|  |  |
| --- | --- |
| **Max Pooling IP 文档** | 文件类型：设计文档 |
| 文件编号：XXXX |
| 面向的部门： AI |
| 保密等级：中 |
| 作者：李冰 |
| 日期：2019-10-25 |
| 版本：1.0V |



版本记录：

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 日期 | 修订者 | 修订内容 |
| 1.0 | 2019-10-25 | 李冰 | 创建此文件 |
|  |  |  |  |
|  |  |  |  |

# 概要

本文是Max Pooling模块的RTL IP文档，包括模块功能说明、接口说明、设计说明等章节。

Pooling是卷积神经网络中的一种数据处理方法，除此之外还有卷积、残差、激活、Full Connect（全连接）等不同的数据处理方法。常见的Pooling分为Max Pooling和Average Pooling，本IP只涉及Max Pooling。

## Why Pooling?

卷积层的一个特点是会精确地记录一个feature在feature map中的位置，这将导致当输入有微小的偏移时，对卷积层输出的feature map影响较大。可以通过对feature map做下采样可以降低其特征精度、丢弃一定的细节，保留相对来说更为重要的结构性信息，同时能够减小计算量。下采样可以通过改变卷积的stride来实现，但更有效的一个方法是Pooling层。Pooling层通常紧跟卷积层后面来操作，下面的章节将对Max Pooling算法以及本模块的实现作介绍。

# 功能说明

本章节描述Max Pooling的算法，并对Max Pooling RTL IP的功能作简单说明。为阐述方便，将算法中涉及的变量定义为表格 1。注意“Input”，“Output”均相对于Pooling层。

表格 1 Pooling算法变量

|  |  |  |
| --- | --- | --- |
| 序号 | 变量名称 | 含义 |
|  | W | Input feature map Width |
|  | H | Input feature map Height |
|  | C | Input feature map Channel(depth) number |
|  | KX | Max pooling kernel size along W dimension |
|  | KY | Max pooling kernel size along H dimension |
|  | SX | Max pooling stride size along W dimension |
|  | SY | Max pooling stride size along H dimension |
|  | OW | Output feature map Width |
|  | OH | Output feature map Height |

## Max Pooling算法

如图 1所示为一个简单的（4\*4）的二维数字矩阵，可以认为它是一个（W, H, C）=（4, 4, 1）的微小feature map。下面将通过4个例子来阐述Max Pooling的运算过程，每个例子采用不同的Pooling 参数。



图 1 一个二维数字矩阵

1. (KX, KY, SX, SY) = (2, 2, 2, 2)，上、下、左、右PAD均为0。

如图 2 所示，对二维矩阵进行（KX, KY, SX, SY）=（2, 2, 2, 2）的Max Pooling操作，上、下、左、右PAD均为0，输出为一个（2\*2）的矩阵，其算法非常简单，首先将输入矩阵分成4个子块（如图中每一块用不同的颜色表示），称为一个“kernel”，Max Pooling运算即是取每一个kernel中的最大值输出， KX=2，KY=2，即kernel size = （2，2）；SX=2，SY=2，即stride 为（2，2），stride表示W方向（或者H方向）两个相邻kernel之间的距离。本例中，stride的值等于kernel的值，输出的feature map的大小为输入的1/4。此外，stride的值还可能小于kernel的值，请看下一个例子。

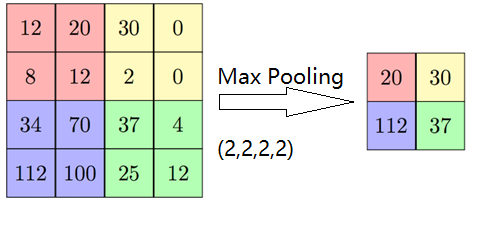


图 2 （KX, KY, SX, SY）=（2, 2, 2, 2）Max Pooling示意图

1. (KX, KY, SX, SY) = (2, 2, 1, 1)，上、下、左、右PAD均为0。

如图 3所示，对相同的二维矩阵作（KX, KY, SX, SY）=（2, 2, 1, 1）的Max Pooling（PAD均为0）处理，每个kernel的size仍为2\*2，但kernel之间的距离为1，相邻kernel之间有1的overlap，则最终输出为一个3\*3的二维矩阵。

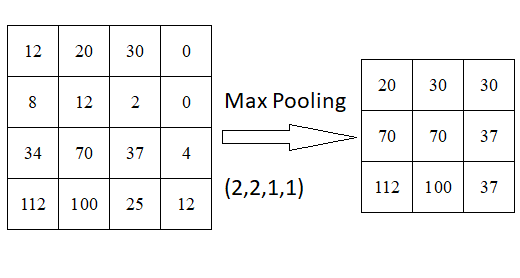


图 3 （KX, KY, SX, SY）=（2, 2, 1, 1）Max Pooling示意图

1. (KX, KY, SX, SY) = (2, 2, 1, 1)，上、左PAD为0，右、下PAD均为1。

如图 4所示，灰色部分表示右、下PAD。PAD可能会影响输出feature map的size和数值，具体地，每一行最后一个kernel和每一列最后一个kernel在做Max Pooling运算时包含PAD数据（可以认为是int8最小值-256），此时输出feature map的size是(4\*4)，与输入的size相同。

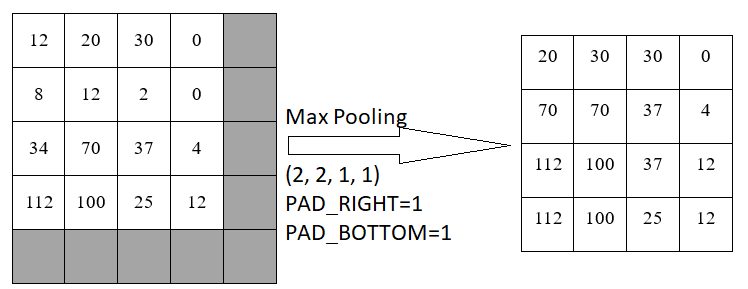


图 4 （KX, KY, SX, SY）=（2, 2, 1, 1）Max Pooling示意图2

1. (KX, KY, SX, SY) = (2, 2, 1, 1)，上、左PAD为1，右、下PAD均为0。

本例与上一个例子唯一不同的地方是上、左PAD为1，而右、下PAD为0，Max pooling 输出的结果不同，size相同。

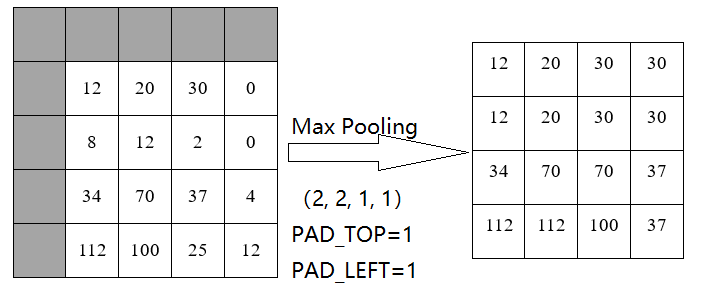


图 5 （KX, KY, SX, SY）=（2, 2, 1, 1）Max Pooling示意图3

以上例子以一个二维数字矩阵作为输入简单描述了Max Pooling的算法，而在一个实际的神经网络中，Max Pooling层的输入通常是一个三维的feature map。下一小节将说明如何对三维 feature map进行Max Pooling处理。

## 三维feature map的Max Pooling

卷积网络中的feature map可以建模为一个3维数字矩阵，W、H、C分别表示输入feature map的Width，Height，Depth。OW、OH分别表述输出feature map的Width，Height。更多变量的定义见表格 1。

图 6为对一幅3-D feature map 作Max Pooling的示意图。如图所示，输入feature map 大小为W\*H\*C = 224\*224\*64，经过（KX，KY，SX，SY）=（2，2，2，2）的Max Pooling之后，输出feature map大小为112\*112\*64。注意Max Pooling只改变W、H维度的大小，不改变C维度（Depth）的大小。

如果固定一个channel，将会得到一片二维的depth slice，图 6中的输入feature map共有64片depth slice，每一片独立进行Max Pooling运算，输出size为112\*112\*1，因此对于整个输入feature map，Max Pooling之后会得到112\*112\*64的输出。注意，与卷积运算不同的是，Max Pooling运算中各通道之间是相互独立的，通道之间的数据互不作用。经过max pooling之后，输出feature map的大小为输入的1/（SX\*SY）。

图片包含 时钟, 物体

描述已自动生成

图 6 Max Pooling on a 224\*224\*64 feature map

## IP功能说明

本IP对卷积输出的feature map 做Pooling运算，即，其输入为卷积层输出的一幅完整的feature map，输出为按照Max Pooling算法处理之后的feature map。IP采用并行设计的方法，性能和资源均scalable，可配置为1x，2x，4x，8x，配置为8x性能时，在clock为1GH下，吞吐率达到128GiB/s。

IP的处理latency较小（当stride大于1时，lantency只有6个clock cycle）。

IP主要消耗的资源在RAM BLOCK，实际使用中根据资源配置的需求，RAM BLOCK深度通常较浅（例如32），在FPGA上实现时，综合工具会用distributed RAM，在芯片上实现时，一般会采用register组。

本IP支持KX，KY，SX，SY，上、下、左、右PAD的动态配置。

# IP接口

## IP模块组成

IP 包含以下设计文件：

1. pooling\_block.sv
2. pooling\_buffer.sv
3. pooling\_buffer\_ctrl.sv
4. pooling\_line.sv
5. pooling\_params.sv
6. pooling\_sfifo.v
7. pooling\_split.sv
8. pooling\_top.sv

## IP顶层参数

顶层module为pooling\_top.sv。表格 2中1~8为模块顶层参数，可以通过改变它们的值将IP配置为不同的性能，具体地：

1. PIX\_DW。PIX\_DW=8时，支持int8网络；=16时，支持int16网络；
2. IN\_DW。输入数据的位宽，例如，int8网络，同时处理16个通道，IN\_DW=16\*8=128；

int16网络，同时处理8个通道，IN\_DW=8\*16=128；

1. XMAX。IP所能处理的最大“X”值（what is “X”? refer to 表格 3）；
2. HMAX。IP所能处理的最大“BLOCK\_Y”值；
3. KMAX。IP所能处理的“KX”或“KY”或“SX”或“SY”的最大值；
4. BMAX。IP所能处理的“Z\_BLOCKS”，“Y\_BLOCKS”的最大值；
5. FIFO\_SHOW\_AHEAD。控制IP中使用FIFO是否show ahead，为0时，会优化FIFO部分的时序，建议在FPGA上使用时设置为1，在芯片上实现时设置为0；
6. PORT\_N。IP并行处理的分块数（along W dimension）。可以通过配置此参数来控制IP的数据吞吐率和资源消耗，当PORT\_N增加时，IP的数据的吞吐率和资源消耗将按比例增加。

9~15为衍生参数，例化时不允许修改。

表格 2 Top parameters

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **参数名称** | **默认值** | **描述** |
|  | PIX\_DW | 8 | int8~8; int16~16 |
|  | IN\_DW | 128 | Input data bit width |
|  | XMAX | 63 | Max “X” value |
|  | HMAX | 1024 | Max “BLOCI\_Y’ value |
|  | KMAX | 15 | Max “KX” or “KY” value |
|  | BMAX | 64 | Max “Z\_BLOCKS” or “Y\_BLOCKS” value |
|  | FIFO\_SHOW\_AHEAD | 1 | 1~FIFO show ahead; 0~FIFO NOT show ahead. |
|  | PORT\_N | 8 | Parallel ports number |
|  | KW | 4 | Derived. KW===$clog2(KMAX+1) |
|  | PAGE\_N | 16 | Derived. PAGE\_N===IN\_DW/PIX\_DW |
|  | YMAX | 1024 | Derived. YMAX===HMAX |
|  | XBITS | 6 | Derived. XBITS===$clog2(XMAX+1) |
|  | YBITS | 11 | Derived. YBITS===$clog2(YMAX+1) |
|  | BBITS | 7 | Derived. BBITS=$clog2(BMAX+1) |
|  | WBITS | 9 | Derived. WBITS=XBITS+$clog2(PORT\_N) |

## IP顶层端口

表格 3为IP的顶层端口列表。对以下端口作进一步的解释：

1. reset\_n。全局复位信号，低电平有效，应为异步复位同步释放。reset\_n信号assert时，所有寄存器将复位，IP状态机恢复到IDLE状态；
2. clk。Max Pooling算法时钟；为了使本IP正常工作，外部应该输入有效的时钟信号；
3. bp。bypass enable，高电平有效。当bp为高电平时，本IP绕过Max Pooling算法，直接将输入assign给输出。在输入一个完整的feature map期间（bypass or not），bp信号必须保持一致；
4. KX。Max Pooling kernel size along W dimension；
5. KY。Max Pooling kernel size along H dimension；
6. SX。Max Pooling strides size along W dimension；
7. SY。Max Pooling strides size along H dimension；
8. X。W维度分块后每一块的宽度，例如，PORT\_N=8时，输入为一幅W=416的feature map，则W维度分成PORT\_N块时，每一块的宽度X=416/8=52；
9. W。输入feature map 真实宽度（W维度大小）；
10. PAD\_TOP。Max Pooling算法真实上PAD，由算法决定；
11. PAD\_LEFT。Max Pooling算法真实左PAD，由算法决定；
12. PAD\_RIGHT。Max Pooling算法真实右PAD，由算法决定；
13. PAD\_BOTTOM。Max Pooling算法真实下PAD，由算法决定（关于PAD算法参见4小节）；
14. Z\_BLOCKS。当待处理的feature map在depth方向（channel）过大时，需要将其分成多块分别处理，每一块的输出结果按照正确的地址存储到memory中，从而得到整块pooled feature map。例如输入为一幅W\*H\*C=28\*28\*96的feature map，则沿depth方向需要将其分成Z\_BLOCKS=C/PAGE\_N=96/16=6块分别处理（PAGE\_N为IP并行处理的通道数，其值的定义见表格 2；
15. Y\_BLOCKS。类似地，当待处理的feature map在Height方向过大时，也需要分块；
16. BLOCK\_Y。当Height方向作分块时，BLOCK\_Y为当前处理的分块的高度值。关于Height方向分块算法，参见4.2小节；
17. din0~din7。输入数据，PORT\_N（默认8）组并行。din\_<n>为W方向第n个分块的输入数据。每一个分块的din\_<n>遍历的顺序是先Width方向，后Height方向，n=0,1,…,7，且它们必须是独立的、同步的（因此只有一个din\_v）；
18. din\_v。din0~din7的valid信号。din\_v为高电平时，表示din0~din7有效；
19. in\_ready。握手信号，高电平表示at the current clock cycle，din\_v and din\_<n> are accepted，低电平表示din\_v and din\_<n> are NOT accepted，and they shall hold their values until the next clock cycle after in\_ready is asserted；
20. dout\_v。dout0~dout7的valid信号。dout\_v为高电平时，表示dout0~dout7有效；
21. dout0~dout7。PORT\_N（默认8）组并行。dout\_<n>为W方向第n个分块的输出数据。每一个分块的dout\_<n>遍历的顺序是先Width方向，后Height方向，n=0,1,…,7，且它们是独立的、同步的。

表格 3 Top Ports

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **序号** | **Port name** | **I/O** | **bit width（default）** | **描述** |
|  | reset\_n | input | 1 | global reset\_n, low-active |
|  | clk | input | 1 | pooling clock |
|  | bp | input | 1 | bypass enable, high-active |
|  | KX | input | KW（4） | kernel size, W-dimension |
|  | SX | input | KW（4） | stride size, W-dimension |
|  | KY | input | KW（4） | kernel size, H-dimension |
|  | SY | input | KW（4） | stride size, H-dimension |
|  | X | input | XBITS（6） | W维度分块后每一块的宽度 |
|  | W | input | WBITS（9） | feature map 真实宽度 |
|  | PAD\_TOP | input | KW（4） | 上PAD |
|  | PAD\_LEFT | input | KW（4） | 左PAD |
|  | PAD\_RIGHT | input | KW（4） | 右PAD |
|  | PAD\_BOTTOM | input | KW（4） | 下PAD |
|  | Z\_BLOCKS | input | BBITS（7） | Depth 维度分块个数 |
|  | Y\_BLOCKS | input | BBITS（7） | Height维度分块个数 |
|  | BLOCK\_Y | input | YBITS（11） | 当前输入feature map 分块的的Height |
|  | din0 | input | IN\_DW(128) | W维度第1个分块输入data |
|  | din1 | input | IN\_DW(128) | W维度第2个分块输入data |
|  | din2 | input | IN\_DW(128) | W维度第3个分块输入data |
|  | din3 | input | IN\_DW(128) | W维度第4个分块输入data |
|  | din4 | input | IN\_DW(128) | W维度第5个分块输入data |
|  | din5 | input | IN\_DW(128) | W维度第6个分块输入data |
|  | din6 | input | IN\_DW(128) | W维度第7个分块输入data |
|  | din7 | input | IN\_DW(128) | W维度第8个分块输入data |
|  | din\_v | input | IN\_DW(128) | 输入data valid |
|  | in\_ready | output | IN\_DW(128) | 握手信号，input ready |
|  | dout\_v | output | IN\_DW(128) | 输出data valid |
|  | dout0 | output | IN\_DW(128) | W维度第1个分块输出data |
|  | dout1 | output | IN\_DW(128) | W维度第2个分块输出data |
|  | dout2 | output | IN\_DW(128) | W维度第3个分块输出data |
|  | dout3 | output | IN\_DW(128) | W维度第4个分块输出data |
|  | dout4 | output | IN\_DW(128) | W维度第5个分块输出data |
|  | dout5 | output | IN\_DW(128) | W维度第6个分块输出data |
|  | dout6 | output | IN\_DW(128) | W维度第7个分块输出data |
|  | dout7 | output | IN\_DW(128) | W维度第8个分块输出data |

# PAD与分块算法

## Max Pooling PAD算法

本IP提供4个PAD配置端口：PAD\_TOP， PAD\_LEFT，PAD\_RIGHT，PAD\_BOTTOM。下面主要讨论这四个参数在算法层面的计算，为讨论方便，额外定义两个变量PAD\_HEIGHT，PAD\_WIDTH，它们分别表示Height方向和Width方向的PAD总和，满足如下恒等式：

 公式 4‑1

  公式 4‑2

在算法层面，不同的框架使用的PAD算法可能不一样，使用不同的PAD算法输出的pooled feature map结果可能也不相同。但无论采用何种PAD算法，一旦确定之后，输出feature map的大小OW，OH（参见表格 1）也随之确定。假定KY>=SY且KX>=SX（其余情况很少出现在pooling层，暂不做讨论），那么PAD\_HEIGHT可以按照如下公式计算：

 公式 4‑3

 公式 4‑4

计算出PAD\_HEIGHT和PAD\_WIDTH之后，如果已知PAD\_TOP（或PAD\_BOTTOM），那么可以根据公式 4‑5计算出PAD\_BOTTOM（或者PAD\_TOP），同样，如果已知PAD\_LEFT（或者PAD\_RIGHT），那么可以根据公式 4‑6计算出PAD\_RIGHT（或者PAD\_LEFT）。

下面通过一个例子来说明。假设输入feature map大小为W\*H\*C=(56, 56, 64)，(KX, KY, SX, SY) = (3, 3, 2, 2)，pooling算法输出大小为OW\*OH\*C=(28, 28, 64)，那么可以计算出PAD\_HEIGHT为：



可以计算出PAD\_WIDTH为：



那么可能的2种常见的PAD配置是：

1. PAD\_TOP=1, PAD\_BOTTOM=0, PAD\_LEFT=1, PAD\_RIGHT=0
2. PAD\_TOP=0, PAD\_BOTTOM=1, PAD\_LEFT=0, PAD\_RIGHT=1

注意，本小节（4.1）的内容不是MANDATORY，仅属于INFORMAL范畴。软件应该根据算法提供的数据推算出这4个数值，并正确配置于本IP。

## Height分块算法

1. reshape模块reshape.v
2. 参数介绍

Table 参数介绍表

|  |  |  |
| --- | --- | --- |
| **参数名称** | **默认值** | **描述** |
| **全局参数parameter** | | |
| DATA\_WIDTH | 128 | AXI bus data width |
| ADDR\_WIDTH | 16 | Address width |
| **局部参数localparam** | | |
| CFG\_4BIT\_WIDTH | 4 | 寄存器配置数据宽度4bits |
| CFG\_8BIT\_WIDTH | 8 | 寄存器配置数据宽度8bits |
| CFG\_10BIT\_WIDTH | 10 | 寄存器配置数据宽度10bits |
| CFG\_16BIT\_WIDTH | 16 | 寄存器配置数据宽度16bits |
| BRAM\_RD\_LATENCY | 2 | weight解压缩cache深度 |
| BRAM\_WIDTH | 128 | bram data width |

1. 端口信号描述

Table 端口信号列表

|  |  |  |  |
| --- | --- | --- | --- |
| **端口名称** | **Type** | **Width** | **描述** |
| clk | I | 1 | mem\_clk，500MHz |
| rst\_n | I | 1 | 低电平复位信号 |
| bypass | I | 1 | bypass信号，1：该模块pass掉，0：该模块使能 |
| kernel\_width | I | CFG\_4BIT\_WIDTH | kernel size width |
| kernel\_high | I | CFG\_4BIT\_WIDTH | kernel size high |
| line\_length | I | CFG\_8BIT\_WIDTH | 原图片一个面的num，公式=image\_width\*image\_depth\*INT\_DATA\_WEIGTH/128 |
| image\_high | I | CFG\_16BIT\_WIDTH | 原图片的高度 |
| kernel\_bits | I | CFG\_10BIT\_WIDTH | kernel滑动的bits, 等于kernel\_size\*image\_depth\*INT\_DATA\_WEIGTH |
| stride\_bits | I | CFG\_10BIT\_WIDTH | stride滑动的bits,等于stride\*image\_depth\*INT\_DATA\_WEIGTH |
| left\_pad\_bits | I | CFG\_10BIT\_WIDTH | left\_pad\*image\_depth\*INT\_DATA\_WEIGTH |
| right\_pad\_bits | I | CFG\_10BIT\_WIDTH | right\_pad\*image\_depth\*INT\_DATA\_WEIGTH |
| buf\_ready | I | 1 | 当fifo中数据个数大于缓存的cache个数时，缓存了一个面的数据，line\_length |
| buf\_dout | I | DATA\_WIDTH | fifo读出数据 |
| buf\_vld | I | 1 | fifo读出有效信号 |
| rd\_vld | O | 1 | fifo读使能信号 |
| rd\_addr | O | 9 | fifo读地址信号 |
| block\_done | O | 1 | reshape block done signal |
| o\_data | O | DATA\_WIDTH | reshape out data |
| o\_valid | O | 1 | reshape out data valid |

1. 内部信号描述

Table 内部信号列表

|  |  |  |  |
| --- | --- | --- | --- |
| **端口名称** | **Type** | **Width** | **描述** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 设计详细说明

本设计的关键点在于状态机实现移位操作，如下图所示。状态机共有8个状态：IDLE、RD\_BRAM、SHIFT\_RD、REG\_BUF、PAD\_R、PAD\_R\_WAIT、LINE\_LAST\_1以及LINE\_LAST\_2.



Figure 状态机流程图

# 仿真测试说明

由算法给出了kernel\_size=3、5、7、9，pad =1、2、3,stride =1、2、3时候的reshape数据，经过FPGA仿真，数据一致。Resnet50以及Mobilenet layer0的reshape都是正确的。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12 | 20 | 30 | 0 |  |
| 8 | 12 | 2 | 0 |  |
| 34 | 70 | 37 | 4 |  |
| 112 | 100 | 25 | 12 |  |
|  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 20 | 30 | 30 | 0 |
| 70 | 70 | 37 | 4 |
| 112 | 100 | 37 | 12 |
| 112 | 100 | 25 | 12 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | 12 | 20 | 30 | 0 |
|  | 8 | 12 | 2 | 0 |
|  | 34 | 70 | 37 | 4 |
|  | 112 | 100 | 25 | 12 |

|  |  |  |  |
| --- | --- | --- | --- |
| 12 | 20 | 30 | 30 |
| 12 | 20 | 30 | 30 |
| 34 | 70 | 70 | 37 |
| 112 | 112 | 100 | 37 |