

SHANGHAIJIAOTONGUNIVERSITY

研究生课程项目报告

GRADUATECOURSEPROJECTREPORT

课程：SoC设计方法

项目题目：基于蜂鸟E203的卷积加速实验设计

学生姓名: 郭梦裕

学生学号: 119039910091

任课教师: 王琴 蒋剑飞

学院(系)：电子信息与电气工程学院（微纳电子学系）

开课学期： 2020年（春季）

# 绪论

## 实验背景

蜂鸟E200系列是开源的RISC-V处理器，使用Verilog2001语法编写可综合的RTL代码，可读性较好。网上的开源库不仅提供了处理器核的实现，SoC的实现，FPGA平台和软件实例，还实现了完整的调试方案，具备完整的GDB交互调试功能，为开源处理器提供巨大创新。

本次实验采用的蜂鸟E203，其整体架构如下图所示。图中红色为本次课设内容，需要自行设计卷积加速模块，并依据现有参考资料挂载在私有设备总线上，并为卷积加速的数据提供SRAM模块，挂载在系统存储总线上，再留出接口确保两者交互。



图 1 E203整体架构图

## 实验要求

1. 利用现有工具实现一层卷积运算。
2. 自行编写testbench，保证代码正确性，利用vcs进行仿真。
3. 利用SynopsysDC软件及提供的综合脚本命令tcl综合verilog代码，并进行时序分析。若有建立时间或保持时间违规，指出更改方法并修改以提升性能。
4. 利用ICC软件及提供的脚本命令进行物理设计。
5. 利用C代码实现仿真SoC的CPU计算卷积流程。

# 卷积模块前端设计

## 卷积模块C代码设计

首先根据实验要求设定，设定输入输出的数据规模，如下所示。为了跟硬件实现匹配，将输入数据数组x和卷积核数组y设置为8bit有符号类型，输出数据z为16bit有符号类型，以保存较精确的结果，再通过量化方案使得实际输出为8bit有符号数。

Signed char x[N][C][H][W];//需要运算矩阵

Signed char y[I][O][KH][KW];//卷积核

Signed short z[N][O][H][W];

卷积实现上，因为输入图像通道数C和卷积核的输入通道I本质上相同，因此只需要对余下的七个变量进行循环实现，其公式如下

σ是激活函数，在这里我们选用了ReLU函数进行实现，在进行简单的循环初始化和循环计算后就可以得到最终的程序，详细请参阅附件代码。

## 卷积模块Verilog设计与实现

### 卷积模块设计实现

卷积的核心为乘法累加计算，为了减少数据的读取，增大数据复用的可能，我们参考了脉动阵列的方式实现。其流程如下图所示，将3\*3的卷积核转变成9个1\*1的乘法模块，每一个周期读取的一行数据会首先被第一行的三个乘法模块计算，下一个周期该数据会被送到第二行的三个乘法模块，而第一行的三个乘法模块读取新的一行数据，由此通过寄存器移位存储的方式为每个乘法模块传输数据。而从第三个周期开始，可以将9个1\*1的乘法模块的结果累加，即为输出数据，直到图像在该方向上卷积完成



图 2 乘累加实现

由上设计，每个周期需要3\*8bit数据，然后E203的总线设计上，每个周期传输数据为32bit，存在浪费。为了更充分的利用总线传输的数据，我们为卷积模块设置了18个乘法模块，分别计算两个3\*3卷积核的结果，其中每次传输的4\*8bit数据，前三个作为一个卷积核的输入，后三个作为另一个卷积核的输入，即第2，3数据复用，每次输出两个结果保存在卷积模块的buffer中。



图 3 适配32bit输入

为了适配卷积模块的实现，需要设定特有的数据结构进行存储。由下图可知，原输入图像按列存储在SRAM中，每次按4\*8bit向右读取数据，进行卷积窗口的滑动，直到读取最后一行的32bit数据完成该方向上的卷积。卷积模块会将该方向上2\*32的数据（两行数据每行32个）从buffer中输出，保存完成后开启下一行的计算。此时根据卷积的定义需要读取输入图像的第3行上的数据，其存储位置横跨了两个SRAM的cell，使得数据读取变得复杂。因此我们将数据复制存储，每一行上的数据按图像上的数据，分别存储1，2，3，4，3，4，5，6…像素点的数据，确保每次读取的数据都在同一个SRAM的cell上，减少读取时间。



图 4 内存存储方案

由此完成了卷积计算过程，在此之上添加状态机控制逻辑确保模块正常运行，其状态机如下图所示。其中分别有四个状态，IDLE表示空闲，RWGT表示读权重，RINP表示读输入，WOUT表示写输出。模块中的信号定义如下表所示，其中cmd相关信号用来控制与外界的交互和地址计算，rsp相关信号进行状态机的控制。通过模块中的信号进行状态切换，实现每完成两行结果的计算输出数据保存，每保存一个输出通道的全部数据则切换新的权重重新计算，保存完所有通道数据则切回IDLE并发出信号表示卷积模块计算完成的等功能，详细请参阅附件代码。



图 5 状态机图

表 1 卷积模块信号定义

|  |  |
| --- | --- |
| 信号 | 含义 |
| rwgt\_cmd\_done | 读权重指令发送完成 |
| rwgt\_rsp\_done | 读权重数据接收完成 |
| rinp\_rsp\_done | 读输入指令发送完成 |
| rinp\_cmd\_done | 读输入数据接收完成 |
| wout\_cmd\_all\_done | 写输出指令发送完成(所有通道) |
| wout\_rsp\_all\_done | 写输出数据接收完成(所有通道) |
| wout\_cmd\_row\_done | 写输出指令发送完成(两行) |
| wout\_rsp\_row\_done | 写输出数据接收完成(两行) |
| wout\_cmd\_chn\_done | 写输出指令发送完成(1个通道) |
| wout\_rsp\_chn\_done | 写输出数据接收完成(1个通道) |

### 卷积模块互连与其他

完成了卷积模块的设计后，在顶层添加与总线的连接和ICB互连实现。ICB仅有两个通道，其中cmd通道用来发送读写请求的地址等信息，rsp通道用来返回结果信息。交互通过握手信号实现，ICB在cmd阶段用一对valid/ready信号握手，在rsp阶段用另一对valid/ready信号握手，确保交互的顺利。

基于此，设计模块提供了两部分ICB，一部分以conv\_icb为前缀，通过ICB通道访问SRAM数据，一部分以conv\_ctrl\_icb为前缀，通过ICB通道实现跟CPU的访问。

卷积模块被挂在私有设备总线上，在e203\_subsys\_perips.v中进行修改，原先的设备总线上的ICB2AXI，被替换成了我们的卷积模块，将地址映射修改后完成连线，并将与SRAM访问的连线引出，在e203\_subsys\_mems.v中与原来的mem线合并，通过仲裁器仲裁。仲裁器的设计参考了原有的代码，官方给出的dtcm模块有关于ext2dtcm/lsu2dtcm的仲裁可以作为参考。SRAM模块及其控制使用了原有的SRAM控制和补充材料给出的SRAM模型，再进行例化连线即可。修改完所有文件的连线后，模块即可正常运行。

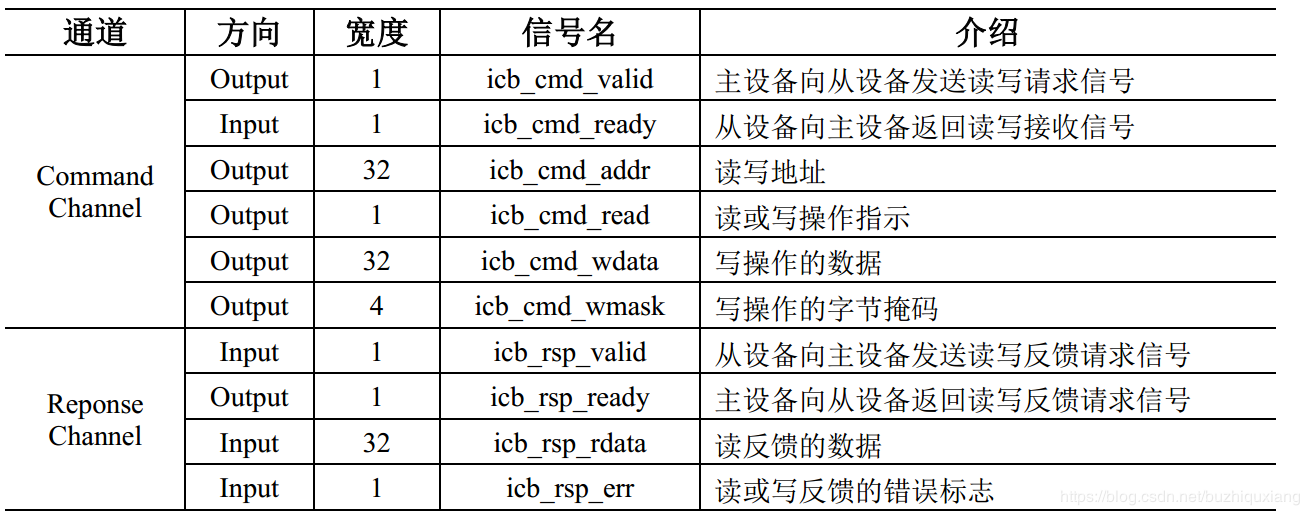


图 6 ICB总线协议

## 卷积模块对比分析

卷积模块从CPU对conv\_ctrl发送命令开始，置位start开始进行计算，其波形图总览如下所示，可以看到模块中的各个信号有序的更替，最终完成整个卷积模块的计算，done信号被拉高。通过标记开始和结束的时间，可以得到卷积模块执行该卷积消耗的时间约为94728ns。

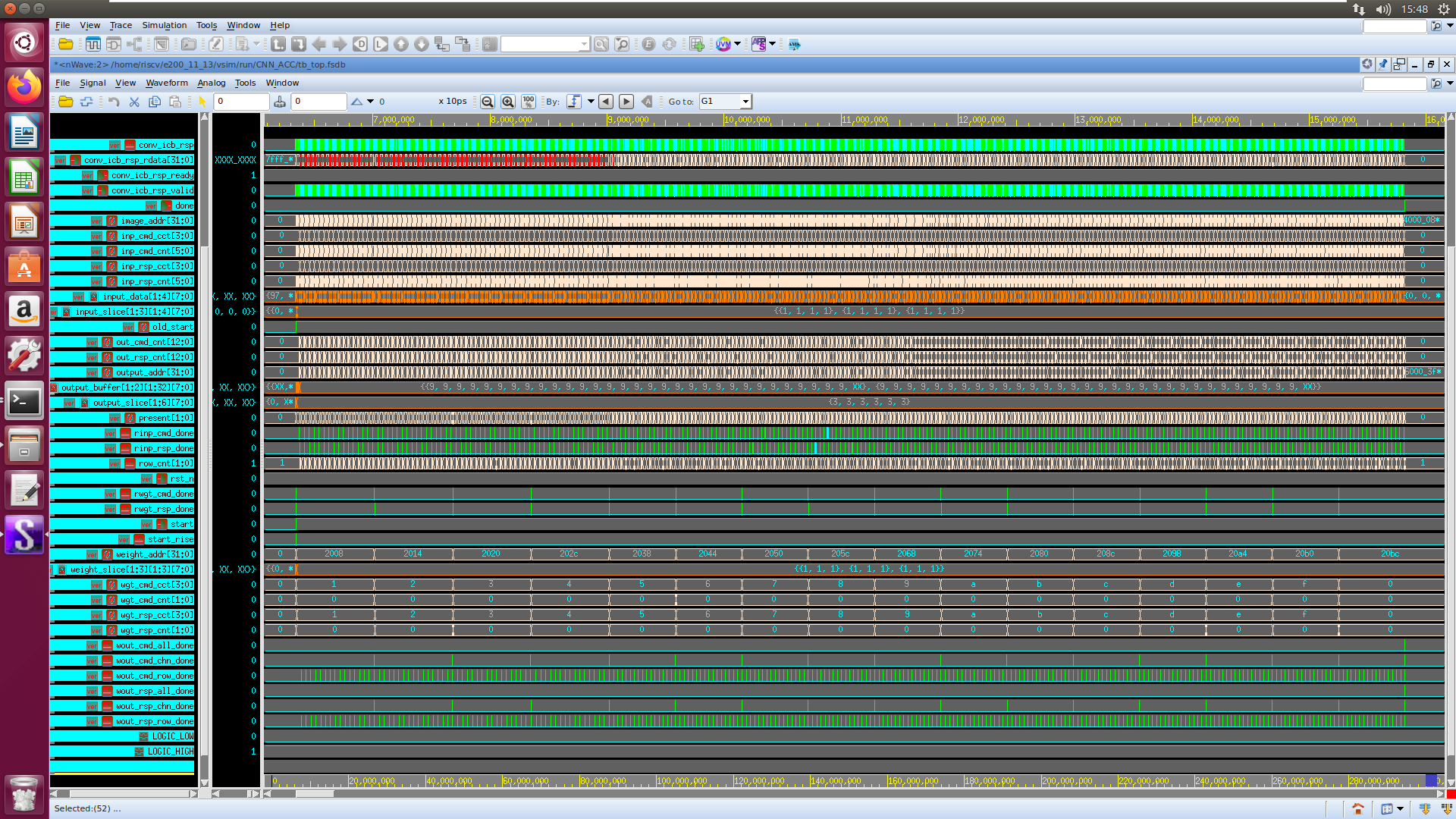


图 7 卷积模块波形总览

但仍存在不足，从波形中可以看出响应信号和原指令信号存在延迟，此时卷积模块无法进行下一波操作而停滞。另一方面，虽然指令连续发送，但返回的响应信号并没有连续返回，其中间存在停顿使得计算的流水线停滞。因此设计仍存在较大的提升空间，但时间有限并没有在此继续深入。

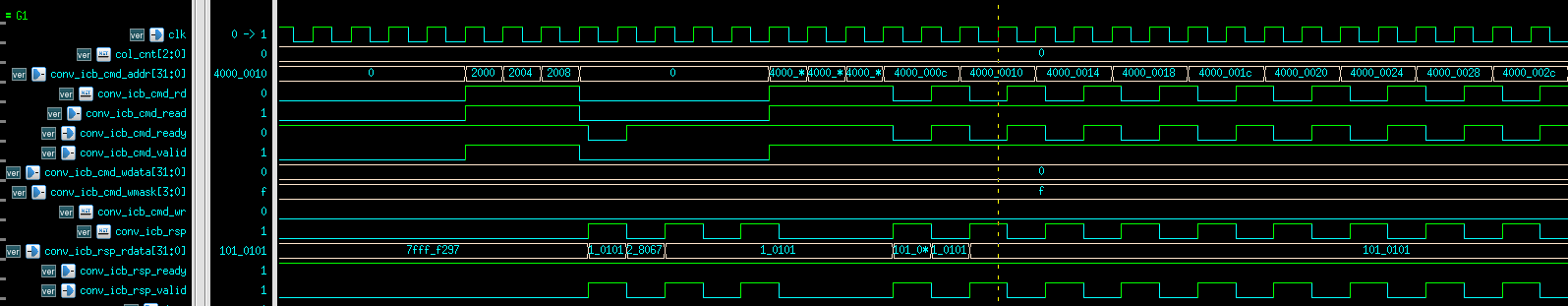


图 8 存在的不足

SoC上的c库文件不足，缺乏常用的time.h等文件，因此不能直接测量运行时间。因此C实现的卷积模块时间评估通过在计算开始前和计算结束后添加寄存器访问，然后在波形中找到相对应的波形进行计算测量，利用mark进行标定。另一方面，不能直接在代码中初始化输入等数据，其会直接生成在内部的dtcm中，极大的减少运算速度。为了控制变量，输入数据和权重数据均来自SRAM，并通过不同方式将结果写回SRAM。其与硬件实现的卷积模块比较结果如下所示。可以看出，相比于CPU实现卷积，硬件实现卷积有了很大的加速。而不同的写回方式也产生了一定的影响，频繁多次短暂写入比一次性连续写入要慢一些，CPU执行写入需要等待rsp，多次写入也影响了计算流程，使得出现如下结果。详细细节查阅代码流程设计。

表 2 运行时间对比

|  |  |  |
| --- | --- | --- |
| 类别 | 写回方式 | 执行时间 |
| 硬件实现CNN | - | 94728ns |
| 软件实现CNN | 每计算4\*8bit写回SRAM | 18368144ns |
| 全部计算完写回SRAM | 18483132ns |
| 软件实现CNN(无SRAM) | - | 约25000ns |

# 卷积模块逻辑综合

## 逻辑综合设置

逻辑综合的代码参考补充资料修改，主要修改了文件路径相关信息。对于卷积模块的单独综合，需要调整clk，因为其不存在两个时钟，而对于全芯片的物理综合，其内部存在hfextclk和lfextclk两个时钟，需要确保前后名称一致不出现差错。

另外，原先的SRAM不能被综合，因此需要替换为补充材料中SRAM库给出的器件，分别为8K\*32bit和4K\*64bit，另外itcm和dtcm也被例化为该SRAM。

此外，受器件库限制，习惯于软件思路设计的硬件模块并不可靠，特别是一些并行化思想和尽量简单的计算操作限制了硬件设计。其中在设计时曾大量运用的取余操作，在逻辑综合中被禁止使用导致综合失败，无奈重写卷积模块剔除所有复杂的计算才使得结果得以顺利通过，其具体结果如下。

## 逻辑综合结果

对于卷积模块，时钟频率设置了100M，其逻辑综合后的面积结果如下所示。可以看出模块大小约27000，其中主要部分是组合逻辑，其功能是实现卷积和状态机相关控制。其余则由于开设了一定数量的reg和互连产生的面积开销。相对应地，DC工具还报告了两条时序信息，一条是clk相关，另一条是reg2out，其slack均为正，均满足时序，详细报告见附件。

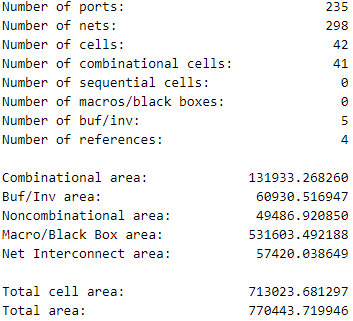
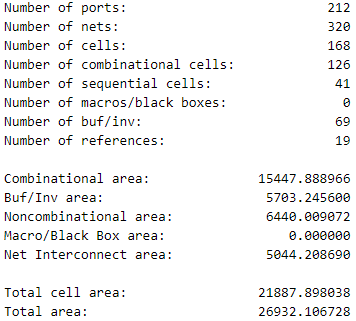


图 9面积报告:(左)卷积模块，(右)完整芯片

对于完整芯片，参考原有的数据，设置了50M的时钟频率。其面积结果如图所示。可以看出全部芯片大小约为770000，其中占据主要部分的Macro/Black Box area是我们使用的SRAM模块，占了约70%。事实上，我们在为模块设计时，为权重数据，输入数据，输出数据分别开设了一块SRAM，其SRAM为工艺文件给定，无法修改大小减少浪费，因此芯片使用了大量的冗余SRAM。为了优化其面积，最好将权重和输入以及输出公用一块SRAM，并且用地址划分三个区域以减小SRAM面积。但这样的划分可能缺少通用性，对于某些专用电路设计，可能用多个SRAM各司其职会更好，此处存在争议因此没有进行修改。

相应地，工具给出了时序报告，其中有关于两个高低时钟的，也有in2out，in2reg，reg2out的。DC工具对时序进行优化，其中最难满足的为高时钟，其最终优化结果为高时钟的slack为0，其余为正。特别地，存在通路的slack高达30000多，存在进一步优化的可能。具体的时序和面积报告请查看附件报告。

# 卷积模块物理设计

## 物理设计设置

物理设计脚本参考给出的补充资料修改，其中修改了大量的路径错误和名称引用错误。特别地，在1\_data\_setup.tcl中将ground\_net相关线路更改为VSS，并增添了VDDPE/VDDCE进行设定，并在后续相关代码中做了相应的改进。2\_floorplan.tcl中更改了芯片的大小为1150\*1150，其参考逻辑综合面积和芯片利用率0.7得到，需要注意的是过小的芯片会使得其在后续的布局布线中产生较大的violation(在1050\*1050有将近300000个violation)，而过大的芯片会使得芯片资源产生较大浪费。同时，注释掉了Filler相关内容，因为库文件中不存在对应的器件。另外，为芯片设置虚拟pad，共四对分设在四边，其余指令按照实验指导修改。后续文件中有因为版本更新导致的指令失效，需要替换为新的指令，详细请参考脚本代码。事实上失效指令均为优化相关，在满足所有容限的情况下，指令失效不会导致后续结果的生成。在修改完全部脚本代码后，即可进行后续的物理设计。

## 物理设计结果

卷积模块相比于完整芯片十分简单，又缺乏一定的约束限制和芯片相关设置，其结果并不具备很高的参考性，其布局布线图如下所示，不做具体讲解。

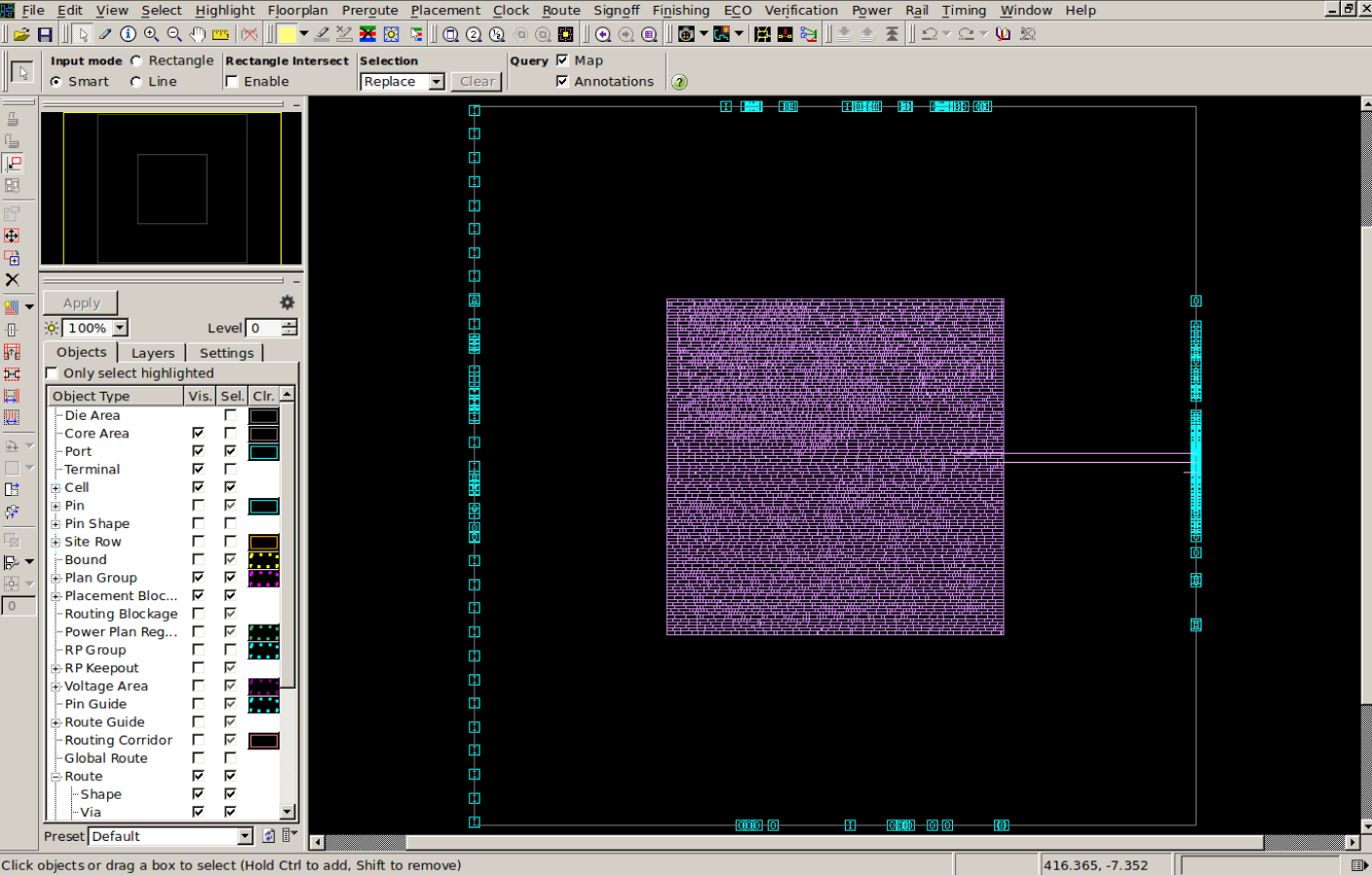


图 10 卷积模块物理设计结果

完整芯片的物理设计结果如下所示。对于物理设计，原先设定0.7的core\_utilization会使得其整个面积被固定在了1050\*1050左右（770000/0.7=1100000），但在布局和时钟树设置完成后的使用面积已经达到了1050000，没有足够的空间用来布线，最终使得其在布线阶段有大量的violation，虽然其内部优化使得violation从5万左右降到了5000，但最终优化失败没有过DRC检测。基于此，我们将芯片设置成了1200\*1200，core\_utilization设置为了0.5，虽然出现了一定的资源浪费，但最终通过了后续的设计。

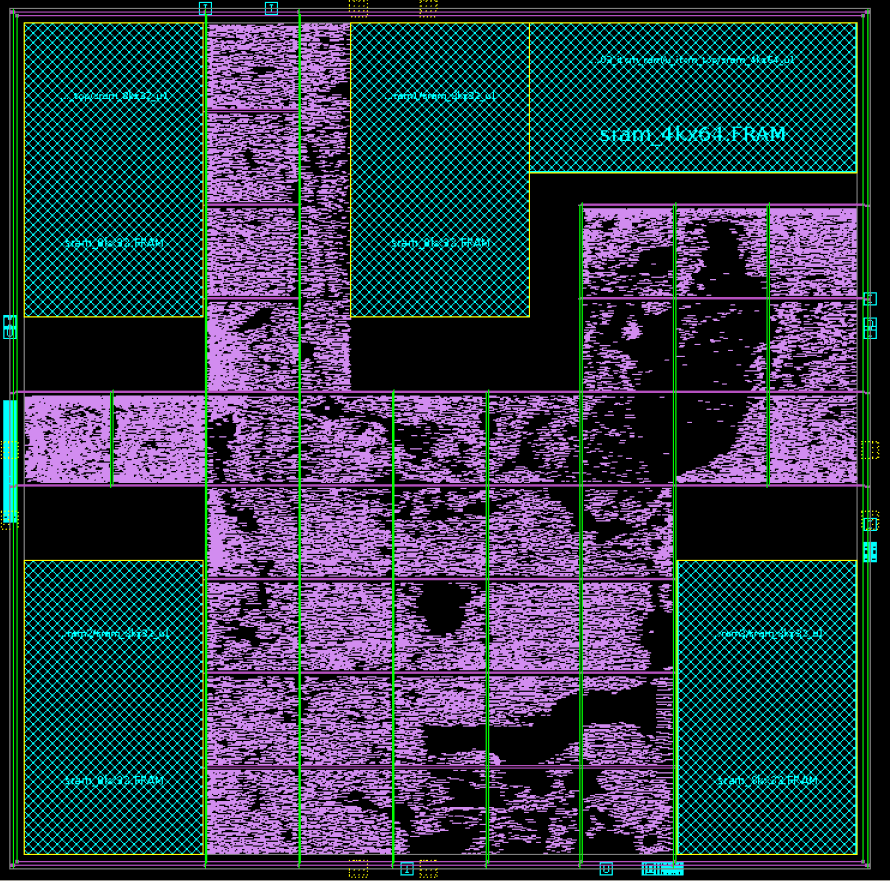


图 11 完整芯片物理设计版图

在最终检查结果报告的时候，发现LVS和PNS存在错误，如下所示。由于设计原因，存在一些浮空port没有被连接，这是正常的。但出去浮空port外，还存在Open Net，分别为VSS和VDD，在PNS也有类似的报错信息。打开UI界面发现存在位置VSS和VDD没有连上，被打上了黄色的叉。询问后怀疑是边上能连线但via大不了，在优化时因为floating被remove掉了，或者是floorplan出了问题导致其没有达到预期的结果，听从他人建议在原有的pre\_pns基础上重新布局，通过“preroute\_standard\_cells -fill\_empty\_rows -remove\_floating\_pieces”重新布局，输出pn\_net.rpt直到发现该项报错消失，再进行后续的物理设计。

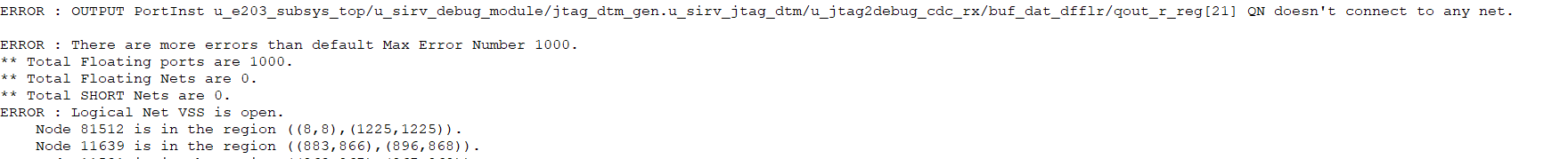


图 12 LVS报错（部分）

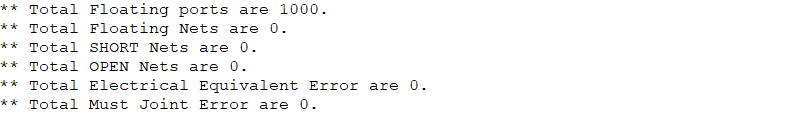


图 13 修正后LVS报告

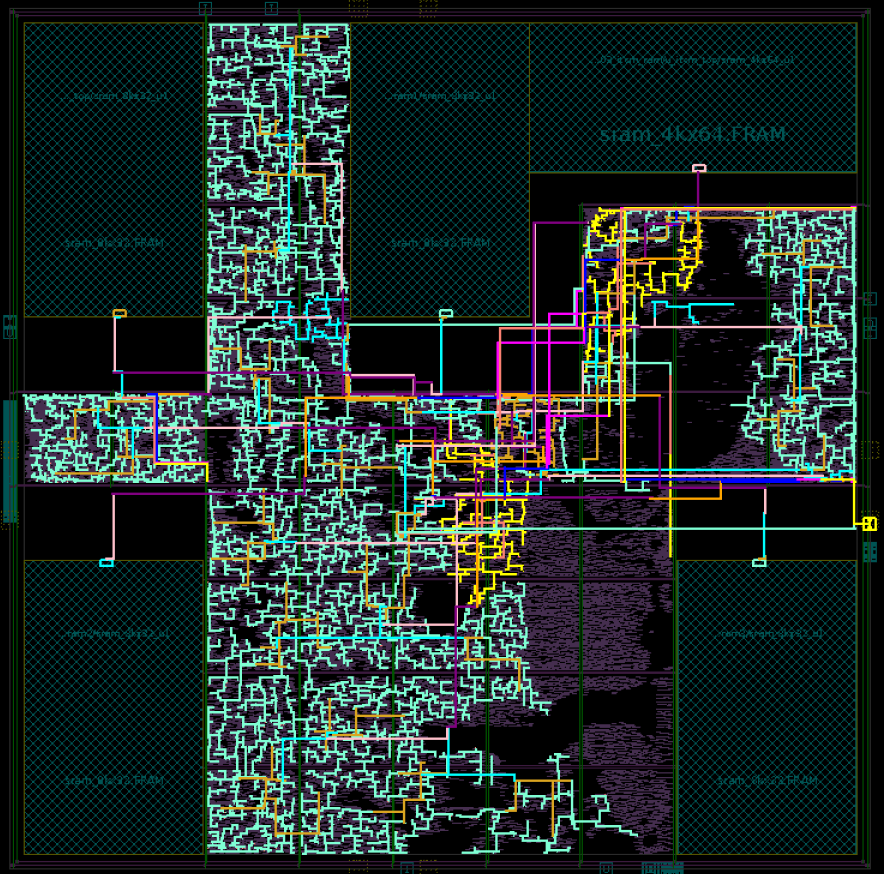


图 14 物理设计时钟树结果

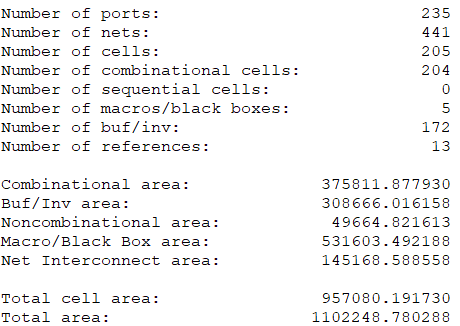


图 15 物理设计面积结果

在物理设计最后的报告中，drc和timing报告没有error，但存在一定违例。其违例在violation.rpt报告中有所体现。受时间和知识水平限制，没能进一步优化。

# 总结与展望

## 实验总结

通过以上实验，初步了解了SoC芯片从前端设计，逻辑综合到后端设计实现的全部流程。在神经网络的大背景下，学习了解了硬件实现卷积模块设计，评估脉动阵列等不同计算流和数据流的性能差异，深刻体会到了内存墙所带来的性能限制。另一方面，通过DC和ICC工具的综合和后端设计，了解到评估芯片的timing和area等信息指标，了解了一些优化和修改时序的方法，对芯片设计有一定的了解。

在此感谢老师们课上的详细教导，课后助教和大佬同学对课程设计的指导和讲解，让我得以顺利完成该课程设计。

## 未来展望

正如文中提到的，原先习惯于软件代码设计的思路在硬件设计上并不可靠，设计仍然存在很多的优化空间，在联系其他硬件上更有一些trick进行协同优化，例如SRAM的设置和内存布局，希望在未来能够写出更好更优秀的代码。

在逻辑综合和物理设计上，虽然最终也得到了可以查阅的版图设计，但其仍存在一定的error和warning需要修改，也存在一定的违例需要改正，离真正的芯片设计还有很长的路要走，希望能再接再厉，继续努力。