# 利用FPGA实现CNN加速器原理

## 卷积神经网络原理

卷积神经网络通常由输入层(Input Layer)、卷积层(Convolution Layer)，池化层(Pooling Layer)和全连接层(Full-connected Layer)按照一定顺序连接而成。卷积层与池化层用于将输入的低级特征解释为高级特征，最后的全连接层用于将有限的高级特征输出为分类。

卷积神经网络的输入层是整个神经网络的输入，其不需要对输入数据进行复杂的预处理，可以直接接受多维数据输入。在处理图像的CNN中，它一般接收图像的三维像素矩阵输入。这三个维度分别代表图像的长、宽和色彩深度。对于彩色RGB图像，其深度为3；对于黑白图像，其深度为1。在每一层，卷积神经网络通过不同的网络结构将当前层的输入矩阵转化为输出矩阵。当前层的输入矩阵又称为输入特征图(Input Feature Map)，输出矩阵则称为输出特征图(Output Feature Map)。

卷积层的作用是提取图像的特征。区别于全连接层，卷积层的输出特征图中的每个节点连接到输入特征图中的一小块，而并非全部。卷积层对输入特征图中的每一小块进行卷积，得到抽象程度更高的特征。这个小块常见的尺寸为3x3或5x5，称为卷积窗口，与该小块神经元进行卷积的对象称为卷积核，也叫Kernel。卷积核在工作时，会在输入特征上扫描，在卷积窗口内对输入特征做卷积进行特征提取。

在卷积层进行特征提取后，输出特征将传递到池化层进行信息过滤。池化运算类似于降低输入图像的分辨率。通常经过卷积后，特征图的节点仍然过多，通过池化可以获得图像的概要特征，进一步减小网络的规模，减小整个网络的参数数量。

在经过多轮特征提取和信息过滤之后，可以认为图像中的信息已被抽象为了高级特征。在提取出高级特征后，通常需要使用全连接神经网络完成最后的结果输出。下面将分别介绍图像卷积与池化采样的原理。

### 图像卷积

数字图像是一个二维的离散信号，对数字图像做卷积操作其实是利用卷积核在图像上滑动，将图像点上的像素灰度值与对应的卷积核上的数值相乘，然后将所有相乘后的值相加作为卷积结果图像上像素的灰度值，并最终完成在完整图像上滑动的过程。

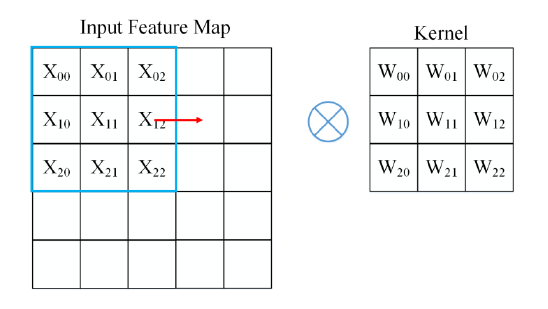


图1 二维卷积原理图

图1表示了卷积过程中一个窗口的卷积运算，该图片选用3x3的卷积核kernel，卷积窗口内共有9个数值，在9个像素值分别与对应卷积核的权值进行乘累加后得到的结果即为一个卷积窗口的卷积结果，如式1所示。

卷积窗口从图片的最左上端开始，沿着图片以从左到右，从上到下的方向以步长为1滑动，每一次滑动后的窗口都进行同样的卷积操作，我们就可以得到一张完整的卷积后的图像。



卷积的关键在于卷积核的选择，图像与不同的卷积核卷积，将得到不同的处理效果。卷积在传统数字图像处理中最常见于边缘提取和图像锐化。通过Prewitt，Sobel等卷积核可以实现边缘提取，得到图像的概要信息；通过使用微分、二值化等卷积核可以实现图像锐化，增强图像的显示效果。

对于卷积神经网络而言，网络本身采用了大量的卷积核用于提取图像特征，其区别于传统方法的点在于，这些卷积核的参数是通过学习训练确定的，而非人为指定。它们专门针对于实现训练所预期达到的目标，该方法避免了复杂的人工提取特征过程，尤其可以应用在不易进行人工提取的场合。

### 池化采样

通过图像卷积得到图像的特征之后，需要将这些特征用于分类。在这之前，往往需要通过池化层来得到概要的特征。池化最重要的目的是减小特征图的尺寸，池化层可以得到比卷积结果尺寸小很多的特征图，极大程度地降低了参数数量，并且可以在一定程度上防止过拟合。此外，池化采样还可以在一定程度上忽略目标平移、旋转、偏斜一类的相对位置变化，以此提高其平移不变特性。

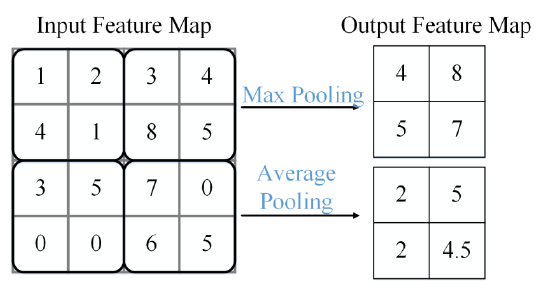


图2 平均池化和最大池化

池化运算一般通过一个2x2大小的池化窗口，在输入图像进行滑动，对每个池化窗口，计算得到1个输出特征图神经元值。池化窗口在图像上的滑动方式与卷积类似，区别是池化窗口移动的步长一般与池化窗口的边长一致，在本例中步长为2，因此池化窗口之间不会有重叠。

池化计算的方式有两种，最大池化(Max Pooling)和平均池化(Average Pooling)。最大池化即取4个神经元中的最大值为结果，平均池化即取4个神经元的平均值为结果。图2-6显示了最大池化和平均池化的计算过程，输入特征图大小为4x4，采用的池化窗口为2x2，因此输出特征图大小为2x2。

## 硬件加速平台

硬件加速是指用硬件计算部件来代替通用处理器上运行的软件程序，来获得性能的提升。可用于硬件加速的平台有通用图形处理器（GPU）、专用集成电路（ASIC）和现场可编程逻辑门阵列（FPGA）。

其中，GPU是由很多的并行计算单元所构成，在GPU内部一般都是通过采用SIMD的方式对应用进行加速，因此，GPU特别适合加速计算密集型的应用。目前，有CUDA、GLSL和OpenCL等成熟的编程框架可用于GPU平台，使得GPU平台的开发门槛相对较低，GPU平台己经成为使用广泛的加速平台。但是，GPU平台仅适用于执行数据级并行的任务，对于深度神经网络计算中不能进行数据级并行的部分任务，加速效果不太明显。

ASIC是专用集成电路，因此只对特定的应用具有很好的加速效果。但是ASIC的灵活性很低，只要应用需求发生微小的改动都需要重新来设计整个的硬件电路。并且，ASIC的开发需要具备很好的硬件知识和经验，因此门槛很高，并且ASIC的开发周期也很长，导致开发的成本也很大。

FPGA芯片内部包含有许多的可编程逻辑模块，它们都可以根据不同的应用场景要求进行重新配置，因此可以提供充分的灵活性。虽然与ASIC加速效果相比，FPGA的稍差，但是随着FPGA技术的不断发展和创新，FPGA芯片的集成度越来越高，芯片的速度也越来越快，并且，FPGA芯片与通用处理器之间的通信也变得越来越简单。更为重要的是，基于FPGA的开发工具链也越来越多，这大大的降低了FPGA的开发门槛。目前，较为成熟的FPGA开发工具链有XIlinx公司的Vivado和Altera公司的基于OpenCL开发的FPGA SDK，它们都能够帮助开发人员从事针对FPGA芯片的编程工作。

综合以上硬件加速平台的特点可知，FPGA是介于ASIC的效率和通用处理器的可编程性之间的具有吸引力的选择。

## 硬件加速技术

### 并行计算

并行计算是指，通过分析软件算法的并行性，将计算任务进行分解，使用多组运算资源并行完成多组运算的过程。例如，某一算法需要完成N次加法运算，且这N次运算之间互相不相关，如使用单加法器，则需要N个周期以完成算法。如果使用N个加法器实现并行计算，则在单周期即可完成算法，加速比为N。

### 流水线技术

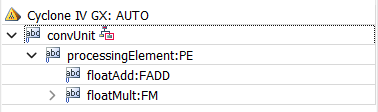
流水线技术的实现是基于将复杂的运算逻辑分为多个组合逻辑段，每一段中间插入寄存器将其切割来实现。这样可以使寄存器间的数据路径更短，更容易满足建立时间的要求，从而提高系统的工作频率。而且可以保证在每个时刻，每一段运算逻辑都在运行，避免了资源浪费。流水线并不能减少单次运算的时间，在插入寄存器延时的情况下，反而会更长。但是在整体上，对于多组数据运算，其运行的总周期将显著减少，并且因为时钟频率得以提升，其吞吐率也将显著增加。

### Winograd算法

Winograd算法是一种快速卷积算法，用于减少卷积运算中的乘法数量，代价是适当增加加法数量。Winograd算法流程与FFT类似,但是运算过程均在实数域完成。相比于FFT , Winograd的转换代价更高,这是因为随着Winograd算法参数增大,常数矩阵中的值会骤增。

# 系统设计

自顶而下分析共包括四个模块：convUnit, processingElement, floatAdd和floatMult。

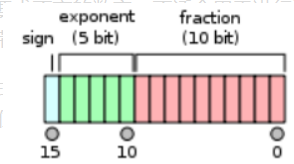


## floatAdd16

执行两个半精度浮点数的加法

### 半精度浮点数

IEEE754-2008包含一种“半精度”格式，只有16位宽。故它又被称之为binary16，这种类型的浮点数只适合用于存储那些对精度要求不高的数字，不适合用于进行计算。与单精度浮点数相比，它的优点是只需要一半的存储空间和带宽。半精度的格式与单精度的格式类似，最左边的一位仍是符号位，指数有5位宽且以余-16（excess-16）的形式存储，尾数有10位宽，但具有隐含1。



如图所示，sign为符号位，0表示这个浮点数为正，1表示这个浮点数为负

先介绍尾数，再说指数，fraction为尾数，有10位长，但是有隐含1，尾数可以理解为是一个浮点数小数点后的数，如1.11，尾数就为1100000000（1），最后的隐含1主要用于计算时，隐含1可能存在可以进位的情况。

exponent为指数位，有5位长，具体表示的值有以下几种情况：

当指数位全为0，尾数位也全为0的时，表示的就是0；

当指数位全为0，尾数位不全为0时，表示为subnormal value，非规格化浮点数，是一个非常小的数；

当指数位全为1，尾数位全为0时，表示的是无穷大，此时如果符号位为0，表示正无穷，符号位为1，表示负无穷；

当指数位全为1，尾数位不全为0时，表示的不是一个数；

其余情况下，指数位的值减去15就是其表示的指数，如11110表示的就是30-15=15；

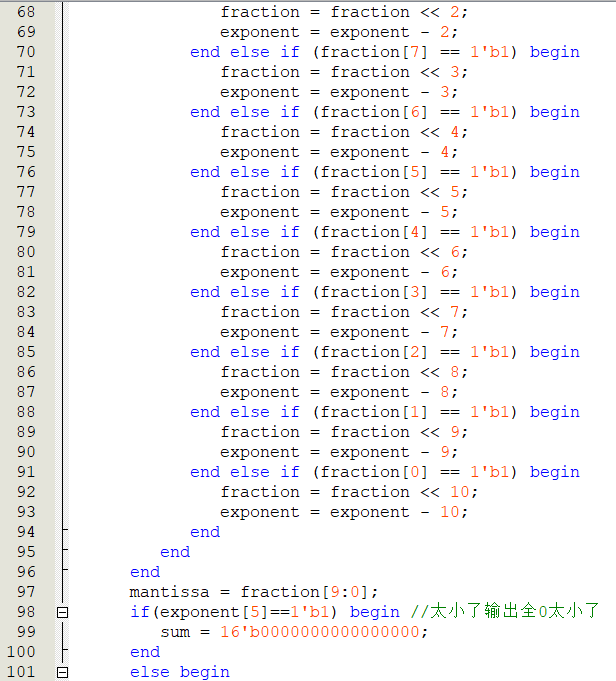
所以我们可以得到，半精度浮点数的值得计算方式为（-1）^sign×2^（指数位的值）×（1+0.尾数位）。

### floatAdd16 Verilog编码原理

半精度浮点数的加法器不同于乘法器，需要配平待相加的两个数据的阶数后再根据正负情况进行加减运算操作，在下文的代码注释中有较为详细的介绍。

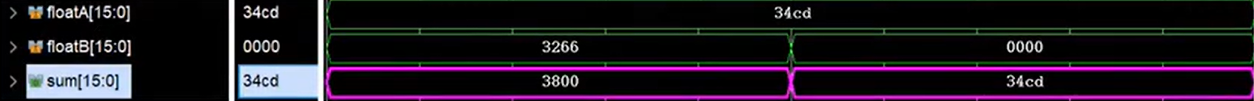






### floatAdd16仿真结果

取输入分别为0.3和0.2，则输出为0.5，0.3和0.2对应的FP16的十六进制形式为34cd和3266，输出为0.5的FP16的十六进制形式3800；再取输入分别为0.3和0，则输出为0.3，0.3和0对应的FP16的十六进制形式为34cd和0000，输出为0.3的FP16的十六进制形式34cd。



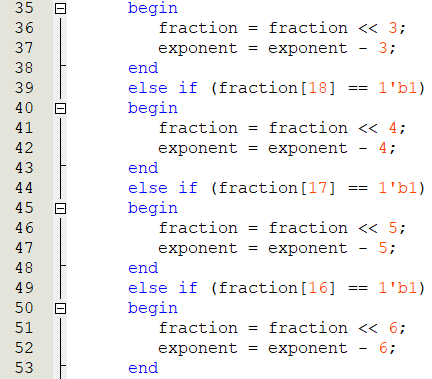
## floatMult16

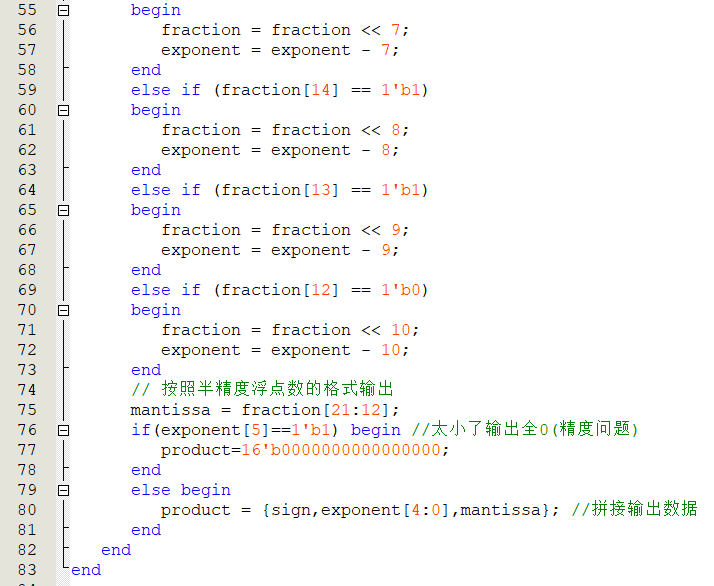
执行两个半精度浮点数的乘法

### floatMult16 Verilog编码原理

半精度浮点数乘法器的实现主要包括对符号位的处理，指数借位，乘法计算，超范围小数等内容。在下文的代码注释中有较为详细的介绍。







### floatMult16仿真结果

取输入分别为4和5，则输出为20，4和5对应的FP16的十六进制形式为4400和4500，输出为20的FP16的十六进制形式4d00；再取输入分别为0.0004125和0，则输出为0，0.0004125和0对应的FP16的十六进制形式为0ec2和0000，输出为0的FP16的十六进制形式0000。

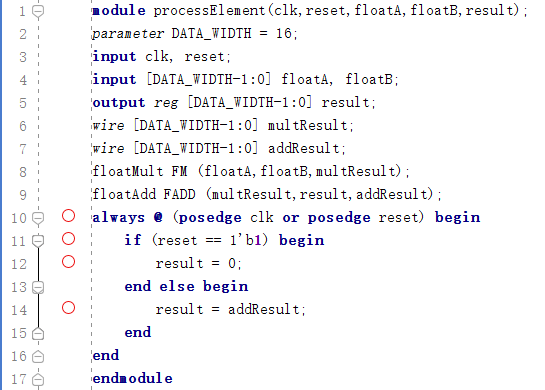


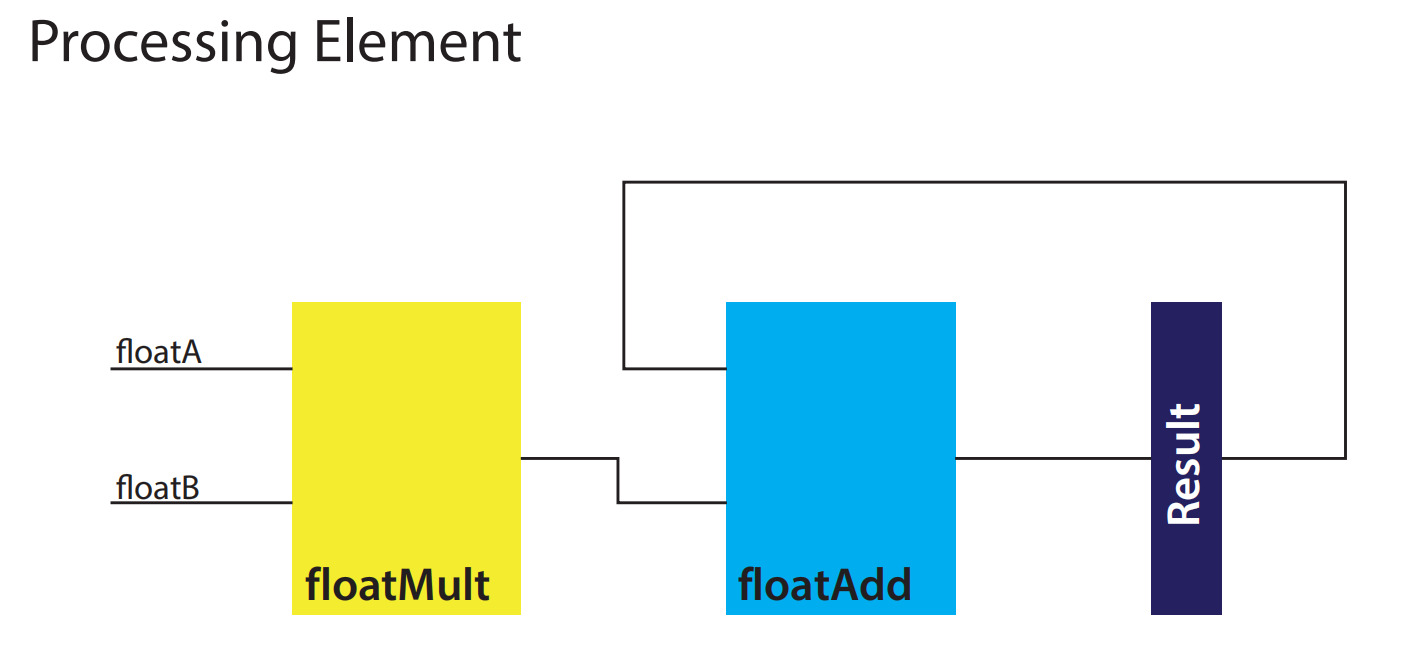
## PE（Processing element）

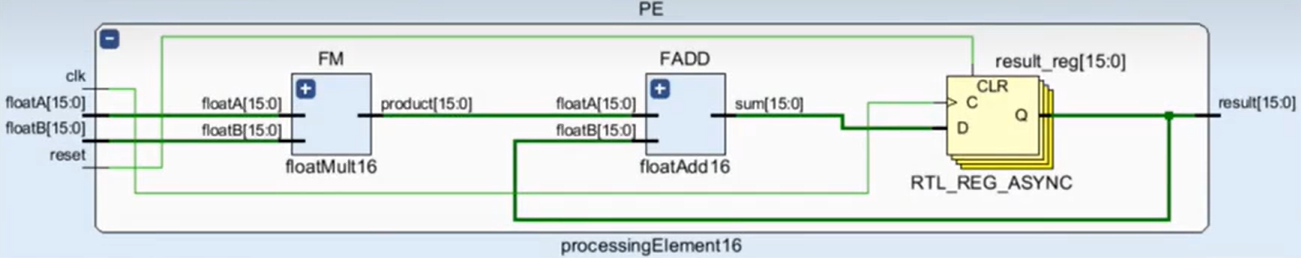
### PE Verilog编码原理

执行具体的卷积操作，即权重与数据的相乘以及相乘结果的累加。

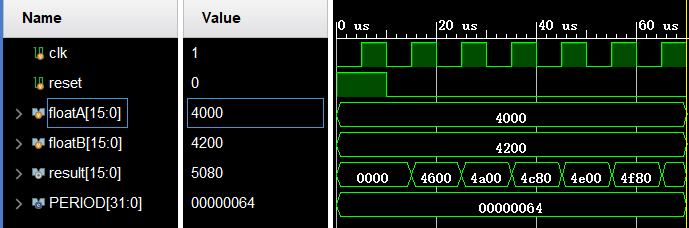
加Result\_reg将电路从组合逻辑转为同步时序电路，保证数据的同步。







### PE仿真结果



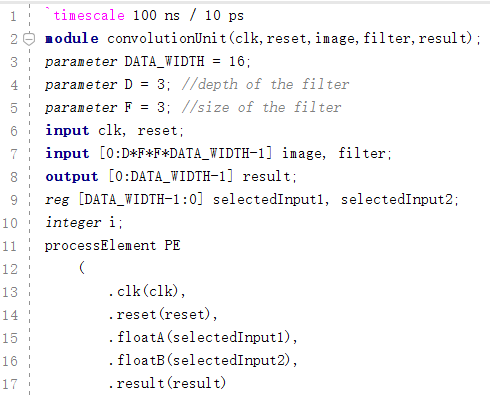
取输入分别为2和3，经过一个时钟周期，输出为两者相乘得6，2和3对应的FP16的十六进制形式为4000和4200，输出为6的FP16的十六进制形式4600；再经过一个时钟周期，输出为相加结果12，12对应的FP16的十六进制形式为4a00；再经过一个时钟周期，输出再加上6，结果为18，18对应的FP16的十六进制形式为4c80，以此类推，每经过一个时钟周期，就对相乘得到的积进行一次累加。

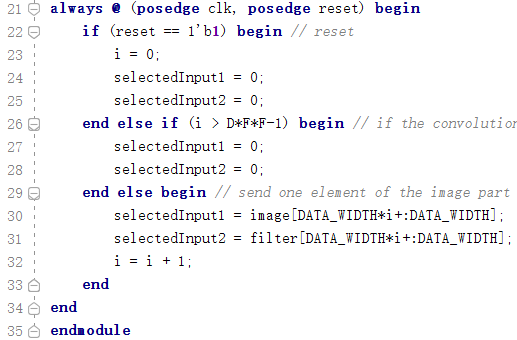
## CU (convolution unit)

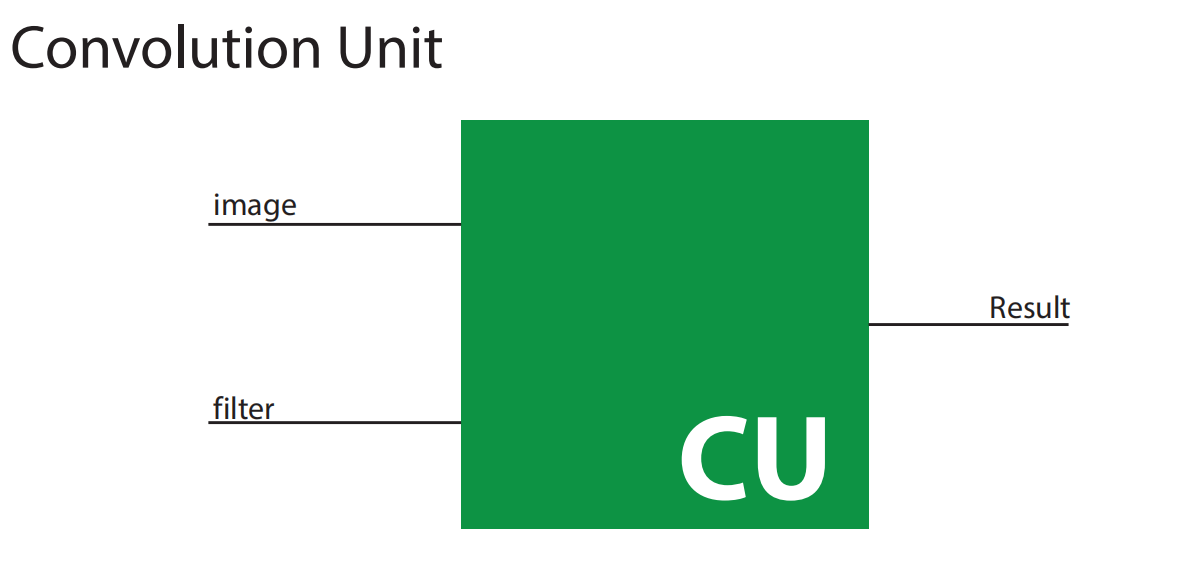
### CU Verilog编码原理

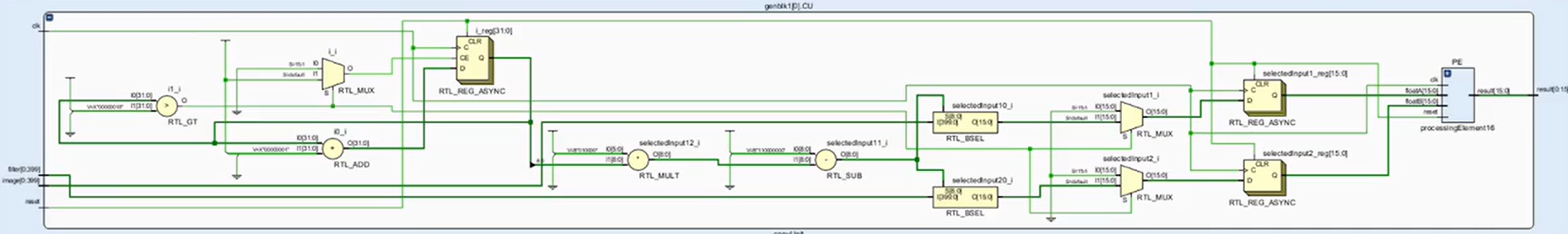
设定卷积核的大小为3x3。每个卷积核与输入图像进行卷积，输出特征图向量。

窗口表示卷积核filter和输入image重合部分，一个窗口卷积出一个计算结果，convUnit的作用是循环使用PE完成一个窗口的卷积运算，并输出最终计算结果。循环使用PE可以减少硬件资源占用，以牺牲速度来换取面积。









### CU仿真结果

Image和filter矩阵均如下，其中filter为三通道：



4.000000的FP16的十六进制形式为4400，卷积结果为432.000000，其FP16的十六进制形式为5ec0，与仿真结果一致。