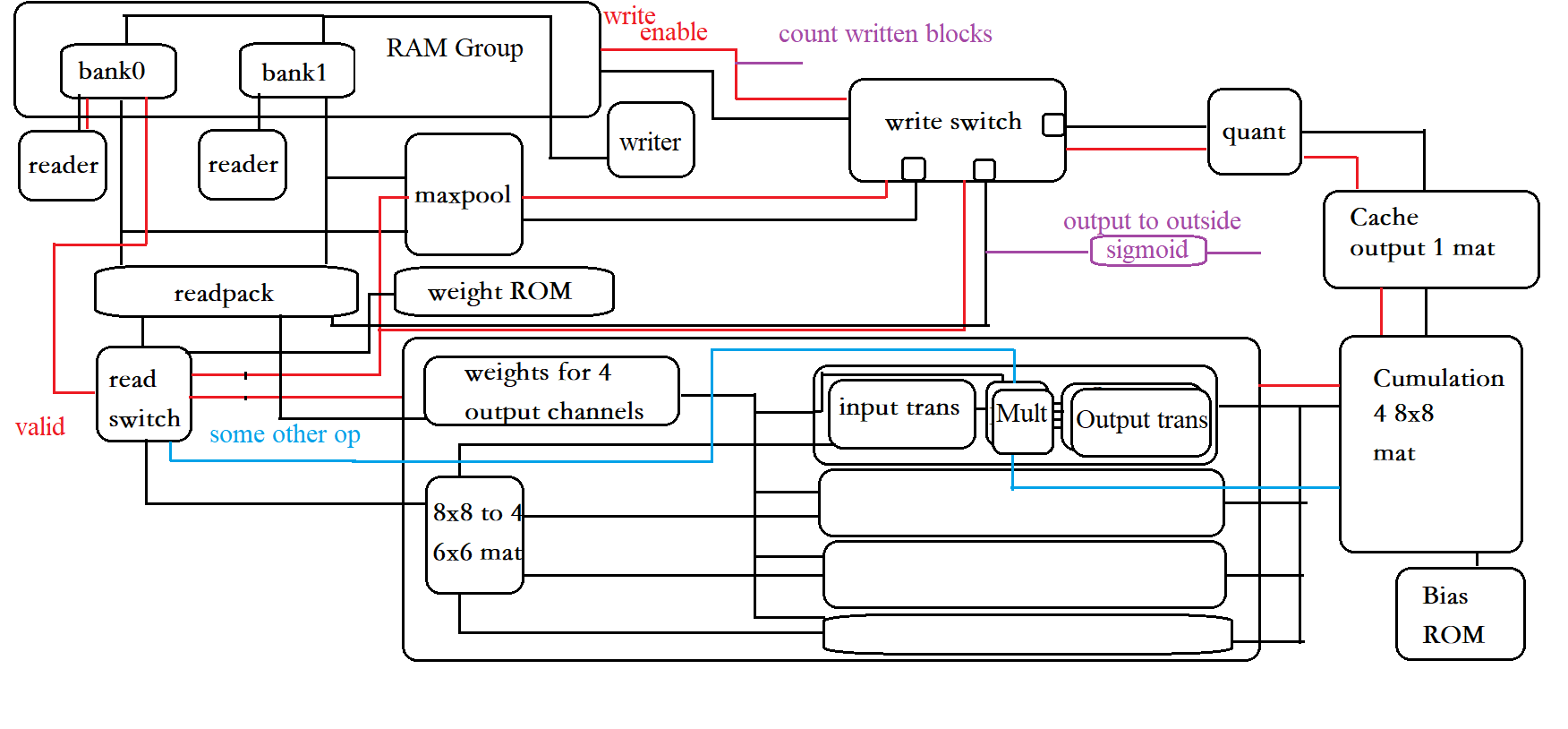
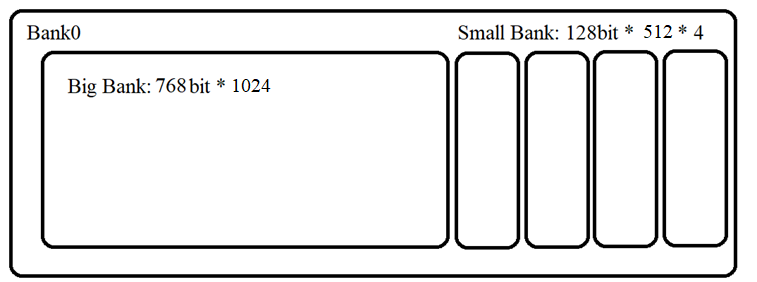
**人工智能芯片入门终期报告**

小组成员：袁方舟

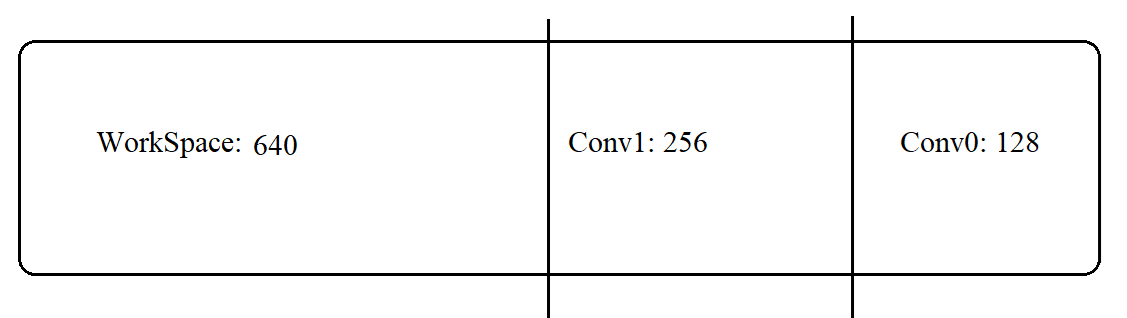
1. **系统架构**



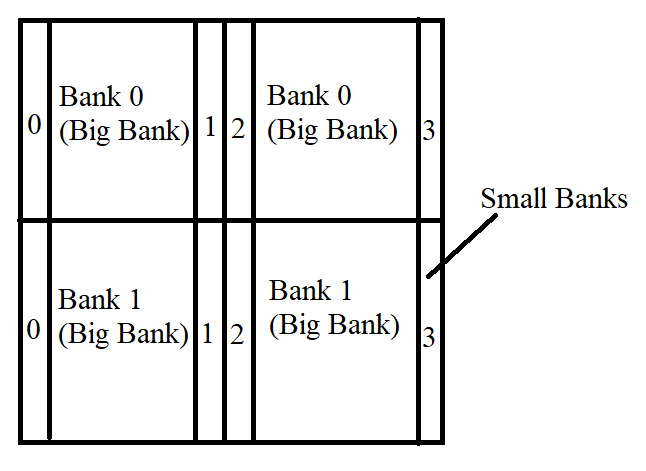
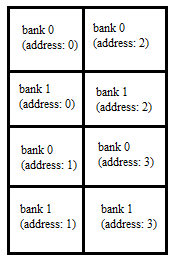
1. 对于卷积，计算流程大致如下：从BRAM里读入数据（模块名：RAMGroup），在readpack（模块名： PackReadData）处组合成8x8矩阵以及该矩阵周围一圈。此时weightROM自动读入所需的卷积核。然后由read switch（模块名：ReadSwitch）将数据和原始3x3卷积核发到主计算单元（模块名：Calc8x8），主计算单元输出8x8的结果矩阵到累加单元做累加运算（模块名：Accumu），累加单元会自动在合适的时刻读入bias。累加完毕后输出到一个cache（模块名：CacheWriter），这个cache支持每次输入若干个8x8矩阵，但只输出一个8x8矩阵。它将矩阵输出到量化单元（模块名：Quant），量化单元将结果输出到write switch（模块名：WriteSwitch），它从3个输入中选择有效的那个输出到BRAM。此时writer同时输出对应的写地址（模块名：RealWriter）。
2. 对于Maxpool，数据会从BRAM直接到达Maxpool（模块名： Maxpool），然后到write switch（模块名：WriteSwitch），然后写到BRAM。
3. 对于Upsample，数据将从BRAM到readpack（模块名：PackReadData），然后到read switch（模块名：ReadSwitch），然后直接被发到write switch（模块名：WriteSwitch）。
4. 对于矩阵按位乘和LeakyReLU，数据流和卷积一样，但是累加单元和Calc8x8会特殊设置为直接乘或ReLU模式以及直接通过。矩阵按位乘还需要特殊的读入模式。
5. 对于最终数据的输出，将直接从read pack（模块名：PackReadData）的输出提取并输入到Sigmoid模块，Sigmoid模块是一个流水线，每一级数据都会与对应的斜率和分界点进行计算。
6. 8x8在BRAM的储存模式和读入、输出模式如下：为了在卷积的时候快速读入8x8矩阵周围一圈的数，我们将8x8矩阵拆成左边一列8个、右边一列8个、中间6列48个的形式，而且矩阵分行的奇偶性、列的奇偶性存在不同的bank里。中间的矩阵需要分行的奇偶性，所以需要2个BRAM，左右需要行的奇偶性和列的奇偶性，所以需要8个BRAM，一共10个BRAM。具体实现上将行奇偶性相同的视为一组，一个reader（模块名：GraphReader）管理一组。我们称一组为一个Bank。一组的结构如下：



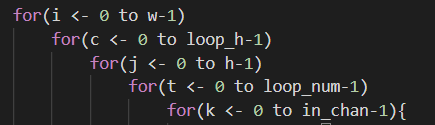
工作区、conv0和conv1的存储区域如下，其中conv0, conv1的大小刚好符合它们数据本身的大小，工作区比理论最高大小512大一点。一个原因是从512提升到640不耗费更多的BRAM，另一个原因是更多的冗余可以避免数据覆盖的错误。



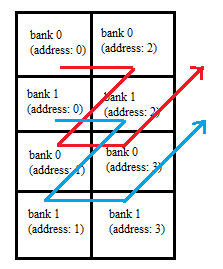
储存矩阵的模式如下（注意我们假设所有输入矩阵的长宽都可以被16\*16整除）：我们首先将输入通道视为2W\*2H组，其中2W是行数量，2H是列数量，每组有输入通道个数个8x8矩阵。然后按如下方法存储：对于(2x,2y)的组存到Bank0的Big Bank, Small Bank0, Small Bank1；对于(2x+1, 2y) 的组存到Bank1的Big Bank, Small Bank0, Small Bank1；对于(2x, 2y+1) 的组存到Bank0的Big Bank, Small Bank2, Small Bank3；对于(2x+1, 2y+1) 的组存到Bank1的Big Bank, Small Bank2, Small Bank3。对于W>1的情况，我们先按列存储，再按行存储，所以可以将2W个组视为一个大组。这个大组按照上述方式分为若干连续的部分存储到Bank里。图示如下：



有两种读入模式，一种是按照列数、行数、输入通道数读，其中有两个参数loop\_h和loop\_num。图示如下：



另外一种是将列两两视为一组，然后进行Z字形读取。图示如下：



1. 读入每个8x8矩阵的方法：每次在一个bank读一组，然后就切换到另一个bank读。每次读1个Big bank和2个Small bank这可以通过调整参数实现：将loop\_num设置为2，将Bank0（行编号偶数）的初始loop\_num设置为1。这样除了第一组以外每个组的地址都会重复输出两遍，方便读入。
2. 读入8x8矩阵周围一圈的方法：每次在一个bank读一组，然后就切换到另一个bank读。每次读所有Big bank和Small bank。这样可以得到8x8矩阵本身以及它左右的一列。在读一个bank的时候，如果它的行数不是最大，那么另一个bank输出的地址刚好是它下一行的对应输入通道。将loop\_num设置为2\*重复次数（重复次数用于计算卷积，它等于输出通道数除以并行度），将Bank0（行编号偶数）的初始loop\_num设置为1\*重复次数 就可以达到这种效果。于是可以很方便地得到8x8矩阵下面的一行。8x8矩阵上面一行通过记录上次8x8矩阵的最后一行得到。
3. 写8x8矩阵的方法：每次在一个bank写一组，然后就切换到另一个bank写。这非常容易实现，只需要按照列、行、输出通道的顺序依次递增即可。
4. 上采样的方法是：由于上采样本质是每个8x8矩阵都变成4个8x8矩阵，所以可以将loop\_num设置为4，将loop\_h设置为2，将Bank0（行编号偶数）的初始loop\_num设置为2。这样它就会重复读入每一列2遍，每一组2遍。
5. 实现矩阵按位乘的方法：它需要每次读入一个矩阵的8x8矩阵后立即跳到另一个矩阵的对应位置。这通过每个Bank再增加一个reader实现，设置一个标志每次轮换得到。
6. 拼接矩阵的方法：我们在写地址产生的时候增加两个参数al,ar表示通道数的范围，然后将矩阵A读入后直接输出到前32个通道，矩阵B读入后直接输出到后32个通道。拼接矩阵的结果为256的8x8矩阵。由于BRAM支持640个8x8矩阵，一定能装下结果。
7. 做卷积运算时读地址和写地址的确定：维护一个地址（这个在生成Verilog的时候维护），初始为0，每次矩阵开头就在这个地址，写地址设为矩阵末尾。由于BRAM支持640个8x8矩阵，而写必须要覆盖到当前读位置的上一列才会产生影响，且最大矩阵大小为512个8x8矩阵，所以没有问题。而每次到最后输出结果的时候将写地址置为0，就可以开启下一次循环。
8. 最终输出结果：每次通过read pack等读入64个数，然后用64个时钟周期输出这些数到Sigmoid，再流水线输出。FinalTop里用了一些方法让输出的同时也可以读入，但由于输出比读入多得多，所以并不能完全达到流水线的效果。
9. 几乎完全的参数化：电路使用chisel3.4.3编写，几乎所有地方的参数都由构造函数的输入决定。事实上，正是得益于此，当一开始设计的4并行无法成功布线的时候，只修改了一个参数，就降低了并行度，大大减少了工作。本电路所用到的参数如下：

|  |  |  |
| --- | --- | --- |
| 参数名 | 值 | 作用 |
| w | 16 | 数据宽度 |
| bias\_w | 36 | bias宽度 |
| bias\_addr\_w | 8 | bias rom 地址宽度 |
| weight\_w | 144 | 卷积核9个数的总宽度 |
| weight\_addr\_w | 13 | weight rom 地址宽度 |
| big\_w | 16 | 任何需要很大数字的计数器，比如总迭代次数等等 |
| h\_w | 3 | 矩阵行数/8 |
| addr\_w | 10 | 地址宽度 |
| c\_w | 6 | 通道数宽度 |
| id\_w | 8 | Small Bank编号（实现上Small Bank用one-hot vector编码） |
| dsp\_w | 18 | DSP的乘法器宽度 |
| para\_num | 2 | 并行度 |
| big\_global\_size | 1024 | Big Bank大小 |
| small\_global\_size | 512 | Small Bank大小 |
| big\_global\_addr | big\_global\_size-1 | 最大地址 |
| big\_max\_addr | big\_global\_size-128-256-1 | 最大工作区地址 |
| big\_conv0\_addr | big\_global\_size-128 | Conv0区最大地址 |
| big\_conv1\_addr | big\_global\_size-128-256 | Conv1区最大地址 |
| small \_global\_addr | small\_global\_size-1 | 最大地址（Small Bank） |
| small \_max\_addr | small \_global\_size-128-256-1 | 最大工作区地址（Small Bank） |
| small \_conv0\_addr | small \_global\_size-128 | Conv0区最大地址（Small Bank） |
| small \_conv1\_addr | small \_global\_size-128-256 | Conv1区最大地址（Small Bank） |

理论上DSP个数和LUT个数可以支持4并行的架构（4并行只需要3万左右时钟周期），但考虑到4并行不仅布线困难，而且频率也许会降低很多，所以最终还是选择了2并行的架构。这样主计算单元只使用了368个DSP，其他多余的DSP可以放在其他模块使用，如累加单元。

1. 各模块作用以及参数列表如下：
   1. RAM Group对10个BRAM的接口进行封装。它输入读写地址和写数据并输出读数据。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | chisel类型 | 位宽 | 作用 |
| rd\_valid\_in | Bool | 1 | 输入读使能信号 |
| rd\_valid\_out | Bool | 1 | 输出读使能信号 |
| rd\_addr1 | AddressReadGroup(addr\_w, id\_w) | 5\*(addr\_w+id\_w) | 输入读地址1 |
| rd\_addr2 | AddressReadGroup(addr\_w, id\_w) | 5\*(addr\_w+id\_w) | 输入读地址2 |
| rd\_big | Vec(2, BigBankReadData(w)) | 96\*w | 输出读数据1（来自Big Bank） |
| rd\_small | Vec(2, Vec(4, SmallBankReadData(w))) | 64\*w | 输出读数据2（来自Small Bank） |
| wr\_addr | AddressWriteGroup(addr\_w, id\_w) | 3\*(addr\_w+id\_w) | 输入写地址 |
| wr\_valid\_in | Bool | 1 | 输入写使能信号 |
| to\_bigbank | BigBankWriteData(w) | 48\*w | 输入写数据1 |
| to\_smallbank | Vec(2, SmallBankWriteData(w)) | 16\*w | 输入写数据2 |

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | chisel类型 | 位宽 | 作用 |
| valid\_in | Bool | 1 | 输入读使能信号 |
| valid\_out | Bool | 1 | 输出读使能信号 |
| job | ReadJobs(addr\_w, h\_w, c\_w, id\_w, big\_w) | (addr\_w\*5+id\_w+h\_w\*2+c\_w)\*5+4\*c\_w+1+(4\*c\_w+big\_w+id\_w+2)\*4 | 读地址参数 |
| job2 | ReadJobs(addr\_w, h\_w, c\_w, id\_w, big\_w) | (addr\_w\*5+id\_w+h\_w\*2+c\_w)\*5+4\*c\_w+1+(4\*c\_w+big\_w+id\_w+2)\*4 | 读地址参数2（矩阵按位乘时使用） |
| signal | Bool | 1 | 是否启用读地址参数2（矩阵按位乘时使用） |
| job\_type | ReadType | 1 | 读取模式：Copy还是Maxp |
| to\_banks | AddressReadGroup(addr\_w, id\_w) | 5\*(addr\_w+id\_w) | 输出读地址 |
| flag\_job | Bool | 1 | 是否开始设定参数 |

* 1. Reader（模块名：GraphReader）输出读地址。一个RAMGroup需要两个Reader。
  2. read pack（模块名：PackReadData）将读入的原始数据打包成可用格式。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | chisel类型 | 位宽 | 作用 |
| valid\_in | Bool | 1 | 输入读使能信号 |
| valid\_out | Bool | 1 | 输出读使能信号 |
| job | PackJob(w, h\_w, c\_w, big\_w) | h\_w\*2+c\_w+big\_w | 参数 |
| from\_big | Vec(2, BigBankReadData(w)) | 96\*w | 输入读数据1（来自Big Bank） |
| from\_small | Vec(2, Vec(4, SmallBankReadData(w))) | 64\*w | 输入读数据2（来自Small Bank） |
| output | PackedData(w) | 100\*w | 输出数据：8x8矩阵以及周围一圈 |
| flag\_job | Bool | 1 | 是否开始设定参数 |

* 1. read switch将read pack的输出导向合适的地方。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | chisel类型 | 位宽 | 作用 |
| valid\_in | Bool | 1 | 输入使能信号 |
| flag\_job | Bool | 1 | 是否开始设定参数 |
| job | ReadSwitchType | 3 | 任务类型，有6种：idle（什么都不做）,toConv（输出到卷积）,toMult2（输出到矩阵按位乘）,toMaxp（输出使能信号到Maxpool）,toCopy（直接输出到write switch）,toUps（执行上采样操作后输出到write switch）。 |
| in\_h | UInt(h\_w.W) | h\_w | 矩阵的列大小 |
| in\_chan | UInt(c\_w.W) | c\_w | 矩阵的输出通道数 |
| valid\_out\_calc8x8 | Bool | 1 | 输出到Calc8x8的使能信号 |
| valid\_out\_copy | Bool | 1 | 输出到write switch的使能信号 |
| valid\_out\_maxp | Bool | 1 | 输出到Maxpool的使能信号 |
| from | PackedData(w) | 100\*w | read pack传来的数据 |
| from\_weight | Vec(para\_num, Vec(9, SInt(16.W)) | 144\*para\_num | 卷积核数据 |
| to\_calc8x8 | PackedData(w) | 100\*w | 传给主计算单元的数据 |
| to\_copy | QuantedData(w) | 64\*w | 传给write switch的数据 |
| to\_weight | Vec(para\_num, CalcWeightData()) | 256\*para\_num | 传给主计算单元的卷积核 |
| to\_multmap | Vec(4, CalcWeightData()) | 4\*256 | 传给主计算单元的权重，仅当矩阵按位乘时使用。 |
| valid\_in\_from\_user | Bool | 1 | 使能信号，读入神经网络的输入数据时使用 |
| data\_in\_from\_user | QuantedData(w) | 64\*w | 数据，读入神经网络的输入数据时使用 |

* 1. Calc8x8是主计算单元。它能计算LeakyReLU、矩阵按位乘和3x3卷积。Calc8x8单元由4个Calc6x6单元组成，执行6x6对3x3 winograd。每个单元里有para\_num个组。每次输入para\_num个卷积核和矩阵后，将矩阵切割为4个6x6矩阵下方到各单元进行transform，同时卷积核也进行转换。通过计算流水线时刻，在同一时间转换后的卷积核和对应的矩阵输入到PE。PE有两种，一种是乘法和几个Mux，用来执行两个数乘法或者LeakyReLU。另外一种是复数乘法，需要3个乘法器。由于乘法器位宽为25x18，而winograd转换后将超出18位，所以在这里必须将它截取（与范围取max和取min）后再输入。对于输出同时执行output transform，这样一共有4\*para\_num个4x4矩阵的转换（比输入矩阵和卷积核的转换多）。最终将4个4x4矩阵拼起来为一个8x8矩阵，输出para\_num个8x8矩阵。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | chisel类型 | 位宽 | 作用 |
| valid\_in | Bool | 1 | 输入使能信号 |
| valid\_out | Bool | 1 | 输出使能信号 |
| flag | CalcType | 2 | 计算模式 |
| mask | UInt(para\_num.W) | para\_num | para\_num个输出口中哪些是有效输出 |
| weight | Vec(para\_num, CalcWeightData()) | 256\*para\_num | 卷积核 |
| multmap | Vec(4, CalcWeightData()) | 4\*256 | 权重 |
| output | Vec(para\_num, RawData(w)) | para\_num\*64\*(w\*2+4) | 输出para\_num个结果 |

* 1. 累加单元（模块名：Accumu）输入矩阵并累加。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | chisel类型 | 位宽 | 作用 |
| valid\_in | Bool | 1 | 输入使能信号 |
| valid\_out | Bool | 1 | 输出使能信号 |
| flag\_job | Bool | 1 | 是否开始设定参数 |
| in\_from\_calc8x8 | Vec(para\_num, RawData(w)) | para\_num\*64\*(w\*2+4) | 来自Calc8x8的结果 |
| result | Vec(para\_num, AccumuRawData(w)) | para\_num\*64\*(w\*2+10) | 输出结果 |
| csum | UInt(addr\_w.W) | addr\_w | 加多少次 |
| bias\_begin\_addr | UInt(addr\_w.W) | addr\_w | ROMHalfBias/ROMBias的读取开始地址 |
| bias\_end\_addr | UInt(addr\_w.W) | addr\_w | ROMHalfBias/ROMBias的读取结束地址 |
| bias\_addr | UInt(addr\_w.W) | addr\_w | 输出到ROMHalfBias/ROMBias的地址 |
| bias\_in | Vec(para\_num, SInt(bias\_w.W)) | para\_num\*bias\_w | 输入bias |
| is\_in\_use | Bool | 1 | 标记是否直接通过 |

* 1. 量化单元（模块名：Quant）转换进制，量化结果。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | chisel类型 | 位宽 | 作用 |
| valid\_in | Bool | 1 | 输入使能信号 |
| valid\_out | Bool | 1 | 输出使能信号 |
| flag\_job | Bool | 1 | 是否开始设定参数 |
| in\_from\_accumu | AccumuRawData(w) | 64\*(w\*2+10) | 接受来自CacheWriter的结果 |
| result | QuantedData(w) | 64\*w | 输出结果 |
| quant\_in | UInt(5.W) | 5 | 进制转换时的右移数 |

* 1. Write switch（模块名：WriteSwitch，似乎被Synthesis消除了）

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | chisel类型 | 位宽 | 作用 |
| valid\_in | Bool | 3 | 输入使能信号 |
| valid\_out | Bool | 1 | 输出使能信号 |
| input | Vec(num, QuantedData(w)) | 3\*64\*w | 输入数据 |
| output | QuantedData(w) | 64\*w | 输出数据 |

* 1. Maxpool（模块名：Maxpool）

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | chisel类型 | 位宽 | 作用 |
| valid\_in | Bool | 1 | 输入使能信号 |
| valid\_out | Bool | 1 | 输出使能信号 |
| from\_big | Vec(2, BigBankReadData(w)) | 96\*w | 输入读数据1（来自两个Bank的Big Bank） |
| from\_small | Vec(2, Vec(2, SmallBankReadData(w))) | 32\*w | 输入读数据2（来自两个Bank的Small Bank） |
| result | QuantedData(w) | 64\*w | 输出数据 |

* 1. CacheWriter 将输入数据做一个缓冲，输入para\_num个矩阵，输出一个矩阵。

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | chisel类型 | 位宽 | 作用 |
| valid\_in | UInt(para\_num.W) | para\_num | 输入使能信号 |
| valid\_out | Bool | 1 | 输出使能信号 |
| in\_from\_accumu | Vec(para\_num, AccumuRawData(w)) | para\_num\*64\*(w\*2+10) | 接受来自Accumu的结果 |
| out | AccumuRawData(w) | 64\*(w\*2+10) | 输出数据 |

* 1. Writer（模块名：RealWriter）实现上输入数据，进行简单的加工（简单的连线）以及准备好对应的地址后输出到RAMGroup

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | chisel类型 | 位宽 | 作用 |
| valid\_in | UInt(para\_num.W) | para\_num | 输入使能信号 |
| valid\_out | Bool | 1 | 输出使能信号 |
| flag\_job | Bool | 1 | 是否要设定参数 |
| job | RealWriteJobs(addr\_w,h\_w,c\_w,id\_w) | ((addr\_w\*5+id\_w+h\_w\*2+c\_w)\*3+c\_w+c\_w\*2+id\_w\*2)\*2+c\_w | 读入地址参数 |
| to\_banks | AddressWriteGroup(addr\_w, id\_w) | 3\*(addr\_w+id\_w) | 输出写地址 |
| to\_bigbank | BigBankWriteData(w) | 48\*w | 输出写数据1 |
| to\_smallbank | Vec(2, SmallBankWriteData(w)) | 16\*w | 输出写数据2 |

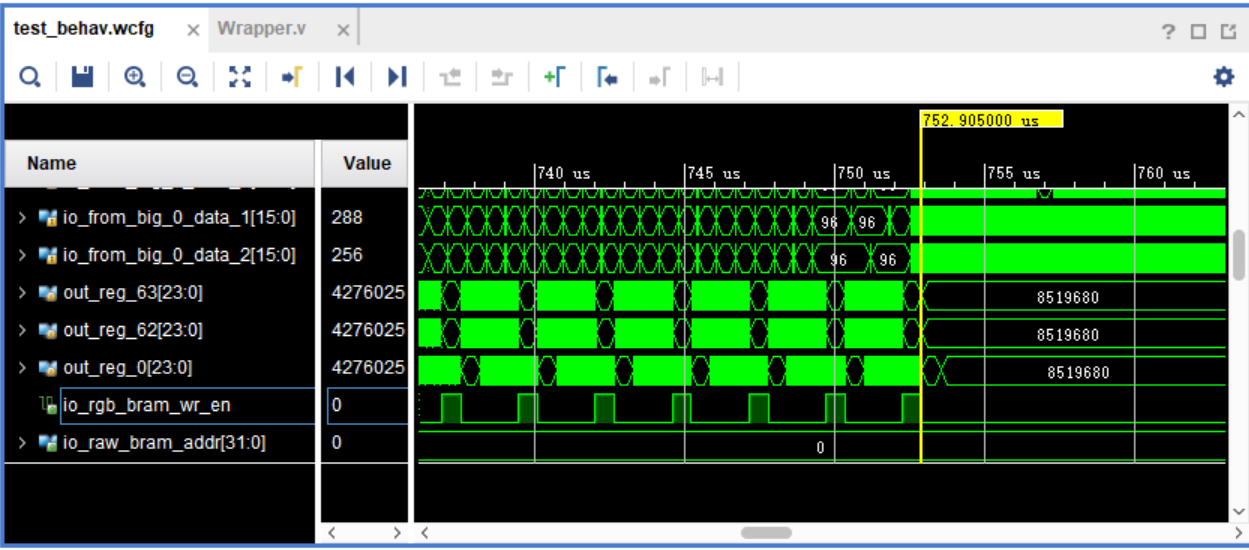
* 1. ROMHalfBias、ROMHalfWeight等直接按照ROM接口组织。
  2. Wrapper是对于NPU的一个包装，它的输入输出参数和raw\_read相同。由于NPU内部数据存储格式与外面不同，它还需要对地址等做一个转换。
  3. Sigmoid模块输入输出参数如下

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | chisel类型 | 位宽 | 作用 |
| valid\_in | Bool | 1 | 输入使能信号 |
| valid\_out | Bool | 1 | 输出使能信号 |
| input | SInt(16.W) | 16 | 输入数据 |
| output | SInt(8.W) | 8 | 输出最终结果 |

1. **硬件资源**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 模块名 | LUT | FF | BRAM | DSP |
| Sigmoid | 1188 | 1173 | 0 | 30 |
| Accumu | 10882 | 69 | 0 | 128 |
| CacheWriter | 2715 | 5378 | 0 | 0 |
| Calc8x8 | 55772 | 63531 | 0 | 368 |
| GraphReader | 1325 | 613 | 0 | 0 |
| GraphReader\_1 | 1402 | 649 | 0 | 0 |
| Maxpool | 0 | 513 | 0 | 0 |
| PackReadData | 3077 | 2948 | 0 | 0 |
| Quant | 10093 | 2802 | 0 | 0 |
| RAMGroup | 10916 | 2618 | 76 | 0 |
| ReadSwitch | 2075 | 1043 | 0 | 0 |
| RealWriter | 377 | 186 | 0 | 0 |
| ROMHalfBias | 0 | 0 | 1 | 0 |
| ROMHalfWeight | 490 | 22 | 40 | 0 |
| WeightReader | 28 | 24 | 0 | 0 |
| FinalTop | 144 | 408 | 0 | 0 |
| Wrapper | 135 | 189 | 0 | 0 |
| 工程框架本身耗费资源 | 15510 | 36695 | 78 | 52 |
| Synthesis后共计（策略使用Default） | 116129(67.56%) | 120034(34.91%) | 195(39%) | 578(64.22%) |
| Implementation后共计（策略使用Flow\_RunPhysOpt） | 111402(64.81%) | 117556(34.14%) | 185(37%) | 578(64.22%) |

1. **仿真结果**



最终误差如下（raw\_data20.txt）：

1. 最大通道误差：56(21.88%)
2. 最大像素距离误差（以三维空间欧氏距离度量）：3.146%
3. 平均通道误差：4.4299(1.73%)
4. 不同的通道数量：85.34%
5. **FPGA实验结果**

（线下演示）

|  |  |
| --- | --- |
| 时钟频率 | 125MHz |
| 能否正确输出模型结果 | 是 |
| 吞吐率（需要包括图片读入和结果输出） | 1733张图片/秒 |
| 是否摄像头-显示器端到端 | 是/否 |

1. **分工**

|  |  |  |
| --- | --- | --- |
| 组员姓名 | 分工 | 贡献占比（%） |
| 袁方舟 | 编写电路和软件模型，测试 | 100% |
|  |  |  |
|  |  |  |

1. **关于提交文件**

提交了vivado工程和chisel工程，chisel工程可以使用sbt编译运行，并在指定路径下生成Verilog文件。如果要生成，请打开src/main/scala/top.scala，将XDriver下的路径改为指定路径，并在chisel-b/fpga\_final\_project/下运行sbt “runMain final\_project.XDriver”。如果要测试，请运行sbt “test”，这会运行Calc8x8和读写地址的测试。另外，125MHz的版本在目录下的design\_1\_wrapper\_100mhz\_125mhz.bit，运行SDK时务必选择这个版本。