**实验4：卷积神经网络**

# CNN算法

CNN算法的核心分为两部分：卷积层操作和全连接层（矩阵乘法）操作。卷积层的操作是：将每一个输入通道（5\*5或7\*7图像）看作一个元素，这些元素排成一个行矩阵，所有的卷积核排成矩阵，所有的偏置矩阵排成行矩阵，则输出矩阵为列向量（包含幅图像），满足：

注意：上式中的代表矩阵分块卷积。卷积操作必须先翻转输入图像，然后再进行卷积。

全连接层可以看成是卷积核大小为的特殊卷积层，此时上式中的由分块矩阵退化为普通矩阵。直接实现矩阵乘法算法即可构建全连接层。

# HLS代码

cnn.cpp文件和cnn.h文件是实现FPGA硬件卷积神经网络加速器的核心程序文件。文件结构说明如下：

void digit\_rec(uint64\_t input\_data, int32\_t\* data\_out);

此函数实现手写数字识别卷积神经网络的卷积部分计算。其输入为7\*7的比特图像，输出为第二层卷积神经网络的输出，输出数组长度为64通道x3x3=576floats。

void cnn(uint32\_t \* data\_in, int32\_t\* data\_out);

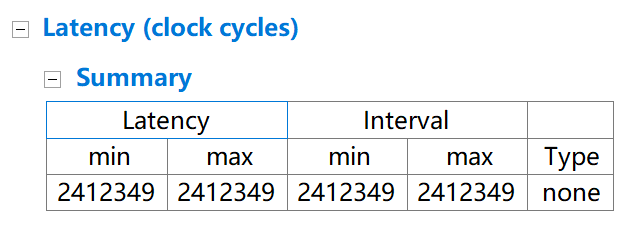
此函数是HLS综合的顶层模块，用于定义PL和PS的接口。其输入输出都是32位整型数指针。

void conv(const float \* input, float \* output, const float \* kernel, int i\_size, int o\_size, int k\_size);

此函数实现一幅图像和卷积核的卷积运算。

# HLS综合

第一次HLS综合：未优化



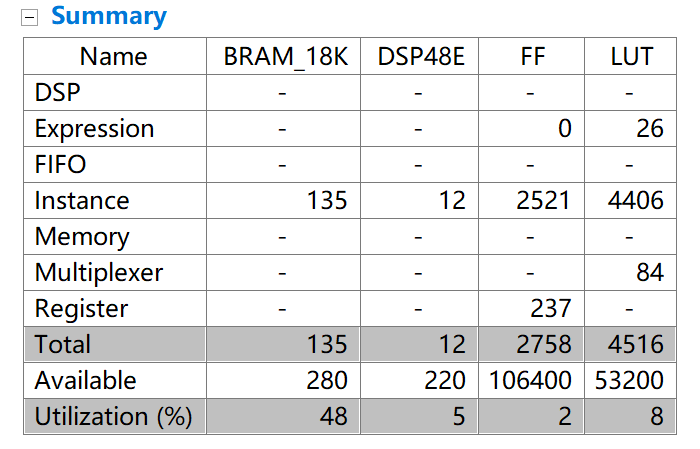


图1：第一次HLS综合资源使用情况-预计

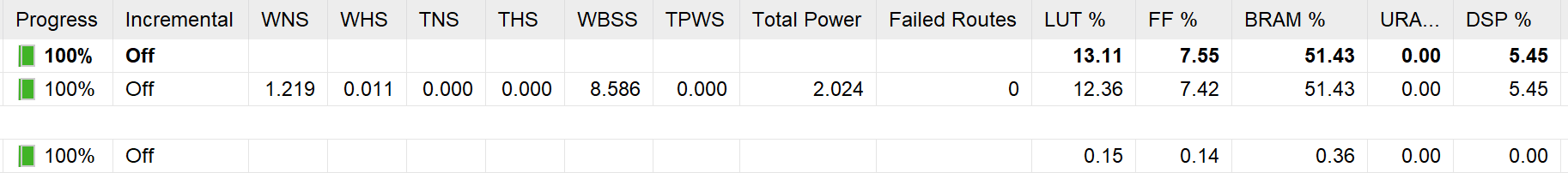


图2：第一次综合实际资源使用情况

为了提升算法执行效率，现对HLS代码进行优化处理：

**优化方法：**尽量减少代码中for循环的层数，并且尽量将for循环封装为函数模块。代码中修改的是cnn.cpp中的conv函数，修改它的原因是卷积函数被频繁调用。修改方法为减少卷积循环的次数，详见代码注释。

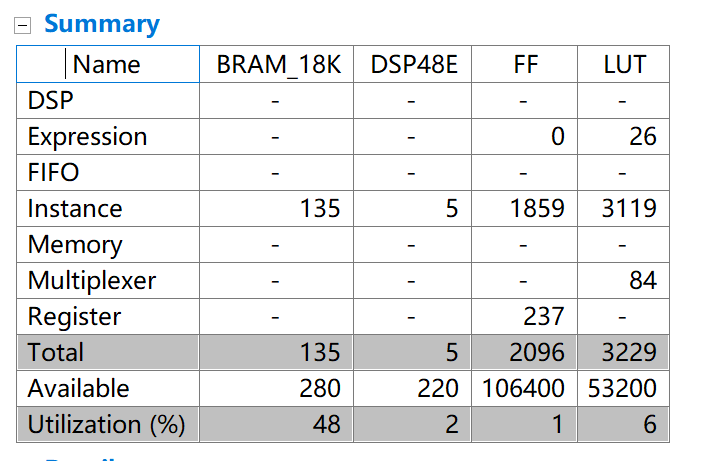


图3：第二次HLS综合（优化后）资源使用情况-预计

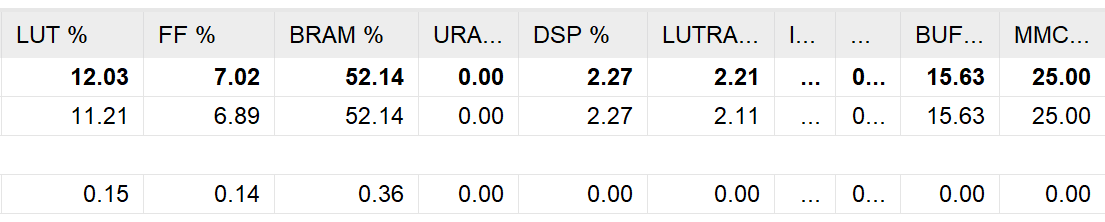


图4：第二次HLS综合（优化后）实际资源使用情况

# 性能评测

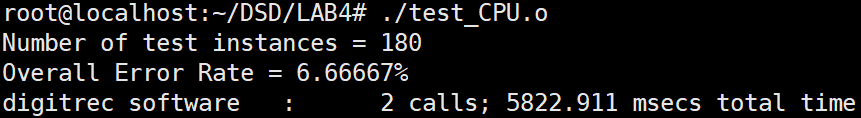


图5：Xillinux CPU执行典型结果

|  |  |  |
| --- | --- | --- |
| **运行耗时** | **第一次综合** | **硬件优化后** |
| **CPU执行** | 5823.372ms | 5822.911ms |
| **PL执行** | 4362.085ms | 4261.953ms |
|  |  |  |
| **DSP资源占用%** | 5.45% | 2.27% |
| **LUT资源占用%** | 12.36% | 11.21% |