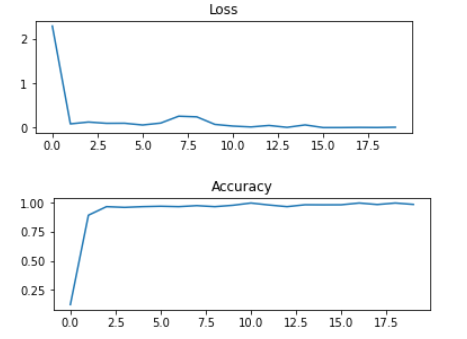
指令集的粗粒度和细粒度：怎么去思考：实际上每个原语都实现一整层，但现在的网络远比这个大，而且另一方面这个架构没有流水，写的时候感觉就很简单，没有什么时序上的问题

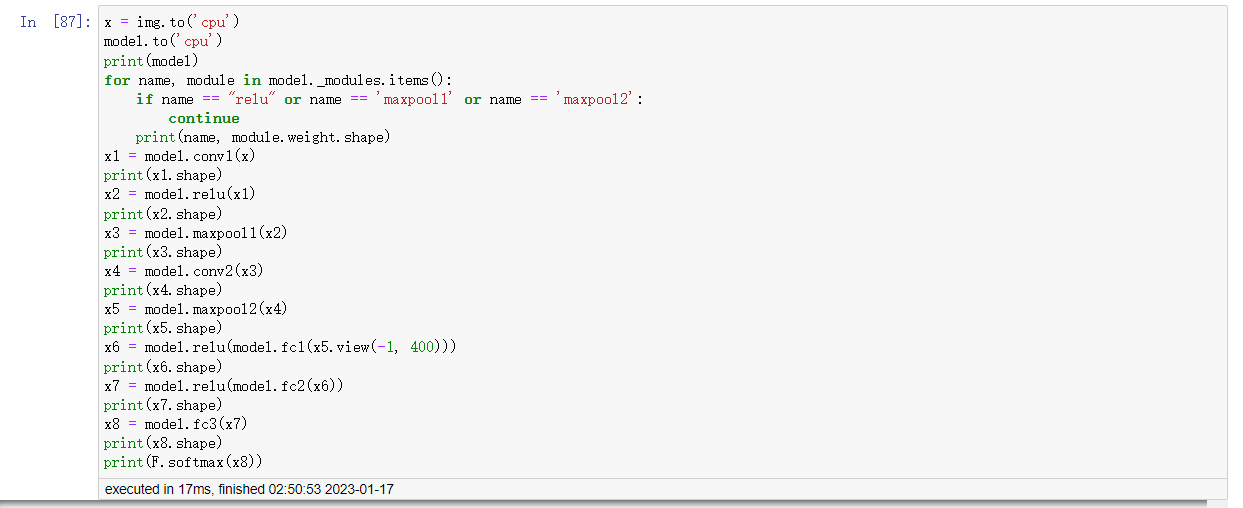
Pre:

出发点：设计一个能跑Lenet-5的架构

1. 设计原语：excel表格（讲解一下？尤其是pool和conv的is\_bias，还有stride，还有其实padding本来也是这么考虑的，这跟浮点数的表示的同一个技巧，就是加上默认会有的东西，节省空间）
2. 添加原语：本文档第五页第六点、加入了matrix\_mul, relu, conv2d, maxpool
3. 调整数据类型：考虑到Lenet-5和mnist都是float类型，写了类型转换。
4. Lenet-5结构：第四页，第五页可以做成表格（后面切分的时候讲吧）。
5. 训练Lenet-5和显示。.ipynb文件里面扣几张图出来，比如训练的loss和acc，还有我们选取的图片以及获取数据的操作：









1. 部署Lenet-5。从MLP开始部署（做成表格）：

硬件资源

|  |  |  |  |
| --- | --- | --- | --- |
|  | Depth | Address | Size of float |
| Prim\_buffer | 100 | 1000-1100 |  |
| Scratchpad | 1000 | 0-1000 | 8000 |
| Dram | 10000 | 0-10000 | 80000 |

Lenet-5

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Layer | input | Output | Ker | str | Pad | Input\_size | Output\_size | Para\_size | Divide |  | Para\_addr |
| Conv2d\_1 | 32\*32\*1 | 28\*28\*6 | 5 | 1 | 0 | 1024 | 4704 | 150 | No |  |  |
| ReLU |  |  |  |  |  |  |  |  |  |  | 1900-1919 |
| Pooling\_1 | 28\*28\*6 | 14\*14\*6 | 2 |  |  | 4704 | 1176 |  | No |  |  |
| Conv2d\_2 | 14\*14\*6 | 10\*10\*16 | 5 | 1 | 0 | 1176 | 1600 | 2400 | No |  | 2000-2300 |
| ReLU |  |  |  |  |  |  |  |  |  |  |  |
| Pooling\_2 | 10\*10\*16 | 5\*5\*16 | 2 |  |  | 1600 | 400 |  | No |  |  |
| Fc1 | 400 | 120 |  |  |  | 400 | 120 | 48000 | Yes | 400\*16\*7+  400\*8 | 2400-8400 |
| ReLU |  |  |  |  |  |  |  |  |  |  |  |
| Fc2 | 120 | 84 |  |  |  | 120 | 84 | 10080 | Yes | 120\*40+  120\*44 | 8500-9760 |
| ReLU |  |  |  |  |  |  |  |  |  |  |  |
| Fc3 | 84 | 10 |  |  |  | 84 | 10 | 840 | No |  | 9800-9905 |

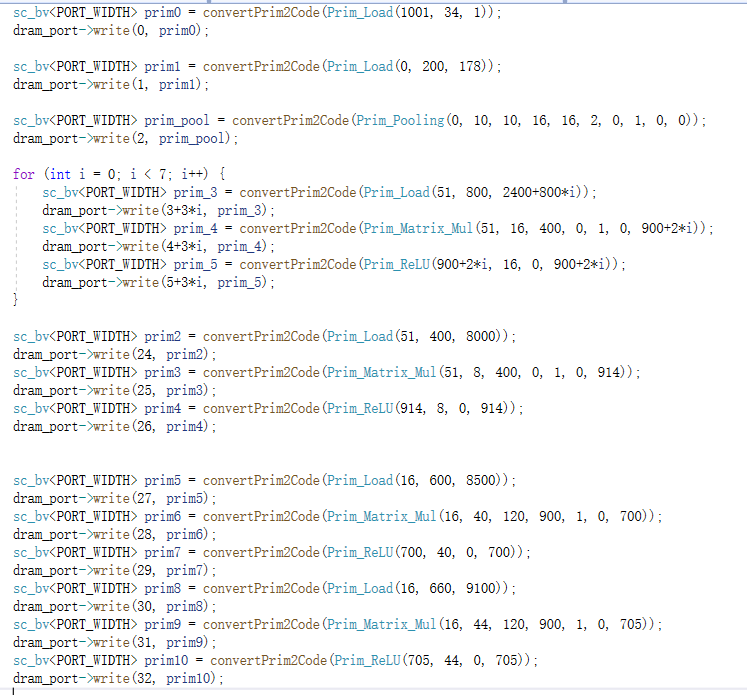
1. How to divide

事实上这边要考虑两件事，一个是硬件资源，input+output+parameter>sram就需要切分。

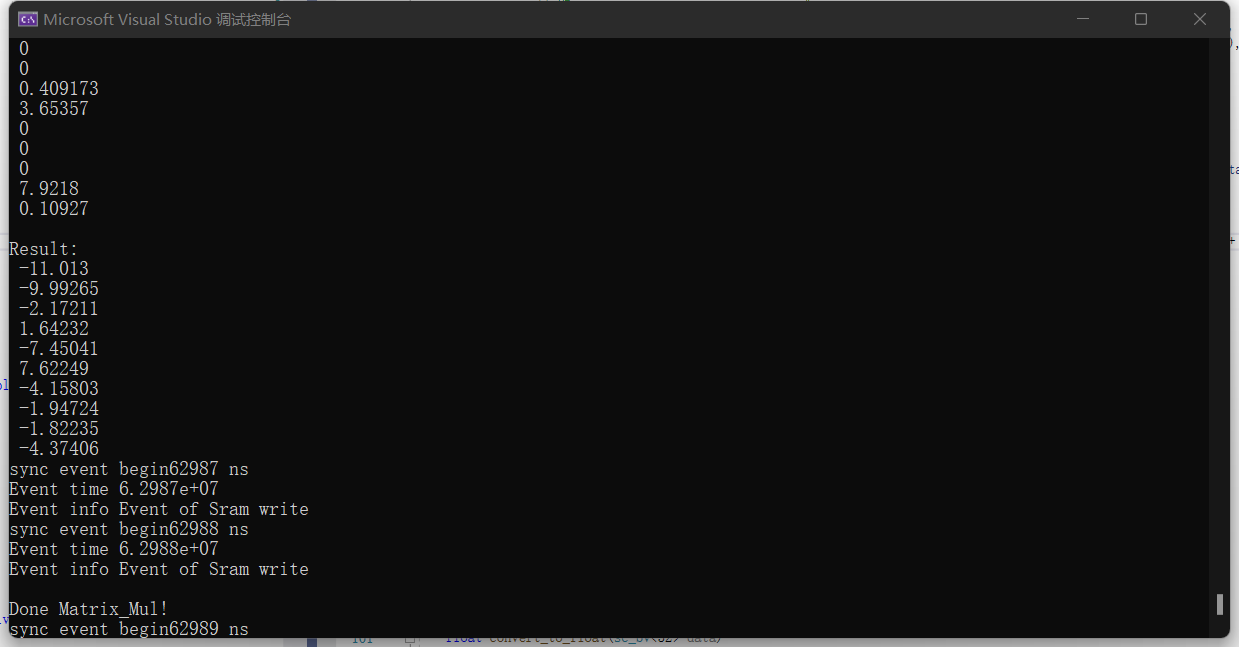
怎么切分：为方便下一步计算，考虑到每个address实际上存储8个数（由于补0机制比较繁琐），所以尽量切成前面半段结果是8的整数倍个数。如10080是切成120\*40+120\*44，这样和120\*1计算之后会得到8\*5和8\*5+4，而不是切分成120\*42\*2，否则存储不方便，也不方便下一次计算。

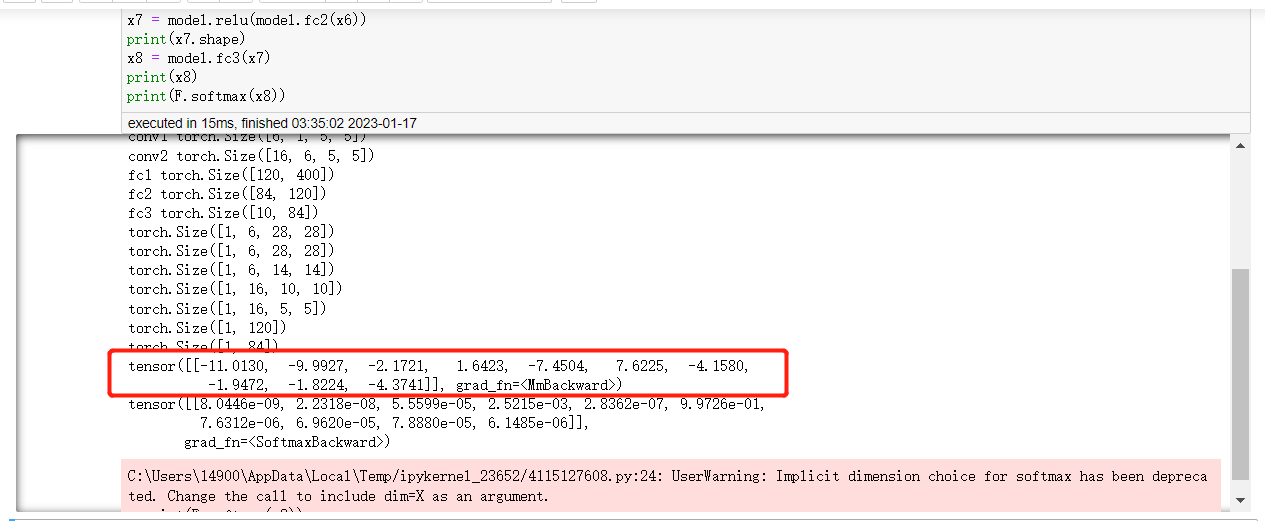
1. 实际操作：

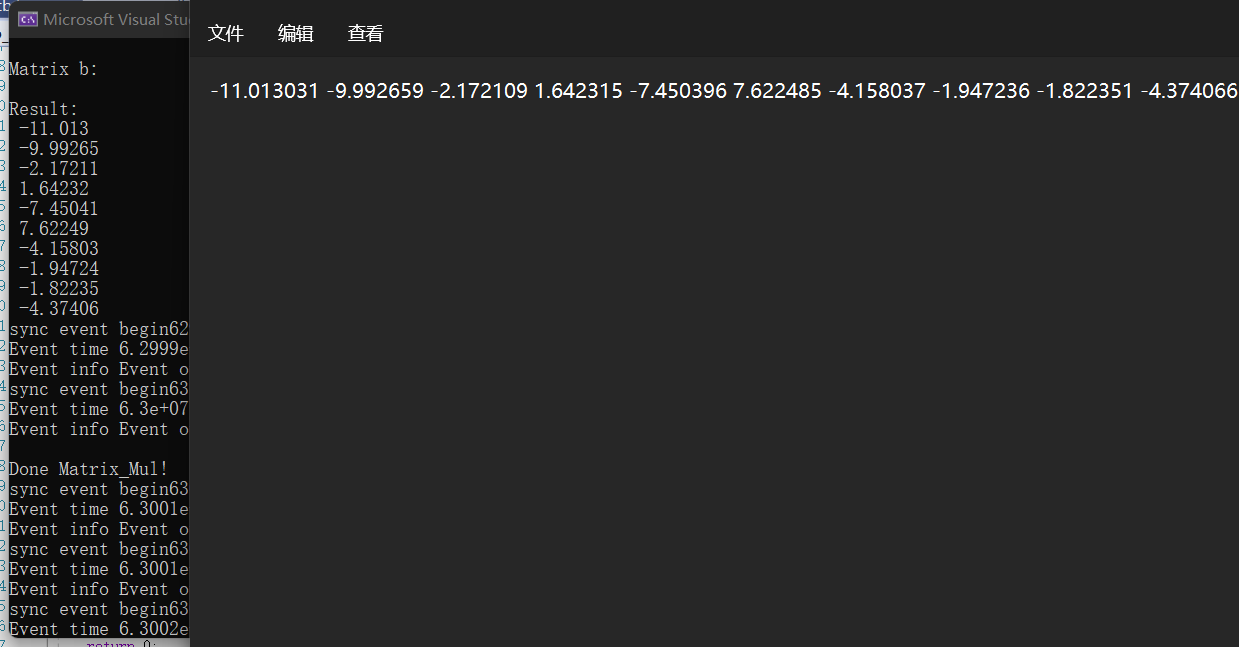
将Parameter加载到dram对应位置，如上表。然后在第一条原语中load后续原语，在第二条原语中load Data，后续原语中每层根据操作与切分分解成load\_parameter，operate，(relu)。



1. 目前实际完成从第一条MLP开始运行可以得到最终结果，截一个效果图。







1. 不调整原语的顺序和次数，DRAM最小可以（1024(data)+450+2400+48000+10080+840+10(可以直接放在data处)）/8=(62794+(10))/8 =7850+(1)目前的DRAM可以做到。SRAM最小只需能进行切分

（400\*16+400+120）/8=865，实际上由于切分的最后一次并不需要400\*16，而且前面切分都不需要补零，所以最小可以做到（400\*16+400+112）/8=864，目前的SRAM也可以做到。

1. 自动调整原语：一种简单的方法（即输入输出的大小数量级都比SRAM容量小）仅针对矩阵乘的话。确定input 和output的神经元个数，之后找到剩余空间所能容忍的最大output\*8\*N（为了防止补0对后续操作出现影响）个数，即可完成切分。

如上述例子中的400—>120，parameters=48000。SRAM大小为8000，则可以切分8000-400-120 = 7480 = 400\*8\*2+1080。事实上对于矩阵乘，若输入输出过大，可以切分[[A1 B1][C1 D1]][[A2 B2] [C2 D2]] = [[A1A2+B1C2 A1B2+B1D2] [C1A2+D1C2 C1B2+D1D2]]，因此可以不断拆分计算出A1A2等元素，再通过乘以[1 1]矩阵进行相加即可。

回答架构探索问题：

1. 单一的面积、速度或功耗都不是衡量的标准。很简单的一个例子就是暗硅效应，为了减小面积增大速度，我们可以不断增加并行数量同时使不同块芯片之间相互靠近，此时总功耗可能没有增加，但带来的散热等问题会导致芯片无法继续工作，使得我们不得不关闭某些芯片来解决该问题，因此缩小面积和增加并行所带来的效益可能并不高，所以考虑散热问题时，功耗与面积的比值不失为一个衡量标准。

同时，评价芯片的指标十分多，包含准确率、吞吐率、延迟、能效、功耗、硬件成本、灵活性、可扩展性等指标（第六讲PPT），很简单的例子就是在本实验中的芯片加入的convert\_to\_float模块一定会增加功耗而且会使速度变慢，但明显提高了芯片的应用范围。

Conv 32\*32\*3 ->28\*28\*6 channel 3->6 kernel 5\*5\*6 stride 1 padding 0

1024 4704 450

relu

Pooling 28\*28\*6 ->14\*14\*6 channel 6 kernel 2\*2 stride 2 padding 0 max\_pool

4704 1176

Conv 14\*14\*6 -> 10\*10\*16 channel 6->16 kernel 5\*5 stride 1 padding 0

1176 1600 2400

Relu

Pooling 10\*10\*16 -> 5\*5\*16 channel 16 kernel 2\*2 stride 2 padding 0 max\_pool

1600 400

Mlp 400-> 120 400\*120?????? 400\*16\*7+3200 即6400 800 3200 400

400 120 48000 1

relu

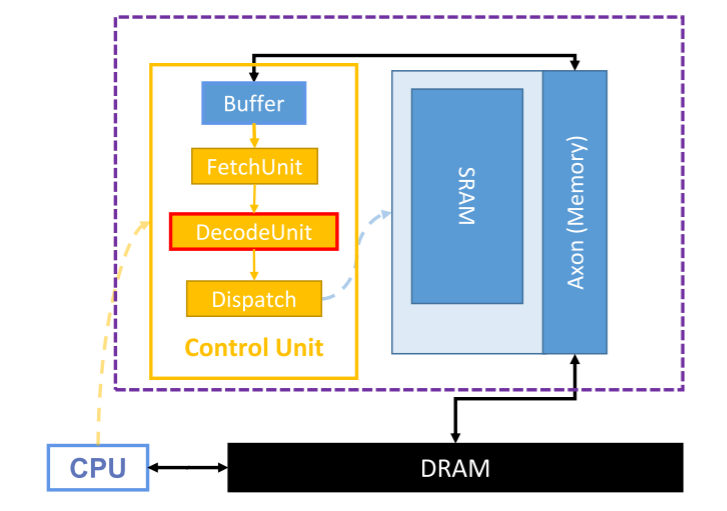
Mlp 120-> 84 120\*84 切分 40\*120 +44\*120 即 4800 600 5280 660

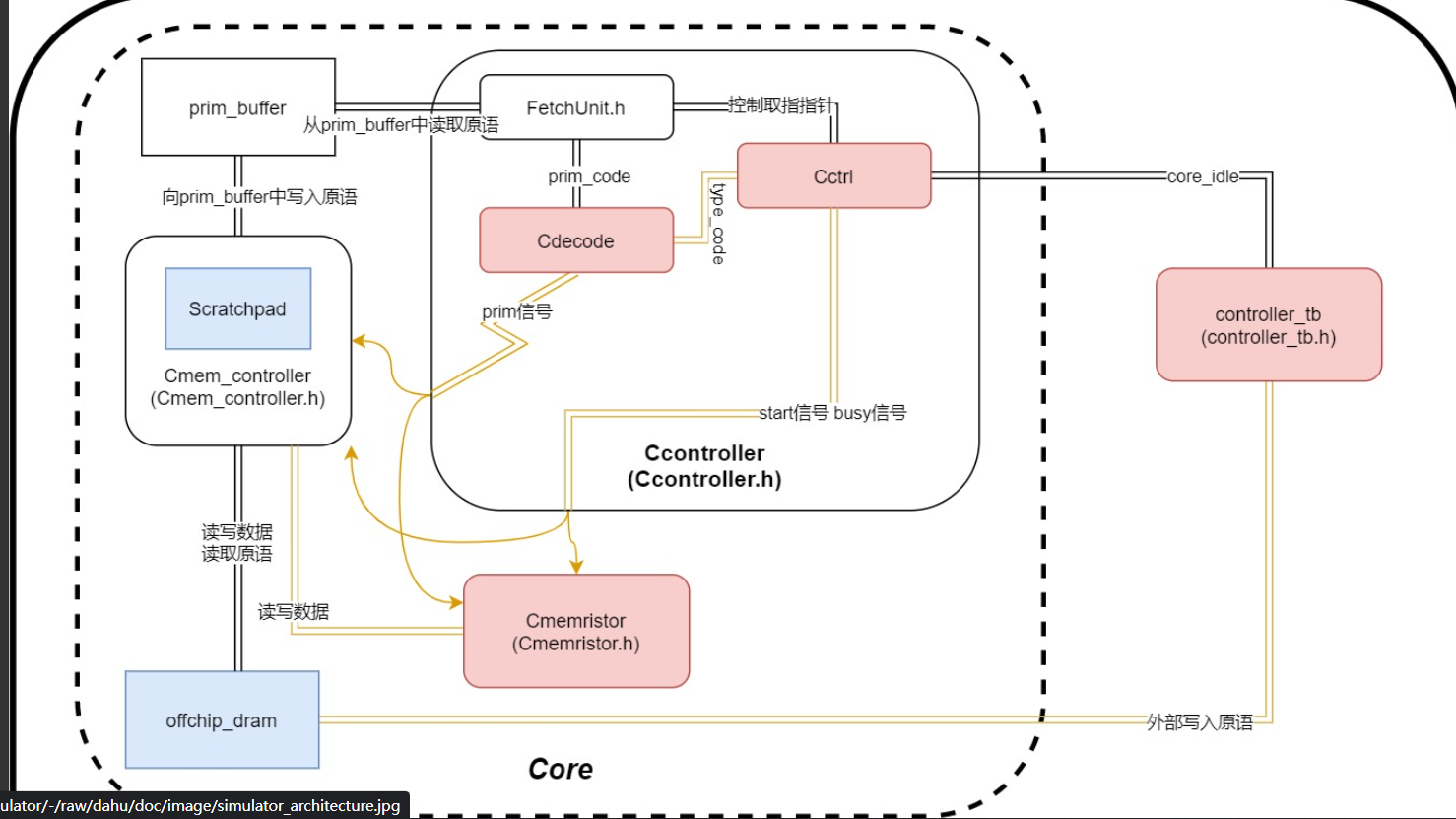
120 84 10080 1

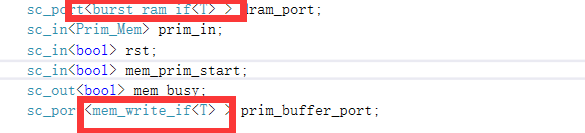
Relu

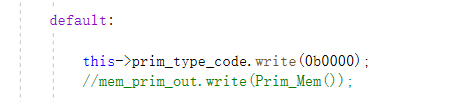
Mlp 84-> 10 840 1

84 10



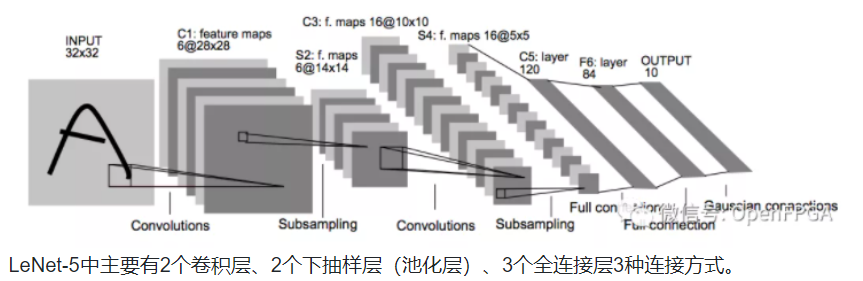


这两个不是dram和ram吗

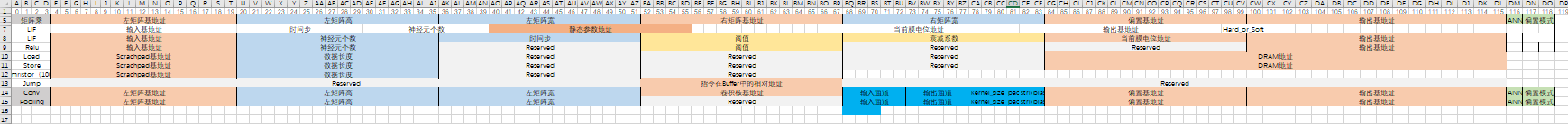
为什么注释掉了

加入MAC、Conv、pooling、relu、LIF

目标：跑Lenet-5为代表的卷积神经网络



1. 设计原语mac，conv, pooling, relu/sigmoid



设计了一下原语

1. 添加原语mac，conv，pooling，relu/sigmoid
   1. mac实现。

测试一下脉动阵列实现？问题：脉动阵列的面积需要评估吗？不然的话，感觉肯定比较快？

2.2 conv，复用数据，一层一层加进去，

1. 翻译lenet-5，把数据存储地址什么的都算出来

内容：

实现lenet5

要写的原语：矩阵乘、卷积、pooling、ReLU。进度：设计好指令集了

展示一下，讲一讲

性能上：矩阵乘基本的/忆阻器的；改sram大小，测试不同sram对卷积层的速度影响；卷积怎么复用

卷积层考虑bias？

Conv 32\*32\*3 ->28\*28\*6 channel 3->6 kernel 5\*5\*6 stride 1 padding 0

1024 4704 450

relu

Pooling 28\*28\*6 ->14\*14\*6 channel 6 kernel 2\*2 stride 2 padding 0 max\_pool

4704 1176

Conv 14\*14\*6 -> 10\*10\*16 channel 6->16 kernel 5\*5 stride 1 padding 0

1176 1600 2400

Relu

Pooling 10\*10\*16 -> 5\*5\*16 channel 16 kernel 2\*2 stride 2 padding 0 max\_pool

1600 400

Mlp 400-> 120 400\*120?????? 400\*16\*7+3200 即6400 800 3200 400

400 120 48000 1

relu

Mlp 120-> 84 120\*84 切分 40\*120 +44\*120 即 4800 600 5280 660

120 84 10080 1

Relu

Mlp 84-> 10 840 1

84 10

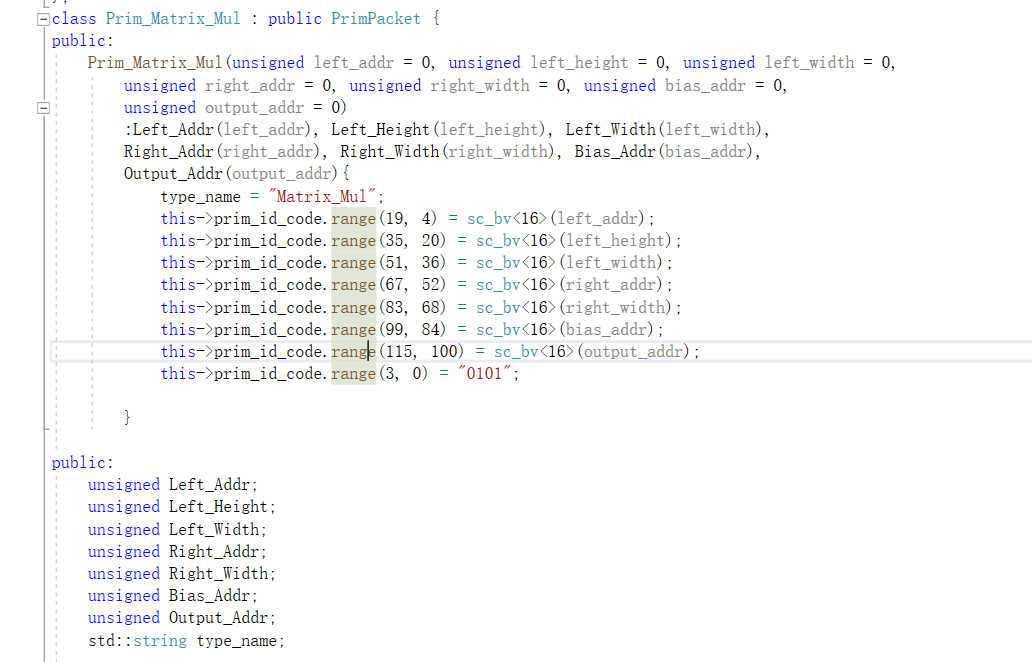
注：

1. 关于Controller\_tb：在dram里面写入三条原语，输入Idle开始。
2. Scratchpad->load：从dram\_port中load data，然后写入scratchpad中，如果m\_end\_address比scratchpad\_addr大，则写入prim\_buffer\_port中
3. Port的问题，dram中的每一位是sc\_bv<256>=32\*8，所以一个地址里面有8个32位数（int，8个字节）。Load的时候要注意这样的长度。
4. 关于启动和PC的问题，start\_prim\_rom里面有一条指令是Load(1000,3,0)，（在CController里，改3可以改指令数），开始第一件事是执行这条指令。PC+1

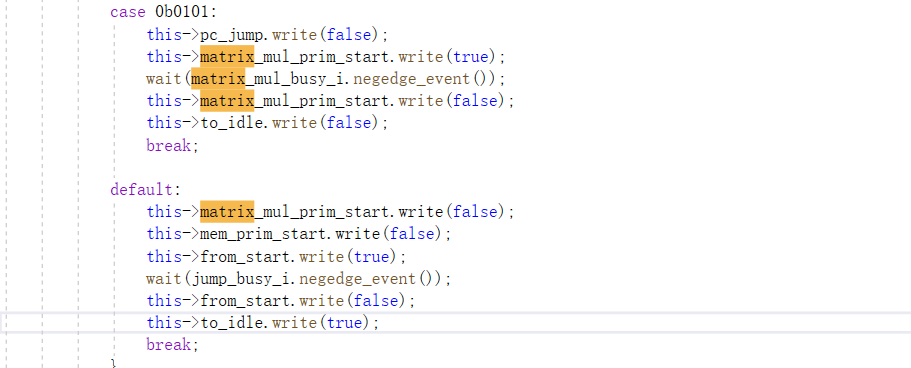
SRAM：scratchpad 0-1000， 且load 里的scratchpad\_addr>=end\_addr时，会load到Prim buffer中的scratchpad\_addr-end\_addr+i中。也就是会第一条指令是Load到prim\_buffer的0位置，之后是1，2，3。

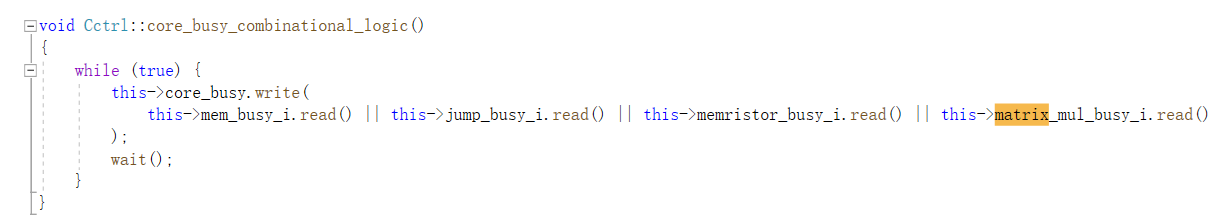
这样就要注意prim\_buffer中只有0-100也就是只能写100条指令。

1. 拆分、simulation时间不够，被stop了
2. 矩阵乘：
3. Prim.h

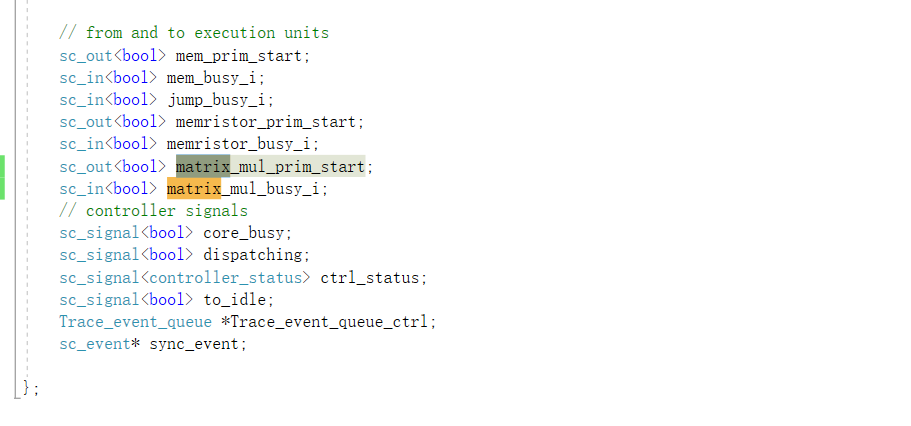


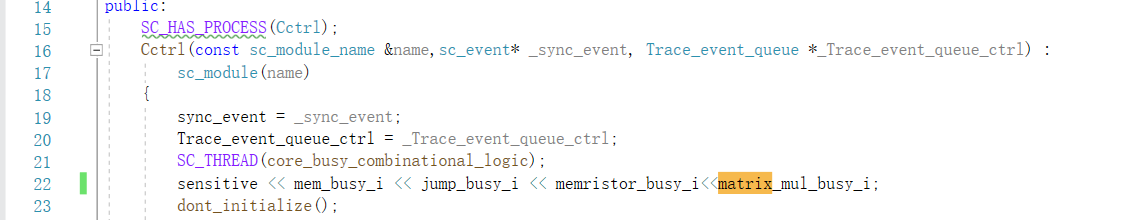
1. Cctrl.cpp,



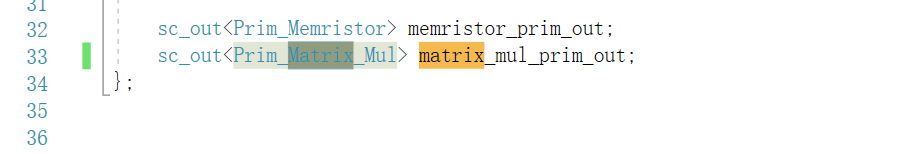


1. Cctrl.h

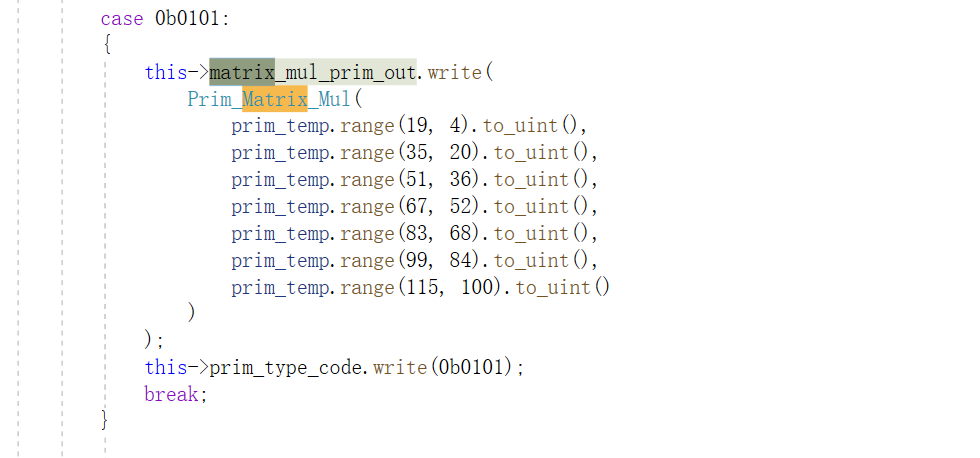




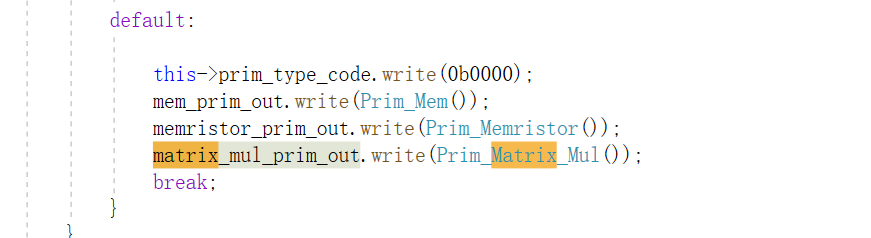
1. Cdecode.h



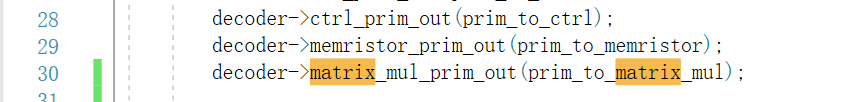
1. Cdecode.cpp: case，

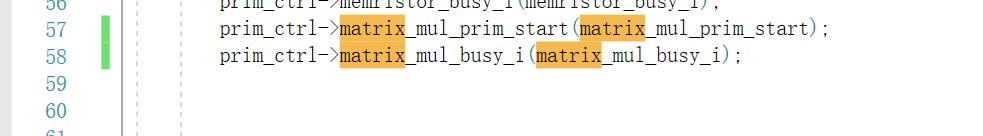


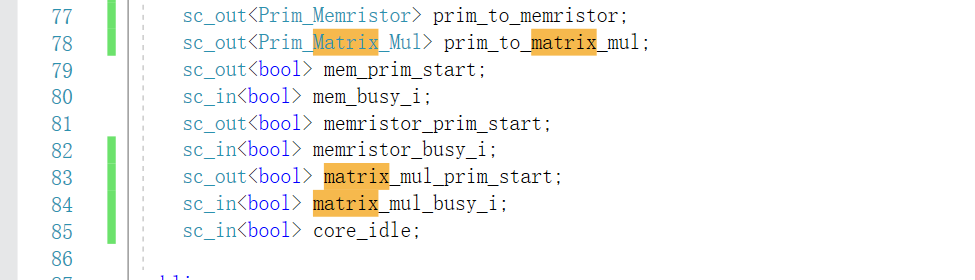
Default



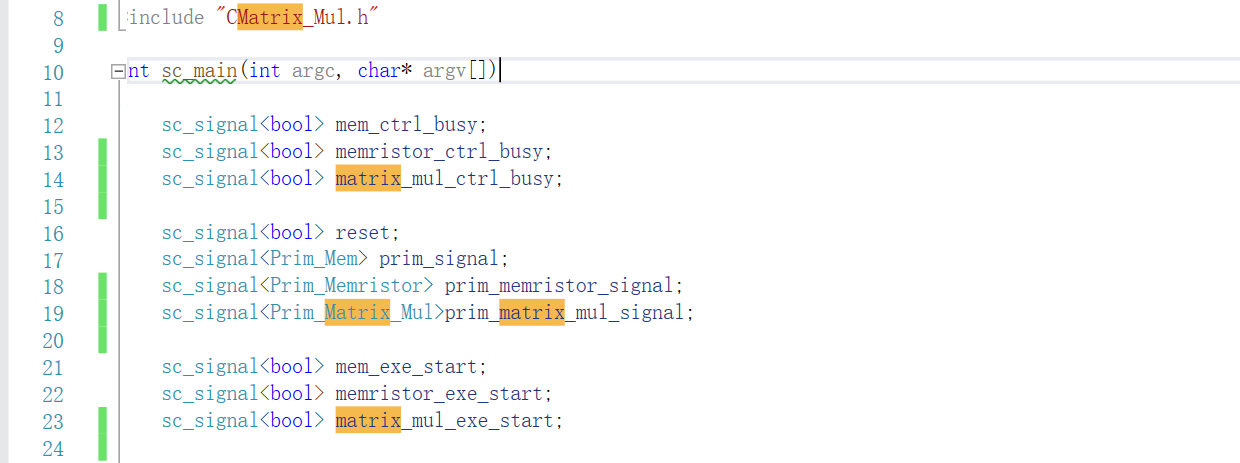
1. Ccontroller.h连线

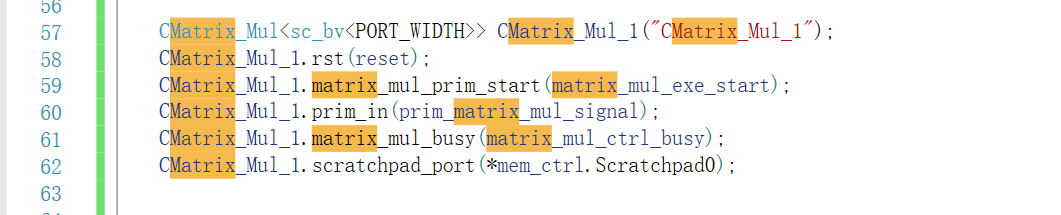


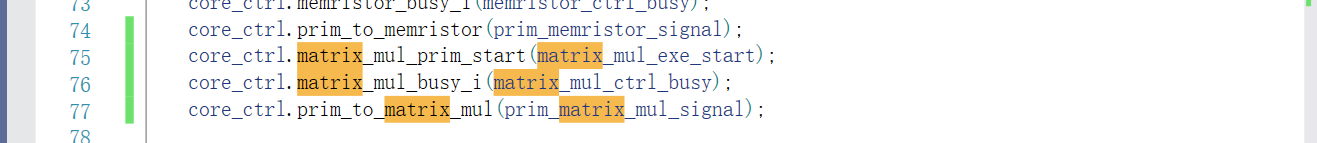


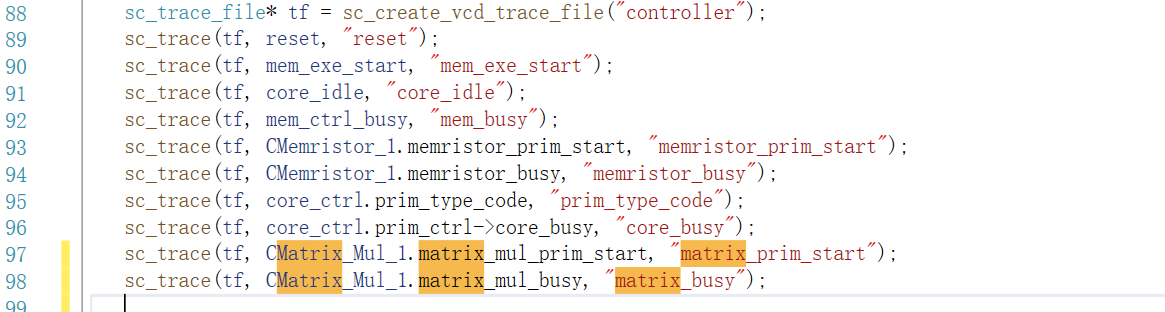


1. Core\_controller









1. CMatrix\_mul

1.15 晚

加入MLP 加入RELU 加入