电子科技大学示范性微电子学院

实 验 报 告

(2024.11 -2024.12)

课程名称 IC 综合实验 2

实验名称 IC 综合实验 2

指导老师 王忆文

学生姓名 冯梦奇，章海，练安达，贾豪峰

学生学号 2022340103005 2022340105030
2022340103012 2022340104009

电 子 科 技 大 学

实 验 报 告

实验地点：清水河校区国际菁蓉创新中心 B523

实验时间：2024.11-2024.12

报告目录

实验报告成绩：_____

附页：

四、实验原理：

4.1 项目简介

随着物联网（IoT）、智能设备和人工智能（AI）技术的快速发展，边缘计算成为了现代技术架构中的一个重要组成部分。边缘设备通常指的是那些接近数据源端、具备一定计算能力的终端设备，如智能摄像头、无人机、机器人、智能穿戴设备等。这些设备通过内置的传感器和处理单元能够实时捕获并处理数据，从而实现对环境的感知和智能决策。

图像识别技术作为计算机视觉领域的重要应用，已经在多个行业中得到了广泛的应用，如安防监控、智能交通、医疗影像分析、工业自动化等。然而，传统的图像识别系统大多依赖于云端处理，要求将大量图像数据传输至云端进行计算分析，这不仅带来巨大的网络带宽和延迟负担，也存在隐私安全等风险。

因此，如何将图像识别算法有效地部署到边缘设备上，成为了当前计算机视觉领域的一个重要研究课题。边缘设备具有本地计算和存储能力，能够在数据采集端进行初步处理和实时分析，减少对网络带宽的依赖，同时降低数据传输的延迟，提升实时性。与此同时，由于边缘设备的计算资源和功耗限制，如何在有限的硬件环境下实现高效、准确的图像识别，仍然是一个具有挑战性的技术难题。

本项目旨在探索并解决在边缘设备上部署图像识别算法时面临的挑战，设计高效的模型和优化策略，以实现在有限的面积上部署高速及高精度的图像识别解决方案，推动边缘计算在智能感知领域的应用与发展。

4.1.1 当前边缘设备图像识别方案的问题分析

随着边缘计算和人工智能的快速发展，边缘设备上的图像识别应用逐渐成为热点。然而，当前边缘设备上的图像识别方案在实际应用中仍面临一些显著的挑战和局限性。

（1）面积和硬件资源限制

边缘设备通常具有较小的体积和有限的计算资源，尤其是在低功耗设备中，这使得其在执行复杂的图像识别任务时面临很大的挑战。大多数现有的图像识别模型，如卷积神经网络（CNN）等，通常需要较大的存储空间和强大的计算能力，而边缘设备的硬件资源（如处理器、内存、存储等）通常无法满足这些需求。为了在边缘设备上高效运行这些模型，必须对模型进行压缩和优化，但这种优化可能会牺牲模型的准确性或计算效率。

（2）高速和高准确率的要求

边缘设备上的图像识别任务常常要求在实时性和准确性之间找到平衡。例如，在自动驾驶、安防监控等场景中，识别任务必须在极短的时间内做出决策，以确保系统的响应速度和安全性。然而，传统的图像识别模型往往需要较长的计算时间和高昂的计算资源消耗，导致无法满足边缘设备对低延迟和高实时性的要求。此外，边缘设备的计算能力有限，这使得在保证准确率的前提下，模型往往会受到计算速度的限制。

（3）功耗和计算效率

由于边缘设备往往依赖于电池供电，功耗控制是设计高效图像识别系统时必须考虑的另一个重要因素。过高的功耗不仅会影响设备的续航时间，还可能导致设备过热、性能下降等问题。因此，如何在保证高精度的同时，减少功耗，提升计算效率，是边缘设备图像识别方案中的关键问题。

（4）算法优化的挑战

为了使图像识别模型适应边缘设备，常见的优化方法包括模型剪枝、量化、知识蒸馏等。然而，这些方法虽然可以有效地减小模型的体积和计算复杂度，但可能会导致精度下降。如何在减小模型大小、提高计算效率和保持较高精度之间找到合适的平衡，是当前研究的一个难点。

4.2.2 项目目标

本项目将以手写体数字识别任务为基础，探索并提出一套针对边缘设备的高效图像识别解决方案，具体目标包括以下几点：

（1）轻量级模型设计

通过设计和选择适合边缘设备的轻量级深度学习模型（如 BNN 等），减少模型的参数量和计算量，从而降低边缘设备的硬件需求。这些轻量级模型在保证图像识别精度的基础上，能够大幅度减少计算资源的消耗，提高处理速度。

（2）模型压缩与优化

结合模型剪枝、量化和量化损失感知策略等技术，进一步优化图像识别模型，减小模型的体积和计算量。这些优化技术能够在不显著降低识别精度的情况下，显著提升边缘设备的计算效率和响应速度，从而满足实时性要求。

（3）低功耗优化策略

在模型设计和实现过程中，重点考虑功耗控制问题。通过减少不必要的计算操作、使用低功耗的硬件架构（如门控时钟等），以及采用节能算法优化，尽可能降低系统的功耗，使得边缘设备能够在长时间内稳定运行。

（4）定制的网络结构与硬件契合

针对边缘设备的硬件架构，利用硬件和软件协同优化的策略，以最大限度地发挥硬件的计算能力，确保高效的实时处理。进一步提升图像识别任务的处理速度和效率。

（5）手写体数字识别任务验证

本项目将选择手写体数字识别作为具体任务进行验证。手写体数字识别在很

多实际应用场景中具有重要意义，如银行票据处理、自动表单识别等。通过对手写体数字识别任务的实现，探索不同优化方案对识别精度、计算速度和功耗的影响，最终提出一个能够在边缘设备上高效执行的解决方案。

本项目将通过上述优化策略，针对当前边缘设备图像识别方案的瓶颈问题，提出有效的解决方案，以实现**高精度、低延迟、低功耗**的图像识别功能，推动边缘设备在智能感知领域的应用发展。

4.2 软件算法设计

4.2.1 轻量化网络选取

采用 BNN 而非 CNN 进行轻量化图像识别的方案

在边缘设备上进行图像识别时，计算资源和功耗是必须考虑的关键因素。传统的卷积神经网络（CNN）虽然在图像识别任务中取得了显著的成功，但由于其较大的计算复杂度和存储需求，难以适应边缘设备的资源限制。为了实现高效的图像识别，二值神经网络（Binary Neural Networks，BNN）作为一种轻量化的网络结构，近年来引起了广泛关注。BNN 通过将神经网络的权重二值化，大幅度减少了计算资源的需求，从而提供了一种适用于边缘设备的高效解决方案。

4.2.1.1 BNN 的基本概念

BNN 是一种将神经网络中的权重和激活函数限制为二值（±1）而不是实数值的神经网络模型。与传统 CNN 不同，BNN 的核心思想是在前向传播和反向传播过程中，所有的权重都被强制转换为二值形式。这意味着在计算过程中，不再进行浮点数运算，而是通过二值乘法和加法操作来代替常规的矩阵运算。由于二值计算在硬件上的实现更加高效，BNN 可以显著减少计算复杂度并降低存储需求。

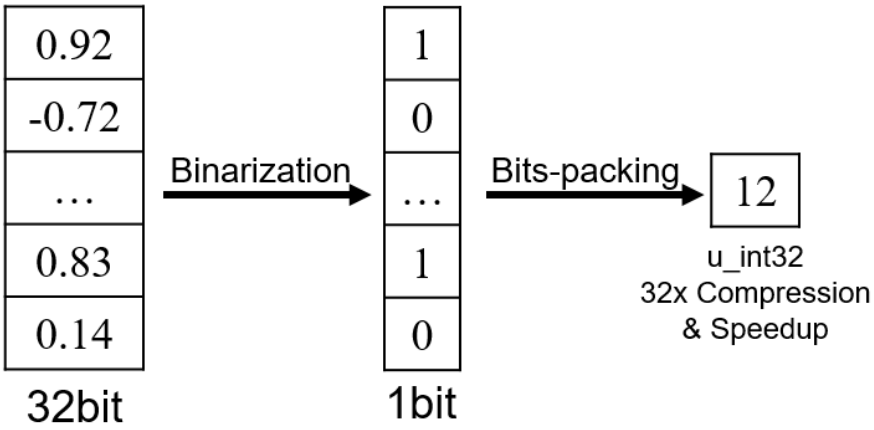


图 1. 全精度权重的二值化与压缩操作

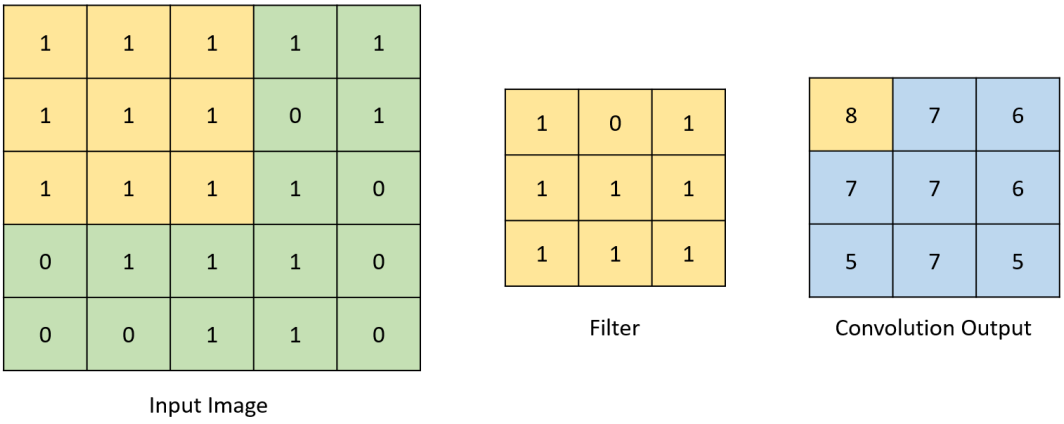


图 2. 二值化卷积核卷积操作

4.2.1.2 BNN 与 CNN 的对比

1.计算复杂度

在传统的 CNN 中，每个卷积操作都涉及到浮点数的乘法和加法，这对于边缘设备来说是一个沉重的负担，尤其是在实时性要求较高的应用场景中。相比之下，BNN 将权重二值化后，计算过程主要依赖于二值乘法（XOR 操作）和加法，极大地降低了计算量。二值化操作不仅简化了矩阵运算，还可以通过硬件优化（如专用的二值计算加速器）进一步加速计算速度。

2.存储需求

CNN 的模型通常需要大量的存储空间来存储高精度的权重和激活值，尤其是在大规模数据集上训练时，存储需求会呈指数级增长。而 BNN 通过将每个权重限制为一个比特（0 或 1），显著减少了存储空间的需求。BNN 的权重矩阵可以用二进制格式存储，相比传统 FP32 的 CNN 节省了大约 31 倍的存储空间，使得模型能够在边缘设备上运行。

3.精度与性能

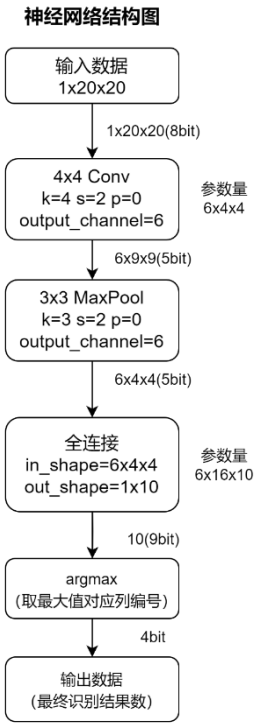
虽然 BNN 的二值化操作会对精度造成一定影响，但通过合适的网络结构设计和优化策略，BNN 可以在许多应用中保持与 CNN 相当的识别精度。例如，通过使用较深的网络结构和适当的训练技巧（如量化损失感知策略、scaling 策略和剪枝等），BNN 能够在精度损失和计算效率之间取得良好的平衡。事实上，针对手写体数字识别等任务，BNN 已经在一些公开数据集（如 MNIST、CIFAR-10）上取得了与 CNN 相近的性能。

4.2.1.3 总结

采用 BNN 进行轻量化网络设计，为边缘设备上的图像识别提供了一种高效、低功耗的解决方案。通过二值化计算，BNN 不仅减少了模型的存储需求和计算复杂度，还能在保持较高精度的同时，实现快速、实时的图像识别处理。通过在手写体数字识别任务中的应用，本项目将验证 BNN 在边缘设备上的可行性，推动边缘计算技术在智能感知领域的应用发展。

4.2.2 定制化网络结构设计

对 MNIST 数据集图片进行中心裁剪得到输入数据为 20*20 的图像矩阵，经过卷积核（4*4*6 通道，s=2）得到卷积层输出 9*9*6 矩阵，再经过池化层（3*3，s=2）得到池化层输出 4*4*6 矩阵，然后通过全连接层（4*4*6*10）得到全连接层输出 1*10，最后通过 Argmax 输出预测的 0-9 其中的一个数字。通过定制的网络结构可以发现全连接核与卷积核大小相同，因此可以通过复用卷积核进行全连接层的计算来减小面积开支。



4.2.2.1 网络结构中可选择的超参量和选择时的考虑因素

网络结构中网络结构可选的超参量：

1.输入数据压缩程度

2.卷积层、池化层、全连接层的大小

3.通道数

选择的综合考虑因素：

1.面积限制

2.预测准确率要求

3.与硬件适配性（e.g. 卷积层与全连接层复用）

4.2.2.2 各参量计算过程

设 x 为输入数据的大小，a 为复用的卷积核与全连接层尺寸，b 为池化层尺寸，根据卷积的操作规则可以得出以下公式：

$$\frac{\left(\frac{(x-a)}{2}+1\right)-b}{2}+1=a$$

其中要求 x、a、b 为整数且必须处在一定合理的区间，通过适当迭代后我们选择 x=20，a=4，b=3。卷积核的通道数越大预测准确率越高，但是过大的并行卷积核会产生无法承受的面积代价，已通过 python 验证选择通道为 3、6、10、12 时预测准确率大概为 85%、91%、92%、93%，最终综合考虑面积、准确率与速度的要求后选择通道数为 6。

4.2.3 训练问题与解决方案

4.2.3.1 二值化前向函数不可导

由于二值化操作使前向函数中加入了一个 sign 函数，sign 函数具有不可导性，这会在训练反向传播时产生一些问题，最直接的解决方法就是使用 STE (Straight-Through-Estimator), 但是有时简单的 STE 并不能达到期望的精度，于是本次实验我们采用了 Sigmoid 函数来近似 sign 函数从而进行反向传播训练。

4.2.3.2 量化与精度损失

由于全连接层巨大的参数量导致暂存数据和权重数据的存储压力显著增加，为了减小存储压力，本工作采用一种量化策略减小暂存数据的位宽。通过打印暂存值发现大部分数据在高位宽出现数据冗余的现象，通过有选择的截取位宽可以减小存储压力的同时控制精度的小幅度下降。由于本工作二值化量化和数据位宽量化的量化操作导致预测准确率相较于全精度网络下降，为了减小精度的损失，在训练时引入这种量化损失帮助网络能应对量化误差，即量化损失感知策略。具体操作为在网络训练的代价函数中加入前馈时量化误差，将减小量化误差也作为网络训练的目标。同时为了使梯度回传时能与原梯度处于近似相等的梯度空间中，我们引入了一个 scaling 策略来缩放回传的梯度值

4.2.4 模拟硬件的推理过程

实际的 Pytorch 环境只能支持全精度的浮点数运算，为了能更加模拟实际的电路硬件的推理过程，我们定义了一个 watch_net 可以实时打印任何一个数据在任何一个层的中间激活值，主要是使用软件模拟二值化乘加的补码运算与位宽截取的操作

4.3 硬件芯片实现过程

通过撰写 verilog 代码实现硬件与软件系统协同设计，而在硬件设计中尤其需要注意以下三个流程，才能对结果的正确与否有一个完整的认知,以下将详细介绍这三个流程的详细内容，具体硬件实现过程见实验内容和实验步骤。

4.3.1 前仿真

前仿真，也称为功能仿真、行为仿真或 RTL 仿真，是指在设计阶段早期对电路的功能和性能进行仿真验证的环节。其主要目的是验证电路在理想环境下的行为和设计构想是否一致，电路功能是否符合规格和设计要求。前仿真不考虑电路门延迟与线延迟，也不涉及实际电路的时序延迟和寄生效应，主要是验证电路与理想情况是否一致。它的特点包括仿真速度快，可以根据需要观察电路输入输出端口和电路内部任一信号和寄存器的波形。

总的来说，前仿真是一个在设计早期阶段用于验证电路功能和性能的重要环节，它帮助设计者在实际制造之前发现并修正潜在问题，从而提高设计的成功率和效率。

4.3.2 门级仿真

门级仿真（Gate Level Simulation，GLS）是数字 IC 设计流程中的一个重要步骤，通常也称为后仿真。它是在综合或实现后生成的门级模型上进行的仿真，主要用于验证设计的功能和时序正确性。以下是门级仿真的原理和应用：

4.3.2.1 门级仿真的原理

1. 门级网表（Netlist）：

门级网表是综合后的电路描述，包含基础单元（如标准单元、SRAM 单元）之间的连线列表。网表中不包含基础单元本身的行为，这些行为通过各自的仿真模型（model）来体现，通常独立于门级网表存在。

门级网表可以是综合后的网表（不带时钟树），也可以是布局布线后的网表（带有时钟树）。

2.时序信息（SDF）：

时序信息通常保存在标准延迟格式（Standard Delay Format，SDF）文件中。SDF 文件包含连线的延迟信息和单元的延迟信息。单元的延迟信息是根据单元时序库（timing library）中定义的 timing arc 从电路图形中间接提取计算出来的。

时序信息的反标（back-annotation）是将实际的时序信息标注到网表中，覆

盖默认的时序信息。这个过程是一个回溯的过程，因此称为反向标注。而在本实验当中，考虑到操作的复杂程度，基本忽略时序反标。

3. 仿真类型：

- 零延时仿真（Zero-Delay Simulation）：不带 SDF 反标的门级仿真，通常在综合后的网表上进行，不考虑时序延迟。
- 时序仿真（Timing Simulation）：带 SDF 反标的门级仿真，通常在布局布线后的网表上进行，考虑实际的时序延迟。

4.3.2.2 门级仿真的应用

1. 双重保险：

门级仿真可以验证综合、布局布线过程中插入的逻辑电路是否正确，防止逻辑不一致的问题。

2. 时序验证：

检查时序约束（SDC）的完备性：防止约束的遗漏。

检查异步电路的时序：STA 工具无法覆盖异步电路的时序问题，门级仿真可以发现这些问题。

精确的时序验证：门级仿真可以发现 STA 工具无法检测到的时序违例，如 false path 和 multi cycle path 等时序例外设置错误。

3. 网表完整性检查：

检查网表的完备性，防止综合、布局布线过程中的意外，如 Macro 使用不正确导致 RTL 与 Netlist 逻辑不一致。

4. 功耗和压降分析：

为后续的功耗（Power）分析和压降（IR Drop）分析提供更准确的波形。

5. 测试向量仿真：

门级网表包含扫描链，可以对 ATPT Pattern 进行仿真，进行 BIST、BISR、DFT 验证。

6. 系统初始化和复位时序验证：

门级仿真可以对系统初始化及复位时序进行较为准确的验证，发现可能存在的 removal 或 recovery 错误。

4.3.2.3 门级仿真的步骤

1. 创建或获取门级网表：

使用综合工具从高层次描述（如 Verilog 源代码）生成门级网表文件。

2. 设置仿真工具：

启动仿真工具（如 ModelSim），创建新项目并添加门级网表文件。

3. 编写测试平台：

创建测试平台文件，定义测试矢量、输入激励和预期的输出结果。测试平台可以使用任何支持的 HDL 编写。

4. 运行仿真：

在仿真工具中打开测试平台文件，编译并运行仿真。仿真工具将执行测试平台中的所有测试矢量，并显示仿真结果。

5. 分析结果：

检查仿真日志，确认输出信号是否符合预期。如果发现错误，需要回到设计阶段修正问题。

通过以上步骤，可以有效地进行门级仿真，验证设计的功能和时序正确性，确保芯片设计的可靠性和性能。

4.3.3 后仿真

后仿真（Post-simulation）是指在系统设计和开发过程完成后，对系统进行的一系列模拟测试活动。它旨在验证系统是否按照预期工作，发现潜在问题，并进行风险评估。在集成电路设计中，后仿真通常指的是在完成了物理设计之后，对电路进行功能验证和性能分析的过程。

4.3.3.1 后仿真的定义：

后仿真也可以称为时序仿真、布局布线后仿真，主要针对布局布线之后的网表，加入时序分析，对功能正确性进行仿真验证。后仿真在有时序信息（器件自身的延迟，传输线上的延时等，与工艺器件有关）的情况下进行，非常接近真实器件运行情况的仿真。

4.3.3.2 后仿真的应用

1. 验证时序和功能：

后仿真不仅验证设计的功能正确性，还验证时序是否满足要求。它确保在引入实际时延之后系统功能仍然正确，避免因时延问题而导致系统时序功能的错误。2. 多时钟域和异步逻辑验证：

后仿真可以验证多时钟域的时序确认（跨时钟域信号的同步处理）和异步逻辑的时序问题，这些是静态时序分析（STA）无法完全覆盖的。

2. DFT 逻辑验证：

后仿真可以验证 DFT（Design for Test）逻辑的插入是否导致功能问题，确保测试向量的有效性。

3. 初始化和复位序列验证：

后仿真可以验证系统初始化和复位序列的正确性，确保系统在启动时能够正确初始化。

4. 功耗分析：

后仿真可以生成 VCD 文件，用于功耗分析，帮助设计者优化电路的功耗性能。

4.3.3.3 后仿真的流程

1.创建或获取门级网表：

使用综合工具从高层次描述（如 Verilog 源代码）生成门级网表文件。

2.提取寄生参数：

使用工具如 Calibre 提取寄生电阻和电容等参数，准备后仿真的模型。

3. 设置仿真工具：

启动仿真工具（如 ModelSim,dve），创建新项目并添加门级网表文件和 SDF 文件。

4. 编写测试平台：

创建测试平台文件，定义测试矢量、输入激励和预期的输出结果。

5. 运行仿真：

在仿真工具中打开测试平台文件，编译并运行仿真。仿真工具将执行测试平台中的所有测试矢量，并显示仿真结果。

6. 分析结果：

检查仿真日志，确认输出信号是否符合预期。如果发现错误，需要回到设计阶段修正问题。

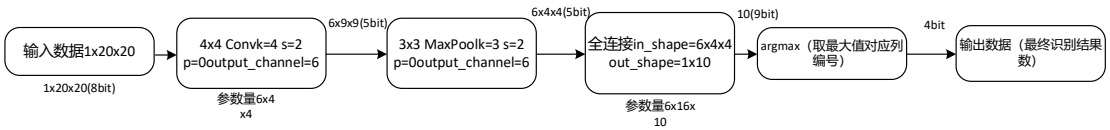
通过以上步骤，可以有效地进行后仿真，验证设计的功能和时序正确性，确保芯片设计的可靠性和性能。

五、实验目的：

针对现在市场对于边缘场景，应用高速高精度图像识别技术的人工智能加速器进行设计，完成图像识别功能，独立完成全部流片过程，最终完成自己的流片。

六、实验内容：

6.1 神经网络运算流程（结构划分）



神经网络结构图

6.1.1 输入数据

由于图像识别加速器对存储的消耗较大，因此为了缓解面积压力，我们根据先针对图片的特征进行压缩，我们发现边缘信息比较少，因此我们最后选择裁剪为 20×20 的照片大小。

6.1.2 卷积层

为了满足全连接层复用，我们定制化设计卷积层大小，我们选用 4×4 的卷积核大小，为了提高准确率我们采用 6 个通道，同时选用步长为 2 的设计降低冗余信息处理。

6.1.3 池化层

为了满足全连接复用，我们采用 3×3 的池化层大小，同样使用 6 个通道。

6.1.4 全连接层

为了保证可以全连接可以连续计算，我们将全连接所有需要处理的数据全部保存下来，方便我们定制化处理，同时全连接我们所有权重全部预先存储下来，这样可以保证直接切换，而因为总共有 10 个数字预测结果，因此一共需要重复计算 10 次类似过程，这会造成巨大的计算开销，为了极大减小面积开销，我们设计复用卷积核模块进行计算。

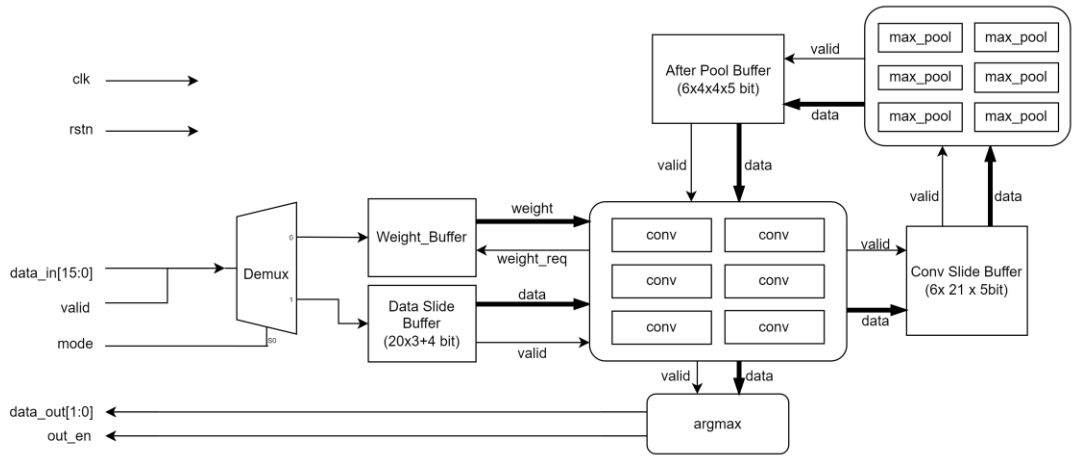
6.1.5Argmax 函数

由于我们的功能是为了实现图像识别，因此我们需要在 10 个数字中判断出一个概率最高的值作为预测结果，而为了减少面积开销，我们采取逐次比较，将每一个数字预测概率依次比较存储，我们只存下概率最大的对应数字，并以此作为最终结果输出。

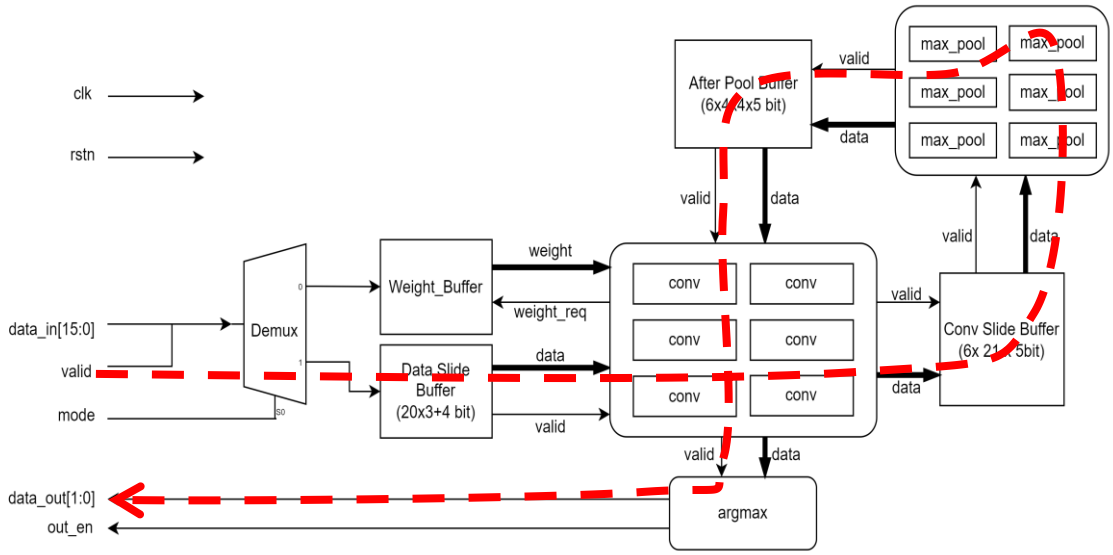
6.1.6 输出函数

我们直接用前面比较最后寄存器里面的数字，转化输出最后的预测数字结果。

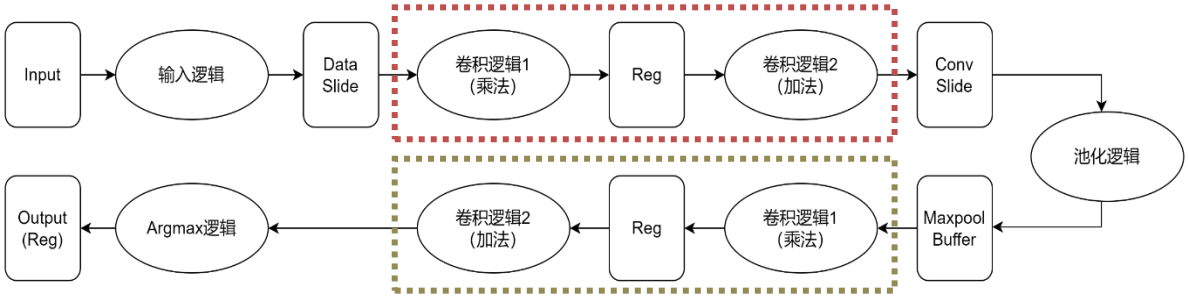
6.2 芯片硬件架构设计



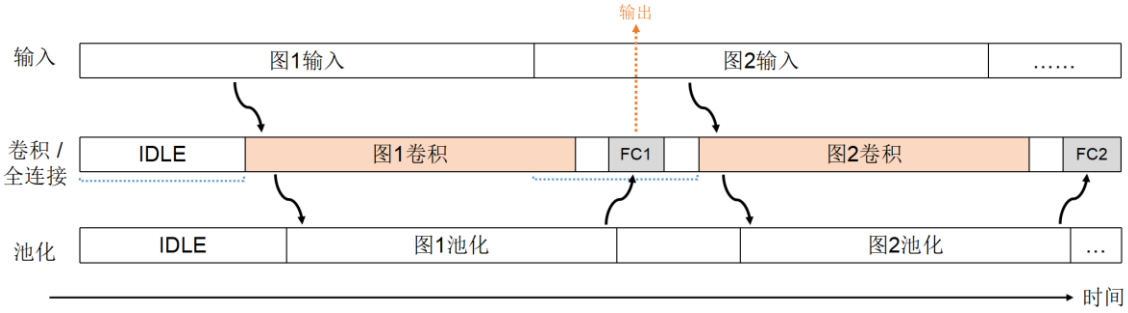
通过架构图不难发现我们的硬件是一个传递 valid 信号的伪流水线结构。同时我们可以分析数据流的过程如下：



图中的红色路径就是我们本次的数据流通过路径，我们会发现数据量两次经过我们的全连接层复用，因而和我们的设计初衷是相一致符合的。我们也可以用另外一种方式呈现我们的数据处理过程：



与此同时，我们的硬件架构也能在空间上实现多张照片连续输入处理，也就是意味着我们的处理总周期基本等于输入周期数（后面也能通过数据验证得到）。



我们可以从时间图中分析硬件的实现过程，首先是第一张照片的输入，然后通过滑窗模块进入第一张照片的卷积模块，同时再通过池化的滑窗，将第一张照片的数据传送到全连接层，送给全连接进行复用计算，而我们会发现，此时图中的卷积层已经不在计算，而且由于我们的卷积滑窗模块需要预先加载一些缓存数据，所以我们需要考虑一段等待周期，卷积核才会重新进行卷积计算，而在该等待周期之内，我们可以同时输入第二张照片的数据，因此照片之间是完全可以做到连续输入的。

七、实验器材（设备、元器件）：

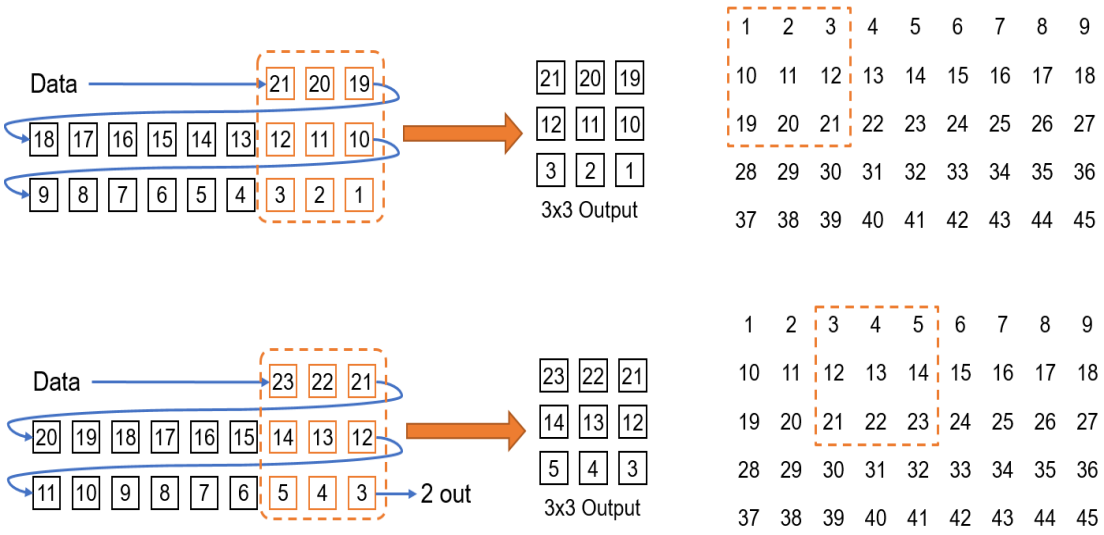
1. 个人电脑
2. 网线
3. 景行锐创应用门户
4. VCS 、DC 、ICC、DVE 等软件
5. Vscode 代码编辑器

八、实验步骤：

8.1 前端设计

8.1.1 卷积层滑窗模块

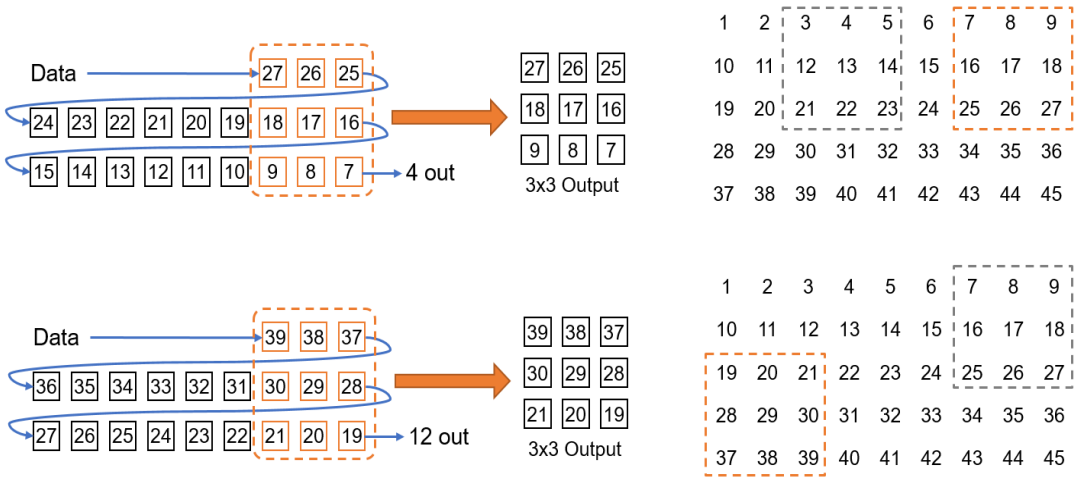
以一个 9x9 输入图像，3x3 卷积核为例，单行卷积计算滑窗移动过程：



首先由于我们的卷积核为 3×3，因此我们的滑窗可以用一个 3 行的寄存器进行替代，而为了简化控制逻辑和降低面积消耗，我们第三行可以只保留三个寄存器，然后将这些寄存器全部串接起来，形成一个 21 位的移位寄存器，但是我们会选择其中特定位置排列下的寄存器输出结果与卷积层权重结合进行卷积计算。那我们是如何移动这个长移位寄存器组呢？

首先假设一次能读入 1 个数据，而我们的卷积核移动步长为 2，由此，当我们将这 21 个寄存器数据全部填满以后，我们在计算每一行输入数据的卷积时，读入 2 个数据以后，就可以开始 1 次卷积核的计算，然后直到第一行的卷积全部计算完成，而由于我们的步长为 2，因此不能马上进行第二行卷积计算，需要进行一个换行操作过程。

换行操作滑窗移动过程：



对于换行操作而言，我们分为两步，第一步首先要把已经计算完的该行数据全部挤出去，同时也需要把第二行的数据全部更新完成，也就是总共需要移动 3+9=12 个步长，这里可能会使得暂存时间积累较长，但是和整体的输入数据相隔时间相比又可以基本忽略不计。

对于我们本次的定制化设计而言，我们会发现，我们输入数据是一个 20×

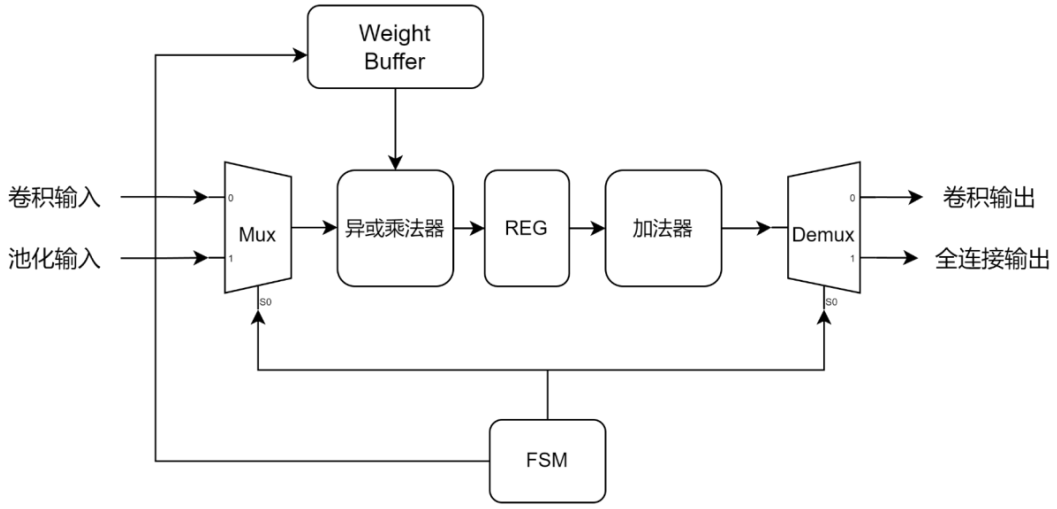
20×8bit 的输入数据集，而第一层卷积核的大小为 4×4，也就是说我们一共是需要 3×20+4=64 个移位寄存器，相较于把所有的权重全部存下来，我们总共需要 64 位的移位寄存器长度。

同时为了减小后面的计算量，我们在一开始对数据做了量化处理，由于数据集的简单性和黑白特性，再利用 BNN 网络的针对对象就是 1bit 数据量，我们将采集的数据在硬件内部量化成 1bit，而后面应经过检测发现准确率略有下降，但仍然处于可接受范围之内。同时由于外部输入数据为 16bit 通道，而我们一个输入数据是 8bit，因此我们最终一次传入两个数据，也就意味着，在卷积计算的时候，能够与数据更新同步进行，仅在换行的时候会被打断，但是输入数据不会停止传入，故利用效率仍然达到很高。

我们利用这样的蛇形串行滑窗，很大程度上减小了输入数据缓存的开销，相较于原始的把所有输入数据全部存下来的思路，我们这种思路实现从 400->64, 降低了 84%的面积成分。同时也能够配合运算结构，实现流水线化处理。

8.1.2 卷积层计算模块

卷积，全连接复用计算模块如下图所示：



从图中我们不难发现，数据的流向会经过一个选择器，这个选择器主要是为了区分当前计算是卷积核计算还是全连接复用计算。而状态选择我们是使用一个状态机进行表示，来表示当前进入什么样的状态，状态机共有三个状态：卷积计算、全连接计算、权重更新。

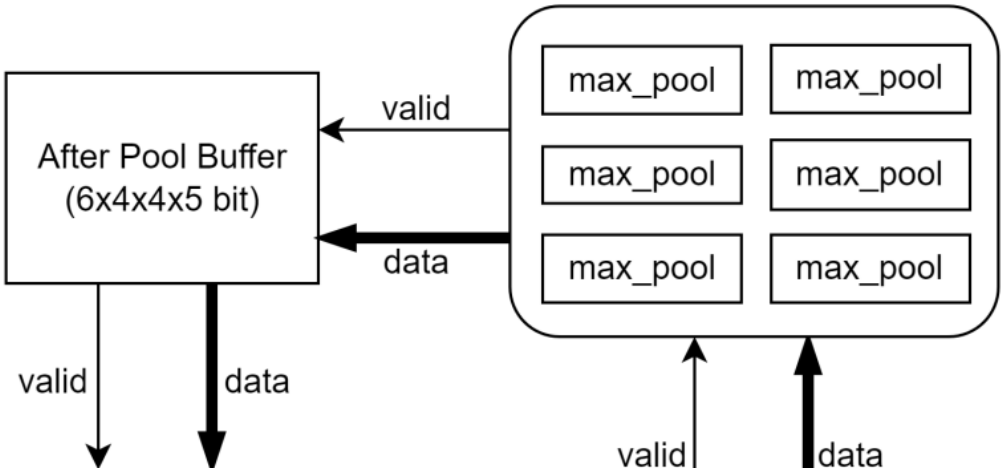
我们的输入数据进入内部计算模块以后，首先会通过状态机的权重更新状态更新权重，以便于将要进行的计算，然后首先会根据权重对于输入数据进行异或乘法器的计算，因为我们使用的是 BNN 网络，所以对于权重的 0 或者 1，要对数据乘以-1 或 1，在硬件中可以表示为异或操作，而通过异或乘法器计算以后，数据流会被第一次暂存起来，实现流水线处理。这是因为我们通过后续的时序分析中，发现该条计算路径是最长的路径，我们选择在中间插入寄存器，面积上消

耗较小的同时也能够提高电路处理的最高速度。

与此同时，由于流水线的操作，我们整个处理结果也会自然而然地顺延一拍输出，需要改变后续的控制逻辑与之配合延一拍。在下个周期开始后，数据会通过加法器计算得到结果答案，而结果又会通过一个选择器，这个选择器也是用与前面相同的状态机控制，当状态机表示当前处于卷积计算状态时，便会将相加的结果送往池化前的滑窗模块，用于后续池化计算，而当状态机表示当前处于全连接计算模块的时候，我们得到的输出结果将会直接输出到 Argmax 模块，用比较存储。

8.1.3 池化层模块：

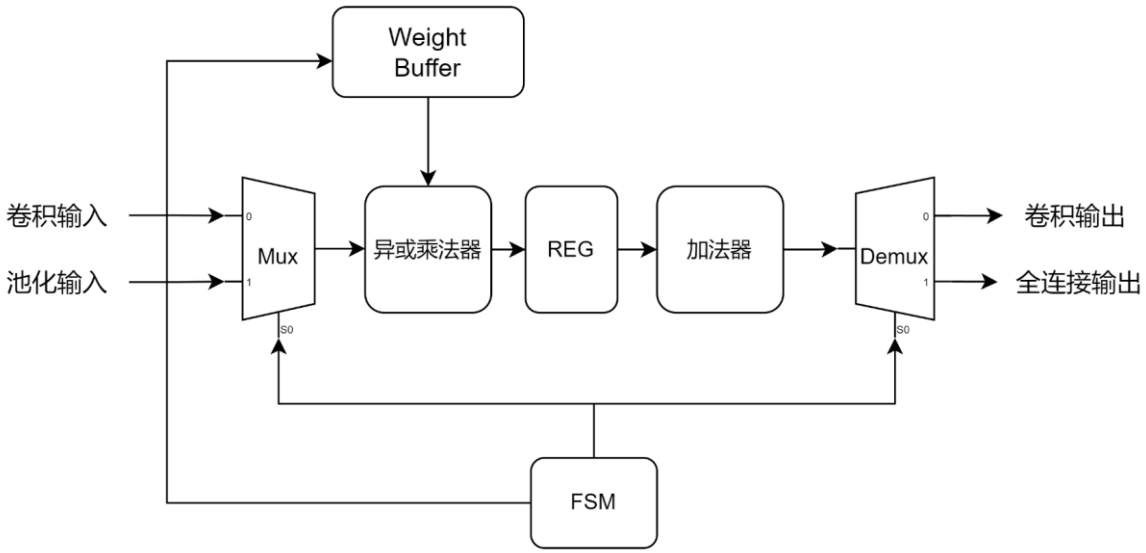
池化结构如下：



我们可以把池化层看成一个简单版的卷积计算，由于根据设计我们的池化层大小恰好与前面所示范的例子匹配，相当于我们将一个 9×9 的图像，通过一个 3×3 的池化层，再将比较得到的结果最后用一个大的存储模块全部存下来用于全连接复用计算。对于这个模块，我们会发现输入数据的缓存也能实现从 81->21，降低了 74%的面积占比，效果较明显。

8.1.4 全连接复用模块：

卷积，全连接复用计算模块：



分析过程同卷积层计算中已阐述，不再详述，值得一提的是我们的全连接层权重一共有十层，所以参数量非常庞大，而对于不同的数字判断，对应的权重也有所不同，因此造成了控制逻辑的优化显得至关重要。

8.1.5 Argmax 函数模块：

这个模块也较为简单，为了降低面积的消耗，我们只选择使用一个寄存器用于存储每次比较后最大概率的数值结果，我们每一次全连接计算都会得到一个概率值，然后我们每次得到一个概率值，会操作其与上次概率值进行比较，最后将得到的最大概率对应数值作为输出结果存储下来，进行输出。

8.1.6 输出预测结果：

为了减少 IO 端口占用，我们用两个输出端口分两步输出 4bit 结果，速度上会慢一拍，但是结构上仍然不影响整体的流水线结构。

8.2 撰写测试程序验证代码

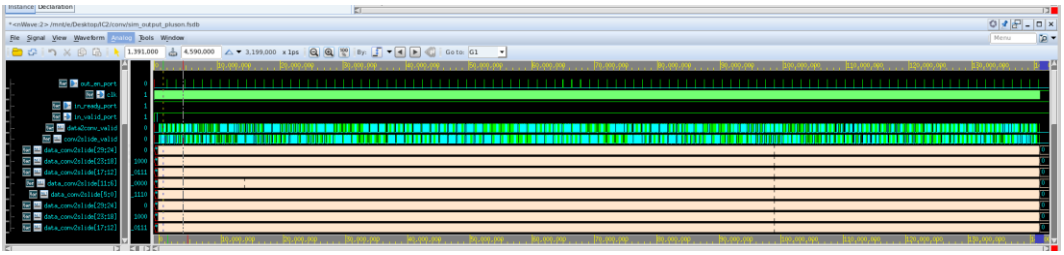
我们最终需要针对整个 MNIST 数据集进行预测，共计 1 万张图片预测结果。通过 testbench 按照顺序输入一张图片的各个像素的数据，并将输出与标准答案相比对，判断预测结果是否合理，并不断重复这个过程。我们也可以针对一定数量的图片进行抽样预测，观察其准确率是否处于合理范围之内。

8.3 前仿真（功能仿真）

```
*Verdi* FSDB WARNING: The FSDB file already exists. Overwriting the FSDB file may crash the
 file.
*Verdi* : Create FSDB file 'sim_output_pluson.fsdb'
*Verdi* : Begin traversing the scope (top_tb), layer (0).
*Verdi* : End of traversing.
*Verdi* : Begin traversing the MDAs, layer (0).
*Verdi* : Enable +mda and +packedmda dumping.
*Verdi* : End of traversing the MDAs.
Mismatch at test 11: expected   6, got   5,at             17476000
Mismatch at test 33: expected   4, got   0,at             48276000
Mismatch at test 65: expected   4, got   1,at             93076000
Mismatch at test 75: expected   7, got   1,at            107076000
Mismatch at test 87: expected   3, got   5,at            123876000
Simulation finished  100,error:   5
$finish called from file "./rtl/top_tb_weight.v", line 135.
$finish at simulation time           140830000
          V C S   S i m u l a t i o n   R e p o r t
Time: 140830000 ps
CPU Time:      1.950 seconds;       Data structure size:   0.1Mb
Sat Jan 11 20:39:28 2025
20:39:28 (snpslmd) IN: "VCSRuntime_Net" fmq03@LAPTOP-PETERFMQ  [snps_checkout_1736599166]
```

代码前仿预测率与预期一致（100 张图片）！

前仿中我们得到的波形如下：



8.4 门级仿真

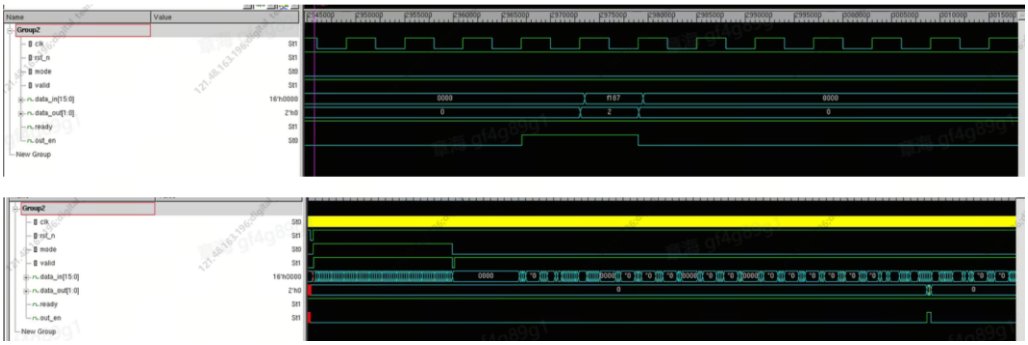
使用网表文件和工艺库的仿真文件进行仿真，验证逻辑综合后的设计正确性，可见预测准确率与预期一致（100 张图片）。

```
  Please verify that the first argument to $readmem is a file that exists with
  proper permissions.

VCD+ Writer O-2018.09-SP2_Full64 Copyright (c) 1991-2018 by Synopsys Inc.
Mismatch at test 11: expected   6, got   5,at             14979000
Mismatch at test 33: expected   4, got   0,at             41379000
Mismatch at test 65: expected   4, got   1,at             79779000
Mismatch at test 75: expected   7, got   1,at             91779000
Mismatch at test 87: expected   3, got   5,at            106179000
Simulation finished  100,error:   5
$finish called from file "./rtl/top_tb_weight.v", line 126.
$finish at simulation time           120761500
          V C S   S i m u l a t i o n   R e p o r t
Time: 120761500 ps
CPU Time:      9.020 seconds;       Data structure size:   4.4Mb
Fri Dec 27 22:10:32 2024
[digital_team04_2@linuxapp6_conv]$
```

```
Mismatch at test 9883: expected   5, got   1,at          13838276000
Mismatch at test 9888: expected   6, got   0,at          13845276000
Mismatch at test 9900: expected   8, got   1,at          13862076000
Mismatch at test 9904: expected   2, got   3,at          13867676000
Mismatch at test 9905: expected   3, got   9,at          13869076000
Mismatch at test 9906: expected   4, got   2,at          13870476000
Mismatch at test 9913: expected   2, got   8,at          13880276000
Mismatch at test 9926: expected   8, got   1,at          13898476000
Mismatch at test 9944: expected   3, got   8,at          13923676000
Mismatch at test 9947: expected   4, got   0,at          13927876000
Mismatch at test 9959: expected   8, got   1,at          13944676000
Mismatch at test 9970: expected   5, got   3,at          13960076000
Mismatch at test 9982: expected   5, got   6,at          13976876000
Simulation finished 10000,error:1051
$finish called from file "./rtl/top_tb_weight.v", line 126.
$finish at simulation time          14000830000
           V C S   S i m u l a t i o n   R e p o r t
Time: 14000830000 ps
```

代码门仿全部通过（1 万张），预测率与软件预期一致，证明了逻辑综合前后设计的一致性!!

在DVE 中查看的仿真波形如下：



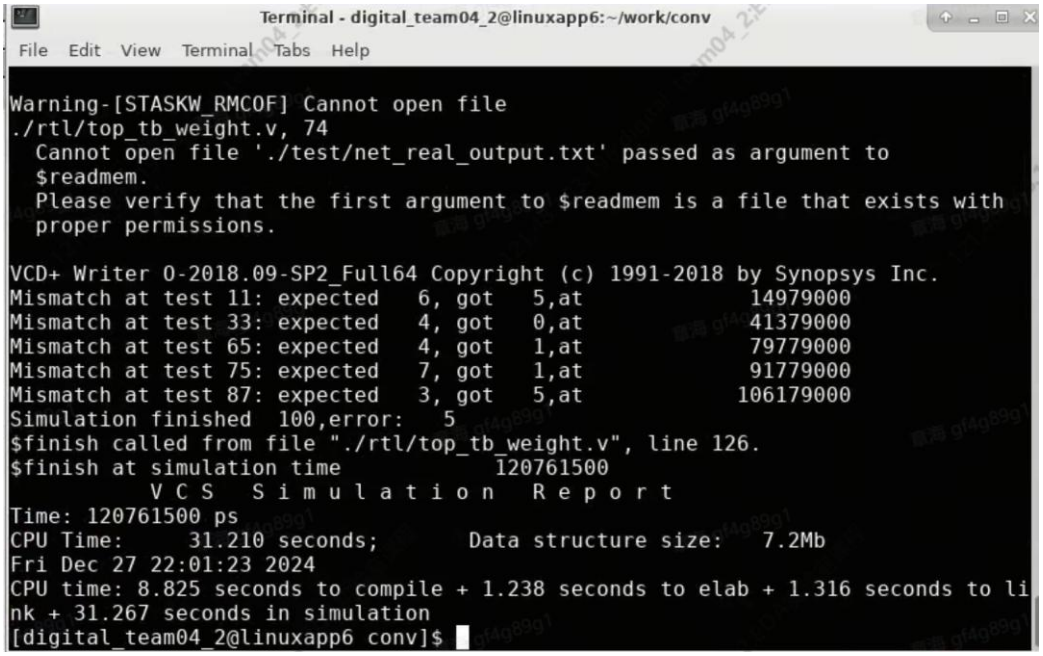
符合预期结果！！

8.5 后仿真：

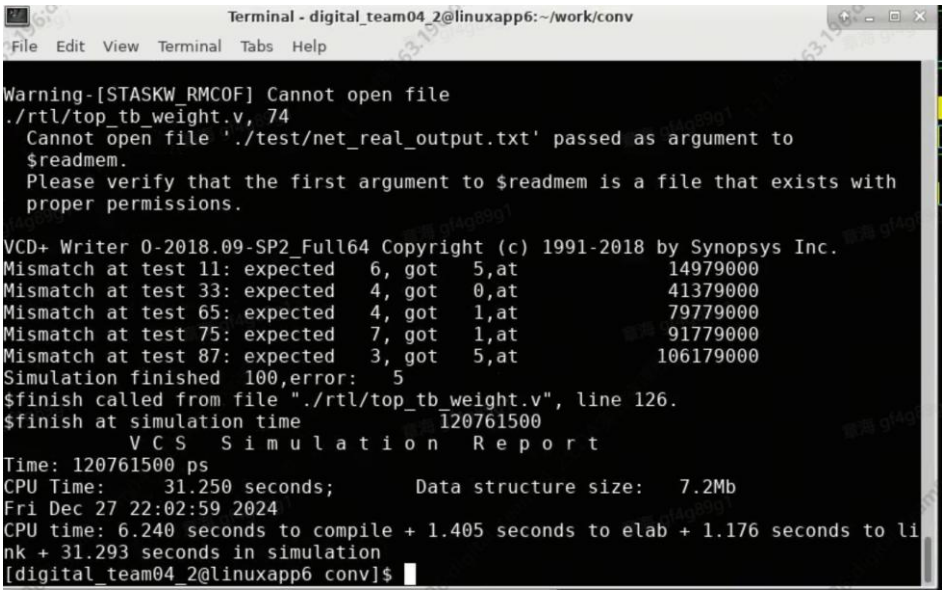
在verilog 设计进行布局布线之后，提取出寄生参数，得到 sdf 时序反标文件，并将其和工艺库仿真文件、TB 文件一起中进行仿真。通过这个过程来验证带物理参数的情况下电路是否能正常工作。具体仿真结果如图：

100 个抽样图片测试结果仿真验证：

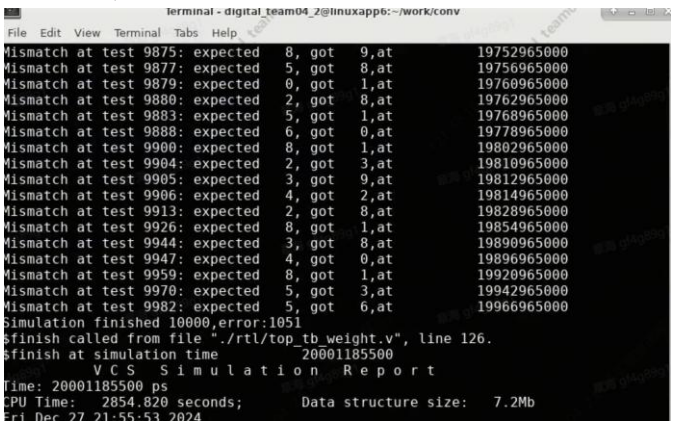
Min 路径：



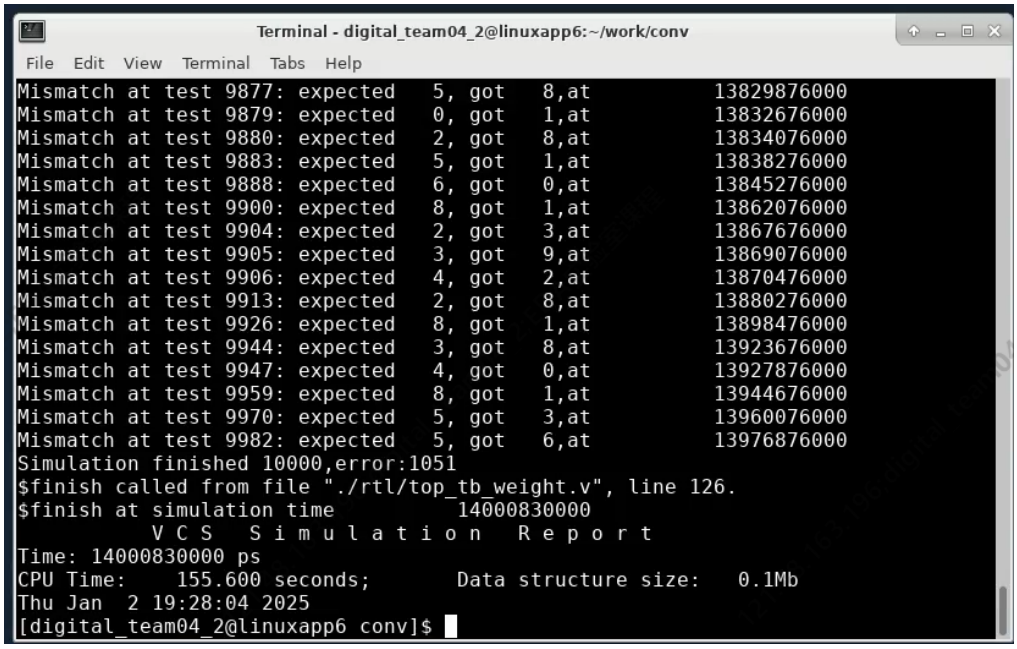
Max 路径：



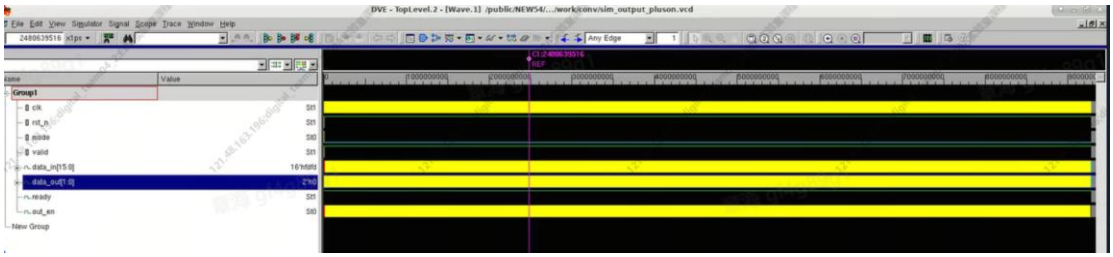
10000 个结果图（10ns 周期测试）：



10000 个结果图（7ns 周期测试）：



波形图验证（10ns 周期测试）：



综上所述，我们不难发现无论是 100 个抽样调查测试，Min 路径还是 Max 路径，又或者 10000 张照片集，7ns 又或是 10ns 最终得到的预测率与前仿结果完全一致，也证明了后仿完全通过！！

九、实验数据及结果分析：

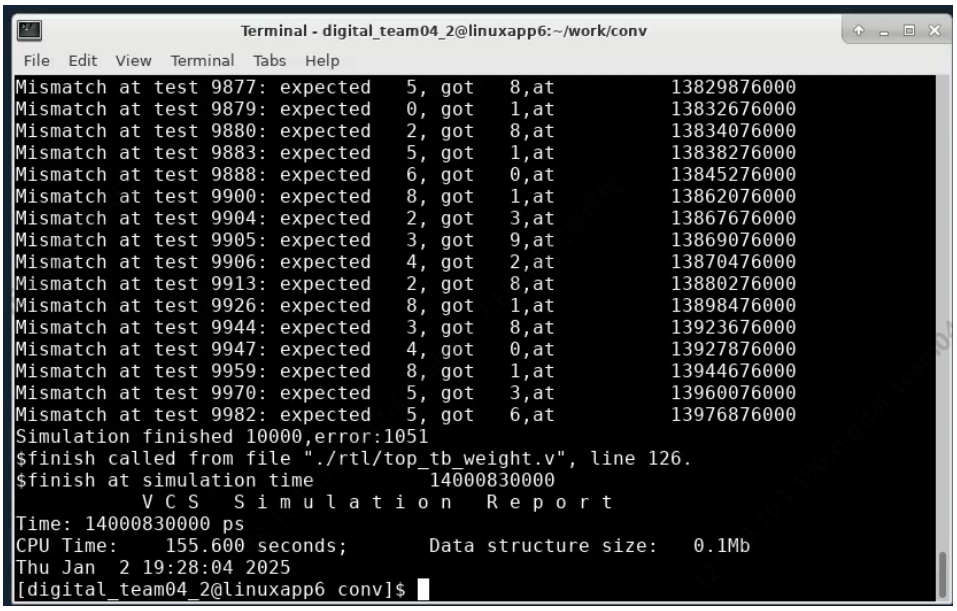
9.1 预测准确率对比

软件模型识别准确率：在整个（一万张）MNIST 测试集（手写数字识别数据集）上识别准确率达 91%。

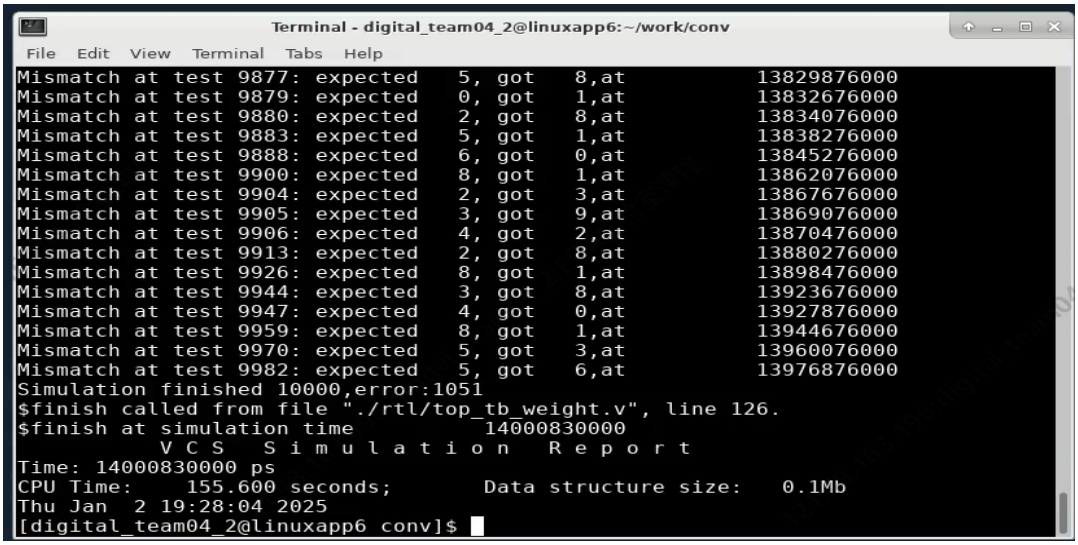
```python
with torch.no_grad():
    for i,data in enumerate(testloader):
        images, labels = data
        images=images[:,:, crop_size:-crop_size, crop_size:-crop_size]
        images[0,0,:,:]= (images[0,0,:,:]>0.5)
        images = images.to(device)
        labels = labels.to(device)
        outputs,x_max[i] = net_predict(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        ans_predict=torch.cat((ans_predict,predicted))
        if i>10000/128: #一个batch 128
            break
ans_predict=ans_predict.view(-1)
print('Accuracy of the network on the 10000 test images: %d %%' % (100 * correct / total))
print(correct,total)
print(ans_predict)

# 文件路径
file_path = "net_real_output.txt"

# 写入文件
with open(file_path, "w") as f:
    # 将张量的每个元素写入文件，按行保存
    for value in ans_predict:
        f.write(f"{value.item()}\n")  # 使用 .item() 提取标量值
```

```
Accuracy of the network on the 10000 test images: 91 %
9123 10000
```

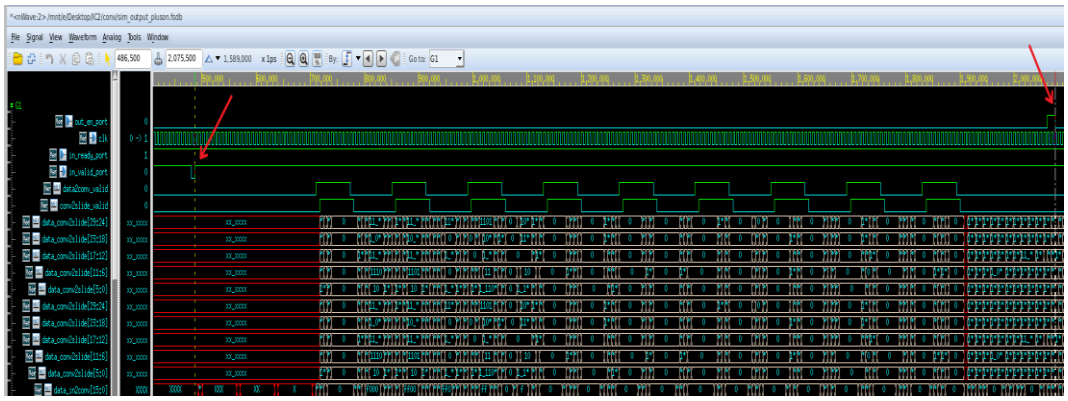
硬件架构识别准确率：在整个（一万张）MNIST 测试集（手写数字识别数据集）上识别准确率约为 **90%。**



9.2 吞吐量和图片处理周期



1 万张图片处理总时间图



单张图片处理结果图

综上，我们可以得到以下数据：

- 一万张图片处理总时间：14000830000ps（**周期 7ns**）
- 平均单张图片处理周期：**200cycle**
- 时钟频率：143MHz
- 单张图片处理周期：227cycle
- 吞吐量=数据量/处理时间=（20*20*8bit） / (200*7ns)=**2.3Gbps**

我们将本次设计的所有用到指标进行罗列对比：

参数指标	自拟设计	原题目要求
输入数据量	20*20*8bit=3200bit	11*11*8bit=968bit
参数量（权重）	11*4*4*6*1bit=1056bit	2*3*3*3*8bit=432bit
识别准确率	90%	—
核利用率	79.19%	—
平均处理周期数	200cycle	—
吞吐量	2.3Gbps	—
时钟频率	143MHz	>50MHz
面积	850um*850um	<=850um*850um
功耗	14mw（143MHz）	—

从最后结果上看，我们的吞吐量大小在本届设计中居于领先地位，基本在该工艺条件和面积限制下实现了高速的要求，同时仍然能够保持 **90%**的硬件准确率，达到了设计初衷，完成了**高速高准确率小面积**的**图像识别加速器设计**。

十、实验结论：

在合理的分工下，合理安排时序问题，通过对时序约束的来回检查，我们最终成功设计出符合要求的成果。

十一、总结及心得体会：

（1）认识到代码规范的重要性，更加理解代码背后综合出的电路是什么样的，提升了coding 能力。

在初期设计时，我们的代码仿真遇到了较多问题，会发现都是由于控制逻辑之间关系没有理清楚，可能就是相差一两拍的数据错位，而导致了预测结果千差万别，而这些也和良好的书写习惯息息相关。

（2）初步认识VCS 、DVE 、DC 、ICC 和Cadence 软件的功能和使用方法。

（3）对于设计敢于去创新和尝试

在选择自拟题的时候其实内心也十分坎坷，不知道到底能不能做出来，也经历了非常多的失败，因为图像识别设计相对来说规模还是庞大了很多，我们一直

针对软件算法和硬件架构做相应地优化，一点点的改进和尝试，某种意义上来说是自己摸着墙学会了走路，但是收获也是非常巨大的，对于神经网络的深层次应用我们更为了解，对于 BNN 网络的特点，我们也更加熟悉 ，在这个过程我们也遇到了很多问题，比如存储面积庞大，参数量过大，是否能够存下所有的权重等至关重要的问题，但是最后这些问题都得到了解决，在保持较高准确率的同时仍然能够实现较高的速率。

（4）对于 IC 验证的了解大大提升

我们对于程序的验证十分关键的，因为自拟题目是没有老师给的平台可以帮忙辅助验证，因此我们验证的 TB 至关重要，有的时候出现逻辑混乱的问题也不一定存在于代码上，更在于 TB 验证与实际输入数据的协同性。

（5）更深入地了解软件算法与硬件协同的关系以及实现的主要问题在哪些方面。

（6）项目小组成员的明确分工很重要。

前端设计人员因为需要对 verilog 设计进行不断地迭代替换，没有经力和时间去跟上老师对实验步骤的讲解，因此需要对组内成员进行明确地分工。由于我们在一开始就分好了工，后来在前端设计迭代的过程中，也有同学跟上老师的进度，并在最后帮助我们跑通了后端流程。

十二、对本实验过程及方法、手段的改进建议：

可以采用更好的工艺以及更宽容的面积限制，我们可以有更多拓展的空间与想法的实施验证，可以实现更加功能完整的芯片设计。

附录：具体代码

（一）软件算法：

（1）训练需要的函数与类(class)定义

```
def SigmoidFunc(x, alpha=4.):   # 带参数的Sigmoid函数
    return torch.sigmoid(alpha * x)

def SigmoidGFunc(x, alpha=4.):   # 带参数Sigmoid函数的导数
    fx = torch.sigmoid(alpha * x)
    return alpha * fx * (1. - fx)

def BinaryForFunc(x):   # 权值量化前向函数
    return torch.sign(x)

def BinaryWBackFunc(x, alpha=4.):   # 权值量化反向梯度函数
    return 2 * SigmoidGFunc(x, alpha)

class BinaryWeight(torch.autograd.Function):
    @staticmethod
    def forward(ctx, input):
        ctx.save_for_backward(input)
        return BinaryForFunc(input)

    @staticmethod
    def backward(ctx, grad_output):
        input, = ctx.saved_tensors
        return grad_output * BinaryWBackFunc(input, alpha=4.)

class BinaryLinear(nn.Linear):
    def __init__(self, in_features, out_features, bias=False):
        super(BinaryLinear, self).__init__(in_features, out_features, bias)
        #self.alpha = nn.Parameter(torch.tensor(1.0))  # 可学习的缩放因子

    def forward(self, x):
        w = self.weight
        #bw = BinaryWeight.apply(w)
        scaling_factor = torch.mean(torch.mean(abs(w),dim=1,keepdim=True),dim=0,keepdim=True)
        scaling_factor = scaling_factor.detach()
        bw = scaling_factor * BinaryWeight.apply(w)
        bw = BinaryWeight.apply(bw)
        return F.linear(x, bw, self.bias)

class BinaryConv2d(nn.Conv2d):
    def __init__(self, in_channels, out_channels, kernel_size, stride=1,
                 padding=0, dilation=1, groups=1, bias=False):
        super(BinaryConv2d, self).__init__(in_channels, out_channels,
            kernel_size, stride, padding, dilation, groups, bias)
        #self.alpha = nn.Parameter(torch.tensor(1.0))  # 可学习的缩放因子

    def forward(self, x):
        w = self.weight
        #bw = BinaryWeight.apply(w)
        scaling_factor = torch.mean(torch.mean(torch.mean(torch.mean(abs(w),dim=3,keepdim=True),dim=2,keepdim=True),
                                               dim=1,keepdim=True),dim=0,keepdim=True)
        scaling_factor = scaling_factor.detach()
        bw = scaling_factor * BinaryWeight.apply(w)
        bw = BinaryWeight.apply(bw)
        return F.conv2d(x, bw, self.bias, self.stride,
                        self.padding, self.dilation, self.groups)
```

（2）数据集加载过程

```python
#加载MNIST数据集
transform = torchvision.transforms.Compose([torchvision.transforms.ToTensor()])
train_dataset = torchvision.datasets.MNIST(root='D:\Python_code\IC2', train=True, transform=transform, download=True)
test_dataset = torchvision.datasets.MNIST(root='D:\Python_code\IC2', train=False, transform=transform, download=True)
print(len(train_dataset),len(test_dataset))

#加载MNIST数据集的第一张图片
image, label = train_dataset[1]
#将图片用matplotlib画出来
import matplotlib.pyplot as plt
# 假设 train_dataset 是一个包含 (image, label) 的数据集
fig = plt.figure(figsize=(15, 5))  # 创建图像窗口并设置大小
# 绘制 10 张图像
crop_size=4
for i in range(1):
    image, label = test_dataset[-i]
    image_cropped = image[:, crop_size:-crop_size, crop_size:-crop_size]
    ax = plt.subplot(1, 10, i + 1)  # 创建 1 行 10 列的子图
    ax.imshow(image_cropped.squeeze().numpy(), cmap="gray")  # 绘制图像
    ax.axis("off")  # 隐藏坐标轴
plt.tight_layout()  # 自动调整布局
plt.show()  # 最后统一显示
print(image_cropped.shape)

#讲image转化为int8类型
image_cropped = image_cropped * 255
np.set_printoptions(linewidth=700)  # 设置行宽
print(image_cropped.int(),end=" ")
print(label)
#with open("image_cropped.txt", "w") as f:
    #for row in image_cropped:
    #   f.write(" ".join(map(str, row.tolist())) + "\n")
```

（3）训练与预测网络类的定义

```python
class LeNet(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        ch_in=6
        ch_out=10
        self.conv1 = BinaryConv2d(1,ch_in, kernel_size=4, stride=2, padding=0, bias=False)
        self.pool1 = nn.MaxPool2d(kernel_size=3, stride=2)
        #self.relu1 = nn.ReLU()
        #self.conv2 = BinaryConv2d(6, 12, kernel_size=5, stride=1, padding=0, bias=False)
        #self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        #self.relu2 = nn.ReLU()
        self.fc1 = BinaryLinear(ch_in*4*4, 10, bias=False)
        #self.softmax=nn.Softmax(dim=1)
        #self.conv_ch = BinaryConv2d(ch_in,ch_out, kernel_size=4, stride=1, padding=0, bias=False)
        #self.conv_num = BinaryConv2d(ch_out,10, kernel_size=1, stride=1, padding=0, bias=False)
    def forward(self, x):
        #x=F.pad(x,(0,1,0,1))
        #x = x[:, crop_size+1:-crop_size, crop_size+1:-crop_size]
        x = self.conv1(x)
        #x = self.relu1(x)
        #x = nbit_quantizer(x, n_bits_signed=5)
        #x = self.relu1(x)
        #x=F.pad(x,(0,1,0,1))
        x = self.pool1(x)
        #x = nbit_quantizer(x, n_bits_signed=5)
        #x = x.view(x.size(0), -1)
        #x = self.conv_ch(x)
        #x = self.conv_num(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        return x
```

```python
class LeNet_predict(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        ch_in=6
        ch_out=10
        self.conv1 = BinaryConv2d(1,ch_in, kernel_size=4, stride=2, padding=0, bias=False)
        self.pool1 = nn.MaxPool2d(kernel_size=3, stride=2)
        self.fc1 = BinaryLinear(ch_in*4*4, 10, bias=False)
        self.conv_ch = BinaryConv2d(ch_in,ch_out, kernel_size=4, stride=1, padding=0, bias=False)
        self.conv_num = BinaryConv2d(ch_out,10, kernel_size=1, stride=1, padding=0, bias=False)
    def forward(self, x):
        x = self.conv1(x)
        x_max=torch.min(x)
        x = torch.where(x>0,x-(x//16)*16,x-(x//-16)*16)
        #x = nbit_quantizer(x, n_bits_signed=5)
        x = self.pool1(x)
        #x = x-x%2
        #x = nbit_quantizer(x, n_bits_signed=5)
        #x = self.conv_ch(x)
        #x = self.conv_num(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        return x,x_max
```

（4）训练过程

```
from tqdm import tqdm
np.int = int
classes = ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9')
criterion = nn.CrossEntropyLoss()
lambda_reg = 0.01
def loss_fn(output, target, weight, binary_weight):
    task_loss = criterion(output, target)
    quant_loss = sum(lambda_reg * torch.norm(w - bw, p=2) for w,bw in zip(weight,binary_weight))
    return task_loss + quant_loss

optimizer = optim.Adam(net.parameters(), lr=0.002)
EPOCH = 20
device = torch.device('cpu')
net.to(device)
crop_size=4
target_acc=0.91

for epoch in range(EPOCH):
    net.train()
    for i ,data in enumerate(dataloader):
        inputs, labels = data
        inputs=inputs[:,:, crop_size:-crop_size, crop_size:-crop_size]
        inputs[0,0,:,:]=(inputs[0,0,:,:]>0.5)
        inputs = inputs.to(device)
        labels = labels.to(device)
        outputs = net(inputs)
        binary_weights = [layer.weight.sign() for layer in [net.conv1, net.fc1]]
        weights = [layer.weight for layer in [net.conv1, net.fc1]]
        total_loss = loss_fn(outputs, labels, weights, binary_weights)
        loss = criterion(outputs, labels)
        optimizer.zero_grad()
        #loss.backward()
        total_loss.backward()
        optimizer.step()
        functional.reset_net(net)
        #if i % 400 == 0:
            #print('[%d, %5d] loss: %.3f' % (epoch + 1, i + 1, loss.item()))
    net.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for i,data in enumerate(testloader):
            images, labels = data
            images=images[:,:, crop_size:-crop_size, crop_size:-crop_size]
            images[0,0,:,:]=(images[0,0,:,:]>0.5)
            images = images.to(device)
            labels = labels.to(device)
            outputs = net(images)
            _, predicted = torch.max(outputs, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    print('Accuracy of the network on the 10000 test images: %d %%' % (100 * correct / total))
    if correct/total>target_acc:
        break
```

（5）推理预测过程

```
net_predict=LeNet_predict()
#将权重加载，二值化，保存为另外一个pth文件
'''net.load_state_dict(torch.load('D:\Python_code\IC2_python\weight_lenet3.pth',weights_only=True))
for name, param in net.named_parameters():
    if 'weight' in name:
        param.data = BinaryWeight.apply(param.data)
torch.save(net.state_dict(), 'D:\Python_code\IC2_python\weight_binary_lenet3.pth')'''
#加载二值化后的权重
net_predict.load_state_dict(torch.load('D:\Python_code\IC2_python\weight_binary_lenet2.pth',weights_only=True))
net_predict.to(device)
net_predict.eval()
correct = 0
total = 0
x_max=torch.zeros(10000)
ans_predict=torch.empty(0)
with torch.no_grad():
    for i,data in enumerate(testloader):
        images, labels = data
        images=images[:,:, crop_size:-crop_size, crop_size:-crop_size]
        images[0,0,:,:]= (images[0,0,:,:]>0.5)
        images = images.to(device)
        labels = labels.to(device)
        outputs,x_max[i] = net_predict(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        ans_predict=torch.cat((ans_predict,predicted))
        if i>10000/128: #一个batch 128
            break
ans_predict=ans_predict.view(-1)
print('Accuracy of the network on the 10000 test images: %d %%' % (100 * correct / total))
print(correct,total)
print(ans_predict)

# 文件路径
file_path = "net_real_output.txt"

# 写入文件
with open(file_path, "w") as f:
    # 将张量的每个元素写入文件，按行保存
    for value in ans_predict:
        f.write(f"{value.item()}\n")  # 使用 .item() 提取标量值
```

```
Accuracy of the network on the 10000 test images: 91 %
9123 10000
```

（6）网络权重导出过程

```
#将网络第一层的权重导出
conv1_weight = net_predict.conv1.weight.data
#print(conv1_weight)
ch_in=6
conv1_weight_each_ch=torch.zeros(ch_in,16)
for i in range(ch_in):
    conv1_weight_each_ch[i,:] = conv1_weight[i,:,:].view(-1)
#print(conv1_weight_each_ch.shape)
conv1_weight_int = conv1_weight_each_ch.int()
#print(conv1_weight_int.shape)
#将第一层的权重导出为txt文件
with open('D:\Python_code\IC2_python\cov1_weight.txt', 'w') as f:
    for i in range(ch_in):
        for j in range(16):
            if conv1_weight_int[i,j] < 0:
                conv1_weight_int[i,j]=0             #-1以0形式储存，+1以1形式储存
            f.write(format(conv1_weight_int[i,j], 'b')+('  ') + (j==15)*'\n')
print(conv1_weight_int.shape)


#将网络第二层的权重导出
fc1_weight = net_predict.fc1.weight.data
print(fc1_weight.shape)
fc1_weight_ch_num=torch.zeros(10,ch_in,4*4)
with open('D:\Python_code\IC2_python/fc1_weight.txt', 'w') as f:
    for k in range(10):
        for m in range(ch_in):
            fc1_weight_ch_num[k,m,:] = fc1_weight[k,4*4*m:4*4*(m+1)]
        fc1_weight_ch_num_int=fc1_weight_ch_num.int()
        #with open('D:\Python_code\IC2/fc1_weight.txt', 'w') as f:
        for i in range(ch_in):
            for j in range(16):
                if fc1_weight_ch_num_int[k,i,j] < 0:
                    fc1_weight_ch_num_int[k,i,j]=0             #-1以0形式储存，+1以1形式储存
                f.write(format(fc1_weight_ch_num_int[k,i,j], 'b')+('  ') + (j==15)*'\n') #每个通道换一行，每个数字换三行
print(fc1_weight_ch_num_int.shape)
```

（7）测试集数据导出过程

```
import os
import math
#按行将image转化为txt文件
'''
pad_zero_index=torch.zeros(9,8)
pad_zero_index[0,:]=torch.tensor([61,65,67,69,71,73,75,79])
pad_zero_index[1,:]=torch.tensor([101,105,107,109,111,113,115,119])
pad_zero_index[2,:]=torch.tensor([141,145,147,149,151,153,155,159])
pad_zero_index[3,:]=torch.tensor([181,185,187,189,191,193,195,199])
pad_zero_index[4,:]=torch.tensor([221,225,227,229,231,233,235,239])
pad_zero_index[5,:]=torch.tensor([261,265,267,269,271,273,275,279])
pad_zero_index[6,:]=torch.tensor([301,305,307,309,311,313,315,319])
pad_zero_index[7,:]=torch.tensor([341,345,347,349,351,353,355,359])
pad_zero_index[8,:]=torch.tensor([381,385,387,389,391,393,395,399])'''
#print(pad_zero_index)
base_dir_pre="D:\Python_code\IC2_python\image_matrix\image_test_pre"
base_dir="D:\Python_code\IC2_python\image_matrix\image_test"
base_dir_label="D:\Python_code\IC2_python\image_matrix\image_label"
for m in range(10000):
    image, label = test_dataset[m]
    image_cropped = image[:, crop_size:-crop_size, crop_size:-crop_size]*255
    image_cropped_int=image_cropped.int()
    image_test_pre_path=os.path.join(base_dir_pre, f"MNIST_images_{m}_pre.txt")
    image_test_path=os.path.join(base_dir, f"MNIST_images_{m}.txt")
    label_test_path=os.path.join(base_dir_label, f"MNIST_label_{m}.txt")
    with open(label_test_path, 'w') as f:
        f.write(f"{label}"+'\n')
    with open(image_test_pre_path, 'w') as f:
        for i in range(20):
            for j in range(20):
                flag=((i*20+((math.floor(j/4)+1)*4-j%4))==pad_zero_index).any()  #flag=1说明需要在后面补零
                flag=0
                f.write(format(image_cropped_int[0, i, (math.floor(j/2)+1)*2-j%2-1].item(), '08b') + (flag==1)*"")   #一行存四个数据  4 3 2 1 , 8 7 6 5 , 12 11 10 9
                #f.write(f"{(i*20+((math.floor(j/2)+1)*2-j%2)):0{8}d}"+(flag==1)*"000000")

    # 读取原始文件并重新组织内容
    input_file = image_test_pre_path  # 原始文件
    output_file = image_test_path  # 输出文件

    # 读取原始文件内容
    with open(input_file, "r") as infile:
        data = infile.read()  # 读取整个文件内容为字符串

    # 计算需要的行数
    line_length = 16  # 每行16个字符
    lines = [data[i:i + line_length] for i in range(0, len(data)+1, line_length)]

    # 将处理后的内容写入新文件
    with open(output_file, "w") as outfile:
        outfile.write("\n".join(lines))

    # 文件路径
    file_path = image_test_path

    # 要交换的两行索引（从 0 开始计数）
    line1_index = [17,19,21,23, 31,33,35,37, 45,47,49,51, 59,61,63,65, 73,75,77,79, 87,89,91,93, 101,103,105,107, 115,117,119,121, 129,131,133,135]
    line2_index=line1_index
    line1_index=[x - 1 for x in line1_index]


    # 读取文件内容
    with open(file_path, "r") as f:
        lines = f.readlines()  # 按行读取文件内容到列表

    for i in range(36):
        lines[line1_index[i]], lines[line2_index[i]] = lines[line2_index[i]], lines[line1_index[i]]
        # 写回文件
        with open(file_path, "w") as f:
            f.writelines(lines)
```

（8）Debug 使用的 watch_net 定义

```
def to_6bit_twos_complement(value):
    """
    将整数转换为 6 位补码表示。
    :param value: 要转换的整数（范围 -32 到 31）
    :return: 6 位补码二进制字符串
    """
    if value < -32 or value > 31:
        raise ValueError("整数超出 6 位补码表示范围 [-32, 31]")

    # 对负数进行补码处理
    if value < 0:
        value = (1 << 6) + value  # 将负数转换为其补码形式

    # 转换为二进制字符串并截取后 6 位
    return f"{value:06b}"  # 格式化为 6 位二进制字符串

# 测试
#print(to_6bit_twos_complement(-15))   # 输出: 110001

class watch_leNet(nn.Module):
    def __init__(self, num_classes=10):
        super().__init__()
        ch_in=6
        ch_out=10
        self.conv1 = BinaryConv2d(1,ch_in, kernel_size=4, stride=2, padding=0, bias=False)
        self.pool1 = nn.MaxPool2d(kernel_size=3, stride=2)
        self.fc1 = BinaryLinear(ch_in*4*4, 10, bias=False)
        self.conv_ch = BinaryConv2d(ch_in,ch_out, kernel_size=4, stride=1, padding=0, bias=False)
        self.conv_num = BinaryConv2d(ch_out,10, kernel_size=1, stride=1, padding=0, bias=False)
    def forward(self, x):
        x = self.conv1(x)
        ch_in=6
        #print(self.conv1.weight)
        #print(x)
        #print(x.int())
        x_int_cov=x.int()
        with open('D:\Python_code\IC2_python\conv1_result.txt', 'w') as f:
            for k in range(ch_in):
                for i in range(9):
                    for j in range(9):
                        conv1_6b_2comp=to_6bit_twos_complement(x_int_cov[0,k-1,i,j])
                        f.write(f"{conv1_6b_2comp}  "+(j==8)*'\n')
        x = self.pool1(x)
        x_int_pool=x.int()
        with open('D:\Python_code\IC2_python\pool1_result.txt', 'w') as f:
            for k in range(ch_in):
                for i in range(4):
                    for j in range(4):
                        pool1_6b_2comp=to_6bit_twos_complement(x_int_pool[0,k-1,i,j])
                        f.write(f"{pool1_6b_2comp}  "+(j==3)*'\n')
        #print(x.shape)
        #print(x.int())
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        return x
image, label = test_dataset[12-1]
image_cropped = image[:, crop_size:-crop_size, crop_size:-crop_size]*15

net_watch = watch_leNet()
net_watch.load_state_dict(torch.load('D:\Python_code\IC2_python\weight_binary_lenet2.pth',weights_only=True))
image_test=torch.floor(image_cropped)
image_test = image_test.unsqueeze(0)
image_test[0,0,:,:]=(image_test[0,0,:,:]>7.0)
#print(image_test)
#print(image_test.shape)
output_test = net_watch(image_test)
print(output_test)
```

（二）硬件实现代码：

（1）顶层模块（TOP）

```
module top(

    input clk,

    input rst_n,

    input mode_port,

    input [15:0]data_in_port,

    input in_valid_port,

    output [1:0]data_out_port,

    output in_ready_port,

    output out_en_port

);

parameter CHANNEL_NUM = 6;

//data in and out 3-state-gate

wire data_out_en;

// assign data_out_en = 1'b0;//test

assign out_en_port = data_out_en;

wire [15:0] data_in;

wire [1:0]data_out;

assign data_out_port = data_out;

assign data_in = data_in_port;

wire [CHANNEL_NUM*16-1:0] weight;

wire weight_req;//,weight_OK;

wire data2conv_valid;

wire conv2slide_valid;

wire slilde2pool_valid;

// wire pool2fc_valid;

wire fc2out_valid;

wire [16-1:0]data_in2conv;

wire [CHANNEL_NUM*5-1:0]data_conv2slide;

wire [CHANNEL_NUM*9*5-1:0]data_slide2pool;

wire [CHANNEL_NUM*16*5-1:0]data_pool2fc;

wire [CHANNEL_NUM*9-1:0]data_fc2out;

assign in_ready_port = 1'b1;

// wire soft_rst;

data u_data(

    .clk            (clk                ),//input

    .rst_n          (rst_n              ),//input

    .mode           (mode_port          ),//input
```

```
    .valid          (in_valid_port      ),//input
    .data_in        (data_in            ),//input
    .weight_req     (weight_req         ),//input
    // .weight_rst     (soft_rst           ),//input
    // .weight_OK      (weight_OK          ),//output
    .weight0        (weight[15:0]       ),//output
    .weight1        (weight[31:16]      ),//output
    .weight2        (weight[47:32]      ),//output
    .weight3        (weight[63:48]      ),//output
    .weight4        (weight[79:64]      ),//output
    .weight5        (weight[95:80]      ),//output
    .conv_data_pass (data2conv_valid     ),//output
    .data0          (data_in2conv[0]  ),//output
    .data1          (data_in2conv[1]  ),//output
    .data2          (data_in2conv[2]  ),//output
    .data3          (data_in2conv[3]  ),//output
    .data4          (data_in2conv[4]  ),//output
    .data5          (data_in2conv[5]  ),//output
    .data6          (data_in2conv[6]  ),//output
    .data7          (data_in2conv[7]  ),//output
    .data8          (data_in2conv[8]  ),//output
    .data9          (data_in2conv[9]  ),//output
    .data10         (data_in2conv[10] ),//output
    .data11         (data_in2conv[11] ),//output
    .data12         (data_in2conv[12] ),//output
    .data13         (data_in2conv[13] ),//output
    .data14         (data_in2conv[14] ),//output
    .data15         (data_in2conv[15] )//output
    // .ready          (in_ready_port      ) //output
);

conv_top #(
      .CONV_DATA_WIDTH(1),
      .FC_DATA_WIDTH(5),
      .K(4),
      .LOGK(4),
      .CH_NUM(6)
    ) u_conv_top(
    // Clock input
    .clk            (clk            ), // input
    .rstn           (rst_n           ), // input
    .conv_valid     (data2conv_valid ), // input
    // .conv_ready     (      ), // output
    // .fc_valid       (pool2fc_valid), // input
    // .fc_ready       (       ), // output
    // .weight_ready   (weight_OK      ), // input
```

```
    .weight_req     (weight_req    ), // output
    .out_conv_valid (conv2slide_valid ), // output
    .out_fc_valid   (fc2out_valid   ), // output
    .weight         (weight        ), // input
    .conv_data_in   (data_in2conv   ), // input
    .conv_data_out  (data_conv2slide), // output
    .fc_data_in     (data_pool2fc   ), // input
    .fc_data_out    (data_fc2out    )  // output
);
conv_slide #(
      .CH_NUM(CHANNEL_NUM),
      .DATA_WIDTH(5),
      .K(3),
      .LEN(9)
    )u_conv_slide(
    .clk    (clk    ),//input
    .rstn   (rst_n   ),//input
    .ivalid (conv2slide_valid ),//input
    .idata  (data_conv2slide  ),//input
    .dout   (data_slide2pool   ),//output
    .ovalid (slilde2pool_valid ) //output
);
pool_top u_pool_top(
    .clk      (clk      ),//input
    .rst      (rst_n      ),//input
    .valid    (slilde2pool_valid),//input
    .data_in  (data_slide2pool  ),//input
    .memory_0 (data_pool2fc[0     +:16*5]),//output
    .memory_1 (data_pool2fc[16*5*1+:16*5] ),//output
    .memory_2 (data_pool2fc[16*5*2+:16*5] ),//output
    .memory_3 (data_pool2fc[16*5*3+:16*5] ),//output
    .memory_4 (data_pool2fc[16*5*4+:16*5] ),//output
    .memory_5 (data_pool2fc[16*5*5+:16*5] ) //output
);
compare_probabilities u_compare_probabilities(
    .clk        (clk        ),
    .reset      (rst_n      ),
    .data_in    (data_fc2out    ),
    .data_valid (fc2out_valid ),
    .data_out   (data_out   ),
    .done       (data_out_en )
    // .soft_rst   (soft_rst   )
);
endmodule
```

（2）卷积+全连接权重存储控制系统（含卷积滑窗计算模块）

```
module data(

    input clk,            // 时钟信号

    input rst_n,          // 复位信号，低电平有效

    input mode,           // 模式选择信号,为 1 是权重加载，为 0 是数据加载计算

    input valid,           //外部握手信号，表示此时外面可以发送信号

    input [15:0] data_in,  // 共 16 位输入数据

    input weight_req,//表示权重矩阵开始重新加载数据量

    output  reg[15:0] weight0,// 六个卷积核模块

    output  reg[15:0] weight1,// 六个卷积核模块

    output  reg[15:0] weight2,// 六个卷积核模块

    output  reg[15:0] weight3,// 六个卷积核模块

    output  reg[15:0] weight4,// 六个卷积核模块

    output  reg[15:0] weight5,// 六个卷积核模块

    output reg conv_data_pass,//处理模块的准备信号

    output  data0,

    output  data1,

    output  data2,

    output  data3,

    output  data4,

    output  data5,

    output  data6,

    output  data7,

    output  data8,

    output  data9,

    output  data10,

    output  data11,

    output  data12,

    output  data13,

    output  data14,

    output  data15

);

// 内部变量

reg [95:0] conv_weight_buffer;

reg [959:0] weight_buffer; // 用于存储输入权重的缓冲区

reg [10:0] weight_count;// 用于计数权重已经发送的位数（包含两个权重矩阵）

reg [8:0] data_count;//用于计数到底多少 20x20 数据已经被传递到 5x29 寄存器内部

reg data_buffer[63:0];//3x20+4 寄存器计算单元

reg [3:0] flag1;//用来判断每次计算时需要进行两次数据移位和打入操作

reg [3:0] flag2;//用来判断最后计算过程的终止

reg [3:0] flag3;

always @(posedge clk or negedge rst_n) begin

    if(!rst_n)begin
```

```
        weight_count<=0;
    end
    else begin
        if(mode && valid)begin
            if (weight_count < 6) begin
                // First 6 chunks go into conv_weight_buffer
                conv_weight_buffer[(weight_count*16) +: 16] <= data_in;
                weight_count <= weight_count + 1;
            end
            else begin
                // Remaining 60 chunks go into weight_buffer
                weight_buffer[((weight_count - 6)*16) +: 16] <= data_in;
                if (weight_count == 65)
                    weight_count <= 0;
                else
                    weight_count <= weight_count + 1;
            end
        end
    end
end
integer i,j,k;
// 数据输入处理
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        flag1 <= 4'b0;
        flag2 <= 4'b0;
        data_count <= 9'd0;
        conv_data_pass <= 1'b0;
    end else begin
        // Only proceed in data mode when valid is high
        if ((!mode) && valid) begin


            // Reset conv_data_pass at the beginning of each new data sequence
            if (data_count == 0)
                conv_data_pass <= 1'b0;
```

```
            // First 64 bits load
            if (data_count < 64) begin
                // Shift
                // integer i;
                for (i = 63; i >= 2; i = i - 1)
                    data_buffer[i] <= data_buffer[i - 2];


                // Quantize new bits
                data_buffer[1] <= (data_in[7:0] > 8'd127);
```

```
            data_buffer[0] <= (data_in[15:8] > 8'd127);
```

```
            data_count <= data_count + 2;
```

```
            if (data_count == 62) begin
                conv_data_pass <= 1'b1;
                flag1 <= flag1 + 1'b1;
            end
        // Next loads up to 400 bits
        end else if (data_count < 400) begin
            // Single-line shift
            if (flag1 < 9) begin
                flag2 <= 4'b0;
                // integer j;
                for (j = 63; j >= 2; j = j - 1)
                    data_buffer[j] <= data_buffer[j - 2];

                data_buffer[1] <= (data_in[7:0] > 8'd127);
                data_buffer[0] <= (data_in[15:8] > 8'd127);
```

```
                if (data_count < 398) begin
                    data_count <= data_count + 2;
                    flag1 <= flag1 + 1'b1;
                end else begin
                    data_count <= 9'd0;
                    flag1 <= 4'b0;
                end
            // Double-line shift
            end else begin
                if (conv_data_pass)
                    conv_data_pass <= 1'b0;
                // integer k;
                for (k = 63; k >= 2; k = k - 1)
                    data_buffer[k] <= data_buffer[k - 2];
                data_buffer[1] <= (data_in[7:0] > 8'd127);
                data_buffer[0] <= (data_in[15:8] > 8'd127);
                data_count <= data_count + 2;
                flag2 <= flag2 + 1'b1;
                // After shifting two lines, re-enable conv_data_pass
                if (flag2 == 4'd11) begin
                    flag1 <= 4'd1;
                    conv_data_pass <= 1'b1;
                end
            end
        end
```

```
        end else if (data_count == 0)begin
            conv_data_pass <= 1'b0;
        end
    end
end
always @(posedge clk or negedge rst_n) begin
    if(!rst_n)begin
        flag3<=0;
    end else if (weight_req)begin
        flag3<=flag3==10?0:flag3+1;
    end
end
always@(*)begin
        case (flag3)
            0:begin
                weight0=conv_weight_buffer[15-:16];
                weight1=conv_weight_buffer[31-:16];
                weight2=conv_weight_buffer[47-:16];
                weight3=conv_weight_buffer[63-:16];
                weight4=conv_weight_buffer[79-:16];
                weight5=conv_weight_buffer[95-:16];
            end
            1:begin
                weight0=weight_buffer[15-:16];
                weight1=weight_buffer[31-:16];
                weight2=weight_buffer[47-:16];
                weight3=weight_buffer[63-:16];
                weight4=weight_buffer[79-:16];
                weight5=weight_buffer[95-:16];
            end
            2:begin
                weight0=weight_buffer[111-:16];
                weight1=weight_buffer[127-:16];
                weight2=weight_buffer[143-:16];
                weight3=weight_buffer[159-:16];
                weight4=weight_buffer[175-:16];
                weight5=weight_buffer[191-:16];
            end
            3:begin
                weight0=weight_buffer[207-:16];
                weight1=weight_buffer[223-:16];
                weight2=weight_buffer[239-:16];
                weight3=weight_buffer[255-:16];
                weight4=weight_buffer[271-:16];
                weight5=weight_buffer[287-:16];
```

```
        end
        4:begin
            weight0=weight_buffer[303-:16];
            weight1=weight_buffer[319-:16];
            weight2=weight_buffer[335-:16];
            weight3=weight_buffer[351-:16];
            weight4=weight_buffer[367-:16];
            weight5=weight_buffer[383-:16];
        end
        5:begin
            weight0=weight_buffer[399-:16];
            weight1=weight_buffer[415-:16];
            weight2=weight_buffer[431-:16];
            weight3=weight_buffer[447-:16];
            weight4=weight_buffer[463-:16];
            weight5=weight_buffer[479-:16];
        end
        6:begin
            weight0=weight_buffer[495-:16];
            weight1=weight_buffer[511-:16];
            weight2=weight_buffer[527-:16];
            weight3=weight_buffer[543-:16];
            weight4=weight_buffer[559-:16];
            weight5=weight_buffer[575-:16];
        end
        7:begin
            weight0=weight_buffer[591-:16];
            weight1=weight_buffer[607-:16];
            weight2=weight_buffer[623-:16];
            weight3=weight_buffer[639-:16];
            weight4=weight_buffer[655-:16];
            weight5=weight_buffer[671-:16];
        end
        8:begin
            weight0=weight_buffer[687-:16];
            weight1=weight_buffer[703-:16];
            weight2=weight_buffer[719-:16];
            weight3=weight_buffer[735-:16];
            weight4=weight_buffer[751-:16];
            weight5=weight_buffer[767-:16];
        end
        9:begin
            weight0=weight_buffer[783-:16];
            weight1=weight_buffer[799-:16];
            weight2=weight_buffer[815-:16];
```

```
            weight3=weight_buffer[831-:16];
            weight4=weight_buffer[847-:16];
            weight5=weight_buffer[863-:16];
        end
        10:begin
            weight0=weight_buffer[879-:16];
            weight1=weight_buffer[895-:16];
            weight2=weight_buffer[911-:16];
            weight3=weight_buffer[927-:16];
            weight4=weight_buffer[943-:16];
            weight5=weight_buffer[959-:16];
        end
```

```
        default: begin
            weight0=0;
            weight1=0;
            weight2=0;
            weight3=0;
            weight4=0;
            weight5=0;
        end
    endcase
end
assign data0=data_buffer[63];
assign data1=data_buffer[62];
assign data2=data_buffer[61];
assign data3=data_buffer[60];
assign data4=data_buffer[43];
assign data5=data_buffer[42];
assign data6=data_buffer[41];
assign data7=data_buffer[40];
assign data8=data_buffer[23];
assign data9=data_buffer[22];
assign data10=data_buffer[21];
assign data11=data_buffer[20];
assign data12=data_buffer[3];
assign data13=data_buffer[2];
assign data14=data_buffer[1];
assign data15=data_buffer[0];
endmodule
```

（3）卷积全连接复用计算模块：

1. 顶层模块：

```
module conv_top#(

    parameter CONV_DATA_WIDTH = 1,

    parameter FC_DATA_WIDTH = 6,

    parameter K = 4,

    parameter LOGK = 4,

    parameter CH_NUM = 6

)(

    input wire clk,

    input wire rstn,

    input wire conv_valid,

    // output wire conv_ready,

    // input wire fc_valid,

    // output wire fc_ready,

    // input wire weight_ready,

    output wire weight_req,

    output wire out_conv_valid,

    output wire out_fc_valid,

    input wire [CH_NUM*K*K-1:0] weight,

    input wire [K*K*CONV_DATA_WIDTH-1:0] conv_data_in,

    output wire [CH_NUM*(CONV_DATA_WIDTH+LOGK+1-1) -1:0] conv_data_out,

    input wire [CH_NUM*K*K*FC_DATA_WIDTH-1:0] fc_data_in,

    output wire [CH_NUM*(FC_DATA_WIDTH+LOGK)-1:0] fc_data_out

);

reg mode; //0 for conv, 1 for fc

// wire [CH_NUM*K*K*FC_DATA_WIDTH-1:0] pad_conv_data_in;

wire [CH_NUM*K*K*FC_DATA_WIDTH-1:0] data_in;

wire [CH_NUM*(FC_DATA_WIDTH+LOGK)-1:0] data_out;

//ready&req signal to:weight, conv_slide_input, fc_input

// reg conv_ready_r;

// reg fc_ready_r;

reg weight_req_r;

// assign conv_ready = conv_ready_r;

// assign fc_ready = fc_ready_r;

assign weight_req = weight_req_r;

//output valid signal for:conv_output, fc_output

reg out_conv_valid_r,out_conv_valid_rr;

reg out_fc_valid_r,out_fc_valid_rr;

assign out_conv_valid = out_conv_valid_rr;

assign out_fc_valid = out_fc_valid_rr;

always @(posedge clk or negedge rstn) begin

    if(~rstn)begin

        out_conv_valid_rr<=0;

        out_fc_valid_rr<=0;

        // out_conv_valid_rrr<=0;

        // out_fc_valid_rrr<=0;
```

```
    end
    else begin
        out_conv_valid_rr<=out_conv_valid_r;
        // out_conv_valid_rrr<=out_conv_valid_rr;
        out_fc_valid_rr<=out_fc_valid_r;
        // out_fc_valid_rrr<=out_fc_valid_rr;
    end
end
//-------------FSM and Counter update--------------------
reg [6:0] conv_cnt,conv_cnt_nxt;
reg [3:0] fc_cnt,fc_cnt_nxt;

reg [2:0] cur_state,nxt_state;
localparam S_CONV = 3'b001;
localparam S_WEIGHT = 3'b010;
localparam S_FC = 3'b100;
always @(posedge clk or negedge rstn) begin
    if(~rstn)begin
        cur_state<= S_CONV;
        conv_cnt<=0;
        fc_cnt<=0;
    end
    else begin
        cur_state<=nxt_state;
        conv_cnt<=conv_cnt_nxt;
        fc_cnt<=fc_cnt_nxt;
    end
end
always @(*) begin
    case(cur_state)
        S_CONV:begin
            nxt_state = conv_cnt == 81 ? S_WEIGHT : S_CONV;
        end
        S_WEIGHT:begin
            // nxt_state = weight_ready ? S_FC : S_WEIGHT;
            nxt_state = (fc_cnt==10)?S_CONV:S_FC;
        end
        S_FC:begin
            nxt_state =  S_WEIGHT;
        end
        default:begin
            nxt_state = S_CONV;
        end
    endcase
end
```

```
always @(*) begin
    case(cur_state)
        S_CONV:begin
            mode = 0;
            // conv_ready_r = 1;
            // fc_ready_r = 0;
            weight_req_r = 0;
            out_conv_valid_r = conv_valid;
            out_fc_valid_r = 0;
        end
        S_WEIGHT:begin
            mode = 1;
            // conv_ready_r = 0;
            // fc_ready_r = 0;
            weight_req_r = 1;
            out_conv_valid_r = 0;
            out_fc_valid_r = 0;
        end
        S_FC:begin
            mode = 1;
            // conv_ready_r = 0;
            // fc_ready_r = 1;
            weight_req_r = 0;
            out_conv_valid_r = 0;
            out_fc_valid_r = 1;
        end
        default:begin
            mode = 0;
            // conv_ready_r = 0;
            // fc_ready_r = 0;
            weight_req_r = 0;
            out_conv_valid_r = 0;
            out_fc_valid_r = 0;
        end
    endcase
end
//counter update
always @(*) begin
    case(cur_state)
        S_CONV:begin
            conv_cnt_nxt = conv_valid ? conv_cnt+1 : conv_cnt;
```

```
            fc_cnt_nxt = 0;
        end
        S_WEIGHT:begin
            conv_cnt_nxt = 0;
            fc_cnt_nxt = fc_cnt;
        end
        S_FC:begin
            conv_cnt_nxt = 0;
            fc_cnt_nxt =  fc_cnt+1;
        end
        default:begin
            conv_cnt_nxt = 0;
            fc_cnt_nxt = 0;
        end
    endcase
end
// //conv_in_data copy and pad
// generate
//     genvar i0;
//     for(i0=0;i0<CH_NUM*K*K;i0=i0+1)begin:conv_in_pad
//         assign pad_conv_data_in[i0*FC_DATA_WIDTH +: FC_DATA_WIDTH] = conv_data_in[i0*CONV_DATA_WIDTH +: CONV_DATA_WIDTH];
//     end
// endgenerate
//input mux
generate
    genvar i1;
    for(i1=0;i1<CH_NUM*K*K;i1=i1+1)begin:data_in_mux_instance
        assign data_in[FC_DATA_WIDTH*i1 +:FC_DATA_WIDTH] =  mode ? fc_data_in[i1*FC_DATA_WIDTH +: FC_DATA_WIDTH]
                                    : {{(FC_DATA_WIDTH-CONV_DATA_WIDTH){1'b0}}, conv_data_in[(i1%(K*K))*CONV_DATA_WIDTH +:CONV_DATA_WIDTH]};
    end
endgenerate
//conv instance
generate
    genvar i2;
    for(i2=0;i2<CH_NUM;i2=i2+1)begin:conv_inst
        conv #(
            .DATA_WIDTH(FC_DATA_WIDTH),
            .K(K),
            .LOGK(LOGK)
        )conv_inst(
            .clk(clk),
            .idata(data_in[i2*K*K*FC_DATA_WIDTH +: K*K*FC_DATA_WIDTH]),
            .weight(weight[i2*K*K +: K*K]),
            .dout(data_out[i2*(FC_DATA_WIDTH+LOGK) +: (FC_DATA_WIDTH+LOGK)])
        );
```

```
    end
endgenerate
//output change
generate
    genvar i3;
    for(i3=0;i3<CH_NUM;i3=i3+1)begin:data_out_mux_instance
        assign fc_data_out[i3*(FC_DATA_WIDTH+LOGK) +: (FC_DATA_WIDTH+LOGK)] = data_out[i3*(FC_DATA_WIDTH+LOGK) +: (FC_DATA_WIDTH+LOGK)];
        assign conv_data_out[i3*(CONV_DATA_WIDTH+LOGK+1-1) +: (CONV_DATA_WIDTH+LOGK+1-1) ] =
                            {data_out[i3*(FC_DATA_WIDTH+LOGK)+CONV_DATA_WIDTH+LOGK],data_out[i3*(FC_DATA_WIDTH+LOGK) +:CONV_DATA_WIDTH+LOGK+1-2]};
    end
endgenerate
endmodule
```

2. 分模块：卷积计算模块：

```
module conv #(
    parameter DATA_WIDTH = 4,
    parameter K = 4,
    parameter LOGK = 4
)(
    input wire clk,
    input wire [K*K*DATA_WIDTH -1:0] idata,
    input wire [K*K -1:0] weight, //0 1 2 3 4(top) / 5 6 7 8 9 / 10 11 12 13 14 / 15 16 17 18 19 / 20 21 22 23 24(bottom) like that
    output wire [DATA_WIDTH+LOGK -1:0] dout
);
wire [DATA_WIDTH-1 :0]data_vert[K*K-1:0];
wire [DATA_WIDTH + LOGK -1:0]data_mul[K*K-1:0];
wire [DATA_WIDTH + LOGK -1:0]sum;
reg [DATA_WIDTH-1 :0]data_vert_r[K*K-1:0];
generate
    genvar j;
    for(j=0;j<K*K;j=j+1)begin
        assign data_vert[j] = ({DATA_WIDTH{weight[j]}} ~^ idata[((j+1)*DATA_WIDTH-1):j*DATA_WIDTH]) + {{(DATA_WIDTH-1){1'b0}}, ~weight[j]};
        assign data_mul[j] = {{LOGK{data_vert_r[j][DATA_WIDTH-1]}}, data_vert_r[j]};
    end
endgenerate
integer  i;
always @(posedge clk) begin
    for(int i=0;i<K*K;i=i+1)begin
        data_vert_r[i]<=data_vert[i];
    end
end
assign sum=data_mul[0]+data_mul[1]+data_mul[2]+data_mul[3]+data_mul[4]+
           data_mul[5]+data_mul[6]+data_mul[7]+data_mul[8]+data_mul[9]+
           data_mul[10]+data_mul[11]+data_mul[12]+data_mul[13]+data_mul[14]+
```

```
          data_mul[15];

assign dout = sum;

endmodule
```

（4）池化滑窗模块：

TOP：

```
module conv_slide #(

    parameter CH_NUM = 6,

    parameter DATA_WIDTH = 6,

    parameter K = 3,

    parameter LEN = 9

)(

    input wire clk,

    input wire rstn,

    input wire ivalid,

    input wire [CH_NUM*DATA_WIDTH-1:0] idata,

    output wire [CH_NUM*K*K*DATA_WIDTH-1:0] dout,

    output wire ovalid

);

localparam NUM = LEN*(K-1)+K;

reg slide_en_reg;

reg [3:0] cur_state,nxt_state;

// reg [4:0] pre_cnt,pre_cnt_nxt;

reg ovalid_reg;

reg col_valid;

reg [3:0] shift_cnt,shift_cnt_nxt;

reg [3:0] line_cnt,line_cnt_nxt;

assign ovalid = ovalid_reg & col_valid;

localparam S_PRE = 4'b0001;

localparam S_NORMAL = 4'b0010;

localparam S_CHANGELINE = 4'b0100;

localparam S_END = 4'b1000;

always @       nxt_state = S_PRE;

           end

       end

       S_NORMAL:begin

           if(shift_cnt_nxt==1 && shift_cnt!=shift_cnt_nxt)begin

               nxt_state = S_CHANGELINE;

           end

           else if(shift_cnt_nxt==LEN && line_cnt==LEN-1)begin

               nxt_state = S_END;

           end
```

```
            else begin
                nxt_state = S_NORMAL;
            end
        end
        S_CHANGELINE:begin
            if(shift_cnt_nxt==1 && shift_cnt!=shift_cnt_nxt )begin
                nxt_state = (line_cnt==LEN-1)?S_END:S_NORMAL;
            end
            else begin
                nxt_state = S_CHANGELINE;
            end
        end
        S_END:begin
            nxt_state = S_PRE;
        end
        default:begin
            nxt_state = S_PRE;
        end
    endcase
end
always @(*) begin
    case(cur_state)
        S_PRE:begin
            col_valid = 0;
            slide_en_reg = ivalid;
        end
        S_NORMAL:begin
            col_valid  = (shift_cnt[0]==1)&(shift_cnt>K-1) ;
            slide_en_reg = ivalid;
        end
        S_CHANGELINE:begin
            col_valid = 0;
            slide_en_reg = ivalid;
        end
        S_END:begin
            col_valid = (shift_cnt[0]==1)&(shift_cnt>K-1);
            slide_en_reg = ivalid;
        end
        default:begin
            col_valid = 0;
            slide_en_reg = 0;
        end
    endcase
end
always @(*) begin
```

```
    if(ivalid)begin
        shift_cnt_nxt=(shift_cnt==LEN)?1:shift_cnt+1;
        line_cnt_nxt=(shift_cnt==LEN)?line_cnt+1:line_cnt;
    end
    else begin
        shift_cnt_nxt=(cur_state==S_END)?0:shift_cnt;
        line_cnt_nxt=(cur_state==S_END)?0:line_cnt;
    end
end
generate
    genvar n;
    for(n=0;n<CH_NUM;n=n+1)begin:conv_slide_instance
        conv_single_slide #(
            .DATA_WIDTH(DATA_WIDTH),
            .K(K),
            .LEN(LEN)
        )u_conv_single_slide(
            .clk    (clk   ),
            .rstn   (rstn  ),
            .ivalid (slide_en_reg),
            .idata  (idata[n*DATA_WIDTH +: DATA_WIDTH]),
            .odata  (dout[n*K*K*DATA_WIDTH +: K*K*DATA_WIDTH])
        );
    end
endgenerate
endmodule(posedge clk or negedge rstn) begin
    if(~rstn)begin
        cur_state<= S_PRE;
        shift_cnt<=0;
        line_cnt<=0;
        ovalid_reg<=0;
    end
    else begin
        ovalid_reg <= slide_en_reg;
        cur_state<=nxt_state;
        shift_cnt<=shift_cnt_nxt;
        line_cnt<=line_cnt_nxt;
    end
end
always @(*) begin
    case(cur_state)
        S_PRE:begin
            if(line_cnt_nxt==K-1 && shift_cnt_nxt==1)begin
                nxt_state = S_NORMAL;
            end
```

```
          else begin
```

Single：

```
module conv_single_slide #(
    parameter DATA_WIDTH = 6,
    parameter K = 3,
    parameter LEN = 9
)(
    input wire clk,
    input wire rstn,
    input wire ivalid,
    input wire [DATA_WIDTH-1:0] idata,
    output wire [K*K*DATA_WIDTH-1:0] odata
);
localparam NUM = LEN*(K-1)+K;
reg [DATA_WIDTH-1:0] data[NUM];
integer i;
always @(posedge clk) begin
    if(ivalid)begin
        for(i=0;i<NUM-1;i=i+1)begin
            data[i]<=data[i+1];
        end
        data[NUM-1]<=idata;
    end
end
assign odata = {data[20],data[19],data[18],data[11],data[10],data[9],data[2],data[1],data[0]};
endmodule
```

（5）池化模块

1. 池化操作

Top：

```
module pool_top (
    input wire clk,                         // 时钟信号
    input wire rst,                         // 复位信号
    input wire valid,                // 写使能信号
    input wire [5*3*3*6-1:0] data_in,       // 输入数据：6 个通道的 3x3 矩阵，每个矩阵元素为 6 位
    output wire [5*16-1:0] memory_0,             // 存储模块 0 输出
    output wire [5*16-1:0] memory_1,             // 存储模块 1 输出
```

```
    output wire [5*16-1:0] memory_2,             // 存储模块 2 输出
    output wire [5*16-1:0] memory_3,             // 存储模块 3 输出
    output wire [5*16-1:0] memory_4,             // 存储模块 4 输出
    output wire [5*16-1:0] memory_5              // 存储模块 5 输出
);
    // 输出数据，池化后的结果
    wire [4:0] pool_data_out_0;
    wire [4:0] pool_data_out_1;
    wire [4:0] pool_data_out_2;
    wire [4:0] pool_data_out_3;
    wire [4:0] pool_data_out_4;
    wire [4:0] pool_data_out_5;
    // 实例化池化层模块
    pooling_layer #(
        .NUM_CHANNELS(6),
        .DATA_WIDTH(5),
        .MATRIX_DIM(3)
    ) pooling_inst (
        .data_in(data_in),                      // 输入数据（6 通道，每个通道是一个 3x3 矩阵）
        .data_out({pool_data_out_5, pool_data_out_4, pool_data_out_3, pool_data_out_2, pool_data_out_1, pool_data_out_0}) // 池化结果输出
    );
    // 实例化存储模块
    storage_layer #(
        .DATA_WIDTH(5),
        .NUM_ELEMENTS(16)
    ) storage_inst (
        .clk(clk),                              // 时钟信号
        .reset(rst),                            // 复位信号
        .valid(valid),            // 写使能信号
        .data_in_0(pool_data_out_0),            // 从池化层得到的第 0 通道数据
        .data_in_1(pool_data_out_1),            // 从池化层得到的第 1 通道数据
        .data_in_2(pool_data_out_2),            // 从池化层得到的第 2 通道数据
        .data_in_3(pool_data_out_3),            // 从池化层得到的第 3 通道数据
        .data_in_4(pool_data_out_4),            // 从池化层得到的第 4 通道数据
        .data_in_5(pool_data_out_5),            // 从池化层得到的第 5 通道数据
        .memory_0(memory_0),                    // 存储模块 0 输出
        .memory_1(memory_1),                    // 存储模块 1 输出
        .memory_2(memory_2),                    // 存储模块 2 输出
        .memory_3(memory_3),                    // 存储模块 3 输出
        .memory_4(memory_4),                    // 存储模块 4 输出
        .memory_5(memory_5)                     // 存储模块 5 输出
    );
endmodule
```

Single：

```
module pooling_channel #(
parameter DATA_WIDTH = 6,    // 数据位宽
parameter MATRIX_DIM = 3     // 输入矩阵维度 (3x3)
)
(
input  wire [MATRIX_DIM*MATRIX_DIM*DATA_WIDTH-1:0] data_in, // 输入数据：3x3 矩阵，每个元素为 DATA_WIDTH 位
output wire [DATA_WIDTH-1:0] data_out                    // 输出数据：最大池化值
);
// 内部信号
wire [DATA_WIDTH-1:0] elements [MATRIX_DIM*MATRIX_DIM-1:0]; // 展开后的矩阵元素
// 提取输入数据
genvar i;
generate
for (i = 0; i < MATRIX_DIM*MATRIX_DIM; i = i + 1) begin : extract_data
assign elements[i] = data_in[(5*9-1-i*DATA_WIDTH)-: DATA_WIDTH];
end
endgenerate
// 比较逻辑：逐步比较所有 9 个元素，找出最大值
wire [DATA_WIDTH-1:0] max_1, max_2, max_3, max_4, max_5, max_6, max_7, max_8;
assign max_1 = ($signed(elements[0]) > $signed(elements[1])) ? $signed(elements[0]) : $signed(elements[1]);
assign max_2 = ($signed(elements[2]) > $signed(elements[3])) ? $signed(elements[2]) : $signed(elements[3]);
assign max_3 = ($signed(elements[4]) > $signed(elements[5])) ? $signed(elements[4]) : $signed(elements[5]);
assign max_4 = ($signed(elements[6]) > $signed(elements[7])) ? $signed(elements[6]) : $signed(elements[7]);
assign max_5 = ($signed(max_1) > $signed(max_2)) ? $signed(max_1) : $signed(max_2);
assign max_6 = ($signed(max_3) >$signed(max_4)) ? $signed(max_3) : $signed(max_4);
assign max_7 = ($signed(max_5) > $signed(max_6)) ? $signed(max_5) :$signed(max_6);
assign max_8 = ($signed(max_7) > $signed(elements[8])) ? $signed(max_7) : $signed(elements[8]);
// 控制输出仅在 valid 高时有效
assign data_out = max_8 ;
endmodule
module pooling_layer #(
    parameter NUM_CHANNELS = 6,        // 通道数
    parameter DATA_WIDTH = 6,          // 每个数据位宽
    parameter MATRIX_DIM = 3           // 输入矩阵的维度（3x3）
)(
    input wire [NUM_CHANNELS * MATRIX_DIM * MATRIX_DIM * DATA_WIDTH - 1:0] data_in, // 输入数据（每个通道的 3x3 矩阵）
    output wire [NUM_CHANNELS * DATA_WIDTH - 1:0] data_out    // 输出池化结果（每个通道一个池化值）
);
    // 生成多个并行的池化模块
    genvar i;
    generate
        for (i = 0; i < NUM_CHANNELS; i = i + 1) begin : gen_pooling_channel
            pooling_channel #(
                .DATA_WIDTH(DATA_WIDTH),
                .MATRIX_DIM(MATRIX_DIM)
```

```
        ) channel_inst (
            .data_in(data_in[(i+1)*MATRIX_DIM*MATRIX_DIM*DATA_WIDTH-1:i*MATRIX_DIM*MATRIX_DIM*DATA_WIDTH]),
            .data_out(data_out[(i+1)*DATA_WIDTH-1:i*DATA_WIDTH])
        );
    end
  endgenerate
endmodule
```

2.池化存储

```
module storage_layer #(
    parameter DATA_WIDTH = 6,        // 每个数据位宽
    parameter NUM_ELEMENTS = 16      // 存储元素的数量（假设每个存储模块存储 16 个元素）
)(
    input wire clk,                   // 时钟信号
    input wire reset,                 // 复位信号
    input wire valid,           // 写使能信号
    input wire [DATA_WIDTH-1:0] data_in_0,  // 存储池化模块 0 的输出数据
    input wire [DATA_WIDTH-1:0] data_in_1,  // 存储池化模块 1 的输出数据
    input wire [DATA_WIDTH-1:0] data_in_2,  // 存储池化模块 2 的输出数据
    input wire [DATA_WIDTH-1:0] data_in_3,  // 存储池化模块 3 的输出数据
    input wire [DATA_WIDTH-1:0] data_in_4,  // 存储池化模块 4 的输出数据
    input wire [DATA_WIDTH-1:0] data_in_5,  // 存储池化模块 5 的输出数据

    output wire [DATA_WIDTH*NUM_ELEMENTS-1:0] memory_0,  // 存储模块 0 输出
    output wire [DATA_WIDTH*NUM_ELEMENTS-1:0] memory_1,  // 存储模块 1 输出
    output wire [DATA_WIDTH*NUM_ELEMENTS-1:0] memory_2,  // 存储模块 2 输出
    output wire [DATA_WIDTH*NUM_ELEMENTS-1:0] memory_3,  // 存储模块 3 输出
    output wire [DATA_WIDTH*NUM_ELEMENTS-1:0] memory_4,  // 存储模块 4 输出
    output wire [DATA_WIDTH*NUM_ELEMENTS-1:0] memory_5   // 存储模块 5 输出
);
    // 定义 6 个存储模块，用于存储池化层的输出
    reg [DATA_WIDTH-1:0] memory_0_reg [0:NUM_ELEMENTS-1];
    reg [DATA_WIDTH-1:0] memory_1_reg [0:NUM_ELEMENTS-1];
    reg [DATA_WIDTH-1:0] memory_2_reg [0:NUM_ELEMENTS-1];
    reg [DATA_WIDTH-1:0] memory_3_reg [0:NUM_ELEMENTS-1];
    reg [DATA_WIDTH-1:0] memory_4_reg [0:NUM_ELEMENTS-1];
    reg [DATA_WIDTH-1:0] memory_5_reg [0:NUM_ELEMENTS-1];
    // 计数器，用于确定每次写入哪个位置
    reg [4:0] write_address;
    // 写操作
    always @(posedge clk or negedge reset) begin
        if (!reset) begin
            write_address <= 0; // 复位时计数器归零
```

```
        end else if (valid) begin
            // 将池化模块的输出数据写入对应的存储模块
            memory_0_reg[write_address] <= data_in_0;
            memory_1_reg[write_address] <= data_in_1;
            memory_2_reg[write_address] <= data_in_2;
            memory_3_reg[write_address] <= data_in_3;
            memory_4_reg[write_address] <= data_in_4;
            memory_5_reg[write_address] <= data_in_5;

            if (write_address == NUM_ELEMENTS-1)
                write_address <= 0;  // 达到最后一个位置后，归零计数器
            else
                write_address <= write_address + 1;  // 否则递增计数器
        end
    end
    // 将存储的结果输出到顶层模块
genvar i;
generate
for (i = 0; i <NUM_ELEMENTS ; i = i + 1)
    begin :cunchu
    assign memory_0[(i+1)*5-1:i*5] = memory_0_reg[i];
    assign memory_1[(i+1)*5-1:i*5] = memory_1_reg[i];
    assign memory_2[(i+1)*5-1:i*5] = memory_2_reg[i];
    assign memory_3[(i+1)*5-1:i*5] = memory_3_reg[i];
    assign memory_4[(i+1)*5-1:i*5] = memory_4_reg[i];
    assign memory_5[(i+1)*5-1:i*5] = memory_5_reg[i];
end
endgenerate
endmodule
```

（6）Argmax 比较输出

```
module compare_probabilities # (
    parameter DATA_WIDTH = 9    // 数据位宽
) (
    input clk,                // 时钟信号
    input reset,              // 重置信号
    input [6*DATA_WIDTH-1:0] data_in,       // 输入的 6 个 10 位数据
    input data_valid,          // 数据有效标志，表示数据是否已经有效
    output wire [1:0] data_out, // 输出最大概率对应的数字索引，分两次，先是高位，再是低位
    output reg done           // 标志输出，表示比较完成
    // output reg soft_rst
```

```
);

    // 内部信号声明

    //reg signed [14:0] prob [9:0];   // 存储10个数据

    wire signed [DATA_WIDTH+4-1:0] sum ;

    reg signed [DATA_WIDTH+4-1:0] max_val,max_val_next ;

    reg [3:0] counter;      // 用于计数输入的次数

    reg [3:0] max_index_next; // 最大概率对应的数字索引

    reg [3:0] max_index;    // 最大概率对应的数字索引

    reg [6*DATA_WIDTH-1:0] data_in_r;

    reg data_valid_r;

    always @(posedge clk or negedge reset) begin

        if(!reset)begin

            data_valid_r <= 0;

        end

        else begin

            data_valid_r <= data_valid;

            data_in_r <= data_in;

        end

    end

    assign sum = $signed(data_in_r[0+:DATA_WIDTH])+$signed(data_in_r[DATA_WIDTH+:DATA_WIDTH])+$signed(data_in_r[DATA_WIDTH*2+:DATA_WIDTH])

                +$signed(data_in_r[DATA_WIDTH*3+:DATA_WIDTH])+$signed(data_in_r[DATA_WIDTH*4+:DATA_WIDTH])+$signed(data_in_r[DATA_WIDTH*5+:DATA_WIDTH])
;

    always @(posedge clk or negedge reset) begin

        // 当收集完所有数据后，开始比较最大概率

        if(~reset)begin

            max_index <= 0;

            max_val<={1'b1,{(DATA_WIDTH+4-1){1'b0}}};

            done <= 0;

            counter <=0;

            // soft_rst<=0;

        end else begin

            if(counter<10)begin

                if(data_valid_r)begin

                    max_index <= max_index_next;

                    max_val<=max_val_next;

                    counter <= counter +1;

                end

            end

            else begin

                if(counter==10)begin

                    done<=1;

                    counter<=counter+1;

                end

                else if(counter==11)begin

                    counter<=counter+1;
```

```
                // soft_rst<=1;
            end
            else begin
                counter<=0;
                max_index <= 0;
                max_val<={1'b1,{(DATA_WIDTH+4-1){1'b0}}};
                done <= 0;
                // soft_rst<=0;
            end
        end
        // if (counter == 10) begin
        //     // max_index <= max_index_out_w[9*4+:4];
        // end
    end
    assign data_out = (counter==11)?max_index[3:2]:(counter==12?max_index[1:0]:2'b0);
    always@(*)begin
        if ($signed(sum)>max_val) begin
            max_val_next =sum;
            max_index_next =counter;
        end
        else begin
            max_val_next =max_val;
            max_index_next =max_index;
        end
    end
endmodule
```

（7）端口引用（IOPAD）

```
module top_pad(
    input clk_pad,
    input rst_n_pad,
    input mode_pad,
    input [15:0]data_in_pad,
    input in_valid_pad,
    output [1:0]data_out_pad,
    output in_ready_pad,
    output out_en_pad
);
wire clk         ;
wire rst_n       ;
wire mode_port   ;
wire [15:0]data_in_port ;
```

```
wire in_valid_port;

wire [1:0]data_out_port;

wire in_ready_port;

wire out_en_port  ;

PIW IO_PAD1(.PAD(clk_pad),.C(clk));

PISW IO_PAD2(.PAD(rst_n_pad),.C(rst_n));

PIW IO_PAD3(.PAD(mode_pad),.C(mode_port));

PIW IO_PAD4(.PAD(data_in_pad[0]),.C(data_in_port[0]));

PIW IO_PAD5(.PAD(data_in_pad[1]),.C(data_in_port[1]));

PIW IO_PAD6(.PAD(data_in_pad[2]),.C(data_in_port[2]));

PIW IO_PAD7(.PAD(data_in_pad[3]),.C(data_in_port[3]));

PIW IO_PAD8(.PAD(data_in_pad[4]),.C(data_in_port[4]));

PIW IO_PAD9(.PAD(data_in_pad[5]),.C(data_in_port[5]));

PIW IO_PAD10(.PAD(data_in_pad[6]),.C(data_in_port[6]));

PIW IO_PAD11(.PAD(data_in_pad[7]),.C(data_in_port[7]));

PIW IO_PAD12(.PAD(data_in_pad[8]),.C(data_in_port[8]));

PIW IO_PAD13(.PAD(data_in_pad[9]),.C(data_in_port[9]));

PIW IO_PAD14(.PAD(data_in_pad[10]),.C(data_in_port[10]));

PIW IO_PAD15(.PAD(data_in_pad[11]),.C(data_in_port[11]));

PIW IO_PAD16(.PAD(data_in_pad[12]),.C(data_in_port[12]));

PIW IO_PAD17(.PAD(data_in_pad[13]),.C(data_in_port[13]));

PIW IO_PAD18(.PAD(data_in_pad[14]),.C(data_in_port[14]));

PIW IO_PAD19(.PAD(data_in_pad[15]),.C(data_in_port[15]));

PIW IO_PAD20(.PAD(in_valid_pad),.C(in_valid_port));

PO12W IO_PAD21(.I(data_out_port[0]),.PAD(data_out_pad[0]));

PO12W IO_PAD22(.I(data_out_port[1]),.PAD(data_out_pad[1]));

PO12W IO_PAD23(.I(in_ready_port),.PAD(in_ready_pad));

PO12W IO_PAD24(.I(out_en_port),.PAD(out_en_pad));

top u_top(

    .clk          (clk          ),

    .rst_n        (rst_n        ),

    .mode_port    (mode_port    ),

    .data_in_port  (data_in_port  ),

    .in_valid_port (in_valid_port ),

    .data_out_port (data_out_port ),

    .in_ready_port (in_ready_port ),

    .out_en_port   (out_en_port   )

);

endmodule
```

（8）测试程序（Testbench）

```
`timescale 1ns/1ps

module top_tb;

// Parameter for clock period

parameter CLOCK_PERIOD = 80;

// Clock and reset signals

reg clk;

reg rst_n;

// Input signals

reg mode;

reg valid;

wire [15:0] data_in;

wire [3:0] inout_in;

// Output signals

wire ready;

wire out_en;

// Instantiate top module

top dut (

    .clk(clk),

    .rst_n(rst_n),

    .mode_port(mode),

    .data_inout_port(inout_in),

    .data_in_port(data_in[15:4]),

    .in_valid_port(valid),

    .in_ready_port(ready),

    .out_en_port(out_en)

);

// Tristate buffer for inout_in

assign inout_in = out_en ? 4'bz : data_in[3:0];

wire [3:0] inout_out;

assign inout_out = inout_in;

reg [15:0] weight_in, map_in;

assign data_in = mode ? {weight_in[0], weight_in[1], weight_in[2], weight_in[3], weight_in[4], weight_in[5], weight_in[6], weight_in[7], weight_in[8],

weight_in[9], weight_in[10], weight_in[11], weight_in[12], weight_in[13], weight_in[14], weight_in[15]}: map_in;

// Clock generation

initial clk = 0;

always #(CLOCK_PERIOD / 2) clk = ~clk; // Parameterized clock

// Declare filename as a string

reg [100*8:1] filename;

// Testbench logic

integer i,j,T;

reg [15:0] weight1 [6];
```

```
reg [15:0] weight2 [60];

reg [15:0] data [136];

reg [3:0] expected_label [0:0];

initial begin

    for(T=0;T<10;T=T+1)begin

        // Initialize signals

        rst_n = 0;

        mode = 0;

        valid = 0;

        map_in = 16'd0;

        // Load weights and data from files

        $readmemb("./test/2/cov1_weight.txt", weight1);

        $readmemb("./test/2/fc1_weight.txt", weight2);

        // $readmemb("./test/MNIST_images_0.txt", data);

        // Load multiple data files

        // for (i = 0; i < 10; i = i + 1) begin

        $sformat(filename, "./test/MNIST_images_%0d.txt", T);

        $readmemb(filename, data);

        // end

            // Release reset

        #40;

        rst_n = 1;

        // Load weight1

        mode = 1;

        for (i = 0; i < 6; i = i + 1) begin

            @(negedge clk)begin

                valid = 1;

                weight_in = weight1[i];

            end

        end

        @(negedge clk)begin

            mode = 0;

            valid = 0;

        end

        #400;

        // Load data

        for (i = 0; i < 136; i = i + 1) begin

            @(negedge clk)begin

                valid = 1;

                map_in = data[i];

            end

        end

        @(negedge clk)valid = 0;

        #400;

        // Load weight2
```

```
        @(negedge clk)mode = 1;
        for (i = 0; i < 10; i = i + 1) begin
            wait(ready);
            for(j = 0; j < 6 ; j=j+1)begin
                @(negedge clk)begin
                    valid = 1;
                    weight_in = weight2[i*6+j];
                end
            end
            @(negedge clk );
        end
        @(negedge clk)valid = 0;
        // Compare inout_out with expected values from MNIST_label files
        $sformat(filename, "./test/MNIST_label_%0d.txt", T);
        $readmemh(filename, expected_label);
        @(negedge clk);
        wait(out_en);
        @(negedge clk);
        if (inout_out !== expected_label[0]) begin
            $display("Mismatch at test %2d: expected %3d, got %3d", T, expected_label[0], inout_out);
        end else begin
            $display("   Match at test %2d: expected %3d, got %3d", T, expected_label[0], inout_out);
        end
        // End simulation
        #400;
    end
    $finish;
end
initial begin
    $fsdbDumpfile("sim_output_pluson.fsdb");
    $fsdbDumpvars(0, top_tb);
    $fsdbDumpMDA();
end
endmodule
```