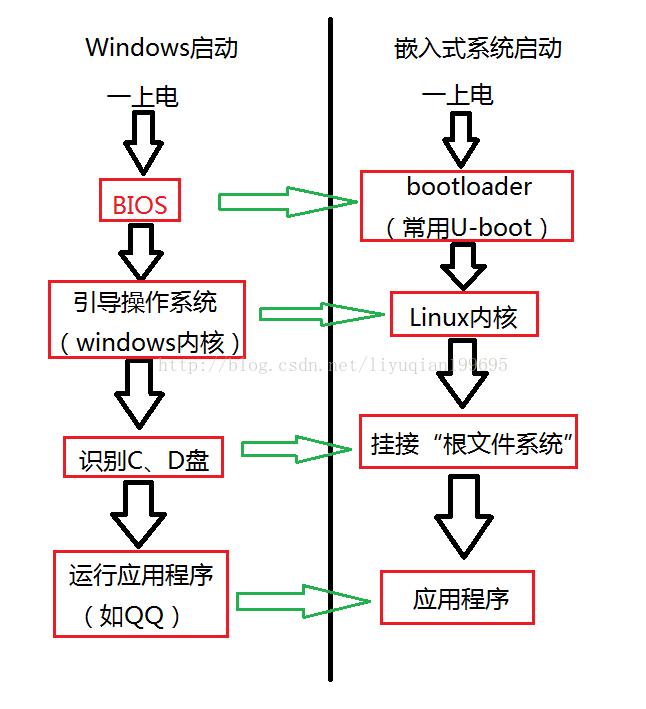
1. linux最小系统的构建（lab3 ZYNQ最小系统、lab4 SD卡启动）



1. 准备uboot、Linux系统的内核、文件系统、设备树等文件

<https://its201.com/article/u010580016/80430138>

1. 下载交叉编译器arm-xilinx-linux-gnueabi-，编译uboot，生成u-boot.elf文件

由于版本问题，后将交叉编译器改为arm-linux-gnueabihf-

1. 编译linux内核，选择arm配置，得到uImage以及dtc编译工具
2. 利用dtc编译工具将zynq\_zed.dts编译为devicetree.dtb，指令如下

🡪<https://blog.csdn.net/EmSoftEn/article/details/42431931>

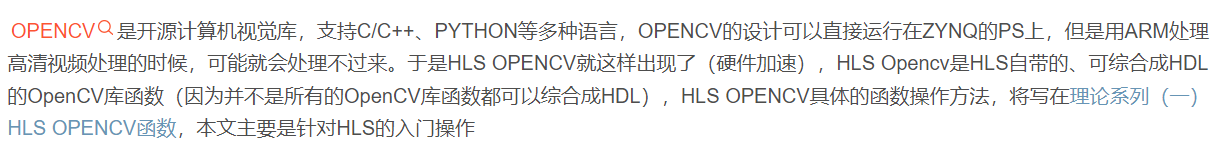
此处会报错zynq\_zed.dts有语法错误，需要对源文件进行修改。

🡪参见<https://www.jianshu.com/p/6f2154c91eac>的第六部分

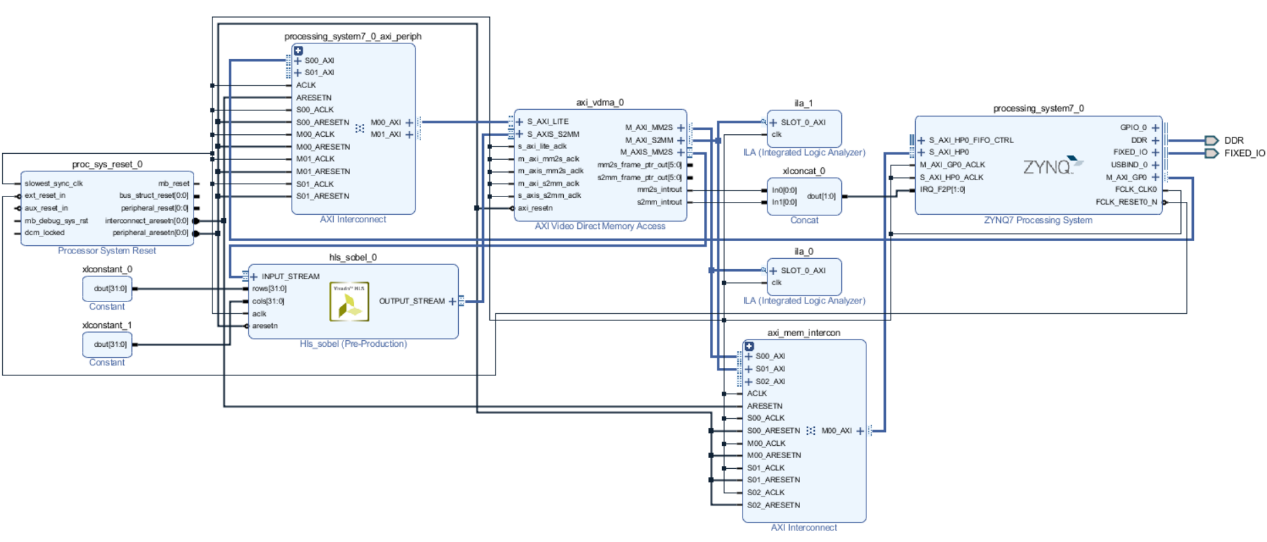
1. 下载官方的文件系统压缩文件🡪

<https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842473/Build+and+Modify+a+Rootfs>

1. 硬件系统的准备
2. 生成HLS\_Sobel的IP核，用于边缘检测🡪硬件加速



1. 创建block design，导出bit



1. 在SDK中生成BOOT.BIN，将uImage、devicetree.dtb、uramdisk.image.gz、7z010.bit、BOOT.BIN拷入SD卡
2. 上电启动。
3. 文件系统的更改

Xilinx官方提供的linux文件系统太弱了，很多应用都没有，因此利用ubuntu提供的base重新制作了一个文件系统，但无法上电启动（根文件系统的读取存在问题）。尝试配置新的内核无果，还弄坏了服务器，最后选择采用最初的根文件系统，在服务器上交叉编译得到可执行文件后，放到板子上去运行。

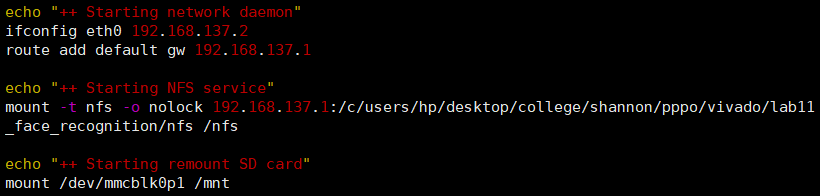
1. 开发板联网

通过网线将开发板与电脑相连，电脑开启网络共享，此时可以得到以太网的ip为192.168.137.1，在开发板上输入指令ifconfig eth0 192.168.137.2，开启以太网，再输入指令route add default gw 192.168.137.1，尝试ping外网，不通。经过多次测试，发现有可能是因为电脑在开发板连接前已经开启网络共享（同时防火墙打开），需要将防火墙关闭后重新将电脑的网络共享关闭再打开才能联通外网。

为了不再每次修改文件都需要插拔SD卡，我设置了windows的nfs服务，教程如下：

<https://blog.csdn.net/Turix/article/details/101445321>。

与此同时，修改了嵌入式的启动文件/etc/init.d/rcS，使得其每次启动时自动联网并完成文件夹的挂载，添加代码如下：



注意，其中的ip地址和挂载的windows目录因人而异，与电脑设置和自己的配置有关。

1. 开发板上可执行程序的运行

由于开发板上的glibc库的版本相对较低，从服务器编译过来的可执行程序无法执行，需要将服务器上的动态库复制到开发板的根文件系统中并完成软链接，相对应的指令为：

ln -snf /lib/libc-2.27.so /lib/libc.so.6

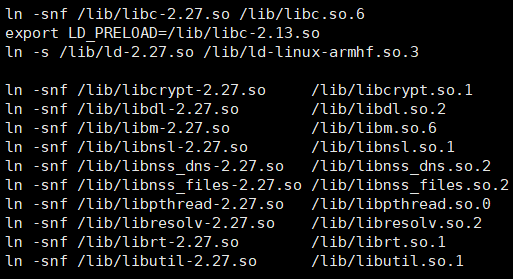
export LD\_PRELOAD=/lib/libc-2.13.so

ln -s /lib/ld-2.27.so /lib/ld-linux-armhf.so.3

需要注意的是，gilbc库是系统最底层的库，一旦更改，将会导致很多基础的指令无法使用，或者导致系统无法正常启动。尝试将上述三条指令放在/etc/init.d/rcS文件中，发现系统启动后会一直报错-bin/ash: error while loading shared libraries: /lib/libc.so.6。查阅资料发现，linux在启动时会先执行/etc/rcS中的命令，再执行/etc/profile中的命令。考虑到报错时没有shell，怀疑是profile设置命令工具所在位置时由于libc.so.6的变动而产生错误，遂将上述三条指令放到了profile的最后面，重新生成根文件系统，上电启动，系统正常运行。

然而，虽然通过这样的操作，开发板上可以正常运行可执行程序，但每次执行可执行程序时都会报错（与上述三条指令中的第二条有关）。但该条指令不能unset，因为开发板上很多基础指令都需要libc-2.13.so才能够执行。运行程序时产生的报错不会影响程序的正常执行，因此不必修改。

除此之外，在运行测试程序时，我加入了math.h头文件，在开发板上运行报错libm.so不兼容，意识到除了libc.so以外，其余所有动态库也都需要更新。先尝试直接在根文件系统中更新，即在PC上将2.27版本的库拷入根文件系统，做好软链接，再把2.13的库删除，花了很长时间，发现这样系统根本无法启动。无奈，只好把2.27版本的库先拷入rootfs，同时修改profile，待系统启动后，再在板子上做软链接，添加的代码如下：



重新将系统拷入SD卡，开机，可能会报一些错，说什么文件不存在、文件已存在之类的，但其实并无大碍，系统能正常运行。最后测试文件的输出为：



1. OpenCV的移植

<http://xilinx.eetrend.com/blog/2019/100017467.html>

生成opencv的镜像后，需要将其移植到板子上，这里将其直接存放进SD卡中，在开机时完成挂载/usr/opencv\_lib（先将SD卡挂载在/mnt）。注意，此时若直接运行程序，会产生动态库无法找到的报错，这是因为ld路径设置不对。考虑到板子上没有ldconfig指令，因此采用export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/usr/opencv\_lib/lib，其也必须写入到开机的自启动指令中。rcS中的对应指令为：



profile中的对应指令为：



除此之外，当运行程序时，会报错libgcc\_s.so.1找不到，这个库在PC的/usr/lib/gcc-cross/arm-linux-gnueabihf/7中，需要用locate libgcc\_s.so.1找到其位置，并复制到根文件系统的/lib中。同理，libstdc++.so.6在板子的/usr/lib中也存在，但仍会报错，估计是版本问题，将PC上的libstdc++.so.6指向的软连接libstdc++.so.6.0.25复制到板子的/usr/lib中，再在开机时更新软连接，如下：



运行测试程序，可以正确生成out.bmp。（最开始测试时出现段错误，原因是我输入的图片不是bmp格式，没有深度信息）。

1. 人脸识别算法测试

人脸识别算法主要分为两个部分：人脸的收集、训练，人脸识别。在测试阶段，我们将其分为两个部分，分别用两个可执行程序faceCollect和faceRecognition完成。由于摄像设备还没有加入硬件，先下载数据集，静态完成人脸收集，并将model存放在/nfs/trainedData/下的yml文件中，在faceRecognition中读取。另外，由于opencv将人脸识别模块放到了opencv\_contrib中，需要重新打包opencv镜像，在进行cmake时加入语句：-D OPENCV\_EXTRA\_MODULES\_PATH=/root/linux/OpenCV\_Contrib/modules/。在对人脸识别程序进行编译的过程中，由于版本的原因（opencv2和opencv3的很多写法发生了改变），通过查阅博客、文档，对源程序中的类、部分函数进行了修改，如using name space cv::face，LDA::subspaceProject()等，最终顺利完成了编译。

在开发板上测试时，出现了报错找不到libfaceRec.so，使用readelf在PC上查看，发现可执行程序确实依赖于libfaceRec.so，但PC上并没有libfaceRec.so（实际上是有的，在我程序的lib文件夹中，被locate误导了）。

测试时发现数据集中图片的读取出现错误，打开/nfs/下的dataset，发现连接中断，猜测可能是文件过大，因此将数据集转移到SD卡中，但仍出现错误，发现是图片的路径不对，但在本地测试时的文件路径正确。直接在identity.txt中保存图片的绝对路径后，imread()却无法读到图片。经过测试，cvLoadImage()能够正确读取到图片，但经过类型转换为Mat后却为空。后来，我又测试了一下通过命令行输入文件路径，发现imread()能够正常读入图片。因此，可以确定问题出现在图片的路径上。在修改代码的过程中，我发现通过nfs传送的可执行文件的实时性不佳，这导致我有可能在修改时已经修改正确，但误以为错误，最好重启再运行程序。在最后，确定是由于图片的读取路径包含了最后的换行符”\n”，需要将string转成char\*后将\n修改为\0，再转成const char\*。

最后，linux中没有现成的键盘处理函数，而opencv的waitKey()在开发板上无法使用，因此考虑使用硬件控制模式转换。

1. 硬件系统的完成

首先需要实现的是使用摄像头捕捉到帧，这里先参考lab10的硬件设计，其中有一个IP叫做OV\_Sensor\_ML，这个是自己加入的，但老师和工程师没有把IP的源代码给我们，于是自己去网上搜了一个，看他的接口图和文档里的一样，就先试着用一下吧。网址：<https://blog.csdn.net/weixin_30950887/article/details/95946562>。IP生成后，再按照lab10的文档完成block design。由于完整的连线图都没有给出，因此只能按照自己的理解、比对局部图完成连线。

之后就是实现软件驱动摄像头，这个是真的nt，只好去网上找资源。注意的是，当ov7725输出RGB565格式时，只需要使用D[9:2]这高8位的数据，引脚部分需要对应。

经过一天的摸索，发现SDK中外设的驱动与嵌入式linux的开发方法存在较大差异，应当在linux的设备树中加入PL端的外设，因此在新的硬件基础上，尝试用SDK生成的devicetree.dtb启动，但失败了，怀疑是由于内核配置与设备树不相容。无奈，考虑使用petalinux生成linux系统。

1. Petalinux安装与使用

这玩意儿安装也是一身坑。由于root权限下安装可能会导致系统错误，xilinx官方要求只能用非root用户安装。按照下面这篇blog安装<https://www.cxyzjd.com/article/ZXDDBK/108615460>，还是会报错“ERROR: Exiting Installer: Cannot install as root user !”当使用su temp切换到普通用户（temp是我自己通过useradd添加的用户），再执行安装指令时，又会报错没有权限。

在<https://askubuntu.com/questions/1104868/petalinux-v2018-2-installation-error>中看到了类似的问题。最开始我也以为是因为在非root用户下没有权限运行安装程序，结果按照第一个人给的方法，无果。最后按照第二个人的说法，是log文件没有权限，添加权限后，能够成功运行程序。晕！

同理，在使用petalinux生成linux系统的时候，也要求在非root用户下进行编译。虽然创建工程时可以使用root权限，但建议从开始就在普通用户下进行。这里su temp进入普通用户后，需要先source /home/petalinux/install/settings.sh，配置运行环境，再输入petalinux-create --type project --template zynq --name faceRec。这里可以说是很深刻地教学了我linux下的用户权限。由于temp是普通用户，mkdir等指令一概没有权限，必须在root中chmod -R 777 /home/petalinux，赋予其权限，才能正常运行（也正是因此，我把工程目录转移到了/home/petalinux下）。具体的配置方法见如下blog：

<https://juejin.cn/post/7032676226208628767>

<https://www.daimajiaoliu.com/daima/47618b5f89003e4>

<http://xilinx.eetrend.com/blog/2021/100554618.html>

<http://a-suozhang.xyz/2019/09/06/ZCU102-Set-Up/>

按照成功boot的案例，boot args应该设置为console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk rootfstype=ext4 rootwait devtmpfs.mount=0。可惜，在编译时会报错，原因是ubuntu18.04不支持petalinux2017.4，于是退而求其次，找到了petalinux2017.4的docker，如下：<https://github.com/shleemars/Petalinux-Zybo-Z7-10>。对其源码做小小的修改后，即可构建image（在/home/petalinux/package目录下）。

解决docker占用磁盘内存过多的问题：很可能是以前的image未清理。指令：

docker system df (-v) 查看文件大小

docker system prune 自动清理空间

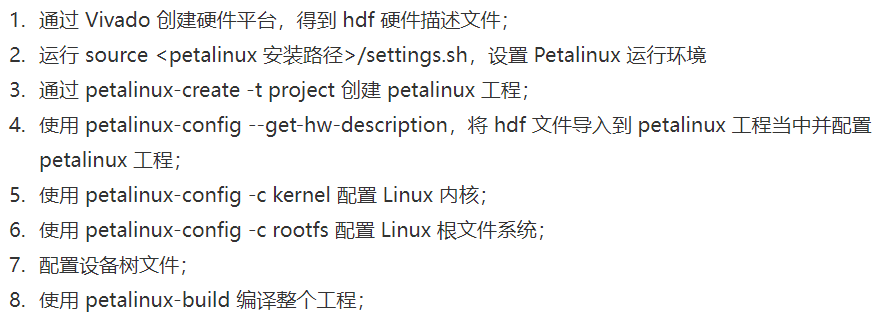
经过多次尝试，上面那个git方法所需的内存实在是太大，于是换了一个，build时只安装必须的安装包：<https://github.com/gitzhangzhao/petalinux_2017.04>。docker中的各个参数为：image名称：petalinux；docker中普通用户名称：temp，密码：SrtpGo!!

后面需要进入docker镜像时，只需输入指令

docker start petalinux

docker exec --user temp -it petalinux /bin/bash

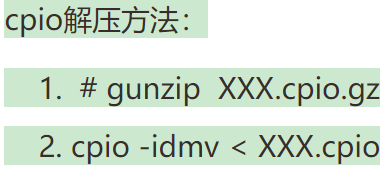
petalinux下打包镜像和启动镜像的指令：



9. 最后再images/linux/下执行：petalinux-package --boot --format BIN --fsbl zynq\_fsbl.elf --fpga design\_1\_wrapper.bit --u-boot，打包启动镜像。此时该目录下即放置这image.ub和BOOT.BIN，需要移动到SD卡的FAT32分区（第一个分区）。

按照petalinux版本的要求，此时的新的文件系统不能采用以前的ramdisk.image.gz的形式，而且必须放在SD卡的第二个分区。因此，需要先把以前的文件系统打包成cpio.gz的格式，并把SD卡分成两个分区，一个为FAT32格式，另一个是ext4格式（根文件系统放在ext4分区中）。windows下的分区软件为DiskGenius。装好ubuntu虚拟机后，可以在虚拟机上查看SD卡的ext4分区。cpio压缩指令：find ./\* | cpio -H tar -o > rootfs.cpio

把已经打包好的rootfs.cpio.gz放到ext4分区当中，采用下面两条指令解压：

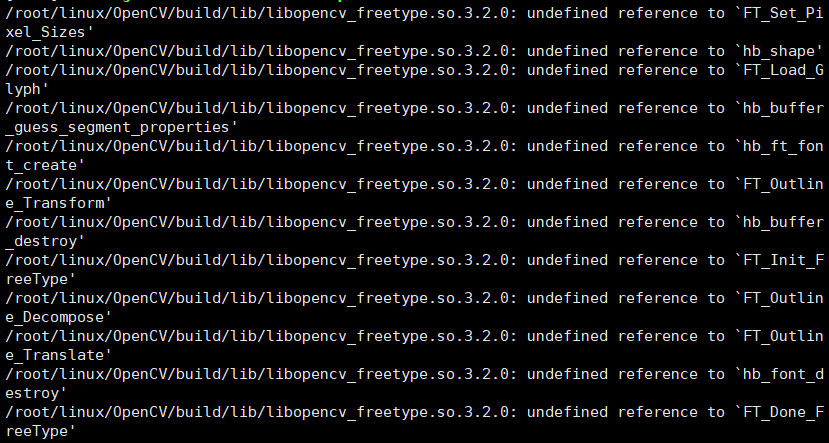


可以得到文件夹rootfs。注意，必须把rootfs中的所有文件move到目录外面来，即在rootfs同级目录执行：sudo mv rootfs/\* .

这样linux启动时才能找到根文件系统。相比之下，这种根文件系统启动方式能更加方便地修改根文件系统，省去了打包等操作。记录一个ubuntu安装时不能从主机复制文件到虚拟机的坑：安装vmware-tool——



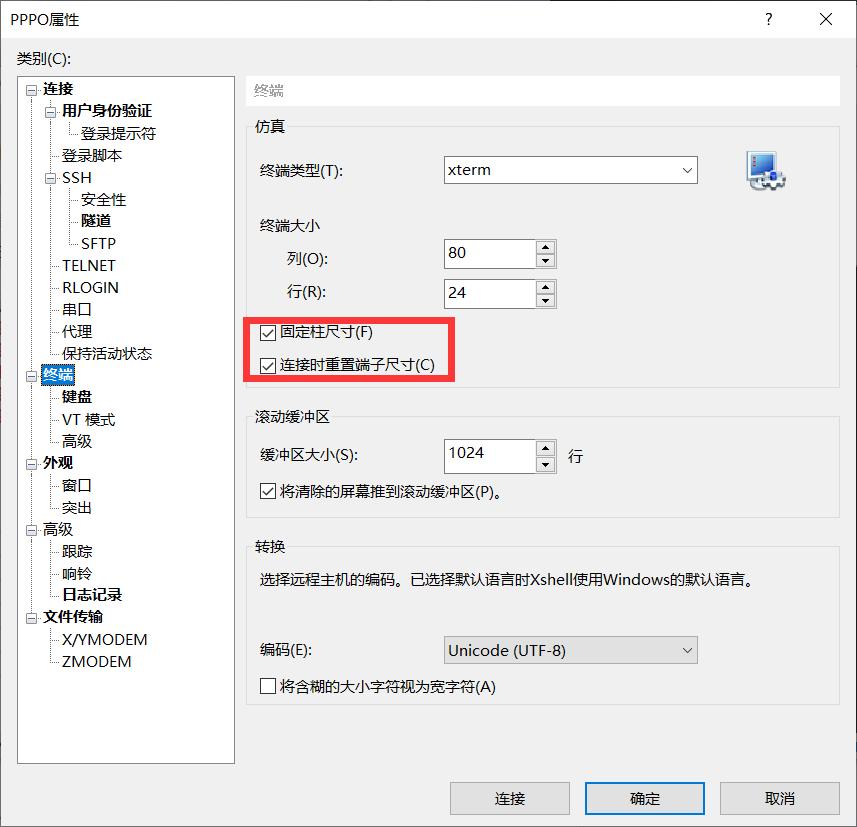
虽然装好了文件系统，但第二次启动时发现，由于第一次启动后修改了lib的文件连接关系导致无法启动。尝试关机时将链接修正，但发现启动后程序无法正常执行，于是使用了petalinux的根文件系统，发现好像还行（这样以前做的都白费了！人麻辣！！！！）petalinux的根文件系统登录后的用户名和密码都是root。有一些库文件没有，版本问题版本问题！！！不想再像前面那样满是错误了，考虑将服务器上的交叉编译器改成较低的版本，使用GLIBC\_2.23以下的库，这需要重新编译opencv和opencv\_contrib，先试试吧。2016.02的arm-linux-gnueabihf的glibc是2.21版本，2017.01（5.4）也是2.21版本，2017.10（5.5）也是2.21版本，2017.11（6.4）是2.23版本，遂采用这个编译器。然而，在重新安装opencv时却出现了问题。我先是采用了3.2版本的opencv，但在编译项目时报错：



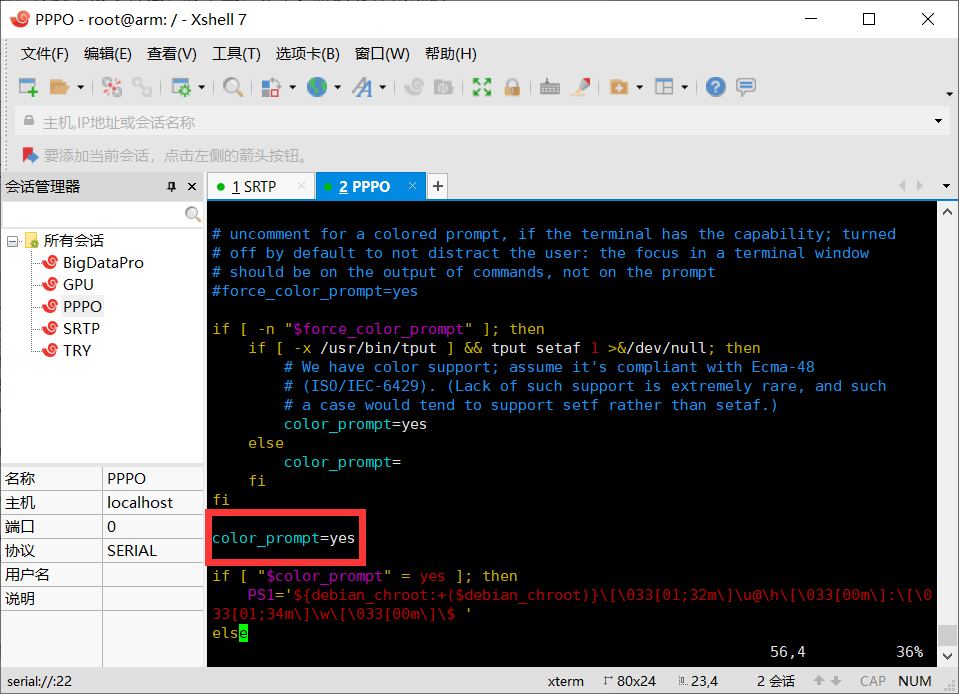
网上说是有个东西的CMakeLists.txt写错了，改了之后重新编译opencv，直接报错找不到libfreetype.so和libharfbuzz.so，但以前那个编译器就没有这个错误啊。然后我又找了opencv3.4，也是一样的错误，于是想找原来那个编译器有没有这个库，发现没有，然后我又想自己编译这两个库，但是一直报错，啊晕！

在网上找了一个ubuntu的根文件系统，但其用户权限设置存在问题，无法切换到root，导致mount指令不能使用，这还弄个屁，干脆自己做一个ubuntu的根文件系统好了。哎，装那些基础的软件也遇到装不上的问题，尝试了很多方法，装不上，算了。新的文件系统的root密码：zynq。普通用户zynq密码：zynq。在刚刚登录时，会提示输入用户名，注意只能以非root用户登入，即zynq。登录完毕后，再输入su，输入密码zynq，即可切换到root用户。

记录一下遇到的坑吧。第一个是网络问题，如果要使用网络，必须要在主机上关闭网络共享再打开，只配置开发板是不行的。第二个是显示问题，读取文本后xshell窗口下面几行就卡住了，不会再更新，可以固定xshell的长度解决，如图：



第三个是颜色的问题，兜兜转转，先是强制把路径名和用户名显示成彩色，这个是在~/.bashrc中进行修改，如下：



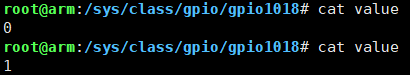
另一个是ls不显示颜色，这个网上都说可以在~/.bashrc里面加一句alias ls=’ls --color’解决，但我仔细看了文件，它是有颜色的设置的。找了很多资料，还有说修改xshell配置的，都不对，最后没找着解决方法，又想到了vim也不显示颜色的问题，于是搜了相关的帖子，发现下面这个：<https://blog.51cto.com/dengyong/1349794>。根据它的方法2，我发现开发板上TERM是vt220，于是修改了TERM的值：export TERM=xterm，source一下，颜色就出来了。人麻了。至此，根文件系统的版本和显示就已经完全搞定了，后面就要抓紧时间搞硬件了（尝试了一下安装板子的图形界面，但startx会报错找不到屏幕，看来只有老老实实搞VGA显示了）。

1. 戴口罩人脸识别

按照demo的思路，我们利用opencv或blog提供的级联分类器文件xml实现照片中人脸、戴口罩人脸和眼睛的寻找，其流程为：先寻找戴口罩的人脸（口罩不能是黑色），若没有找到，就寻找普通的人脸。当且仅当找到了人脸以后，才会寻找眼睛，其中寻找眼睛的位置预先设定，这就使得拍摄的照片不能过于离谱，导致实际眼睛位置不在预先设定的查找位置之中。只有人脸和眼睛都找到以后，我们对人脸进行处理，包括人脸对齐、次要边缘的剔除等等，最后把处理后的人脸和label配对保存（戴口罩的人脸和普通人脸共用label，但保存在不同的人脸vector中）。待得所有照片都处理完毕后，便可以进行训练。在识别阶段，我们将加载训练得到的两个yml文件得到model。对输入的照片，同样进行人脸和眼睛的查找，并根据有无戴口罩选择不同的model。只有人脸和眼睛都找到后，才会利用model进行预测。预测的阈值设为1.8时，发现预测结果同样正确。

1. 按键控制

这个部分走了很多弯路，主要还是感觉对硬件的功能、分类啊之类的不太熟悉导致的吧。因为是linux下的驱动，最开始查的资料都是linux设备树、内核之类的，也看了正点原子的文档，但他们都是直接在设备树文件dts里面改，可我们在硬件设计部分就已经把按键这一部分放进去了呀。另一方面，文档里的做法看起来也挺复杂的，驱动文件属实看不懂。然后又去看gpio的驱动，这里也是看的正点原子的文档，里面提到了什么linux的input之类的东西，也是在设备树里面添加，然后在/dev/input下面就会有event设备文件出现，但它需要知道每个按键的gpio号，去查板子的文档，发现没有相关的信息，也就不了了之。最后搜到了一个与axi\_gpio相关的资料，才知道驱动设备有MIO、EMIO、GPIO等方法，先去试了MIO之类的方法，如下：<https://www.cnblogs.com/milinker/p/5891100.html>。一方面，这里会直接在PS7的GPIO口生成端口，但我们硬件里是没有的；另一方面，他们是用SDK做的，参考价值不大。后来去搜axi\_gpio的方法，发现一片博客，是自己写的驱动：<https://www.its404.com/article/Mliameikoo/115051974>，本来想试试的，但它里面有个主设备号，我不知道是该自己设一个还是linux固定的，就想着有没有别的方法。又搜到几篇博客，讲的是sysfs方式驱动gpio，目录是在/sys/class/gpio下面。这个目录我前面查资料的时候是找到过的，但当时下面这几个文件夹是什么意思我都不知道，也不知道怎么用，只知道gpiochip1018下面的label和我们设备树里axi\_gpio\_1是一样的，当时就觉得这俩是对应的。<https://blog.csdn.net/gardenyou/article/details/103032322>、<https://www.cnblogs.com/Ph-one/p/8929547.html>、<https://www.cnblogs.com/hellokitty2/p/12500546.html>，这三篇博客让我豁然开朗。原来gpio目录下的东西就是和axi\_gpio一一对应的，回去看硬件设计图，用的也是axi\_gpio的ip，这下没得跑了，就用它。但这里涉及到了gpio号的问题，首先我们知道了那几个按键对应gpiochip1018，那么gpio的基地址就是1018，按照第三篇博客所说的，这四个按键的gpio号分别为1018/1019/1020/1021。尝试导入了一下，都是可以的，导入完成后去改了一下开关的状态，发现value变了，成了！接下来就可以写交互了。



记录一下板子上开关和gpio号的对应关系：

SW0🡪gpio1018、 SW1🡪gpio1019、 SW2🡪gpio1020、 SW3🡪gpio1021

后面又加了led，gpio号从1014开始，从低到高对应为：

led0🡪gpio1014、led1🡪gpio1015、 led2🡪gpio1016、 led3🡪gpio1017

10、用户交互

有了按键的基础，接下来就可以做程序和用户的交互了。我们可以读取各个gpio的value值（用文件打开），就可以判断用户是否更改了开关，从而确定当前状态。在程序运行时，每读入一张照片，就检测一次状态，使程序运行在相对应的mode中。为了使得程序的打印输出不重复，符合用户使用习惯（如每次进入某个状态，只输出一次提示信息，但开关的值是持续的，不像按键，因此花了很长时间来做限制），花了很多精力，思考了很多逻辑，人都麻了。制定的交互规则如下：

* 1. 程序启动后，处于MODE\_DETECT状态，只有当用户打开SW0时，才会进入MODE\_STARTUP状态；
  2. 在MODE\_STARTUP状态，用户可以自行选择想要运行的状态。若要进入MODE\_COLECT\_FACE，则调整SW1，每次开或关，则意味着收集下一个人的人脸；若要进入MODE\_TRAINING，则打开SW2，此时不管SW1是什么状态都会进入训练模式，注意开始训练有训练集的人脸数量要求，若不满足会报错，如何做到单次遇错只报错一次、每次遇到错误都会提示，就显得很麻烦，而且我还设置了多个报错类型，也涉及到模式之间的转换，花了我很长时间；若要进入MODE\_RECOGNITION，则打开SW3，此时不管SW1和SW2是什么状态，都会进入人脸识别模式，若还未训练就人脸识别，也会报错，遇到的问题同上，麻了。最后，当关闭SW0时，程序结束运行。

11、摄像头启动

本来一点思路都没有的，最开始想着把SDK的程序交叉编译出来到板子上运行，发现会出现编译错误，用的编译器都不一样，link也会出现问题，于是放弃了。后来改了头文件，就会出现段错误，至少程序可以执行了对吧。。。经过讨论知道了原理图里gpio0的作用就是输入，启动摄像头，这下思路就清晰了，直接像按键一样，修改value的值，不就可以完成摄像头的启动了吗？只需要研究一下SDK程序中的sendbyte函数，以及其中关键的DATA\_HIGH等函数，按照同样的方法往摄像头的寄存器里面写入数据，或许就可以实现启动了。gpio对应关系：低位对应gpio1022，高位对应gpio1023。

12、VGA显示

读了官方的文档和代码，VDMA的配置主要是和它的寄存器有关，因此，我们把VDMA的物理地址映射到文件空间，通过地址偏移的方法在程序中向它的寄存器写值，完成配置。主要是，我们在程序里需要将标识人脸的方框加上，然后在VGA显示，这就需要从VDMA的视频流中读取图片，修改完毕后，再将图片写回视频流。显然，这会出现读写不匹配的问题。通过官方的SDK代码，我们实现了图片的读取和写回，但总是会出现VGA显示跳跃、残影的问题，且更改读写方式（从VDMA正在读图片的地方读取图片，在VDMA正在写图片的地方写入图片），虽得到了改善，但图像总会闪烁。通过分析，这是三帧缓存的问题，可能总有一帧是黑色的。除此之外，为了解决读写不匹配的问题，我们不将VDMA的读写地址设为同一个地址，而是分开，按照我们自己的逻辑，往分开的地址中写入图片，再让VDMA读取我们输出的图片。为了读写速度匹配，我们又开了一个后台运行的程序，在主程序里面把人脸的位置记录到某一个文件当中，再在后台程序里打开文件，画好标识人脸的方框、identity、help等，最后写回视频流中。这就产生了多个程序之间数据同步的问题，挺麻烦的。这里还要记录一个小问题，但搞了我们很久。就是物理地址映射的时候，我们读写图片的地址映射太短了，导致最后显示是雪花图，我们还很久都没有发现。最后改为0x1000000后，正常显示了。

13、程序运行

不得不说，整个完整程序的逻辑真是太麻烦了。为了实现更高级的功能，需要有模式的切换、人脸识别模型的保存和程序开始时的读取、图像的显示等等，花了我不少心思。一开始我们用的opencv版本很低，它那个人脸识别的框架也很菜，找不到人脸，于是就换成了4.5.5的版本，利用余诗琪老师的CNN模型识别人脸，再用HSV面积判别是否戴口罩，然后用dnn分开训练、保存图片的特征向量和label，能实现完整的逻辑功能和模式切换。最后测试的时候，发现光线对功能有一定程度的影响（最开始发现两个不同的人的特征向量L2距离居然比本人和本人的更低，震惊！差点就换模型了，但换模型整个系统的数据存储和读取也得改，太麻烦了哈哈。最后发现是光线问题，拍照不要太骚），但总体的准确度是比较让人满意的。前前后后经历了接近两个月，挺不容易的，上课期间也要加班加点地做，很多ddl都堆起来了，但最后的结果还是让我们兴奋的。不容易啊！希望展示、答辩能取得好成绩啦！冲冲冲！

EtHereAlPlusor

2022年3月13日于浙大玉泉校区