# 一、新开一个ubuntu虚拟机建议200G

本节将在VMware虚拟机中安装ubuntu操作系统，主要步骤包括：

（1）在Windows 10操作系统桌面上，找到并双击VMware Workstation Pro图标；或者在Windows 10操作系统桌面左下角选择开始，出现浮动菜单。在浮动菜单下，找到并选择VMware->VMware Workstation Pro，启动VMware Workstation工具。

（2）VMware Workstation主界面，如图11.21所示。在该界面中，通过下面的一种方式创建新的虚拟机：

单击图中的创建新的虚拟机按钮；

在主界面主菜单中，选择文件->创建虚拟机；

（3）弹出新建虚拟机向导-欢迎使用新建虚拟机向导对话框界面，如图11.22所示。在该界面中，勾选“典型（推荐）（T）”前面的复选框。

（4）单击“下一步“按钮。

（5）弹出新建虚拟机向导-安装客户机操作系统对话框界面，如图11.23所示。在该界面中，按如下设置参数：

勾选“安装程序光盘映像文件(iso)(M):”前面的复选框；

并点击该选项后面的“浏览(R)...”，选择之前下载的ubuntu-16.04.7-desktop-amd64.iso文件。

（6）单击“下一步”按钮。

（7）弹出新建虚拟机向导-简易安装信息对话框界面，如图11.24所示。在该界面中，按如下设置参数：

在用户名右侧的文本框中输入名字；

在密码右侧的文本框中输入密码；

在确认右侧的文本框中再次输入相同的密码；

注：读者根据自己的情况设置用户名和密码（建议用户名和密码均不要使用中文名字）。

（8）单击“下一步”按钮。

（9）弹出新建虚拟机向导-命名虚拟机对话框界面，如图11.25所示。在该界面中，按如下设置参数：

标题“虚拟机名称(V)”下的文本框中给出默认的虚拟机名字为Ubuntu 64位；

在标题“位置(L)：”下的文本框中输入路径D:\Ubuntu64。

注：读者可以根据自己的情况选择虚拟机的路径以及虚拟机的名字。

读者可以通过单击浏览(R)按钮，设置位置。具体位置可由读者根据自己的具体情况指定。

（10）单击“下一步”按钮。

（11）弹出新建虚拟机向导-指定磁盘容量对话框界面，如图11.26所示。在该界面中，按如下设置参数：

最大磁盘大小(GB)(S)：200.0；

勾选“将虚拟磁盘拆分为多个文件(M)”前面的复选框；

（12）单击”下一步按钮。

（13）弹出新建虚拟机向导-已准备好创建虚拟机对话框界面，如图11.27所示。

（13）单击“完成”按钮，完成创建新虚拟机的过程。

（14）稍等片刻后，VMware软件会自动进入Ubuntu的安装过程，用户无需操作，等待安装完成即可。安装界面如图11.28所示。

注：自动安装过程需要联网，VMware会自动为虚拟机安装VMware Tools。

（15）安装完成后，如图11.29所示。

（16）输入密码，并按下回车，即可进入操作系统。如图11.30所示。

（17）至此，完成在虚拟机上安装Ubuntu操作系统的过程。

## 附1：设置共享文件夹

本节将设置共享文件夹，并通过共享文件夹将Windows 10操作系统指定文件夹中的文件复制到Ubuntu操作系统的指定文件夹中。主要步骤包括：

1）在VMware Workstation主界面主菜单中，选择虚拟机->设置(S)...，如图11.31所示。

2）弹出虚拟机设置对话框界面，如图11.32所示。在该界面中，单击“选项”标签。在该标签界面中，按如下设置参数：

（1）在右侧窗口的文件夹共享标题栏中，勾选总是启用(E)前面的复选框。

（2）在右侧窗口的文件夹标题栏中，单击“添加(A)...”按钮。

（3）弹出共享文件夹向导-欢迎使用添加文件夹向导对话框界面。

（4）单击“下一步”按钮。

（5）弹出添加共享文件夹向导-命名共享文件夹对话框界面。在该界面中，按如下设置参数：

单击主机路径(H)右侧的“浏览(R)...“按钮。弹出”浏览文件夹“对话框界面。在该对话框界面中，将路径定位到D:\loongson\_linux\_file。单击”确定“按钮。

默认的名称loongson\_linux\_file（读者可以自己修改名字）。

弹出添加共享文件夹向导-指定共享文件夹属性对话框界面。在该界面中，勾选启动此共享(E)前面的复选框。

单击“完成”按钮，退出该对话框界面。

（6）在文件夹共享标题栏下，添加完共享文件夹后的界面，如图11.33所示。

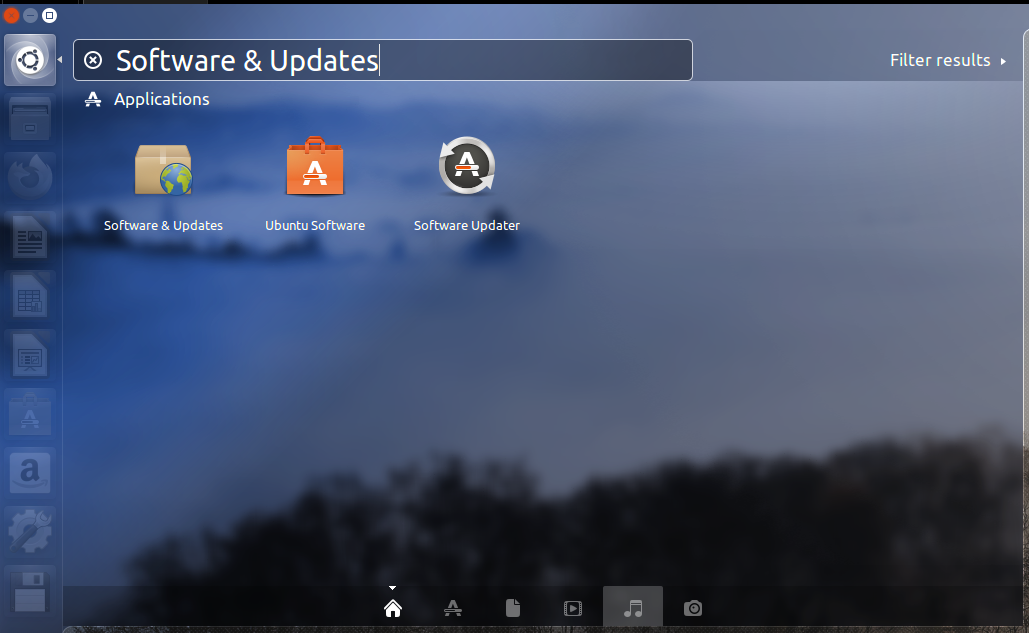
（7）单击图11.32中的“确定”按钮，退出虚拟机设置对话框界面。

（8）如图11.34所示，在Ubuntu操作系统桌面左侧一列窗口中，找到并单击“Files”条项，弹出Files界面。在该界面右侧窗口中，单击Computer条项。在展开的文件夹下，找到并单击mnt子文件夹图标。在进入该文件夹后，找到并单击hgfs子文件夹图标。

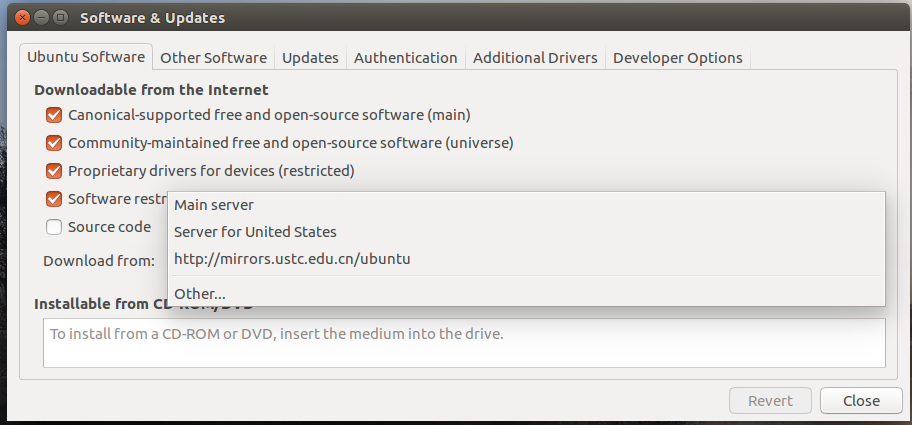
## 附2：设置下载源

1. 首先ping 一下[www.baidu.com](http://www.baidu.com)，检查Linux访问国内网站速度与Windos是否一致。如果不一致就将网络从Net模式改成桥接模式。
2. Ubantu默认的下载源是国外的，需要改成国内的下载源。

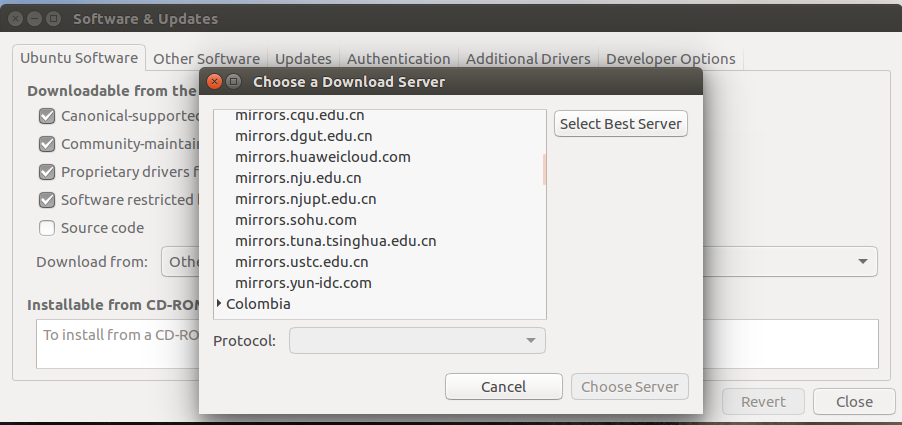
首先点击左侧任务栏第一个，进入搜索栏，搜索Software & updates，如下图所示



如下图所示，Download from 默认的下载源是国外的。



如下图所示，点击更换下载源，点other，向上找china，选择mirrors.ustc中科大源的下载源，再输入次密码确认，完成更换。（如果按照下面步骤操作完重启后还是慢，就再更换aliyun的下载源重新操作一次）

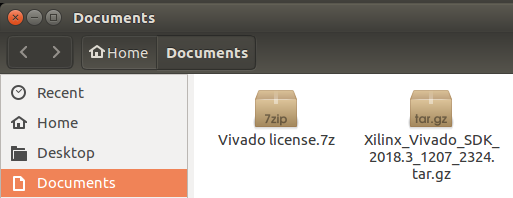


点击close关闭，弹出窗口确认更换软件源，选择Reload，然后等待系统更新缓存，完成后最好重启一次虚拟机。

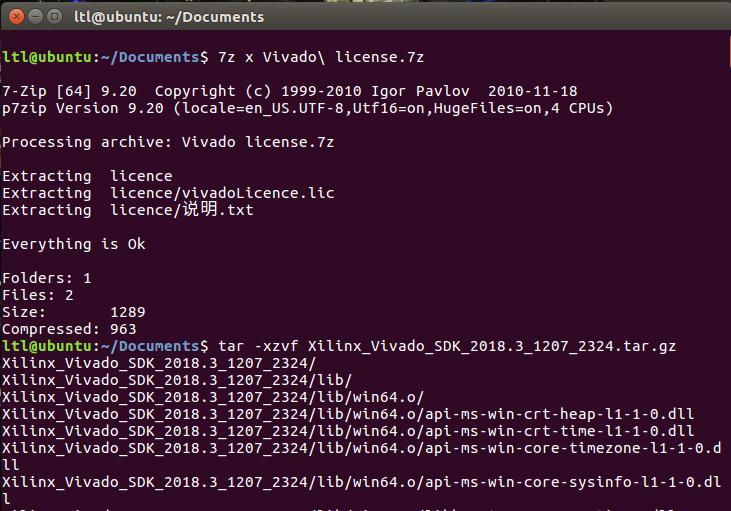
# 二、在虚拟机下安装vivado

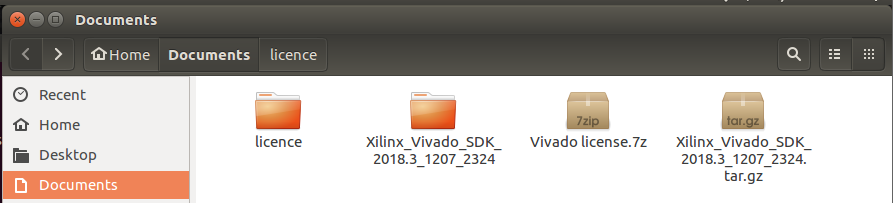
将安装包与License放到共享文件夹。

在虚拟机下，从/mnt/hgfs/Shared\_Folders文件夹下复制压缩包到Documents/。

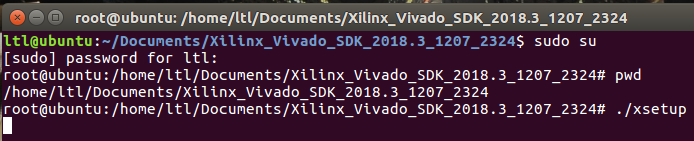


解压出vivado与License。

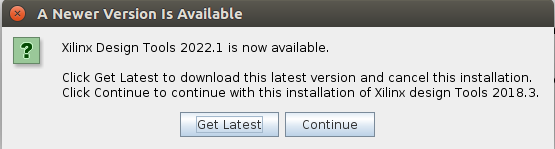


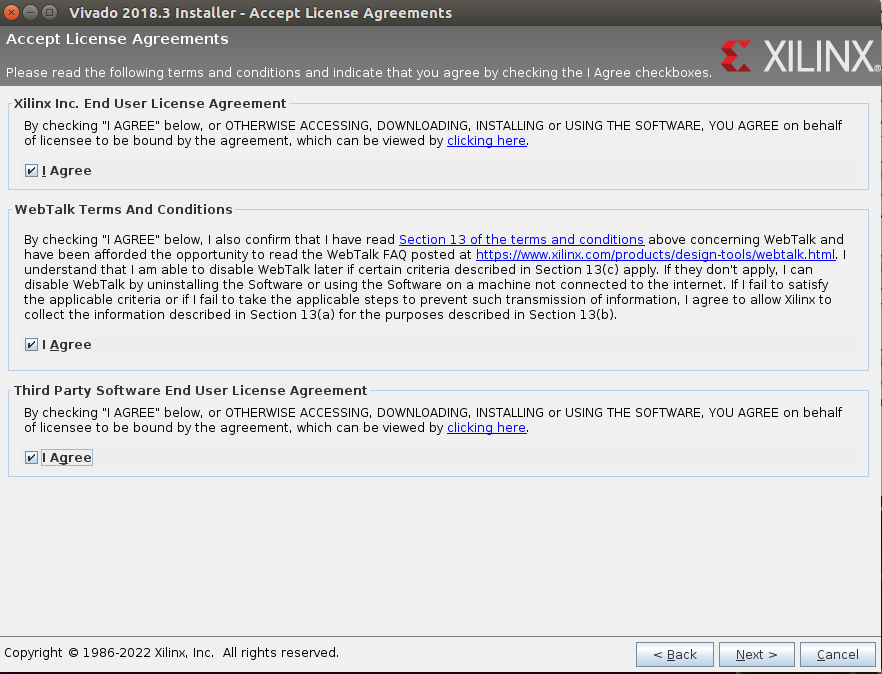


超级用户权限——运行安装程序。

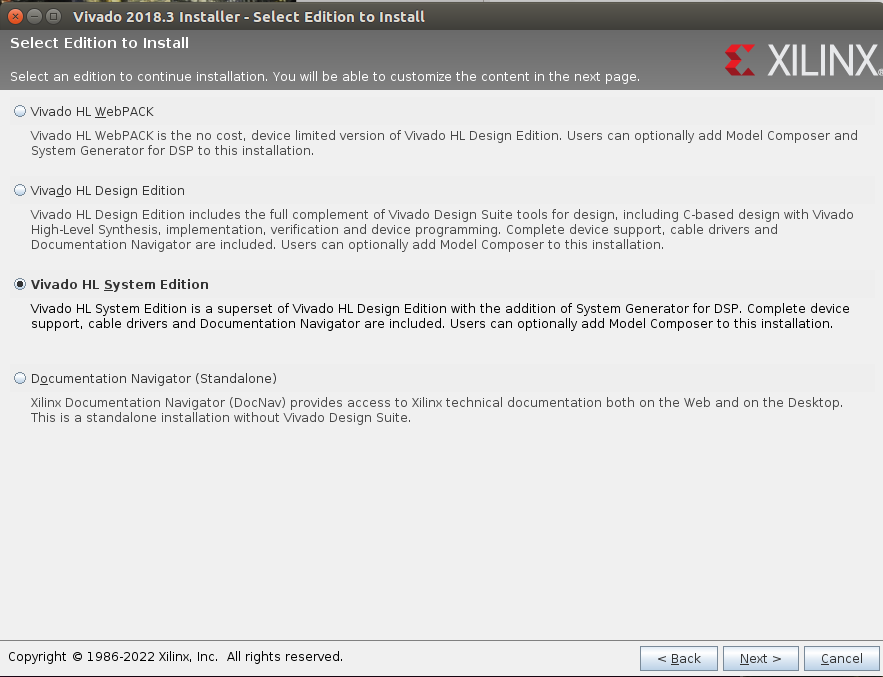


选择Continue、Next、全部同意Next

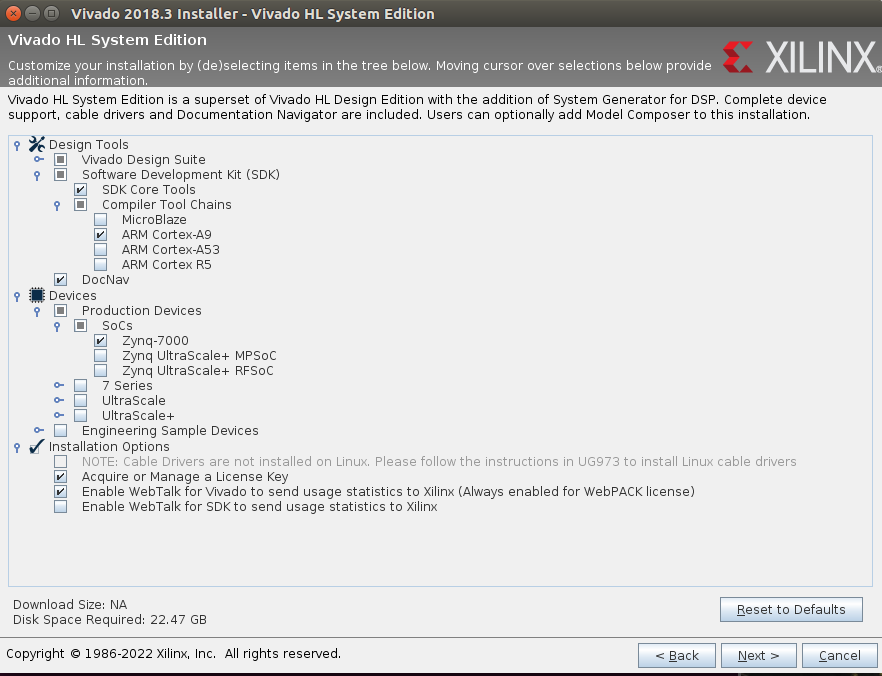




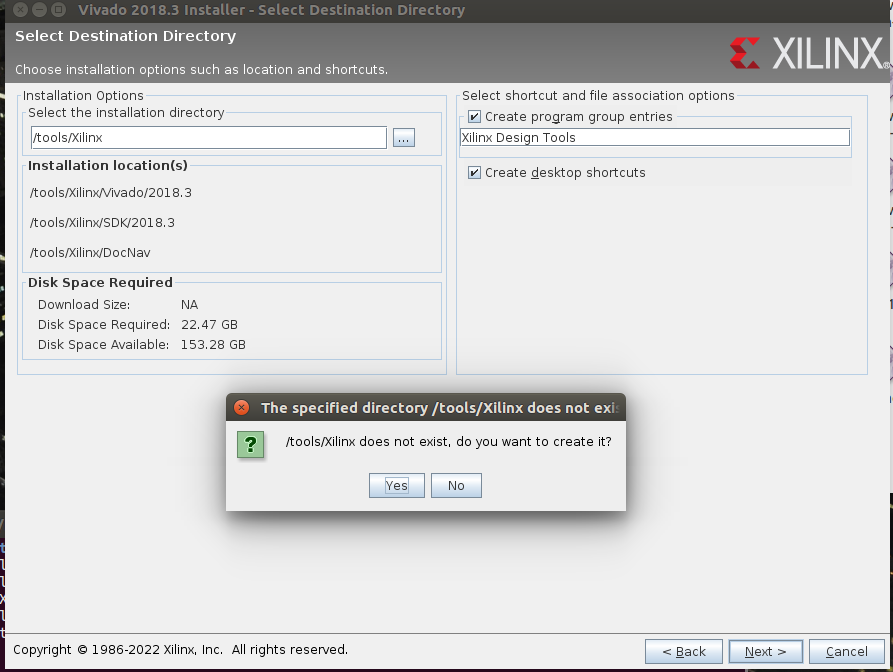
选择第三项，Next。



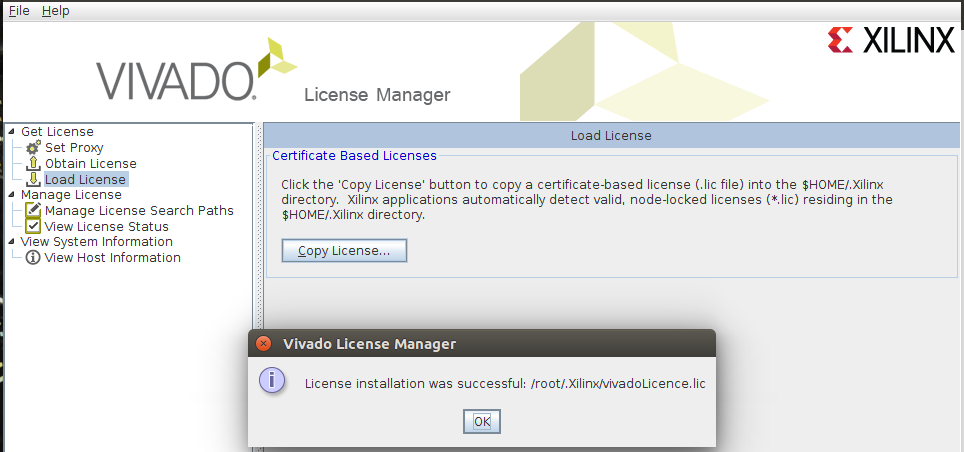
选择需要的组件：

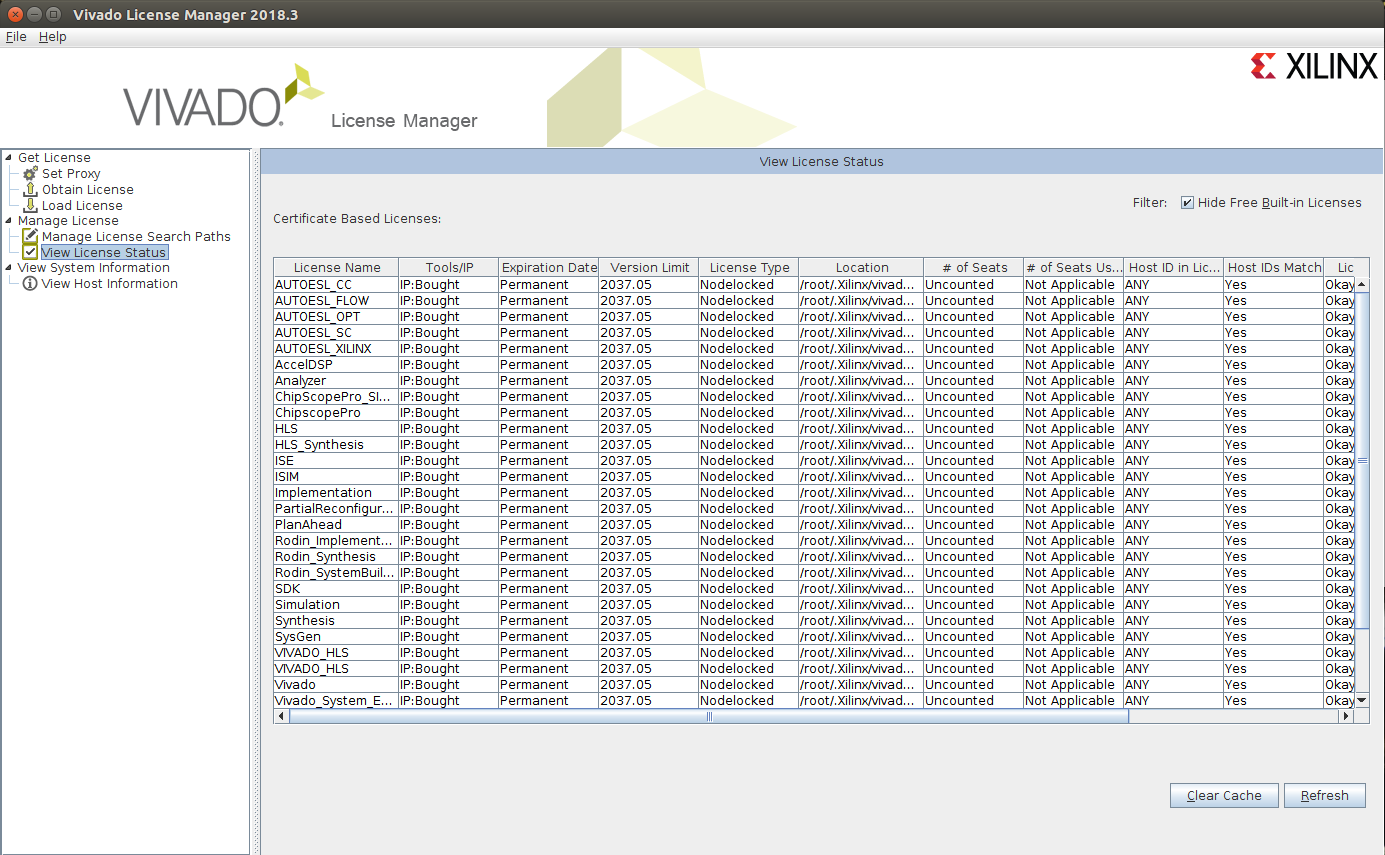


选择安装位置。

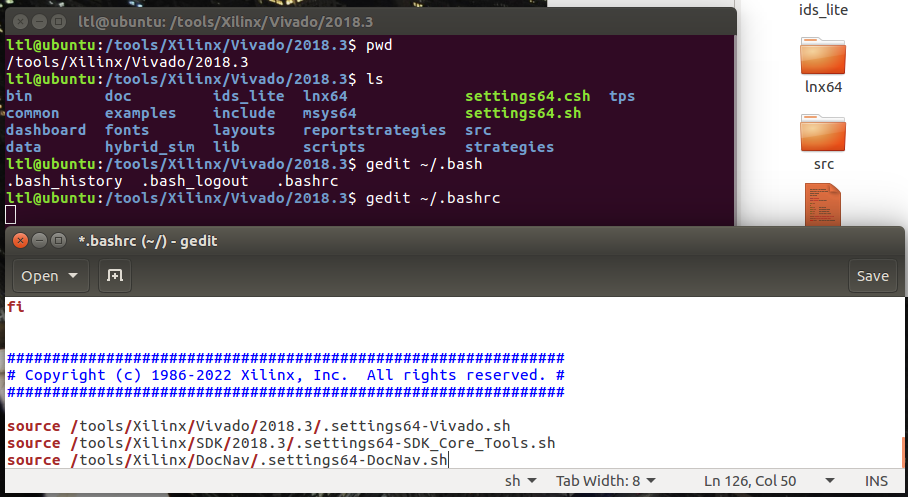


添加license.点击load license,导入。





将vivado/2018.3下面的settings64.sh里的代码复制粘贴道bashrc的文件里面



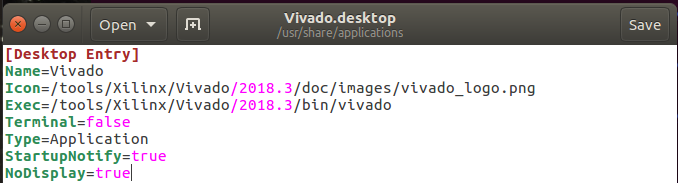
用source ~/.bashrc命令刷新配置，即可在命令行输入vivado启动。

## 附1：配置快捷方式

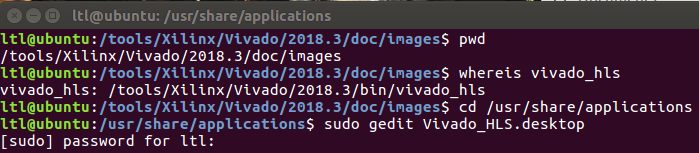
手动创建 Vivado.desktop 到 /usr/share/applications 中。



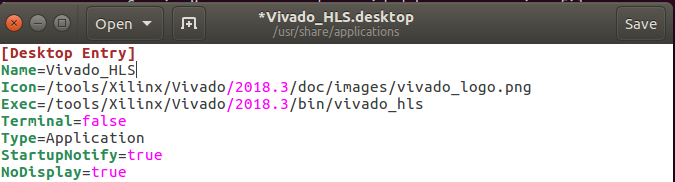
添加描述：



手动创建 Vivado\_HLS.desktop 到 /usr/share/applications 中。



添加描述：



将/usr/share/applications中生成的两个图标复制到桌面。

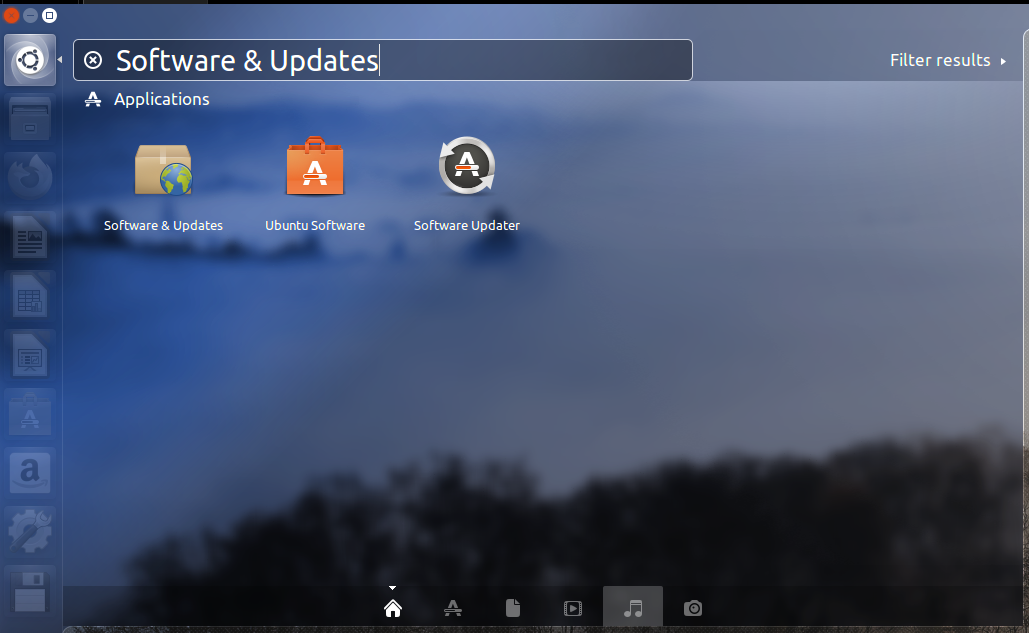


# 三、下载petalinux2018.3

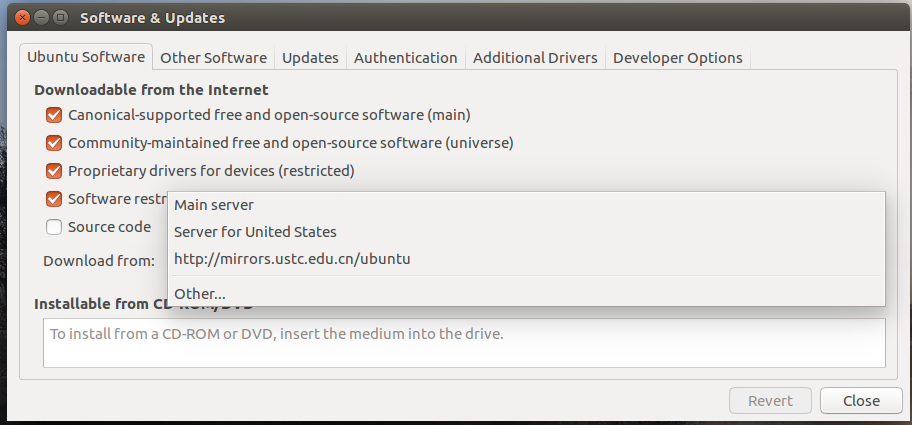
## 1、更换下载源

Ubantu默认的下载源是国外的，需要改成国内的下载源。

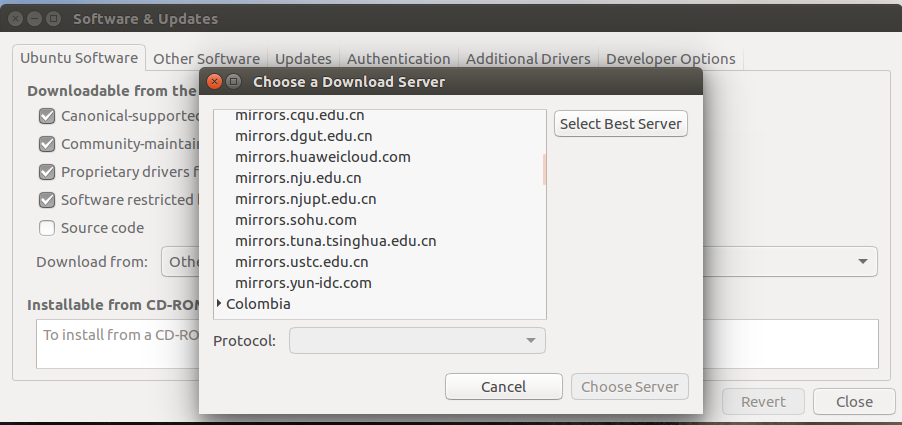
首先点击左侧任务栏第一个，进入搜索栏，搜索Software & updates，如下图所示



如下图所示，Download from 默认的下载源是国外的。



如下图所示，点击更换下载源，点other，向上找china，选择mirrors.ustc中科大源的下载源，再输入次密码确认，完成更换。（如果按照下面步骤操作完重启后还是慢，就再更换aliyun的下载源重新操作一次）



点击close关闭，弹出窗口确认更换软件源，选择Reload，然后等待系统更新缓存。

然后打开终端输入（更新一下源）：

**sudo apt-get update**

**sudo apt-get upgrade 时间比较长**

完成后最好重启一次虚拟机。

## 2、下载需要的包（超级用户）

sudo apt-get install -y gcc git make net-tools libncurses5-dev tftpd zlib1g-dev libssl-dev

sudo apt-get install -y flex bison libselinux1 gnupg wget diffstat chrpath socat xterm autoconf libtool tar unzip

sudo apt-get install -y texinfo zlib1g-dev gcc-multilib build-essential libsdl1.2-dev libglib2.0-dev zlib1g:i386

sudo apt-get install -y screen pax gzip

sudo apt-get install gawk

sudo apt install -y tftpd tftp openbsd-inetd

sudo apt install cobbler

sudo apt-get install chrpath

sudo apt-get install socat

sudo apt install autoconf

sudo apt-get install libtool

sudo apt-get install texinfo

sudo apt-get install gcc-multilib

sudo apt-get install libsdl1.2-dev

sudo apt-get install libglib2.0-dev

sudo apt-get install zlib1g:i386

**配置TFTP：**

sudo gedit /etc/inetd.conf

在文件中增加以下内容

tftp dgram udp wait nobody /usr/sbin/tcpd /usr/sbin/in.tftpd /tftproot

保存并退出

sudo mkdir /tftproot

sudo chmod 777 /tftproot

sudo /etc/init.d/openbsd-inetd restart

netstat -an | more | grep udp

看到有如下输出，则表明TFTP安装成功



## 3、下载软件并安装

创建账户Ltl991015. 下载安装包（6.8G）：

https://www.xilinx.com/member/forms/download/xef.html?filename=petalinux-v2018.3-final-installer.run

将安装包导入共享文件（上面有设置共享文件夹路径）

如果共享文件夹设置无误但不显示，如下处理：

vmware-hgfsclient #输出共享文件夹的名字

sudo apt-get install open-vm-tools #安装命令包

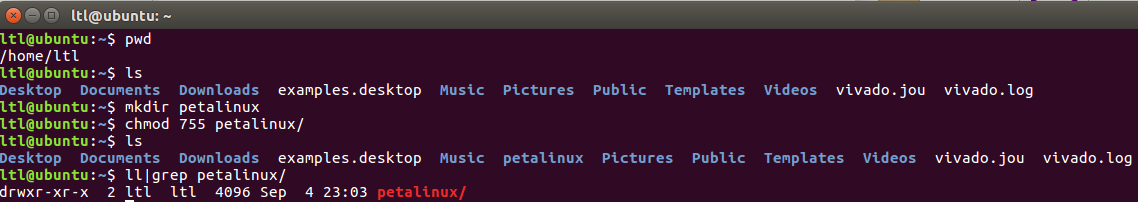
sudo vmhgfs-fuse .host:/ /mnt/hgfs -o allow\_other #挂载

但重启之后会失效，每次需挂载一次。解决重启后共享文件夹失效：

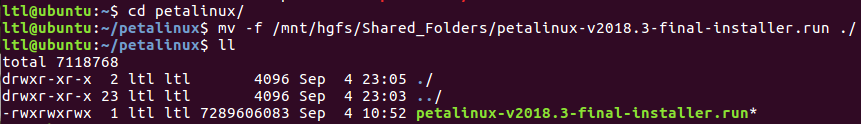
sudo gedit /etc/fstab #并编辑，在最后添加如下内容：

.host:/ /mnt/hgfs fuse.vmhgfs-fuse allow\_other 0 0

创建安装路径（普通用户路径下）：

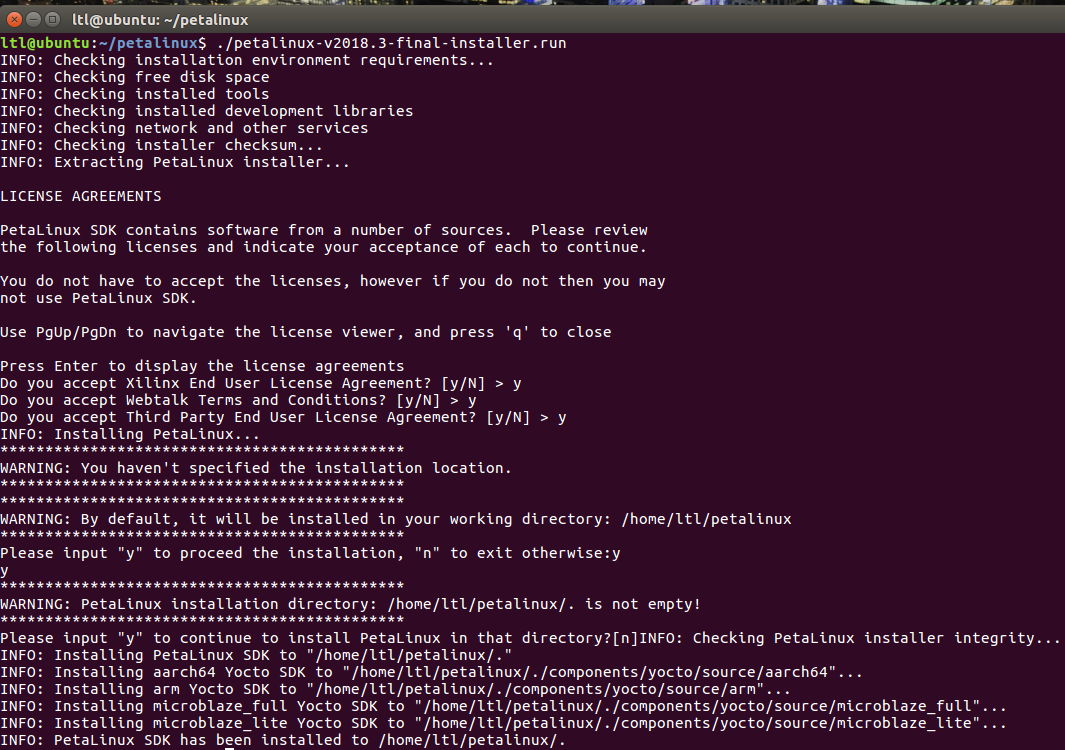


将安装包复制到安装目录下：



运行该程序./petalinux-v2018.3-final-installer.run，执行安装。

协议部分：回车开始查看，q结束查看，y同意协议。（n不同意协议-无法安装）。



## 5、配置软件环境

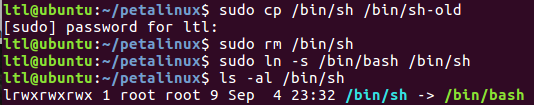
Ubuntu默认shell是dash，用sudo dpkg-reconfigure dash将dash切换为bash。

sudo cp /bin/sh /bin/sh-old

sudo rm /bin/sh

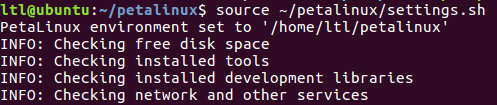
sudo ln -s /bin/bash /bin/sh

ls -al /bin/sh



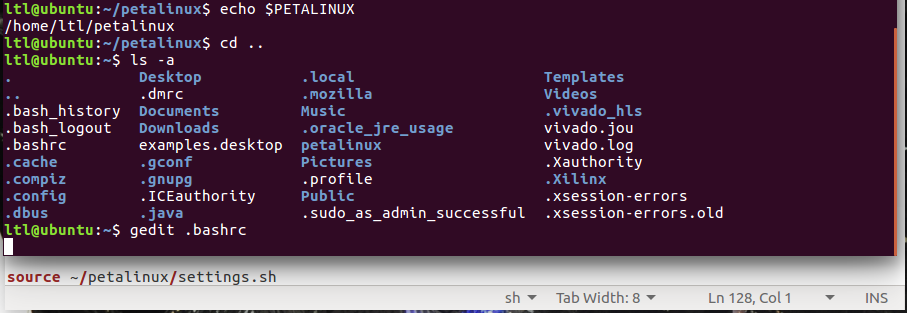
执行source命令，因为我安装的位置是home/ltl目录下，所以在这可以找到settings.sh文件。（linux中.sh文件是脚本文件，一般都是bash脚本）

source ~/petalinux/settings.sh



输入**echo $PETALINUX** 可以看到软件安装路径。

用 **ls -a** 可以看到在 ~ 目录下存在隐藏的终端初始化脚本文件**.bashrc**。修改这个文件，在最后加上 **source ~/petalinux/settings.sh** ，把环境变量配置脚本加在里面，就能在打开的时候自动配置环境了。



## 附加：该工具一般设计流程

通过 Vivado 创建硬件平台，得到 hdf 硬件描述文件；

运行 source <petalinux 安装路径>/settings.sh，设置 Petalinux 运行环境

通过 petalinux-create -t project 创建 petalinux 工程；

使用 petalinux-config --get-hw-description， 将 hdf 文件导入到 petalinux 工程当中并配置 petalinux 工程；

使用 petalinux-config -c kernel 配置 Linux 内核；

使用 petalinux-config -c rootfs 配置 Linux 根文件系统；

配置设备树文件；

使用 petalinux-build 编译整个工程；

使用 petalinux-package --boot 制作 BOOT.BIN 启动文件；

制作 SD 启动卡，将 BOOT.BIN 和 image.ub 以及根文件系统部署到 SD 卡中；

将 SD 卡插入开发板，并将开发板启动模式设置为从 SD 卡启动；

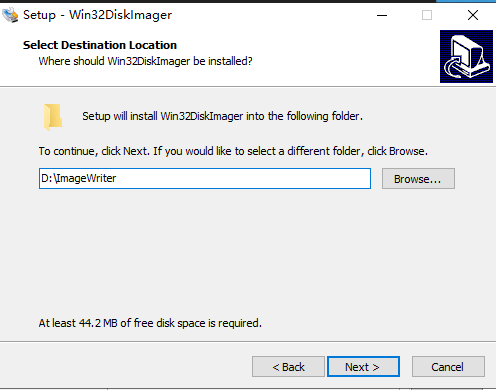
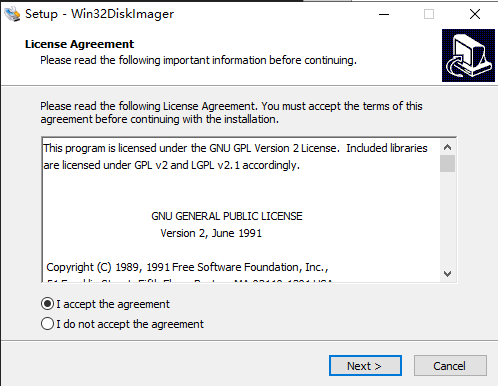
开发板连接串口线并上电启动，串口上位机打印启动信息， 登录进入 Linux 系统。

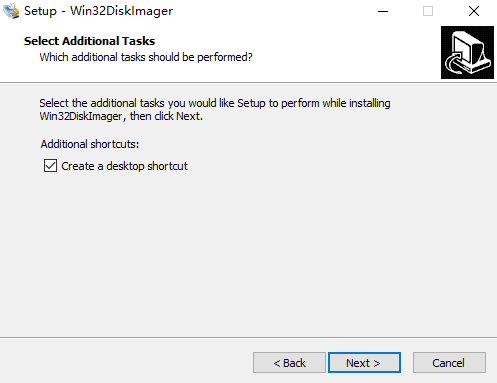
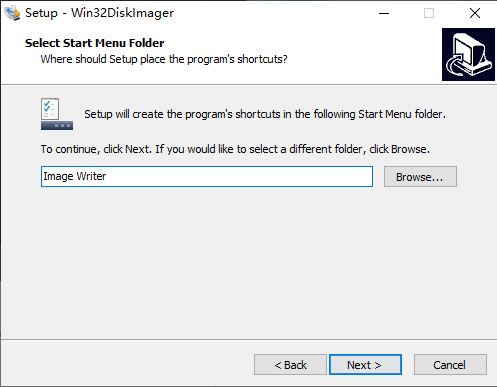
# 四、制作PYNQ V2.4镜像

## 1、准备工作

★Windos下载**Win32DiskImager**工具。下载链接：

<https://www.aliyundrive.com/s/NXfamShJt1T>





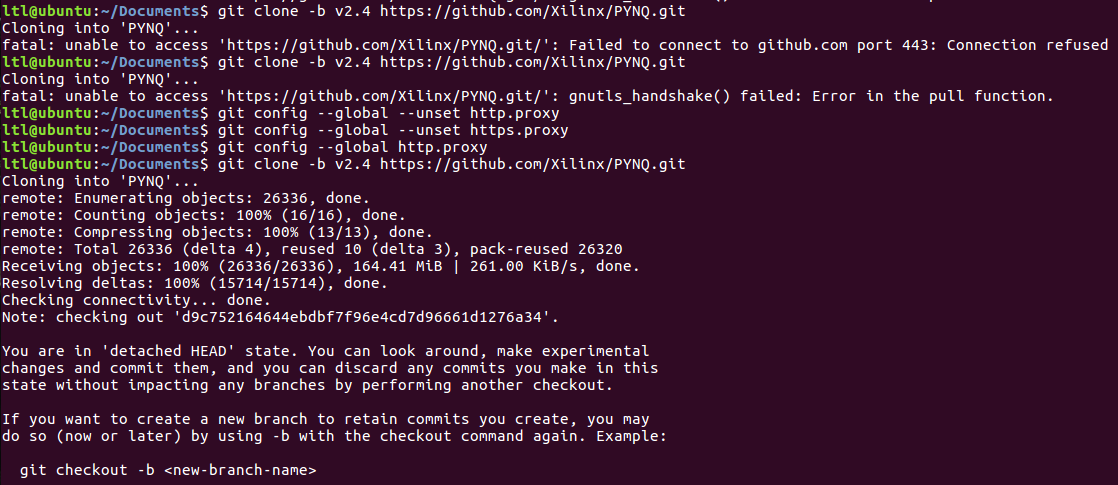
★下载**pynq\_rootfs\_arm\_v2.4.zip**文件系统备用，并存放到共享文件夹。下载地址：

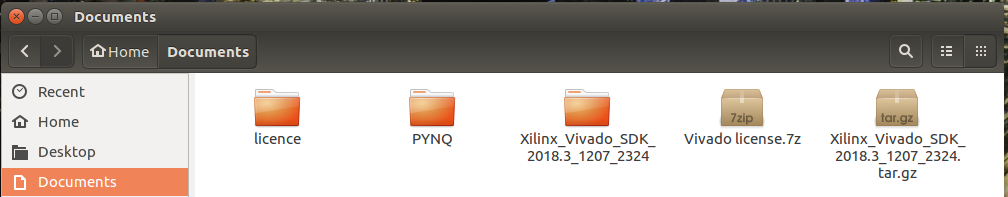
https://www.xilinx.com/member/forms/download/xef.html?filename=pynq\_rootfs\_arm\_v2.4.zip

★使用命令 **sudo apt install git-all**安装git。设置用户名**git config --global user.name litianling**和邮箱地址**git config --global user.email [2467280868@qq.com](mailto:2467280868@qq.com)**检查**git config --list**。

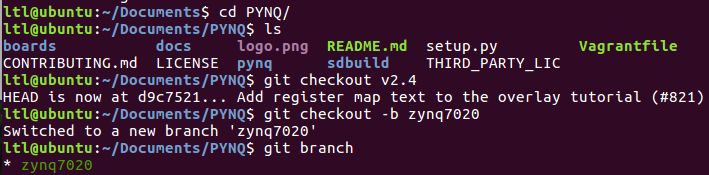
★在虚拟机Documents目录中下载pynq备用（如果出错就按图片处理）：

**git clone -b v2.4 https://github.com/Xilinx/PYNQ.git**

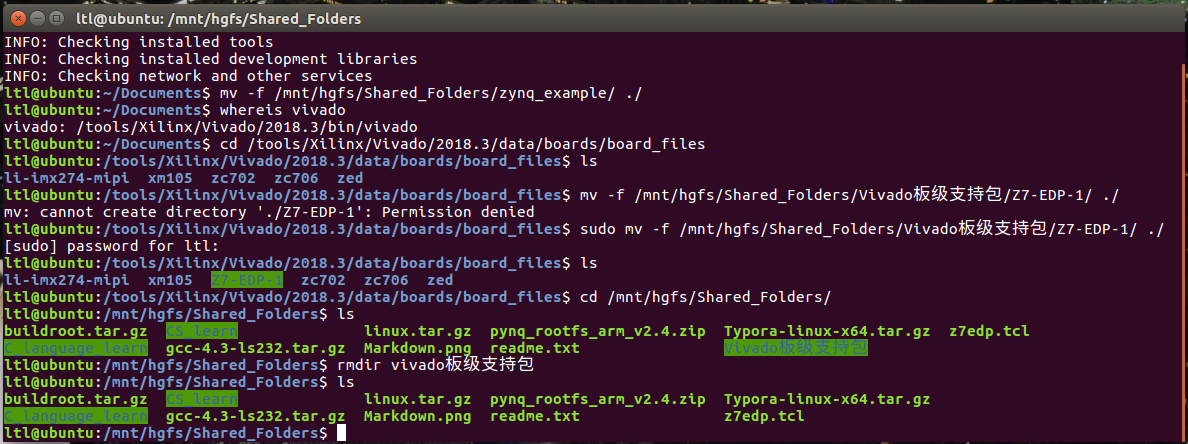




★配置pynq：用**cd PYNQ**进入目录，从v2.4分支创建一个新的分支，名为zynq7020。

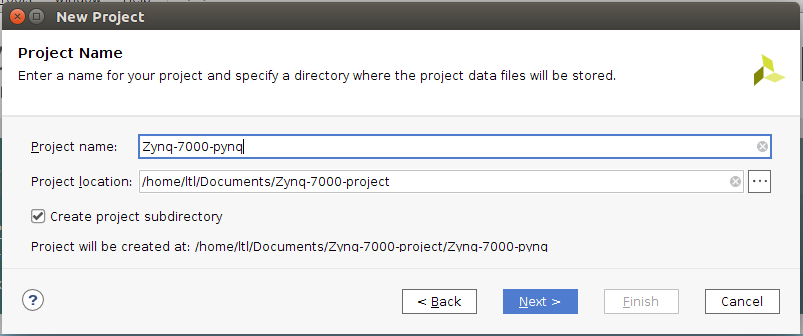


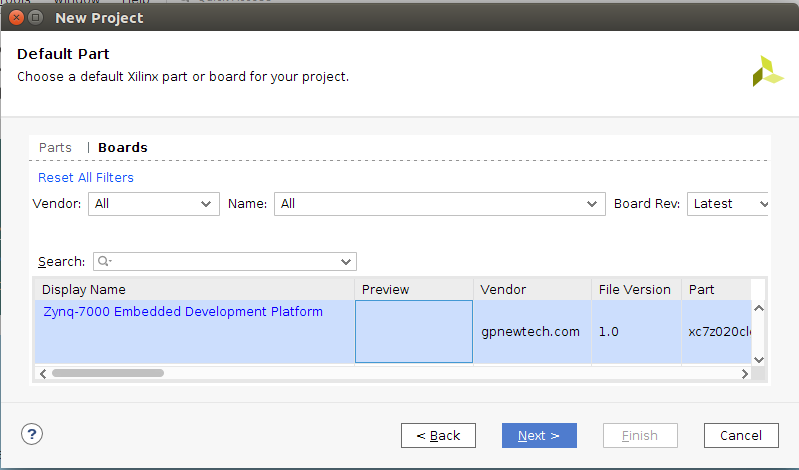
★在<http://www.edawiki.com/index.php?category-view-68.htm>下载例程与板级支持包。将其资源传入共享文件夹。将共享文件夹下例程移动到Documents，将共享文件夹下Vivado板级支持包中Z7-EDP-1文件夹移动到/tools/Xilinx/Vivado/2018.3/data/boards/board\_files，删除共享文件夹下Vivado板级支持包文件夹。



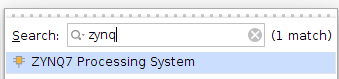
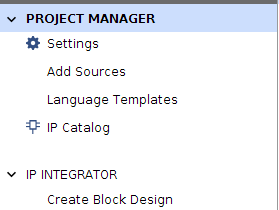
## 2、创建Vivado工程，生成.bit文件和.hdf文件

创建工程：

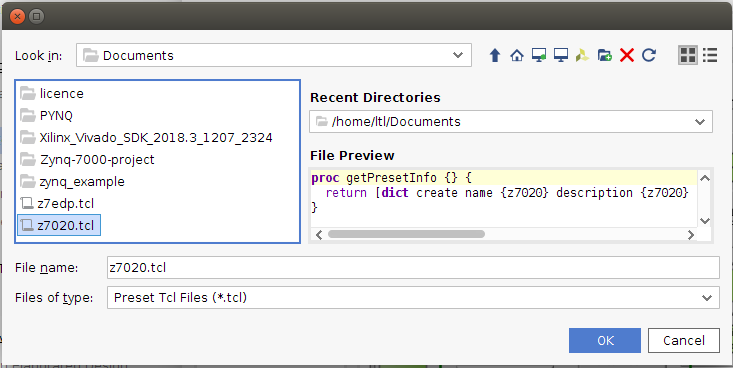




创建IP核：



导入默认设置：



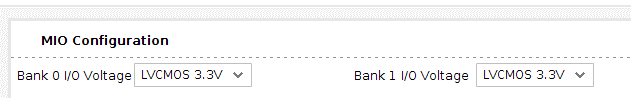
双击IP核，MIO勾选配置如下：

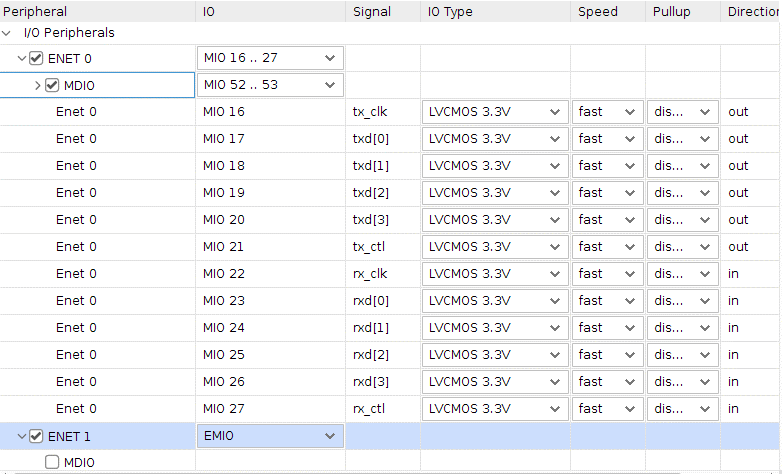
ENET0(MIO16…27)，下拉勾选MDIO(MIO52…53)，Speed选项选择"fast"

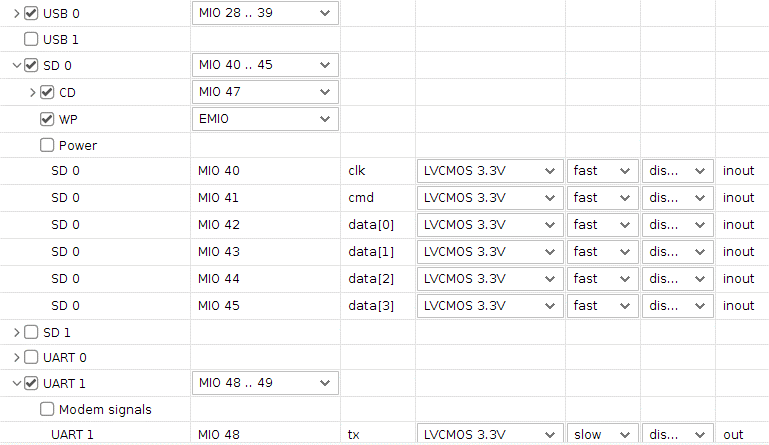
ENET1选择EMIO**，USB0(28…39)**，UART1(MIO48…49)，

SD0(MIO40…45)，**下拉框CD(MIO47)和WP(EMIO)防止SD卡只读造成文件系统无法挂载**

**Bank1 I/O Voltage选择 “LVCMOS 3.3V”**（根据开发板原理图选择）

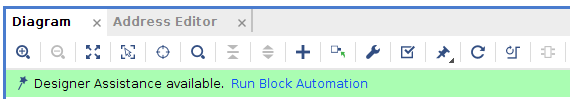


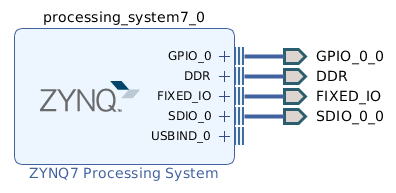




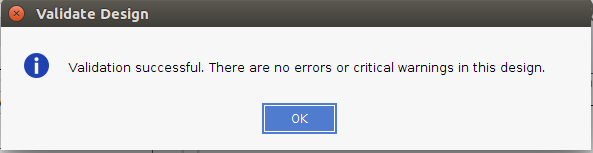
自动布线/手动布线结果如下：

右键单击“GPIO\_0+“和“SDIO\_0”，选择”Make External“添加外部引脚。

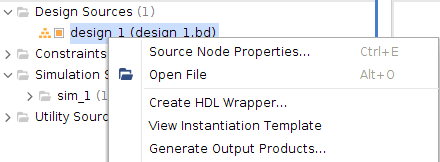




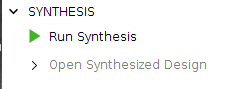
最后用快捷键F6验证Block Design，没问题的话保存。



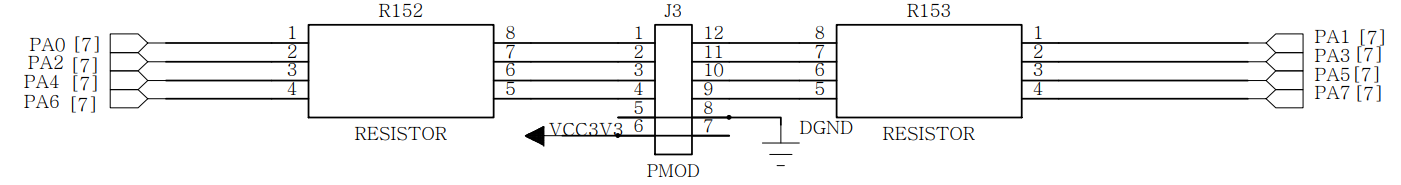
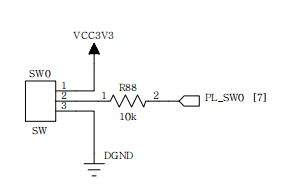
找到源文件中的.bd文件执行"Generate Output Products"、“Create HDL Wrapper”。

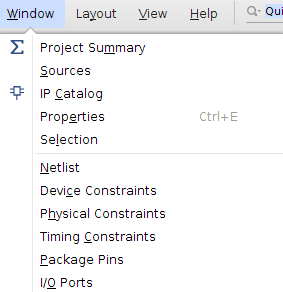
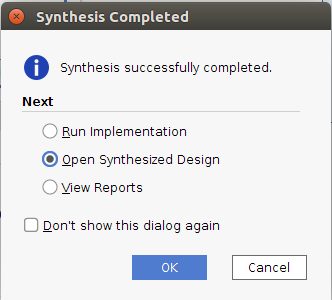
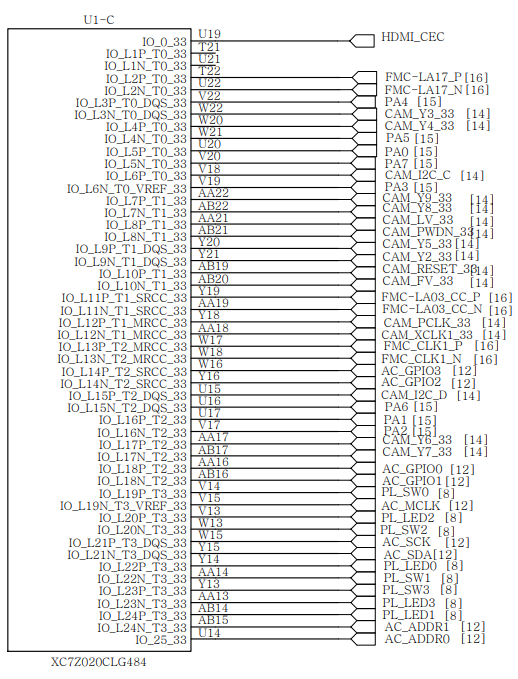


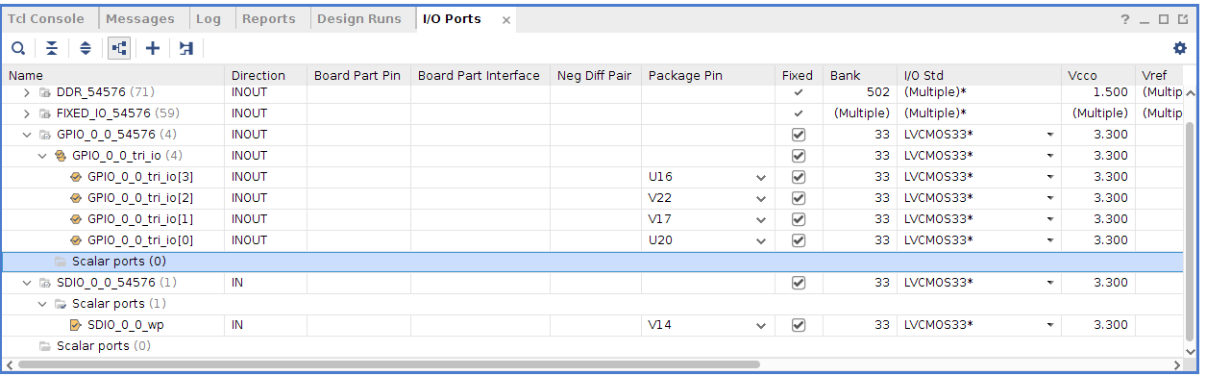
综合：



打开综合后的设计添加引脚约束：



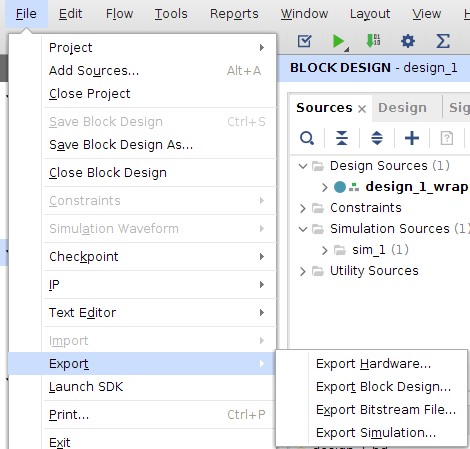


按下“Ctrl+S”保存为“zynq7020-1”文件

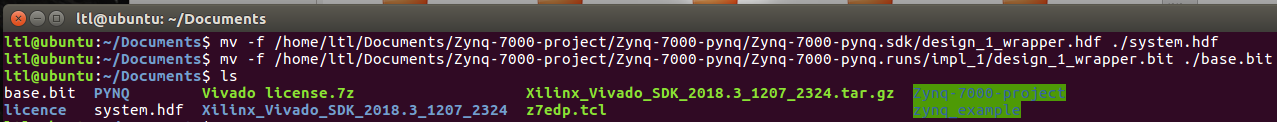
生成比特流文件：

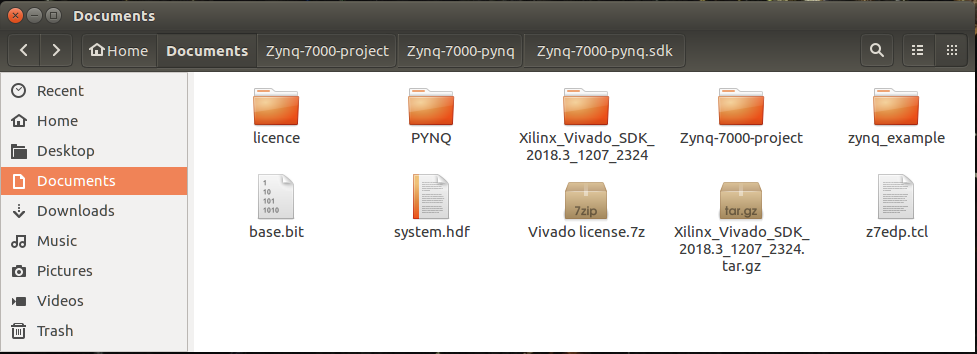


最后导出Hardware和Bitstream。



这样就得到了我们需要的.bit文件和.hdf文件了，在Vivado工程\*\*.sdk目录和\*\*.runs/imp\_1目录下。分别重命名为system.hdf和base.bit，复制到Docunments备用。

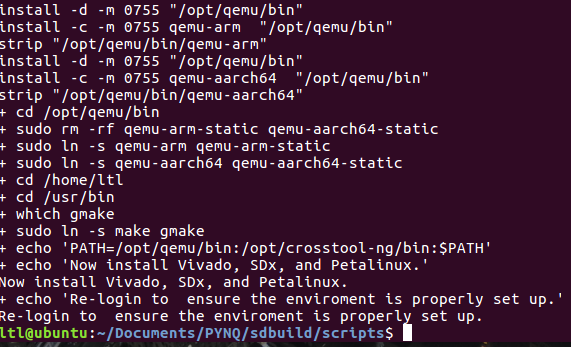


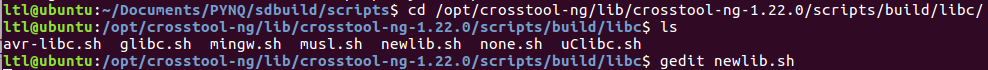


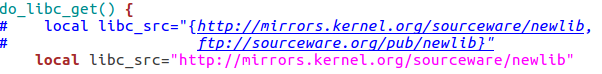
## 3、配置PYNQ

**cd PYNQ/sdbuild/scripts/** 执行**./setup\_host.sh** 把qemu和crosstool-ng下载下来，默认下载到/opt 目录，（用超级用户权限执行该脚本）。

 如果出现下列dpkg报错，就将文件夹/var/lib/dpkg/info/备份，并创建一个新的空文件夹继续执行

下载好之后**cd /opt/crosstool-ng/lib/crosstool-ng-1.22.0/scripts/build/libc/**修改newlib.sh。找到local libc\_src=将后面的内容修改为如下内容：local libc\_src="http://mirrors.kernel.org/sourceware/newlib"





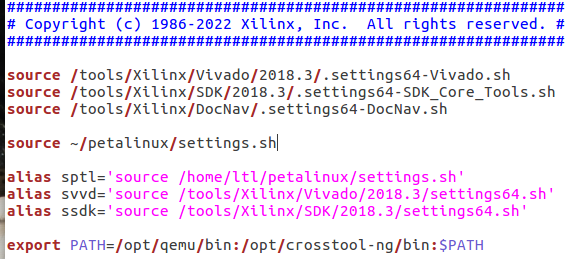
配置一下环境变量，**sudo gedit ~/.bashrc**在文末写入如下内容（**根据自己安装路径设**）：

**alias sptl='source /home/ltl/petalinux/settings.sh'**

**alias svvd='source /tools/Xilinx/Vivado/2018.3/settings64.sh'**

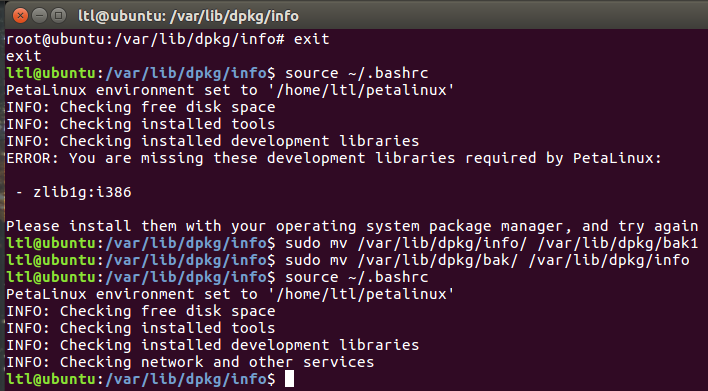
**alias ssdk='source /tools/Xilinx/SDK/2018.3/settings64.sh'**

**export PATH=/opt/qemu/bin:/opt/crosstool-ng/bin:$PATH**



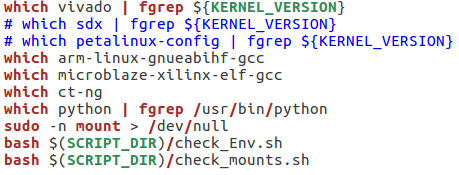
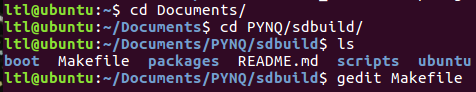
注意petalinux、Vivado、SDK的路径，使能环境变量。

**有错误，恢复info文件夹。**



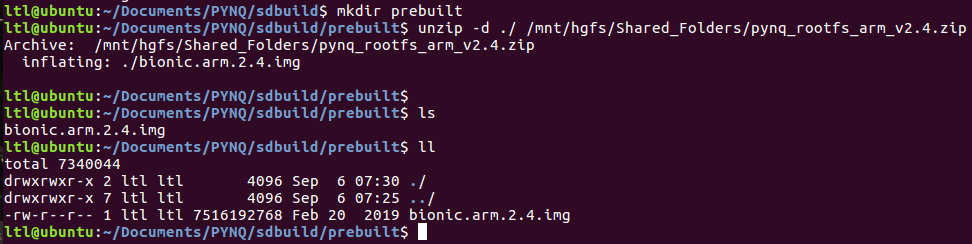
由于没有安装SDx和petalinux环境，所以**cd PYNQ/sdbuild/**，打开Makefile

找到并删除which sdx | fgrep ${KERNEL\_VERSION} 和下一行pata……



## 4、准备与编译

在PYNQ/sdbuild目录下新建文件夹prebuilt，将下载好的文件系统镜像bionic.arm.2.4.img解压至该目录。



在PYNQ/boards/目录下新建文件夹zynq7020，在zynq7020目录下再新建两个文件夹分别为base和petalinux\_bsp。将之前准备好的base.bit拷贝到base文件夹，在petalinux\_bsp 中再建立文件夹 hardware\_project，将system.hdf拷贝到hardware\_project文件夹。

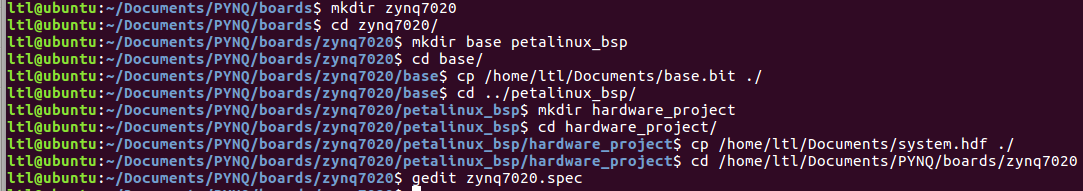
zynq7020目录下新建zynq7020.spec，并编辑以下内容，然后保存

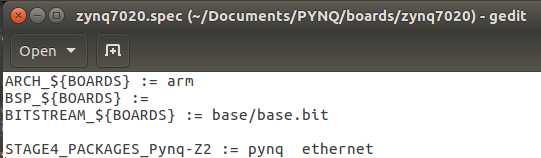
**ARCH\_${BOARDS} := arm**

**BSP\_${BOARDS} :=**

**BITSTREAM\_${BOARDS} := base/base.bit**

**STAGE4\_PACKAGES\_Pynq-Z2 := pynq ethernet**





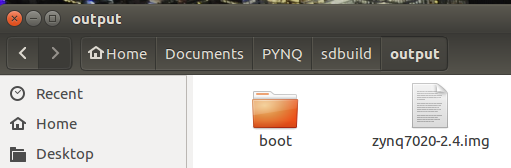
make之前先执行任意sudo操作，比如sudo ifconfig命令，否则make会报错。

**cd PYNQ/sdbuild**

**make BOARDS=zynq7020 PREBUILT=./prebuilt/bionic.arm.2.4.img**

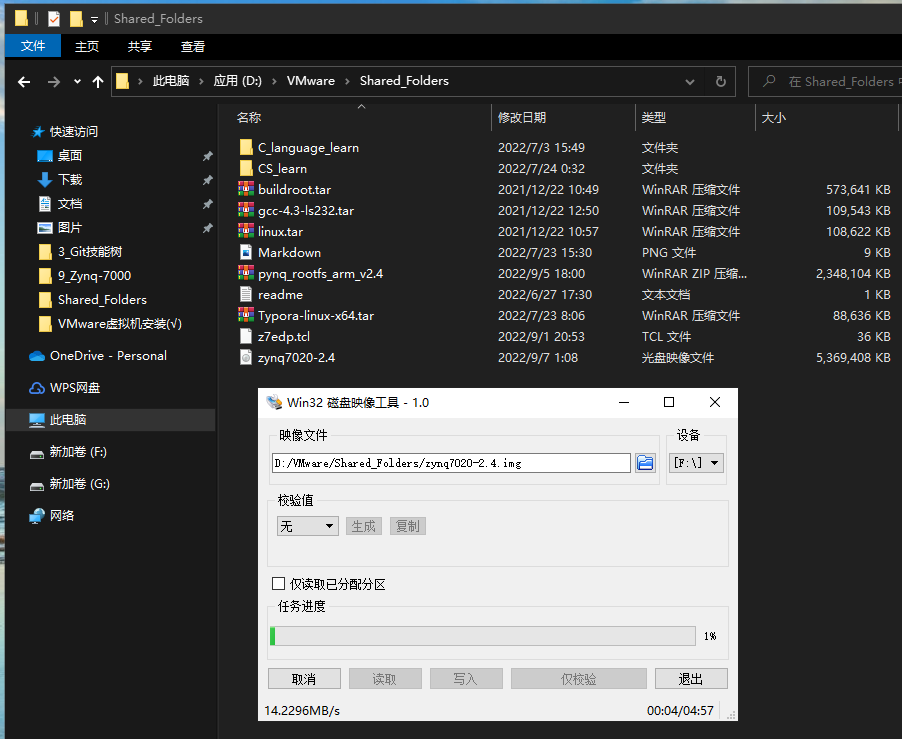
如果make发生报错，解决问题后先执行**make clean**，再重新make

大概编译了半小时就生成了镜像文件，在目录PYNQ/sdbuild/output。



## 5、烧写镜像启动开发板

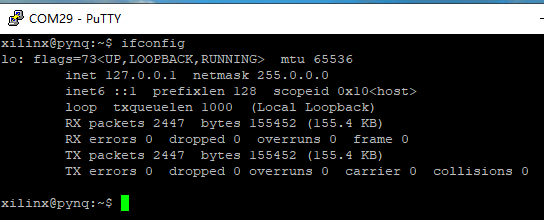
将生成的镜像zynq7010-2.4.img通过共享文件夹传送给Windos，使用Win32DiskImager将镜像烧写到SD卡上。软件会自动检测SD卡，选择好镜像后直接点击“写入”即可，大约10分钟写完SD卡中其实就包含两个文件 image.ub 和 BOOT.BIN



将SD卡插入开发板，设置开发板启动方式为从SD卡启动，插上电源，插上网线，连接串口，打开Putty，打开电源，查看串口。（挂载失败更改开关）

## 6、开发板网络配置

查看一下开发板网络，发现没有以太网的配置：



自己写一个以太网配置即可：

cd /etc/network/

sudo chmod 777 interfaces.d/ ##修改权限为777，**初始密码为xilinx**

cd interfaces.d/

vi eth0

在eth0输入以下内容：

auto eth0

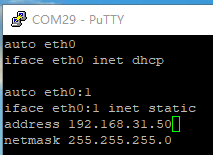
iface eth0 inet dhcp

auto eth0:1

iface eth0:1 inet static

address 192.168.31.50

netmask 255.255.255.0

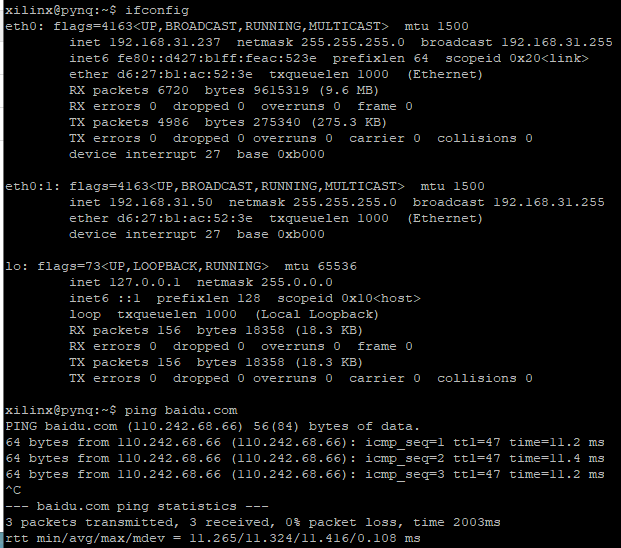


保存并退出：

cd /etc/network/

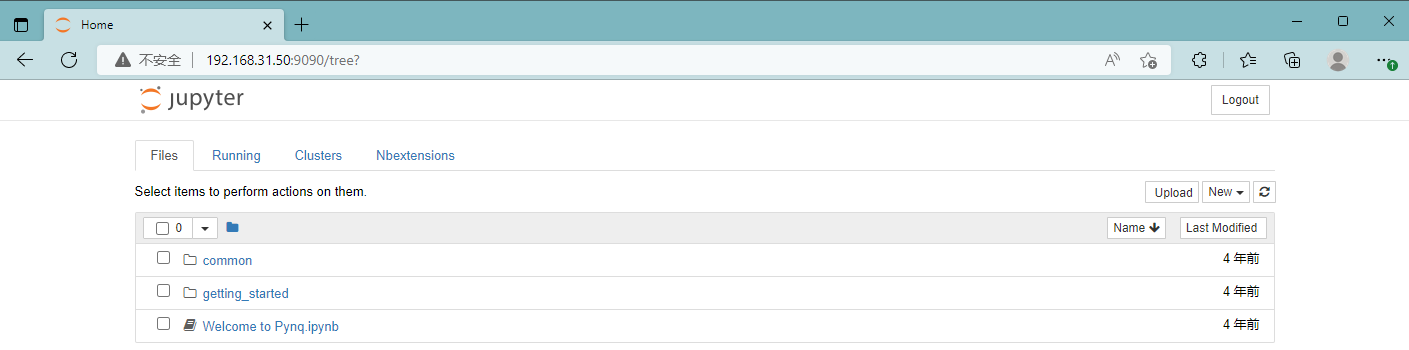
chmod 755 interfaces.d/ ##把权限再修改回来

重启开发板查看网络配置：



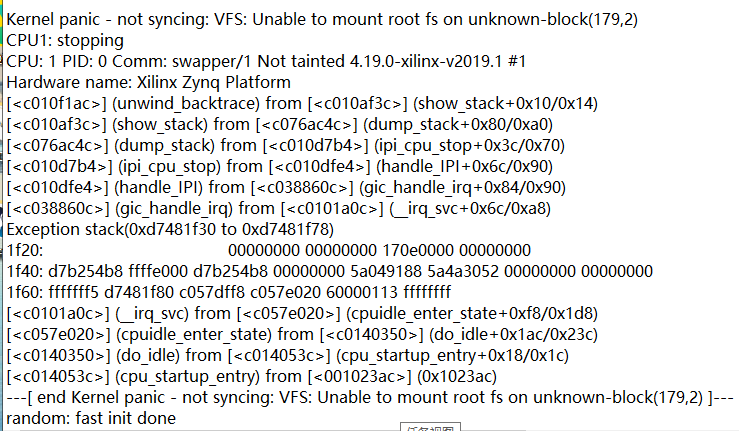
## 7、打开Jupyter Notebook

备注：相当于ZYNQ是个服务器，PC机输其端口号即可连接，我的端口是：192.168.31.50：9090



## 附加、SD卡只读造成无法挂载文件系统

问题：



原因：



解决：

<https://blog.csdn.net/Chitanda_Eru_/article/details/112069134?spm=1001.2014.3001.5502>

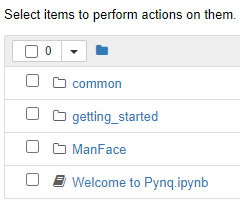
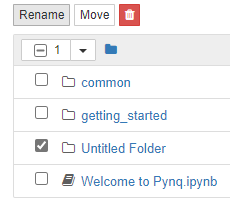
# 五、搭建人脸识别系统

## 5.1 构建数据库与测试库

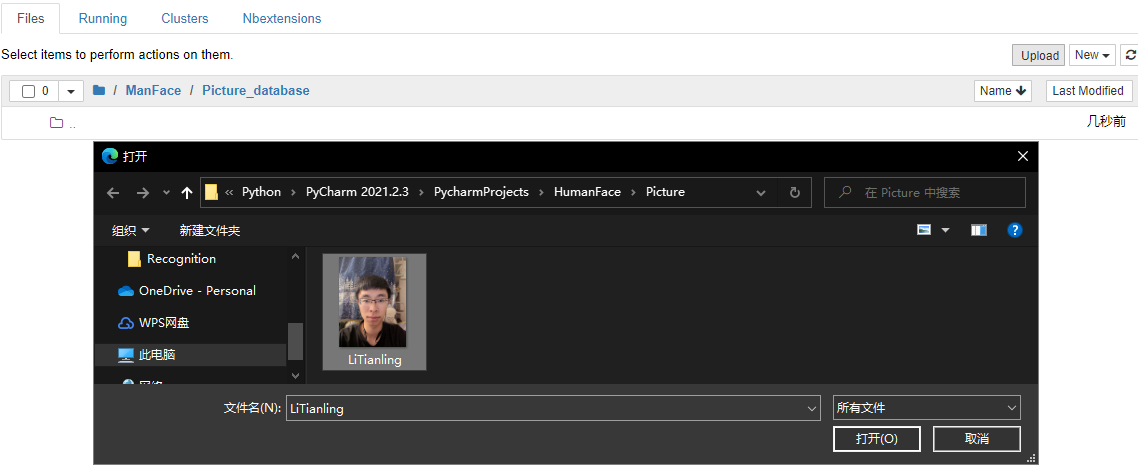
新建一个文件夹：



重命名为ManFace：



同上，在ManFace新建一个Picture\_database，导入人脸数据库图片：我的路径是D:\Python\PyCharm 2021.2.3\PycharmProjects\HumanFace\Picture





同理在ManFace新建一个Picture\_datatest，导入人脸识别测试图片：我的路径是C:/Users/LTL/Desktop/并重命名为DATAtest.jpg



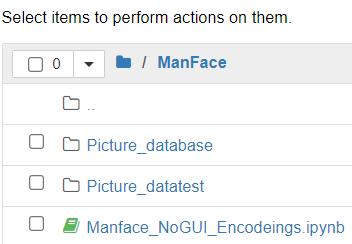
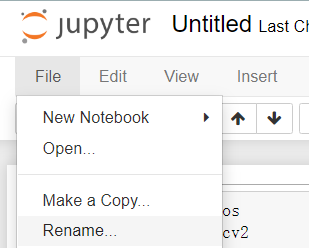
## 5.2 添加代码——人脸识别测试代码

新建第一个代码文件：数据库识别与编码



|  |
| --- |
| import os  import cv2  import numpy as np  import face\_recognition  Image\_database\_path = 'Picture\_database' # 图像数据库路径  images = [] # 数据库所有图像数据集合  EncodeingList = [] # 数据库所有图像编码集合  classNames = [] # 数据库所有图像的名称（分类）  # 读取图像数据库  def readImageDatabase(Image\_database\_path):  imgList = os.listdir(Image\_database\_path) # 列出图像数据库的所有图片名称列表  for pic in imgList: # 读取数据库  img = cv2.imread('{}/{}'.format(Image\_database\_path, pic)) # 读取数据库其中一张图片的数据  images.append(img) # 将读取到的图片数据存入列表  classNames.append(os.path.splitext(pic)[0]) # 存入图片对应的名字  print('Image\_database include:', classNames)  # 对人脸集合进行编码处理  def findEncodeings(images):  for img in images:  img = cv2.cvtColor(src=img, code=cv2.COLOR\_BGR2RGB) # 对img进行灰度处理  encode = face\_recognition.face\_encodings(img)[0] # face\_encodings对图片对象a\_images进行编码并返回数组0位置编码结果  EncodeingList.append(encode) # 把编码存储在编码列表中  print(EncodeingList)  print('Coding is complete !!!')  if not os.path.exists("database"): # 如果不存在数据库文件夹就创建（保存编码后数据）  os.mkdir("database")  np.savez('database/EncodeingList', EncodeingList) # 保存编码数据库以数据及名称  np.savez('database/classNames', classNames)  if \_\_name\_\_ == '\_\_main\_\_':  readImageDatabase(Image\_database\_path) # 读取数据库  findEncodeings(images) # 对数据库进行编码处理 |

重命名为Manface\_NoGUI\_Encodeings



添加第二个代码文件：

|  |
| --- |
| import cv2  import numpy as np  import face\_recognition  picture\_path = 'Picture\_datatest/DATAtest.jpg' # 要识别的图片路径  # 人脸识别的主体  def faceRecognition(picture\_path):  frame = cv2.imread(picture\_path) # 读取待识别图片  frameRGB = cv2.cvtColor(src=frame, code=cv2.COLOR\_BGR2RGB)  facesLocate = face\_recognition.face\_locations(frameRGB) # 对摄像头读取的检测人脸  faceEncoded = face\_recognition.face\_encodings(frameRGB, facesLocate) # 进行特征编码  for face\_encoding in faceEncoded: # 遍历检测的人脸和库中读取的图片进行对比，计算其相似度  matchs = face\_recognition.compare\_faces(EncodeingList, face\_encoding) # 进行匹配  distance = face\_recognition.face\_distance(EncodeingList, face\_encoding) # 计算相似度  lab = 'unknow'  for index, item in enumerate(distance):  Similarity = 1 - item  if Similarity > 0.7:  if matchs[index]:  lab = 'name:{}; Similarity:{}'.format(classNames[index], Similarity) # 得到匹配到的图片名称与相似度值  break  print(lab) # 输出识别信息  if \_\_name\_\_ == '\_\_main\_\_':  EncodeingList = np.load('database/EncodeingList.npz') # 导入数据库\_\_\_\_\_\_\_数据库所有图像编码集合  EncodeingList = EncodeingList['arr\_0']  classNames = np.load('database/classNames.npz') # 导入数据库\_\_\_\_\_\_\_数据库所有图像的名称（分类）  classNames = classNames['arr\_0']  faceRecognition(picture\_path) # 人脸识别 |

重命名为：Manface\_NoGUI\_Recognition

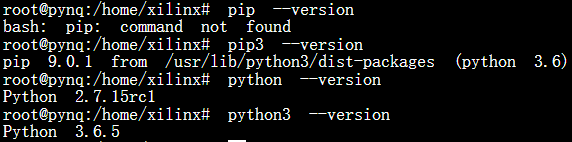


最终结果如下：

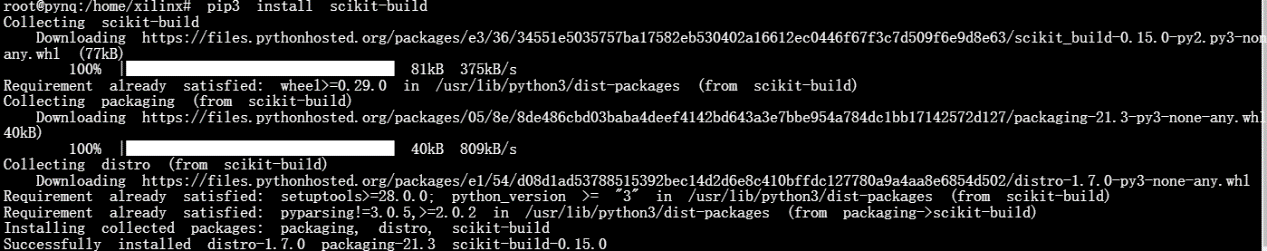


## 5.3 环境配置

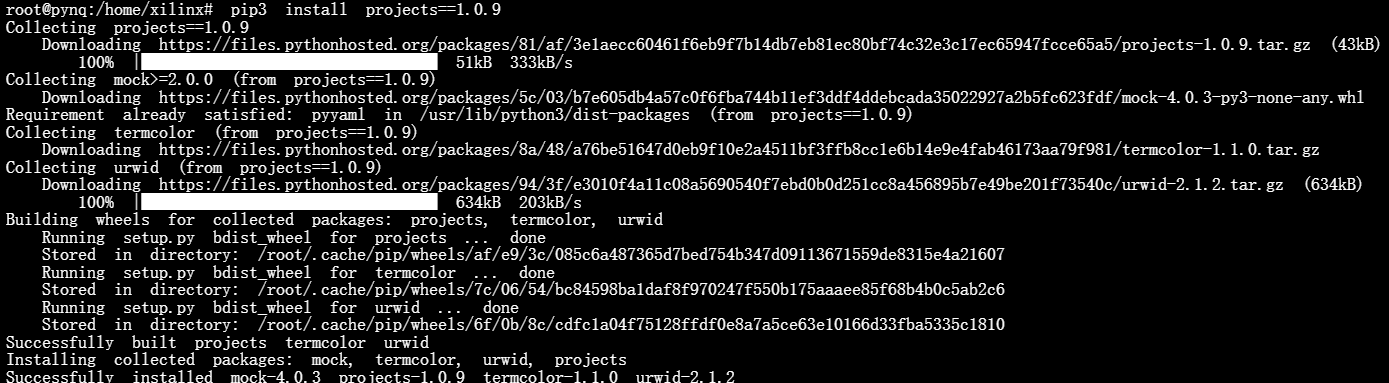
最初状态如下：



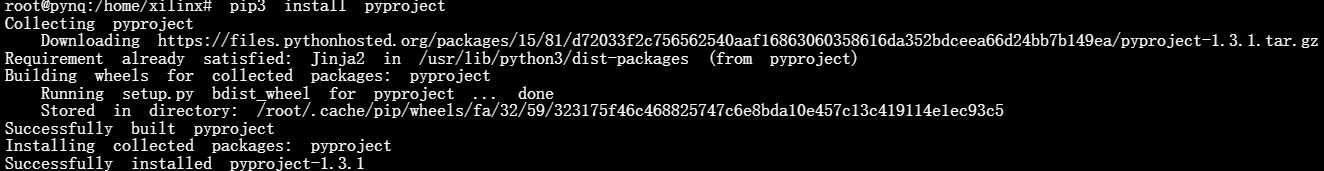
pip3 install scikit-build



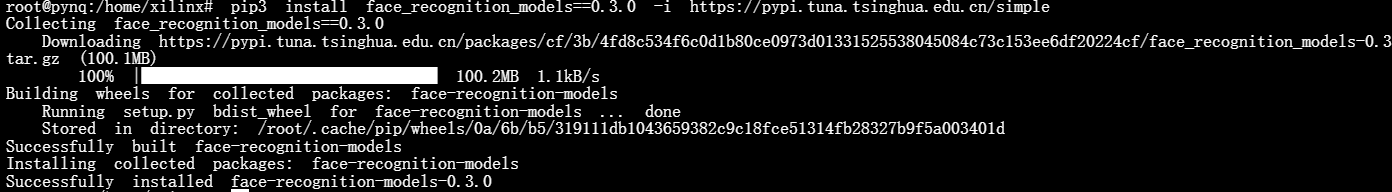
pip3 install projects==1.0.9



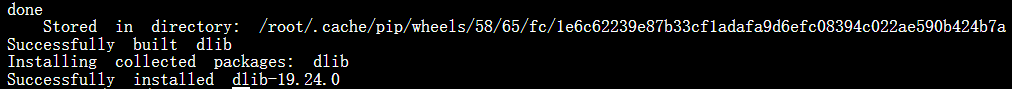
pip3 install pyproject



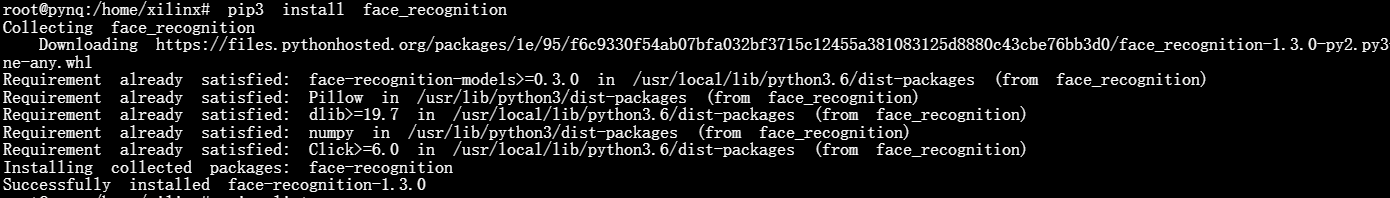
pip3 install face\_recognition\_models==0.3.0 -i https://pypi.tuna.tsinghua.edu.cn/simple



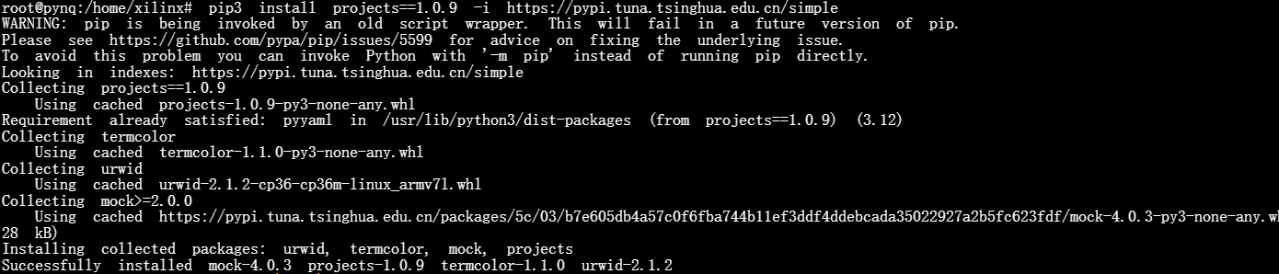
pip3 install dlib -i https://pypi.tuna.tsinghua.edu.cn/simple



pip3 install face\_recognition

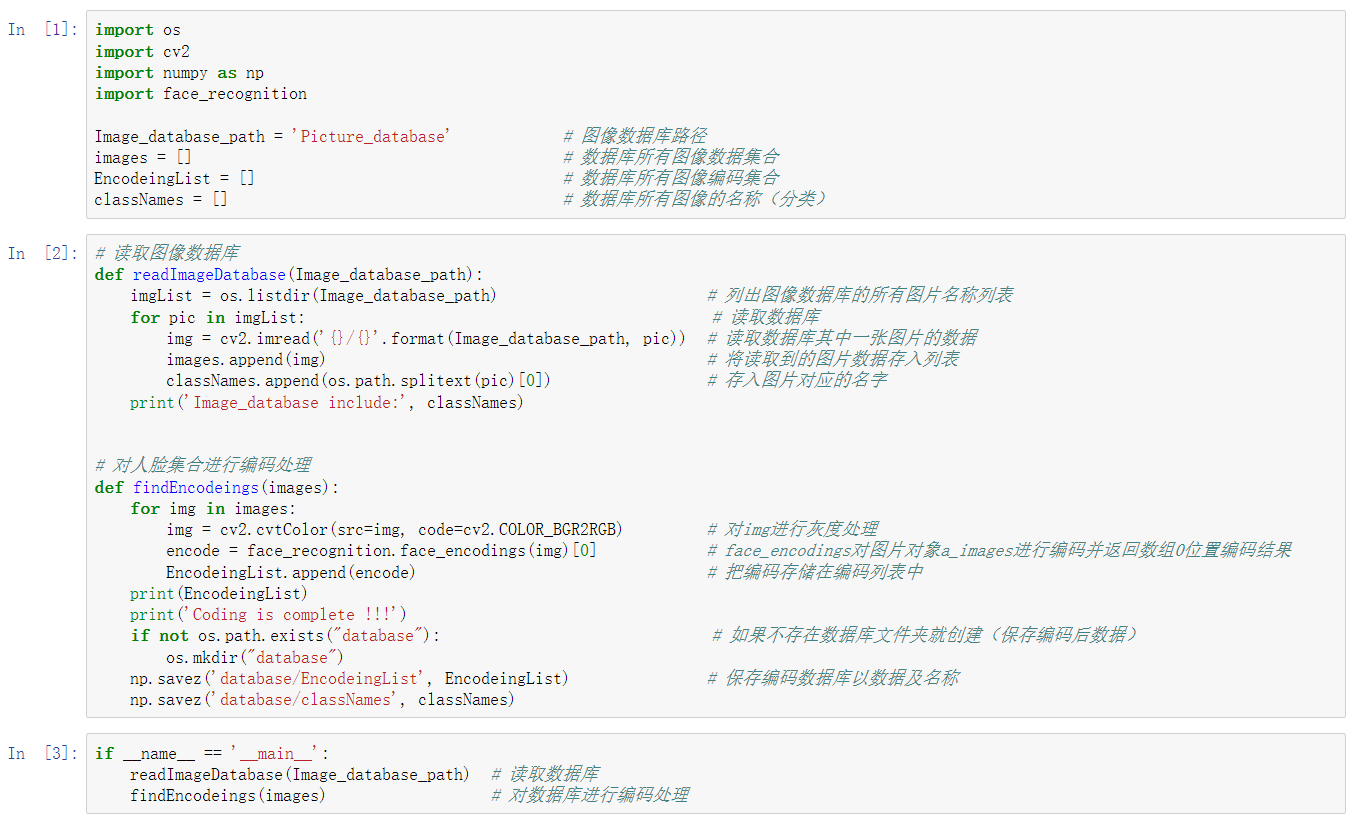


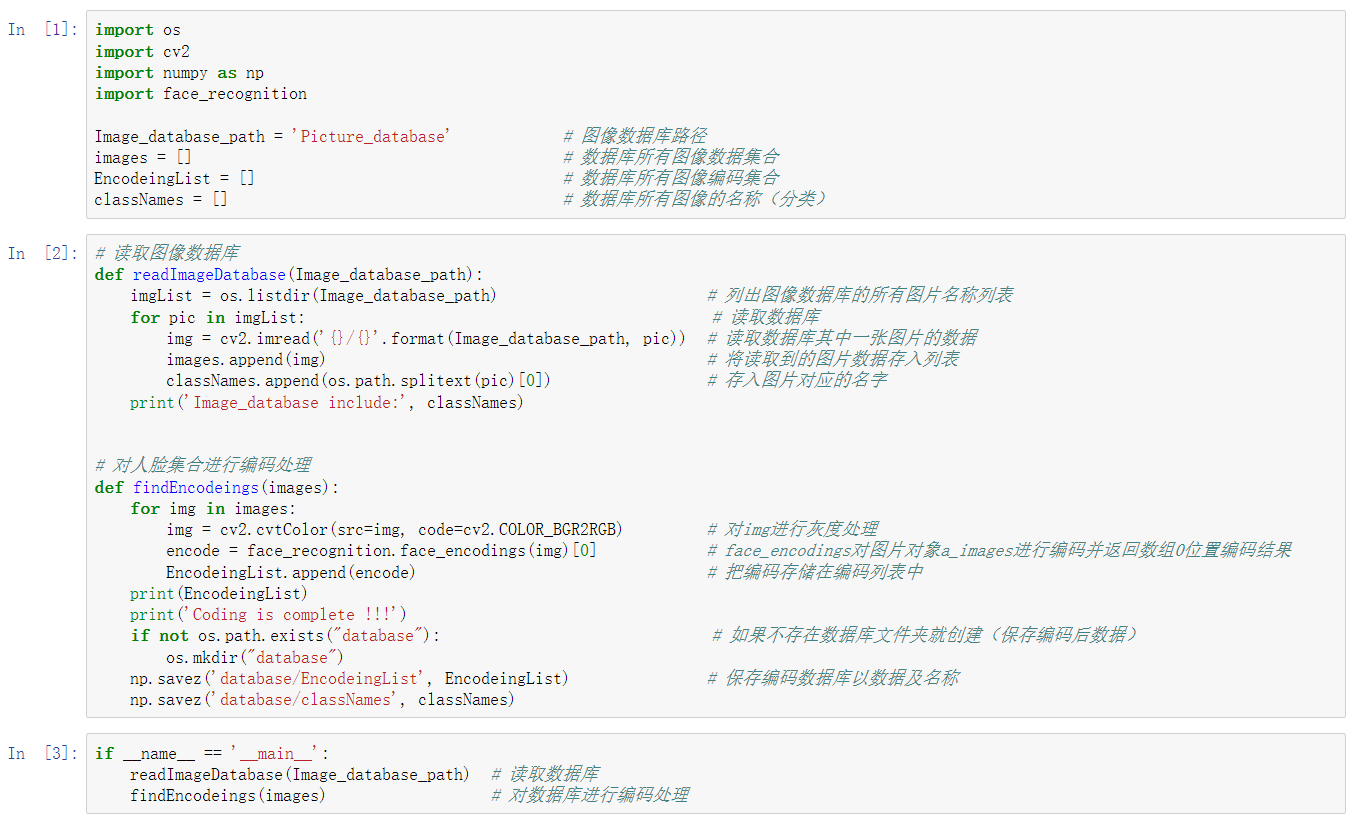
**注意：板子是32位Linux，PC机是64位Windos。-i <https://pypi.tuna.tsinghua.edu.cn/simple>是临时使用清华镜像源。**附加：将pip版本升级到最新,就不会出现Command "python setup.py egg\_info" failed with error code 1的情况了）重启后用pip –version查看pip版本（下边这个出问题再执行）



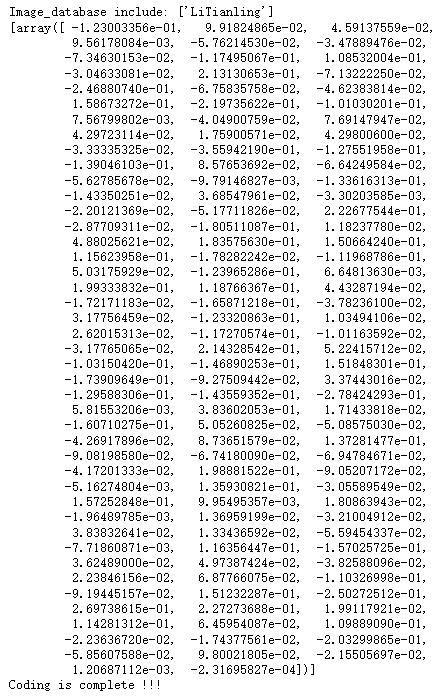
## 5.4 运行测试代码

编码部分：





编码结果：



识别部分及结果：



# 六、融入摄像头（96开发板）

**Z7不识别摄像头换96开发板。**

## 6.1设计新的识别代码

总体思路是：对数据库编码[不修改]（只要数据库图像不改变就不用重新执行，**变更数据库图像务必重新编码**）---->图像识别[修改]（**摄像头采集新的图像替换之前旧的图像**-->用采集到的图像信息对比数据库中的编码得出结论）

数据库编码的代码不用修改。本小结将修改识别部分的代码，将Manface\_NoGUI\_Recognition融入摄像头wecam，升级为Manface\_NoGUI\_Recognition\_wecam。在该设计中添加如下代码：

第一部分，导入相关包：

|  |
| --- |
| import cv2  import numpy as np  import face\_recognition  from PIL import Image as PIL\_Image  picture\_path = 'Picture\_datatest/DATAtest.jpg' # 要识别的图片路径 |

第二部分，定义相关函数：

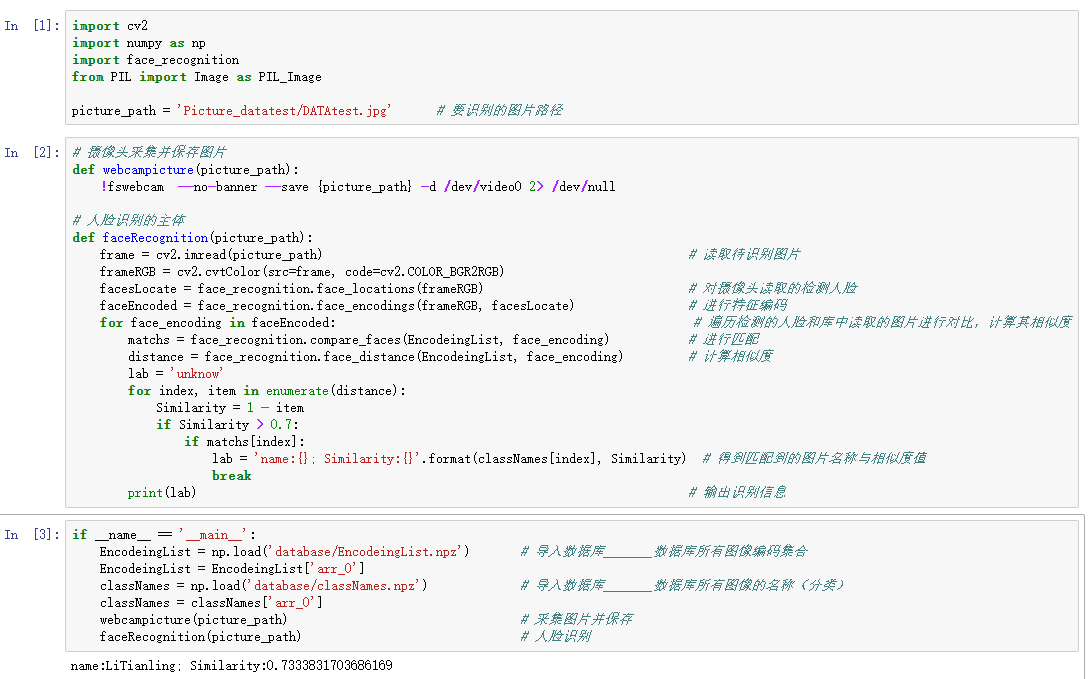
|  |
| --- |
| # 摄像头采集并保存图片  def webcampicture(picture\_path):  !fswebcam --no-banner --save {picture\_path} -d /dev/video0 2> /dev/null    # 人脸识别的主体  def faceRecognition(picture\_path):  frame = cv2.imread(picture\_path) # 读取待识别图片  frameRGB = cv2.cvtColor(src=frame, code=cv2.COLOR\_BGR2RGB)  facesLocate = face\_recognition.face\_locations(frameRGB) # 对摄像头读取的检测人脸  faceEncoded = face\_recognition.face\_encodings(frameRGB, facesLocate) # 进行特征编码  for face\_encoding in faceEncoded: # 遍历检测的人脸和库中读取的图片进行对比，计算其相似度  matchs = face\_recognition.compare\_faces(EncodeingList, face\_encoding) # 进行匹配  distance = face\_recognition.face\_distance(EncodeingList, face\_encoding) # 计算相似度  lab = 'unknow'  for index, item in enumerate(distance):  Similarity = 1 - item  if Similarity > 0.7:  if matchs[index]:  lab = 'name:{}; Similarity:{}'.format(classNames[index], Similarity) # 得到匹配到的图片名称与相似度值  break  print(lab) # 输出识别信息 |

第三部分，实现：

|  |
| --- |
| if \_\_name\_\_ == '\_\_main\_\_':  EncodeingList = np.load('database/EncodeingList.npz') # 导入数据库\_\_\_\_\_\_\_数据库所有图像编码集合  EncodeingList = EncodeingList['arr\_0']  classNames = np.load('database/classNames.npz') # 导入数据库\_\_\_\_\_\_\_数据库所有图像的名称（分类）  classNames = classNames['arr\_0']  webcampicture(picture\_path) # 采集图片并保存  faceRecognition(picture\_path) # 人脸识别 |

## 6.2 运行结果

摄像头亮灯的时候即在采集新的数据。



# 七、测试GPIO（用Z7测试）EMIO

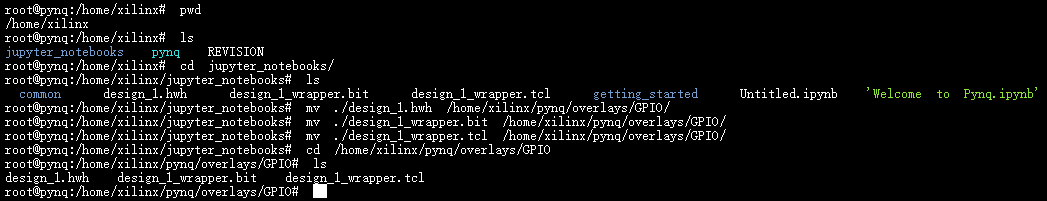
## 7.1 初步测试

需要.bit与.tcl与.hwh文件/home/ltl/Documents/Zynq-7000-project/Zynq-7000-pynq/Zynq-7000-pynq.runs/impl\_1下有前两个，/home/ltl/Documents/Zynq-7000-project/Zynq-7000-pynq/Zynq-7000-pynq.srcs/sources\_1/bd/design\_1/hw\_handoff下有第三个文件，将他们拷贝到共享文件夹。

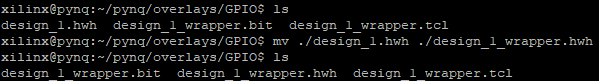
通过jupyter将文件上传到开发板：



在开发板/home/xilinx/pynq/overlays路径下新建一个GPIO文件夹用来存放三个文件。



**需要注意的是，上述三个文件，文件名前缀需要改为一致（如1.tcl；1.bit；1.hwh），才能在PYNQ中正确调用。所以要将design\_1.hwh改名为design\_1\_wrapper.hwh。**

****

在juputer添加设计代码并运行：

****

结果如下：第一部分导入需要的IP，第二部分查询相关信息（查询结果如下图），第三部分创建IO口，第四部分控制IO口（0低电压1高电压，LED共阳极低电压点亮，前四个灯三个亮一个灭）。

****

## 7.2 和人脸识别结合

创建一个Manface\_NoGUI\_Recognition\_GPIO的代码文件。添加如下代码：

第一部分：

|  |
| --- |
| import cv2  import numpy as np  import face\_recognition  from pynq import Overlay  from pynq import GPIO  picture\_path = 'Picture\_datatest/DATAtest.jpg' # 要识别的图片路径  ol = Overlay('/home/xilinx/pynq/overlays/GPIO/design\_1\_wrapper.bit')  LED0 = GPIO(GPIO.get\_gpio\_pin(0),'out')  LED1 = GPIO(GPIO.get\_gpio\_pin(1),'out')  LED2 = GPIO(GPIO.get\_gpio\_pin(2),'out')  LED3 = GPIO(GPIO.get\_gpio\_pin(3),'out') |

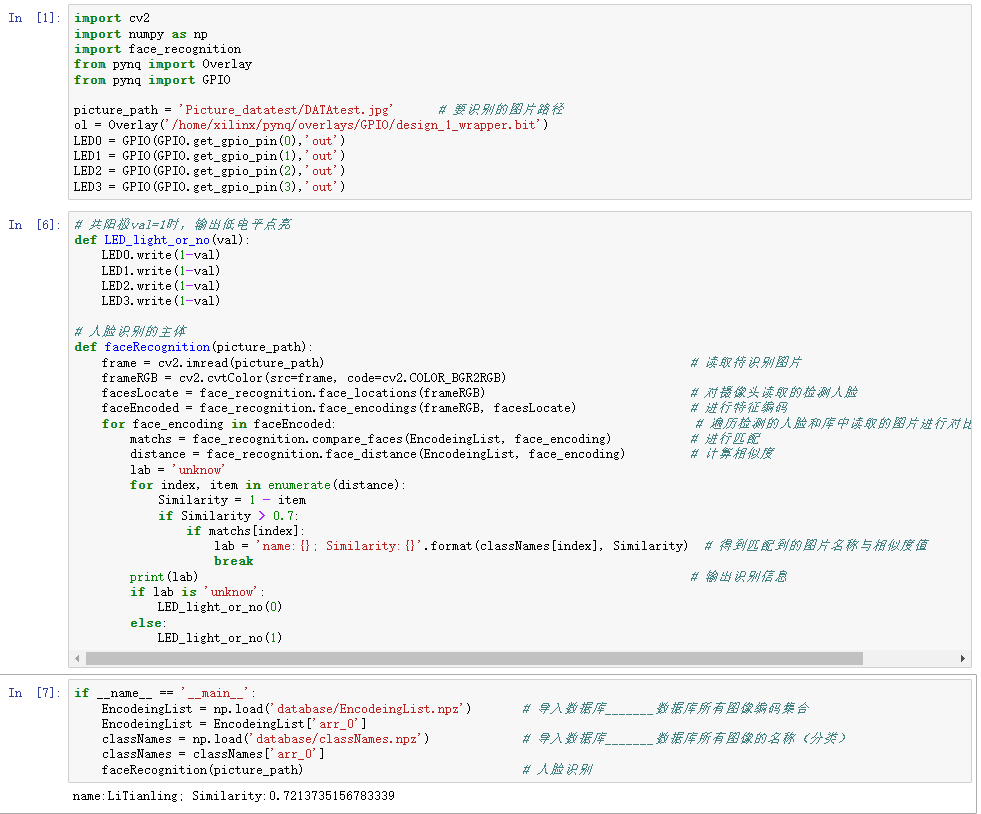
第二部分：

|  |
| --- |
| # 共阳极val=1时，输出低电平点亮  def LED\_light\_or\_no(val):  LED0.write(1-val)  LED1.write(1-val)  LED2.write(1-val)  LED3.write(1-val)  # 人脸识别的主体  def faceRecognition(picture\_path):  frame = cv2.imread(picture\_path) # 读取待识别图片  frameRGB = cv2.cvtColor(src=frame, code=cv2.COLOR\_BGR2RGB)  facesLocate = face\_recognition.face\_locations(frameRGB) # 对摄像头读取的检测人脸  faceEncoded = face\_recognition.face\_encodings(frameRGB, facesLocate) # 进行特征编码  for face\_encoding in faceEncoded: # 遍历检测的人脸和库中读取的图片进行对比，计算其相似度  matchs = face\_recognition.compare\_faces(EncodeingList, face\_encoding) # 进行匹配  distance = face\_recognition.face\_distance(EncodeingList, face\_encoding) # 计算相似度  lab = 'unknow'  for index, item in enumerate(distance):  Similarity = 1 - item  if Similarity > 0.7:  if matchs[index]:  lab = 'name:{}; Similarity:{}'.format(classNames[index], Similarity) # 得到匹配到的图片名称与相似度值  break  print(lab) # 输出识别信息  if lab is 'unknow':  LED\_light\_or\_no(0)  else:  LED\_light\_or\_no(1) |

第三部分：

|  |
| --- |
| if \_\_name\_\_ == '\_\_main\_\_':  EncodeingList = np.load('database/EncodeingList.npz') # 导入数据库\_\_\_\_\_\_\_数据库所有图像编码集合  EncodeingList = EncodeingList['arr\_0']  classNames = np.load('database/classNames.npz') # 导入数据库\_\_\_\_\_\_\_数据库所有图像的名称（分类）  classNames = classNames['arr\_0']  faceRecognition(picture\_path) # 人脸识别 |

运行结果：检测成功，4灯全亮。



## 附加：启动没反应，串口无提示信息

断开所有外设与网线，多次尝试软/硬启动。

# 八、最终实现（96开发板）

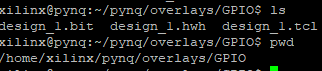
摄像头wecam + 数据库编码 + 人脸识别 + GPIO输出。

## 8.1 准备阶段

同样在/home/xilinx/pynq/overlays路径下创建GPIO文件夹存储.bit、.hwh、.tcl三文件：



将三文件通过jupyter上传到开发板，并移动至/home/xilinx/pynq/overlays/GPIO下：



连接硬件并测试，参考7.1中的步骤。

## 8.2 融合GPIO并全自动识别

在Manface\_NoGUI\_Recognition\_wecam.ipynb中融入GPIO并显示原始图片以及自动擦除历史数据使其升级为：Manface\_NoGUI\_Recognition\_wecam\_GPIO.ipynb。设计代码如下：

第一部分：

|  |
| --- |
| import cv2  import numpy as np  import face\_recognition  from PIL import Image as PIL\_Image  from pynq import Overlay  from pynq import GPIO  from IPython.display import clear\_output as clear  import matplotlib.pyplot as plt  import time  picture\_path = 'Picture\_datatest/DATAtest.jpg' # 要识别的图片路径  ol = Overlay('/home/xilinx/pynq/overlays/GPIO/design\_1.bit')  LED0 = GPIO(GPIO.get\_gpio\_pin(0),'out')  LED1 = GPIO(GPIO.get\_gpio\_pin(1),'out')  LED2 = GPIO(GPIO.get\_gpio\_pin(2),'out')  LED3 = GPIO(GPIO.get\_gpio\_pin(3),'out') |

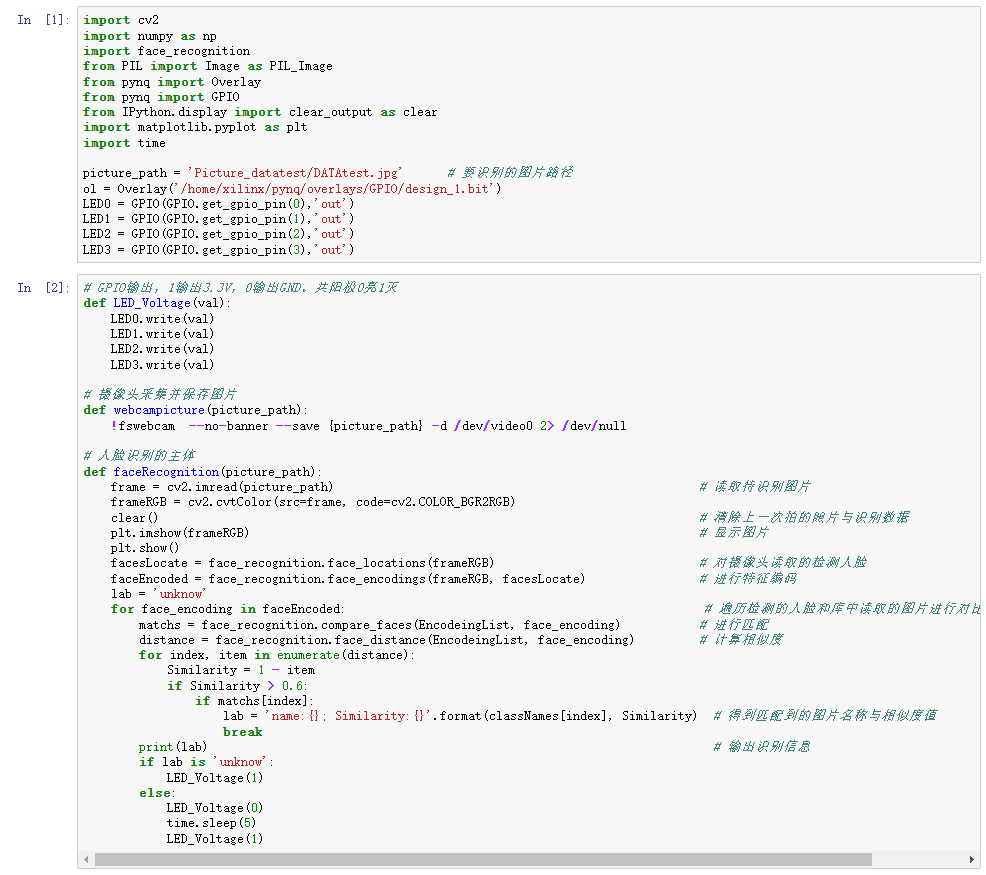
第二部分：

|  |
| --- |
| # GPIO输出，1输出3.3V，0输出GND。共阳极0亮1灭  def LED\_Voltage(val):  LED0.write(val)  LED1.write(val)  LED2.write(val)  LED3.write(val)  # 摄像头采集并保存图片  def webcampicture(picture\_path):  !fswebcam --no-banner --save {picture\_path} -d /dev/video0 2> /dev/null  # 人脸识别的主体  def faceRecognition(picture\_path):  frame = cv2.imread(picture\_path) # 读取待识别图片  frameRGB = cv2.cvtColor(src=frame, code=cv2.COLOR\_BGR2RGB)  clear() # 清除上一次拍的照片与识别数据  plt.imshow(frameRGB) # 显示图片  plt.show()  facesLocate = face\_recognition.face\_locations(frameRGB) # 对摄像头读取的检测人脸  faceEncoded = face\_recognition.face\_encodings(frameRGB, facesLocate) # 进行特征编码  lab = 'unknow'  for face\_encoding in faceEncoded: # 遍历检测的人脸和库中读取的图片进行对比，计算其相似度  matchs = face\_recognition.compare\_faces(EncodeingList, face\_encoding) # 进行匹配  distance = face\_recognition.face\_distance(EncodeingList, face\_encoding) # 计算相似度  for index, item in enumerate(distance):  Similarity = 1 - item  if Similarity > 0.6:  if matchs[index]:  lab = 'name:{}; Similarity:{}'.format(classNames[index], Similarity) # 得到匹配到的图片名称与相似度值  break  print(lab) # 输出识别信息  if lab is 'unknow':  LED\_Voltage(1)  else:  LED\_Voltage(0)  time.sleep(5)  LED\_Voltage(1) |

第三部分：

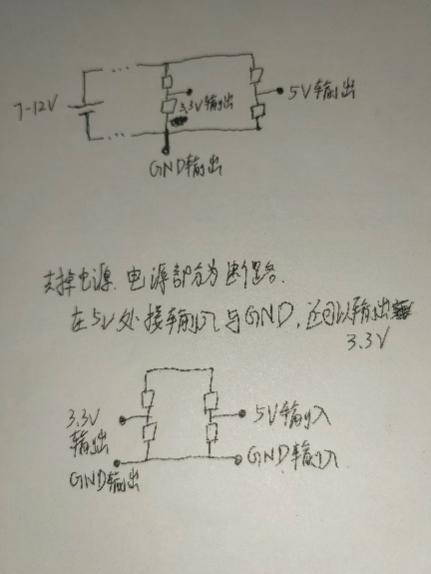
|  |
| --- |
| if \_\_name\_\_ == '\_\_main\_\_':  EncodeingList = np.load('database/EncodeingList.npz') # 导入数据库\_\_\_\_\_\_\_数据库所有图像编码集合  EncodeingList = EncodeingList['arr\_0']  classNames = np.load('database/classNames.npz') # 导入数据库\_\_\_\_\_\_\_数据库所有图像的名称（分类）  classNames = classNames['arr\_0']  while 1:  webcampicture(picture\_path) # 采集图片并保存  faceRecognition(picture\_path) # 人脸识别 |

## 8.3 最终测试





96开发板通过摄像头采集数据，如果识别成功就点亮4个LED，如果是‘unknow’就熄灭。加了while 1 循环，所以采集图像和识别一直交替执行。【如果识别成功低电平持续5s来让龙芯1B读取】



**【附加】：有一个问题：LED是共阳极小灯标准工作电压3.3V，96开发板GPIO输出电压也是3.3V满足要求，但是96开发板上只有5V输出。【解决】：我有一个电源模块，自己算了一下，不加标准外界电源可作为降压器使用把5V降到3.3V。**

# 九、最终之上的改进

更换了图片的采集与输出方式，以视频帧的方式采集，处理后通过HDMI显示在显示屏上，由于显示屏分辨率（1280\*720）图像较大运算时间较长，所以人脸识别的帧率不是特别高。

第一部分：

|  |
| --- |
| import cv2  import numpy as np  import face\_recognition  from PIL import Image as PIL\_Image  from pynq import Overlay  from pynq import GPIO  from IPython.display import clear\_output as clear  import matplotlib.pyplot as plt  import time  from pynq.lib.video import \*  displayport = DisplayPort() # 设置视频输出端口与分辨率  displayport.configure(VideoMode(1280, 720, 24), PIXEL\_RGB)  ol = Overlay('/home/xilinx/pynq/overlays/GPIO/design\_1.bit')  LED0 = GPIO(GPIO.get\_gpio\_pin(0),'out')  LED1 = GPIO(GPIO.get\_gpio\_pin(1),'out')  LED2 = GPIO(GPIO.get\_gpio\_pin(2),'out')  LED3 = GPIO(GPIO.get\_gpio\_pin(3),'out') |

第二部分：

|  |
| --- |
| # GPIO输出，1输出3.3V，0输出GND。共阳极0亮1灭  def GPIO\_Voltage(val):  LED0.write(val)  LED1.write(val)  LED2.write(val)  LED3.write(val)  # 摄像头捕获图片  def picture\_capture():  capture = cv2.VideoCapture(0) # 设置视频捕获端口与分辨率  capture.set(3, 1280)  capture.set(4, 720)  frame = displayport.newframe() # 新建空白帧  capture.read(frame) # 向空白帧写入数据——未处理图像  capture.release() # 关闭捕获端口  return frame    # 人脸识别的主体  def faceRecognition():  frame = picture\_capture()    frameRGB = cv2.cvtColor(src=frame, code=cv2.COLOR\_BGR2RGB)  facesLocate = face\_recognition.face\_locations(frameRGB) # 对摄像头读取的检测人脸  faceEncoded = face\_recognition.face\_encodings(frameRGB, facesLocate) # 进行特征编码    name = 'unknow'  lab = 'unknow'  color1 = (0, 255, 0) # 初始化面部捕捉框显示绿色    for (top, right, bottom, left), face\_encoding in zip(facesLocate, faceEncoded): # 遍历检测的人脸和库中读取的图片进行对比，计算其相似度  matchs = face\_recognition.compare\_faces(EncodeingList, face\_encoding) # 进行匹配  distance = face\_recognition.face\_distance(EncodeingList, face\_encoding) # 计算相似度  for index, item in enumerate(distance):  Similarity = 1 - item  if Similarity > 0.7:  if matchs[index]:  lab = 'name:{}; Similarity:{}'.format(classNames[index], Similarity) # 得到匹配到的图片名称与相似度值  break    cv2.rectangle(img=frame, pt1=(left, top), pt2=(right, bottom), color=color1, thickness=3) # 绘制画面部捕捉框  cv2.putText(frame, lab, (left, top - 8), cv2.FONT\_HERSHEY\_SIMPLEX, 0.7, color1, 2) # 在捕捉框上添加匹配到的图片信息  # cv2.imwrite('{}/{}.png'.format(picture\_path, name), frame) # 保存图片  displayport.writeframe(frame) # 输出图像帧——处理过的  frame = []    print(lab) # 输出识别信息  if lab is 'unknow':  GPIO\_Voltage(1)  else:  GPIO\_Voltage(0) |

第三部分：

|  |
| --- |
| if \_\_name\_\_ == '\_\_main\_\_':  EncodeingList = np.load('database/EncodeingList.npz') # 导入数据库\_\_\_\_\_\_\_数据库所有图像编码集合  EncodeingList = EncodeingList['arr\_0']  classNames = np.load('database/classNames.npz') # 导入数据库\_\_\_\_\_\_\_数据库所有图像的名称（分类）  classNames = classNames['arr\_0']  while 1:  faceRecognition() # 人脸识别 |

