将接收到的输入神经元广播给 n 个 PE 单元。
- 步骤三：PE 接收 H 树广播的输入神经元，与从 WRAM 读取的权重数据做内积运算，并将内积结果保存至部分和寄存器，或将内积结果累加到部分和寄存器。
- 步骤四：PE 收到维度为 $C_i\times K_h\times K_w$ 的数据子块的所有数据后将计算的部分和输出。

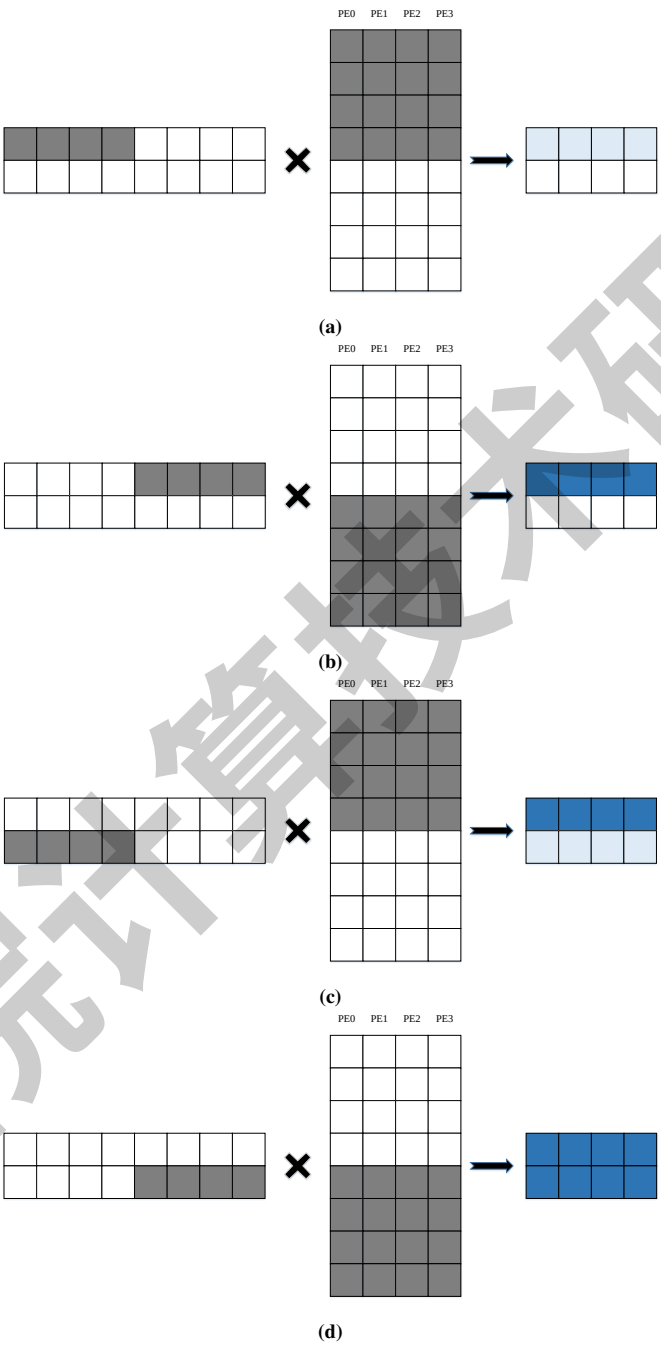

图 6.3　包含 4 个 PE 的矩阵运算单元完成矩阵乘的示意图

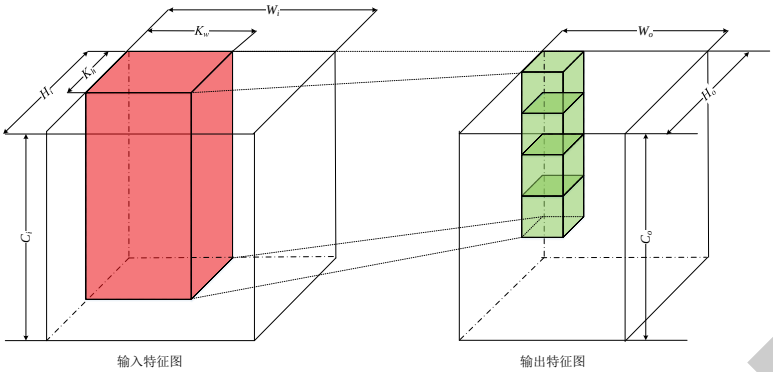

图 6.4　卷积层计算示意图

通过矩阵运算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。因此，本实验将详细介绍内积运算器的设计。

6.3　实验环境

使用 Verilog 语言实现的模块需要使用 HDL 仿真工具进行编译调试。本节以 Mentor 公司的 ModelSim 仿真工具为例，介绍实验环境的工具安装和代码文件组织。

6.3.1　工具安装

本节以 Mentor 公司的 ModelSim 10.4a（学生版）为例，介绍工具的安装。软件安装包可从 Mentor 官网下载页面（https://www.mentor.com/company/higher_ed/modelsim-student-edition）。下载安装包后，按照如下步骤安装程序：

（1）运行安装包，进入如图6.5a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6.5b所示 license 说明界面。

（3）点击“Yes”同意 license 说明内容，进入如图6.5c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6.5d界面。

（5）在“Path”栏填写软件安装路径，并点击“OK”，进入图6.5e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6.5f界面。

（7）点击“Next”确认继续安装，进入如图6.5g界面。

（8）点击“是”确认在桌面创建软件快捷方式，进入如图6.5h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6.5i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。

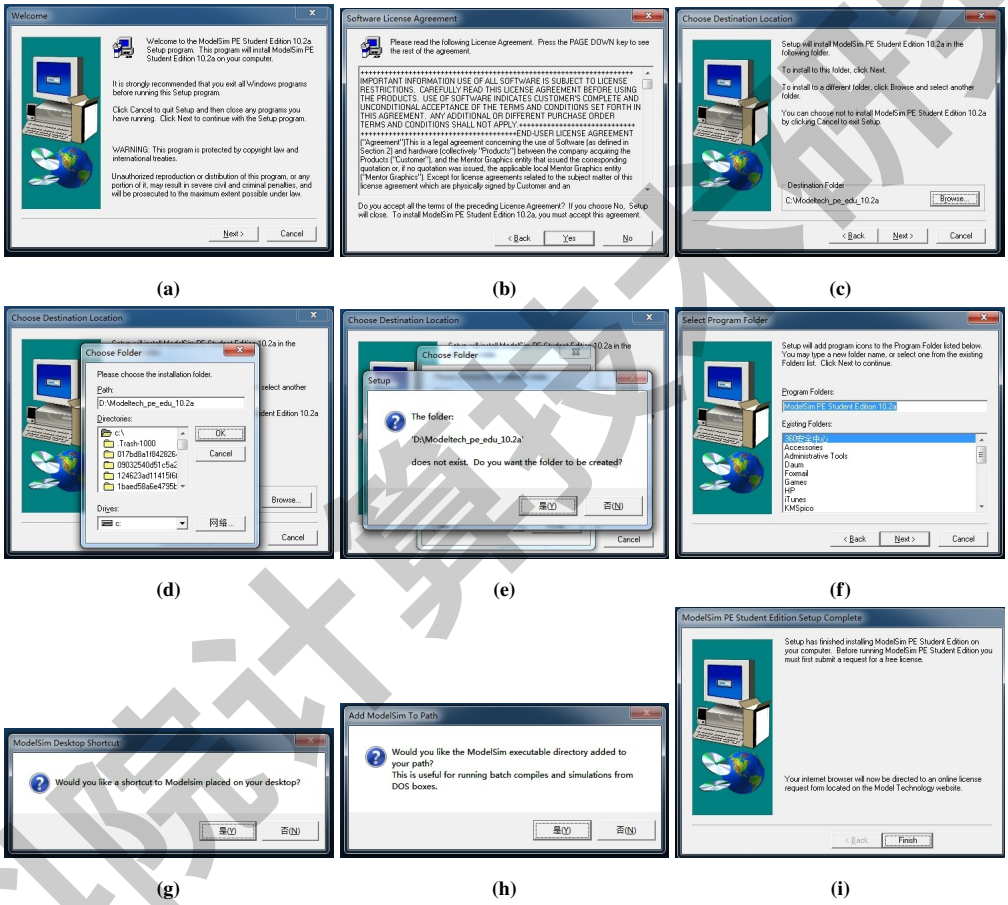

图 6.5　Modelsim 安装步骤

6.3.2 代码文件组织

实验环境文件夹组织如下：

- 目录 `src`，包含编写的实验 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，实例化矩阵运算子单元模块。
  - 文件 `build.do`，编译脚本，用于编译顶层文件和实验 Verilog 源代码。
  - 文件 `compile.f`，编译文件列表，用于指定编译脚本需要编译的文件。
  - 文件 `sim_run.do`，仿真执行脚本，用于执行仿真。
- 目录 `data`，包含仿真输入输出数据文件。
  - 向量规模描述文件 `inst`，描述需进行内积计算的向量规模。
  - 输入神经元文件 `neuron`，存储输入的神经元数据。
  - 输入权重文件 `weight`，存储输入的权重数据。
  - 输出结果文件 `result`，存储运算结果。

6.4 实验内容

本实验将实现矩阵运算子单元的内积运算器的设计。矩阵运算子单元的内积运算器的功能是对长度可变的神经元向量 `neuron` 和权重向量 `weight` 做内积运算，然后将结果输出。其功能伪代码如代码示例6.1所示。

代码示例 6.1 内积运算器功能代码

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

在通用处理器中，神经元数据和权重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，一般使用低精度运算器代替浮点运算器。根据文献[16] 所述，INT16 或者 INT8 类型已经能够满足深度学习算法的应用需求。为了简化实验，本实验实现的内积运算器的所有神经元/权重数据都采用 INT16 类型表示。

为了便于理解深度学习矩阵运算子单元的设计和迭代开发，本实验分为三个步骤：

1) 串行内积运算器设计：串行内积运算器每拍处理一个神经元和一个权重分量的乘累加运算。

2) 并行内积运算器设计：并行内积运算器每拍并行处理多个神经元和权重分量的乘累加运算。该运算器是矩阵运算子单元的基本运算单元。

3) 矩阵运算子单元设计：矩阵运算子单元（PE）可根据控制信号接收神经元数据和权重数据，然后进行内积运算。PE 可以通过在并行内积运算器基础上增加控制逻辑来实现。

最后介绍如何使用仿真工具对实现的代码进行编译和调试。

6.5 实验步骤

6.5.1 串行内积运算器

串行内积运算器的结构如图6.6所示，主要包括一个乘法器、一个加法器、一个部分和寄存器 psum 和一个数据选择器 mux。串行内积运算器每拍最多接收一个神经元（neuron）和一个权重分量（weight）进行乘法运算，然后将乘法结果累加到部分和寄存器。当串行内积运算器处理的神经元/权重数据是一组神经元/权重向量的第一个元素时，乘法结果直接写入部分和寄存器，不需进行累加；当处理的神经元/权重数据是一组神经元/权重向量的最后一个元素时，乘法结果累加到部分和寄存器，然后将部分和寄存器的值输出到 result 端口。数据选择器用于选择写入部分和寄存器的源数据。

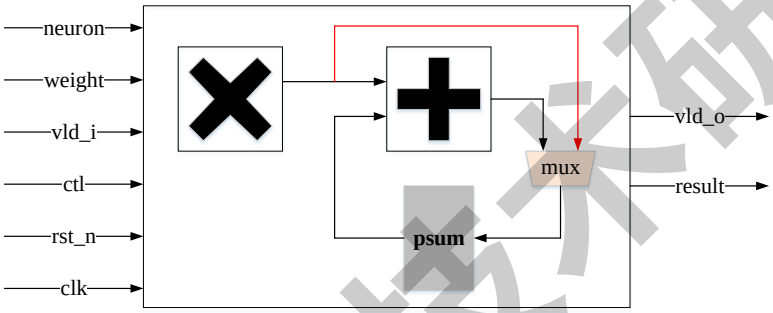

图 6.6 串行内积运算器

串行内积运算器的输入输出接口信号如表6.1所示。输入神经元和权重均是 1 个 INT16 类型的数据，输出神经元是 1 个 32 位宽的数据。输入控制信号 ctl 的位宽为 2，其中 ctl[0] 表示输入神经元/权重数据是一组神经元/权重向量的第一个元素，ctl[1] 表示输入神经元/权重数据是一组神经元/权重向量的最后一个元素。

串行内积运算器的 Verilog 实现如代码示例6.2所示。串行内积运算器采用异步复位方式，复位信号 rst_n 低电平有效。复位时，部分和寄存器被清零。输入信号 vld_i 为输入神经元/权重数据和控制信号有效标志，当 vld_i 为高电平时，串行内积运算器接收输入的神经元和权重分量进行乘法运算，然后再用乘法结果更新部分和寄存器。当输入控制信号最低位 ctl[0] 为高电平时，乘法结果直接写入部分和寄存器；否则，乘法结果先与部分和寄存器累加，再将累加结果写入部分和寄存器。当输入控制信号最高位 ctl[1] 为高电平时，串行内积运算器在下一个时钟周期将部分和寄存器的值输出到 result 端口，并将输出内积结果有效标志 vld_o 置起一拍。

输入神经元向量和权重向量中的每个神经元和权重数据都是有符号数据，乘法运算得到的部分和也是有符号数据，且部分和数据位宽为对应神经元位宽与权重位宽之和。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式说明。

代码示例 6.2 串行内积运算器代码

```
1 /*file: serial_pe.v*/
2 module serial_pe(
3   input               clk,
4   input               rst_n,
```

表 6.1　串行内积运算器输入输出接口信号

域	位宽	功能描述
neuron	16	输入 INT16 神经元分量
weight	16	输入 INT16 权重分量
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/权重数据是一组神经元/权重向量的第一个元素，高电平有效 ctl[1]: 输入神经元/权重数据是一组神经元/权重向量的最后一个元素，高电平有效
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

```
  input  signed [15:0] neuron,
  input  signed [15:0] weight,
  input         [ 1:0] ctl,
  input                vld_i,
  output        [31:0] result,
  output reg           vld_o
);
/*multiplier*/ /*TODO*/
wire signed [31:0] mult_res = ____________________________;
reg [31:0] psum_r;

/*adder*/ /*TODO*/
wire [31:0] psum_d = ____________________________;

/*partial sum reg*/
always@(posedge clk or negedge rst_n)
if(!rst_n) begin
  psum_r <= 32'h0;
end else if(vld_i) begin
  psum_r <= psum_d;
end

always@(posedge clk or negedge rst_n)
if(!rst_n) begin
  vld_o <= 1'b0;
end else if(____________________________) begin
  vld_o <= 1'b1;
end else begin
  vld_o <= 1'b0;
end

assign result = psum_r;

endmodule
```

6.5.2　并行内积运算器

不同于串行内积运算器每拍最多接收一个神经元和一个权重分量进行乘法运算，并行内积运算器每拍接收多个神经元和权重分量。并行内积运算器的结构如图6.7所示，包含一组（32 个）并行乘法器、一个累加单元、一个加法器、一个部分和寄存器 psum 和一个数据选择器 mux。并行内积运算器每次接收一个神经元向量（包含 32 个 INT16 分量）和一

个权重向量（包含 32 个 INT16 分量）进行向量内积运算，然后通过对连续 32 个 INT16 分量的内积结果的累加来支持更长神经元向量和权重向量的内积运算。

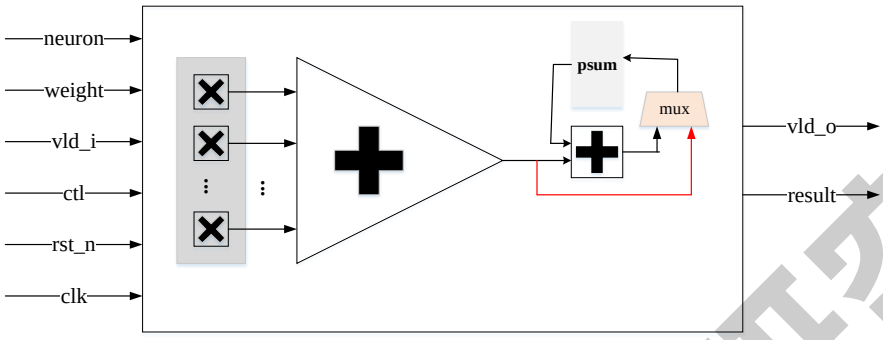

图 6.7　并行内积运算器

并行内积运算器的输入输出接口信号如表6.2所示，与串行内积运算器的接口信号类似。不同之处在于，并行内积运算器输入的神经元数据和权重数据均为维度为 32 的 INT16 向量，而串行内积运算器输入的神经元数据和权重数据均是 1 个 INT16 类型的数据。输入控制信号 `ctl` 的定义与串行内积运算器类似，`ctl[0]` 表示输入神经元/权重数据是一组神经元/权重向量的第一个子向量，`ctl[1]` 表示输入神经元/权重数据是一组神经元/权重向量的最后一个子向量。

表 6.2　并行内积运算器输入输出接口信号

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	输入权重数据，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/权重数据是一组神经元/权重向量的第一个子向量，高电平有效 `ctl[1]`: 输入神经元/权重数据是一组神经元/权重向量的最后一个子向量，高电平有效
`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

下面依次介绍并行内积运算器中的并行乘法器、累加单元和顶层模块的实现。

32 个并行乘法器的 Verilog 实现如代码示例6.3所示。对于输入的 INT16 神经元向量和 INT16 权重向量，将不同的神经元分量和权重分量输入对应的乘法器进行乘法运算得到部分积，然后将部分积输出。

代码示例 6.3　并行乘法运算器代码

```
1 /* file: pe_mult.v */
2 module pe_mult(
3   input  [ 511:0] mult_neuron,
4   input  [ 511:0] mult_weight,
5   output [1023:0] mult_result
6 );
```

```
7
8  /* int16 mult */
9  genvar i;
10 wire signed [15:0] int16_neuron[31:0];
11 wire signed [15:0] int16_weight[31:0];
12 wire signed [31:0] int16_mult_result[31:0];
13 generate
14   for(i=0; i<32; i=i+1)
15   begin:int16_mult      /* TODO */
16     ____________________________
17     ____________________________
18     ____________________________
19     ____________________________
20   end
21 endgenerate
22
23 endmodule
```

累加单元的 Verilog 实现如代码示例6.4所示，将 32 个输入部分积累加成一个部分和。

代码示例 6.4　累加单元代码

```
1  /*file: pe_acc.v*/
2  module pe_acc(
3    input  [1023:0] mult_result,
4    output [  31:0] acc_result
5  );
6  genvar i;
7  genvar j;
8
9  /* int16 add tree */
10 wire [31:0] int16_result[5:0][31:0];
11 for(i=0; i<=5; i=i+1)
12 begin:int16_add_tree
13   for(j=0; j<(32/(2**i)); j=j+1)
14   begin:int16_adder
15     if(i==0) begin /* TODO */
16       assign int16_result[0][j] = ____________________________;
17     end else begin /* TODO */
18       assign int16_result[i][j] = ____________________________;
19     end
20   end
21 end
22 assign acc_result = int16_result[5][0];
23 endmodule
```

并行内积运算器的顶层模块实现如代码示例6.5所示。当输入信号 `vld_i` 为高电平时，并行内积运算器接收输入的 INT16 神经元向量和 INT16 权重向量，通过并行乘法器将权重分量和神经元分量相乘得到多个部分积，然后通过累加单元将多个部分积累加为一个部分和，最后通过控制信号将累加单元输出的结果和部分和寄存器进行累加，并生成输出结果有效信号。当控制信号最低位 `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。当控制信号最高位 `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的值输出到 `result` 端口，并将输出内积结果有效标志 `vld_o` 置起一拍。

代码示例 6.5　并行内积运算器代码

```
/*file: parallel_pe.v*/
module parallel_pe(
  input                 clk,
  input                 rst_n,
  input       [511:0]   neuron,
  input       [511:0]   weight,
  input       [  1:0]   ctl,
  input                 vld_i,
  output      [ 31:0]   result,
  output reg            vld_o
);
wire [1023:0] mult_result;
pe_mult u_pe_mult(
  .mult_neuron (neuron),
  .mult_weight (weight),
  .mult_result (mult_result)
);

wire [31:0] acc_result;
pe_acc u_pe_acc(
  .mult_result (mult_result),
  .acc_result  (acc_result)
);

reg [31:0] psum_r;
wire [31:0] psum_d = ______________________________; /*TODO*/

always@(posedge clk or negedge rst_n)
if(!rst_n) begin
  psum_r <= 32'h0;
end else if(vld_i) begin
  psum_r <= psum_d;
end

always@(posedge clk or negedge rst_n)
if(!rst_n) begin
  vld_o <= 1'b0;
end else if(_____________________________) begin
  vld_o <= 1'b1;
end else begin
  vld_o <= 1'b0;
end

assign result = acc_result;
endmodule
```

6.5.3 矩阵运算子单元

矩阵运算子单元的结构如图6.8所示，在并行内积运算器的基础上增加了用于控制神经元/权重数据、控制信号输入并行内积运算器以及部分和输出的控制单元（记作 CTL）。根据第6.2.2节的描述，矩阵运算子单元根据控制信号来接收 H 树广播的神经元向量和从 WRAM 读取的权重向量数据进行内积运算。由于矩阵运算子单元和 H 树总线、WRAM 相连接，对应神经元、权重、输出结果的接口信号使用 valid-ready 握手机制的总线信号。

矩阵运算子单元的输入输出接口信号如表6.3所示。输入神经元/权重/控制信号都分别包含输入数据/信号、输入有效标志 `valid`、以及控制单元可接收神经元/权重/控制信号标志 `ready`。当且仅当表示输入数据有效标志 `valid` 和控制单元可接收标志 `ready` 都有效

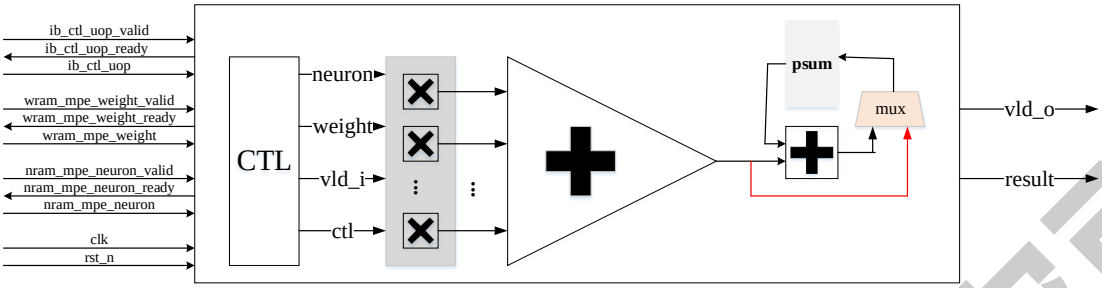

图 6.8　矩阵运算子单元

时，控制单元才成功接收输入数据/控制信号。输入控制信号 `ib_ctl_uop` 表示计算一个输出神经元所对应的输入神经元/权重的长度（长度单位为 64 字节），该信号的位宽为 8。与串行/并行内积运算器的输出一样，输出神经元是 1 个 32 位宽的数据。

表 6.3　矩阵运算子单元输入输出接口信号

域	位宽	功能描述
`ib_ctl_uop_valid`	1	输入控制信号有效标志，高电平有效
`ib_ctl_uop_ready`	1	控制单元可接收控制信号标志，高电平有效
`ib_ctl_uop`	8	输入控制信号，计算的神经元/权重数据的长度（单位：64 字节）
`wram_mpe_weight_valid`	1	输入权重有效标志，高电平有效
`wram_mpe_weight_ready`	1	控制单元可接收权重标志，高电平有效
`wram_mpe_weight`	512	输入权重数据，包括 32 个 INT16 分量
`nram_mpe_neuron_valid`	1	输入神经元有效标志，高电平有效
`nram_mpe_neuron_ready`	1	控制单元可接收神经元标志，高电平有效
`nram_mpe_neuron`	512	输入神经元数据，包括 32 个 INT16 分量
`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

矩阵运算子单元的 Verilog 实现如代码示例6.6所示。控制单元接收输入控制信号 `ib_ctl_uop`，译码生成输出给并行内积运算器的控制信号 `pe_ctl`。假设 `ib_ctl_uop` 的值为 k，则计算一个输出神经元需要循环输入 k 个输入神经元/权重向量，并做 k 次乘累加，因此控制单元将生成 k 条给并行内积运算器的控制信号 `pe_ctl`。生成的第一条控制信号中 `pe_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；生成的最后一条控制信号中 `pe_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。当前输入控制信号 `ib_ctl_uop` 译码生成所有控制信号 `pe_ctl` 之后，控制单元可以接收下一条输入控制信号进行译码，输出信号 `ib_ctl_uop_ready` 变为高电平。仅当输入的神经元数据、权重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并行内积运算器，并完成输入神经元/权重数据的握手。

代码示例 6.6　矩阵运算子单元

```
1 /*file: matrix_pe.v*/
2 module matrix_pe(
3   input                clk,
4   input                rst_n,
```

```
  input          [511:0] nram_mpe_neuron,
  input                  nram_mpe_neuron_valid,
  output                 nram_mpe_neuron_ready,
  input          [511:0] wram_mpe_weight,
  input                  wram_mpe_weight_valid,
  output                 wram_mpe_weight_ready,
  input          [  7:0] ib_ctl_uop,
  input                  ib_ctl_uop_valid,
  output reg             ib_ctl_uop_ready,
  output         [ 31:0] result,
  output                   vld_o
);
reg inst_vld;
reg [7:0] inst, iter; /*inst存放输入控制信号ib_ctl_uop的值*/
always@(posedge clk or negedge rst_n) begin
  /*TODO: inst_vld & inst*/
  ____________________________
  ____________________________
  ____________________________
  ____________________________
end
always@(posedge clk or negedge rst_n) begin
  /*TODO: iter*/
  ____________________________
  ____________________________
  ____________________________
  ____________________________
end
always@(posedge clk or negedge rst_n) begin
  /*TODO: ib_ctl_uop_ready*/
  ____________________________
  ____________________________
  ____________________________
  ____________________________
end

wire [1:0] pe_ctl;
assign pe_ctl[0]  = ____________________________; /*TODO*/
assign pe_ctl[1]  = ____________________________; /*TODO*/
wire pe_vld_i     = ____________________________; /*TODO*/
wire [31:0] pe_result;
wire pe_vld_o;
parallel_pe u_parallel_pe (
/*TODO*/
  ____________________________
  ____________________________
  ____________________________
  ____________________________
);

assign nram_mpe_neuron_ready = ____________________________; /*TODO*/
assign wram_mpe_weight_ready = ____________________________; /*TODO*/
assign result = pe_result;
assign vld_o = pe_vld_o;
endmodule
```

6.5.4 编译调试

6.5.4.1 编写仿真顶层文件

仿真顶层文件 `tb_top.v` 将读取控制信号文件 `inst`，输入到矩阵运算子单元的控制信号端口；并读取神经元数据文件 `neuron` 和权重数据文件 `weight`，输入到矩阵运算子单元的神经元端口和权重端口。

在 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 `$readmemb` 系统任务，每个数字必须使用二进制表示，对于 `$readmemh` 系统任务，每个数字必须使用十六进制表示。这两个系统函数可以在仿真的任何时刻被执行，使用格式共六种：

- `$readmemb(“数据文件名”, 存储器名);`
- `$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”表示被读取的数据文件，包含输入文件的路径和文件名，如代码示例6.7所示。

代码示例 6.7 设置仿真数据路径

```
initial
begin
  $readmemb("D:/pe_exp/data/inst", inst);
  $readmemh("D:/pe_exp/data/neuron", neuron);
  $readmemh("D:/pe_exp/data/weight", weight);
  $readmemh("D:/pe_exp/data/result", result);
 end
```

6.5.4.2 编译代码

仿真顶层文件编写完成后，可通过 ModelSim 工具编译实验源代码和对应的顶层文件。代码编译前，首先需通过代码示例6.8所示的编译列表指定需编译的顶层文件和源代码文件。

代码示例 6.8 编译列表

```
//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
```

然后，在 ModelSim 软件命令行窗口输入“`do path to build/build.do`”命令，工具开始执行代码示例6.9中的编译脚本来编译测试顶层文件和源代码。

代码示例 6.9　编译脚本

```
# 设置方针环境路径
# TODO
set sim_home Path/to/simulation/Dir

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

#将work目录下的数据文件应设为一个叫做work的方针哭
vmap work ${sim_home}/work

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

ModelSim 编译代码时，命令行窗口中将显示编译的文件、顶层文件以及编译错误和警告数量，如图6.9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 6.9　编译日志

6.5.4.3　启动仿真

在 ModelSim 命令行窗口输入“`do path to build/sim_run.do`”命令，运行代码示例6.10所示的执行脚本启动仿真。

代码示例 6.10　执行脚本

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

执行脚本中关键参数含义为：

- `+nowarnTSCALE` 表示忽略没有时间尺度定义的文件，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`。

ModelSim 运行执行脚本后将显示如图6.10所示的仿真实例面板，以及仿真模块的层次关系和模块的信号名称。

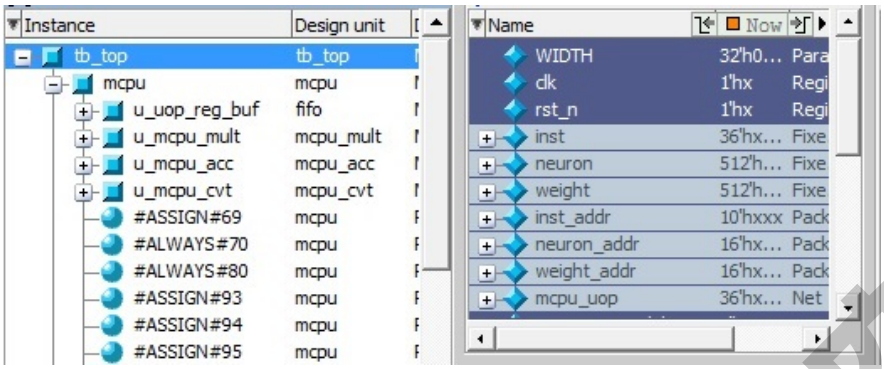

图 6.10　仿真实例面板

6.5.4.4　添加观测信号

在仿真实例面板，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。

6.5.4.5　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。

6.5.4.6　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：

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

6.6　实验评估

实现串行内积运算器、并行内积运算器、矩阵运算子单元，然后分别通过 ModelSim 仿真 4 组不同规模的向量进行内积运算。内积运算结果和 `result` 文件中的结果都能比对正确，说明实现的串行内积运算器、并行内积运算器和矩阵运算子单元功能正确。

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。
- 100 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：完成串行内积运算器、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

1) 对比分析串行内积运算器和并行内积运算器完成 `neuron` 文件和 `weight` 文件内积运算所需的时钟周期数。

2) 请说明深度学习处理器加速卷积计算的原理是什么?