电子科技大学示范性微电子学院

设计与仿真报告

(2024.11 -2025.01)

课程名称 IC 综合实验 2

实验名称 IC 综合实验 2

指导老师 王忆文

学生姓名 李卓霖，况明远，邓锦琪，徐子涵

学生学号 2022340101012、2022340102004、2022020915006、2022340104032

组　　号 11

电 子 科 技 大 学

实 验 报 告

实验地点：清水河校区国际创新中心 B523

实验时间：2024.11-2025.01

报告目录

实验报告成绩： _____

1. 课程设计需求

本次 IC2 实验课程（数字方向）的设计内容为一个已知结构的 AI 加速器的 ASIC 全流程设计并通过评估和流片。已知结构的 AI 加速器具有 8bit 输入和 8bit 输出，可以工作在至少 50Mhz 的时钟下，能够完成给定卷积神经网络的运算。总体的面积要求为在不带 I/O 的情况下通过 DC 综合得到的网表面积为 $24 \times 10^4 \mathrm{um}^2$，版图的面积最大限制为 900um * 900um，可以使用的最大的引脚数量为 28 个（含 4 到 6 个电源引脚）。

2. 设计规划与架构

2.1. 设计基本要求：

给定的卷积核的结构包含卷积层、池化层、全连接层各一层，共计 3 层。各层的详细介绍如下。

卷积层：在 11×11 的矩阵数据外添加 Padding 后，对对应的矩阵进行卷积操作，其中，卷积核大小为 3x3，步长为 2，共有 3 个卷积核；

池化层：对输出的数据进行最大池化，其中池化核大小为 2x2，步长为 2；

全连接层：全连接层对应的权重矩阵大小为 3x3x3，与池化输出一一对应，需要完成一次乘加操作。

芯片的输入数据为 1x1 的矩阵，是位宽为 8bit 的有符号数补码，按照卷积核、全连接权重、100 组 11×11 的矩阵的数据依次输入。

芯片的输出数据为 1x1 的矩阵，位宽为 8bit，输出的为全连接的结果通过量化所得到的内容。

由于在数据处理过程中得到的乘积数据为 16bit，加和后需要用 20bit（全连接 21bit）的数据记录才能保证数据不发生溢出，而后续进行池化、全连接和输出所需要的数据均为 8bit，因此需要一种量化规则来进行 20bit 或 21bit 的到 8bit 的量化处理。给定的卷积核的量化规则为：在保证符号正确的情况下拓展到 20 位来表示，然后将低 8 位抹除，剩余的 12 位若超过 int8 的表示范围，则取 int8 范围的最大值，反之，取这 12 位的低 8 位。由于数据均由补码表示，因此所得的结果只需要将最高位符号位向前扩展即可得到扩展数据。

2.2. 卷积计算的周期与流水规划设计

我们将每张图片输入的单个数据的周期按照所在矩阵位置排列，就得到了如

图 2-1 和图 2-2 所示的排布，其中图中标红的周期为按照周期数排序可以取到对应周期数据的最早周期（例如 14 周期为取到最左上角的卷积运算区域的最早可以取得的区域）。由于每张图的输入需要占用 121 个周期，而如果将乘加器复用，则可以在 3 个卷积核全并行（采用 27 个乘加器）的情况下，只需要 37 个周期即可执行完成，即使乘加器计算同一个卷积核总共耗费 3 个周期，总共所需要的计算周期仅为 111 个周期，相对于 121 个周期还有冗余，因此可以提出采用 2 个或者 3 个周期才完成乘加计算来优化设计，有两种优化方向可以基于这个提出。

一种优化方向采用 2 周期或 3 周期在并行的情况下进行计算，可以在外界数据高速输入的情况下在内部以 2 分频和 3 分频的情况下进行计算，这样在输入数据带宽受限但是速度较高而乘加器无法匹配的时候使得乘加器能够工作，缺点是纯并行计算的 27 个乘法器需要消耗较大的面积，而外部的 IO 被限制到了 50MHz，设计时乘法器的瓶颈不是主要矛盾。

0	1	2	3	4	5	6	7	8	9	10	11
1	1	2	3	4	5	6	7	8	9	10	11
2	12	13	14	15	16	17	18	19	20	21	22
3	23	24	25	26	27	28	29	*30*	31	32	33
4	34	35	36	37	38	39	40	41	42	43	44
5	45	46	47	48	49	50	51	52	53	54	55
6	56	57	53	54	55	56	57	58	59	60	61
7	67	68	69	70	71	72	73	74	75	76	77
8	78	79	80	81	82	83	84	85	86	87	88
9	89	90	91	92	93	94	95	96	97	98	99
10	100	101	102	103	104	105	106	107	108	109	110
11	111	112	113	114	115	116	117	118	119	120	121

图 2-1 双周期卷积周期功能与流水设计

另一种优化方向时由于可以在 3 周期完成一次卷积，一次可以考虑拆分卷积核，在 3 个周期内各完成一次卷积运算同时在全连接层计满 9 个时也抢断卷积完成一次运算，将所得数据存储。在这种设计下可以只使用 9 个乘法器完成计算，并且由于可以流水输入，完全不停止，在 100 周期的数据处理下乘法器的利用率达到了 **91.5%**。

0	1	2	3	4	5	6	7	8	9	10	11
1	1	2	3	4	5	6	7	8	9	10	11
2	12	13	14	15	16	17	18	19	20	21	22
3	23	24	25	26	27	28	29	**30**	31	32	33
4	34	35	36	37	38	39	40	41	42	43	44
5	45	46	47	48	49	50	51	52	53	54	55
6	56	57	53	54	55	56	57	58	59	60	61
7	67	68	69	70	71	72	73	74	75	76	77
8	78	79	80	81	82	83	84	85	86	87	88
9	89	90	91	92	93	94	95	96	97	98	99
10	100	101	**102**	103	104	105	106	107	108	109	110
11	111	112	113	114	115	116	117	118	119	120	121

图 2-2 三周期卷积周期功能与流水设计

由于考虑到需要控制简单，因此如果能使得每一组矩阵数据开始卷积处理之后便不停止，即为最容易的控制逻辑，由于 108，110，111 这几个周期的特殊性，在某些周期开始不间断的 3 周期运算容易发生数据未送达便完成计算的违例行为，因此编写了一个 python 程序用于寻找最早开始不间断的 3 周期的运算而不导致数据违例的最好的起始周期。最终通过 python 找到的最早的周期为第 30 个周期。而由于完成 **12 个卷积**之后的周期，会出现 fc 数据存满的抢断，实际的开始周期可以选择第 29 个周期而不发生卷积数据的违例。

2.3. 输入存储模块的大小规划

最早的设想中，我们打算采用 13×13 的寄存器，将 Padding 存储后，以最容易想到的框选控制逻辑进行数据的存储与向下一级传递。通过 DC 综合后发现，改规模的寄存器具有较大的面积同时，原先的逻辑产生了 2 组 13×13 的 MUX 组（准确的说，一组 MUX 和一组 DEMUX），不仅有较大的组合逻辑面积，同时也有很大概率在布线时发生较大的问题。

基于此，将 Padding 的内容通过了组合逻辑实现。

此外，由于已经完成卷积计算而且超出步长的数据可以被覆盖，因此只需要找到一种**最小的存储结构**，使得当最新的数据输入时恰好不覆盖还未被卷积计算的数据，该结构如果为每行 11 个寄存器，可以比较好的编写控制逻辑。

借助于 python 程序遍历得到了在 2.2 中规划的 3 周期串行的最早起始条件下卷积运算中所需数据与输入数据的最长距离，发现最长路径出现在第一行最后的卷积运算中，最长的行数达到了 5 行，而由于计算第一行数据时，有一行数据为组合逻辑产生的 Padding 数据，因此只需要采用 **11×4** 的寄存器组即可完成 2.2

设计的 3 周期每个卷积核串行计算的需求。

2.4. 并行架构设计

第一版的设计采用了计算的并行结构，该版本在卷积和全连接的计算时复用了乘加器，采用了 13*13*8 的寄存器储存输入数据，采用 2 个 9*3*8 的寄存器分别储存三个卷积核和全连接权重，采用 3*12*8 个寄存器储存卷积送出的池化预存数据，采用 9*3*8 储存池化结果。

由于非常大的数据存储矩阵和 MUX 结构以及 27 个乘法器，在这一版本下的面积远超出了设计的要求，因此弃用。

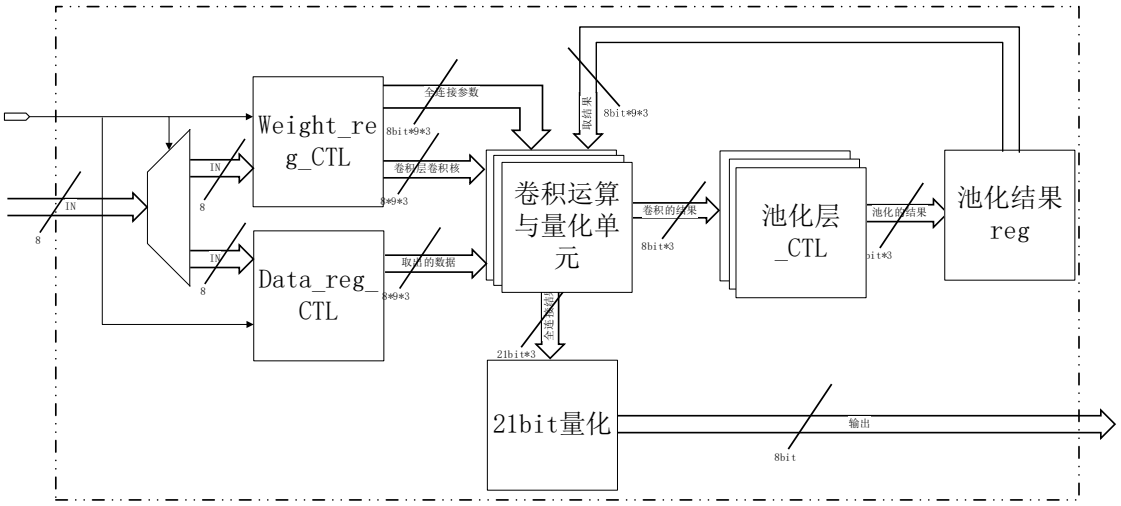

图 2-3 并行版本架构

2.5. 串行架构设计

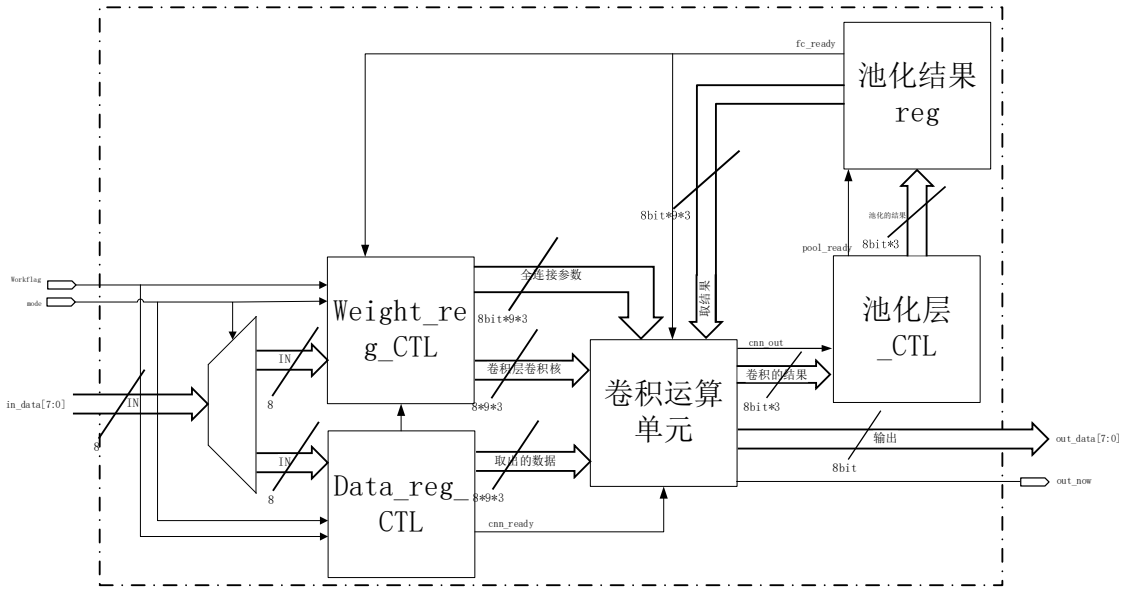

图 2-4 **串行版本架构**

在并行的架构面积过大的情况下，通过减少输入存储的寄存器组的大小，把大量寄存器用移位寄存器方式替代以及使用 9 个乘法器的串行结构后，面积有所改观。串行架构的部分设计逻辑的合理性在 2.2 和 2.3 节已进行了详细的论述和验证，在这里不过多补充。

串行架构由 5 个主要模块组成，其中权重存储模块在储存完权重后，根据需求给出当前所需的权重和卷积核的数据；数据存储模块进行数据的暂存、框选、卷积的开始与结束以及当全连接层占用乘加器的时候进行卷积暂停；卷积及输出模块需要仲裁 MAC 的使用、进行输出部分数据的暂存、调用 MAC 并仲裁结果的传递是输出还是池化层；池化层暂存部分数据并完成最大池化；全连接数据（池化输出数据）存储模块需要完成池化结果的存储并在每 9 个存满后完成数据的传递。

2.6. IO 分配

芯片的 IO 分配详见表格 1 IO 分配。

表格 1 IO **分配**

PIN		I/O	功能描述
NAME	占用 IO		
VDD	3	POWER	电源，包含 2 个 IO 供电和 1 个 core 供电
VSS	**4**	POWER	地，包含 2 个 IO 供电和 **2** 个 core 供电
clk	1	I	时钟输入

rst_n	1	I	复位信号
in_data[7:0]	8	I	8bit 数据和权重输入，为 2 进制补码有符号 8bit 数，其中输入数据为 11×11 的矩阵，数据位宽为 8bit，一共有一百个 11×11 的矩阵
mode	1	I	模式信号，当信号为 1 时输入为权重，为 0 时输入为数据
work_flag	1	I	控制输入信号，当信号为 1 时输入为有效输入，为 0 时数据为无效输入
out_data[7:0]	8	O	8bit 数据输出
out_now	1	O	数据输出时拉高

3. 各个具体的模块设计：

3.1. 权重存储模块设计

由于要减少 MUX 的存在，以及配合后续的串行的卷积核数据和全连接权重的给出，权重存储模块的输入数据的存储采用了以 9 个为一组的移位寄存器输入结构。其中，在 work_flag 信号和 mode 信号均为 1 的时候进行输入。

由于权重完成一次存储后便不需要输入，而如果从寄存器中选择卷积核和全连接权重需要 144 组 3 选 1 的 MUX，布线较为复杂且逻辑较麻烦，因此将其中 9 个寄存器数据直接输出到卷积运算模块，同时通过每 9 个寄存器为 1 组形成 3 组移位寄存器组，在全连接和卷积的计算周期进行适当的移位配合运算。

最终的权重存储模块设计如图 3-1 所示。

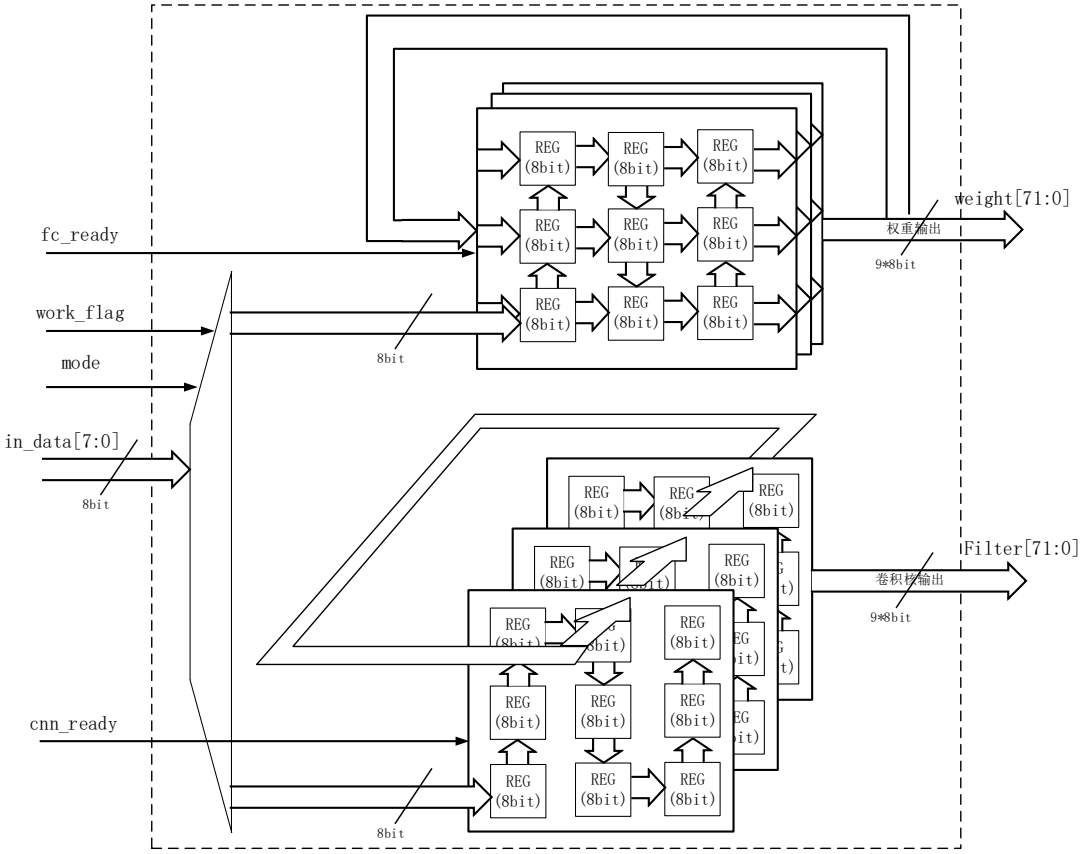

图 3-1 权重存储模块设计

3.2. 输入数据存储和框选模块

输入数据的存储与框选，共使用了 11×4 的 8bit 寄存器组，详细的设计见图 3-2，其设计的合理性在 2.3 中详细的论述，此处不再多做讨论。

由于存储模块与逻辑中的模块不一致，因此在 x 方向只采用一个计数器处理输入，在 y 方向采用两个计数器，同时处理逻辑的行数和实际的行数，逻辑的计数器给出卷积运算的开始信号，并给卷积模块和权重模块执行卷积操作的允许信号 cnn_ready，实际计数器给出实际的存储位置。

当外部 cnn_busy 传入时，内部卷积信号拉低，权重计数等均暂停，同时 cnn_ready 也拉低。

框选模块同样使用两种计数器，由于框选中 y 的变化在最后一行与其他行逻辑不同，因此要靠逻辑计数器给出边界信号。

通过框选时的逻辑寄存器的边界位置判断，通过控制电路直接给 Padding 时添加了 8bit 的 0。

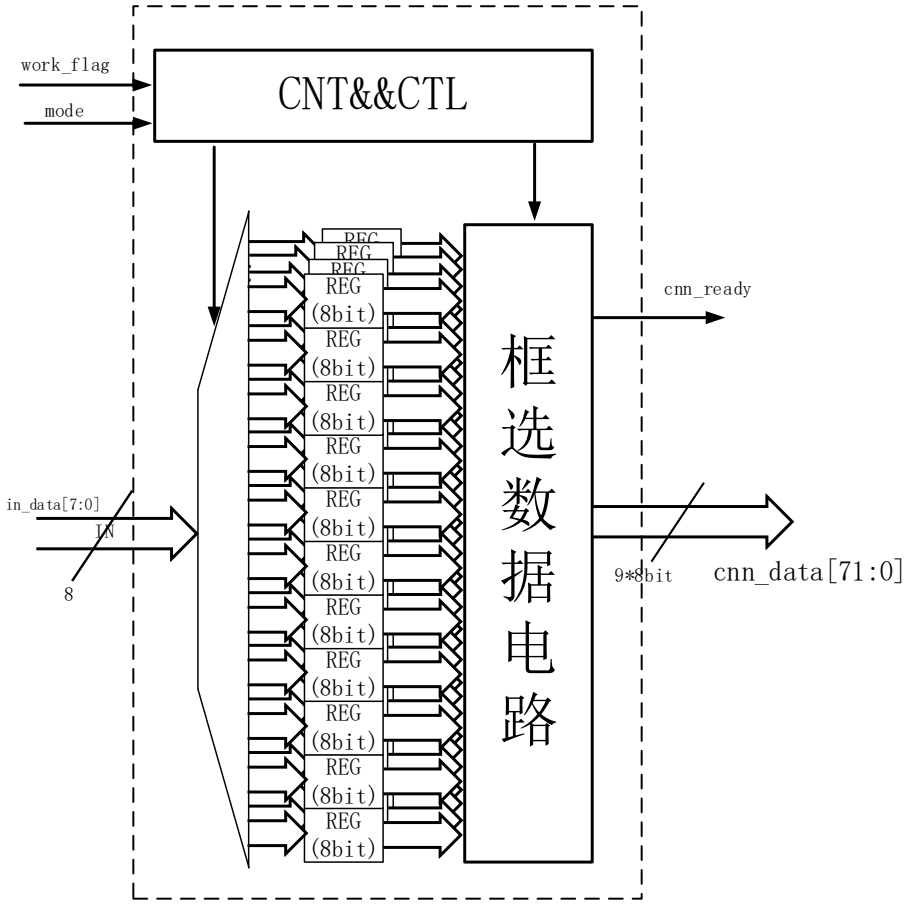

图 3-2 输入数据存储和框选模块

3.3. 卷积处理与输出模块设计

卷积处理与输出模块设计见图 3-3，通过 cnn_ready 和 fc_ready 两个信号仲裁乘法器的数据选项，其中 fc_ready 信号拉高时具有高优先级，优先执行全连接层的计算。

数据处理包含 9 个 8*8 的有符号补码乘法器，9 个 20bit 的有符号补码加法器，一个量化器输入数据仲裁模块，3 个 21bit 的有符号补码加法器，一个 21bit 转 20bit 的量化器。

实验发现，采用“*”和“+”在 DC 综合中得到的面积和速度均时最优的，因此乘加器采用简单的写法。

由于采用了串行结构，根据数据流的结构，同一周期内只需要 1 个量化器即可完成处理，而由于量化规则一致，扩展不影响数字大小，20bit 有符号补码扩展到 21bit 后通过 21bit 转 8bit 的量化器得到的量化结果与直接通过 20bit 转 8bit 的量化器的结果是一致的，因此整个模块的复用程度达到了最高。

通过一个有限状态机（FSM）结合仲裁信号，对加法结果是否存储、量化的

输入数据等进行了控制。

此外，通过 cnn_out 和 out_now 来指示对应后续的模块来捕获对应的数据。

由于 out_data 的数据需要连接到外部，因此在获取到数据后，由 FSM 先控制一组输出寄存器将对应的数据存储后再进行输出，有效的保证了输出信号的稳定性。

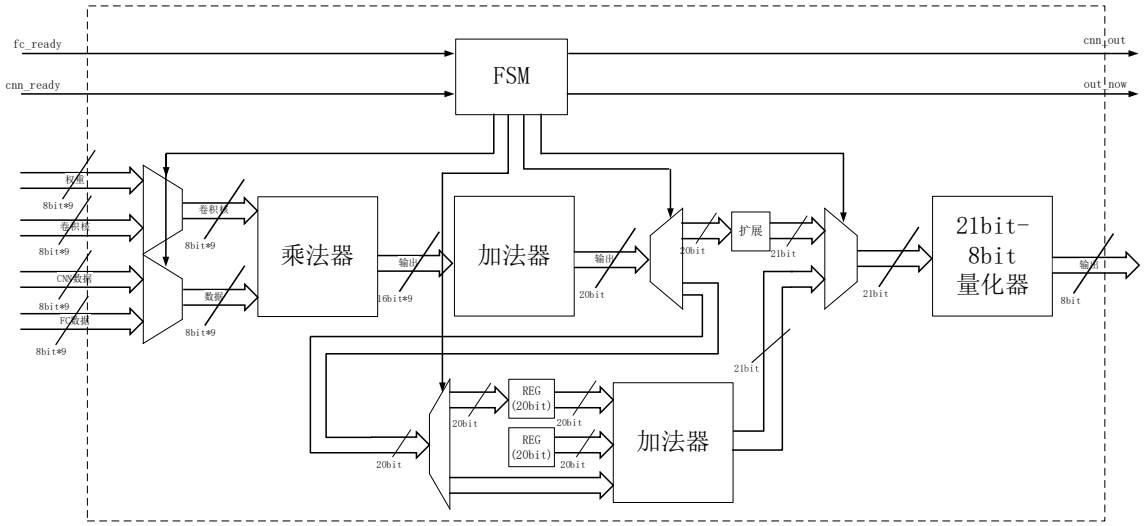

图 3-3 卷积处理与输出模块

3.4. 池化模块设计

由于池化模块为 2*2，步长为 2 的最大池化，因此在顺序卷积下需要存储 7*3 组的 8bit 数据。图 3-4 详细的展示了这部分的存储模块，采用了 3 个长为 7 的 8bit 移位寄存器。由于每 3 个周期内会有同一个卷积位置的不同卷积核的卷积结果输出，因此通过一个 FSM 不断在对应的周期把数据送到 3 个卷积核对应的移位寄存器，并在图示的位置片选出对应的结果进行比较。由于硬件电路不能终止比较器，因此只使用计数器模块在适合的周期产生 pool_flag 给下一个模块标志此时的输出为得到的输出。

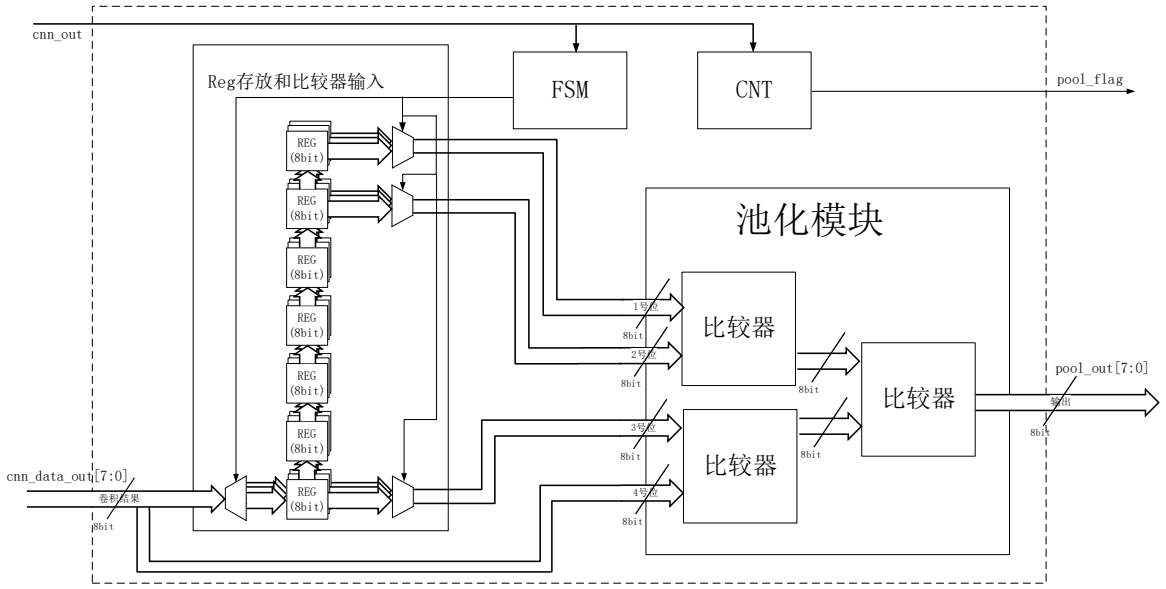

图 3-4 池化模块

3.5. 全连接数据（池化输出数据）存储模块设计

全连接数据（池化输出数据）存储模块设计见图 3-5，该模块的设计包含一个长 9 的 8bit 移位寄存器，通过计数器记录到数据记录满时将信号 fc_ready 拉高，由卷积处理模块仲裁并处理全连接数据。

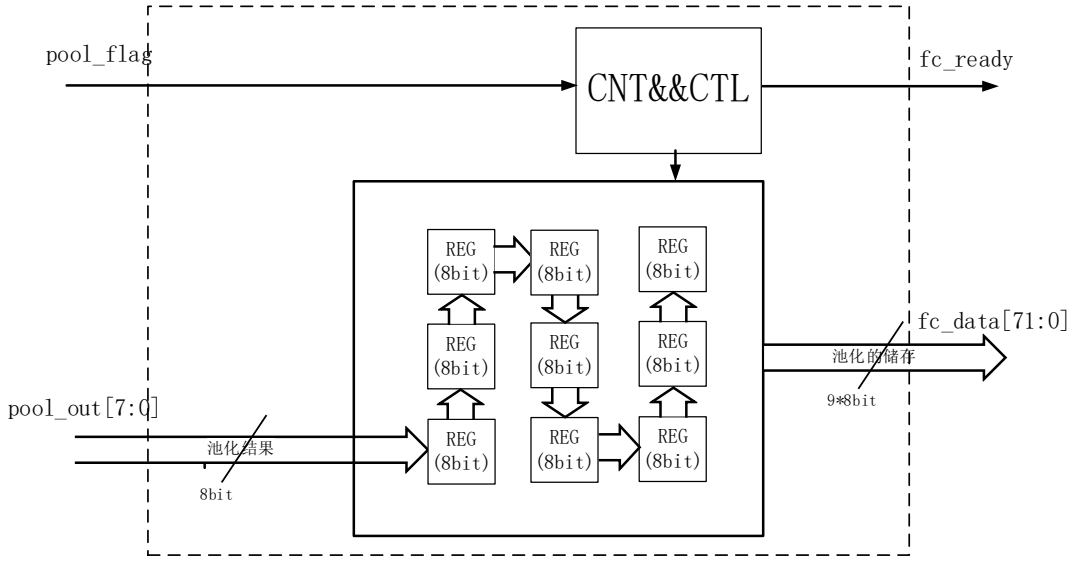

图 3-5 全连接数据（池化输出数据）存储模块

4. 仿真与分析

4.1. 整体仿真结果

所设计的电路在多种测试数据下均正确通过，在我们组设计的 TB 和助教的随机数据行为级模型 TB 下的 100 张图片测试结果全部通过。

根据记录，第一个矩阵加权重输入共计 195 个周期完成，第一个矩阵净需要 141 个周期，总计 121 个矩阵 12174 个周期，净需要 12120 个周期。

后续由于关键路径的原因在中间插入了一级或几级寄存器，最终需要的周期会有所增加，具体的周期数见 5.2 中的的表格 2 优化策略与效果。

```
out  80 : out = 11110110, ture = 11110110, ***PASS***
out  81 : out = 11110101, ture = 11110101, ***PASS***
out  82 : out = 11110011, ture = 11110011, ***PASS***
out  83 : out = 11110101, ture = 11110101, ***PASS***
out  84 : out = 11110011, ture = 11110011, ***PASS***
out  85 : out = 11110100, ture = 11110100, ***PASS***
out  86 : out = 11111000, ture = 11111000, ***PASS***
out  87 : out = 11111001, ture = 11111001, ***PASS***
out  88 : out = 11111000, ture = 11111000, ***PASS***
out  89 : out = 11110101, ture = 11110101, ***PASS***
out  90 : out = 11110111, ture = 11110111, ***PASS***
out  91 : out = 11110100, ture = 11110100, ***PASS***
out  92 : out = 11110010, ture = 11110010, ***PASS***
out  93 : out = 11110011, ture = 11110011, ***PASS***
out  94 : out = 11110010, ture = 11110010, ***PASS***
out  95 : out = 11101110, ture = 11101110, ***PASS***
out  96 : out = 11110000, ture = 11110000, ***PASS***
out  97 : out = 00010100, ture = 00010100, ***PASS***
out  98 : out = 00101101, ture = 00101101, ***PASS***
out  99 : out = 00001100, ture = 00001100, ***PASS***
**************PASS**************
```

图 4-1 仿真断言的结果

```
check_ans_82 70 (dut) == 70 (ref)
check_ans_83 29 (dut) == 29 (ref)
check_ans_84 26 (dut) == 26 (ref)
check_ans_85 44 (dut) == 44 (ref)
check_ans_86 45 (dut) == 45 (ref)
check_ans_87 65 (dut) == 65 (ref)
check_ans_88 f1 (dut) == f1 (ref)
check_ans_89 33 (dut) == 33 (ref)
check_ans_90 55 (dut) == 55 (ref)
check_ans_91 0d (dut) == 0d (ref)
check_ans_92 63 (dut) == 63 (ref)
check_ans_93 36 (dut) == 36 (ref)
check_ans_94 76 (dut) == 76 (ref)
check_ans_95 4f (dut) == 4f (ref)
check_ans_96 6b (dut) == 6b (ref)
check_ans_97 30 (dut) == 30 (ref)
check_ans_98 56 (dut) == 56 (ref)
check_ans_99 5a (dut) == 5a (ref)
**************PASS**************
first_img_time                  392
total time                    24350
```

图 4-2 师兄随机版本的仿真断言结果和所用周期

4.2. 局部数据比较

在设计早期，由于部分的逻辑的错拍，导致所取得数据偏移而引起了大量的不定态。修复 bug 后的仿真具体波形如图 4-3、图 4-4、图 4-5、图 4-6 所示，各个关键模块的控制波形正确运行，其中可以中图 4-3 的 fc_ready 的每个矩阵规律性的三次拉高以及 cnn_ready 中途的 2 次拉低判断得到卷积处理模块的仲裁器正确工作。此外，其余几个关键模块的控制信号也按照预期的波形给出。

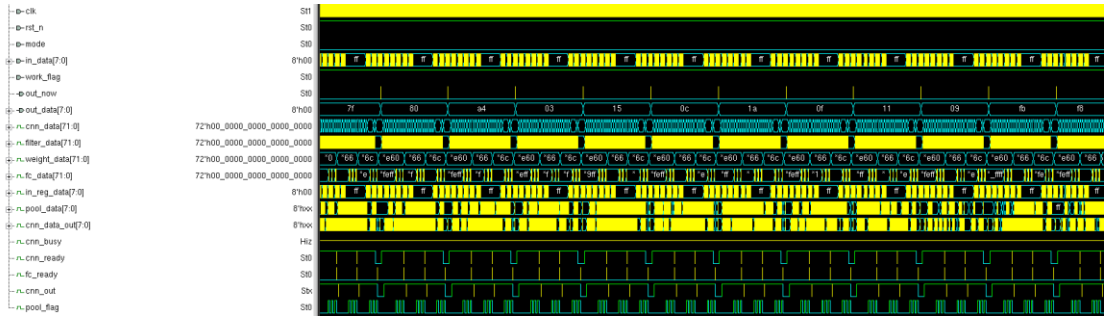

图 4-3 顶层交互波形

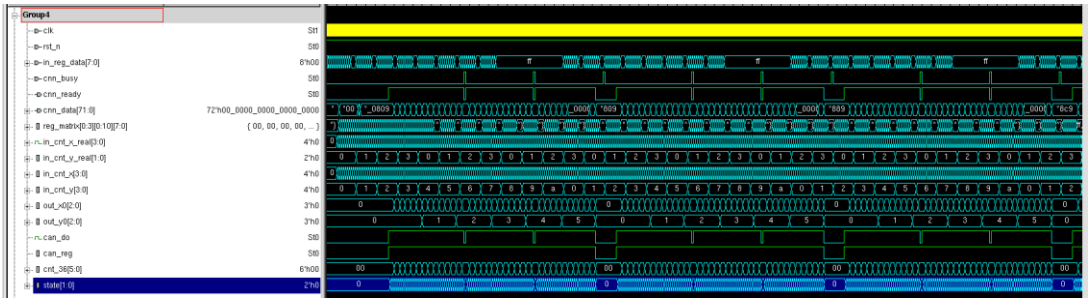

图 4-4 数据存储与框选模块波形

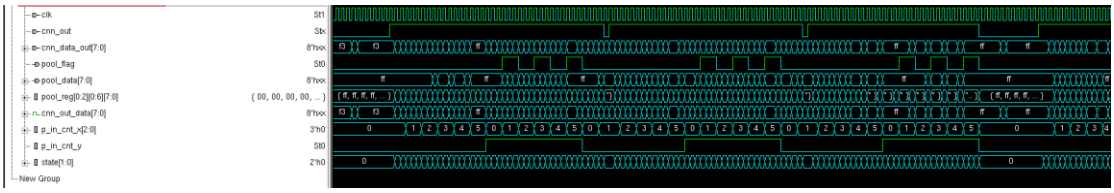

图 4-5 池化层仿真波形

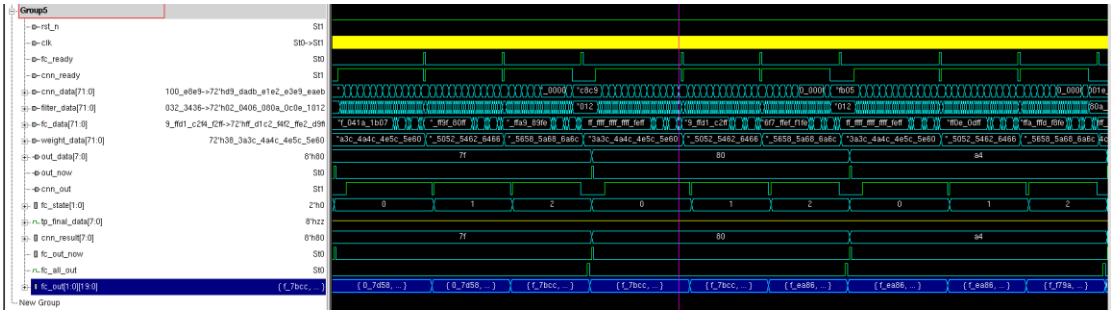

图 4-6 卷积控制模块波形

在图 4-7 中截取了某个周期当 fc_ready 拉高时输入乘法器的数据，与图 4-8 中 ai model 输出的乘法器计算时的 pool 的输出数据和权重比较，发现相应的数据和权重一一对应，同时 9 组数据的顺序也与参考数据的一致，说明了所设计的系统的运行的准确性。

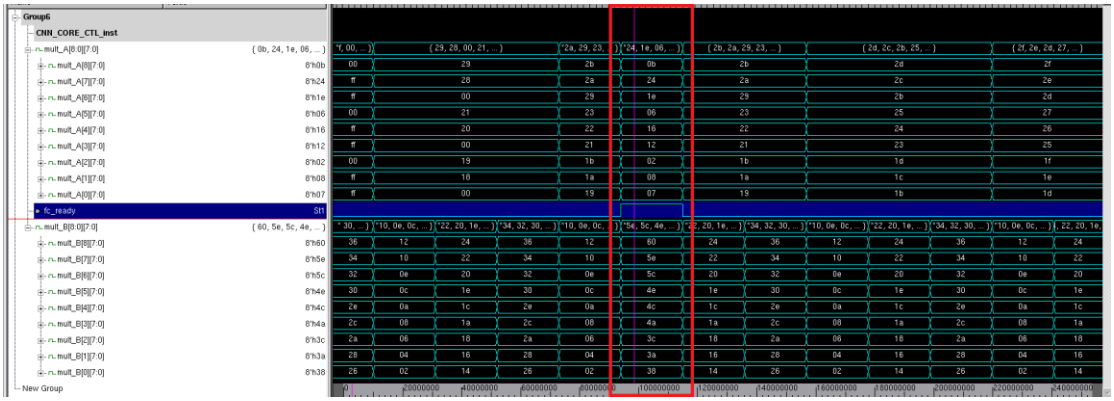

图 4-7 乘法器运行中的输入数据

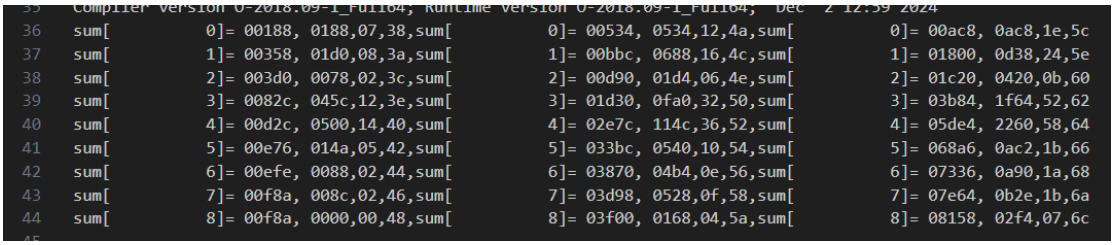

图 4-8 ai_model 输出的 pool 结果

4.3. Testbench 的改进

由于早期设计的 TB 的数据的变化与时钟上升沿同步，因此后仿真时进行模拟由于不同传输路径造成的数据毛刺导致的数据异常无法避免，根据和助教的交流和参考一些通信模块的设计例程，发现 ICC 优化的时钟路径仅限于芯片内部，因此时钟上升沿的外部数据应该保证已经稳定一段时间，通过修改 TB 在时钟的下降沿进行数据的发送，成功使得芯片内部再数据稳定的情况下读入数据。

5. 设计的关键路径与改进

5.1. 关键路径

设计中的关键路径具体可以见图 5-1，由输入数据寄存器到池化输出寄存器。这两个寄存器之间包含框选电路、乘法器、加法器、量化器、比较器 5 个组合的大模块。

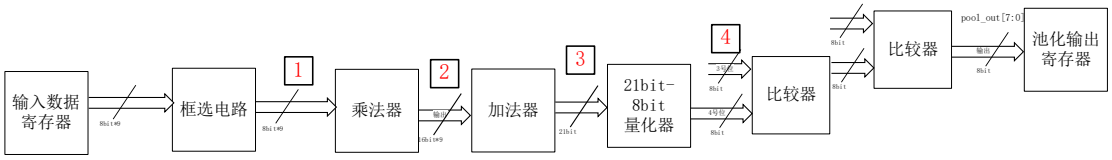

图 5-1 关键路径

5.2. 关键路径的改进

在图 5-1 中可以看到有 4 的独立的数据节点，通过在这些节点之间适当的插入寄存器可以得到不同的面积和最高频率的结果。

在优化过程中，共在原先的关键路径上改进了 5 版内容，具体可以见表格 2。

由于各部分设计的独立性较好，寄存器的插入改进非常容易，下面的优化所得代码前仿真均没有问题。

下述代码优化的具体时序和面积的表现将会在综合报告中详细给出，在这里就不在做多余的说明。

表格 2 优化策略与效果

版本	插入节点	说明	第一个矩阵周期数	第一个矩阵（无权重）	总周期数	总周期数（无权重）
431	原版本	未进行时序优化的版本	195	141	12174	12120
433	节点 2	单个插入对时序的改进效果最好	196	142	12175	12121
434	节点 3	插入寄存器后面积最小	196	142	12175	12121
435	节点 2、3	时序改进好于单个节点插入的	197	143	12176	12122
436	节点 2、3、4	最高频率进一步提高	198	144	12177	12123
437	节点 1、2、3、4	最高频率最高，面积代价可以接受	199	145	12178	12124

附录

附 1 三周期卷积计算的最早非违例起点周期的模拟 python

```
a=[14,16,18,20,22,23,36,38,40,42,44,45,53,55,57,59,61,67,80,82,84,86,88,89,102,104,106,108,110,111,113,115,117,119,121,122]
c = []
for i in range(0, 1):
    for val in a:
        c.append(val +121 * i)
good = []

with open('c.txt', 'w') as file:
    for start in range(0,122):
        for i in range(0,36):
            if(c[i] > (start + 3 * i + 1)):
                print(c[i],(start + 3 * i + 1),i, file=file)
                print('\n', file=file)
                break
            else:
                print(c[i],(start + 3 * i + 1),i, file=file)
                if(i == 35):
                    good.append(start)
                    print('\n', file=file)
  with open('output.txt', 'w') as file:
    for item in good:
        print(item, file=file)
```

附 2 三周期最小规模输入存储寄存器验证 python

```
a=[14,16,18,20,22,23,36,38,40,42,44,45,53,55,57,59,61,67,80,82,84,86,88,89,102,104,106,108,110,111,113,115,117,119,121,122]
c = []
for i in range(0, 1):
    for val in a:
        c.append(val +121 * i)
good = []
ycell = 0
xcaca = 0
with open('d0.txt', 'w') as file:
    start = 30
    for i in range(0,36):
```

```
        now_x = (int) (i %6) * 2
        now_y = (int) (i /6) * 2
        now_x1 = now_x +1
        now_x2 = now_x +2
        now_y1 = now_y +1
        now_y2 = now_y +2
        in_x = (((start + 3 * i + 1))%11)
        in_y = (int)(((start + 3 * i + 1))/11)
        in_y1 = (int)(((start + 3 * i + 2))/11)
        in_y2 = (int)(((start + 3 * i + 3))/11)
        if(in_y - now_y + 1 > ycell):
            ycell = in_y - now_y + 1
        if(in_y1 - now_y + 1 > ycell):
            ycell = in_y1 - now_y + 1
        if(in_y2 - now_y + 1 > ycell):
            ycell = in_y2 - now_y + 1
        print(in_y,in_y1,in_y2,now_y,i+1, file=file)
        print('\n', file=file)
    print(ycell,file=file)
```