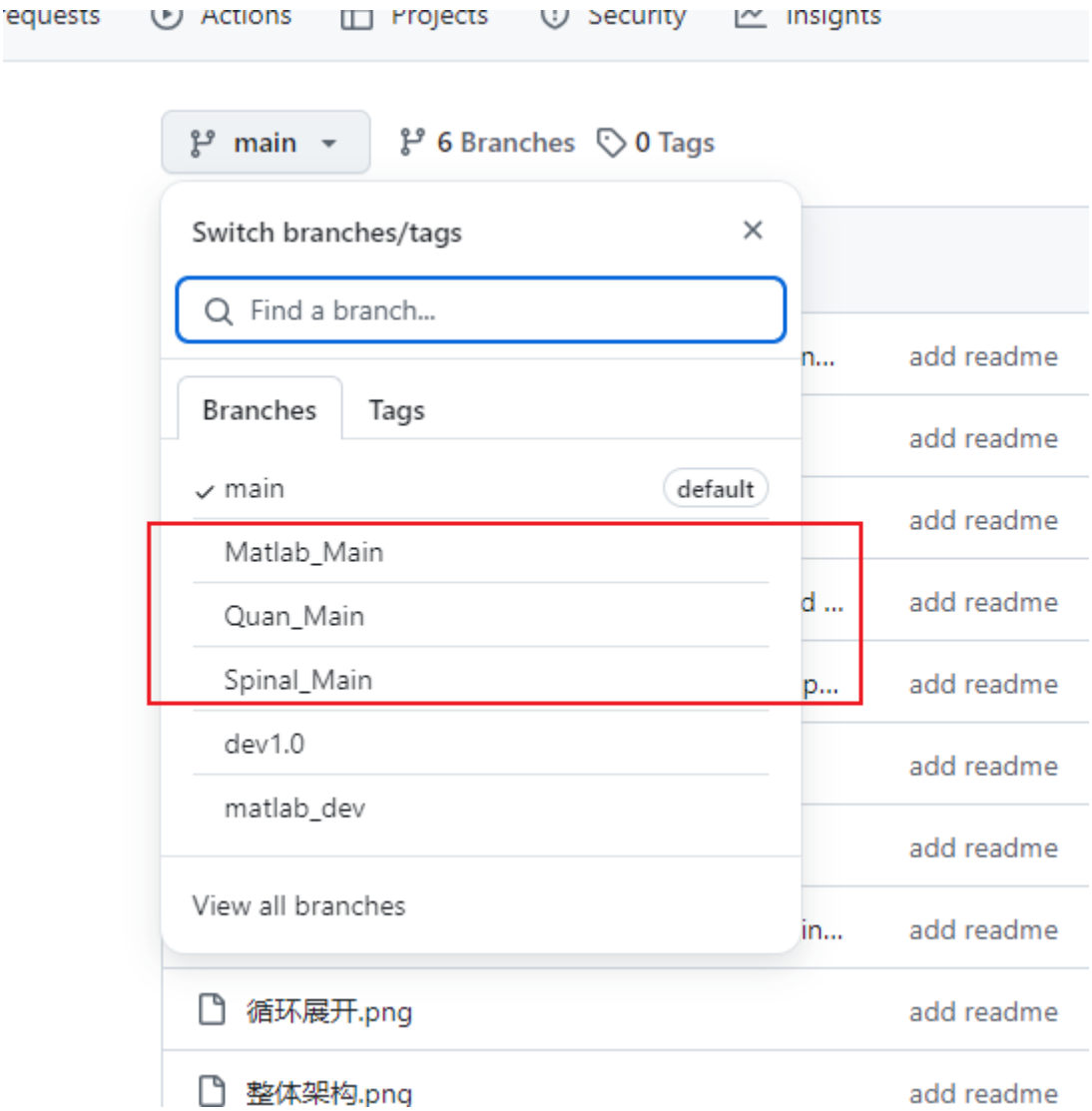
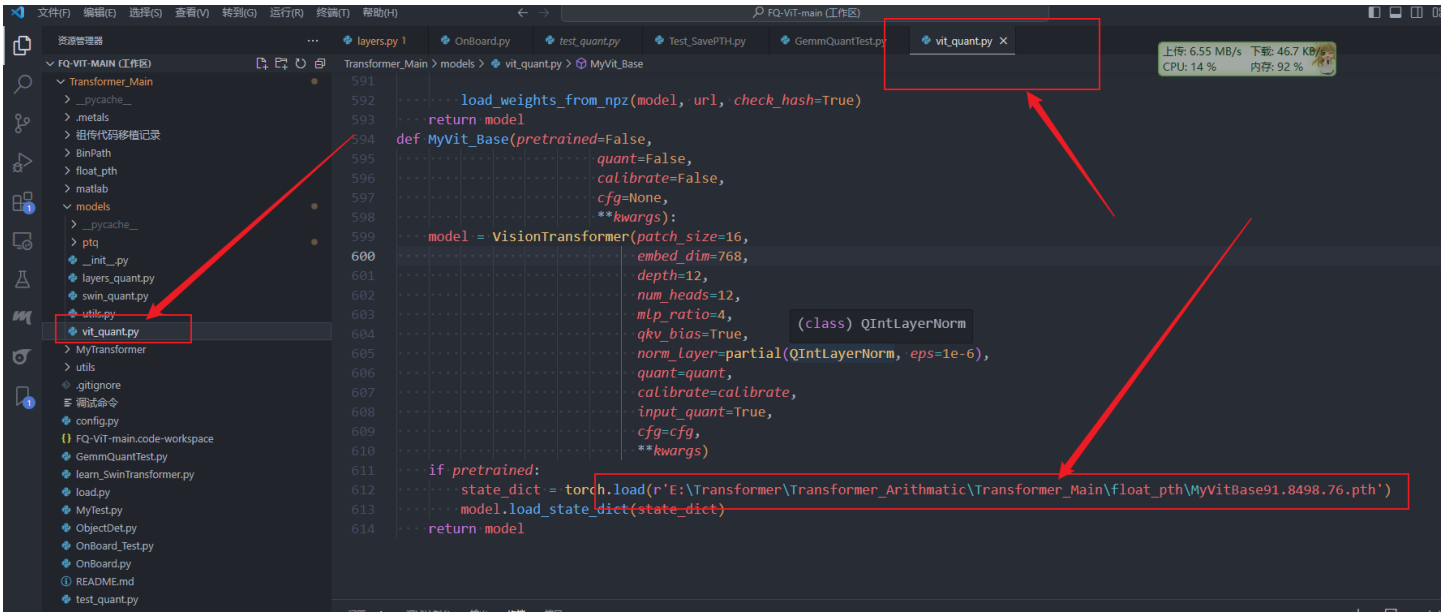
# 上板测试流程（仿真流程也差不多)

git仓库：https://github.com/Simon-K1/DETR/tree/main(代尧+翟锐凯）

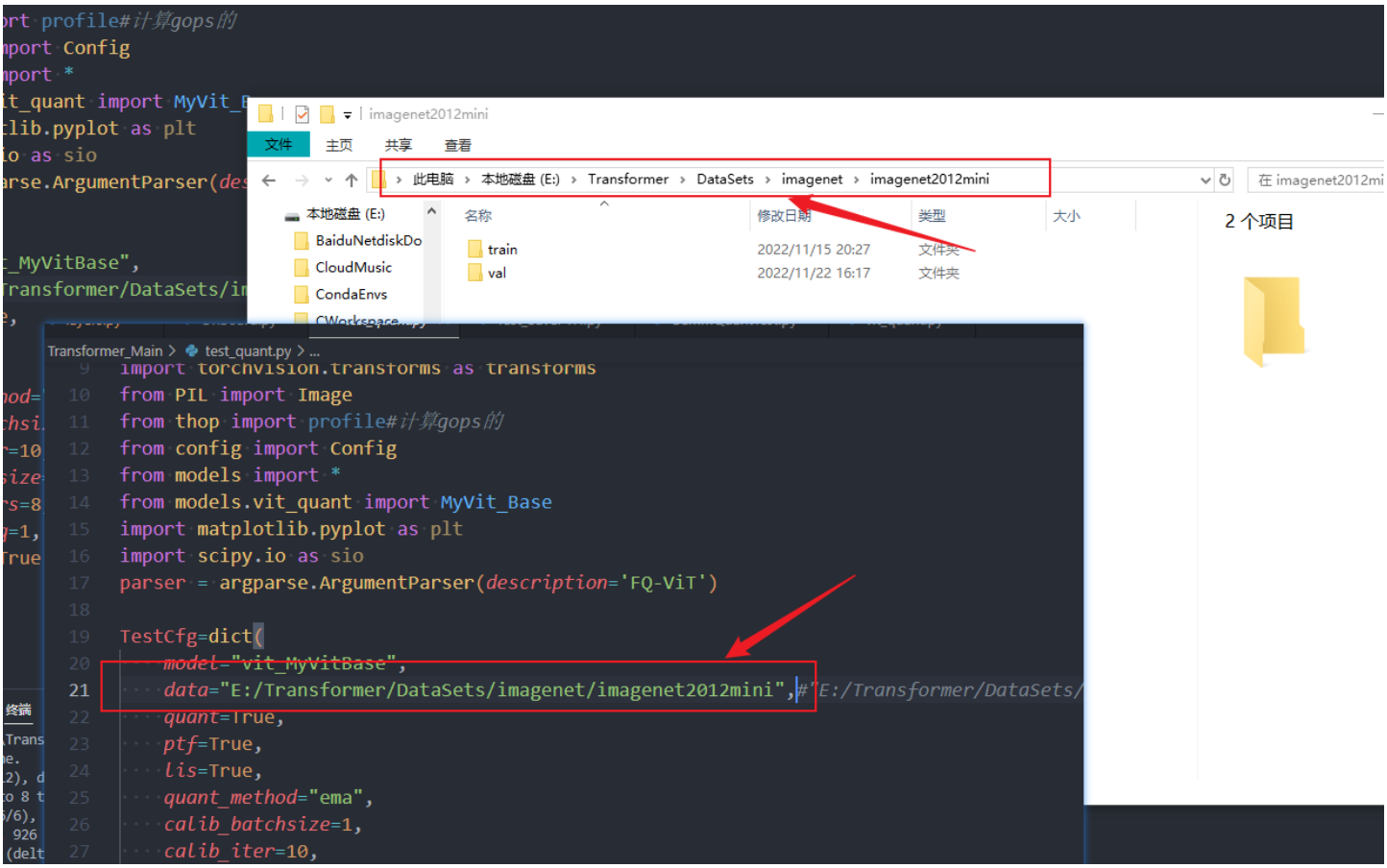
需要Clone下来的分支分别为Spinal_Main,Quan_Main和Matlab_Main




(创建三个文件夹，分别把这三个分支Clone下来，最新代码都是能跑的，Quan_Main跑不通是因为需要数据集和训练好的模型来跑，数据集和模型放在下面链接，这里面的模型是我自己随便训的，只能分类三个类，需要自己训练就需要跑train.py的代码）



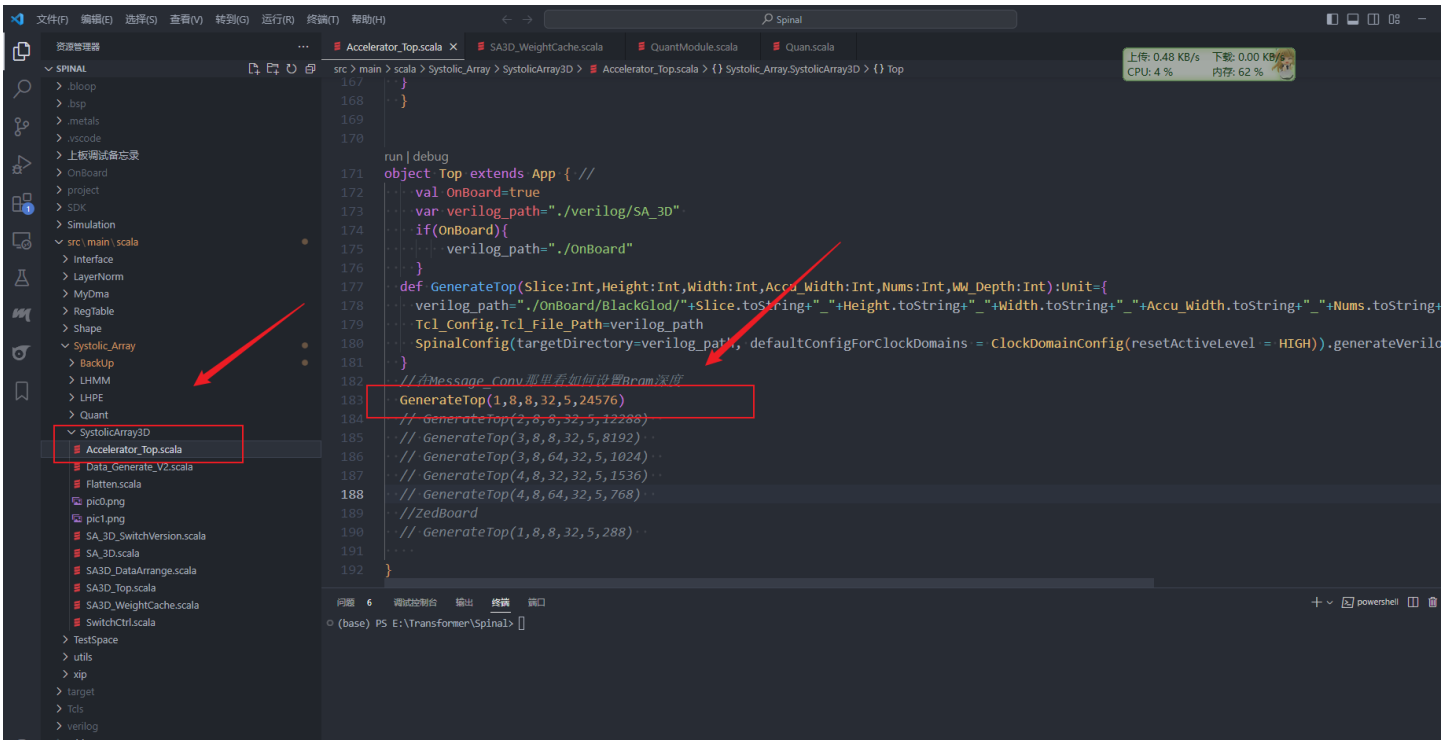

数据集：解压即用，在test_quant的cfg中改一下路径就行




# 第一步：生成RTL

分支：Spinal_V3--目前进度：卷积量化和矩阵量化均实现了，剩下的就是无穷无尽的测试和调试了。。。

SpinalHDL现在可以一键生成各种配置的加速器，现在3*8*64的上面验证通过了卷积+完整量化，1*8*8上面验证通过了矩阵计算+完整量化。



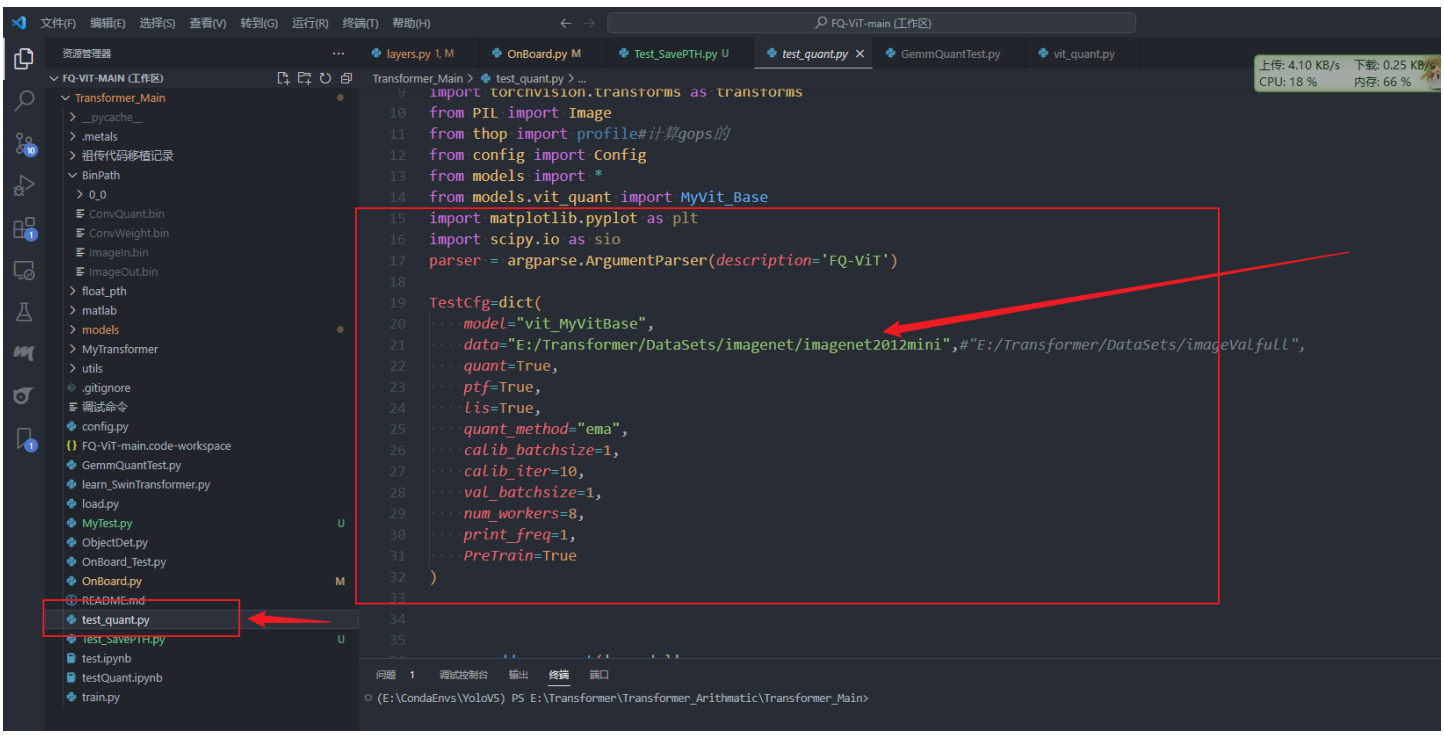

[Tile，Height，Width，32(累加和位宽，不要改），5（不要改），权重缓存Bram的深度(位宽为64bit)]

拿到RTL代码后自己搭一下BD，导入TCL（这部分待更新）

# 第二步：Pytorch生成真实的权重，图片和量化参数

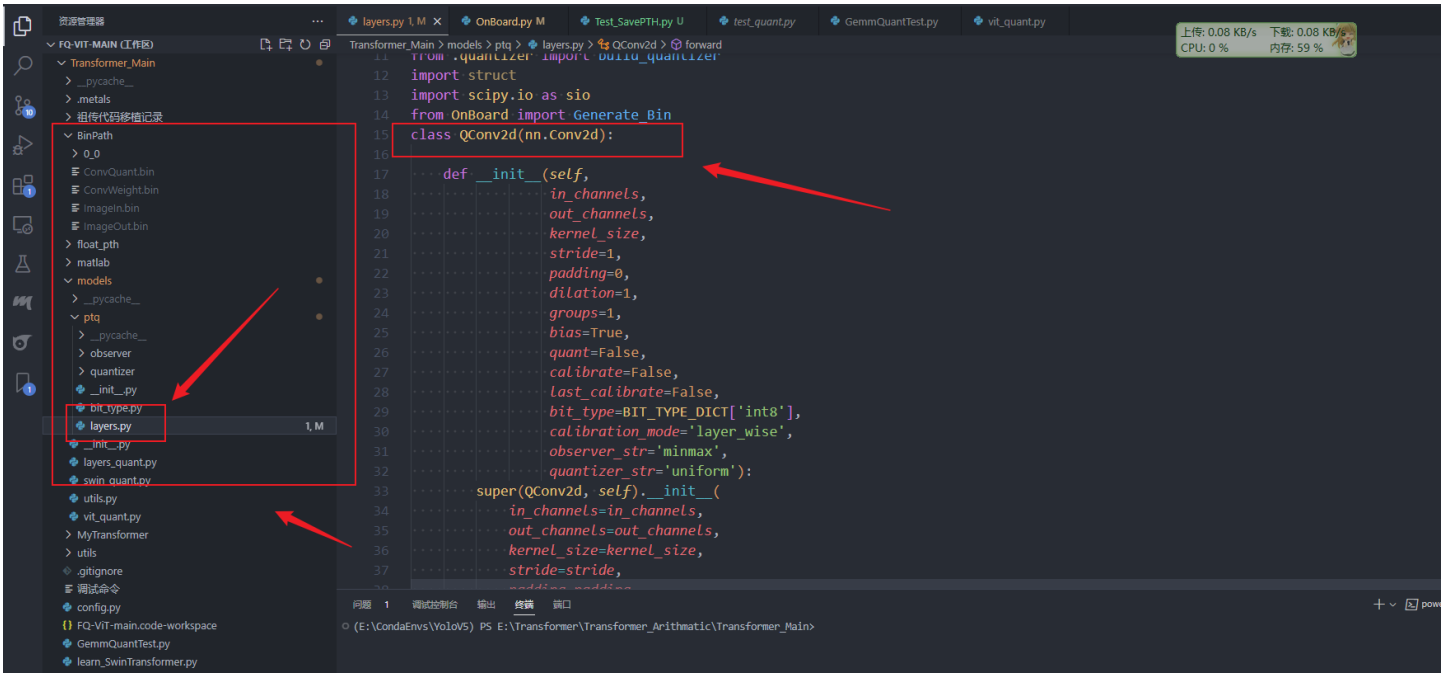
分支：Quan_v6

在test_Quant.py中配置好图片路径等参数

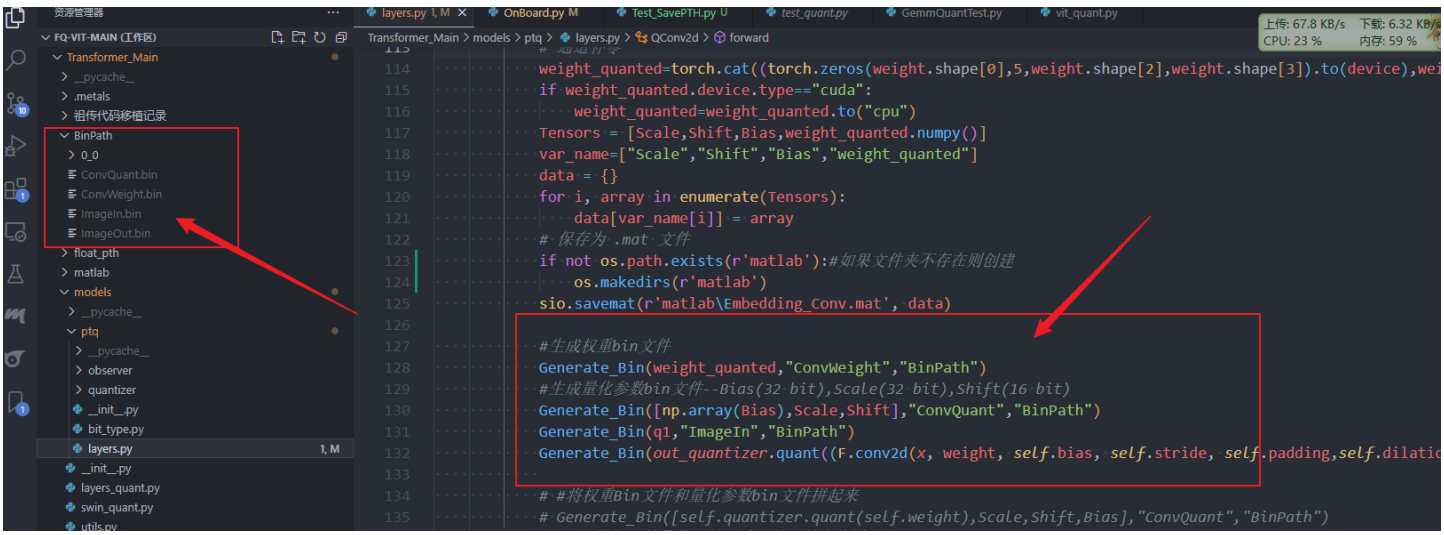


然后找到QConv2这个类，断点调试打到这里，祖传量化已经被移植到这里了，

校准完生成Scale，Zeropoint后，程序运行到这里会自动生成权重，图片和量化参数和输出图片共四个bin文件。




下面这四行代码用于生成bin文件，生成完bin文件后就可以退出调试

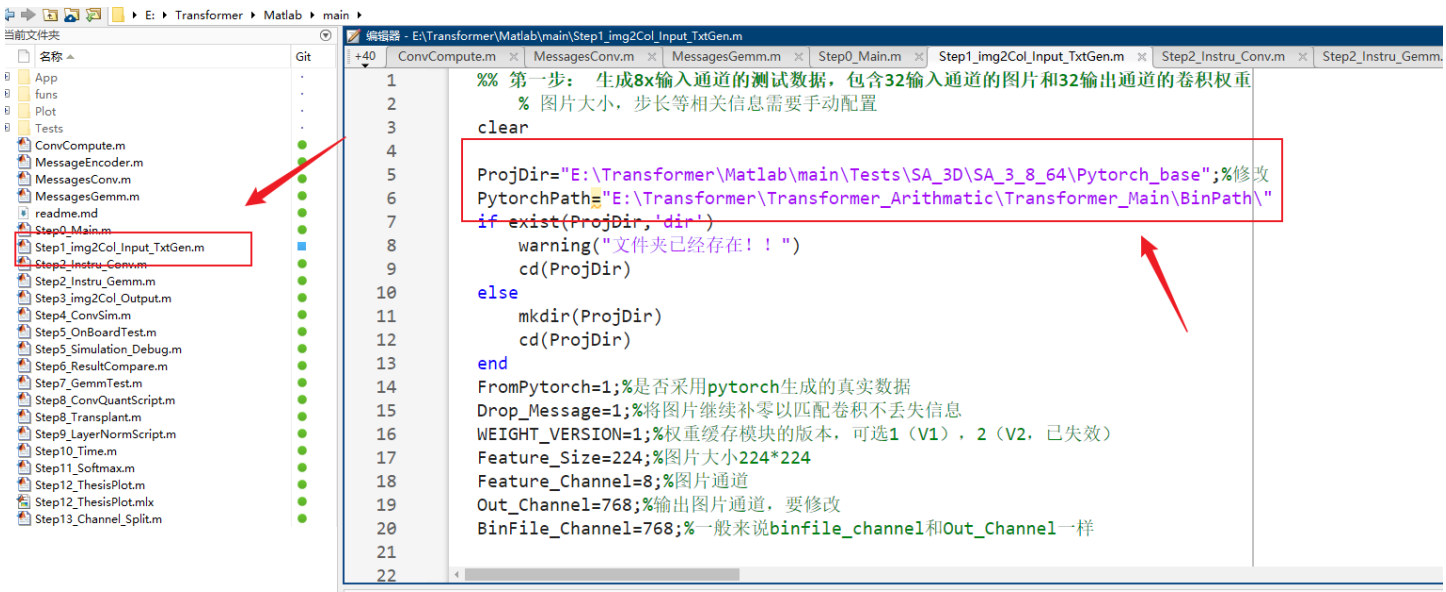



# 第三步：打开Matlab，用Matlab读取Pytorch生成的真实数据并生成指令

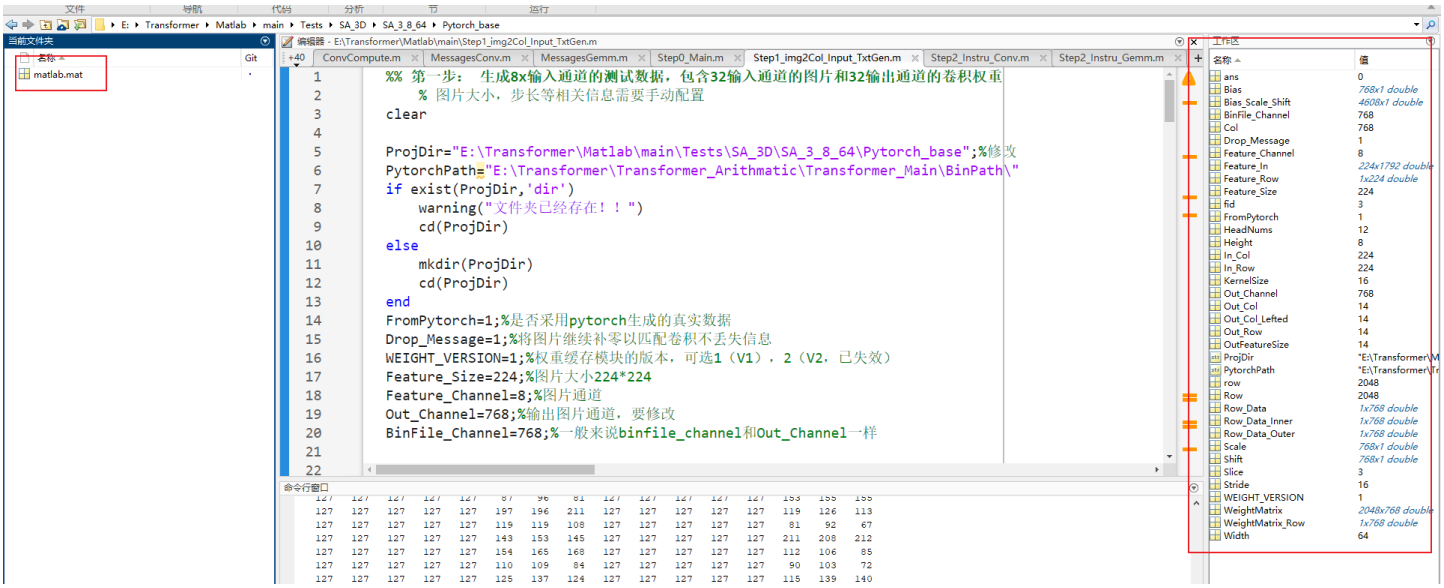
如果要上板测的话，就几步而已

# 进入matlab的Step1，

- 配置好自己的ProjDir，以及Pytorch生成的bin文件目录
- FromPytorch选择1
- Drop_Message选择1，不用管他，用来padding的，现在还没实现padding，不理他
- WEIGHT_VERSIOn:默认1，因为权重有两种数据排列格式：行优先和列优先，现在我们用的全是列优先
- Feature_Size;与Pytorch对应，Pytorch是224的图片
- Feature_Channel:图片通道，默认8通道，Pytorch那虽然图片是3通道，但是我在生成bin文件的时候补成了8通道
- Out_Channel：这个得和Vit-Base对应，768输出通道
- BinFile_Channel：Pytorch生成的BinFile中卷积是多少通道就填多少通道。
- 剩下的待补充



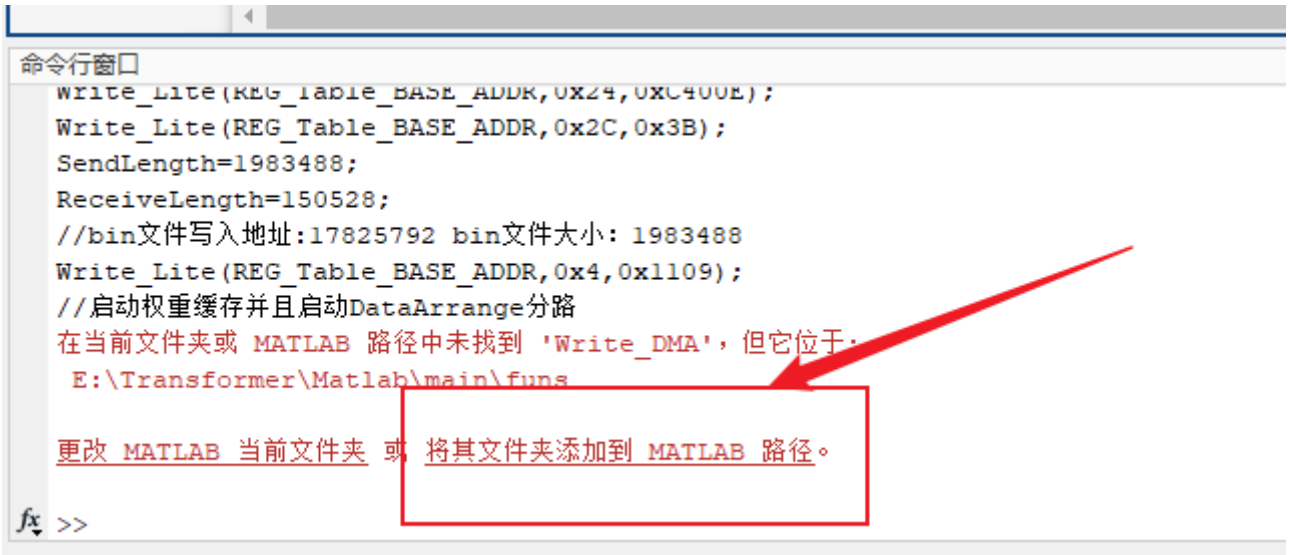

全部配好后直接运行Step1，自动创建包含了真实数据的全部变量，保存在matlab.mat文件中，以后只需要读这个matlab.mat文件即可



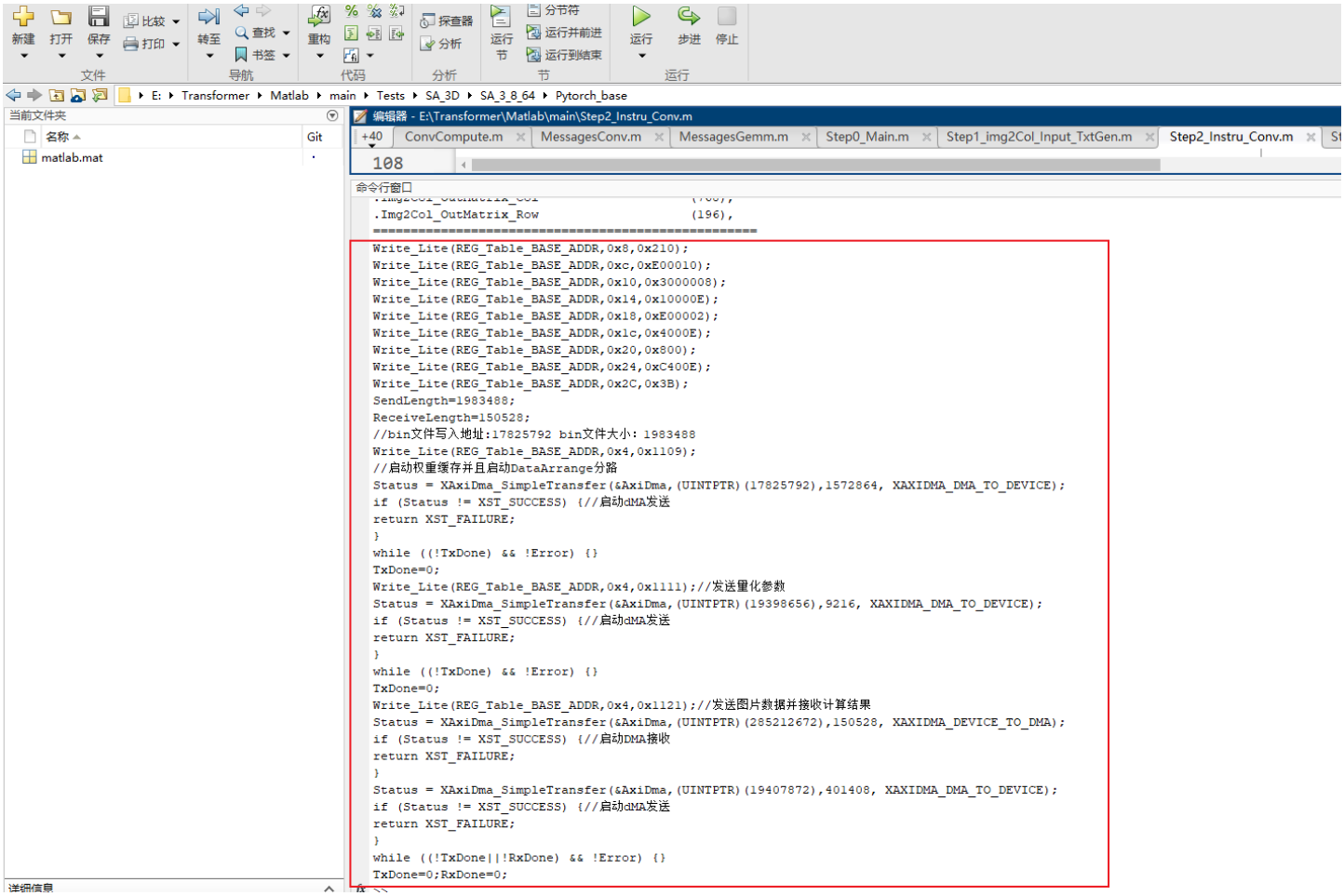

# 进入Step2_Instru_Conv

Step2用来生成指令，应该没啥问题，测了好多遍了

如果报这个错，是因为funs这个文件夹没有被添加到搜索路径，将其添加一下就好再重新运行即可



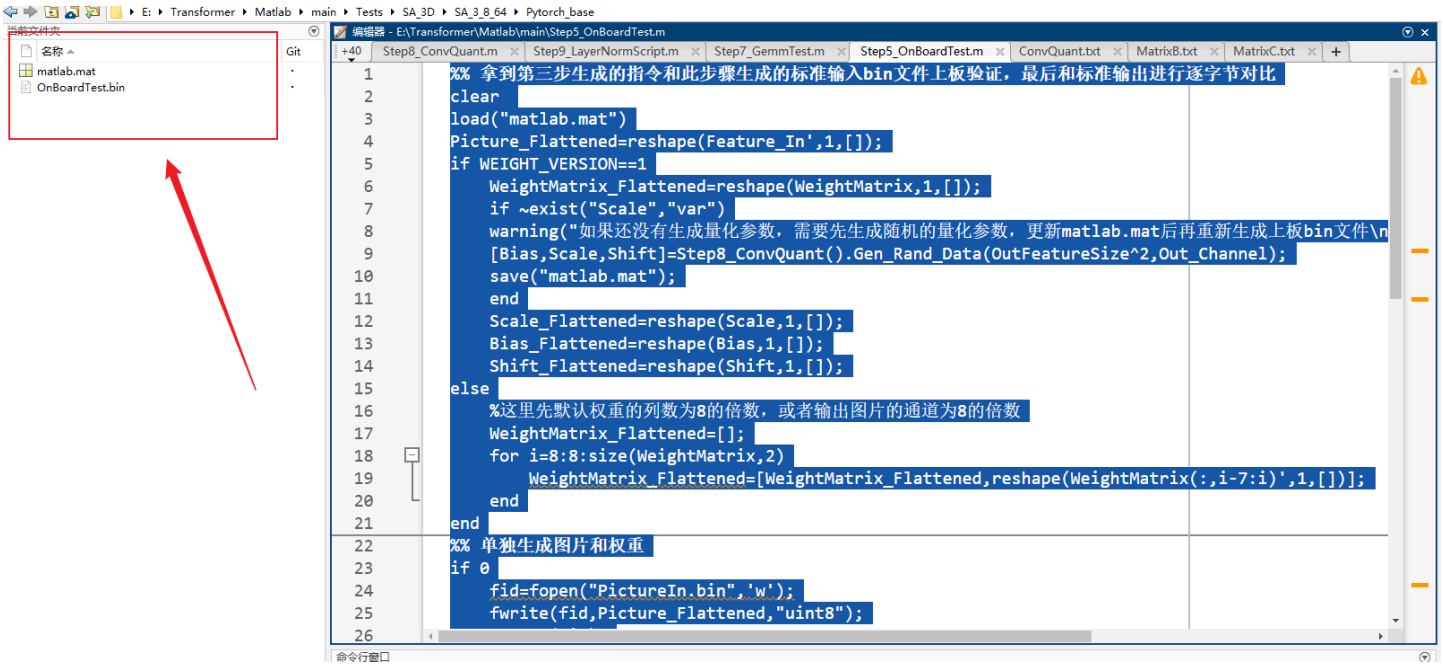

跑完后生成上板测试指令和仿真指令，到时候复制粘贴就行



```
Write_Lite(REG_Table_BASE_ADDR,0x8,0x210);
Write_Lite(REG_Table_BASE_ADDR,0xc,0xE00010);
Write_Lite(REG_Table_BASE_ADDR,0x10,0x3000008);
Write_Lite(REG_Table_BASE_ADDR,0x14,0x10000E);
Write_Lite(REG_Table_BASE_ADDR,0x18,0xE00002);
Write_Lite(REG_Table_BASE_ADDR,0x1c,0x4000E);
Write_Lite(REG_Table_BASE_ADDR,0x20,0x800);
Write_Lite(REG_Table_BASE_ADDR,0x24,0xC400E);
Write_Lite(REG_Table_BASE_ADDR,0x2C,0x3B);
SendLength=1983488;
ReceiveLength=150528;
//bin文件写入地址:17825792 bin文件大小: 1983488
Write_Lite(REG_Table_BASE_ADDR,0x4,0x1109);
//启动权重缓存并且启动DataArrange分路
Status = XAxiDma_SimpleTransfer(&AxiDma,(UINTPTR)(17825792),1572864, XAXIDMA_DMA_TO_DEVICE);
if (Status != XST_SUCCESS) {//启动dMA发送
return XST_FAILURE;
}
while ((!TxDone) && !Error) {}
TxDone=0;
Write_Lite(REG_Table_BASE_ADDR,0x4,0x1111);//发送量化参数
Status = XAxiDma_SimpleTransfer(&AxiDma,(UINTPTR)(19398656),9216, XAXIDMA_DMA_TO_DEVICE);
if (Status != XST_SUCCESS) {//启动dMA发送
return XST_FAILURE;
}
while ((!TxDone) && !Error) {}
TxDone=0;
Write_Lite(REG_Table_BASE_ADDR,0x4,0x1121);//发送图片数据并接收计算结果
Status = XAxiDma_SimpleTransfer(&AxiDma,(UINTPTR)(285212672),150528, XAXIDMA_DEVICE_TO_DMA);
if (Status != XST_SUCCESS) {//启动DMA接收
return XST_FAILURE;
}
Status = XAxiDma_SimpleTransfer(&AxiDma,(UINTPTR)(19407872),401408, XAXIDMA_DMA_TO_DEVICE);
if (Status != XST_SUCCESS) {//启动dMA发送
return XST_FAILURE;
}
while ((!TxDone||!RxDone) && !Error) {}
TxDone=0;RxDone=0;
```

# 进入Step5-OnBoadeTest

运行即可一键生成上板测试bin文件，然后将其导入到Zynq的ddr里即可

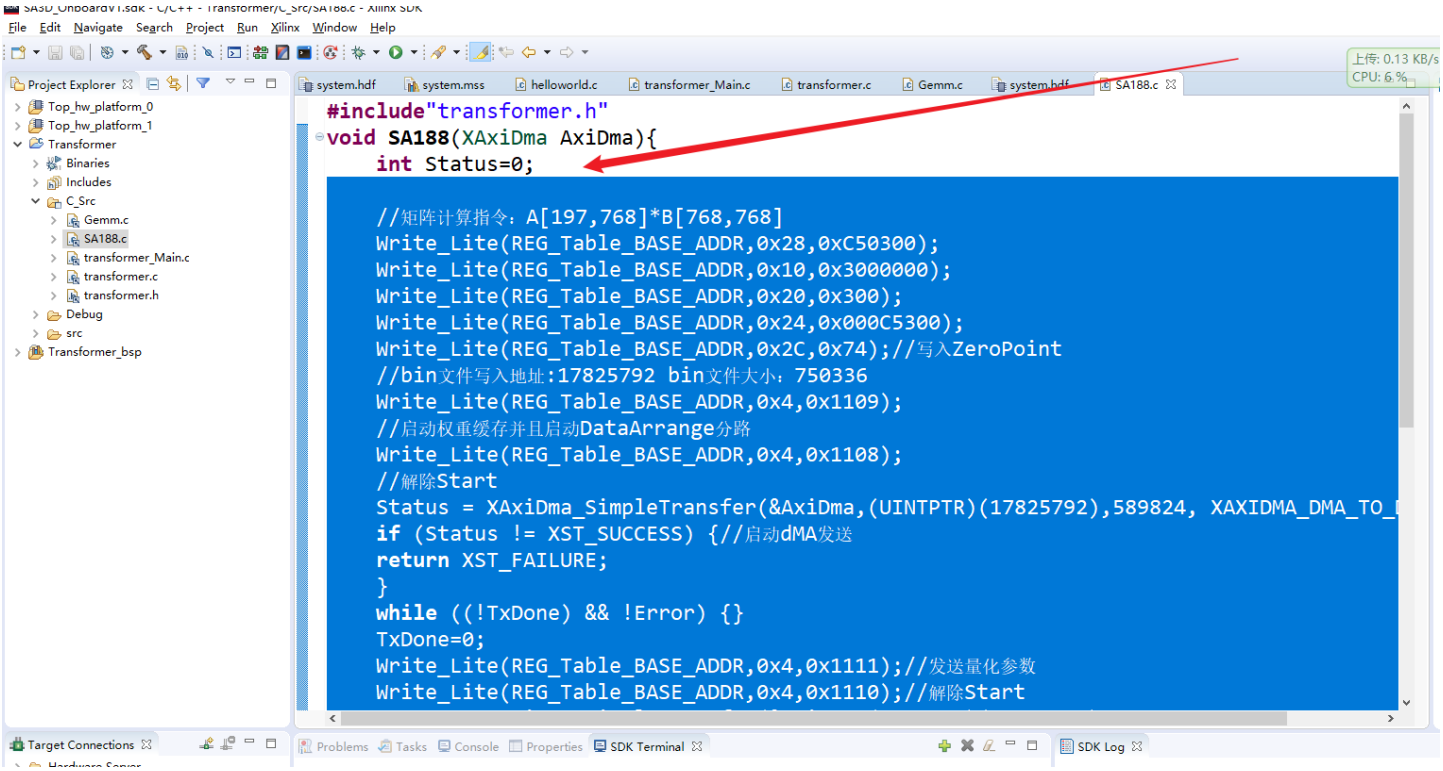


```
%% 拿到第三步生成的指令和此步骤生成的标准输入bin文件上板验证，最后和标准输出进行逐字节对比
clear
load("matlab.mat")
Picture_Flattened=reshape(Feature_In',1,[]);
if WEIGHT_VERSION==1
    WeightMatrix_Flattened=reshape(WeightMatrix,1,[]);
    if ~exist("Scale","var")
    warning("如果还没有生成量化参数，需要先生成随机的量化参数，更新matlab.mat后再重新生成上板bin文件\n
    [Bias,Scale,Shift]=Step8_ConvQuant().Gen_Rand_Data(OutFeatureSize^2,Out_Channel);
    save("matlab.mat");
    end
    Scale_Flattened=reshape(Scale,1,[]);
    Bias_Flattened=reshape(Bias,1,[]);
    Shift_Flattened=reshape(Shift,1,[]);
else
    %这里先默认权重的列数为8的倍数，或者输出图片的通道为8的倍数
    WeightMatrix_Flattened=[];
    for i=8:8:size(WeightMatrix,2)
        WeightMatrix_Flattened=[WeightMatrix_Flattened,reshape(WeightMatrix(:,i-7:i)',1,[])];
    end
end
%% 单独生成图片和权重
if 0
    fid=fopen("PictureIn.bin",'w');
    fwrite(fid,Picture_Flattened,"uint8");
```

# 第四步：打开SDK

SDK那边就4文件，需要修改的文件就是SA188.c或者Gemm.c（这种文件可以自己创建多个，再在transforme_main中调用即可。

把刚刚Matlab生成的C代码复制到SA188.c中（SA188.c只是一个源文件，可以自己创建任意名称的文件并include"transformer.h"），然后再在transformer_Main中调用即可。




SA188.c

transformer_Main.c

transformer.h

transformer.c

# 第五步：写入Bin文件

写入生成好的OnBoardTest.bin文件，然后跑一把看看结果。

**Bin文件的写入数据量和写入地址在刚刚复制的C代码中已经标出来了**

```c
#include"transformer.h"
void SA188(XAxiDma AxiDma){
    int Status=0;

    //矩阵计算指令：A[197,768]*B[768,768]
    Write_Lite(REG_Table_BASE_ADDR,0x28,0xC50300);
    Write_Lite(REG_Table_BASE_ADDR,0x10,0x3000000);
    Write_Lite(REG_Table_BASE_ADDR,0x20,0x300);
    Write_Lite(REG_Table_BASE_ADDR,0x24,0x000C5300);
    Write_Lite(REG_Table_BASE_ADDR,0x2C,0x74);//写入ZeroPoint
    //bin文件写入地址:17825792 bin文件大小：750336
    Write_Lite(REG_Table_BASE_ADDR,0x4,0x1109);
    //启动权重缓存并且启动DataArrange分路
    Write_Lite(REG_Table_BASE_ADDR,0x4,0x1108);
```

# 第六步：FPGA算完后导出bin文件

从哪导出导出多少数据量，具体需要查看最后的DMA接收的那行代码中的数据：

```c
    return XST_FAILURE;
    }
    while ((!TxDone) && !Error) {}
    TxDone=0;
    Write_Lite(REG_Table_BASE_ADDR,0x4,0x1181);//发送图片数据并接收计算结果
    Write_Lite(REG_Table_BASE_ADDR,0x4,0x1180);//解除Start
    Status = XAxiDma_SimpleTransfer(&AxiDma,(UINTPTR)(285212672),151296, XAXIDMA_DEVICE_
    if (Status != XST_SUCCESS) {//启动DMA接收
    return XST_FAILURE;
```

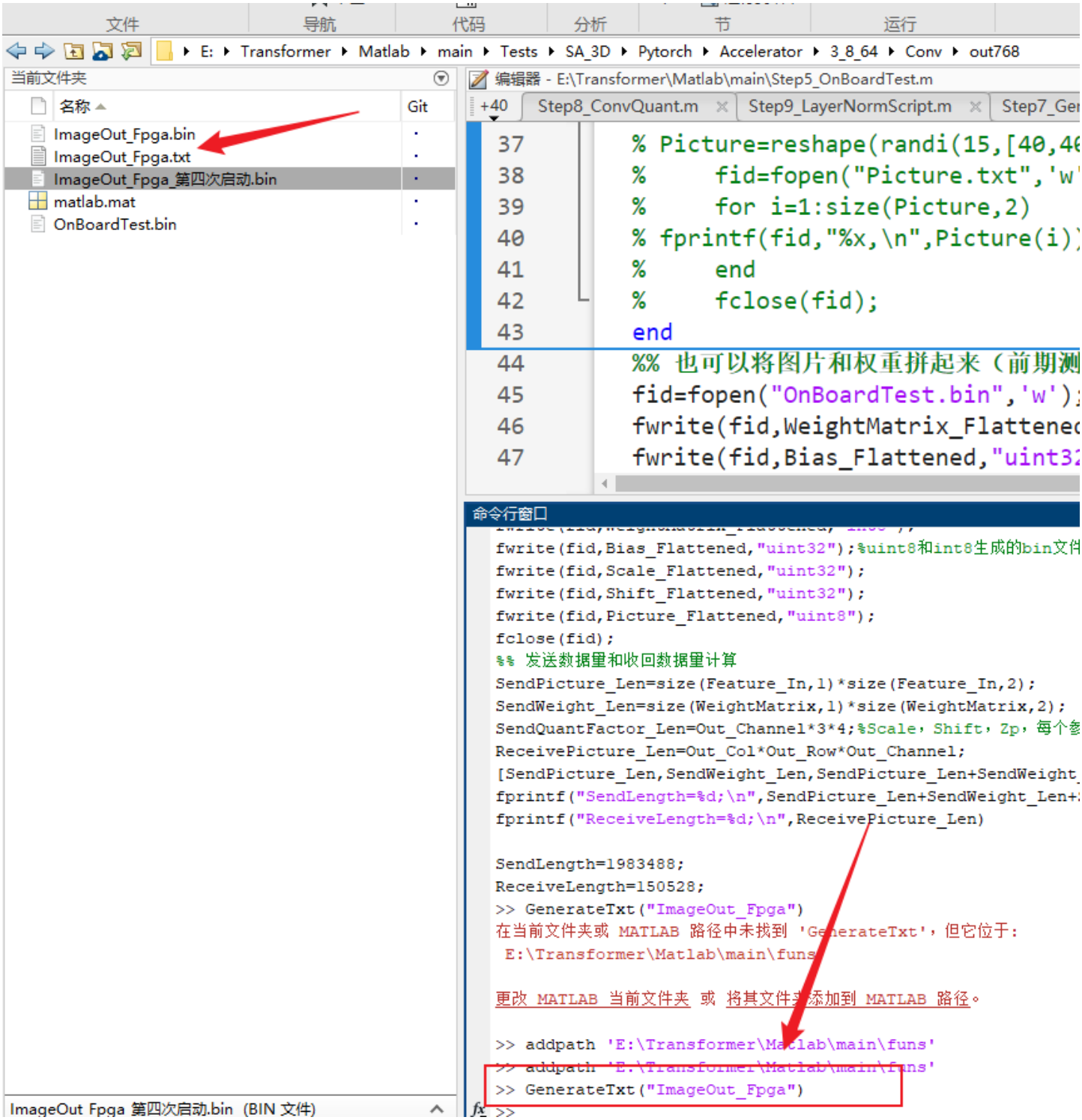
接收地址

接收数据量

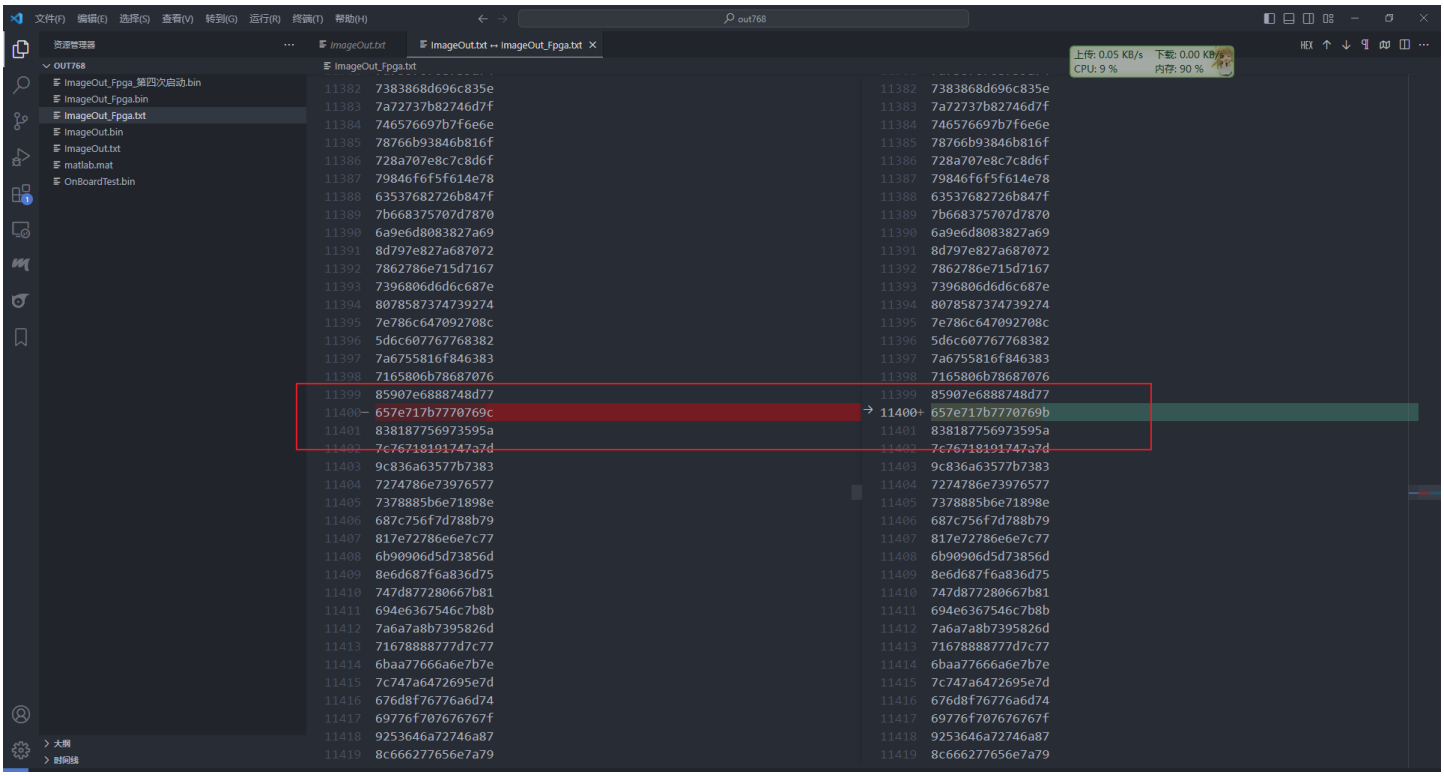
# 第七步：拿导出的bin文件和Pytorch的输出进行逐一字节的对比

很多方法，这里可以用matlab生成txt然后对比：

命令行中输入GenerateTxt("ImageOut_Fpga")即可自动生成对应bin文件的txt文件




同样地，将Pytorch生成的输出图片的bin文件也转成txt然后用vscode对比即可：


文件(F) 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)
out768
资源管理器
OUT768
ImageOut_Fpga_第四次启动.bin
ImageOut_Fpga.bin
ImageOut_Fpga.txt
ImageOut.bin
ImageOut.txt
matlab.mat
OnBoardTest.bin
ImageOut.txt
ImageOut.txt ↔ ImageOut_Fpga.txt
ImageOut_Fpga.txt
上传: 0.05 KB/s 下载: 0.00 KB/s
CPU: 9 % 内存: 90 %
HEX
11382 7383868d696c835e
11383 7a72737b82746d7f
11384 746576697b7f6e6e
11385 78766b93846b816f
11386 728a707e8c7c8d6f
11387 79846f6f5f614e78
11388 63537682726b847f
11389 7b668375707d7870
11390 6a9e6d8083827a69
11391 8d797e827a687072
11392 7862786e715d7167
11393 7396806d6d6c687e
11394 8078587374739274
11395 7e786c647092708c
11396 5d6c607767768382
11397 7a6755816f846383
11398 7165806b78687076
11399 85907e6888748d77
11400- 657e717b7770769c
11401 838187756973595a
11402 7c76718191747a7d
11403 9c836a63577b7383
11404 7274786e73976577
11405 7378885b6e71898e
11406 687c756f7d788b79
11407 817e72786e6e7c77
11408 6b90906d5d73856d
11409 8e6d687f6a836d75
11410 747d877280667b81
11411 694e6367546c7b8b
11412 7a6a7a8b7395826d
11413 71678888777d7c77
11414 6baa77666a6e7b7e
11415 7c747a6472695e7d
11416 676d8f76776a6d74
11417 69776f707676767f
11418 9253646a72746a87
11419 8c666277656e7a79
11400+ 657e717b7770769b
大纲
时间线

最后就1bit没对上。