# 加速器与HPS连接关系



# 功能简介

CNN卷积模块（cnn\_top）实现普通卷积、dw卷积、relu激活等功能。

数据重排模块（data\_reorganize）实现了输入数据重排功能与输出数据重排功能，原始图像的数据要经过输入重排，将数据排列成CNN加速器卷积需要的数据格式进行卷积计算；当框架要使用卷积结果时，需要将卷积结果进行输出重排，框架使用输出重排的结果进行识别。

# 工作流程

1、Hps通过avalon总线向数据重排模块（data\_reorganize）下发：输入重排模式、通道数、特征图尺寸、输入数据地址、输出数据地址，配置完成后，再下发开始命令；数据重排模块会将原始图像数据排列成CNN加速器需要的数据格式。

2、Hps通过avalon总线向CNN卷积模块（cnn\_top）下发：parameter、scale、input、weight、output在DDR中的基地址，然后在下发开始命令；CNN卷积模块启动对数据进行卷积。

3、卷积完成后，框架要使用卷积结果需要进行输出数据重排，Hps通过avalon总线向数据重排模块（data\_reorganize）下发：输数重排模式、通道数、特征图尺寸、输入数据地址、输出数据地址，配置完成后，再下发开始命令；数据重排模块将数据排列成框架能使用的数据模式。

# 接口说明

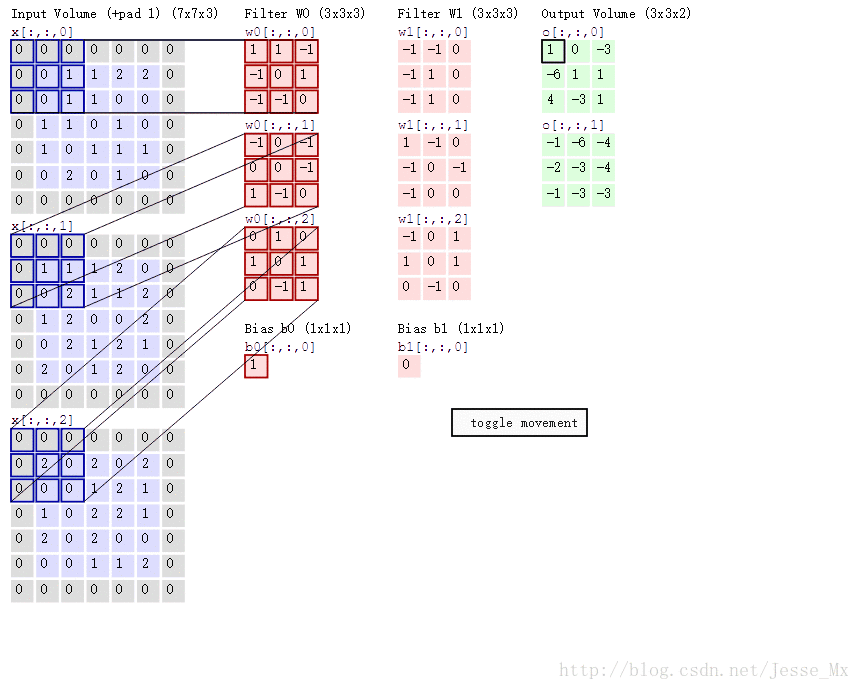
#### CNN接口说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 信号名 | 信号位宽 | 端口类型（I/O） | 信号来源/去向 | 详细描述 |
| sysclk | 1 | I | PLL | 系统时钟 |
| rst\_n | 1 | I | external | 系统复位 |
| as\_rd | 1 | I | HPS | 与HPS的HPS连接的avalon总线从机接口，接收parameter、scale、input、weight、output在DDR的基地址以及卷积开始命令 |
| as\_wr | 1 | I |
| as\_addr | 3 | I |
| as\_rd\_data | 32 | O |
| as\_wr\_data | 32 | I |
| as\_data\_waitquest | 1 | O |
| cfg\_avm\_read | 1 | O | DDR | 与HPS的DDR控制器连接的avalon总线，从DDR中读取卷积控制参数 |
| cfg\_avm\_address | 32 | O |
| cfg\_avm\_readdata | 32 | I |
| cfg\_avm\_readdatavalid | 1 | I |
| cfg\_avm\_byteenable | 4 | O |
| cfg\_avm\_burstcount | 3 | O |
| cfg\_avm\_waitrequest | 1 | I |
| load\_avm\_read | 1 | O | DDR | 与HPS的DDR控制器连接的avalon总线，从DDR中读取input、weight |
| load\_avm\_address | 32 | O |
| load\_avm\_readdata | 128 | I |
| load\_avm\_readdatavalid | 1 | I |
| load\_avm\_byteenable | 16 | O |
| load\_avm\_burstcount | 3 | O |
| load\_avm\_waitrequest | 1 | I |
| scale\_avm\_read | 1 | O | DDR | 与HPS的DDR控制器连接的avalon总线，从DDR中读取store模块计算需要的scale、weight\_scale参数 |
| scale\_avm\_address | 32 | O |
| scale\_avm\_readdata | 32 | I |
| scale\_avm\_readdatavalid | 1 | I |
| scale\_avm\_byteenable | 4 | O |
| scale\_avm\_burstcount | 3 | O |
| scale\_avm\_waitrequest | 1 | I |
| output\_avm\_write | 1 | O | DDR | 与HPS的DDR控制器连接的avalon总线，将卷积计算结果写回DDR中 |
| output\_avm\_address | 32 | O |
| output\_avm\_writedata | 128 | O |
| output\_avm\_byteenable | 16 | O |
| output\_avm\_waitrequest | 1 | I |
|  |  |  |  |  |

#### DATA\_REORG接口说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 信号名 | 信号位宽 | 端口类型（I/O） | 信号来源/去向 | 详细描述 |
| exclk | 1 | I | externa | 外部时钟 |
| rst\_n | 1 | I | external | 外部复位 |
| as\_rd | 1 | I | HPS | 与HPS连接的avalon总线从机接口， |
| as\_wr | 1 | I |
| as\_addr | 8 | I |
| as\_rd\_data | 32 | O |
| as\_wr\_data | 32 | I |
| as\_data\_waitquest | 1 | O |
| r\_avm\_waitrequest | 1 | I | DDR | 与HPS的DDR控制器连接的avalon总线 |
| r\_avm\_read | 1 | O |
| r\_avm\_address | 32 | O |
| r\_avm\_readdata | 64 | I |
| r\_avm\_readdatavalid | 1 | I |
| r\_avm\_byteenable | 8 | O |
| r\_avm\_burstcount | 5 | O |
| w\_avm\_waitrequest | 1 | I | DDR | 与HPS的DDR控制器连接的avalon总线 |
| w\_avm\_write | 1 | O |
| w\_avm\_address | 32 | O |
| w\_avm\_writedata | 64 | O |
| w\_avm\_byteenable | 8 | O |
| w\_avm\_burstcount | 5 | O |

# 数据重排说明



如图展示的是一个3输入通道，2输出通道的卷积处理流程。

FPGA要实现这样的功能是比较简单的，简单计算一下可以得出大概需要：

存储空间：219byte

乘法器：9\*3个

加法器：10个

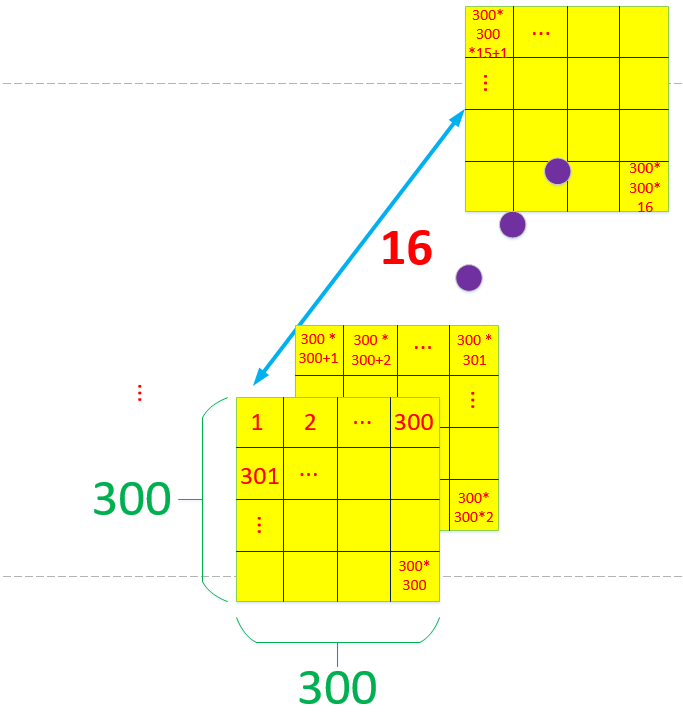
当特征图尺寸变大、输入通道、输出通道变多之后，要实现卷积功能所需要的存储空间、乘法器、加法器数量将以几何倍数增。MobileNetV1模型特征尺寸最大为224\*224，输入输出通道最多为1024，按照上面的卷积方法在FPGA上难以实现



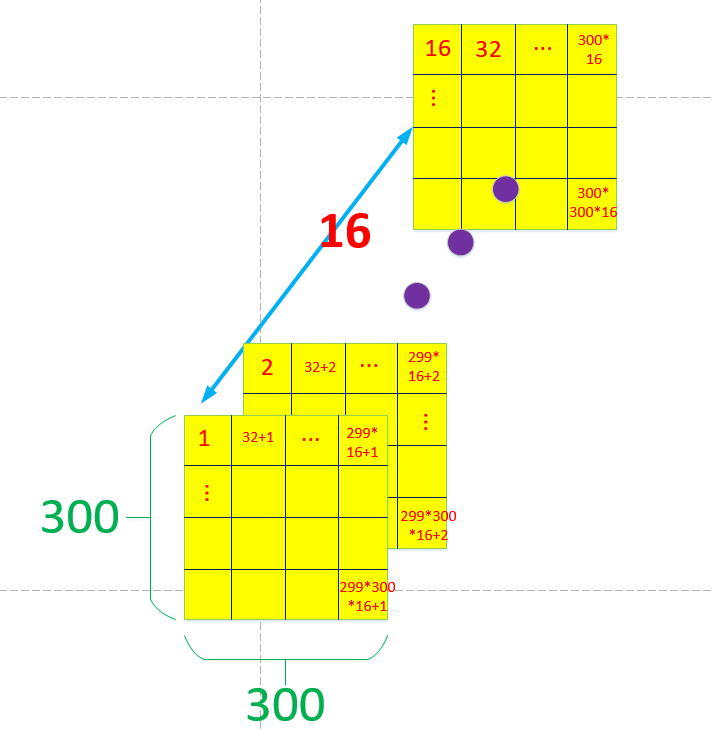
为解决FPGA资源问题进行通道接切片减少乘法器数量，如下图所示，将输入通道和输出通道划分成一个个通道切片，每次对一个输入/输出通道进行卷积。



一个切片的数据排列如下图所示：



这样的排列在CNN加速器读取数据时，因为地址并不连续所以读取效率低下，所以需要数据重新排成下图的数据格式方便CNN加速器读取数据的连续性，提升数据读取的效率。



**原理代码参考**

1. **void** input\_reorganized(int8\_t \*src, int8\_t \*dst, **int** in\_c, **int** in\_h, **int** in\_w)
2. {
3. **int** input\_c = up\_round(in\_c, INPUT\_EXTEND\_SCALE);
4. **for** (**int** i = 0; i < input\_c; i++)
5. {
6. **for** (**int** r = 0; r < in\_h; r++)
7. {
8. **for** (**int** c = 0; c < in\_w; c++)
9. {
10. **for** (**int** k = 0; k < INPUT\_EXTEND\_SCALE; k++)
11. {
12. **if** (i \* INPUT\_EXTEND\_SCALE + k < in\_c)
13. {
14. int8\_t temp =
15. src[((i \* INPUT\_EXTEND\_SCALE + k) \* in\_h + r) \* in\_w + c];
16. dst[((i \* in\_h + r) \* in\_w + c) \* INPUT\_EXTEND\_SCALE + k] = temp;
17. }
18. **else**
19. {
20. dst[((i \* in\_h + r) \* in\_w + c) \* INPUT\_EXTEND\_SCALE + k] = 0;
21. }
22. }
23. }
24. }
25. }
26. }
28. **void** output\_reorganize(int8\_t \*src, int8\_t \*dst, **int** out\_c, **int** out\_h, **int** out\_w)
29. {
30. **int** output\_channel\_block = (out\_c - 1) / INPUT\_EXTEND\_SCALE + 1;
31. **for** (**int** i = 0; i < output\_channel\_block; i++)
32. {
33. **for** (**int** r = 0; r < out\_h; r++)
34. {
35. **for** (**int** c = 0; c < out\_w; c++)
36. {
37. **for** (**int** k = 0; k < INPUT\_EXTEND\_SCALE; k++)
38. {
39. **if** (i \* INPUT\_EXTEND\_SCALE + k < out\_c)
40. {
41. **int** sw\_index =
42. (i \* INPUT\_EXTEND\_SCALE + k) \* out\_h \* out\_w + r \* out\_w + c;
43. **int** hw\_index =
44. ((i \* out\_h + r) \* out\_w + c) \* INPUT\_EXTEND\_SCALE + k;
45. dst[sw\_index] = src[hw\_index];
46. }
47. }
48. }
49. }
50. }
51. }

# 寄存器说明

#### CNN寄存器说明

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 寄存器名 | 数据位宽 | 偏移地址 | 基地址 | 详细描述 |
| cnn\_ddr\_in | 32 | 0x10 |  | 输入数据存放在ddr中的物理地址 |
| cnn\_ddr\_out | 32 | 0x28 |  | 输出数据存放在ddr中的物理地址 |
| cnn\_ddr\_scale | 32 | 0x40 |  | scale参数存放在ddr中的物理地址 |
| cnn\_ddr\_w | 32 | 0x1C |  | 权重数据存放在ddr中的物理地址 |
| cnn\_ddr\_param | 32 | 0x34 |  | 卷积参数存放在ddr中的物理地址 |
| cnn\_start | 32 | 0x00 |  | cnn\_start[31:1]:保留  cnn\_start[0]: 写1启动卷积；读1正在进行卷积操作，读0卷积操作完成。 |

# IP调用说明

启动IP前，需要先给各寄存器写入相应数据，配置好IP

**配置IP**

1. **int** fpga\_init(**void**)
2. {
3. **if** (fpga\_init\_status)
4. {
5. **return** 0;
6. }
7. devmem\_fd = open("/dev/mem", O\_RDWR | O\_SYNC);
8. fpga\_reg\_address\_map(devmem\_fd);
10. devcma\_fd = open("/dev/cmadrv0", O\_RDWR);
11. **if** (devcma\_fd < 0)
12. {
13. printf("open drvier failed\n");
14. exit(-1);
15. }
16. // cma\_blk\_s cb\_data,cb\_weight,cb\_scale,cb\_param,cb\_org;
17. cb\_data.size = FPGADATA\_CNN\_DATA\_SIZE;
18. cb\_weight.size = FPGADATA\_CNN\_WEIGHT\_SIZE;
19. cb\_param.size = FPGADATA\_CNN\_PARAM\_SIZE;
20. cb\_scale.size = FPGADATA\_CNN\_SCALE\_SIZE;
21. cb\_org.size = FPGADATA\_ORGANIZE\_DATA\_SIZE;
23. fpga\_data\_address\_cmamap(devcma\_fd, &cb\_data, &udata);
24. fpga\_data\_address\_cmamap(devcma\_fd, &cb\_weight, &uweight);
25. fpga\_data\_address\_cmamap(devcma\_fd, &cb\_param, &uparam);
26. fpga\_data\_address\_cmamap(devcma\_fd, &cb\_scale, &uscale);
27. fpga\_data\_address\_cmamap(devcma\_fd, &cb\_org, &uorganize);
29. printf("cb\_data.phy:%x\r\n", cb\_data.phys);
30. printf("cb\_weight.phy:%x\r\n", cb\_weight.phys);
31. printf("cb\_param.phy:%x\r\n", cb\_param.phys);
32. printf("cb\_scale.phy:%x\r\n", cb\_scale.phys);
33. printf("cb\_org.phy:%x\r\n", cb\_org.phys);
35. foo\_set(foo, FPGAREG\_CNN\_DDROUT, cb\_data.phys);
36. foo\_set(foo, FPGAREG\_CNN\_DDRIN, cb\_data.phys);
37. foo\_set(foo, FPGAREG\_CNN\_DDRW, cb\_weight.phys);
38. foo\_set(foo, FPGAREG\_CNN\_PARAM, cb\_param.phys);
39. foo\_set(foo, FPGAREG\_CNN\_SCALE, cb\_scale.phys);
41. fpga\_init\_status = **true**;
43. **return** 0;
44. }

此处把寄存器里写入了相对应的地址。

1. IP的输出数据起始地址
2. IP的输入数据起始地址
3. 权重数据的起始地址
4. 参数数据的地址
5. scale的地址

IP启动时，会在对应地址读取数据。所以在启动IP前还需要把相应数据写入到这里配置的地址处。

1. 写入参数数据到参数数据的地址处（参数数据里包含了输入数据偏移量，输出数据偏移量，权重数据偏移量等信息）
2. 写入scale数据到scale的地址处
3. 启动IP（启动时，会读取参数数，在参数数据里读出各类数据偏移量，后续启动IP后会使用）

**启动IP（start寄存器写1表示启动，判断到start寄存器清零表示IP执行完成）**

1. **int** start\_fpga(uint32\_t \*ip, uint32\_t start\_reg\_addr, **const** **char** \*ip\_name)
2. {
3. uint32\_t status;
4. status = foo\_get(ip, start\_reg\_addr);
5. status |= 0x1;
6. foo\_set(ip, start\_reg\_addr, status);
8. status = foo\_get(ip, start\_reg\_addr);
9. auto waitip\_start = std::chrono::steady\_clock::now();
10. **while** (status & 1){
11. status = foo\_get(ip, start\_reg\_addr);
12. std::chrono::duration<**float**> wait\_ip\_time = std::chrono::steady\_clock::now() - waitip\_start;
13. **if** (wait\_ip\_time.count() >= 5)
14. {
15. printf("error:wait ip fail:%s\n",ip\_name);
16. fpga\_release();
17. exit(-1);
18. }
19. }
20. **return** 0;
21. }

参数结构体如下：

1. **struct** FpgaConvParam: **public** NodeParam {
2. **float**             lr; // Leaky Relu alpha
3. int8\_t\*            ia{nullptr}; // input address, [N,Ci,Hi,Wi]
4. int8\_t\*            d\_ia{nullptr}; // device input address, [N,Ci/INPUT\_CHANNLE\_TILE,Hi,Wi,INPUT\_CHANNLE\_TILE]
5. int8\_t\*            ka{nullptr}; // kernel address, [Co,Ci,Hk,Wk]
6. int8\_t\*            d\_ka{nullptr}; // device kernel address, [Co/INPUT\_CHANNLE\_TILE,Ci,Hk,Wk,INPUT\_CHANNLE\_TILE]
7. int8\_t\*            oa{nullptr}; // output address, [N,Co,Ho,Wo]
8. int8\_t\*            d\_oa{nullptr}; // output address, [N,Co/INPUT\_CHANNLE\_TILE,Ho,Wo,INPUT\_CHANNLE\_TILE]
9. **float** \*            scale{nullptr}; // filter scale
10. **struct** parameter param;
11. };
12. **struct** parameter{
13. **int** input\_offset;
14. **int** weight\_offset;
15. **int** scale\_offset;
16. **int** output\_offset;
18. **int** input\_c;
19. **int** input\_h;
20. **int** input\_w;
21. **int** output\_c;
22. **int** output\_h;
23. **int** output\_w;
24. **int** kernel;
25. **int** in\_pad;
26. **int** out\_pad;
27. **int** stride;
28. **int** relu;
29. **int** type;//conv 1; dw\_conv:4
31. **int** output\_channel\_block\_num;
32. **int** output\_row\_tile;
33. **int** input\_row\_tile;
34. **int** output\_row\_block\_num;
36. **float** input\_scale;
37. **float** output\_scale;
39. **float** lr;
40. **int** dilation;
41. **int** weight\_size;
42. **int** input\_size;
43. **int** output\_size;
44. };