3 不同层的部署方法

FPGA 中神经网络的计算过程和训练网络时的前向推理过程基本相同，但不用使用 dropout 函数防止过拟合，本章主要介绍在卷积神经网络中常见的三种层：卷积层，池化层，以及全连接层的电路结构以及如何进行并行处理。其中卷积层的计算过程最为复杂，而池化层的电路结构和卷积层的结构基本相同。

3.1 卷积层的设计

3.1.1 卷积层的总体架构

卷积层是神经网络中计算最为密集的地方，也是加速的主要层次，其中数据处理单元 PE (processing element)采用了移位寄存器结构，控制逻辑采用了有穷状态机。下面以 AlexNet 的第一层的卷积部分说明具体的设计，其总体的结构如图 3.1 所示，输入特征图大小为 227*227*3，卷积核大小为 11*11*3，共有 96 个，步长 stride 为 4，图中 PE 共有 96*3 个，实际计算时，可将三维的特征图计算过程分为三个通道分别同时被 96 个 PE 同时计算，权值也被同时输入至 PE 中，再经由 96 个加法器相加，每个加法器同时处理三个 PE 端输入和一个偏置输入 bias，卷积结果可存放至 BRAM 中用于下一层的计算或者直接输入激活函数中。因此可以实现，输入特征图的的各个通道被并行处理，输入并行度为 3，输出并行度为 96。

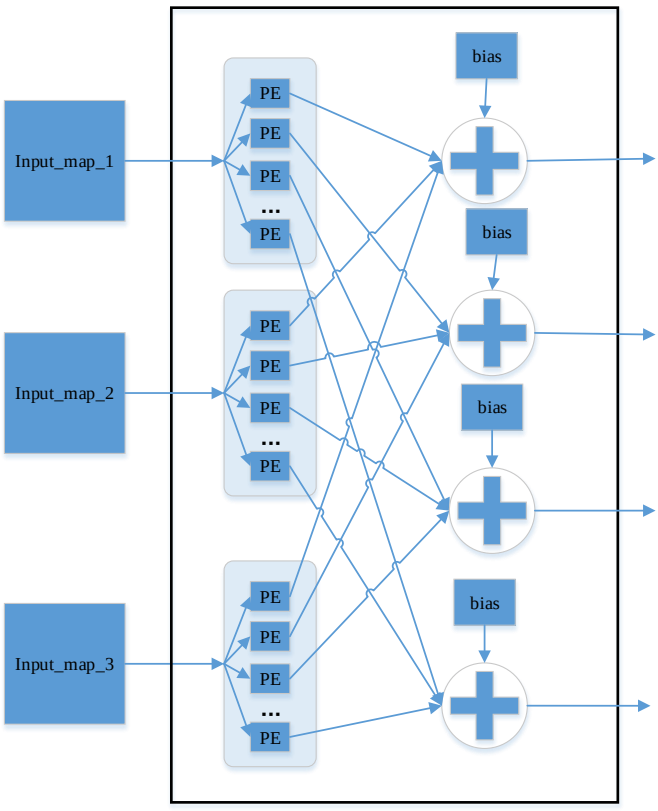

图 3.1 卷积层电路的总体结构

3.1.2 卷积计算单元的数据流

卷积层的计算主要由 PE 完成，仍然以 AlexNet 的第一层为例，对于单个 PE 来说，其处理数据为 227*227 的矩阵，卷积核大小为 11*11，步长为 4,，结构如图 3.2 所示，计算开始时，先将 227*227 的矩阵中的数据逐个输入，因此输入数据总数为 51529 个，移位寄存器的深度和特征图的尺寸相同为 227，移位寄存器的个数应和卷积核尺寸相同为 11，此外还应有 121 个寄存器x_i存放卷积窗的当前窗口值，121 个寄存器w_i存放权值参数，当第一个数据移入寄存器x_0时，寄存器组x_i存放了第一个滑窗的特征图的值，当控制逻辑给出计算信号时，将x_i与w_i相乘，此后将每行中的 11 个乘积相加，再将每一行的和相加，相加结果作为计算单元 PE 的输出。

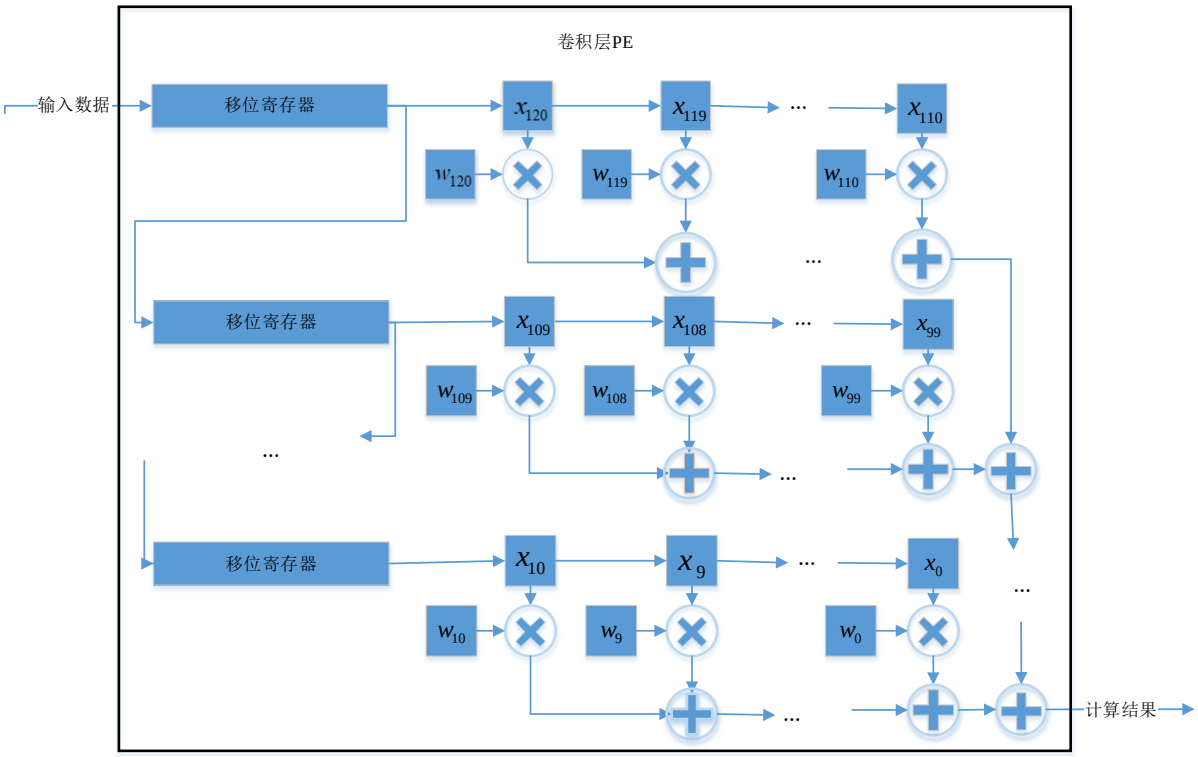

图 3.2 卷积层计算单元的电路结构

3.1.3 卷积计算单元中控制流

PE 中的控制逻辑采用有穷状态机实现，根据卷积的计算过程，可分为三个状态，预填态，右移态，下移态。预填态是指第一个数据进入移位寄存器到第一个数据移入寄存器x_0这段时间的状态，右移态是指卷积窗口右移的状态，如图 3.3 示，下移态是指卷积窗口从行末移动到下一次计算的行初的状态，如图 3.4 示。

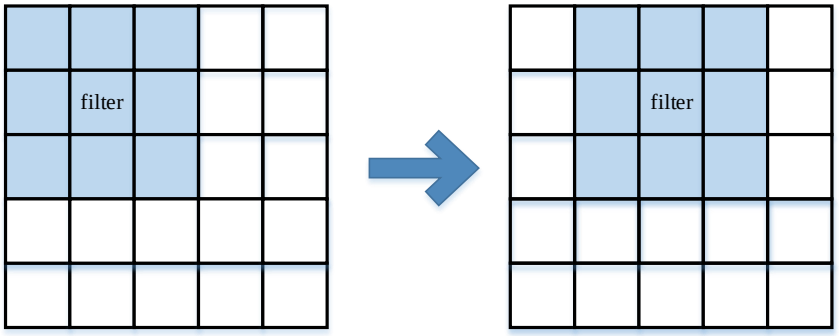

图 3.3 右移态

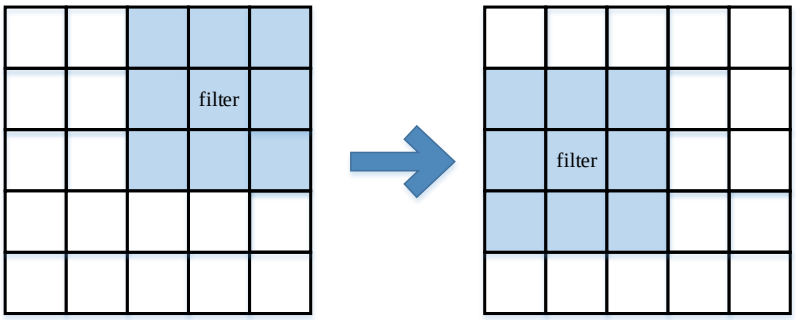

图 3.4 下移态

寄存器$count_{data}$用于记录当前状态移入数据个数，寄存器$count_{result}$用来记录卷积结果的个数，寄存器 start 控制计算是否开始，start 为 1 则下个时钟周期上升沿开始计算，单次计算完成后 start 置 0，$count_{result}$加一。每次时钟上升沿时，移入一个数据，$count_{data}$加一。每次状态转换后$count_{data}$置 1，对于确定的输入特征图大小$size_{map}$，卷积核大小$size_{filter}$，步长stride，当满足式 3.1 时，预填态变为右移态，执行一次计算过程。处于右移态时，每输入 stride 个数据时，执行一次计算过程，当满足式 3.2 时，右移态变为下移态。当满足 3.3 式时，下移态变为右移态，执行一次计算过程。满足式 3.4 时，说明卷积计算全部完成。

$$count_{data} = size_{map} * size_{filter} + size_{filter} \quad 式(3.1)$$

$$count_{data} = \left[\frac{(size_{map} - size_{filter})}{stride}\right] * stride \quad 式(3.2)$$

$$count_{data} = (stride - 1) * size_{map} + (size_{map} - size_{filter})\%stride \quad 式(3.3)$$

$$count_{result} = (\left[\frac{(size_{map} - size_{filter})}{stride}\right] + 1)^2 \quad 式(3.4)$$

卷积计算单元 PE 的控制逻辑的状态机转换的 Verilog 代码可表示如下，其中寄存器 delay 存放状态转换的延时时钟周期数，current_state 存放当前状态，next_state 存放下一状态。

算法 3.1 卷积计算单元控制逻辑算法

输入：时钟信号，使能信号 enable，输入特征图数据

1. always@(时钟上升沿)
2. begin
3. if(enable) begin
4. 数据移入移位寄存器，更新卷积窗口寄存器组x_i
5. if($count_{data}$==delay)
6. $count_{data}$<=1；//表示状态变化,移入数据计数重置
7. else
8. $count_{data}$<=$count_{data}$+1；//状态不变，移入数据加一
9. end
10. end
11. always@(时钟上升沿)
12. begin
13. If(enable)
14. current_state<=next_state; //状态转换
15. end
16. always@(current_state or $count_{data}$) //状态变换或者移入数据则执行
17. begin
18. case(current_state):
19. 预填态: begin delay=$size_{map} * size_{filter} + size_{filter}$；
20. f($count_{data}$==delay) //状态变化的条件
21. start=1;next_state=右移态; //卷积窗口第一次数据完全移入
22. else
23. start=0;
24. end
25. 右移态: begin delay=$\left[\frac{(size_{map}-size_{filter})}{stride}\right] * stride$ ；

26. if($count_{data}$==delay)
27. next_state=下移态;
28. if($count_{data}$%stride==0)
29. start=1; //卷积窗口平移 stride 个单位
30. else
31. start=0;
32. end
33. 下移态: begin delay=$(stride-1)*size_{map}+(size_{map}-size_{filter})\%stride$;
34. if($count_{data}$==delay)
35. start=1;next_state=右移态;
36. else
37. start=0;
38. end
39. endcase
40. always@(start)
41. begin
42. if(start&&enable)
43. 执行一次计算过程;
44. $count_{result}=count_{result}+1$;
45. end

输出：单次卷积计算结果，计算次数$count_{result}$

3.2 池化层的设计

池化层的总体架构和卷积层架构基本相同，池化层计算单元 PE 的控制逻辑和数据流与卷积层 PE 的数据流和控制逻辑完全相同，只是计算操作不同，最大池化是从滑动窗口中选取最大值，平均池化是从滑动窗口中计算平均值。以最大池化为例，如果池化窗口为 2*2，需要将 4 个寄存器中的值输入三个级联的二路选择器，每个二路选择器输出两个输入的较大值，如图 3.5 所示。处理平均池化时，只需将二路选择器改为加法器，将和除以窗口大小，即可得到平均池化的结果。

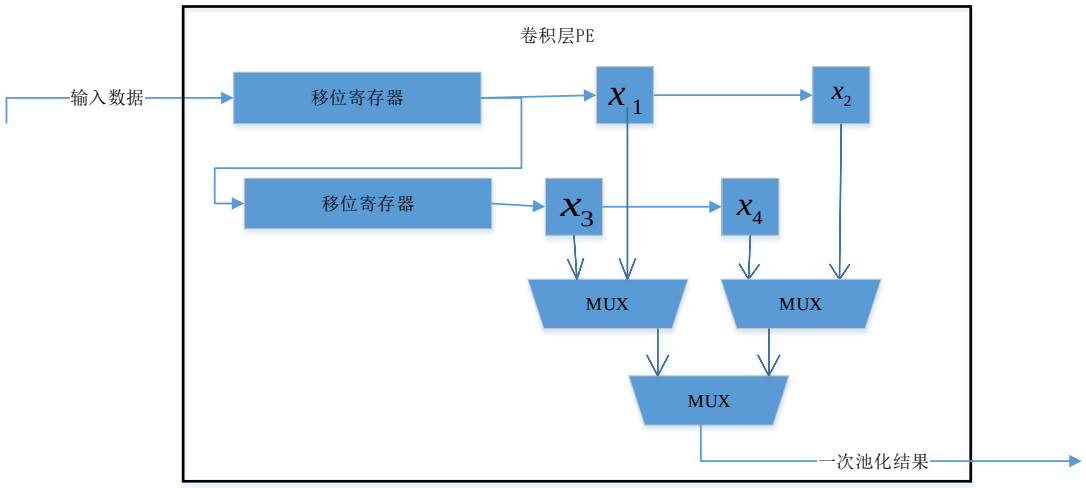

图 3.5 池化层计算单元的计算电路

3.3 全连接层的设计

全连接层的输入通常是 1 维向量，计算过程可以看过 1*N 的矩阵与 N*M 的矩阵相乘，为了提高矩阵的乘法的并行化，可将计算过程分解为一个个 1*N 的矩阵与 M 个 N*1 的矩阵相乘。总体架构如图 3.6 所示，全连接层通常位于网络模型的最后，而前一层的计算结果通常存于片上的 BRAM 中，全连接层计算开始时，1*N 的输入特征图逐个从前一层的缓冲区的 BRAM 中取出输入至全连接层中，全连接层包含 M 个计算单元 PE，每个计算单元完成一个 1*N 的矩阵和一个 N*1 的矩阵相乘，计算结果并行输入至 buffer 中。每个计算单元包含一个乘法器，一个加法器和一个存放结果的寄存器，结构如图 3.7 所示，在 1*N 的矩阵的值全部输入至 PE 前，多路选择器会选择上方的通路，完成矩阵相乘后，多路选择器会选择 bias 通路，此时的和作为输出。

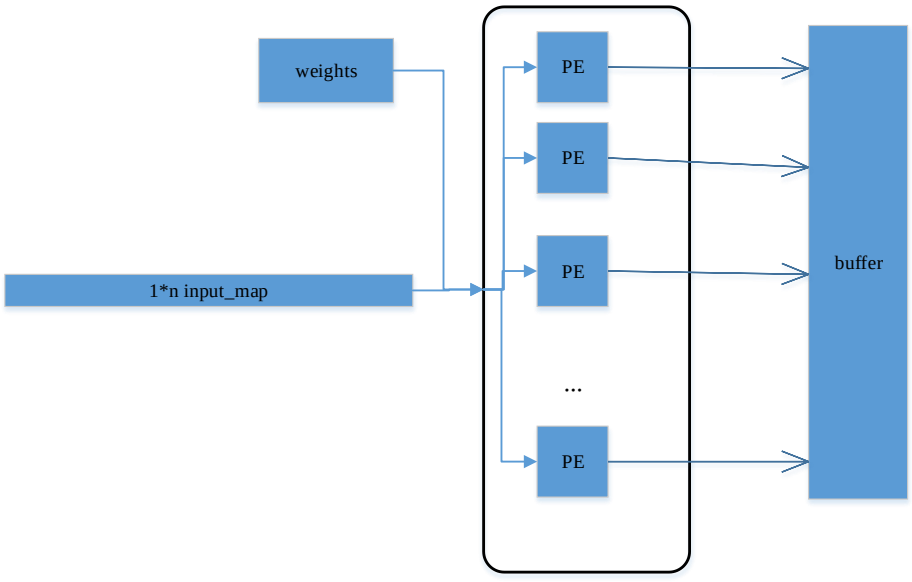

图 3.6 全连接层的总体架构

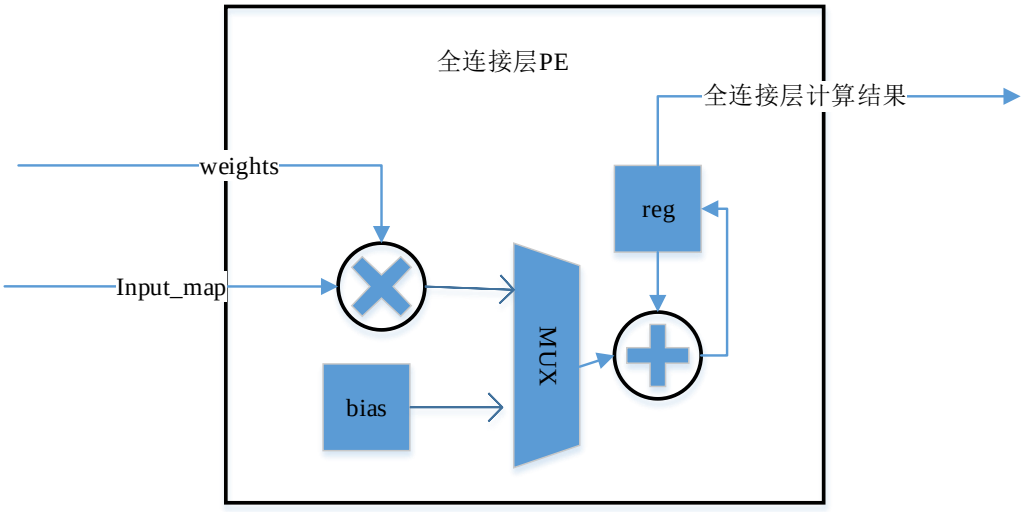

图 3.7 全连接层计算单元电路图

4 LeNet5 设计与实现

4.1 LeNet5 介绍

本次部署实现的 LeNet5 与 Yann LeCun 实现的 LeNet 略有不同，使用了 MNIST 数据集作为训练数据集，包含了多幅 0 到 9 的手写数字图片和相应的标签。网络共有五层，输入图大小为 28*28*1，。每一层的具体介绍如下：

第一层：第一层包含一个卷积层，卷积层的输出经过 ReLU 函数处理后，再输入值最大池化层。卷积层内有 6 个 5*5*1 的卷积核，卷积步长为 1，特征图经过卷积层后输出特征图大小为 24*24*6，24*24*6 的特征图经过 ReLU 函数处理后，输入最大池化层，池化窗口大小为 2*2，步长为 2，最后输出 12*12*6 的特征图。

第二层：第二层同样包含一个卷积层，一个 ReLU 激活函数和一个池化层。第一层的输出作为第二层卷积的输入，卷积层内有 16 个大小为 5*5*6 的卷积核，卷积步长为 1，卷积层输出特征图大小为 8*8*16，8*8*16 的特征图经过 ReLU 函数处理后，输入进池化层，池化窗口大小为 2*2，步长为 2，输出为 4*4*16 的特征图

第三层：第三层包含一个全连接层和一个 ReLU 函数，将 4*4*16 的特征图展开为 1*256 的矩阵后，与 256*120 的矩阵相乘，得到的 1*120 的向量，与 1*120 的 bias 向量相加后经过 ReLU 函数后作为第三层的输出。

第四层：第四层的结构与第三层的结构相同，将输入的 1*120 的向量与 120*84 的矩阵相乘后得到 1*84 的向量，与 1*84 的 bias 向量相加后经过 ReLU 函数处理作为第四层的输出。

第五层：第五层的输入为 1*84 的向量，与 84*10 的矩阵相乘后得到 1*10 的向量，和 1*10 的 bias 向量相加后，得到 1*10 的向量，而数据最大的维度对应的数字的概率最大。

4.2 实现流程

部署的总体过程如图 4.1 示，训练网络导出参数后，将参数定点化处理，存入 coe 文件中，作为 BRAM 的初始化文件，再编写整个网络的 verilog 代码，进行仿真和结果分析。

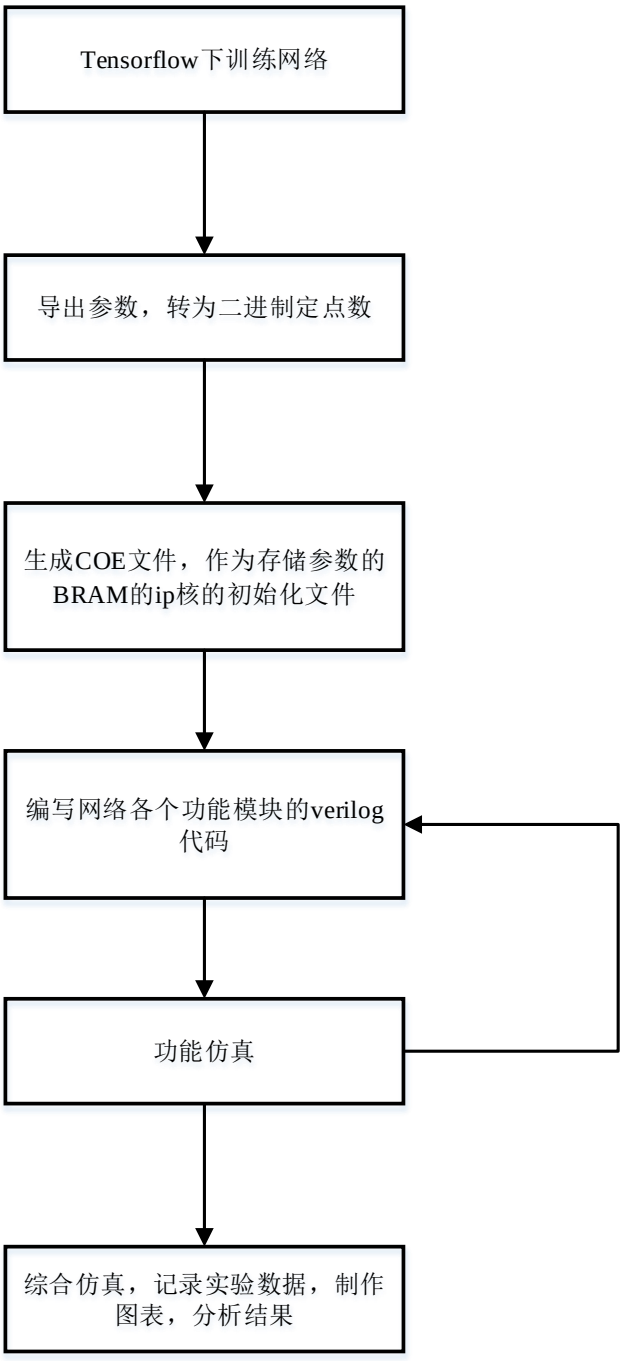

图 4.1 总体流程

4.3 参数定点化处理

在 Tensorflow 下训练模型导出的参数是浮点数，在 FPGA 中实现浮点数的乘法与加法较为复杂而且消耗资源较多，使用定点数运算较为简单且消耗资源较少，而在卷积神经网络的计算中使用定点数代替浮点数，准确率仅下降 0.26%[21]。本次实验中，数据均采用定点数运算。将浮点数转换为定点数的过程如式（4.1），其中x表示浮点数，x_q表示定点数，Q表示定点数小数位宽，int表示取整。

$$x_q = (int)(x * 2^Q) \quad \text{式 4.1}$$

基于参数与输入特征图的取值范围，本次实验中参数与输入特征图均采用 16 位定点数，其中，符号位 1 位，整数位 1 位，小数位 14 位。而定点数的乘法过程如式（4.2），其中Q_x，Q_y，Q_z分别为小数位宽度。

$$z = (x * y) \gg (Q_x + Q_y - Q_z) \quad \text{式 4.2}$$

此次实验中第一层的输出为小数位宽为 13 位的定点数，之后每一层的输出均为小数位宽为 11 的定点数，每一层的权值仍是小数位宽为 14 的 16 位定点数而每层偏置值的小数位宽和每一层输入的小数位宽保持相同，这样在执行定点数加法时，可直接将定点数相加。

4.4 硬件模块的整体架构

实验仿真中生成的 RTL 原理图较为复杂，不易观察，简化的整体架构如图 4.2 所示，weights 模块实际由多个 BRAM 组成，实现并行的给每一层中的计算单元输送权值，而偏置值 bias 由于数量较少，直接在每一层内用寄存器存储。Control 模块的主要作用是产生复位信号，当每一层给出计算全部完成的信号后，control 模块会给出复位信号，每一层会回到初始状态，进行下一轮的计算。层与层之间有两个 buffer 缓冲区以实现乒乓操作，从而在整体上实现了流水线结构，提高吞吐量。其中第一层计算生成数据量较多，故第一层与第二层之间的 buffer 使用了 BRAM 存储。而第二、三、四层生成的数据量较小，故使用了以 LUT 资源实现的 DRAM 来存储数据。

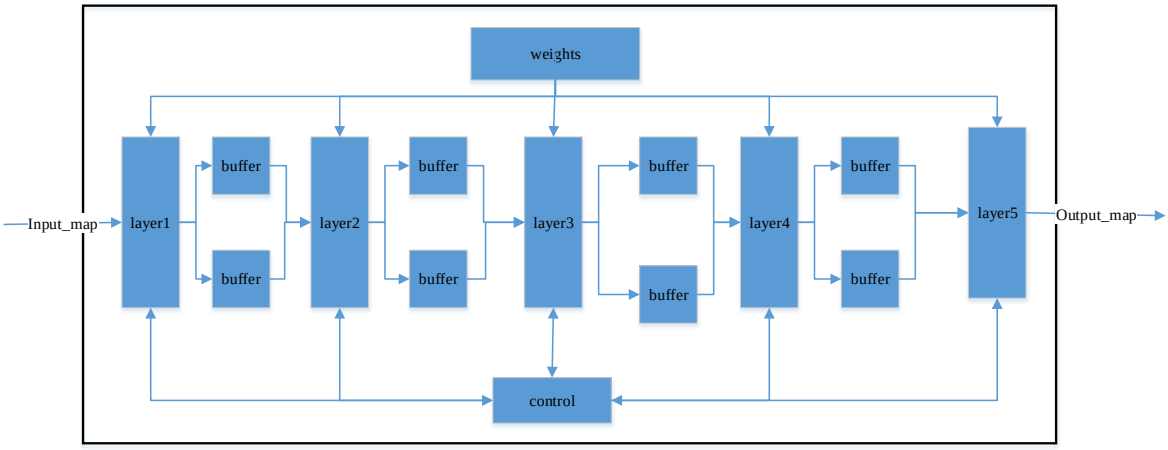

图 4.2 硬件模块的总体架构

4.5 流水线结构

本次实验中，层与层之间使用了两个缓冲区实现乒乓操作，这样可以在整体上实现流水线结构，从而层与层之间可以并行工作。具体过程以第一层与第二层间的交互说明。计算开

始时，第一幅图片的数据输入进第一层后，第一层将计算结果存入 buffer1，所有结果都存入 buffer1 后，第二幅图的数据开始输入第一层，第二幅图的计算结果存入 buffer2，第三幅的计算结果存入 buffer1，后续计算过程以此类推。第一层将第一幅图的输出存入 buffer1 后，buffer1 的数据输入第二层，而此时第一层会将第二图的计算结果存入 buffer2，从而实现第一层的输出和第二层的输入不会有冲突，因此不同层之间可以并行工作，如图 4.3 示。

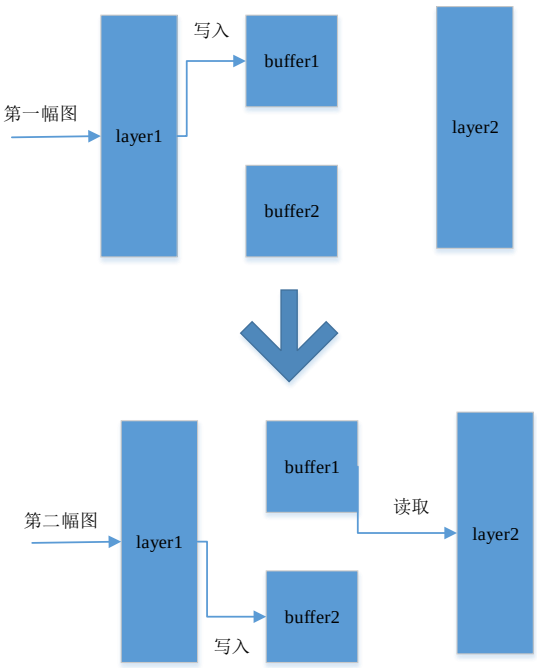

图 4.3 乒乓缓冲

在前向推理阶段，如果第一层的从接受输入到完成计算的时间记为 T，后续每一层的计算时间都小于 T，第一幅图会在 0T-5T 内被计算完成，第二图的计算会在 1T-6T 内完成，第三幅在 2T-7T 内计算完成，时序图如图 4.4 所示。图中 L1,L2,L3,L4,L5 表示当前的计算过程处于第几层。使用流水线结构能极大的提高计算效率，当处理的图片足够多时，使用五级流水线结构的耗时是不使用流水线结构的五分之一。

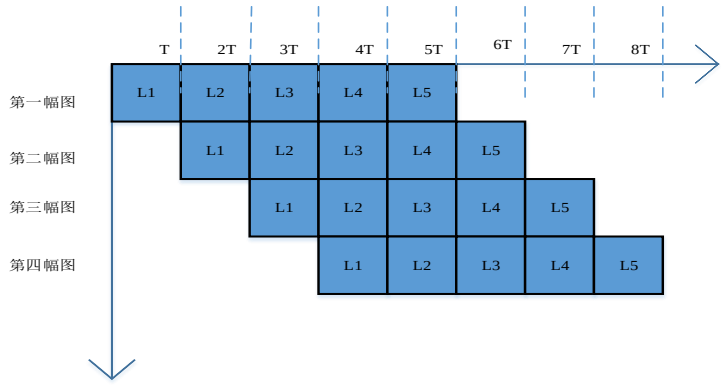

图 4.4 流水线时序图

4.6 参数分布式存储

由于单个 BRAM 一个时钟周期内只能输出一个 18 位数据，而卷积网络中的参数众多，所以如果将参数连续存储在 BRAM 中，需要较长的时间才能全部输入进计算单元中，所以可以将参数分布在不同的 BRAM 中，从而实现权值更快的输入至计算单元中。以第二层为例，共有 16 个 5*5*6 的三维卷积核，从而计算单元 PE 有 96 个，每个计算单元实现卷积核为 5*5*1 的二维卷积运算。实验中该层的参数使用 6 个 BRAM 存储，每个 BRAM 的输出端口与 16 个计算单元相连，第 1 到第 25 个时钟周期内，BRAM 的输出数据会被输入至第一个计算单元，第 26 到第 50 个时钟周期内 BRAM 的输出会被输入至第二个计算单元中，最终，400 个时钟后，6 个 BRAM 可以将权值输入至 96 个计算单元中，由于第一幅图的数据在 784 个时钟周期才全部输入第一层，所以在第二层开始计算时，权值已经全部被加载到计算单元中。

4.7 FIFO 缓存

第二层的 96 个计算单元如果记为PE_i（$1 \le i \le 96$），PE_j，PE_{j+16}，PE_{j+32}，PE_{j+48}，PE_{j+64}，PE_{j+80}的输出结果要经过加法器求和，在实际的计算过程中，这些计算单元的输出并不一定能在同一个时钟周期内到达加法单元，为了能使计算单元的计算的卷积结果同步的输入加法器，在计算单元和加法单元之间加入 FIFO 作为缓存，当每个 FIFO 移入的数据个数都非 0 时，就从 FIFO 中取出一个数据输入加法器，加法单元将六个输入相加后，再加上偏置值 bias，经过整流滤波函数后，输入最大池化层，如图 4.5 所示。而在第二层中由于存在 96 个卷积层计算单元，所以相同的结构还有 16 个。

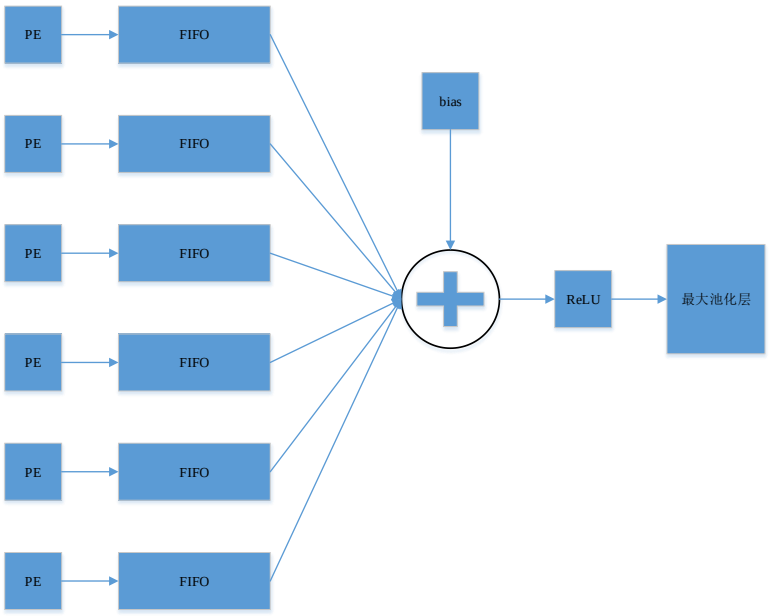

图 4.5 FIFO 缓存同步输入

5 仿真结果分析

在基于第三章和第四章的设计方式，编写 verilog 代码实现相应的功能模块后，由于网络中第一层与与第二层功能与原理相同，第三、四、五层的功能和原理相同，所以分别编写了第一层，第三层和整个网络的 testbench 测试文件，进行功能仿真，并记录计算结果，与 CPU 端的计算结果进行了比较，最后在确保计算结果正确的前提下，进行了前仿和后仿，记录消耗的资源、计算完成的时间和功率。

5.1 开发平台和实验环境

此次研究中，神经网络的训练在 CPU 端完成，型号为 Intel Core i5-6300 HQ(4 core)，频率为 2. 30Ghz, 使用了 TensorFlow2.0 开发框架，编程语言为 python 3.5。而部署到 FPGA 的计算模块的开发环境为 vivado 2016.3，编程语言为 verilog2001, 仿真软件为 vivado XSim2016.3，仿真的开发板为 VC709, 内部 DSP 有 3600 个，LUT 资源 433200 个，FlipFlops 共有 866400 个，1470 块 BRAM。

5.2 第一层的仿真

使用 vivado 综合工具生成的第一层的 RTL 原理图如图 5. 1 所示，仿真时由于图像的数据和权值是从 BRAM 中读取，而 BRAM 有两个时钟周期的读延迟，所以将整个电路的使能信号 enable 接入了模块 delay2，该模块可以将输入的信号延迟两个时钟周期输出，这样可以保证当计算单元的使能信号为 1 时，图像数据和权值均刚好到达计算单元。图中 layer1_conv_PE 是第一层的卷积计算单元，layer1_maxpool 是第一层的最大卷积计算单元，因为 ReLU 函数的实现较为简单，所以在 layer1_conv_PE 中已经实现了该功能。

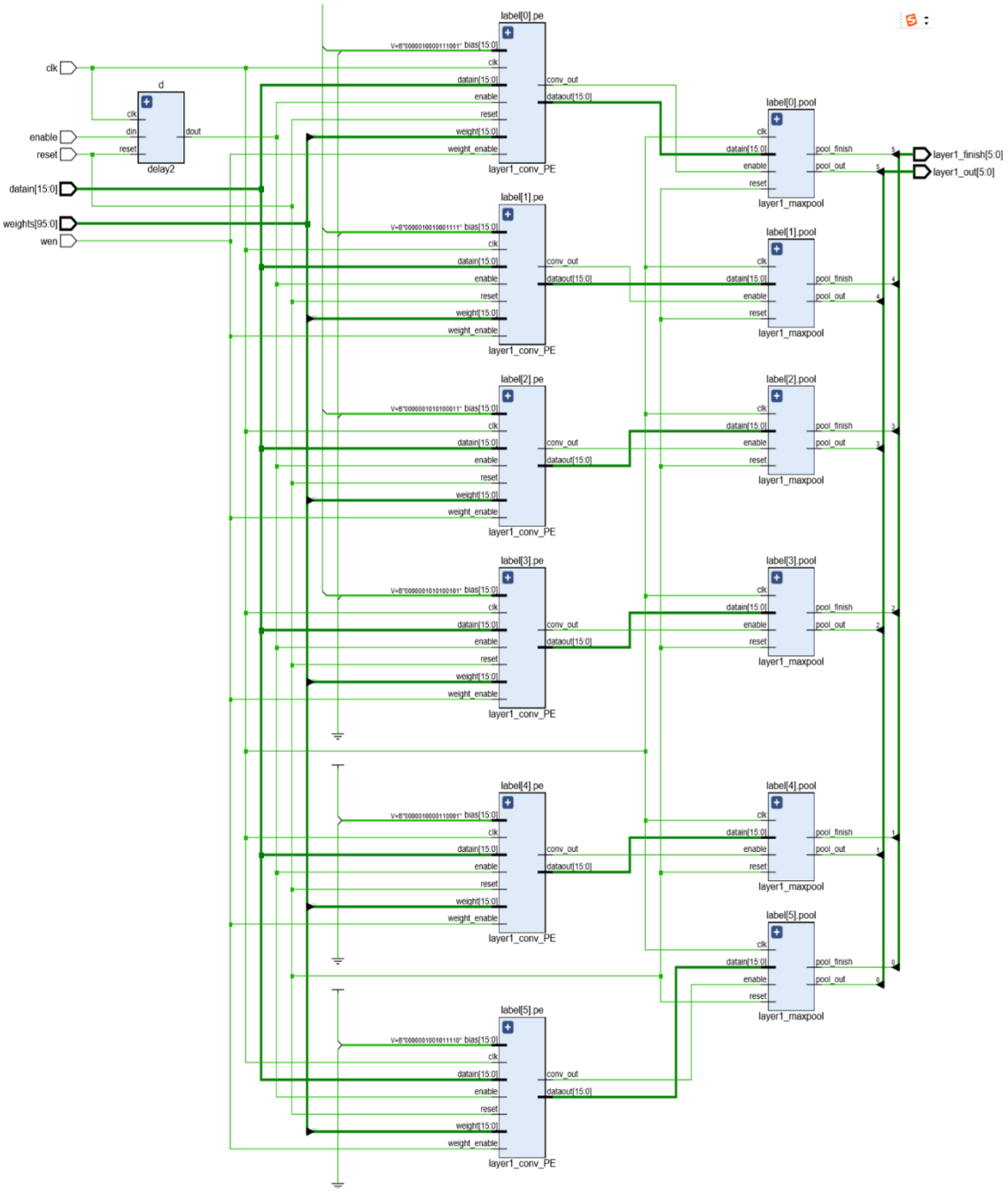

图 5.1 第一层的 RTL 原理图

编写第一层的测试模块，生成的波形图如图 4.8 所示。datain 信号表示输入图像的数据，weights 信号表示权值，每个时钟周期分别给六个 PE 传输一个权值，25 个时钟周期后，能将权值全部输入计算单元中，而计算单元从预填态转变到右移态需要 145 个时钟周期，可以确保计算开始时，权值已经全部传输到计算单元中。wen 是权值输入使能信号，为 1 时表示当前输入权值是有效值，会被计算单元保存在寄存器组中，en 为第一层的使能信号，pool_out 为 1 时表示，输出一个池化操作的结果，pool_finish 表示池化层计算全部完成,l1_dout 表示池化层六个计算单元的结算结果。layer1_reset 是第一层的复位信号，当

pool_finish 信号为 1 时，将 layer1_reset 信号置 1,第一层状态复位,开始下一张图的计算。

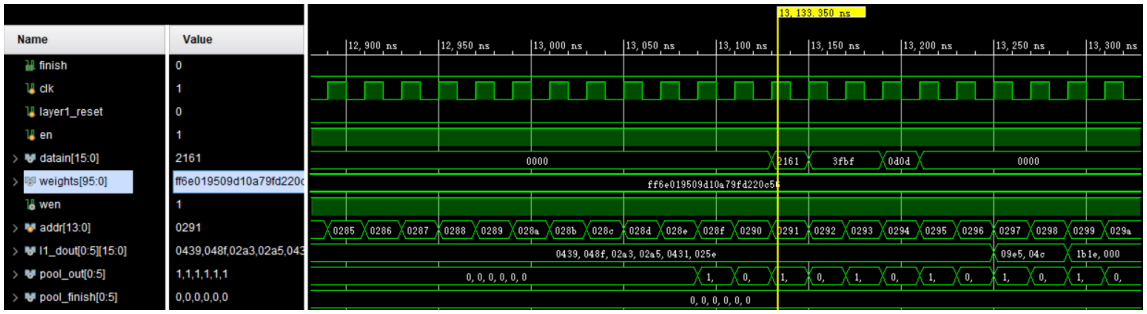

图 5.2 第一层仿真波形图

第一层的仿真模块消耗的资源如表 5.1 所示，一共使用了 6 个 36KB 的 BRAM 和 1 个 18KB 的 BRAM，其中 10 幅输入图像使用了 5 个 36KB 的 BRAM，第一层的权值使用了 1 个 36KB 的 BRAM 和 1 个 18KB 的 BRAM。可以看出卷积层的部署方法会消耗较多的 DSP，因此，在 DSP 资源有限时，可以使用 LUT 资源组成的乘法器代替 DSP。

表 5.1 第一层仿真消耗资源

资源	使用	可用	利用率
LUT	2574	433200	0.59%
LUTRAM	672	174200	0.39%
BRAM	6.50	1470	0.44%
DSP	150	3600	4.17%
FF	6411	866400	0.74%
IO	1	720	0.14%
BUFG	1	32	3.13%

在时钟频率为 50Mhz 的条件下，第一层经过 16.99μs，输出第一幅图片的所有计算结果，由于仿真中数据存放在片内的 BRAM，所以不包括从片外数据传输到片内的时间和功耗。第一层仿真结果与 CPU 计算结果的平均误差为 0.0003，仿真使用 16 位定点数运算与 CPU 使

用 32 位浮点数的计算误差较小，而卷积神经网络本身是对数据精度要求较低的，使用 16 位定点数可以满足要求，此外第一层结果的整数部分最大为 2，在第一层的计算结果使用 2 位整数位，13 位小数位的数据范围已经足够。

在设计约束文件中添加时钟约束条件 create_clock -period 20.000 -name clk -waveform {0.000 10.000} [get_ports clk]后，在 50Mhz 的时钟频率下，功率为 0.473 W，而设备的静态功耗为 0.326W，设备的动态功耗为 0.147W 其中 DSP 占了主要功耗。

5.3 第三层的仿真

第三层是将一个 1*256 的向量与 256*120 的矩阵相乘，得到的 1*120 的向量再与 1*120 的向量相加，结果作为输出。仿真波形图如图 5.3 所示。layer3_finish 信号表示第三层计算完毕，layer3_enable 为 1 时，第三层开始工作。layer3_dout 为第三层计算的 1*120 向量，layer3_save 信号会传递给第三层第四层之间的 buffer，为 1 时 buffer 则存储 layer3_dout。在 50Mhz 的时钟下，第三层从开始接收数据到计算完成用了 5.17μs。

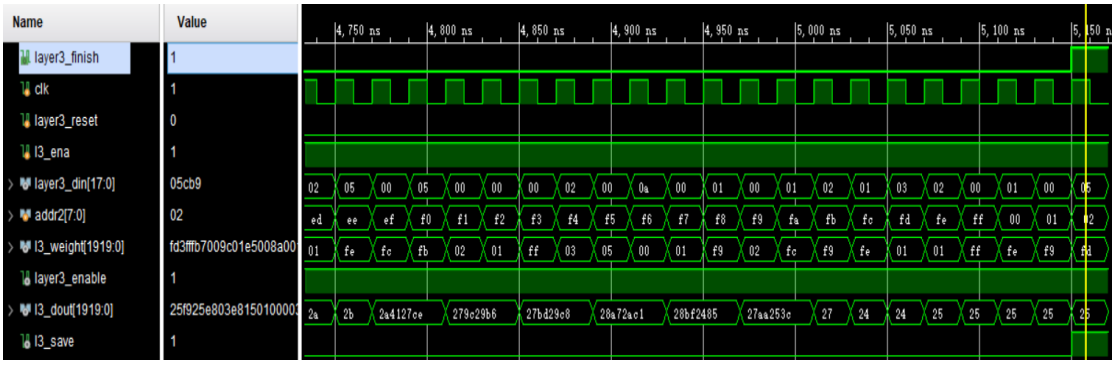

图 5.3 第三层功能仿真波形图

仿真计算出的 120 个结果与 CPU 端计算的结果的平均误差为 0.0364，比卷积层误差较多，随着全连接层的增多，误差会加大。

第三层的仿真文件消耗的资源如表 5.2 所示，其中权值的存储使用了 25 块 BRAM，2.5 个 BRAM 用于存储仿真输入的图片，第三层的参数分布在 25 块 BRAM 中，实际未填满。每个全连接层计算单元使用一个 DSP。

表 5.2 第三层测试消耗资源

资源	使用	可用	利用率
LUT	9292	433200	2.14%

FF	6959	866400	0.80%
BRAM	27.50	1470	1.87%
DSP	120	3600	3.33%
IO	17	720	2.36
BUFG	1	32	3.13%

总功率为 0.586W，其中静态功率为 0.328W，动态功耗为 0.258W，相较与卷积层，全连接层的参数较多，从 BRAM 中读取参数的功耗较大，DSP 功率为 0.064W，BRAM 功率为 0.057W。

5.4 整个网络的仿真

对整个 LeNet 仿真的波形图如图 5.4 所示，index 信号为概率最高的数字的维度，map_in 是输入的图片数据，layer1_finish 为第一层计算完成信号，每次 layer1_finish 为 1，则将 addr 信号改为下一幅图的起始地址，finish 信号为 1 一张图的计算全部完成，从波形图可知第一幅图在 50MHz 的时钟下，经过 30.05μs识别完成，识别结果是 7。

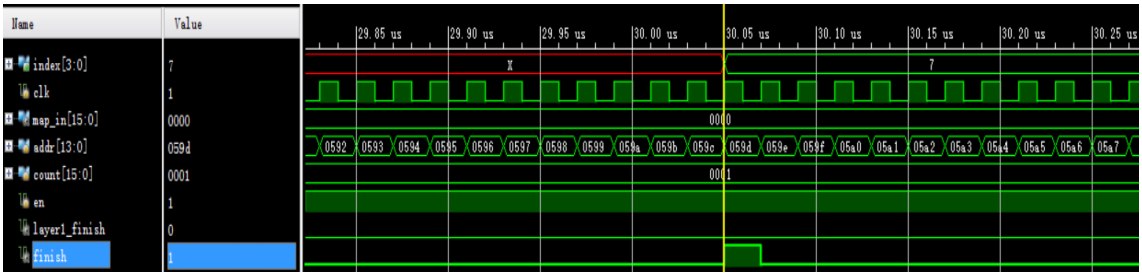

图 5.4 整体仿真波形图

整个网络消耗的资源如下表 5.3 所示。

表 5.3 整个网络消耗资源

资源	使用	可用	利用率
LUT	63089	433200	14.56%

LUTRAM	8929	174200	5.13%
FF	123234	866400	14.22
BRAM	105	1470	7.14%
DSP	2764	3600	76.78%
IO	24	720	3.33%
BUFG	1	32	3.13%

整个网络的功率的功率为 2.794W，动态功耗为 2.442W，静态功耗为 0.352W，其中，用于乘法计算的 DSP 占据了大部分功耗，为 1.435W。

基于以上仿真结果，和 CPU 端的处理时间对比如表 5.4 所示，整个网络生成的 1*10 的向量和 CPU 端生成的 1*10 的向量的平均误差为 0.4188。结合第一层、第三层的仿真结果可知，使用定点数进行前向推理时，卷积层的误差较小，全连接层的误差稍大，随着全连接层变多，误差会变大，所以可以考虑仅将卷积层和池化层部署在 FPGA 上，而全连接层部署在 CPU 端或者 GPU 端。

表 5.4 计算用时对比

处理对象	CPU 用时/us	FPGA 用时/us	FPGA 功率/w	结果误差
第一层	91.43	16.99	0.473	0.0003
第三层	21.22	5.17	0.586	0.0364
整个网络	124.93	30.05	2.442	0.4188

由于本次实现过程中，每一层都采用了输入并行度和输出并行度最大的实现方案，所以加速效果很明显。但是消耗的 DSP 资源较多，所以可以通过降低并行度的方式减少资源消耗，例如第二层的输入并行度为 6，输出并行度为 16，96 个计算单元消耗了 2400 个 DSP，可将输出并行度改为 4，则可将 DSP 消耗数减少至 600 个，第二层的处理时间理论上变为原来的 4 倍，相比与 CPU 仍能显著提高计算速度。