# 基于pynq-z2的面部表情识别

**段卓文：51265902047 工作占比：50%**

**周梓豪：51265902147 工作占比：50%**

目录

[基于pynq-z2的面部表情识别 1](#_Toc157273838)

[一、任务概述 2](#_Toc157273839)

[(1)背景 2](#_Toc157273840)

[(2)任务内容 2](#_Toc157273841)

[二、操作步骤 2](#_Toc157273842)

[(1)PYNQ-Z2开发板准备 2](#_Toc157273843)

[(2)训练模型 3](#_Toc157273844)

[(3)ip核设计 4](#_Toc157273845)

[(4)PS端调用检测 12](#_Toc157273846)

[三、项目运行展示 13](#_Toc157273847)

[四、项目总结 14](#_Toc157273848)

## 一、任务概述

### (1)背景

神经网络在全球多个领域的研究中展现出了巨大的优势，在图像、视频和语言处理领域中提出了各式各样的网络模型，神经网络优秀的特征提取能力和出色的拟合能力使其在许多领域中胜过传统的方法。但是，神经网络存在计算复杂、参数量巨大的缺点，使其依赖于高性能的GPU进行工作，Pytorch、Tensorflow等神经网络框架也在GPU上提供了方便使用的接口，除了CPU和GPU之外，FPGA也是高能效神经网络处理的平台之一，为专用模型设计特定的硬件加速器，实现高度并行化的神经网络加速器也受到广泛关注，FPGA的优势在于可以使用丰富的片上内存减少数据的读取时间，可以在不降低模型准确性的情况下以硬件友好的方式简化模型，满足一些在低功耗低资源场景下使用边缘式设备部署神经网络模型的需求。

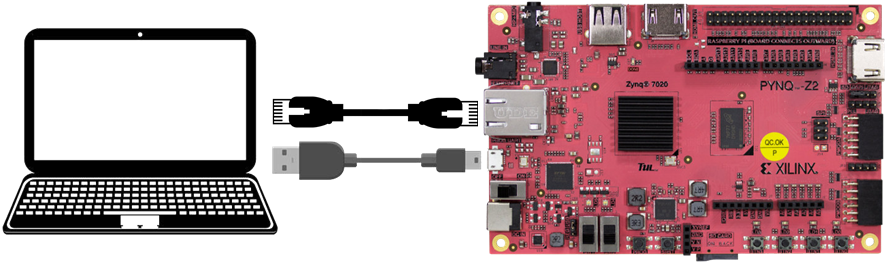
### (2)任务内容

任务目标是在PYNQ-Z2上实现表情图片分类识别的神经网络模型加速器，任务主要分为三大部分内容，第一部分是在GPU上通过Tensorflow神经网络框架训练表情识别模型并导出h5模型文件，将训练好的h5模型中的网络层转化成对应的二进制参数文件，第二部分是设计卷积加速器和池化加速器的的C++代码，并通过HLS将C++代码转换成Verilog的电路描述，随后Export RTL导出ip核，接着在vivado工程block desigin添加需要的组件和导出的ip核之后生成bit和hwh文件，第三部分是将生成的bit和hwh文件放入sd卡，在PS端的jupyter notebook中使用Overlay连接实际的电路，然后使用python脚本读取图片调用PL端进行检测。

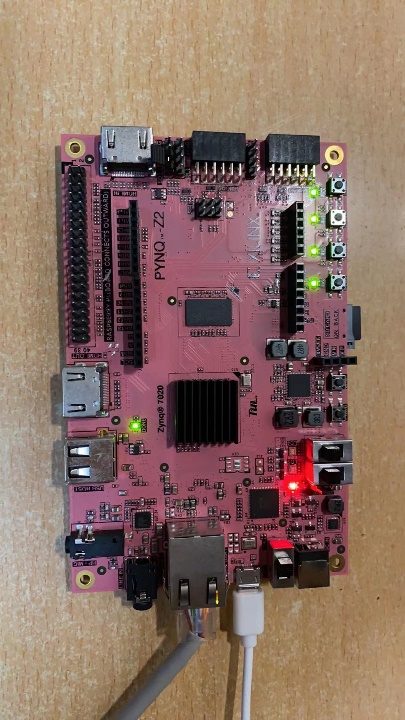
## 二、操作步骤

### (1)PYNQ-Z2开发板准备

下载官方镜像，使用Win32 Disk Imager将下载好的镜像烧录到sd卡中，将两个跳线帽分别置于USB和SD处，表示选择USB供电并使用sd卡中的镜像启动板卡，使用micro USB线将PC和PYNQ的串口连接起来，然后使用以太网线将PYNQ和PC连接到同一个局域网下，同一局域网下是为了之后可以在本机访问开发板PS端的jupyter notebook和文件。连接图如下所示：



连接完成后，将电源开关拨到“ON”位置，启动开发板，上电成功，红灯亮，随后蓝灯和黄灯闪烁，最后只剩下黄灯亮，表示开发板工作正常，如下图所示：



### (2)训练模型

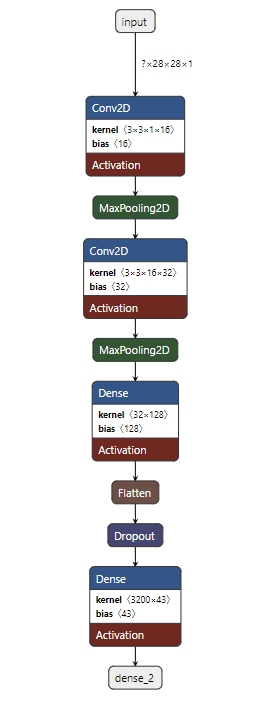
数据集：JAFFE数据集，该数据库是由10位日本女性在实验环境下根据指示做出各种表情，再由照相机拍摄获取的人脸表情图像。

种类：7种表情分别是sad、happy、angry、disgust、surprise、fear、neutral

训练集：随机划分总数据的90%作为训练集

测试集：随机划分总数据的10%作为训练集

模型结构（使用netron工具查看）：

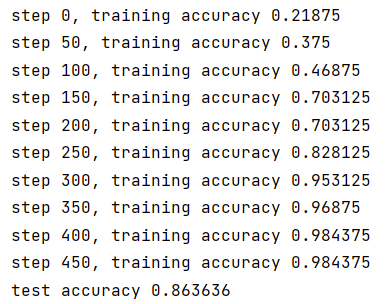


输入图片：28\*28\*1

使用Tensorflow框架训练了500个epoch

模型大小：2.55MB

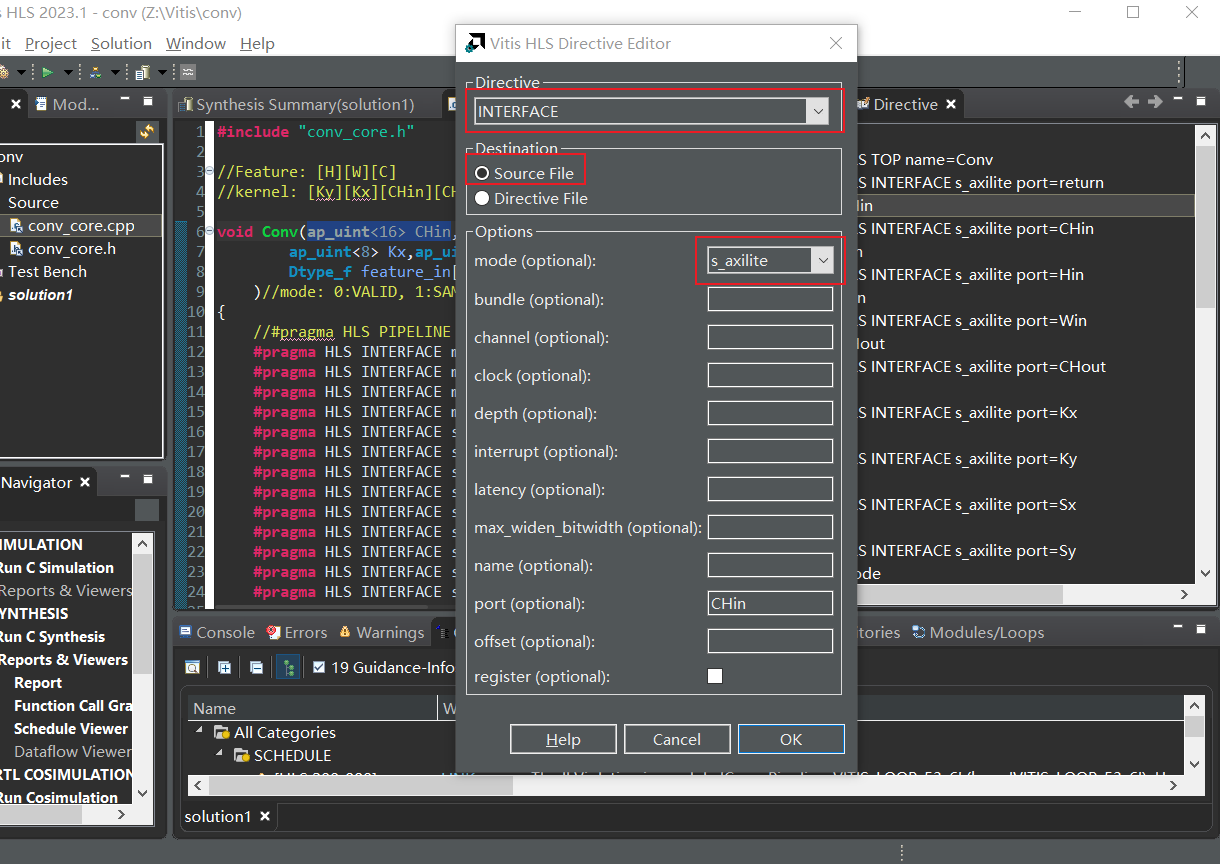
检测精度：如下图所示，最终在训练集的识别精度为0.984375，在测试集的识别精度为0.863636。



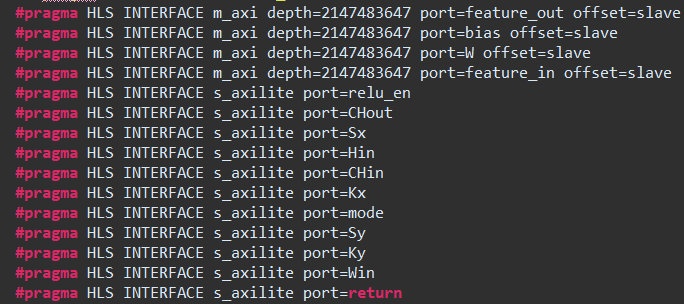
### (3)ip核设计

①HLS

在Vitis HLS中编写HLS代码，将.h文件，.cpp文件和testbench文件放入项目中，选择板子用的芯片以及设置周期。之后，将电路要实现的函数，把输入输出参数挂载到AXI总线上，操作步骤是在Directive窗口中的参数中insert directive，Directive类型选择INTERFACE，mode类型选择s\_axilite，如图所示：



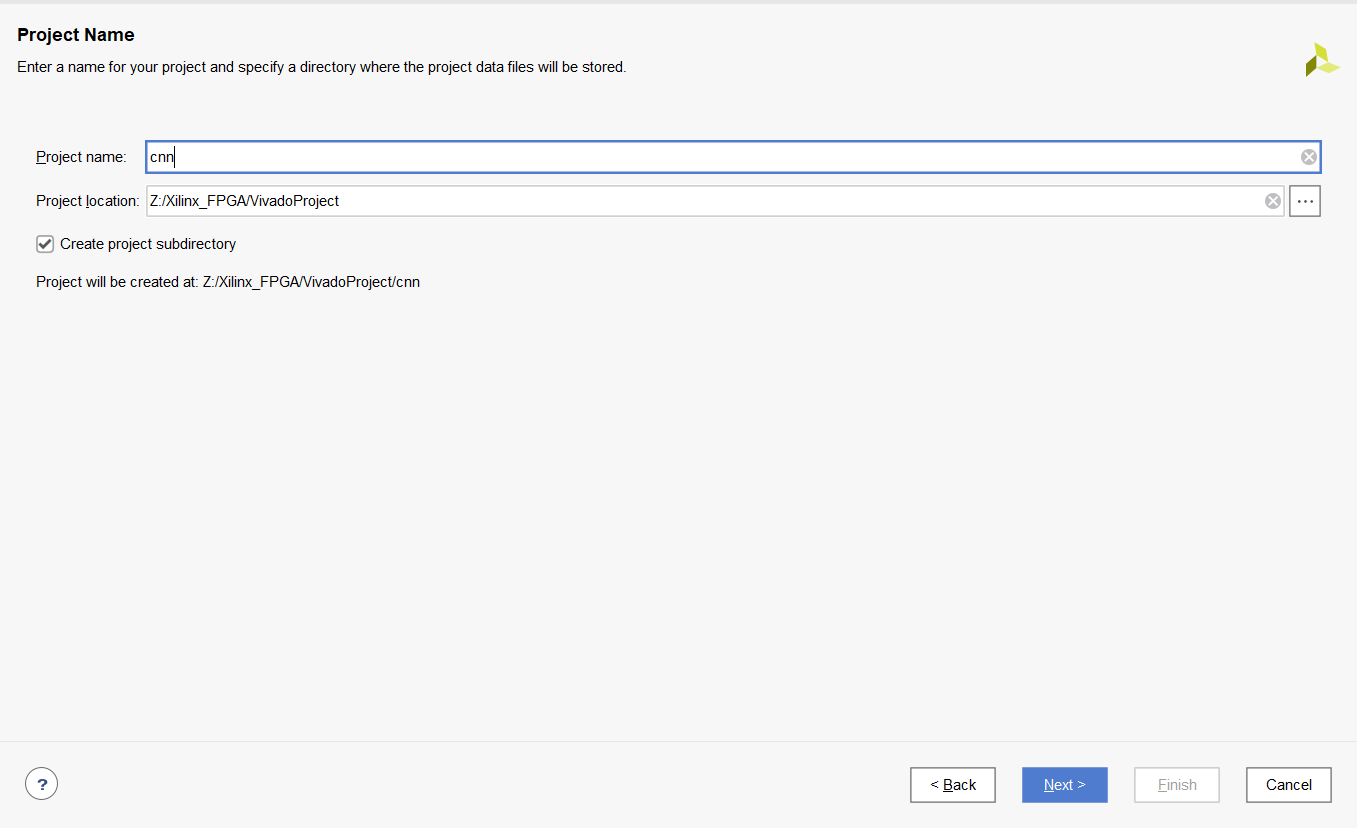
接着指定一个参数是AXI总线上的master端还是slave端，master标志表示之后生成的ip核中该参数是往外传出的，slave标志表示之后生成的ip核中该参数是从外传进的。卷积函数所有参数添加完directive，如下所示：

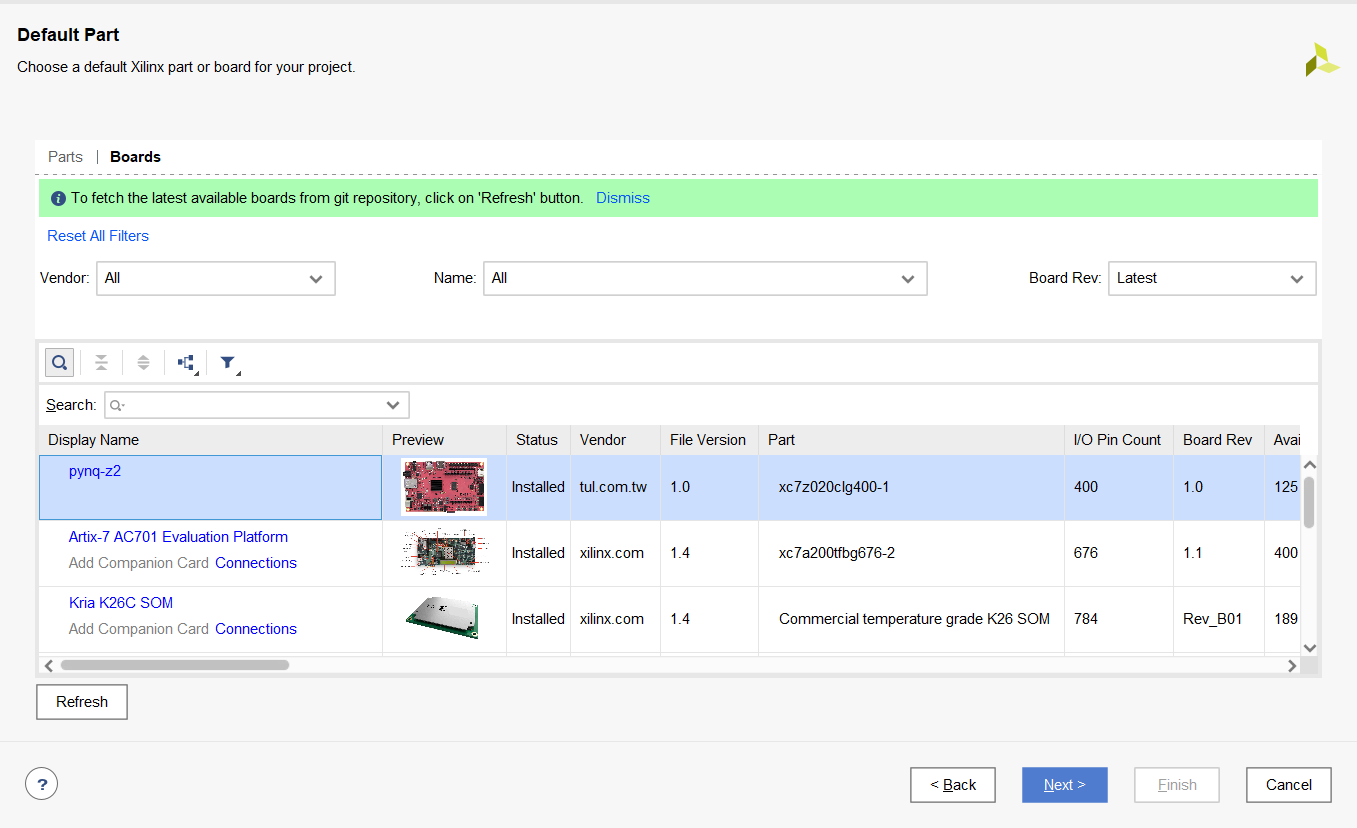


之后进行C Simulation、C Synthesis、Cosimulation，全部通过后就可以Export RTL，导出我们需要的ip核。

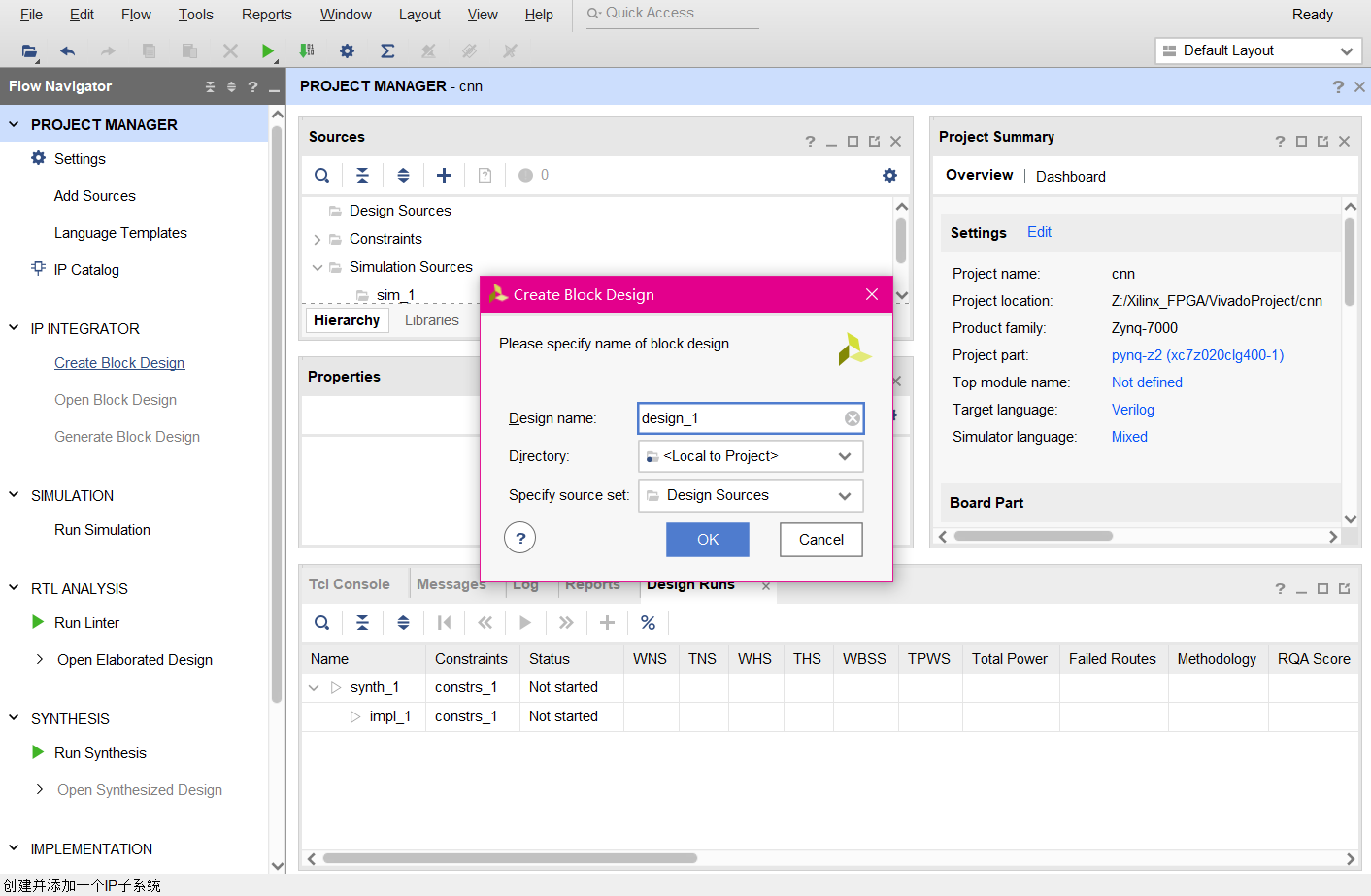
②vivado工程block design

创建vivado工程，选择pynq-z2板子型号，

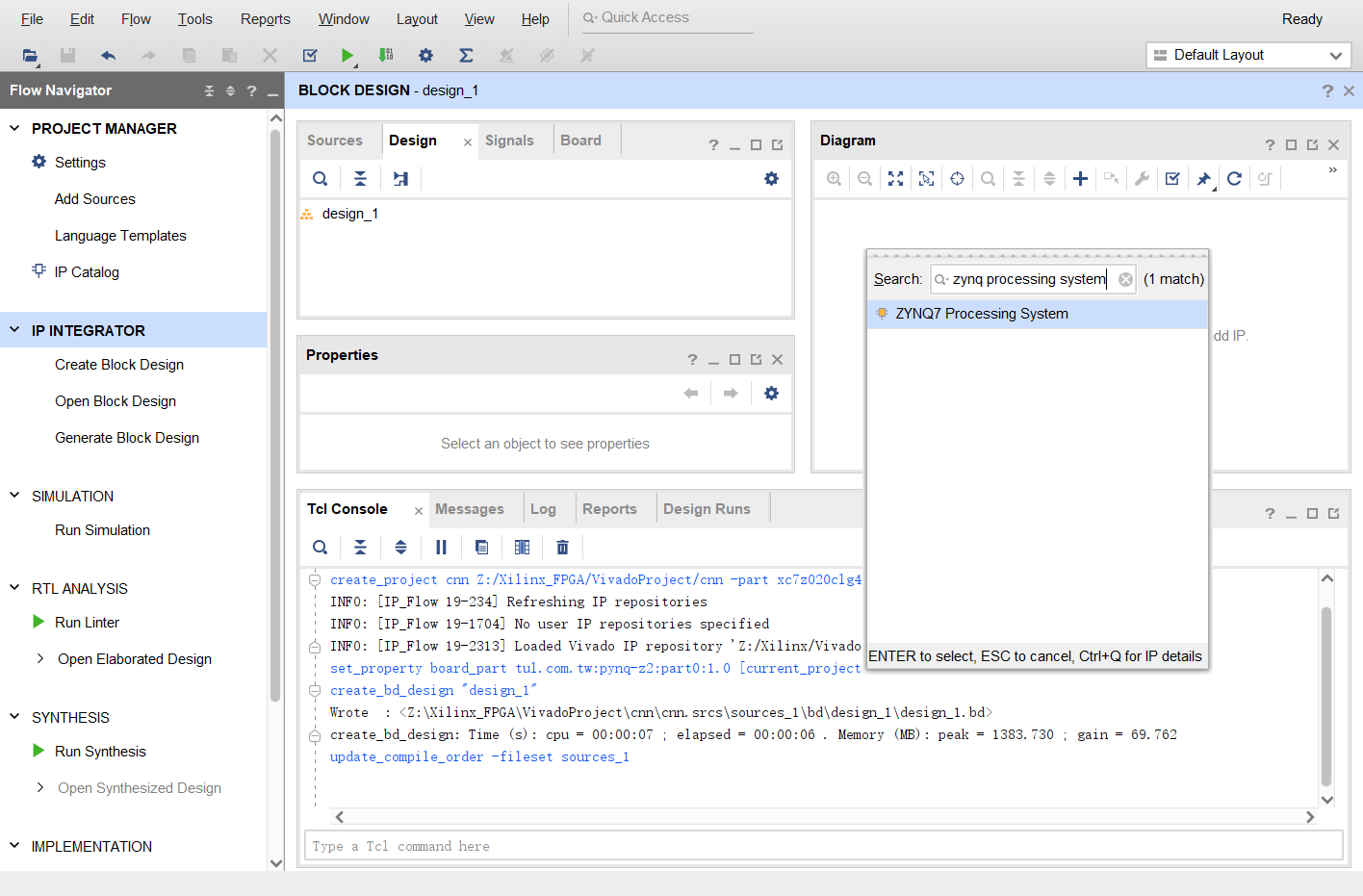


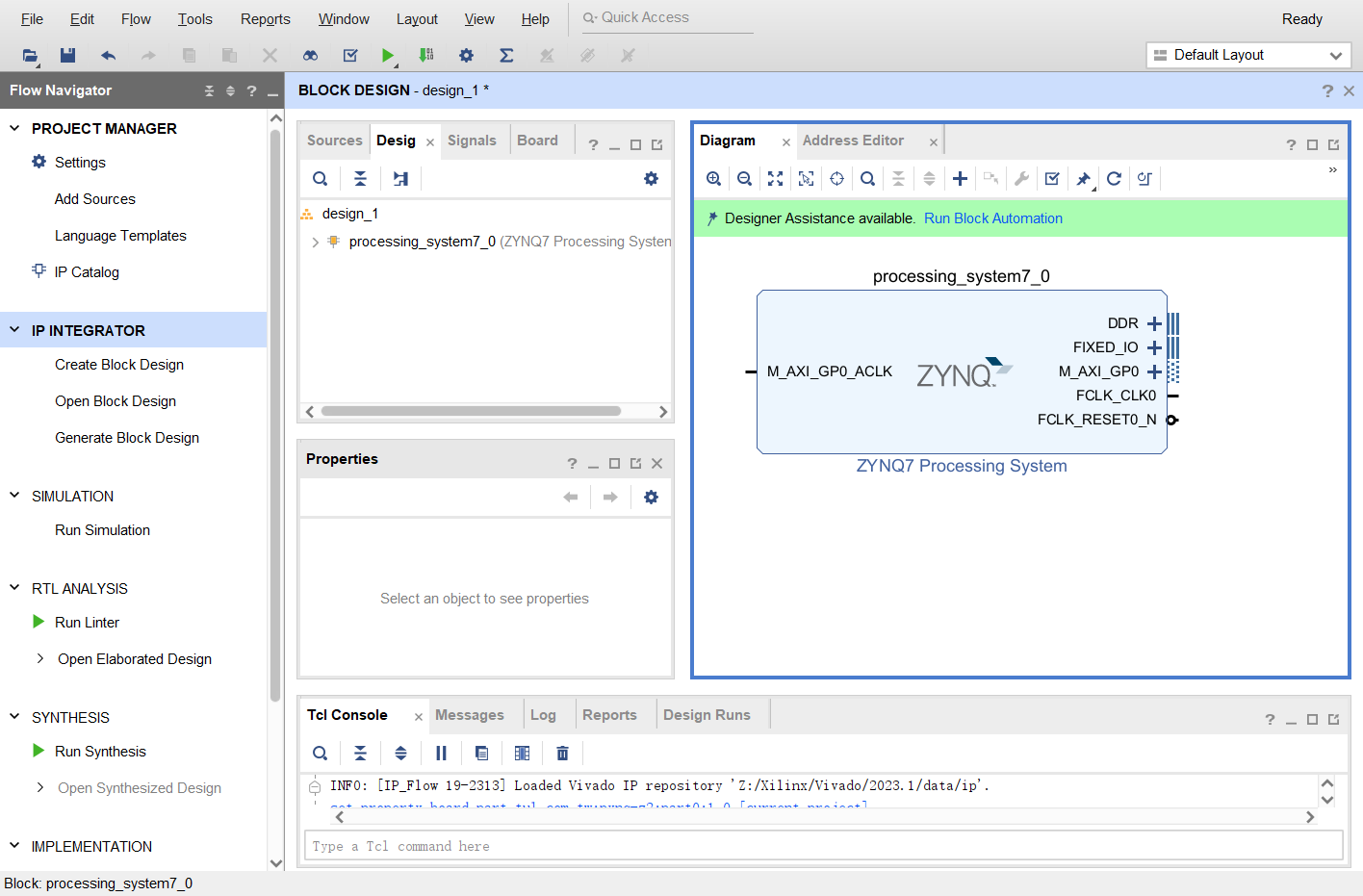


create block design，

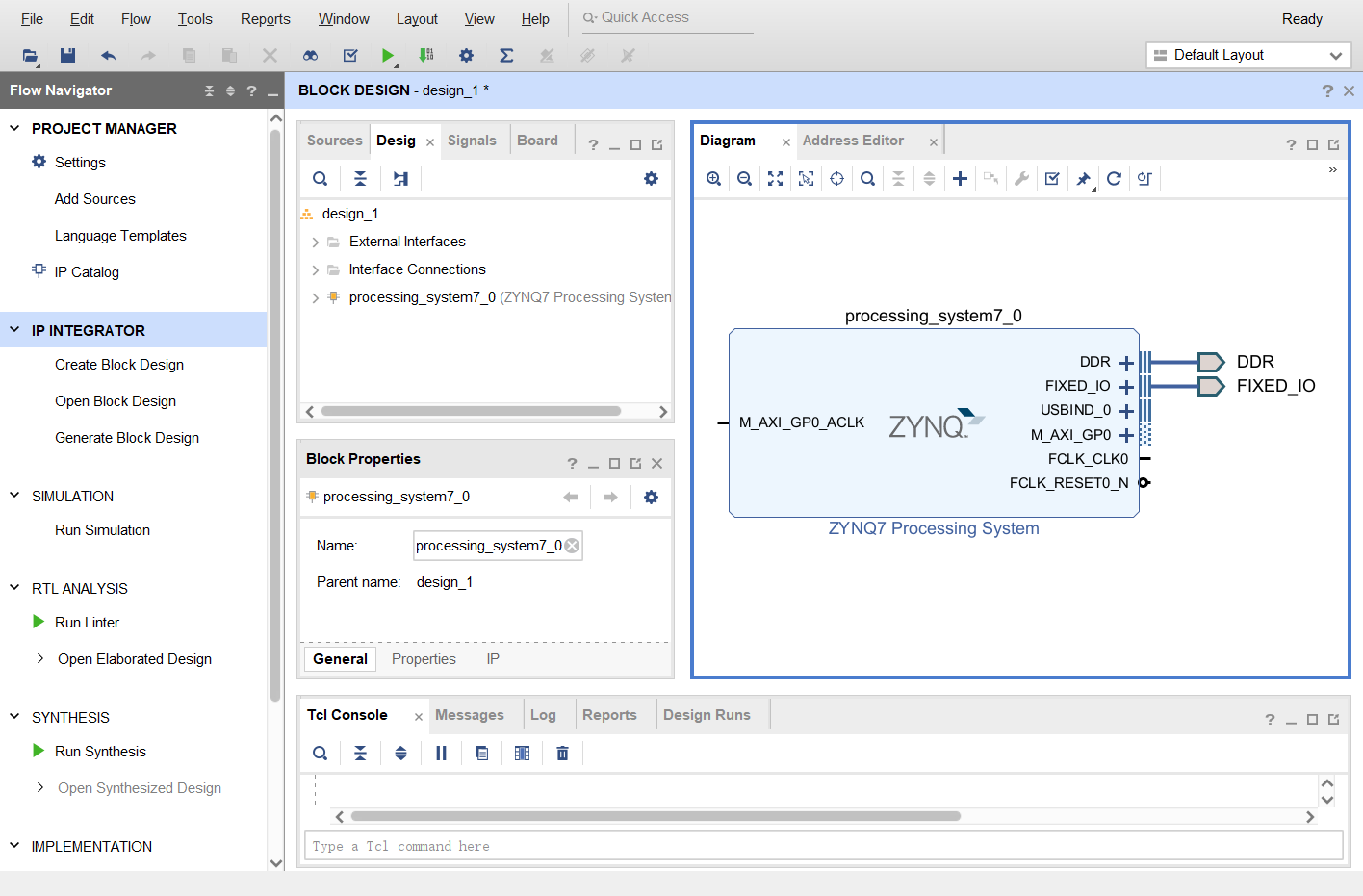


添加ARM核，

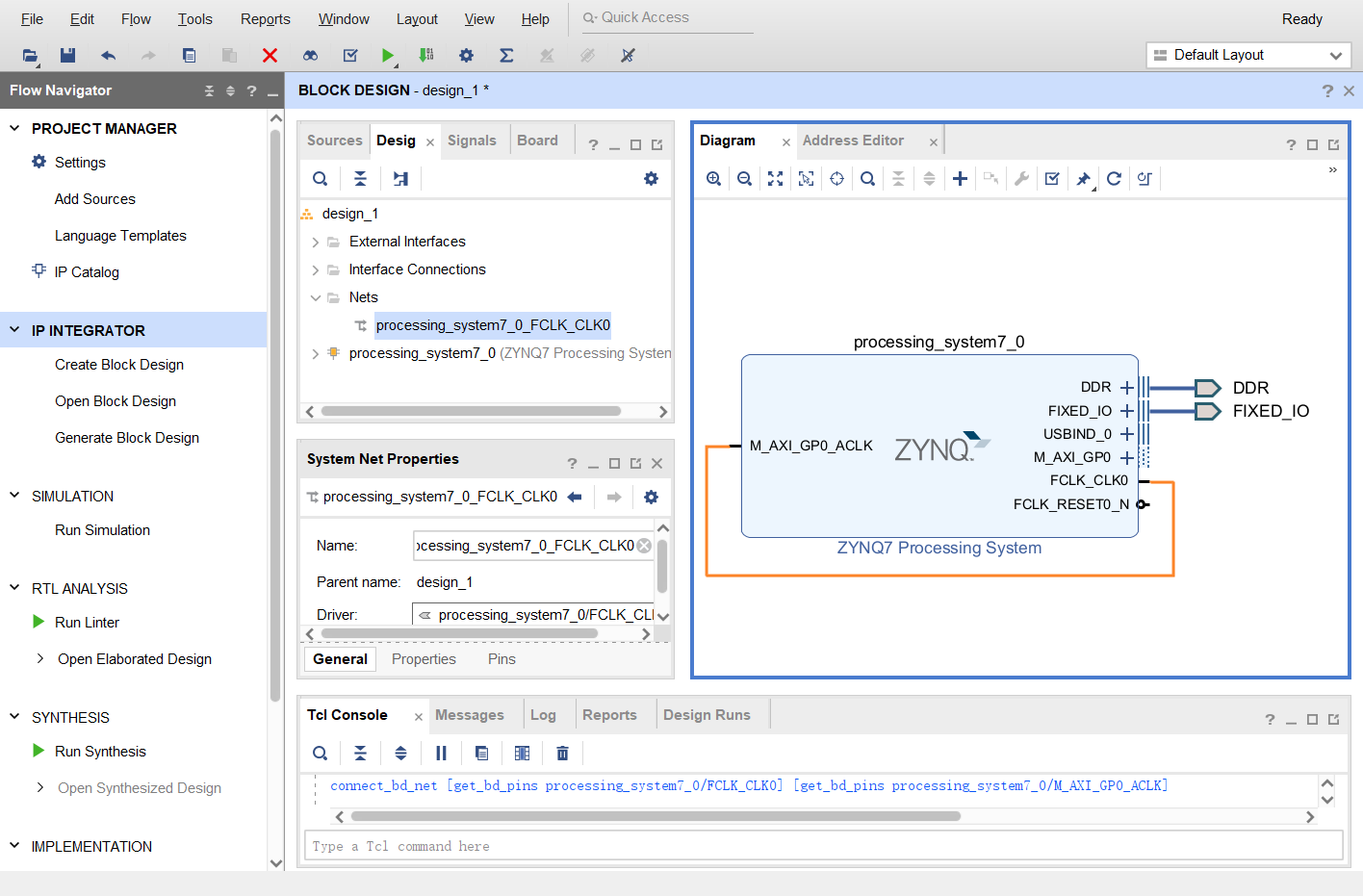




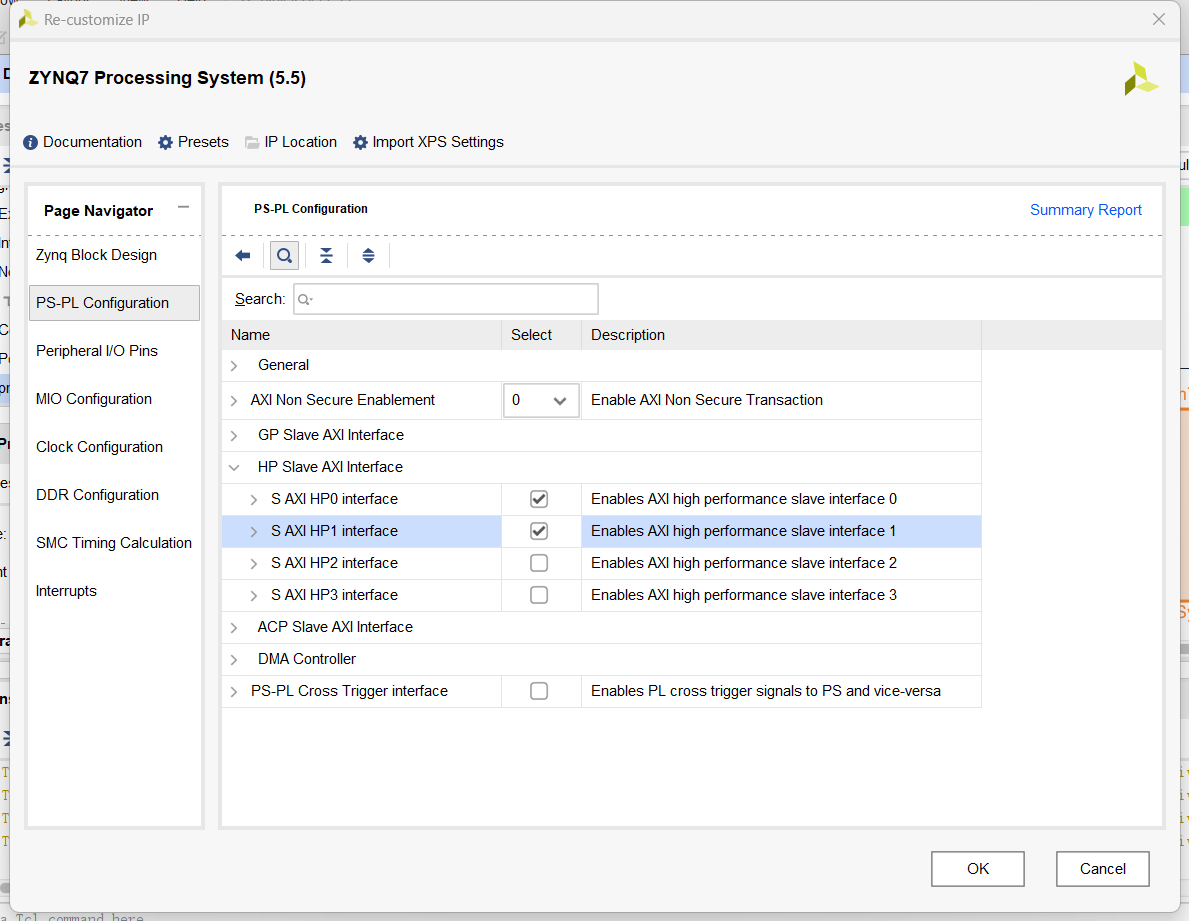
连线，



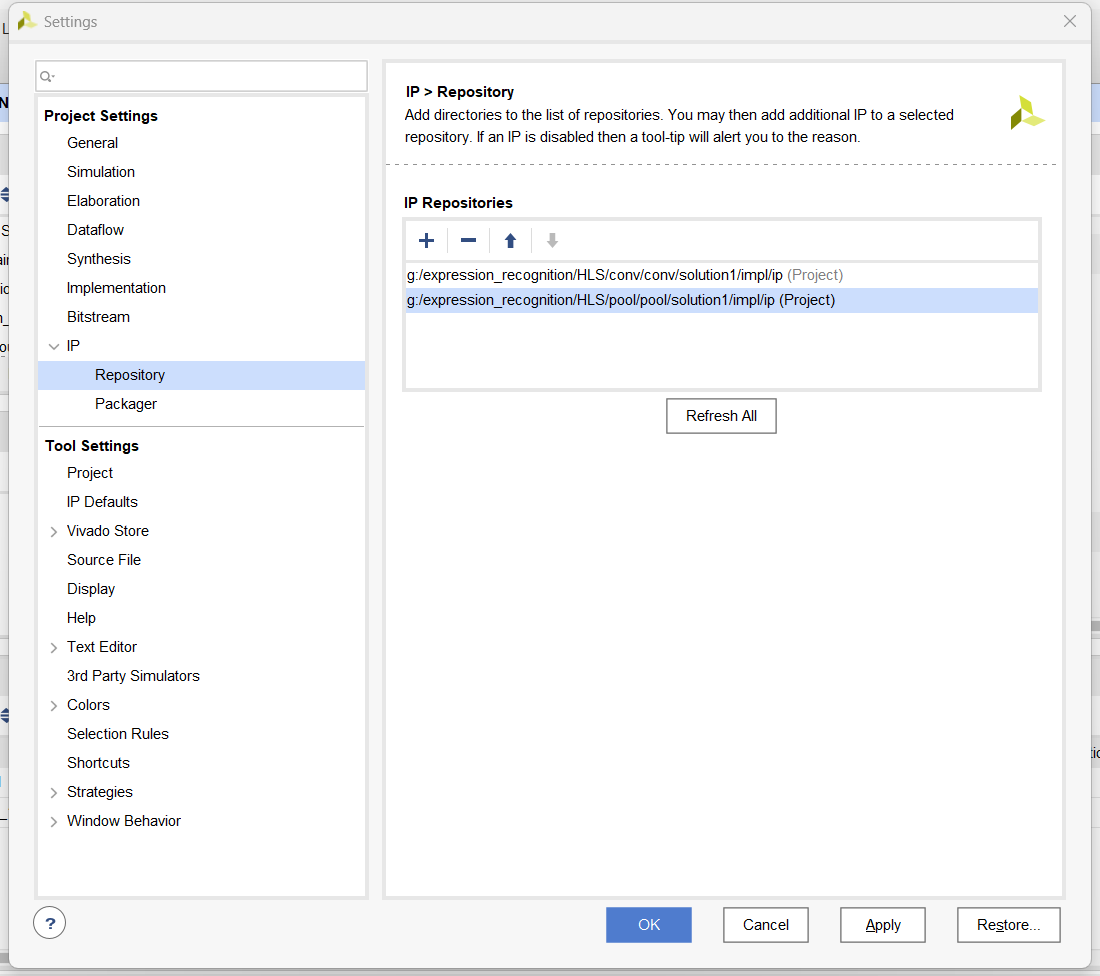
将fpga的主时钟作为AXI总线的时钟，连线

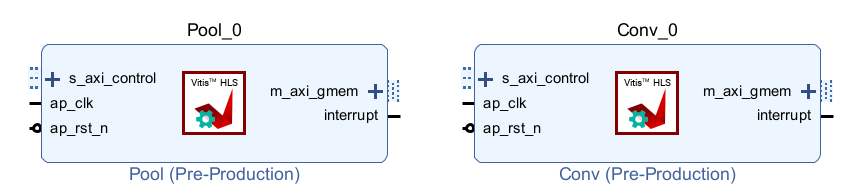


然后勾选两个AXI接口，

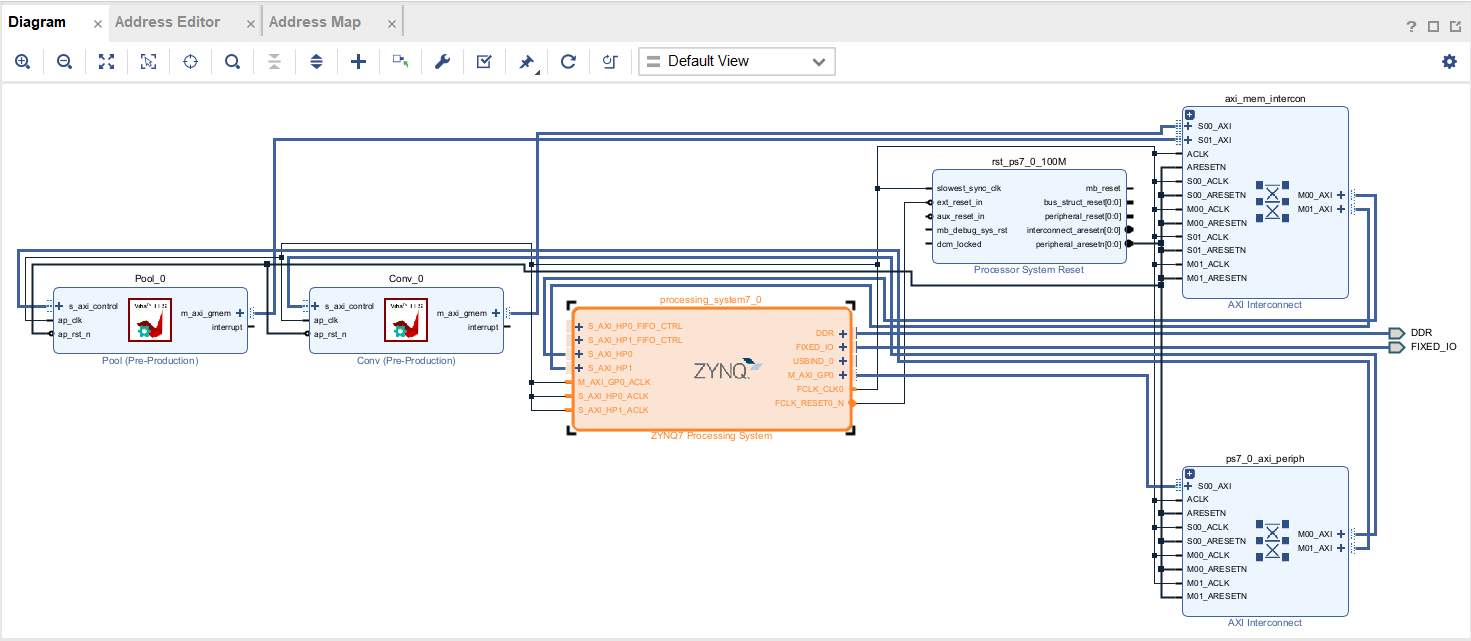


添加导出的ip核到工程中，

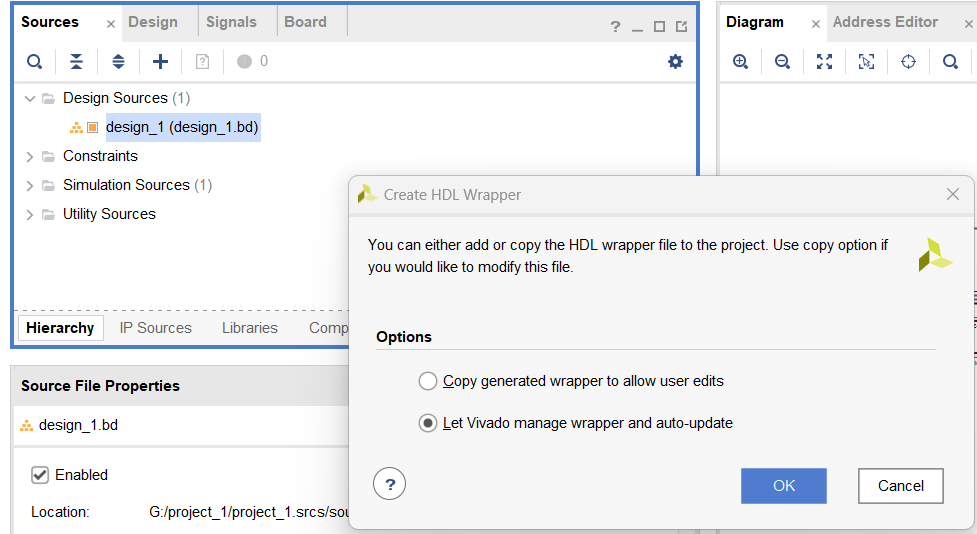




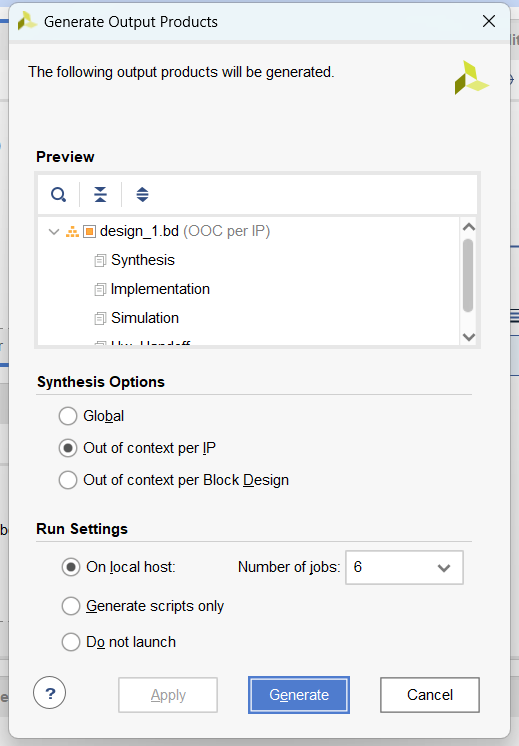
连好线后的Diagram，



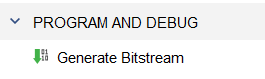
create HDL wrapper



Generate Output Products，



生成bit流文件。



③根据\_hw.h文件编写python代码

在HLS过程中Export RTL导出的文件中包含\_hw.h文件，这个文件描述了C++函数转换成verilog的电路描述后，函数参数被分配的AXI总线的地址，所以我们可以根据这个文件和C++代码分析参数的地址位置，然后在python代码中将数据写到对应的地址中，例如：

overlay = Overlay("cnn.bit")

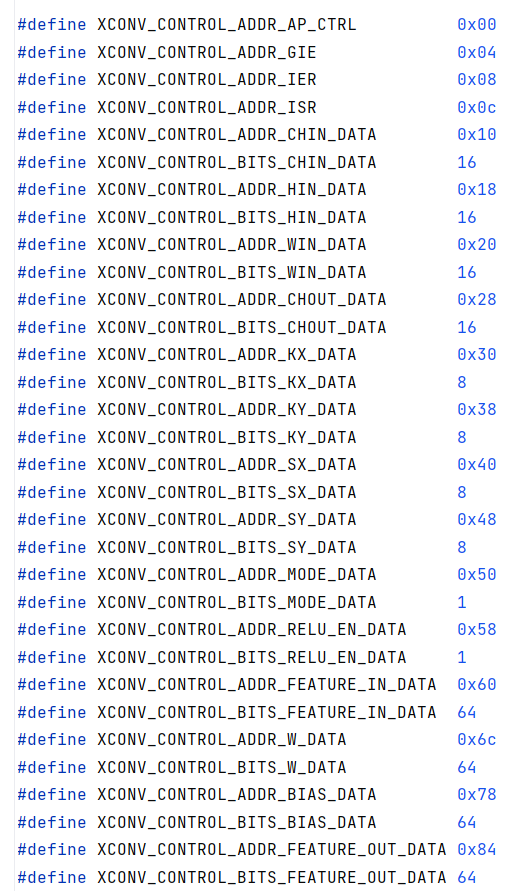
conv=overlay.Conv\_0

conv.write(0x30, Kx)

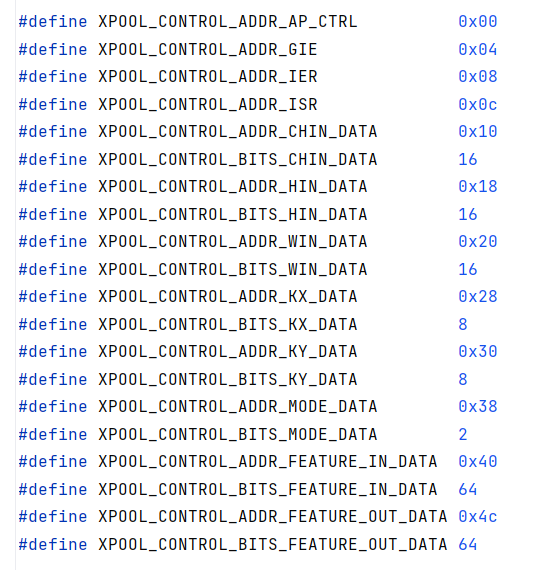
conv.write(0x38, Ky)

表示将卷积核的尺寸参数写入对应的ip核地址中。

Conv的地址如下：



Pool的地址如下：



### (4)PS端调用检测

①表情图像识别过程分析

神经网络模型进行表情识别主要分成四步：生成卷积核、卷积、池化、全连接，这四个步骤对应的数学操作本质上是构造矩阵、卷积运算、求最大值、加权平均，其中构造矩阵和加权平均计算中需要使用到一些提前设置好的参数值，这些参数值的确定和来源方法就是通过机器学习的方法拟合得到，即通过上面的训练模型步骤中得到相应的参数，卷积运算和求最大值操作根据得到的参数值进行运算，这两步操作是通过开发板PL端的电路设计完成计算。

②PL端电路连接

vivado工程跑完generate bitstream可以得到hwh文件和bit文件，其描述了接口关系和硬件设计，之后通过在jupyter notebook中调用Overlay库加载bit文件，动态地将硬件电路连接，进而可以通过在PS端使用python代码控制FPGA 逻辑资源及IO。

③pynq.allocate分配被ip使用的内存

ip访问内存前需要提前使用allocate分配内存缓冲区，分配的缓冲区能够用于在PS端和PL端之间高效地传输数据，连接到AXI Master(HP or ACP ports)的IP能够访问PS DRAM，这个分配过程实际上告诉PL端的ip要访问的PS的DRAM的size和address。python代码调用pynq.allocate返回一个pynq.Buffer对象，是一个numpy.ndarray的子类，因此可以很方便地和numpy.ndarray对象进行数据交换，除此之外，pynq.Buffer对象还有一个device\_address，可以传递给PL的ip，以便其访问pynq.Buffer对象中的数据。

# 分配输入图片的缓冲区

image = allocate(*shape*=(IN\_HEIGHT1, IN\_WIDTH1, IN\_CH1), *dtype*=np.float32)

# 分配卷积核的缓冲区

W\_conv1=allocate(*shape*=(KERNEL\_HEIGHT1, KERNEL\_WIDTH1, IN\_CH1, OUT\_CH1), *dtype*=np.float32)

④cv2读取图片进行前处理

通过python代码调用cv2库读取本地图片，转化为灰度图，数据类型为float32，resize为训练好的神经网络模型输入的尺寸大小，并手动进行图片量化操作，对图片np.ndarray数组除以255，即是将float32转为uint8数据类型。

image = allocate(*shape*=(IN\_HEIGHT1, IN\_WIDTH1, IN\_CH1), *dtype*=np.float32)

detect\_image = cv2.imread("01.ppm", cv2.IMREAD\_GRAYSCALE).astype(np.float32)

detect\_image = cv2.resize(detect\_image, (28, 28))

image[i][j][k] = detect\_image[i][j] / 255

⑤加载卷积、池化和全连接二进制文件

使用python代码从二进制文件中读取卷积层和池化层训练好的参数，将参数通过Overlay传入对应设计好的逻辑资源中，例如overlay = Overlay("cnn.bit")，conv = overlay.Conv\_0，conv.write(0x70, bias.physical\_address)就是卷积层的卷积核偏置参数缓冲区所在地址通过PS端的接口写到对应的逻辑资源中。

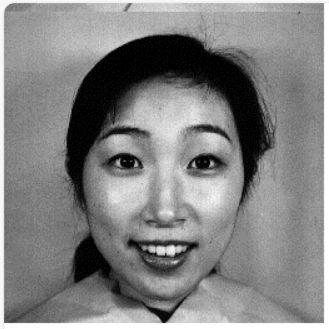
⑥获取检测结果

将对应的参数在PS端的接口通过AXI总线写入PL端的逻辑资源后，使用python代码操作Overlay库对连接到PS端的接口进行解析，读取返回地址的结果数据，返回结果地址中存放的是全连接层预测的结果，维数对应分类的类别数，遍历获取概率值最大的数，通过预先定义好的的映射关系即可得到对应的预测的类别。

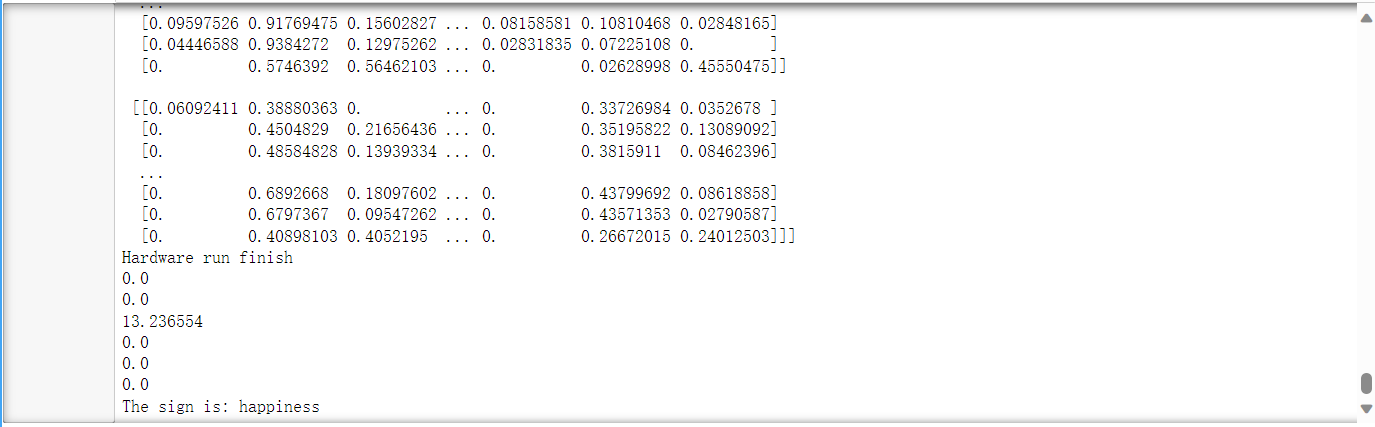
## 三、项目运行展示

cv2读取表情图片：

image1=cv2.imread("/home/xilinx/jupyter\_notebooks/expression\_recognition/image/happiness\_2.jpg", cv2.IMREAD\_GRAYSCALE).astype(np.float32)



获取检测结果：



## 四、项目总结

通过该项目，我们学习到了PYNQ-Z2这块板子的整体开发流程，并且了解到了FPGA相较于CPU、GPU的优势所在；该项目还使我们充分复习到了这学期课程所学的智能系统优化设计相关知识，并掌握了软硬件协同设计这一思想，对日后的项目开发具有很大帮助。但是，本项目也存在一些不足之处，比如模型的结构较为简单，训练集数量较少，所以容易出现模型过拟合的现象，之后可以进一步改进。