Titan-TPU V2 架构与设计原理全解 (The Bible)

1. 项目概述 (Executive Summary)

Titan-TPU V2 是一个基于 SystemVerilog 实现的深度学习加速器（Deep Learning Accelerator），其核心架构源自开源项目 `tiny-tpu-v2` 。该项目旨在在 FPGA/ASIC 平台上实现矩阵乘法加速，采用经典的 **脉动阵列 (Systolic Array)** 架构。

核心特性：

- **数据流模式**：权重驻留 (Weight Stationary)。
- **精度**：16-bit 定点数 (Fixed Point, Q8.8 格式)。
- **架构**：2x2 脉动阵列（可扩展），配合统一缓冲（Unified Buffer）。
- **控制**：基于自定义 ISA 的指令集控制。

2. 核心计算单元：处理引擎 (PE)

Processing Element (PE) 是 TPU 的心脏。在 Titan-TPU V2 中，PE 负责执行乘累加运算 (MAC: Multiply-Accumulate)。

2.1 接口定义与数据流方向

PE 位于脉动阵列的网格中，具有四个方向的连接：

- **North (北)**：接收部分和 (`pe_psum_in`) 或权重加载信号。
- **South (南)**：输出计算后的部分和 (`pe_psum_out`) 给下一行 PE。
- **West (西)**：接收输入激活数据 (`pe_input_in`) 和控制信号。
- **East (东)**：将输入数据 (`pe_input_out`) 传递给下一列 PE。

2.2 内部寄存器策略

PE 内部采用了**双重权重寄存器**设计，以支持计算与加载的流水线化：

1. `weight_reg_active` **(前台寄存器)**：当前计算正在使用的权重。
2. `weight_reg_inactive` **(后台寄存器/影子寄存器)**：用于预加载下一个时刻的权重。

权重更新机制： 当 `pe_switch_in` 信号有效时，后台寄存器的值被复制到前台寄存器。这允许在计算当前层的同时，预加载下一层的权重，消除层与层之间的停顿 (Bubble)。

2.3 计算逻辑 (MAC)

核心算式为：

$$Out_{south} = (In_{west} \times Weight_{active}) + In_{north}$$

SystemVerilog 实现逻辑：

```
// 乘法
fxp_mul mult (.ina(pe_input_in), .inb(weight_reg_active), .out(mult_out), ...);
// 累加
fxp_add adder (.ina(mult_out), .inb(pe_psum_in), .out(mac_out), ...);
```

3. 脉动阵列 (Systolic Array) 数据流

Titan-TPU V2 采用 **Weight Stationary (权重驻留)** 数据流。

3.1 为什么选择权重驻留?

在卷积神经网络 (CNN) 或全连接层中，权重通常被复用多次。将权重固定在 PE 内部，让输入数据 (Input Feature Map) 从左向右流动，让部分和 (Partial Sum) 从上向下流动，可以最大化减少权重的内存访问能耗。

3.2 数据流动时序

1. **预加载阶段 (Preload)**：
   - 通过纵向（从北向南）的链路，将权重串行加载到每一列 PE 的 `weight_reg_inactive` 中。
   - 触发 `pe_switch` ，权重生效。
2. **计算阶段 (Compute)**：
   - 输入数据 `Input` 从阵列左侧进入，每个周期向右移动一格。
   - 部分和 `PSum` 初始化为 0，从阵列顶部进入，每个周期向下移动一格。
   - **错位输入 (Skewing)**：为了保证数据在正确的时间相遇，输入数据需要呈“阶梯状”延时进入阵列。

4. 魔改计划 (Magic Modification Plan)

为了提升 Titan-TPU V2 的工业级可用性，项目包含以下改进计划：

4.1 ECC (Error Correction Code)

- **目标**：在 Unified Buffer (SRAM) 中增加数据保护。
- **方案**：采用 Hamming Code 实现 SECDED (Single Error Correction, Double Error Detection)。
- **影响**：数据位宽将从 16-bit 扩展到 22-bit (16 data + 6 parity)，需要修改 Buffer 的读写接口。

4.2 AXI4 总线接口

- **目标**：使 TPU 能挂载到现代 SoC (如 Xilinx Zynq 或 RISC-V SoC) 总线上。
- **方案**：
  - **AXI-Lite**：用于配置 TPU 的控制寄存器 (Start, Status, Address Pointers)。
  - **AXI-Stream** 或 **AXI-Full**：用于高带宽的数据搬运 (DMA <-> Unified Buffer)。

5. 验证策略 (Verification)

验证分为三个层级：

1. **Block Level (PE)**：验证单个 MAC 运算、权重加载和影子寄存器切换逻辑。使用 Directed Test (定向测试)。
2. **Sub-System Level (Array)**：验证数据流错位 (Skew) 是否正确，矩阵乘法结果是否符合 Python Golden Model。
3. **System Level (Top)**：验证指令集 (ISA) 解析、DMA 数据搬运及完整神经网络推理。