**智能芯片与系统实践 大作业报告**



**姓名：任思剑 学号：2153264**

**姓名：刘卓明 学号：2151406**

**姓名：金昊 学号：2150225**

**专业 : 人工智能**

**课程 : 智能芯片与系统实践**

**指导教师 : 徐和根**

**答辩日期：2023.01.17**

一、原理解析

1.1、开发板Zynq-7020简介

1.2、PYNQ

二、环境配置

2.1、镜像烧入

2.2、芯片与电脑的连接及连网

2.3、Linux环境配置

三、项目成果

3.1、条形码及二维码检测与解码

3.2、人脸检测

3.3、基于Lenet网络进行手写数字识别

四、问题讨论、分析及解决方法

五、总结

# 原理解析

## 开发板Zynq-7020简介

Zynq-7020 是由 Xilinx 公司生产的一款集成电路芯片，属于 Zynq-7000 系列。这个系列的芯片是一种系统级芯片（SoC），将高性能的 ARM 处理器与 FPGA（现场可编程门阵列）技术结合在一起。

ZYNQ 分为PS和PL两个部分，PS有两个ARM的核，在上面运行linux操作系统，在操作系统上再运行python。PL部分就是FPGA的逻辑资源，开发者在PL中添加IP或者将自己用C或者HDL语言写好的模块封装成IP，这些IP都被连接到PS端，一般都是通过AXI总线。PYNQ有一个特有的库叫overlay，使用这个库可以对连接到PS端的接口进行解析，进而控制FPGA 逻辑资源及IO。Overlay是一种特殊的位流（bitstream），用于配置ZYNQ SoC上的FPGA。它包含了一组预定义的硬件设计，比如IP（Intellectual Property）核、接口、逻辑电路等。用户可以通过Python编程接口动态加载和卸载这些overlay，从而改变FPGA的功能。

每次开始一个新的涉及PL端的开发的时候，先在vivado 里面建一个工程，添加需要的各种IP，然后以ZYNQ为核心进行连接设计，经过编译后，会得到一个bit文件和一个tcl文件。bit文件就是你的硬件设计，tcl文件描述了接口关系。将这两个文件复制到PYNQ的目录下，即可进行调用。每一次调用的时候，你设计的硬件都是被动态加载的，这一点不同于大家熟悉的加载过程。动态加载无需重启硬件，操作系统无需重启。这一是一个极有优势的设计，我记得当年调试过intel 和Altera共同推出的阿童木平台完全不同。

Zynq-7020 被广泛应用于多种不同的领域，例如嵌入式系统、工业自动化、视频处理和汽车技术等。其灵活的 FPGA 组件允许开发者根据特定应用的需要进行自定义设计和优化，而集成的 ARM 处理器则提供了强大的计算能力和运行操作系统的能力。

由于其独特的结构，Zynq-7020为需要高度定制和高性能处理的应用提供了一种高效且灵活的解决方案。

## PYNQ

PYNQ是一种针对Xilinx Zynq架构的开发平台，它利用Python语言和库来实现硬件加速。PYNQ框架的核心思想是使得硬件设计对于软件开发人员更加友好，从而简化了使用FPGA（现场可编程门阵列）的复杂性。

它的核心特点有：

1）Python 语言支持：PYNQ使得开发者可以使用Python及其丰富的库来控制和访问Zynq的硬件资源，包括处理器和FPGA逻辑。

2）Jupyter Notebook 集成：PYNQ通常配合Jupyter Notebook使用，提供一个交互式的编程环境，便于实验和原型开发。

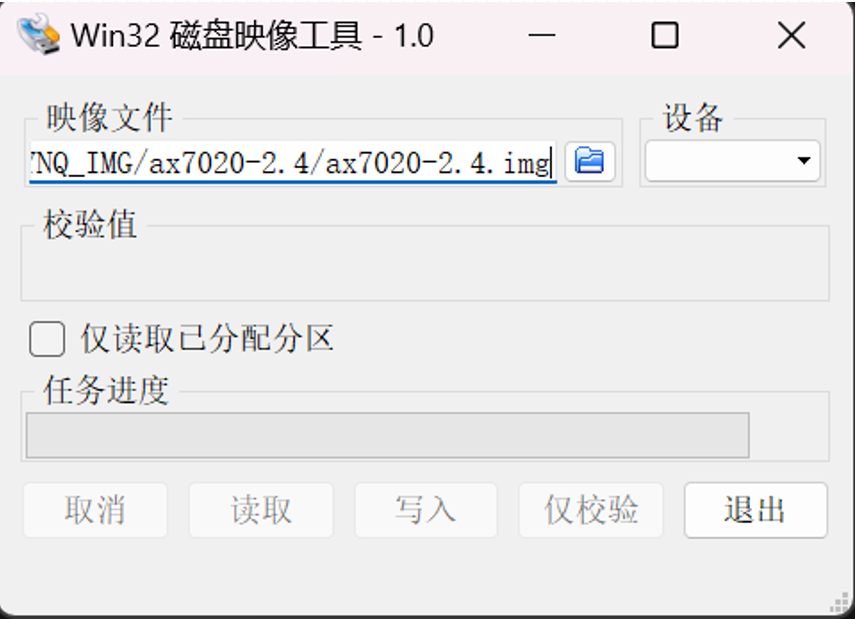
3）硬件加速：用户可以使用高级综合工具（如Xilinx的Vivado HLS）来开发自定义硬件加速器，然后通过Python调用这些加速器。

4）板级支持：PYNQ支持多种Zynq开发板，包括Xilinx的自家板卡以及第三方板卡。

# 环境配置

## 镜像烧入

ZYNQ7020开发板出厂时是不带PYNQ功能的，因而需要手动烧写镜像。首先，从网上下载PYNQ镜像，将ZYNQ7020开发板上的SD卡取出，插入USB读卡器，并将SD卡连接电脑。



在磁盘管理中删除SD卡中的所有分区，然后新建一个分区，命名为PYNQ，再利用Win32DiskImager工具，将PYNQ镜像烧入到SD卡中，再将SD卡插回ZYNQ7020开发板。

## 芯片与电脑的连接及连网



如图连接即可。（绿色的线是网线，黑色的线是摄像头）电脑端共享WIFI给ZYNQ7020开发板（此时电脑端的ip地址会被强行改变），再更改开发板的ip地址，使其与电脑处于同一网络下，从而可以连网。然后打开浏览器输入<http://pynq:9090>，进入到开发板上的Jupyter Notebook主页（同时这也表明连接成功，否则无法进入该网站）。

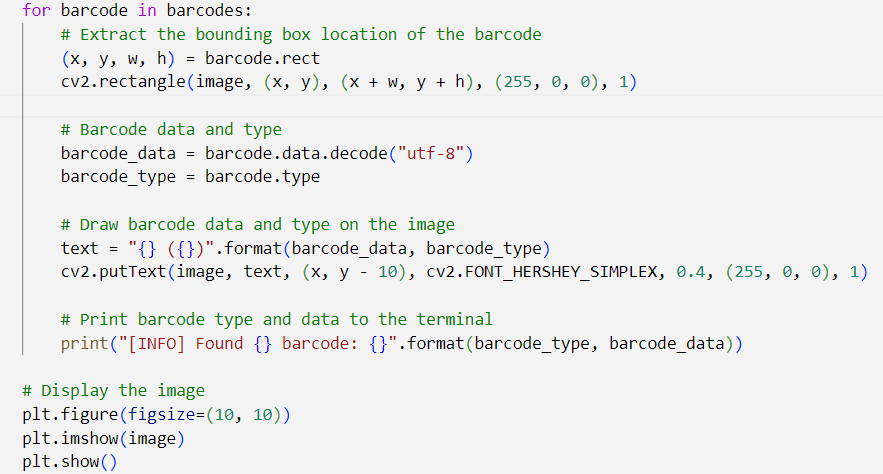
## Linux环境配置

由于使用pynq需要的开发环境是linux系统，所以在联网完成后我们进行了linux环境的配置工作。首先是采用sudo apt-get update和sudo apt-get upgrade指令安装或更新软件包，但是在开始安装的时候无法正常安装，我们尝试了换镜像源的操作，但是换源之后仍然提升无法安装，我们根据报错信息，将镜像源地址中的https换为了http（由于https建立的是安全的私密的连接，所以可能访问受限）修改之后发现可以正常安装软件包；然后由于开发需要的是python环境，而我们导入的pynq镜像中缺失很多关键的包，所以需要补充安装，在配置python环境的时候我们先利用apt-get来下载pip但是下载失败，然后我们尝试了使用python3 get-pip.py的方式进行安装，安装成功。我们利用pip安装Python的包，但是也出现了各种报错问题或者是下载速度过慢的问题，所以对于有些包，我们采用的是离线安装，也即在官网或者镜像网站下载安装包后上传到开发板进行安装；此外，为了不破坏pynq原有的python环境，我们在系统中利用virtualenv创建了虚拟环境进行开发。

# 项目成果

## 条形码与二维码检测及解码

鉴于条形码与二维码在生活中被广泛应用，我们觉得对于条形码与二维码的检测与解码有着十分重要的现实意义。本项目利用opencv和pyzbar两个库，再将摄像头与开发板连接，实时获取图像，并对图像进行条形码与二维码检测（并用红框标出轮廓）及解码（读出信息）。pyzbar 是一个用于解码条形码和二维码的 Python 库。这个库可以读取多种格式的条形码和二维码，并且它能够在不同的操作系统上工作，包括 Windows、macOS 和 Linux。实现的部分代码如下：



其中，x，y为检测到的条形码及二维码的左上角坐标，w和h为条形码或二维码的宽度与高度。rectangle为标注框的函数，给出左上及右下的坐标，即可画出框。

效果图如下：

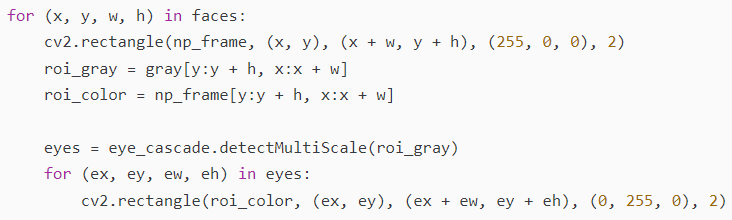
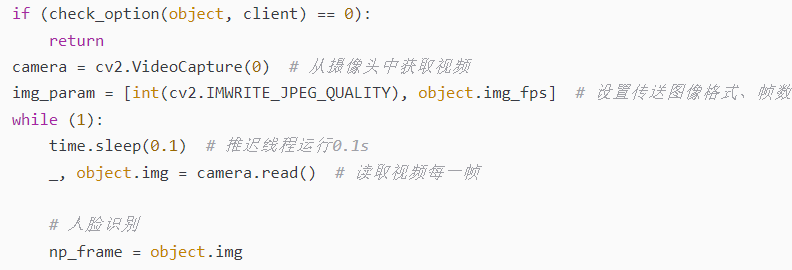
 

不过，限于摄像头性能，拍摄出来的照片对比度不高，因而该算法的效果无法显现。所以我们选择了一些已有的条形码、二维码进行检测与解码。可以看到，输出的图片中对于二维码和条形码进行了加框，并在图片的上方输出了解码后的信息（二维码是一个网址，条形码是书号。）如果使用性能更好的摄像头，估计检测性能会有所提高。

## 人脸检测

在这一任务中，同样将摄像头与开发板连接，实时获取图像，这次可以对实时图像进行实时的检测（对人脸及眼睛进行加框。）同样的，在开发板中部署检测的代码并进行检测，然后将检测的结果通过socket传回给电脑，并在电脑上显示检测结果。实现的部分代码如下：

开发板端：



大部分的解释已在代码的注释中，其中camera.read()的返回值有两个，一个是一个布尔值（True 或 False），还有一个是从视频源捕获的图像帧。

布尔值表明帧是否成功读取。如果帧已成功捕获，则为 True；如果未捕获（例如，相机不工作或已断开连接），则为 False。

图像帧是表示捕获图像的矩阵或类数组结构。这个图像是以 NumPy 数组的形式，包含帧的像素值。这个数组的形状和数据类型取决于摄像头的配置和能力。通常，对于标准的 RGB 彩色图像，这个数组将是一个三维数组，其维度对应于帧的高度、宽度和颜色通道的数量（RGB 通常为 3）。

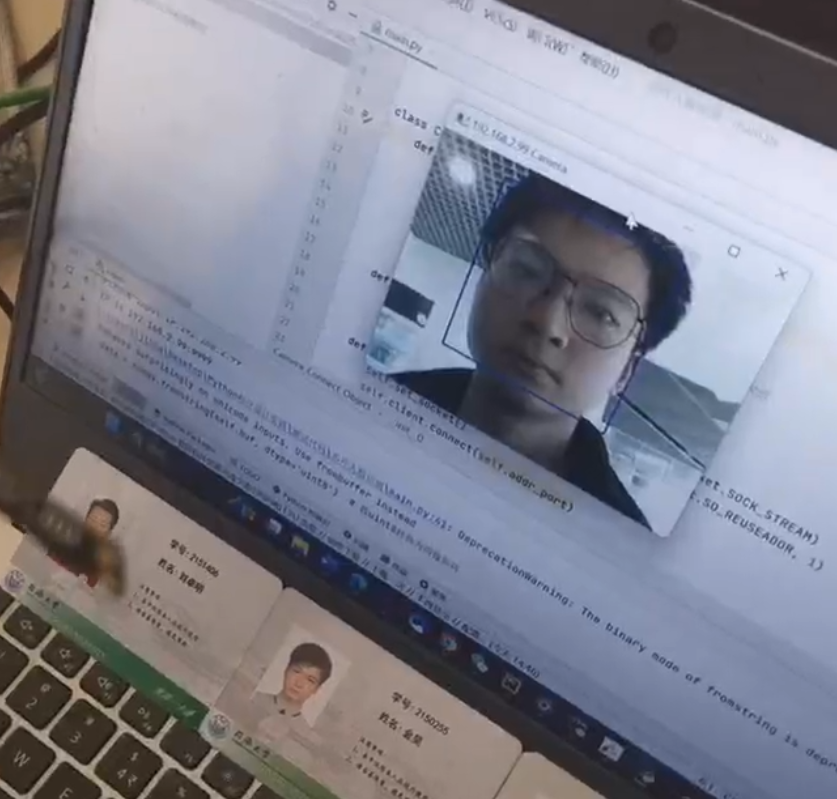
脸和眼睛加框的实现与条形码类似，因而不再赘述。

电脑端：



在电脑端，接收检测的结果，并通过imshow()在电脑上展示出来。

检测的效果图如下：



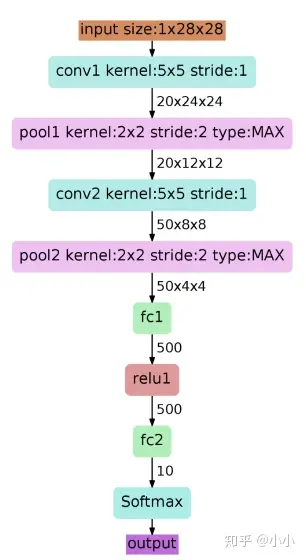
本小组成员 2150255金昊 实时检测效果图

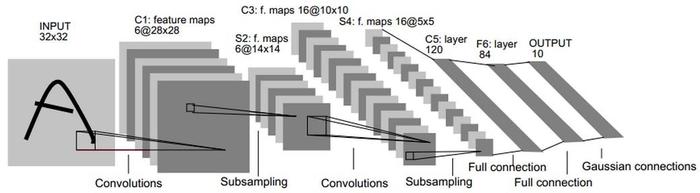
## 基于LeNet网络进行手写数字识别

**3.3.1 LeNet网络介绍**

Lenet 是一系列网络的合称，包括 Lenet1 - Lenet5，由 Yann LeCun 等人在 1990 年《Handwritten Digit Recognition with a Back-Propagation Network》中提出，是卷积神经网络的 HelloWorld。

Lenet是一个 7 层的神经网络，包含 3 个卷积层，2 个池化层，1 个全连接层。其中所有卷积层的所有卷积核都为 5x5，步长 strid=1，池化方法都为全局 pooling，激活函数为 Sigmoid。LeNet这个网络虽然很小，但是它包含了深度学习的基本模块：卷积层，池化层，全连接层。网络结构如下：





LeNet-5共有7层，不包含输入，每层都包含可训练参数；每个层有多个Feature Map，每个FeatureMap通过一种卷积滤波器提取输入的一种特征，然后每个FeatureMap有多个神经元。LeNet 的核心思想是通过交替使用卷积和池化层，逐渐减小特征图的空间尺寸，最终连接全连接层进行分类。

C1层-卷积层：对输入图像进行第一次卷积运算（使用6个大小为 5\*5 的卷积核），得到6个C1特征图（6个大小为28\*28的 feature maps, 32-5+1=28）。、卷积核的大小为5\*5，总共就有6\*（5\*5+1）=156个参数，其中+1是表示一个核有一个bias。对于卷积层C1，C1内的每个像素都与输入图像中的5\*5个像素和1个bias有连接，所以总共有156\*28\*28=122304个连接。总共有122304个连接，但是我们只需要学习156个参数，主要是通过权值共享实现的。

S2层-池化层（下采样层）：第一次卷积之后紧接着就是池化运算，使用 2\*2核进行池化，于是得到了S2，6个14\*14的 特征图（28/2=14）。S2这个pooling层是对C1中的2\*2区域内的像素求和乘以一个权值系数再加上一个偏置，然后将这个结果再做一次映射。于是每个池化核有两个训练参数，所以共有2x6=12个训练参数，但是有5x14x14x6=5880个连接。

C3层-卷积层：C3中的每个特征map是连接到S2中的所有6个或者几个特征map的，表示本层的特征map是上一层提取到的特征map的不同组合。

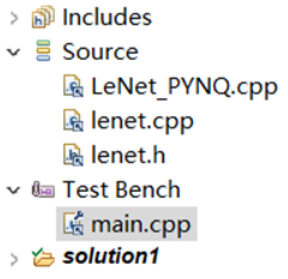
S4层-池化层（下采样层）：S4是pooling层，窗口大小仍然是2\*2，共计16个feature map，C3层的16个10x10的图分别进行以2x2为单位的池化得到16个5x5的特征图。这一层有2x16共32个训练参数，5x5x5x16=2000个连接。连接的方式与S2层类似。

C5层-卷积层：C5层是一个卷积层。由于S4层的16个图的大小为5x5，与卷积核的大小相同，所以卷积后形成的图的大小为1x1。这里形成120个卷积结果。每个都与上一层的16个图相连。所以共有(5x5x16+1)x120 = 48120个参数，同样有48120个连接。

F6层-全连接层：F6层有84个节点，对应于一个7x12的比特图，-1表示白色，1表示黑色，这样每个符号的比特图的黑白色就对应于一个编码。该层的训练参数和连接数是(120 +1)x84=10164。

Output层-全连接层：共有10个节点，分别代表数字0到9。

**3.3.2 创建IP核**

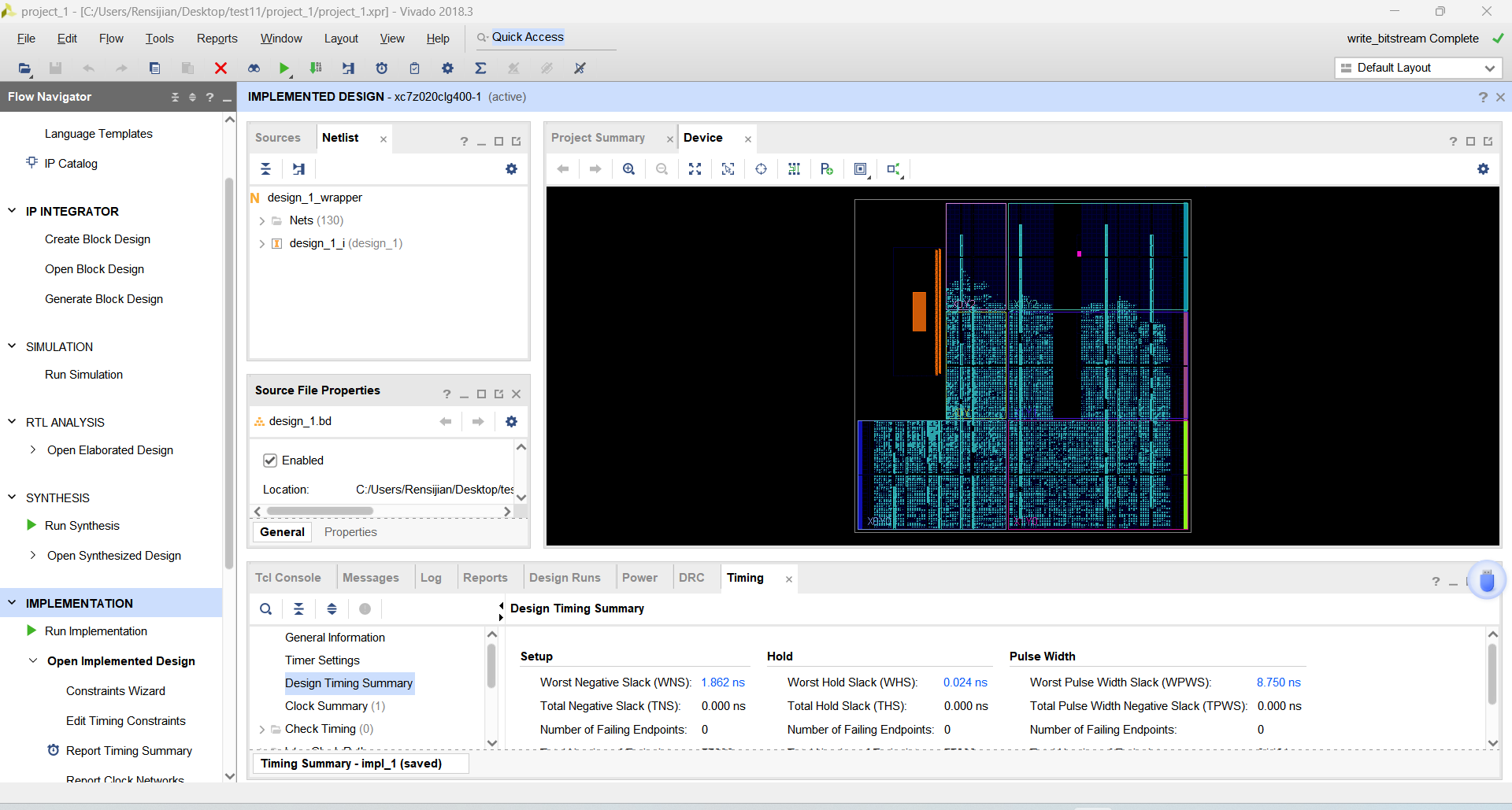


编写好lenet.cpp，LeNet\_PYNQ.cpp，main.cpp等文件，主要的功能是实现Lenet网络；将其导入到Vivado HLS中，并用仿真测试。

**3.3.3 加速优化**

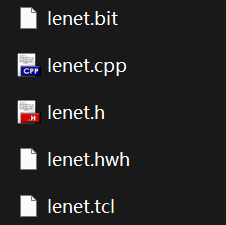
我们主要用到并行计算和数据相关这两种优化。并行计算使用到的有PIPELINE和UNROLL。数据主要用到ARRAY\_PARTITION。

我们使用Directive插入，主要使用PIPELINE和ARRAY\_PARTITION的Directive选择，工程的资源占有量大大提升，成功使用硬件加速。在模块控制接口设置为s\_axilite模式，使用AXI简化的控制接口，并调整数据的读取接口，加速效果明显。

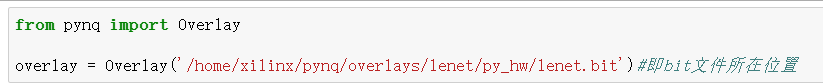


**3.3.4 IP核导出及上板验证**

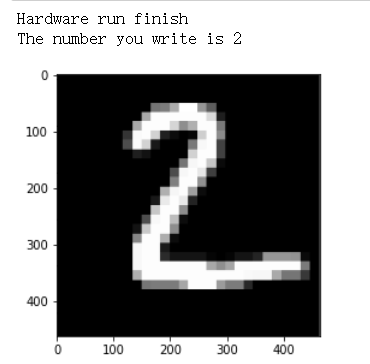
导入Lenet的ip核，将其与zynq7020的ip核进行连接，最终生成bitstream文件。



**3.3.5 将得到的bitstream上传到开发板中，用PYNQ进行调用**



代码如上所示。这话的意思就是让overlay这个软件按照.bit文件所描述的电路把FPGA连好，pynq在看到这话之后会自动在同一个路径下搜索tcl文件，tcl文件也找到了才能正确连电路，这也就是为什么我们需要把tcl和bit文件改成一样的名字的原因。我们最终得到的结果如下：



# 问题讨论、分析及解决方法

## 开发板联网问题

我们遇到的第一个问题是在开发板和电脑用局域网连接后，并设置以太网的ip为192.168.2.x（因为pynq默认的ip地址是192.168.2.99）但是开发板无法上网的问题。我们使用的方法是使用pc端的无线共享给以太网，最开始我们发现共享后仍然不能上网，查询相关资料后认为是DNS的问题，于是调配了开发板的DNS配置，但是发现仍然不能上网；然后经过查阅资料后发现Win11当WiFi被设置为共享模式后，以太网的ip地址会被强制置为192.168.137.x，于是我们尝试将开发板的ip置为了192.168.137.10，这样就实现了开发板与以太网在同一子网下，并且能够共享到WiFi网，实现了开发板联网的目的。但是当我们第二次访问192.168.137.10之后发现不能访问，我们认为是由于Win11下WiFi共享时候，以太网的ip地址都会默认置为192.168.137.x，所以这个ip可能会被占用，于是我们采用了pynq:9090的方式进行访问，并查询到了新的ip地址为192.168.137.168，重新连接后发现可以访问，而开发板依然能进行联网，从而可以下载开发所需要的库和包。

## Linux系统的架构问题

在安装软件包或者是python包的过程中我们发现linux系统具有多种架构，包括x86\_64，amd，aarch，arm等，并且在安装包的时候不同架构对应的安装指令不同（在换源的时候也是如此）。本次实验我们使用的镜像中的架构为armv7l架构，我们试图在pynq上安装pytorch或者是TensorFlow等深度学习的框架时候发现pytorch或TensorFlow的官方并不支持arm架构，所以我们尝试了寻找whl资源进行安装，但是由于wheel的一些bug，最终没有下载成功。但是我们仍然通过实验认识到了不同架构的linux系统的差异。

## PYNQ镜像版本问题

我们在使用PYNQ镜像时，先从官网上下载了PYNQv2.5版本及PYNQv2.7版本，但烧入这两个版本的镜像后始终无法与电脑连接，我们猜测是官网上的镜像配置并不完全。因而我们去网上找到了一个配置较为完全的镜像，烧入后发现可以与电脑连接，因而我们最终选择了PYNQv2.4版本的镜像。

# 总结

本次大作业我们完成了条形码及二维码检测与解码、人脸检测、基于Lenet网络进行手写数字识别三项任务，其中前两项任务是用pynq实现的，而基于Lenet网络进行手写数字识别是先利用vivado HLS软件生成ip核，再将该ip核与其他ip核进行连接，导出bitstream，然后将该bitstream移植到pynq上运行。

在条形码及二维码检测与解码任务中，我们了解到了用于条码及二维码检测的一些库，而我们最终使用的是pyzbar库进行检测，可以检测到条码或者是二维码的位置（二维码是通过定位左上、右上、左下的三个方框来进行定位），并且解读其中的信息，对于条码可以读出其中的数字，而对于二维码可以读出其中的信息（即对应的网址）。

在人脸检测任务中我们利用opencv库并借助外置连接的摄像头，实现了人脸的实时检测，利用在开发板中部署的代码进行检测，然后将检测的结果通过socket传回给电脑，并在电脑上显示检测结果。

在基于Lenet网络进行手写数字识别的任务中，我们创建了ip核，生成了bitstream文件，最后又在PYNQ中利用overlay调用了我们自己生成的bitstream文件，深刻感受到了加速性能。

通过本次大作业，我们深入了解了pynq在嵌入式系统开发中的应用，PYNQ为嵌入式系统提供了强大的开发平台，通过其灵活性和可编程性，我们能够高效地利用硬件资源，加速特定任务的处理；此外我们还掌握了基于Ubuntu系统进行嵌入式开发的一些技巧，也了解到了不同架构的linux系统的差异以及linux系统和windows系统在开发上的差异；第三，我们还掌握了使用OpenCV等框架来进行图像处理或者是目标检测的方法，PYNQ与OpenCV的结合使得在嵌入式系统中可以进行高效的图像处理和计算机视觉应用；最后，通过自己创建ip核与bitstream，深刻感受到了软硬件结合，以及硬件本身优化所带来的强大的任务处理加速效果。