LayerParaScaleFloat16.v代码实现备注

1. layer type

输入信号，表示当前计算层类型

0：初始化层，数据准备阶段

1：卷积层操作

2：池化层

3：全连接层

1. 初始化层

layer\_type = 0

1. 特征数据初始化

外部传入特征数据，分别写入 PARA\_X 个 fm ram 中

使用非add模式写入fm ram

每个 fm ram 每次写入 PARA\_Y 个数据【todo：优化为一次写一行数据】

每个 fm ram 分为上下两部分，分别交替作为：输入读取区域，和，输出写入区域

外部数据传入完毕后，输入信号 init\_fm\_data\_done = 1

此时完成 fm ram 的初始化并输出 init\_fm\_ram\_ready = 1

1. 卷积核数据初始化

外部传入权重数据，分别写入 PARA\_KERNEL 个 weight ram 中

每个 weight ram 每次写入一个 slice 的数据，即 kernel\_size \* kernel\_size

每个 weight ram 分为上下两部分，初始化准备 2 \* PARA\_KERNEL 个 kernel 的数据

两份部分中，一个用于正在进行的运算，另一个用于准备好下一次运算的数据

外部数据传入完毕后，输入信号 weight\_ram\_done = 1

此时完成 fm ram 的初始化并输出 init\_weight\_ram\_ready = 1

1. 初始化完成，切换计算层

fm ram 和 weight ram 都初始化完成后，外部读取到init\_fm\_ram\_ready = 1 和 init\_weight\_ram\_ready = 1，即可状态转换到第一层的运算

通常第一层为卷积层

1. 卷积层

layer\_type = 1

1. fm ram输出域初始化

第一次进入卷积层时，zero\_write\_count 状态信号应该为 0，即还没进行 fm ram输出域初始化

cur\_fm\_swap 表示当前的输入域，取值为 0 或 1

(cur\_fm\_swap+1)-(((cur\_fm\_swap+1)/2)\*2) 表示当前的输出域，取值为 0 或 1

**取余公式：M % N = M-(M/N)\*N**

**以下公式中，不做特殊说明的情况下，除法均为下取整除法**

**与代码实现中的下取整除法保持一致**

输出域的起止范围为：

起始位置：

((cur\_fm\_swap+1)-(((cur\_fm\_swap+1)/2)\*2))\*`FM\_RAM\_HALF;

终止位置：

(((cur\_fm\_swap+1)-(((cur\_fm\_swap+1)/2)\*2))+1)\*`FM\_RAM\_HALF - 1;

`FM\_RAM\_HALF 为宏定义的 fm ram 中间位置

使用 fm 的 zero write 接口，对输出域的起止范围进行零值填充初始化

根据当前输出feature map的padding\_out值，设置**各个 fm ram起始写入位置**

第i个 fm ram 的起始写入位置

((padding\_out-0+`PARA\_X-1)/`PARA\_X)\*((fm\_size\_out+`PARA\_Y-1)/`PARA\_Y)\*`PARA\_Y+padding\_out;

padding\_out: po

PARA\_X: PX

PARA\_Y: PY

fm\_size\_out: fso，输出尺寸值包含了padding值，是整个输出fm 的尺寸

**上取整公式：(M+N-1)/N**

加入padding后

**起始的写入fm ram编号**

cur\_write\_start\_ram <= padding\_out-(padding\_out/`PARA\_X)\*`PARA\_X;

**结束的写入 fm ram 编号**

cur\_write\_end\_ram <= fm\_size\_out-(fm\_size\_out/`PARA\_X)\*`PARA\_X;

1. kernel 更新

在运算过程中，一次并行的kernel运算完成，切换到另一个kernel域读取数据，并发出更新kernel信号

update\_weight\_ram = 1 的情况下读取外部传入的kernel值写入weight ram 中

1. 卷积操作部分

（1）clk\_count = 0

如若当前不是要切换到下一个计算层的状态，则进行卷机操作的读取预设

重置卷积单元 conv\_rst <= 0;

fm ram 的初始化写入端口disable，读取端口 enable

fm ram**读取地址为**

cur\_fm\_swap\*`FM\_RAM\_HALF + cur\_x/`PARA\_X\*((fm\_size+`PARA\_Y-1)/`PARA\_Y)+cur\_y/`PARA\_Y+cur\_slice\*((fm\_size+`PARA\_Y-1)/`PARA\_Y)\*((fm\_size+`PARA\_X-1)/`PARA\_X);

cur\_x: cx

cur\_y: cy

cur\_slice: cl

fm\_size: fs

cur\_fm\_swap: cfSwap

FM\_RAM\_HALF: FMH

**输入空间的起始地址 + 当前行的起始地址 + 当前列的位置+ 当前slice的起始地址**

子读取地址，clk 0 时都为 0

weight ram 读取端口 enable

**weigh ram 读取地址为**

(cur\_kernel\_swap\*`DEPTH\_MAX+cur\_kernel\_slice)\*`KERNEL\_SIZE\_MAX\*`KERNEL\_SIZE\_MAX;

cur\_kernel\_swap: ckSwap

DEPTH\_MAX: DM

cur\_kernel\_slice: cks

KERNEL\_SIZE\_MAX: KSM

clk 0 时，每个 fm ram 都需要读取数据，cur\_fm\_ram <= 0;

（2）非clk\_count = 0 状态

开始进行卷积操作，conv\_rst <= 1

（3）weight ram 读取

clk\_count = 1

同步延迟，不作操作

其他 clk\_count 计数

读取地址累加1，当前读取值赋予conv模块

（4）fm ram 读取

clk\_count = 1

clk\_count = 0 时设置的读取地址获取的值赋予conv模块

读取地址累加1，子读取地址0

clk\_count > 1 && clk\_count <= kernel\_size

仅传入读取值的第一个值，即 `DATA\_WIDTH\*1 - 1:`DATA\_WIDTH\*0 部分

子读取地址累加1

clk\_count = kernel\_size 时，设置下一个clk第一个fm ram 开始读取

cur\_fm\_ram <= 0;

读取地址 fm\_addr\_read[0]为：

后面两部分：ks-1为除了第一个clk，后面还需读取的单个数据数目，[(ks-1)/PARA\_Y]为这些数据占据的fm ram行，一行fm占据的PARA\_Y行数，减去，单个数据读取占据的fm ram行，得到剩下改行还没读取的fm ram行，原地址基础上加上这个值，即跳转到下一个fm行的读取位置

子读取地址为 0

clk\_count%kernel\_size == 1

PARA\_X 组寄存器，之间移动数据，仅最后一组寄存器更新数据

根据cur\_fm\_ram，为当前读取的fm ram的读取地址累加1，子读取地址为0

其他 clk\_count 情况下，clk\_count <= (kernel\_size\*kernel\_size)

每次仅读取cur\_fm\_ram更新一个值

根据cur\_fm\_ram，为当前读取的fm ram的读取地址累加1

如果下一个clk是要切换到更新三个数据值的时刻，cur\_fm\_ram累加1

cur\_fm\_ram+1的读取地址为：

子读取地址为0

（5）conv单元运算ready，结果写入fm ram，更新读取地址、写入地址

对输出域写数据，得在zero\_write\_count = 1，即初始填零已经ready的情况下进行

写入数据得有一个等待时间确认计数 write\_ready\_clk\_count

fm ram使用并行写入模式，enable累加写入方式

ena\_para\_w <= 1; ena\_add\_write <= 1;

写入数据的地址为：

fm\_addr\_para\_write[0] <= fm\_zero\_start\_addr[0] + cur\_out\_slice\*((fm\_size\_out+`PARA\_X-1)/`PARA\_X)\*(((fm\_size\_out+`PARA\_Y-1)/`PARA\_Y)\*`PARA\_Y) + cur\_out\_index[0];

fm\_zero\_start\_addr: fzsa

cur\_out\_slice: cos

cur\_out\_index: coi

在一行内进行卷积时，下一次卷积起始地址，更新cur\_y，cur\_out\_index

cur\_y <= cur\_y + `PARA\_Y

cur\_out\_index[0] <= cur\_out\_index[0] + `PARA\_Y;

一行卷积结束，进入下一行卷积，更新cur\_x，cur\_y，cur\_out\_index

cur\_y <= 0;

如果还有下一行卷积

cur\_x <= cur\_x + `PARA\_X;

cur\_out\_index[0] <= (((cur\_out\_index[0] + `PARA\_Y + padding\_out)+`PARA\_Y-1)/`PARA\_Y)\*`PARA\_Y+padding\_out;

没有下一行卷积，即当前slice卷积已经完成，则进入下一个slice

cur\_slice <= cur\_slice + 1;

cur\_x <= 0;

cur\_y <= 0;

cur\_out\_index[0] <= ((padding\_out-0+`PARA\_X-1)/`PARA\_X)\*((fm\_size\_out+`PARA\_Y-1)/`PARA\_Y)\*`PARA\_Y+padding\_out;;

如果没有下一个slice，则表示当前kernel的卷积深度已完成，进入下一个kernel

cur\_slice <= 0;

cur\_x <= 0;

cur\_y <= 0;

kernel 计数累加:

kernel\_num\_count <= kernel\_num\_count + `PARA\_KERNEL;

kernel读取位置切换:

cur\_kernel\_swap <= ~cur\_kernel\_swap;

cur\_kernel\_slice <= 0;

发出kernel更新信息，以及，更新地址

update\_weight\_ram\_addr <= cur\_kernel\_swap\*`DEPTH\_MAX;

更新写fm ram的索引地址

cur\_out\_index[0] <= ((padding\_out-0+`PARA\_X-1)/`PARA\_X)\*((fm\_size\_out+`PARA\_Y-1)/`PARA\_Y)\*`PARA\_Y+padding\_out;

如果没有下一个kernel，则表示当前整个卷积层操作已完成，进入下一个计算层

conv\_to\_next\_layer <= 1;

（6）写入fm ram 等待计数处理

此部分确保进入下一层之前，最后一次写数据操作已完成

1. 池化层
2. 全连接层