2逐步验证NC,DW,PW三种卷积方式

首先验证NC

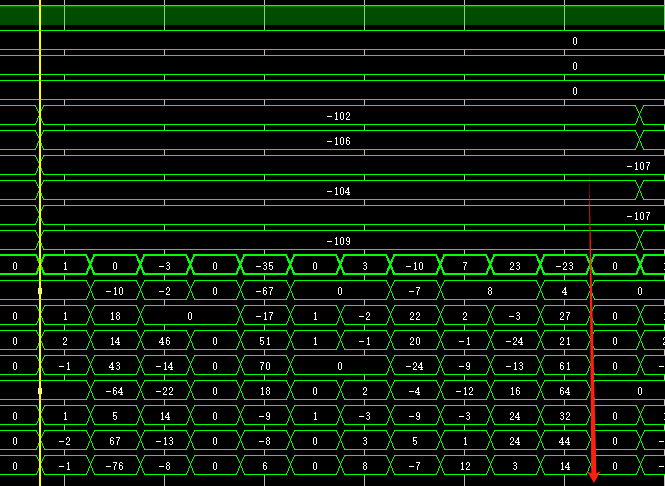
目前存在两个方面的问题：一是计算结果有误、二是写入地址和控制写入哪个bram的信号有问题。

首先研究计算结果出错的原因。

先查看每个mac array读入的输入和权重能否对上

根据mac array1 135ns读入权重，145ns读入data，具体数值都是对的。

除了第一个计算周期，后面的每个周期中，data的持续时间比weight多了一个时钟周期，多出来的时钟周期中weight为0，是否会影响写入？



接下来检查计算结果。

最终数据的写入由ofmap\_write和bram\_write\_wea决定。目前看起来wea这个控制信号是有问题的。输入数据一直没变，说明位置没变，但是wea有时会变为1或者2.

Bram写入的addr每三个周期一变是对的，因为每一个周期输出三个channel的结果，72bit的位宽可以存储9个channel的数值。Ofmap为（32，112，112）那么应该每个bram用四个地址来存储这32个channel的结果。地址顺序应该是5625-5626-5627-5628 然后改变wea开始写入下一个bram，并且地址再次回到5625.

目前看起来addr的变化是有问题的。时序对不上，并且有时候会跳到5630之类的数字。

先检查计算结果是否正确。根据ofmp 第一个输出的72位数应该是

{0，20，0，38，104，33，97，0，26}

那么mode arbiter中的输出应该为

Conv0 = 0，conv1 = 20，conv2 = 0

Conv0 = 38，conv1 = 104，conv2 = 33

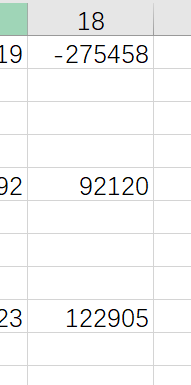
Conv0 = 97，conv1 = 0，conv2 = 26

Conv output全错

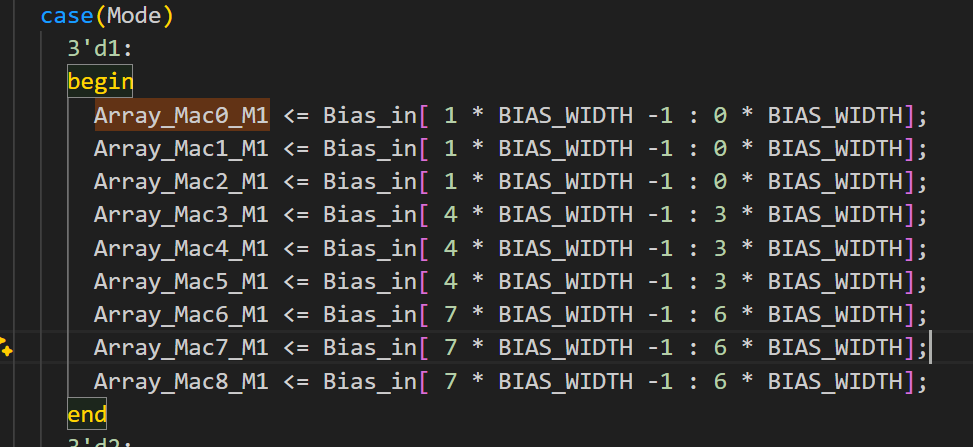
往前查找，先找出每个mac array计算出的结果

根据计算，第一个输出结果经过relu&shift之前的结果是-808279，这也是conv add0的结果。

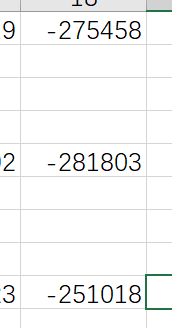
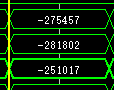
找到原因：第2、3个mac array的M1没有成功输入进去，刚好和没有进行+M1这一步的结果相同。原因应该是存储M1时的方式问题。

不知道为什么counter中，对于普通卷积模式，只有0，3，6三个mac array给m1赋值了？将其修改为

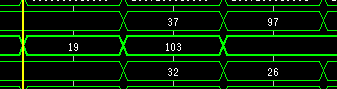


修改后结果与手工计算的结果几乎一致

然而最后输出的结果还是错误的，conv1=84，但是实际结果应该是20

原因在于多加了两个M1，所以之前的代码是对的，只是之前的代码中对于M1的位置有误。



{0，20，0，38，104，33，97，0，26}

输出结果普遍小了1，原因应该是实际硬件中移位操作代表向下取整，而numpy中使用了round四舍五入。改为floor之后，所有的值完全一样。

检查了第一个位置的所有channel结果，都与numpy模型输出的结果一致。

接下来要解决的问题是关于写入的时候，每三个channel的结果在ofmap\_write的位置不对。这个问题是由conv\_cnt这个计数器导致的。

在mode arbiter中添加一级延迟使其对齐。

还有wea和addr的问题尚未解决。

wea迟了一个周期。

将bram内容全部写出来进行检查。

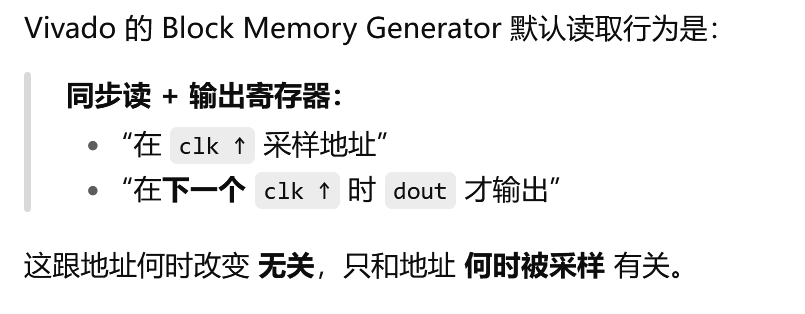
按照输入图像的尺寸，每个bram应该会占用75\*75=5625个地址，也就是说最后一个占用的地址应该是5624，但是bram中5625仍然是一个3channel的数字？

发现addr0和addr1的数字是相同的。

检查发现addr5625的数字与coe文件的最后一个数字是一样的。

根据检查，发现bram中存储的数据应该是正确的。但是有一个奇怪的现象，即写入地址为5625时，写入的却是bram的5626.刚好每个写入地址都+1了，反而导致结果是对的。

这个情况是由于bram的读出机制导致的。



实际上bram的输出就是会延迟一拍，因此导致我读出时误将地址0的数据当成地址1的数据读出来了。

解决方法：每个数据使用2个时钟周期来进行读出。

目前验证了bram写出时的数据量和地址是正确的，但是不确定是否每个数据都是正确写入bram。

如果想要验证所有数据是否正确写入bram，需要将所有bram的数据打印出来。然后使用python对数据进行处理，将其还原成输出的32\*112\*112的形状并与ofmap进行对比。

写出bram之后，发现很多bram中并没有存到对应的数据。发现bram\_write\_addr有问题。

窗口的第2行即3，4，5这三个位置，根据仿真结果可以看到，他的写入地址一直在5625-5628之间变化，没有实现递增，导致这几个位置被反复写入。+

Read\_scenarios有问题，导致地址偏移没有成功计算。

原因：计算偏移时，应该使用write\_scenarios，但是源代码使用了read\_scenarios

修改之后结果正确，并通过python代码将输出反写出来并与numpy模型进行比较，有5个位置的数据不同。

验证DW

首先将第二层的m1、weight写入coe文件

Data和weight有问题。Bram的读取地址有问题。有一些地址出现倒退。

Weight有问题。

对于data：addr送入，时钟上升沿到来后，过一个时钟周期，读出数据，再过一个T到达mac（这个T用于bram分配），mac立刻得出各个权重和输入的乘积，时钟上升沿到来，经过3个T得到累加结果，下一次上升沿到来计算m0\*output+m1.

现在存在的问题是sum\_vld、m1、m0不匹配。

如何对齐？目前尝试将mac array out延迟一拍

问题1：data没有正确变化

问题2：weight只正确读取了2次

Bram\_read\_addr2、5的问题在于偏移地址的计算上：（0，0）点计算时，实际上部分输入是padding，但是会使得这些bram的地址计算的块偏移为负数。而当read\_scenario改变后，还需要一个时钟周期来改变这些地址，导致地址错误。

将这一部分改为组合逻辑。改变后，普通卷积的结果不变。

但是由于padding\_scenario\_decoder仍需一个周期改变，导致实际输入的data仍是错的。

将其修改为组合逻辑。仍不会改变普通卷积的结果。

根据结果，似乎在根据padding\_scenario\_decoder分配输入数据时有误。

padding\_delay\_1 <= {v0, vl, h0, hl, stride};

怀疑是bram dma中read\_scenarios\_arbiter也延迟了一拍导致的。修改为组合逻辑（不行）

每一次变换后，可能是因为某个scenario的问题，导致数据虽然读进去了但是顺序不对。

的确是read\_scenario\_arbiter导致的。将原先的read\_scenario\_delay2送入case，结果普通卷积结果不变，说明可以减少一拍。

此时至少前几个送入的data没有问题。接下来检查kernel问题。

Kernel只有一拍是正常的，其他三个周期都是0.

地址变化有问题。跳转到了后面没有写入的地址。

kernel\_read\_addr1 <= Kernel\_base\_addr + kernel\_read\_base\_p + kernel\_base\_channel;

kernel\_read\_base\_p在dw模式下应该一直为0.但是这里却跳变了。

将mode送入kernel\_bram\_dma,并且使kernel\_read\_base\_p在dw模式下为0.

验证出来有接近9w个位置不同。

在仿真波形里面，至少前面几十个数据都是对的，但是在还原出来的数组中却不一样。两种可能：1写入错误 2还原错误

先检查写入，直接查看输出的bram，是错误的

发现除了bram0，其他的bram\_write\_addr都是错的。

bram1\_write\_addr = Ofmap\_base\_addr + bram\_write\_hori\_block\_addr\_1 + bram\_write\_vert\_block\_addr\_1 + Ofmap\_channel\_num;

错误的原因是第一层普通卷积结束后，bram\_write\_hori\_block\_addr\_1 + bram\_write\_vert\_block\_addr\_1这两个信号没有复位为0，仍然是普通卷积时最后变成的数字。需要一个指示信号来使其复原。

可以用sum\_vld, data\_bram\_ena, bram\_st

在bram\_dma中增加 当bram\_st = 0时，令bram\_write\_hori\_block\_addr、bram\_write\_vert\_block\_addr这些信号置零。

虽然初始地址的问题解决了，但是每四个周期的第一个周期的地址是错误的。



由于地址偏移需要一个周期来计算导致的。尝试将其改为组合逻辑。

修改后，本层输出中有14个地方不同。

现在解决第一层中有五个地方不同的问题。

位置[c=5,h=18,w=56]不同: hard=31, soft=30.0

位置[c=5,h=20,w=78]不同: hard=31,soft=30.0

位罡[c=5,h=36,w=109]不同: hard=31,soft=30.0

位置 [c=22,h=1,w=57]不同: hard=17,soft=16.0

位罡[c=22,h=28,w=57]不同: hard=17,soft=16.0

原因是m1在硬件中偏差了一位，例如channel=5时，正确的m1应该是2551435，但在硬件里是2551436.观察发现channel=14时也错了，正确的是-3672，但coe硬件里面是-3671.进一步发现是在coe文件里面就出错了。

最终发现是因为生成coe文件时，使用了不同的M1（M1 save和M1 cal）导致个别值有1的偏差，如果最终计算结果刚好在整数附近就是导致取整出现错误。

接下来验证pw

Pw中一个点需要8个时钟周期来进行计算，前四个时钟周期计算9个channel，后四个计算7个channel。

送入mac的data和weight都有问题，只有第一个周期是全对的，后面都是错误的。

先处理data问题，首先考虑是不是地址错误。发现有部分bram地址错误。

对于pw，每八个周期可以算出9个点的结果。那么ofmap的坐标不应该是一个一个点变化。

研究pw的计算：

Counter\_dw中，在pw模式下，mac\_array0的weight全部来自于kernel\_bram0。

Ifmap全部来自于window11，也即是3\*3窗口中间。

因此实际上pw是一个一个点算的，而不是九个点九个点算的。

八个周期处理一个点。但是mac array0里面只有前四个周期的data、weight是对的。

第五个周期的data、weight已经错了。

Bram0的addr是对的，bram0读出来的数据也是对的。那么应该是分配的时候出错了。

根据仿真图，read\_scenario\_arbiter还需要延迟两个周期。并设置一个mux来使得在pointwise情况下选择read\_scenario\_arbiter.

另一个问题：read\_scenarios四个周期一变，但是实际上应该要八个周期一变。

原因在于kernel\_cnt. 在counter\_dw中，当kernel\_cnt等于kernel\_num-1时，kernel\_cnt=0,但是在pointwise中kernel\_cnt在每个数字中应该持续四个周期。

需要一个信号进行指示。考虑用addr的ifmap\_channel\_num部分。可以让ifmap\_channel\_num ==3 时kernel\_cnt +=1. 这个信号正好就是pos\_z = channel\_cnt.

修改之后，输入坐标和输出坐标都正确。

但是data、weight仍然存在问题。

八个周期中，第五个周期时weight应该变化为另一个卷积核，但是仍然是之前的。第六至第八个周期才正确。怀疑是kernel的地址变化不及时导致的。

查看kernel地址，发现确实是这个问题。

kernel\_read\_addr0 <= Kernel\_base\_addr + kernel\_read\_base\_p + kernel\_base\_channel;

需要将kernel\_read\_base\_p提前一个周期。

修改了adder tree 9里面a9\_r2 a9\_r3的处理。原来的代码只能传递到a9\_r1

ReluandShift里面处理output\_zero的方式有点奇怪。

为什么output\_zero在ReluandShift模块里面是有符号数？mobilenetv2里面output\_zero都是正数或0.

将其修改为无符号数。

出现了另一个问题：在加上zero\_point之前为负数的结果，最后输出都是ff。

主要原因是无符号数和有符号数混用导致的。已修正。

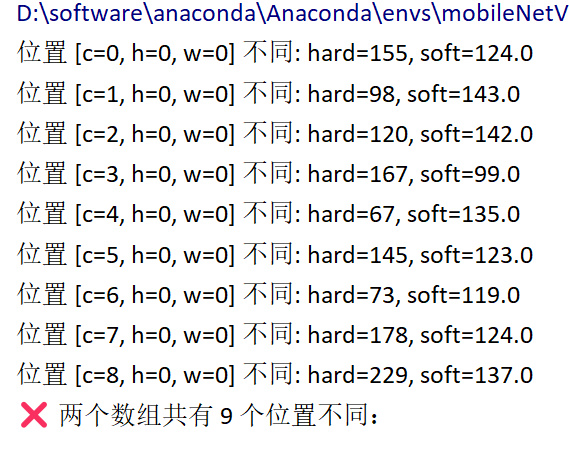
验证结果发现channel0-8都是错的，而channel9-15都是对的。

研究发现每八个周期计算一个点的时候，除整个pw的第一个点之外，剩余的点的八个周期的前三个周期的数值是前一个点的数值。明显属于分配问题。

再次观察仿真波形，并将read\_scenarios\_delay2作为判断信号。

再次验证，只剩下（0，0）点的前九个channel的数值出错。

这是由于在pw开始之前，有部分和储存在mac array中导致第一个结果计算是错误的。



有两种方法：1是在pw的正确地址输入之前，禁止读数据或是送入mac

2是在正确地址即将输入时，送一个指示信号使得mac中存储的数值归0

关键在于找到一个指示信号。

Tlast是mac的指示信号，在counter\_dw模块中这个信号被延迟了四个周期发出。可以设置一个mux使得当mode==2‘d3时，直接发出信号而不进行延迟。

不知为何只是将tlast信号提前了四个周期，却导致几乎所有数据错误。

原因在于tlast不仅会影响mac还会影响adder\_tree\_9，因此需要添加一个专门的tlast\_no\_sync输出到mac中再根据mode进行判断。

修改后，前三层全部通过。

Data bram的depth加深到81920.

修改data bram的地址线为位宽17位

第四、第五层验证完毕。

第六层出现错误。第三、第四层也是pointwise卷积，但是却没有错误。

mac array中出现了pw开始之前存储的错误的部分和。之前使用tlast处理的方法并非完美。在第三层中，tlast\_no\_sync正好出现在channel\_cnt=3的位置。第四层中出现在channel\_cnt=1的位置。经过观察发现，这个指示信号必须在channel\_cnt从0开始变化之后的第三个周期为1.因为，data从bram读出需要一个周期，经bram\_dma重新分配需要一个周期，第三个周期出现在MAC\_data\_in\_offset，这个周期结束时指示信号必须为1，使得mac中的累加结果从头开始计算。因此第三第四层的正确只是凑巧。尝试新建一个指示信号，使其在channel\_cnt为1时拉高并延迟两个周期输出到mac。

修改之后，前六层全部通过。



继续添加层数，添加到第九层。（第九层有shortcut，由于缺少shortcut模块，目前只能验证到此）

第七、第八层通过。但是第九层结果不对，每一个channel中，每个数和软件算出来的值之间有一个差值，不同的channel差值不同，但在同一个channel中几乎一样。

怀疑是M1存错了。发现程序中有两处存了M1，这应该就是问题来源。重新生成M1。

修改后前9层全部相同。

**Shortcut：**

接下来需要解决的问题是缺少shortcut模块。

shortcut模块

使用counter生成坐标

修改bram使其能够生成sc地址

读出来的数据分别送入sc模块，计算

再写回bram

问题：有些参数如何获得？sc计算需要：x、Fx，x层的零点，Fx层的零点，M0，M1，Sc零点。

X、Fx、M1由bram读出来，Fx层零点可以直接从配置中得到。

如何快速验证？如果用第九层验证速度太慢。可以第一层计算完设置为sc，然后观察控制信号是否正确变化。

1.控制信号：确保计算sc过程中不会突然跳到下一层计算

2.坐标计算：确认坐标是否正确变化

3.地址计算：确定sc地址是否正确

4.数据：确认输入数据是否正确（是否同步）

5.输出结果：是否正确

5.写回地址：确认地址是否正确（同步）

6.检查bram

一种可能可行的方案：定义mode=4为sc计算。这样可以解决控制信号问题以及参数问题。

坐标指示：不使用一个一个点，而是使用window的坐标。例如112\*112的形状被划分为38\*38个window。则直接使用0-37作为坐标。或者令坐标始终为3\*3窗口中央。所有bram统一地址即可。两个周期分别读入x、fx

Ifmap\_hori\_num和Ifmap\_vert\_num从1，1开始，每次加3。由bram\_dma中的read\_h\_block\_idx和read\_v\_block\_idx指示是否读到了最后一个block。

每一周期bram可以读出81个数据，转换为3\*3窗口的9个channel的数据。每次需要两周期来读，一个周期用来读x，另一个周期用来读fx。计算完成了就算出了3\*3窗口的9个channel的数据。当所有channel计算完成，跳转到下一窗口。每个窗口需要ceil(ifmap\_channel/9) \* 2 个周期来读取。因此需要两个信号，其中一个可以用channel\_num，指示当前窗口处理了几个周期，另一个只需在0、1之间反复变化即可。这个也同时可以传递到bram\_dma使其改变地址。

修改写入bram的数据。之前的程序中所有bram数据接到同一条线，通过控制信号决定哪个bram写入。但是sc中9个bram同时写入。需要将每个bram的写入数据单独接一条线，并且根据模式进行分配。

接下来修改写入坐标。对于sc，每次写入的时候，每个bram所使用的地址是一样的。对于其他模式，写入地址每次由write\_valid这个信号指示进行跳变。但是sc并没有使用这个信号，因此需要sc中输出一个信号来指示计算完成。可以将data\_src信号延迟3级作为指示信号。

Sc验证完成，1-9层以及第9层的sc验证完毕；

继续添加后面的层，发现10-12层没有问题。但是13层开始出现问题。

原因是central control的配置写错了修改后13-15层通过

但是15层的shortcut验证出来结果是错的。原因是sc的地址写错了。

修改之后1-39层验证通过。39sc有一个数值错误。检查发现在软件输出中这个值是-8，硬件是248。实际上-8的补码就是248，可能是因为软件没有使用无符号数输出导致的。

但是实际上relu操作应该将负数量化为0

51层中有一个软件输出为-14，硬件输出为255的数字

由于是负数，符号位为1。硬件判断为>255从而输出255

检查pytoch中的模型，发现输出并没有负数。因此模型的差别是造成这个问题的原因。

**Round：**

对模型进行修改：将原有的右移操作（floor）改为四舍五入（round）。这个操作并不困难，只需要在右移之前加上2^15即可，也就是1000 0000 0000 0000 .若原结果的第[15]这一位为1，则会产生进位，否则舍去。

修改之后第一第二层有个别数字差1，第三层出现大量数字变成255.

第1、2层的问题是由于M0 M1未更新导致的。观察发现第三层出现255的数值正确的输出都是小于其输出零点130的数字，也就是在加零点前都是负数。

发现问题出于移位之后没有保留符号位，正确的结果高十六位应该都是1，实际结果都是0.原因是括号问题，没有正确进行符号拓展。

接下来发现仍然有些数字不同，判断是因为：Python 的round函数采用 \*\*“四舍六入五成双”**（银行家舍入法），而硬件的 “加偏移量右移” 采用**“四舍五入”\*\*（无条件进位），这是最主要的差异来源。检查发现出错的部分后十六位数确实是1000\_0000\_0000\_0000 .

因此还需要添加一个判断小数以及奇偶数的部分。

当后十六位数是1000\_0000\_0000\_0000时，判断msm\_shifted[16] == zero\_point[0]。如果他们相等，最终输出会是奇数，因此需要减掉1.

发现15层sc有问题，有三个数不同，±1的偏差。Sc层的输入应该全部都是正数，正确的输出都是奇数，说明不是因为没有加入银行家舍入法。检查发现可能是因为M0 M1改成round之后没有改对应的值。修改后通过。

第39层sc出错，有一个本应输出是0的数字输出成了250.判断是因为shortcut模块没有进行clamp。添加clamp逻辑。

修改后1-50层验证通过。

51层出现了5个输出为0的数字，但是本应输出200+

原因是软件实现的时候，没有做clamp。添加后51层通过。52层M0没修改，修改后通过。

**Average pooling：**

接下来还需要处理average pooling 层。将1280\*7\*7的输入平均量化为1280\*1\*1.也就是将49个数字相加并求平均。

这一个操作并不容易，因为原有的pe和控制逻辑的思路都是固定某个点，将这个点所有的通道都处理完毕，再处理下一个点。

现在有两个思路：

1继续利用原有的pe和控制逻辑，增加一种新的模式average pooling。然后单独编写这个模式的坐标变化、channel cnt的变化，将输出送入一个新的模块average pooling中，这个模块内置于mac array里面。这样做的好处是在读出和存储数据的时候，坐标的计算只需要做简单更改。

2.完全重新写一个模块位于system top中，坐标计算、平均池化运算，控制信号都由这个average pooling模块单独控制。

综合看起来，还是第一个思路更简单一些。先尝试一下第一个方案。

那么如何快速验证编写出的模块的正确性呢？现在要跑到52层结束需要至少20分钟，时间太久了。

直接尝试针对输入图像做平均池化。只需设置9个3\*3窗口进行验证即可。也就是算一个9\*9图像的池化，三个通道。只需要检查这三个结果是否正确即可。

每一次处理九个通道，用3\*3的窗口读出所有需要的数据，送入pe进行累加，最后一个数据送出后，写一个控制信号，将累加结果送入除法器，并将累加结果清零。

有一个问题：现有的mac array送入的数据分别是3\*3块的各个位置的数值。但average pooling要求把这些数据都送入一个计算单元中。

很容易得到解决：在counter\_dw模块中，存在对这些数据重组的case语句，只需要在average pooling的case中重组这些数据再送入即可。

为了将1280\*7\*7的输入平均池化为1280\*1\*1的输出。只需要：9个周期读出9个3\*3窗口的数据，九个mac array分别收到九个通道中的所有3\*3的值，将这些值全部相加并存入寄存器中。等待最终的信号。（是否可以利用已有的adder\_tree\_9?）等待累加和除法完成后，送回mac\_array\_x9进行输出。

首先修改IHB IVB。逻辑和sc的逻辑很像，都是令IHB IVB处于每个3\*3窗口的中心。不同的点在于sc是计算完所有channel再跳到下一个坐标，而ap是将前9个channel的所有坐标都遍历之后再跳转channel。先后顺序不同。

修改后验证，发现前三个周期读出来是对的，后面就错了。发现是因为第一层的地址偏移不同，本想读出9\*9，最终读出了第一行的9个3\*3窗口，即3\*27.不过用来调试已经足够。

目前数据读出没有问题。下一步尝试送入adder\_tree\_9。

送入adder\_tree\_9只需要添加mux选择输入，加法树内部始终保持运算。

接下来需要编写avgpool模块。这个模块始终接收加法树的结果并进行累加，当本通道的最后一个加法树结果送入avgpool模块，发出控制信号，使其开始做除法，并将累加结果清零。

写完并验证完avgpool模块，接下来考虑如何写回。

首先需要一个写回信号，使用avgpool\_out\_vld.然后处理写回坐标.写回的坐标始终为0,只有写回channel发生变化.

验证发现最初的9个channel算错了,猜测是因为avgpool层刚开始时有一些别的数据没有清掉.添加新的控制信号后验证通过.

第54层是一个pw将1280\*1\*1的输出分类成1000\*1\*1

不知为何bram\_reshape出来结果都是0.发现是因为bram0.txt最后几个地址的数据都是x,导致被跳过.为什么都是x?

发现最后一段kernel input都是x。发现原因是kernel的深度不够

将x修改后比较发现只有最后那些是x的地址的数据不同.

最后做一个argmax把前五的类别打印出来即可。