**数字逻辑与数字系统**



**课程设计报告**



题目：基于LeNet-5的手写体数字识别

**学 院 求是学部**

**专 业 计算机科学与技术**

**年 级 2019级**

**姓 名 李自安**

2022 年 1 月 7 日

目录

[一、报告摘要](#_Toc1545879946_WPSOffice_Level1) [3](#_Toc1545879946_WPSOffice_Level1)

[二、设计方案](#_Toc1473924309_WPSOffice_Level1) [4](#_Toc1473924309_WPSOffice_Level1)

[2.1 LeNet-5网络准备](#_Toc1473924309_WPSOffice_Level2) [4](#_Toc1473924309_WPSOffice_Level2)

[2.1.1 LeNet-5网络架构](#_Toc1473924309_WPSOffice_Level3) [4](#_Toc1473924309_WPSOffice_Level3)

[2.1.2 LeNet-5的python（pytorch库）实现](#_Toc1021993218_WPSOffice_Level3) [5](#_Toc1021993218_WPSOffice_Level3)

[2.1.3 LeNet-5网络参数训练](#_Toc1065806220_WPSOffice_Level3) [7](#_Toc1065806220_WPSOffice_Level3)

[2.2 “基于LeNet-5的手写体数字识别模型”的设计](#_Toc1021993218_WPSOffice_Level2) [7](#_Toc1021993218_WPSOffice_Level2)

[2.2.1 模型设计框架](#_Toc844039913_WPSOffice_Level3) [7](#_Toc844039913_WPSOffice_Level3)

[2.2.2 “基于LeNet-5的手写体数字识别模型”顶层模块原理图](#_Toc1649329356_WPSOffice_Level3) [8](#_Toc1649329356_WPSOffice_Level3)

[2.2.3 LeNet-5网络参数定点化设计（binary scaling）](#_Toc559570816_WPSOffice_Level3) [9](#_Toc559570816_WPSOffice_Level3)

[2.2.4 LeNet-5网络模型模块原理图设计](#_Toc875814299_WPSOffice_Level3) [10](#_Toc875814299_WPSOffice_Level3)

[2.2.5 LeNet-5网络各层状态机设计](#_Toc958006755_WPSOffice_Level3) [14](#_Toc958006755_WPSOffice_Level3)

[2.2.6 LeNet-5网络数据通路设计](#_Toc1534629726_WPSOffice_Level3) [17](#_Toc1534629726_WPSOffice_Level3)

[2.2.7 LeNet-5网络缓存（BRAM）设计](#_Toc1243204412_WPSOffice_Level3) [18](#_Toc1243204412_WPSOffice_Level3)

[三、设计结果](#_Toc1021993218_WPSOffice_Level1) [19](#_Toc1021993218_WPSOffice_Level1)

[3.1 待识别图片准备](#_Toc1065806220_WPSOffice_Level2) [19](#_Toc1065806220_WPSOffice_Level2)

[3.2 FPGA仿真结果](#_Toc844039913_WPSOffice_Level2) [20](#_Toc844039913_WPSOffice_Level2)

[3.3 FPGA板级验证结果](#_Toc1649329356_WPSOffice_Level2) [23](#_Toc1649329356_WPSOffice_Level2)

[四、 所遇到的困难和解决方法](#_Toc1065806220_WPSOffice_Level1) [25](#_Toc1065806220_WPSOffice_Level1)

[1. FPGA板的LUT资源过少，导致布线过程中报错。](#_Toc559570816_WPSOffice_Level2) [25](#_Toc559570816_WPSOffice_Level2)

[2. 神经网络前向传播过程过于复杂，难以进行正误检验与调试。](#_Toc875814299_WPSOffice_Level2) [26](#_Toc875814299_WPSOffice_Level2)

[五、 未来改进思路](#_Toc844039913_WPSOffice_Level1) [26](#_Toc844039913_WPSOffice_Level1)

[1. 待预测图片的传入方式的改进。](#_Toc958006755_WPSOffice_Level2) [26](#_Toc958006755_WPSOffice_Level2)

## 一、报告摘要

本文主要介绍了“基于LeNet-5的手写体数字识别模型”的设计、搭建、仿真与验证的过程。

首先对本设计中LeNet-5的基本网络架构以及利用python语言对网络模型的搭建过程进行了简要的介绍。此后，利用Mnist数据集对模型进行了训练，得到了精度较高的浮点数参数。在将网络模型参数写入到FPGA的过程中，我们首先对网络参数进行了定点化处理，此后调用IP核完成了待识别图片以及参数数据的存储。我们采用状态机的方式对各层的计算模型以及整体的控制模型进行了详细设计与搭建。通过复用25维乘加器与比较器，完成了各层的计算与采样，提高了代码的复用率以及运行效率。通过调用不同类型的GPIO，得到了一个精度较高、功能较为完善的手写体数字识别板级模型。最后，通过输入具有不同特征的图片进行模型检验，验证了模型的泛化能力与抗噪声能力，完成了仿真与板级验证。

在报告的最后，总结了所遇到的困难与解决方法，并且提出了部分改进思路。

## 二、设计方案

### 2.1 LeNet-5网络准备

2.1.1 LeNet-5网络架构

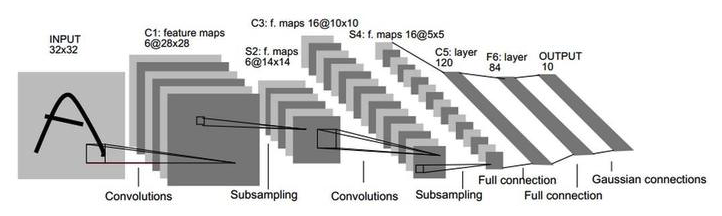


图 1 LeNet-5架构图

LeNet-5出自论文《Gradient-Based Learning Applied to Document Recognition》，是一种较为简单的CNN模型，然而对于手写数字的识别十分高效。

LeNet-5共包括7层，分别是：C1层-卷积层，S2层-池化层，C3-卷积层，S4层-池化层，C5层-卷积层，F6层-全连接层，OUTPUT层-输出层。

C1层的输入为一个通道的灰度图，通过卷积得到6张特征图；C3层输入为6个通道的特征图，通过卷积得到16张特征图；C5层输入为16个通道的特征图，通过卷积得到120张特征图（每张特征图仅包含一个神经元），**在本文的实现中，每一个卷积层后都通过ReLu函数激活。**

S2、S4层均为下采样层，在论文中的采样方式是：4个输入相加，乘以一个可训练参数，再加上一个可训练偏置，结果通过sigmoid，**在本文的实现中，均采用Max Pooling的方式，即取采样区域的最大值。**

F6层与OUTPUT层均为全连接层，**本文的python实现中，F6层采用ReLu激活，OUTPUT层采用sigmoid函数激活。**

**需要注明的是，在论文的实现中，C3层通过一种随机的卷积组合方式对S2层的输出结果进行卷积，论文作者解释这样做的原因是：1）减少参数，2）这种不对称的组合连接的方式有利于提取多种组合特征。在本文中，我们对其进行了简化，将C3层的每一通道与S2层的全部特征图相连接，所得到的结果精确度满足要求。**

各层参数直观说明如下：

表 1 LeNet-5网络参数

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 卷积核大小 | 采样区域 | 输入 | 输出 | 可训练参数 |
| C1层 | 5\*5 | - | 1\*32\*32 | 6\*28\*28 | (5\*5+1)\*6 |
| S2层 | - | 2\*2 最大池 | 6\*28\*28 | 6\*14\*14 | 0 |
| C3层 | 5\*5 | - | 6\*14\*14 | 16\*10\*10 | (5\*5\*6+1)\*16 |
| S4层 | - | 2\*2 最大池 | 16\*10\*10 | 16\*5\*5 | 0 |
| C5层 | 5\*5 | - | 16\*5\*5 | 120\*1\*1 | (5\*5\*16+1)\*120 |
| F6层 | - | - | 120\*1\*1 | 84\*1\*1 | (120+1)\*84 |
| OUTPUT层 | - | - | 84\*1\*1 | 10\*1\*1 | (84+1)\*10 |

2.1.2 LeNet-5的python（pytorch库）实现

为了能够训练得到LeNet-5网络模型的各层参数，首先利用python实现LeNet-5模型及其训练、评估函数。网络模型如下：

1. **class** LeNet5(nn.Module):
2. **def** \_\_init\_\_(self):
3. super(LeNet5, self).\_\_init\_\_()
4. self.c1 = C1()
5. self.c2 = C2()
6. self.c3 = C3()
7. self.f4 = F4()
8. self.f5 = F5()
9. **def** forward(self, img):
10. output = self.c1(img)
11. output = self.c2(output)
12. output = self.c3(output)
13. output = output.view(img.size(0), -1)
14. output = self.f4(output)
15. output = self.f5(output)
16. **return** output

其中，C1、C2、C3、F4、F5层分别对应LeNet-5模型中的三个卷积层和两个全连接层，且S2、S4两个池化层均包含在了C1与C2所定义的网络中。

C1、C2、C3、F4、F5网络模型均采用Pytorch所提供的API生成。

卷积层中，仅展示C1层的实现，其他层类似：

1. **class** C1(nn.Module):
2. **def** \_\_init\_\_(self):
3. super(C1, self).\_\_init\_\_()
4. self.c1 = nn.Sequential(OrderedDict([
5. ('c1', nn.Conv2d(1, 6, kernel\_size=(5, 5))),
6. ('relu1', nn.ReLU()),
7. ('s1', nn.MaxPool2d(kernel\_size=(2, 2), stride=2))
8. ]))
9. **def** forward(self, img):
10. output = self.c1(img)
11. **return** output

全连接层中，仅展示F4层的实现，F5类似：

1. **class** F4(nn.Module):
2. **def** \_\_init\_\_(self):
3. super(F4, self).\_\_init\_\_()
5. self.f4 = nn.Sequential(OrderedDict([
6. ('f4', nn.Linear(120, 84)),
7. ('relu4', nn.ReLU())
8. ]))
10. **def** forward(self, img):
11. output = self.f4(img)
12. **return** output

这里，每一层（除去输出层）均采用ReLU函数激活。

2.1.3 LeNet-5网络参数训练

为了能够精确识别手写数字图片，我们需要对网络进行训练。

训练设置如下：

表 2 LeNet-5 网络训练超参数设置

|  |  |
| --- | --- |
| 网络模型 | LeNet-5 |
| 训练与测试数据集 | Mnist开源手写数字集 |
| 损失函数 | CrossEntropyLoss交叉熵 |
| 优化算法 | Adam算法 |
| batch\_size | 256 |
| lr | 2e-3 |
| epoch | 16 |

其他超参数均采用默认或论文中相同的参数设置。

训练得到参数类型为dtype = float32，**为了能够在FPGA中尽可能节省运算与存储资源，应当对参数进行定点化。这里我们采用训练后定点化的方法，定点化过程见2.2.3节。**

训练过程如下（部分）:

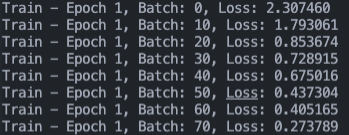


图 2 训练过程（部分）

训练结果如下，可以看到在float32参数类型下，手写图片的识别精度达到了98.99%：



图 3 训练结果评估

### 2.2 “基于LeNet-5的手写体数字识别模型”的设计

2.2.1 模型设计框架

本文主要通过FPGA设备实现手写数字的识别与判断，主要设计思路如下：

1. 将待识别的图片（图片数目不限，本文预先存储了20张图片）的灰度图像素进行定点化，预先存储在FPGA的**BRAM（IP核）**中，等待识别；
2. 将训练精度较高且定点化的参数存储在**BRAM（IP核）**中，供LeNet-5调用；
3. 设置一个用以存储中间结果的**BRAM（IP核）**，将各层之间相互独立；
4. 利用verilog/system verilog语言编写LeNet-5模型，**为了能够减少LUT资源的使用，通过状态机的方式来实现LeNet-5模型，以减少大量组合逻辑电路的生成**。（**事实上这是一种tradeoff，需要通过大量的时钟周期才能够完成模型的前向传播，不过好处是减少了FPGA资源的占用，可以应用在嵌入式设备中**）。
5. 设置一系列的逻辑电路，使得可以通过**GPIO**进行**待识别图片的选择**、**模型前向传播进度的标识**（主要通过**LED灯**实现，若某一LED灯亮起，表示其对应的层finish）、**识别正误的标识**（**通过三色LED等实现**）。

2.2.2 “基于LeNet-5的手写体数字识别模型”顶层模块原理图

“基于LeNet-5的手写体数字识别模型”顶层模块原理图如下所示。

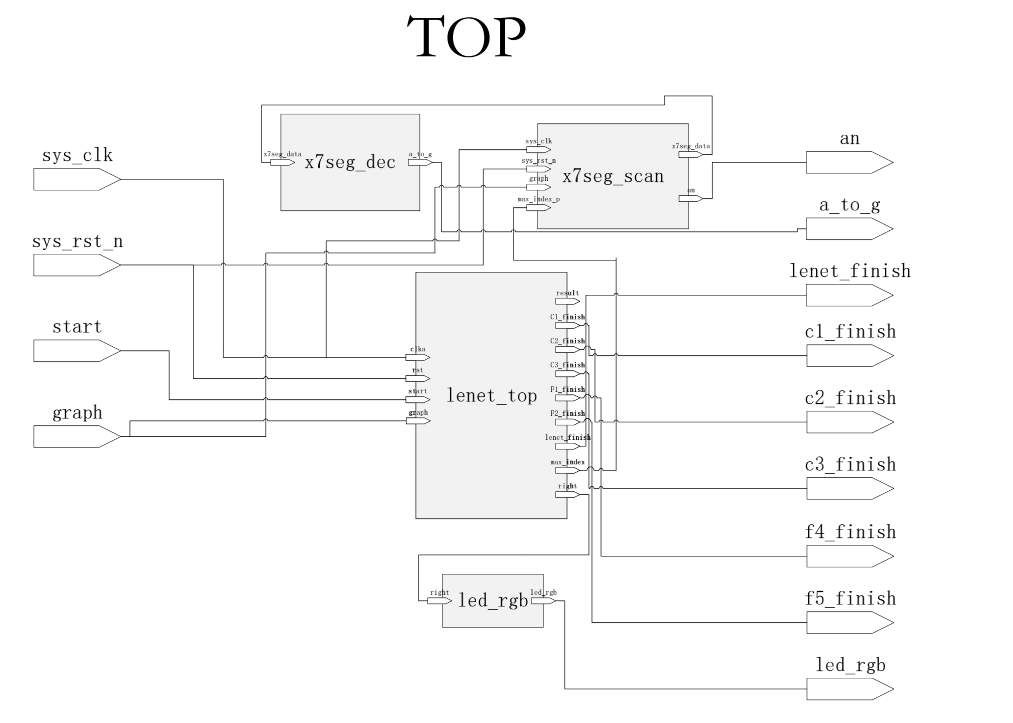


图 4 “基于LeNet-5的手写体数字识别模型”顶层模块原理图

顶层模块TOP共包含四个子模块，分别是**LeNet-5网络模型模块lenet\_top、动态扫描电路模块x7seg\_dec和x7seg\_scan，三色LED灯显示模块led\_rgb**。

**lenet\_top模块**负责图片的前向传播、数值预测以及所有数据的存储（即三个RAM均包含在lenet\_top中）。**该模块的端口介绍以及模块内部的详细介绍均见于2.2.4节。**

**动态扫描电路模块**通过七段数码管来显示预测数值和图片的正确值（即图片的label）。x7seg\_scan模块以1KHZ的频率进行动态电路的扫描，并且在8个数字的**左侧**四个数字处显示图片的label值，在8个数字的**右侧**四个数字处显示图片的预测值。

**led\_rgb模块**负责根据lenet\_top预测的正误来控制三色LED灯的亮起。具体来说：当lenet\_top尚未预测结束时，三色LED灯为蓝色；当lenet\_top预测正确时，三色LED灯为绿色；当lenet\_top预测错误时，三色LED灯为红色。

2.2.3 LeNet-5网络参数定点化设计（binary scaling）

为了能够在FPGA中利用整数乘法代替浮点数的乘法，我们将python模型训练得到的32位浮点数float32进行定点化，**转换为16位的定点数FLOAT16**，具体格式如下：

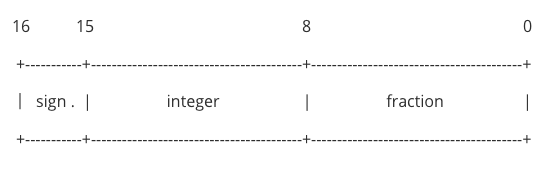


图 5 16位定点数格式

这里，我们的定点数共16位，其中小数部分为8位，整数部分为7位，符号位为1位。

因此，任意一个数a，为了得到它的定点数表示A，需要进行如下转换：

/private/var/folders/0z/7g4kggwd51ng0crfdpfzj2240000gn/T/com.kingsoft.wpsoffice.mac/wpsoffice.CDEgFZwpsoffice

此处的取整为**向0取整**，即直接舍弃乘法结果的小数部分。

在进行定点数的计算时，可以证明，加减法是等效的，即：

wpsoffice

其中trans()是浮点数到定点书的转换函数。

但是乘法需要进行移位操作，如下：

wpsoffice

这一点会在RTL级编程中的乘加器中体现，即**乘法器并不是简单的对两个数进行相乘，而是对其乘法结果进行了移位操作，详见于2.2.6节25维乘加器的设计。**

为了将定点数进行存储，我们将其定点数转换成补码形式，存储在IP核中。从浮点数到定点数的十六进制表示是利用python实现的，大致如下：

1. **def** trans(a):
2. str = hex(int(a\*pow(2, 8)) & 0xffff)[2:]
3. lens = len(str)
4. strnew = "0" \* (4 - lens) + str
5. **return** strnew

其中strnew即a对应的定点数的十六进制表示。

关于参数文件的读取与转化、存储等，**详见源代码paramTrans.ipynb.**

2.2.4 LeNet-5网络模型模块原理图设计

在整个“基于LeNet-5的手写体数字识别模型”中，最重要的模块就是LeNet-5网络模型模块，即下图部分：

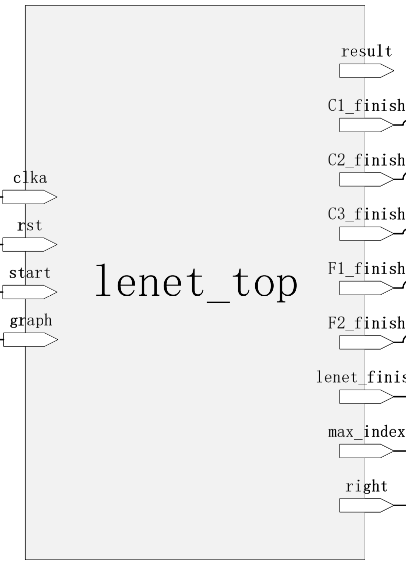


图 6 LeNet-5网络模型模块

该网络模型模块的内部原理图如下所示：

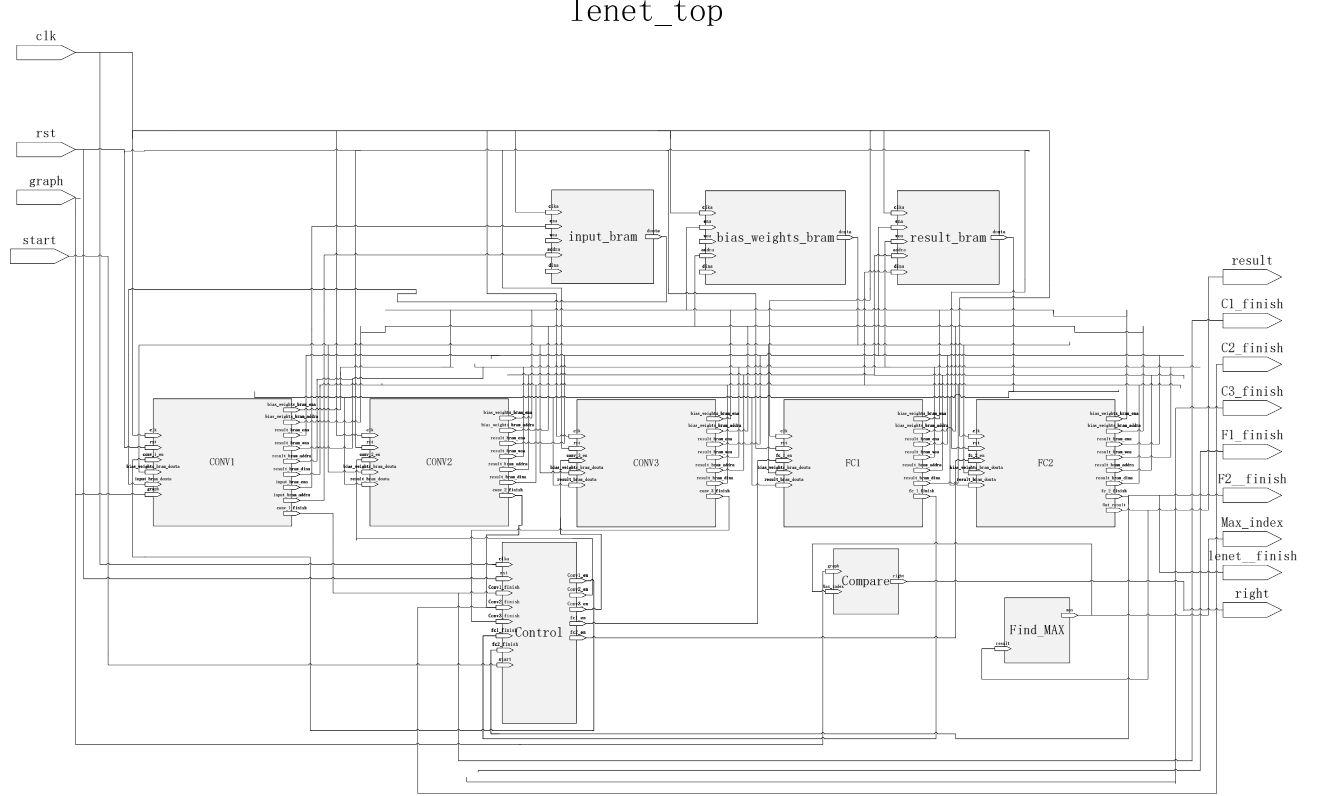


图 7 LeNet-5网络模型模块原理图

（注：该原理图做了一定的简化，包括：1. 没有表示POOL1、POOL2层，**但是由于各层之间在result\_bram和Control模块的作用下相互独立**，因此不影响逻辑表示；2. 各RAM与各层之间的连接并不是直接连接，而是**通过一个控制信号进行选通**，即只有执行到某一个层，才会将该层与对应的RAM之间的连接设为通路，否则断开。这一点是通过**状态机**实现的）。

**端口介绍：**

**lenet\_top模块共有4个输入端口，9个输出端口。各端口功能如下：**

表 3 LeNet-5网络模块端口

|  |  |  |  |
| --- | --- | --- | --- |
| **端口名称** | **端口性质** | **位宽** | **端口介绍** |
| **clk** | **输入端口** | **1** | **时钟信号** |
| **rst** | **输入端口** | **1** | **重置信号** |
| **graph** | **输入端口** | **5** | **选图信号，用以选择进行前向传播的图片。** |
| **start** | **输入端口** | **1** | **开始信号，当start为1时开始前向传播** |
| **result** | **输出端口** | **10\*16** | **输出FC2层计算的结果向量** |
| **C1\_finish** | **输出端口** | **1** | **C1层计算结束后该信号为1，否则为0** |
| **C2\_finish** | **输出端口** | **1** | **C2层计算结束后该信号为1，否则为0** |
| **C3\_finish** | **输出端口** | **1** | **C3层计算结束后该信号为1，否则为0** |
| **F1\_finish** | **输出端口** | **1** | **FC1层计算结束后该信号为1，否则为0** |
| **F2\_finish** | **输出端口** | **1** | **FC2层计算结束后该信号为1，否则为0** |
| **Lenet\_finish** | **输出端口** | **1** | **预测结束后该信号为1，否则为0** |
| **Max\_index** | **输出端口** | **4** | **输出预测数字** |
| **right** | **输出端口** | **2** | **输出预测是否正确，若尚未预测结束，则输出为0；若预测错误，则输出为2；若预测错误，则输出为4。** |

**网络各层模块介绍：**

在LeNet-5网络中，**网络模块**主要包括CONV1、POOL1、CONV2、POOL2、CONV3、FC1、FC2七个模块，分别LeNet-5的七个层。

这里对CONV1层模块进行介绍，其他层的封装与之类似：

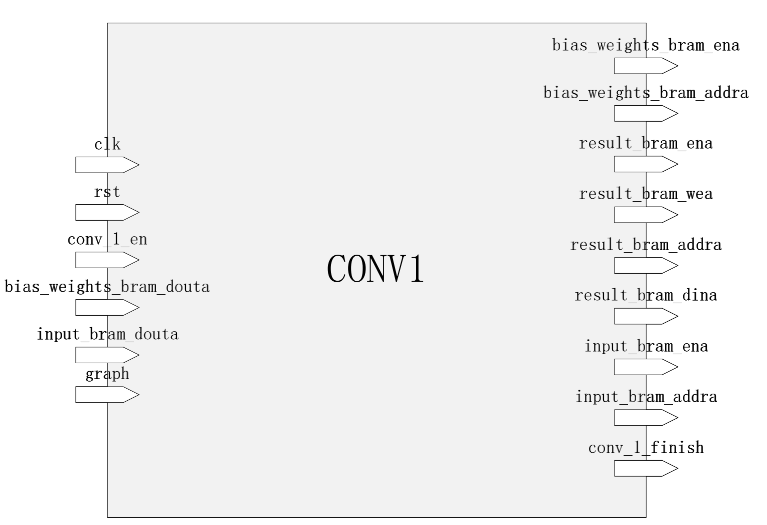


图 8 CONV1层模块

CONV1层包括有6个输入端口和9个输出端口，各端口功能与其命名类似。

其中，clk与rst分别为时钟和重置信号，conv\_1\_en、conv\_1\_finish信号分别为该层的使能信号与结束信号，graph信号为选图信号，在前文中已经介绍。

其余信号均为与RAM的交互信号，包括有**地址传输、数据写入、数据读取、使能信号**等，分别与input\_bram、bias\_weights\_bram、result\_bram三个RAM相连，用以进行该层与这些RAM之间的写入与读取控制信号和数据的传输。

**数据存储模块介绍：**

LeNet-5网络中，**数据存储模块**通过input\_bram、bias\_weights\_bram、result\_bram实现。这三个模块均通过调用BRAM的IP核实现，**且数据位宽均为16位**。BRAM的大小根据需要存储的数据大小决定。

其中，input\_bram用以存储输入图片，这里我们预先存储了20张图。bias\_weights\_bram用以存储模型参数，包括偏置量与权重。result\_bram用以存储运算的中间结果，供后一个层调用作为输入。

以input\_bram为例，如下图，各端口功能与BRAM的IP核相同：

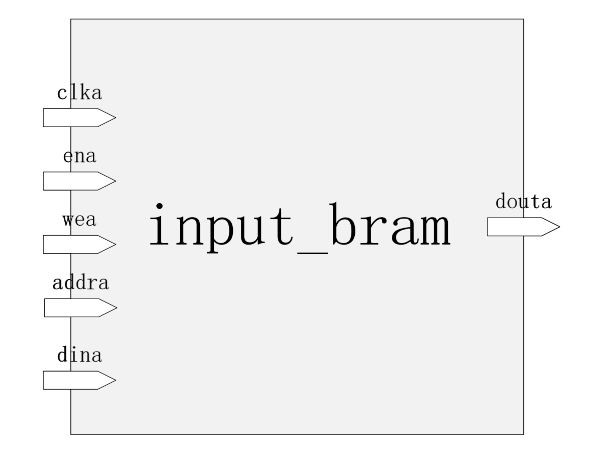


图 9 input\_bram模块

**其他模块介绍：**

LeNet-5中的**控制模块为Control**，用以控制各层模块的使能与暂停。输入为时钟信号与各层的状态信号，输出为各层的使能信号。具体实现是通过状态机的方式。

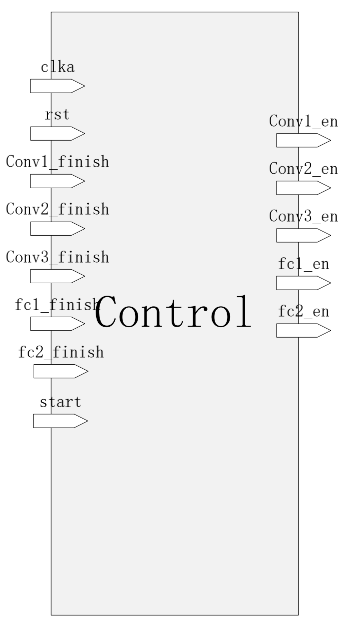


图 10 Control模块

LeNet-5中还存在若干逻辑功能，此处用几个模块来抽象。

Find\_MAX模块用以找出向量中的最大值所对应的index，其输入为FC2（即输出层）的输出结果，输出为结果中最大的值对应的index，即对应的数字。

Compare模块用以进行预测结果和真实值的比较。分别如下：

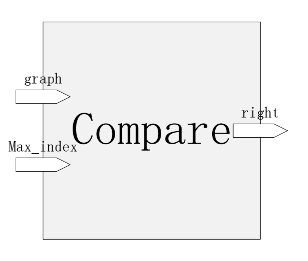


图 11 Compare模块

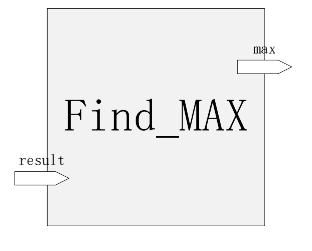


图 12 Find\_MAX模块

2.2.5 LeNet-5网络各层状态机设计

为了能够**减少LeNet-5模型的硬件资源占用**，这里通过状态机的方式进行代码的编写，主要包括以下几个部分：**顶层控制状态机、卷积层状态机、全连接层状态机、池化层状态机**。除此之外，还有若干状态机用以控制RAM的接线情况，即在进行到某一个层的计算时，将RAM与对应的层进行连接。由于状态机与状态转移情况与顶层控制状态机类似，因此不再展示。

**顶层控制状态机：**

顶层控制状态机用以对各层的使能进行控制，在该层完成全部计算并且输出finish信号后，状态进行转移，如下：

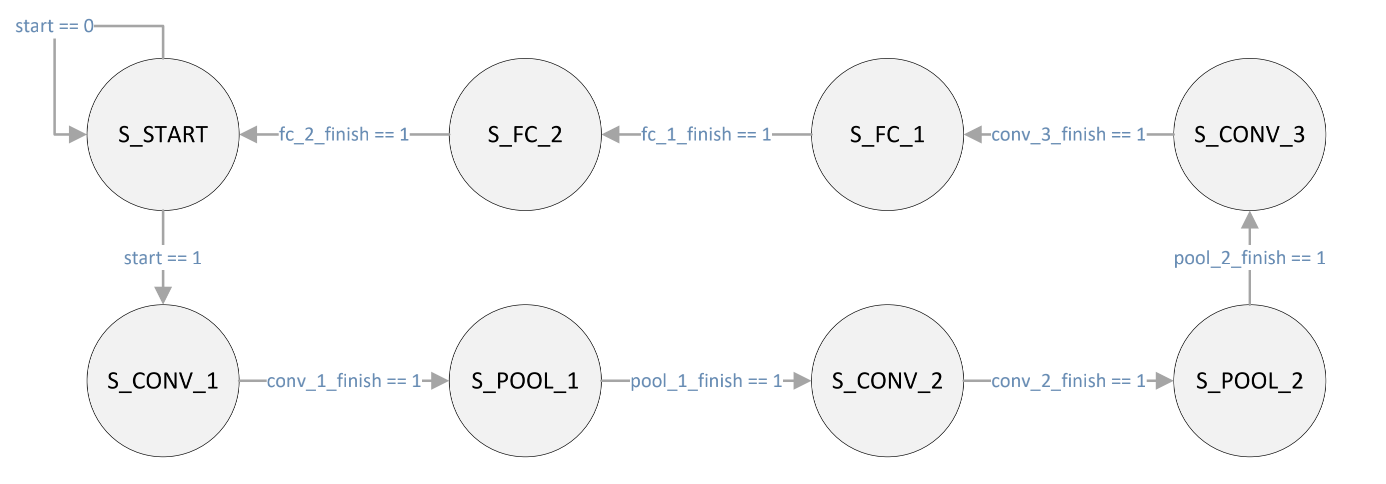


图 13 顶层控制状态机

其中共包括八个状态。除去START状态外，每一个状态对应一个层。

**卷积层状态机：**

卷积层状态机位于三个卷积层当中，用以控制当前计算的步骤，只有当该步骤完成之后才会转移到下一个状态，如下所示：

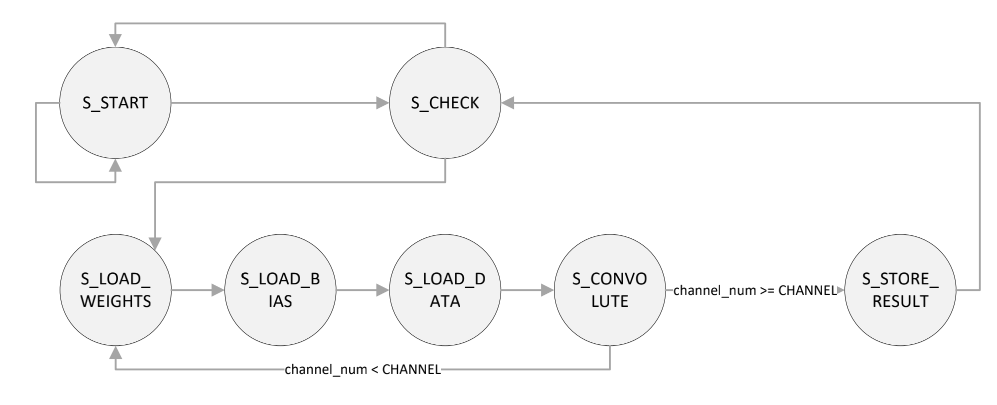


图 14 卷积层状态机

其中，各状态内容如下：

表 4 卷积层状态机状态说明

|  |  |
| --- | --- |
| 状态名称 | 状态内容 |
| S\_LOAD\_WEIGHTS | 从bias\_weights\_bram中取出卷积所需权重 |
| S\_LOAD\_BIAS | 从bias\_weights\_bram中取出卷积所需偏置 |
| S\_LOAD\_DATA | 从result\_bram或input\_bram中取出卷积所需输入 |
| S\_LOAD\_CONVOLUTE | 进行卷积计算 |
| S\_STORE\_RESULT | 将数据存储到result\_bram中 |
| S\_CHECK | 检查该层是否计算完毕，若计算完毕则将finish置1，并转移到START状态，否则转移到S\_LOAD\_WEIGHTS状态。 |

**池化层状态机：**

池化层状态机位于两个卷积层当中，用以控制当前计算的步骤，只有当该步骤完成之后才会转移到下一个状态，如下所示：

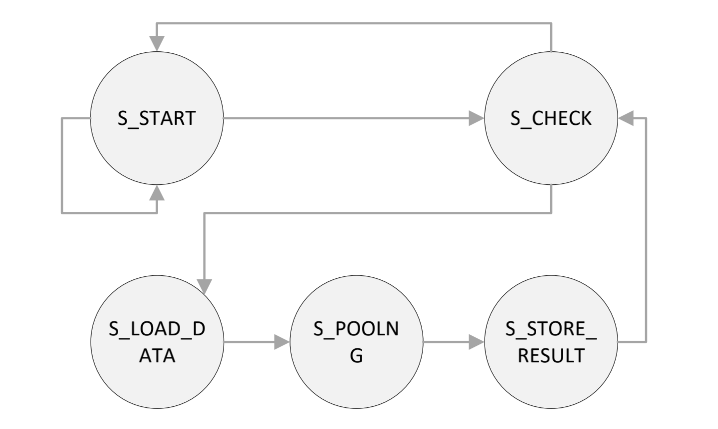


图 15 **池化层状态机**

表 5 池化层状态机状态说明

|  |  |
| --- | --- |
| 状态名称 | 状态内容 |
| S\_LOAD\_DATA | 从result\_bram或input\_bram中取出池化所需输入 |
| S\_POOLING | 进行最大池化采样 |
| S\_STORE\_RESULT | 将数据存储到result\_bram中 |
| S\_CHECK | 检查该层是否计算完毕，若计算完毕则将finish置1，并转移到START状态，否则转移到S\_LOAD\_DATA状态。 |

**全连接层状态机：**

全连接层状态机位于两个全连接层当中，用以控制当前计算的步骤，只有当该步骤完成之后才会转移到下一个状态，如下所示：

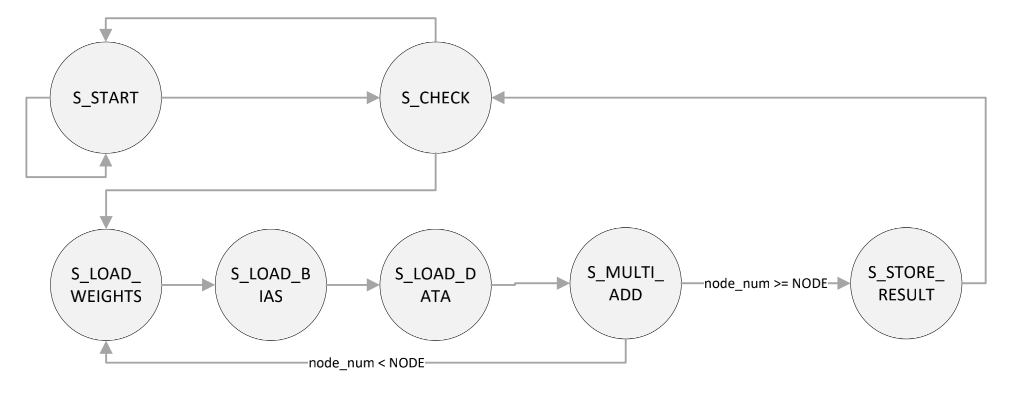


图 16 全连接层状态机

其中，各状态内容如下：

表 6 全连接层状态机状态说明

|  |  |
| --- | --- |
| 状态名称 | 状态内容 |
| S\_LOAD\_WEIGHTS | 从bias\_weights\_bram中取出乘加所需权重 |
| S\_LOAD\_BIAS | 从bias\_weights\_bram中取出乘加所需偏置 |
| S\_LOAD\_DATA | 从result\_bram或input\_bram中取出乘加所需输入 |
| S\_LOAD\_MULTI\_ADD | 进行乘加计算 |
| S\_STORE\_RESULT | 将数据存储到result\_bram中 |
| S\_STORE\_CHECK | 检查该层是否计算完毕，若计算完毕则将finish置1，并转移到START状态，否则转移到S\_LOAD\_WEIGHTS状态。 |

2.2.6 LeNet-5网络数据通路设计

**25维乘加器设计**

由于卷积层的卷积核大小均为5\*5，因此设计该乘加器为25维。其逻辑代码如下所示：

1. **for**(i = 0; i < `LENGTH\*`WIDTH; i = i + 1)
2. begin
3. data\_begin = `DATA\_SIZE \* (`LENGTH \* `WIDTH - i) - 1;
4. item1 = matrix1[data\_begin-:`DATA\_SIZE];
5. item2 = matrix2[data\_begin-:`DATA\_SIZE];
6. midvalue = ({{8{item1[`DATA\_SIZE-1]}},item1[`DATA\_SIZE-1:0]} \* {{8{item2[`DATA\_SIZE-1]}},item2[`DATA\_SIZE-1:0]}) >>> 8;
7. temp = temp + midvalue;
8. end
9. result = temp + bias;

考虑到该处的乘法是通过整数乘法来代替的定点数乘法，并且该定点数存在8位小数位宽，因此在进行乘法之后需要进行8位的数据右移。考虑到位宽限制，在乘法时会因为截断丢失内容，因此首先应当对十六位的数字进行八位扩展。**具体见于上述代码中midvalue的计算方法。**

**需要说明的是：为了提高代码的复用率，除去卷积层外，全连接层也利用该25维乘加器进行乘加。但是考虑到两个全连接层的输入神经元个数并不是25的倍数，因此需要进行神经元的补充，即84个神经元补充为100个，120个神经元补充为125个，多出来的神经元对应的权重设置为0（排除其对输出的影响）。如此一来，便可以通过4次/5次25维乘加器的使用将全连接层的一个输出神经元结果计算完毕。**

**ReLU激活函数设计**

根据激活函数对应的功能，编写代码如下：

1. assign dout = (din[`DATA\_SIZE-1] == 1'b1) ? 16'b0 : din;

**比较器设计** max\_index\_10模块在进行10个数字的比较时，以如下步骤进行比较：

1. 将10个数字分为5组，每一组两两比较，并且输出较大的index与对应的数值；
2. 将前4组中得到的较大数分为2组，每一组两两比较，并且输出较大的index与对应的数值；
3. 将2得到的两个数字进行比较，得到较大的数字再与1中的第5组进行比较，最终输出最大的index。

max\_pool最大池化模块在进行4个数字的比较时，采用类似的分组方法，最终输出较大的数值。

2.2.7 LeNet-5网络缓存（BRAM）设计

在本设计中，共有三个BRAM的IP核，分别是result\_bram（存储中间结果），bias\_weight\_bram（存储各层的参数），input\_bram（存储输入图片），当传入start信号时，各层将按需从各个RAM中读、写数据。

由于各层所需、所写的数据不同，因此对应的读、写数据的基地址也不同，具体分布如下：

1. # bias\_weight\_bram
2. conv1.weights 0  （共150个参数）
3. conv1.bias 150  （共6个参数）
4. conv2.weights 156  （共2400个参数）
5. conv2.bias 2556  （共16个参数）
6. conv3.weights 2572  （共4800个参数）
7. conv3.bias 50572  （共120个参数）
8. f4.weights 50692   （共10500个参数）
9. f4.bias 61192  （共84个参数）
10. f5.weights 61276  （共1000个参数）
11. f5.bias 62276  （共10个参数）
13. # result\_bram
14. conv1.result 0
15. pool1.result 4704
16. conv2.result 5880
17. pool2.result 7480
18. conv3.result 7880
19. fc1.result 8000
20. fc2.result 8084

**需要注明的是：在bias\_weight\_bram中，各层各类参数是紧密排列的，大部分层的参数个数与我们在2.1.1节中所注明的个数相同。但是由于乘加器复用的原因（见于2.2.6节中25维乘加器设计中的注明），f4.weights的个数为**125\*84（包含全零权重），而非120\*84；f5.weights的个数为100\*10（包含全零权重），而非84\*10.

input\_bram中预先存储了20张图片，第graph张图片的基地址为**graph \* 32 \* 32，**bram深度为**20\*32\*32 = 20480**。

## 三、设计结果

### **3.1 待识别图片准备**

为了检验模型的**泛化能力**，这里的待识别图片并非来源于Mnist数据集，而是我们通过不同方式进行手写得到的图片。

图片共20张，前10张为第一组，后10张为第二组。如下所示：

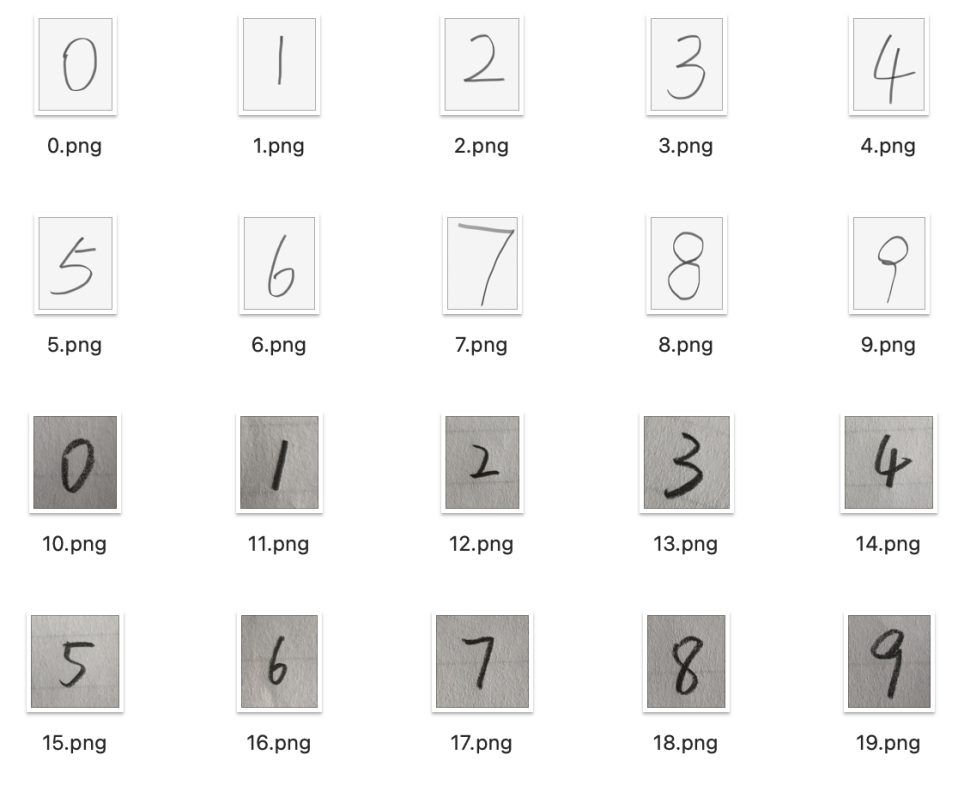


图 17 待识别图片

可以看到，第一组图片的噪声较小，是通过电子笔书写；第二组图片由于**存在真实的光照环境和书写材质问题**，因此**噪声较大**。**由此可以检验模型的泛化能力与抗噪声性能。**

在将图片向RAM中存储之前，需要将图片规格化为32\*32大小的灰度图。这一过程可以通过调用python中的opencv库完成，大致如下：

1. **def** imagePrepare(pic\_path):
2. img = cv2.imread(pic\_path, 1)
3. ret,th1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)
4. # 把opencv图像转化为PIL图像
5. im = Image.fromarray(cv2.cvtColor(th1,cv2.COLOR\_BGR2RGB))
6. # 灰度化
7. im = im.convert('L')
8. # 为图片重新指定尺寸
9. im = im.resize((28,28), Image.ANTIALIAS)
10. # 图像转换为list
11. im\_list = list(im.getdata())
12. # 图像灰度反转
13. result = [(255-x)\*1.0/255.0 **for** x **in** im\_list]

经过图片的转化，其灰度图形式如下（以两组中的8为例）：

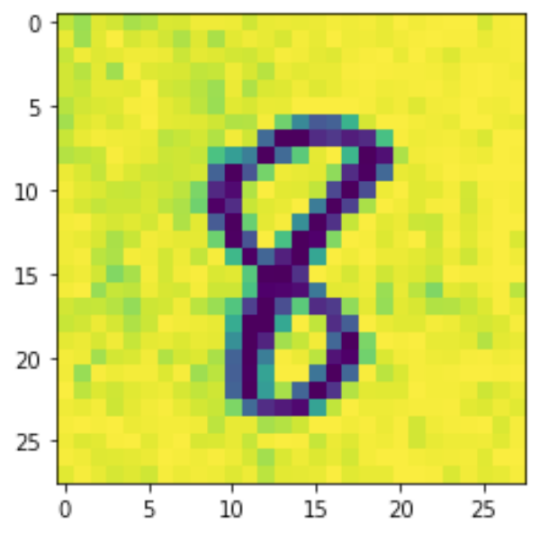
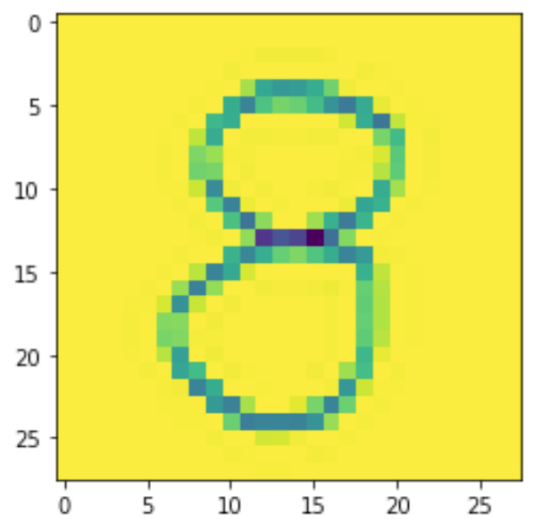


图 18 待识别图片（预处理后）

将以上20张图片利用与2.2.3相同的方式进行定点化，并且存储到input\_bram中，进行板级仿真。

### 3.2 FPGA仿真结果

在仿真中，我们的testbench将会通过修改graph的值依次对这20张图片进行前向传播。结果如下图：

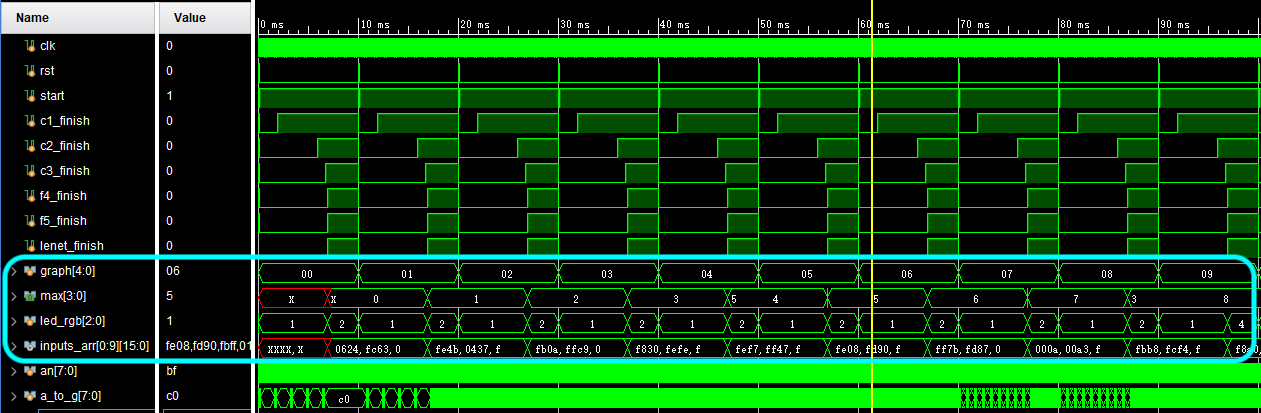


图 19 第一组图片仿真结果

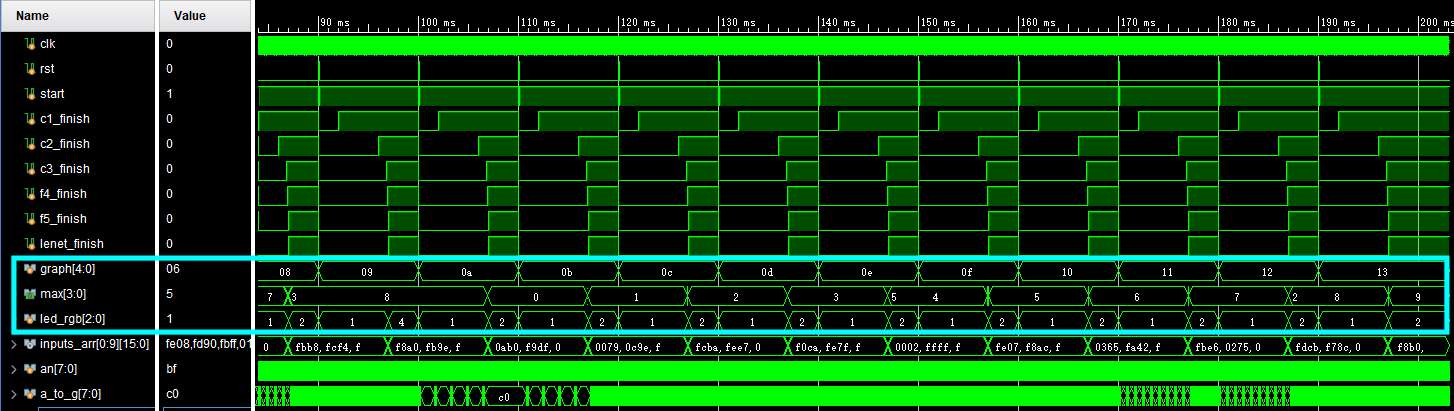


图 20 第二组图片仿真结果

其中，用蓝色框框出的是我们的主要仿真结果：graph为输入的图片编号，由于我们输入的顺序为3.1图中的顺序，因此graph%10的结果便是图片的label；max为我们的模型所得到的预测值；led\_rgb为三色led灯亮起的情况，当led\_rgb为1时，其取值为3’b001，为蓝色（预测尚未结束），当led\_rgb为2时，其取值为3’b010，为绿色（预测成功），当led\_rgb为4时，其取值为3’b100，为红色（预测失败）。

可以看到，**在20张我们预先存储的图片中，仅有1张（第一组的数字9）预测失败。在这批样本中，预测的准确率为95%。**

经过查看预测失败的那张图片所对应的输出向量，可以看到数字8（预测值）和数字9（准确值）之间的差距很小，而其他数字对应的值均很小，甚至为负数。因此这次预测虽然是错误的，但是预测为9的概率也不低。



图 21 图片9输出向量

进一步查看其他预测正确的图片的输出向量（以图片4为例），可以看到：除去准确值在输出向量中对应的值，其余值大多数为负数（若经过sigmoid，则会变成很小的一个概率值）。**由此可以肯定模型参数的准确性。**



图 22 其余图片输出向量

**经过与python模型的输出结果进行比对，output层的输出向量均计算正确，可以初步判断模型细节是正确的。**

### 3.3 FPGA板级验证结果

尚未进行预测时（start板机置0，所有代表某一层完成的LED灯均熄灭，三色LED灯为蓝色）：

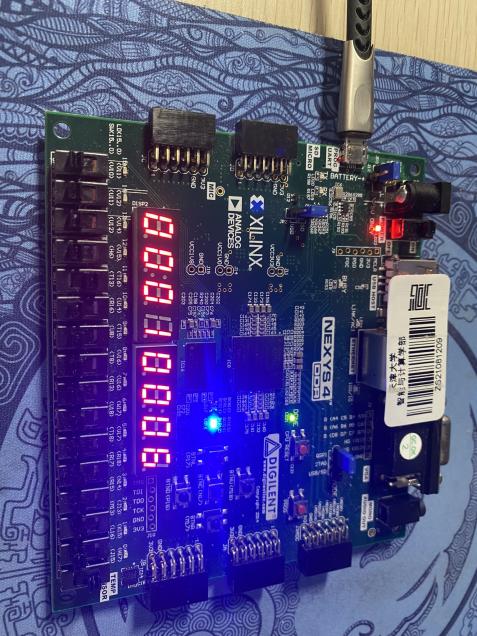


图 23 尚未进行预测时板子状态

注：此时的七段数码管中右侧的数值并非预测值，而是上一次预测的残留值，是没有意义的；左侧代表本次预测希望选择的图片。

对图片1进行预测：



图 24 图片1计算完毕时板子状态

可以看到，此时右侧七段数码管所显示的预测值为1，与左侧七段数码管显示的图片准确值相同；代表某一层计算完成的前五个LED灯均亮起，代表lenet前向传播完成的LED灯（最右侧）亮起；三色LED灯为绿色，代表预测正确。

对剩余图片进行预测，均预测正确（仅做部分展示）：



图 25 图片3计算完毕时板子状态

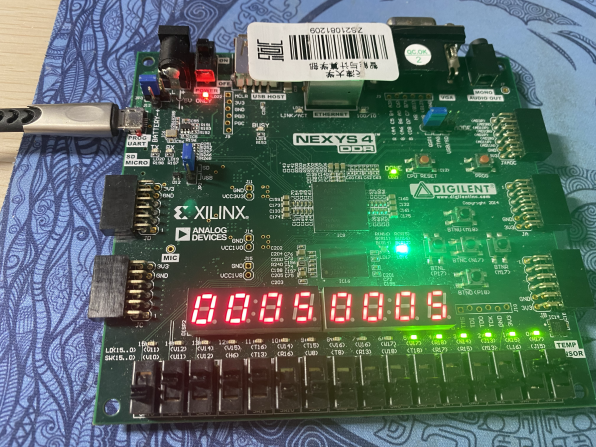


图 26 图片5计算完毕时板子状态

对图片9进行预测（20张图唯一预测错误的图片）：



图 27 图片9计算完毕时板子状态

可以看到，图片准确值为9（左侧），预测值为8（右侧），三色LED灯为红色，表示预测失败。

## 所遇到的困难和解决方法

1. **FPGA板的LUT资源过少，导致布线过程中报错。**

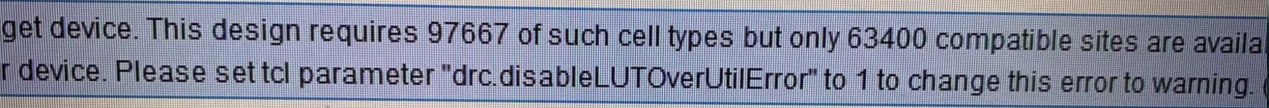


图 28 报错内容

产生原因：经过排查，我发现这一问题事实上与以下几个方面相关：

1. verilog语言中大量组合逻辑电路的编写；
2. verilog语言中大量数组的应用；
3. 模型所处理的数据位宽过大（例如达到32位）。

我们的板子（Artix-7）中仅有六万多个LUT单元，而若大量应用组合逻辑电路，会导致LUT资源的大量占用；若大量使用数组，在优化的过程中也会通过查找表的形式体现，同样会占用大量LUT资源；数据位宽过大时同上。

对于神经网络模型，如果这样去编写，确实可以做到一个时钟周期里计算完毕大量的数值，然而这样做是没必要的，原因如下：

1. 板子的时钟频率很高，哪怕是在多个时钟周期仅计算一个像素（数值），也不会消耗太多的时间资源（事实证明，在100M的时钟下，采用状态机的方式不到几十毫秒便可以完成一次前向传播）；

2. 板子的硬件资源有限，LUT资源是数十k的量级，与电脑的数十G内存无法相比。若采用高级语言一样的风格去编写，会导致仿真成功，而综合、布线失败。

解决方法：

1. 将组合逻辑电路的编写方式转换为状态机的编写风格，在多个时钟周期完成一个像素的计算（在本设计中，取参数、取输入值、存储数值等分别在单独的时钟周期里完成）。
2. 将待使用参数存储在BRAM中，按需去读写，如此不会消耗FPGA的LUT资源；
3. 控制数据位宽的大小。工程经验证明，定点数的数据位宽在8位左右便可以满足要求，本设计以16位为准进行定点化与计算，预测精度可以达到非常高。
4. **神经网络前向传播过程过于复杂，难以进行正误检验与调试。**

解决方法：在进行仿真的过程中，逐层查看输出结果。同时**利用python上编写相同的模型与代码，**进行与仿真平行的计算。将二者进行对比，逐层查看模型的正确性即可。（这一过程确实费时费力，但是亲测很有效）。

## 未来改进思路

1. **待预测图片的传入方式的改进。**

**目前方式：**在本设计中，待预测图片的传入方式是通过预先将20张图进行预处理，然后存入到一个BRAM中，再烧入到板子里。这个过程需要利用电脑，而且每一次想要预测新的图片时都要重新综合、布线、生成二进制数据流、烧入到板子中。

**改进方式：**采用串行通信的方式，将图片展平后通过串口传入到板子，在板子中进行图片像素灰度值的定点化与前向传播。具体数据流可以如下设计：

串口-->图片预处理模块-->input\_bram-->lenet5

在完成图片的加载后，再向lenet5模块中传入start信号，完成后续的传播。

这样一来，可以实时向板子中传入待预测图片，而无需预先存储、反复编译。