1. 系统工作流程（yolov3\_tiny网络为例）



上面的图示展示了yolov3\_tiny网络在系统中的完整运行流程，其他网络的流程与其基本相同，所以这里再不进行赘述。

二.算子的寄存器配置

上面介绍的.param文件是包含网络结构参数的文件，对于完整网络的推理，就是根据.param文件按照ncnn中所定义的层为单位进行顺序解析，配置寄存器，调用NPU进行推理。NPU支持的单个算子主要包括通用卷积、深度卷积、全连接、最大值池化、均值池化、常用的激活函数和ALU操作等，下面以池化为例子，对寄存器的配置过程进行介绍。

在介绍算子寄存器配置之前需要对工程的目录结构进行一个简要介绍。

**src/**

├── config.h

├── hw

├── layer

├── lscript.ld

├── main.cpp

├── ncnn

├── net

├── README.txt

├── testLayer

├── testOp

└── utils

1. **config.h:**存放了整个系统工程全部的宏定义，重要的宏定义将在后文进行介绍；
2. **hw：存放了与硬件相关的函数文件，包括系统的初始化函数，以及底层算子推理的寄存器配置函数；**
3. **layer：**存放了ncnn的layer定义代码；
4. **main.cpp：**存放了系统工程的主函数；
5. **net：**存放了网络推理函数；
6. **testLayer：**存放了单个算子的仿真测试文件，包括卷积、池化、激活函数、ALU等，每种算子都包含了大量仿真case；
7. **utils：**存放了系统工程中调用的一些常用函数；

对于池化算子的寄存器列表和功能介绍可以详见**ncnnAccel RegMap.pdf**文档，这里介绍具体的配置过程。NPU所有寄存器的地址定义全部在**hw/accel\_params.h**文件中，**accel\_params.h**文件中定义了寄存器基地址ACCEL\_REG\_BASEADDR为0x80020000，每个寄存器以4个字节为单位，根据**ncnnAccel RegMap.pdf**文档中寄存器的索引ID可知，池化算子的寄存器地址为ACCEL\_REG\_BASEADDR + 40\*4至ACCEL\_REG\_BASEADDR + 49\*4，如下所示，

1. //POOL
2. #define ACCEL\_REG\_POOL\_CTRL         ACCEL\_REG\_BASEADDR + 40\*4
3. #define ACCEL\_REG\_POOL\_SHAPE\_IC     ACCEL\_REG\_BASEADDR + 41\*4
4. #define ACCEL\_REG\_POOL\_SHAPE\_IWH    ACCEL\_REG\_BASEADDR + 42\*4
5. #define ACCEL\_REG\_POOL\_SHAPE\_ICSTEP ACCEL\_REG\_BASEADDR + 43\*4
6. #define ACCEL\_REG\_POOL\_SHAPE\_OC     ACCEL\_REG\_BASEADDR + 44\*4
7. #define ACCEL\_REG\_POOL\_SHAPE\_OWH    ACCEL\_REG\_BASEADDR + 45\*4
8. #define ACCEL\_REG\_POOL\_SHAPE\_OCSTEP ACCEL\_REG\_BASEADDR + 46\*4
9. #define ACCEL\_REG\_POOL\_IFM\_ADDR     ACCEL\_REG\_BASEADDR + 47\*4
10. #define ACCEL\_REG\_POOL\_OFM\_ADDR     ACCEL\_REG\_BASEADDR + 48\*4
11. #define ACCEL\_REG\_POOL\_PAD\_VALUE    ACCEL\_REG\_BASEADDR + 49\*4

下面介绍池化算子的寄存器配置宏定义，在**hw/accel\_params.h**文件中由下面十个宏定义构成，完成对上述十个寄存器的配置，其中函数Xil\_Out32(ACCEL\_REG\_POOL\_CTRL, x)的功能为往指定地址的寄存器中写入指定的数据，这个函数是Xilinx的库函数，平台移植需要进行替换；

1. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*POOL\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
2. #define POOL\_RESET                Xil\_Out32(ACCEL\_REG\_POOL\_CTRL, 0)
3. #define POOL\_CTRL\_SET(en,op,kernel\_w,kernel\_h,stride\_w,stride\_h,src\_sel,pad\_bottom,pad\_top,pad\_right,pad\_left,pad\_mode) \
4. Xil\_Out32(ACCEL\_REG\_POOL\_CTRL, en | op<<1 | kernel\_w<<3 | kernel\_h<<5 | stride\_w<<7 | stride\_h<<9 | src\_sel<<11 | pad\_bottom<<16 | pad\_top<<18 | pad\_right<<20 | pad\_left<<22 | pad\_mode<<24)
5. #define POOL\_IFM\_C\_SET(c)         Xil\_Out32(ACCEL\_REG\_POOL\_SHAPE\_IC, c)
6. #define POOL\_IFM\_WH\_SET(w,h)      Xil\_Out32(ACCEL\_REG\_POOL\_SHAPE\_IWH, w<<16 | h)
7. #define POOL\_IFM\_CSTEP\_SET(x)     Xil\_Out32(ACCEL\_REG\_POOL\_SHAPE\_ICSTEP, x)
8. #define POOL\_OFM\_C\_SET(c)         Xil\_Out32(ACCEL\_REG\_POOL\_SHAPE\_OC, c)
9. #define POOL\_OFM\_WH\_SET(w,h)      Xil\_Out32(ACCEL\_REG\_POOL\_SHAPE\_OWH, w<<16 | h)
10. #define POOL\_OFM\_CSTEP\_SET(x)     Xil\_Out32(ACCEL\_REG\_POOL\_SHAPE\_OCSTEP, x)
11. #define POOL\_IFM\_ADDR\_SET(x)      Xil\_Out32(ACCEL\_REG\_POOL\_IFM\_ADDR, x)
12. #define POOL\_OFM\_ADDR\_SET(x)      Xil\_Out32(ACCEL\_REG\_POOL\_OFM\_ADDR, x)
13. #define POOL\_PAD\_VALUE\_SET(x)     Xil\_Out32(ACCEL\_REG\_POOL\_PAD\_VALUE, x)

在**hw/hw\_pool.cpp**中函数forward(ncnn::Mat &ifm, ncnn::Mat &ofm)就对上述的十个宏进行调用，代码解析详见注释。不同的case情况会传入相应的值完成寄存器的配置，对于单个池化算子测试的顶层case可以详见**testLayer/test\_layer\_pool.cpp**文件中的函数void test\_layer\_pool\_batch()。

1. **void** hw\_pool::forward(ncnn::Mat &ifm, ncnn::Mat &ofm) {
2. POOL\_RESET;   //寄存器复位
3. Xil\_DCacheFlush();  //刷cache
4. **int**\* src0\_addr0 = ifm;
5. **int**\* dst\_addr0 = ofm;
6. POOL\_IFM\_ADDR\_SET((uint64\_t)src0\_addr0);  //配置输入特征图基地址
7. POOL\_OFM\_ADDR\_SET((uint64\_t)dst\_addr0);  //配置输出特征图基地址
8. POOL\_IFM\_WH\_SET(ifm.w,ifm.h);  //配置输入特征图的w和h
9. POOL\_IFM\_C\_SET(ifm.c);  //配置输入特征图的c
10. POOL\_IFM\_CSTEP\_SET(ifm.cstep);  //配置输入特征图的cstep
11. POOL\_OFM\_WH\_SET(ofm.w,ofm.h);  //配置输出特征图的w和h
12. POOL\_OFM\_C\_SET(ofm.c);  //配置输出特征图的c
13. POOL\_OFM\_CSTEP\_SET(ofm.cstep);  //配置输出特征图的cstep
14. POOL\_PAD\_VALUE\_SET(\*((uint32\_t\*)&pad\_value)); //配置padding的常数值
15. POOL\_CTRL\_SET(1, pool\_type,kernel\_w,kernel\_h,stride\_w,stride\_h,1,pad\_bottom,pad\_top,pad\_right,pad\_left,pad\_mode); //配置池化的使能、类型、kernel的wh，stride的wh，padding的值、padding的模式等
16. wait\_pool\_done();//等待池化计算完成
17. Xil\_DCacheInvalidate();  //刷cache
18. POOL\_RESET;  //池化寄存器复位
19. }

其他算子的寄存器配置过程与池化算子基本相同，这里也不再详细赘述。