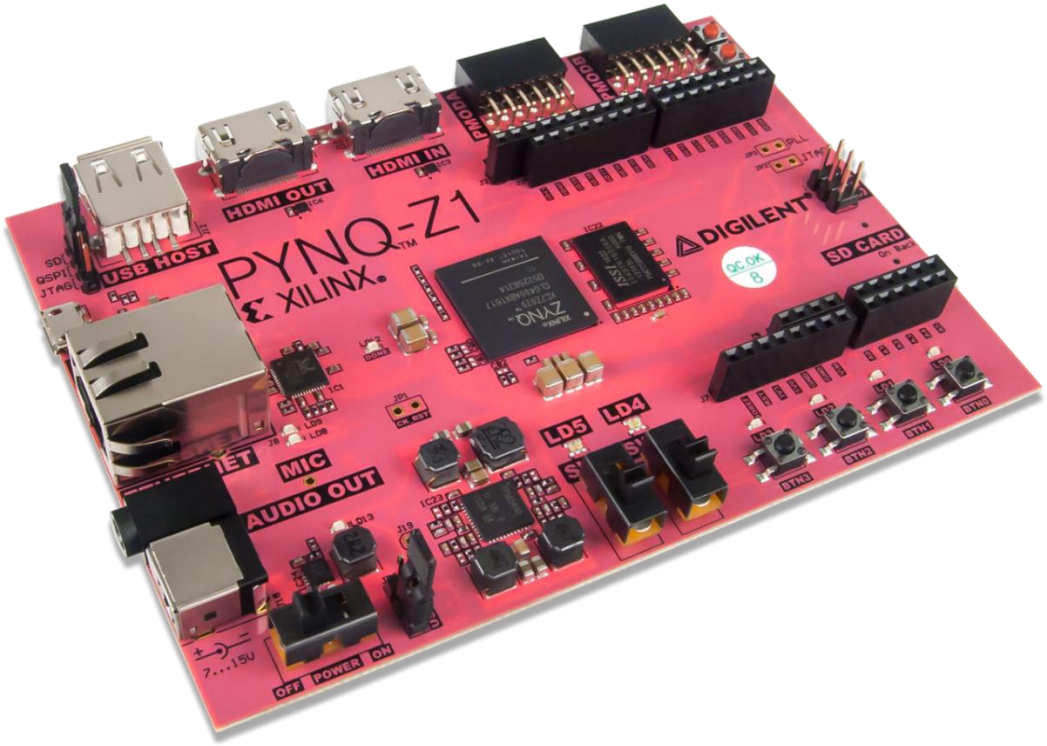
上海浦东新区张东路 1387 号 45 幢
电话：+86-21-58893151
邮箱：china.sales@digilent.com.cn

PYNQ-Z1 官方入门指导手册

在这份指导手册中，我们将说明如何配置硬件及软件平台，以及通过四个实例学习使用 Python 对 PYNQ-Z1 开发板编程。指导手册的内容包括：

- 软硬件准备
- PYNQ-Z1 硬件设置
- 连接到 Jupyter 进行在线编程
- 4 个 PYNQ 入门例程实践

在 PYNQ-Z1 的使用过程中，如有任何问题，欢迎访问：www.DIGILENT.com.cn/community （中文），或 www.PYNQ.io（英文）进行 反馈或寻求支持。



指导手册目录

指导手册目录 **2**

1. PYNQ 简介 **3**

2. Jupyter Notebook 简介 **4**

3. 软硬件准备 **4**

4. PYNQ-Z1 硬件设置 **5**

5. 连接到 Jupyter 进行在线编程 **7**

6. PYNQ 入门例程实践 **14**

实验一：按键控制 LED 14

实验二：录音及音频处理 16

实验三：使用 PmodOLED 19

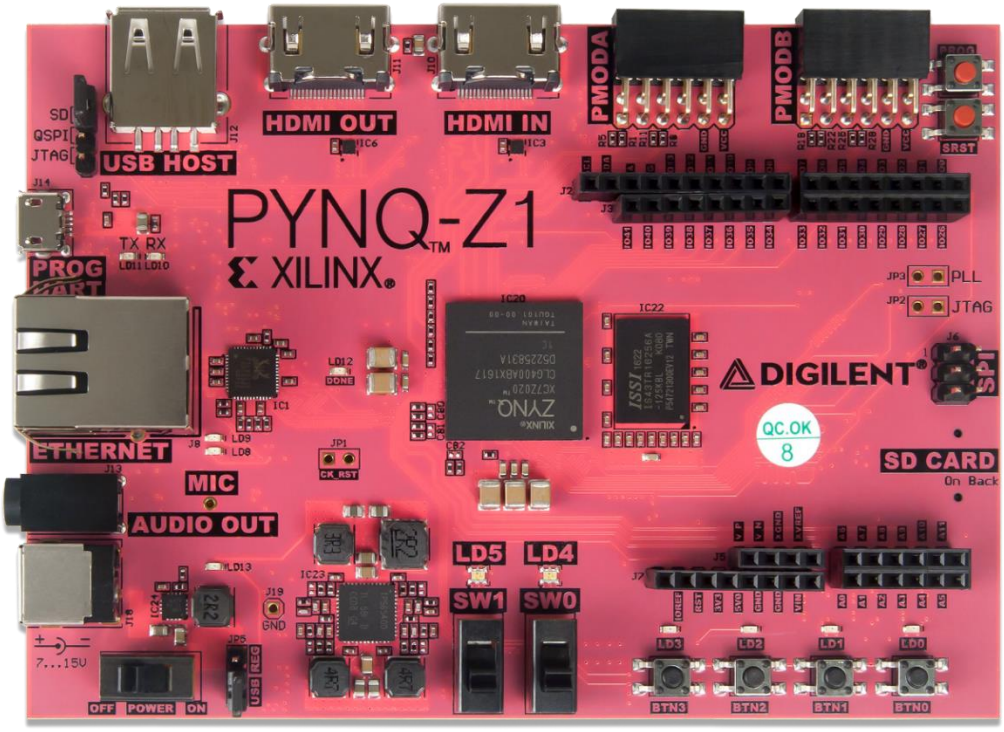
实验四：面部识别 21

1. PYNQ 简介

PYNQ-Z1 开发板支持 PYNQ 项目，这是一个新的开源框架，使嵌入式编程人员能够在无需设计可编程逻辑电路的情况下即可充分发挥 Xilinx Zynq All Programmable SoC（APSoC）的功能。与常规方式不同的是，通过 PYNQ，用户可以使用 Python 进行 APSoC 编程，并且代码可直接在 PYNQ-Z1 上进行开发和测试。通过 PYNQ，可编程逻辑电路将作为硬件库导入并通过其 API 进行编程，其方式与导入和编程软件库基本相同。

PYNQ-Z1 开发板是 PYNQ 开源框架的硬件平台。在 ARM A9 CPU 上运行的软件包括：

- 载有 Jupyter Notebooks 设计环境的网络服务器
- IPython 内核和程序包
- Linux
- FPGA 的基本硬件库和 API



2. Jupyter Notebook 简介

Jupyter Notebook 是一个基于浏览器的交互式编程计算环境。在使用 Jupyter Notebook 编程时，文件里可以包含代码、画图、注释、公式、图片和视频。当 PYNQ 开发板上安装好镜像文件，就可以在 Jupyter Notebook 里轻松地用 Python 编程，使用硬件库及 Overlay 控制硬件平台及交互。

3. 软硬件准备

1) 硬件准备

- PYNQ-Z1 开发板
- 以太网线
- Micro USB 数据线
- 空白 Micro SD 卡（最少 8GB 容量）

2) 软件准备

- 电脑上安装有支持 Jupyter 的浏览器

提示：以下浏览器的最新稳定版本可支持 Jupyter Notebook：*

✓ *Chrome*
✓ *Safari*
✓ *Firefox*

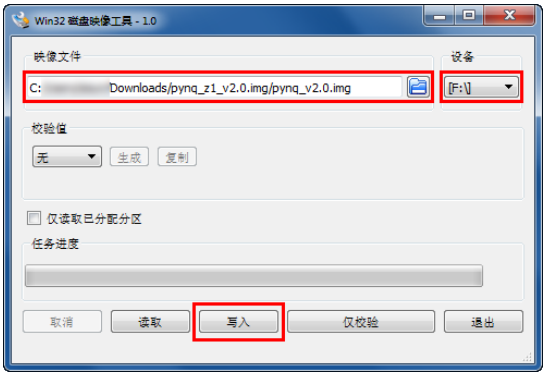
** 主要由 Jupyter Notebook 使用的 WebSockets 和可变沙箱模型决定*

不支持 Jupyter 的浏览器：

× *Safari，低于版本 5*
× *Firefox，低于版本 6*
× *Chrome，低于版本 13*
× *全部 Opera：CSS 渲染原因导致，但是执行时有可能可以用*
× *Internet Explorer 浏览器，低于版本 10*
× *Internet Explorer 浏览器，版本 10 及以上（同 Opera）*
× *基于 IE 的 360 浏览器*

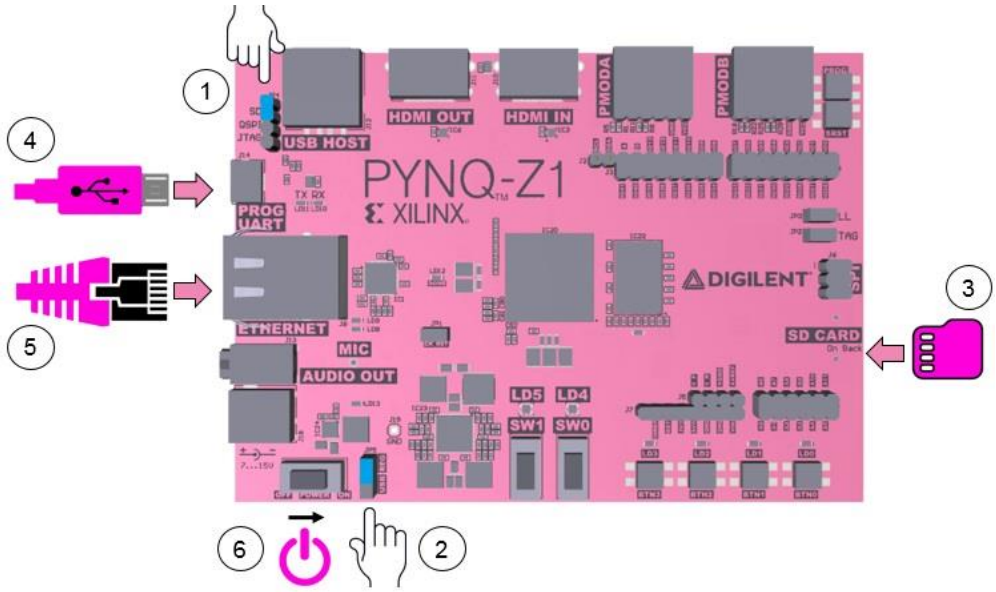
** 请注意，Safari 在 HTTPS（SSL 安全加密的超文本链接协议）和不可信证书下无法正常工作（主要是 WebSockets 无法正常工作）*

- 获取镜像文件
  - 下载 PYNQ-Z1 镜像文件并解压
  - 将空白的 SD 卡插入电脑（最小需 8GB 容量），烧写镜像文件
    - Windows 系统：使用 win32DiskImager 烧写。Image File 选择下载好的镜像文件。Device 选择 SD 卡的位置，一般会自动分配为 E 盘或 F 盘。



    - Linux 系统/MacOS：使用系统自带 dd 命令，在不同操作系统上烧写 Micro SD 的操作细节，可参考教程 Writing the SD card image

4. PYNQ-Z1 硬件设置



1. 如上图所示，将跳帽插在最上边两个排针上，设置 boot 跳线（板上标记的 JP4）到 SD 位置，选择为从 SD 卡驱动
2. 要想通过 Micro USB 线对 PYNQ-Z1 进行供电，需如图所示将跳帽插在的最下边两个排针上，设置电源跳线（JP5）到 USB 的位置（你也可以使用 12V 外部电源对 PYNQ-

Z1 进行供电，将跳帽插在的最上边两个排针上，设置电源跳线（JP5）到 REG 的位置）
3. 将已安装镜像文件的 SD 卡插入 SD 卡槽（如图所示，SD 卡槽在开发板下方右侧边缘）
4. 使用 Micro USB 线将 PYNQ 开发板的 PROG UART（J14）接口连接到电脑。这将用来给 PYNQ 供电以及作为串口通信
5. 使用网线将 PYNQ 开发板连接到路由器或电脑（根据网线端口的选择，后续操作会有不同）*
6. 将开关拨到 ON 以打开 PYNQ，等待系统启动。大约一分钟后将有两个蓝色的 LED 和四个黄绿色的 LED 同时闪动，随后蓝色 LED 关闭，四个黄绿色的 LED 灯亮。此时系统启动完毕。

*** 关于板载以太网连接的详细说明**

你可以将 PYNQ-Z1 的以太网接口和以下设备连接：

- 连接到一个路由器或者交换机上，与你的电脑在同一网络下
- 直接连在电脑的以太网接口上

可以的话，请将你的开发板连接到一个具有以太网访问的网络上。这可以让你更新板子上的软件并可以安装新的软件包。

- **连接到网络**

如果你通过 DHCP 服务器连接到一个局域网络，你的板子会自动获取一个 IP 地址，你必须保证有足够的权限通过网络访问到设备，否则板子可能无法正常访问。

路由器/网络交换机（DHCP）

1. 将板载以太网接口连接到路由器/交换机上
2. 通过浏览器访问 http://pynq:9090
3. 更改主机名称（根据自身需求）
4. 配置代理（根据自身需求）

- **直接连接到电脑**

此时，你需要一台有以太网接口的电脑，同时你需要拥有配置网络接口的权限。通过直接相连，你就可以访问使用 PYNQ-Z1 了。但是这里需要注意，除非你能将以太网与电脑上

具有 Internet 访问的连接进行桥接，否则你的 PYNQ-Z1 是无法访问 Internet 的。在没有 Internet 连接的情况下，你不能更新或者加载新的软件包。

直接连接你的电脑（静态 IP）

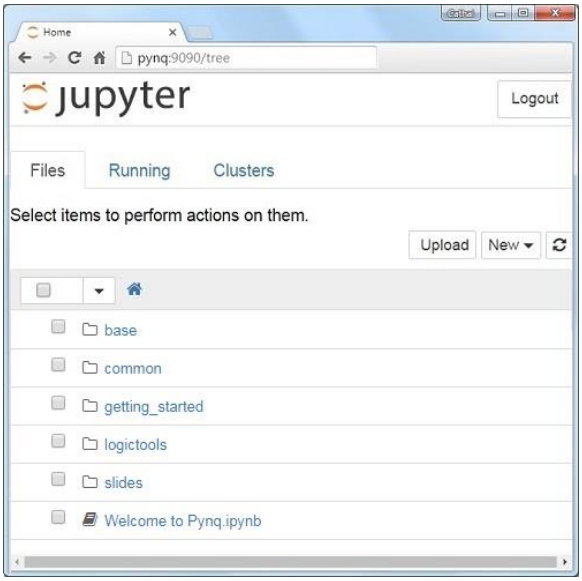
1. 给电脑配置一个静态的 IP
2. 将板载以太网接口与电脑的直接相连
3. 访问 http://192.168.2.99:9090

*如何配置静态 IP 请参见 Assign your PC/Laptop a static IP address

5. 连接到 Jupyter 进行在线编程

如果 PYNQ 通过网线连接到了路由器，PYNQ 将被自动分配地址。打开 http://pynq:9090，用户名和密码都是 xilinx，输入后即可进入以下界面。

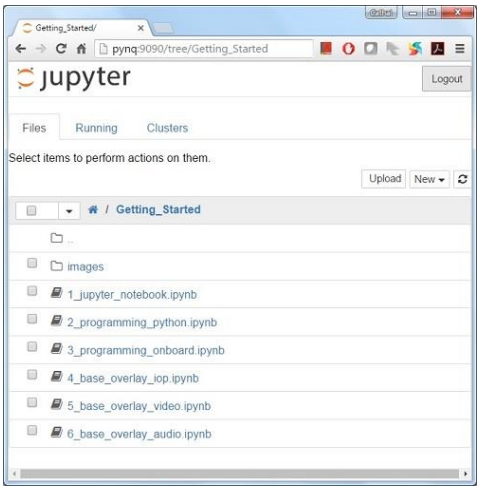
如果 PYNQ 通过网线连接到了电脑，需要先设置电脑的 IP 地址，参考 Assign your PC/Laptop a static IP address，然后再打开 http://192.168.2.99:9090。同样，输入用户名及密码 xilinx，即可进入以下界面。



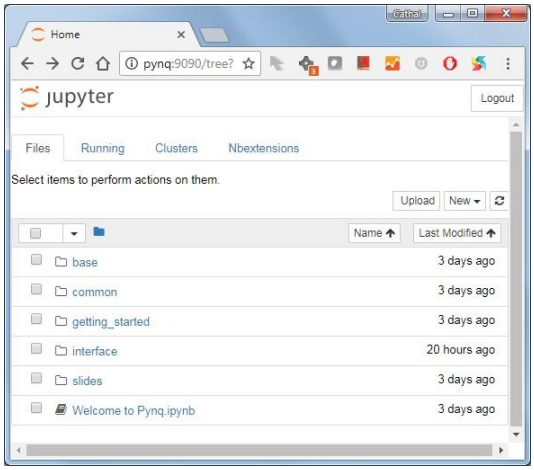
默认的主机名是 pynq，默认静态 IP 地址是 192.168.2.99。如果你改变了主机名称或者板子上的静态 IP 地址，你需要改变你访问的地址。第一次连接时，电脑会花费几秒钟的时间来确定主机名和 IP 地址。

PYNQ 通过 Jupyter Notebook 的形式来提供各种示例文档。你可以以网页形式浏览这些示例项目文档，或者如果你有一个正在运行 PYNQ 镜像的板子，你可以可交互式地查看并运行这些 Notebook 文档。

你也可以在 Jupyter 主页上的 Getting_Started 文件夹中找到可以使用的 Notebook 文档。这里也有许多示例文档来展示如何使用各种板载设备。



此外，我们还提供了一些样例展示如何使用不同的板载外围设备。



目前，所有我们已对所有这些示例文档进行了分类：

- common: 无针对性 overlay 的示例项目
- base: 与 PYNQ-Z1 base overlay 相关的示例项目
- logictools: 与 PYNQ-Z1 logictools overlay 相关的示例项目

当你打开一个笔记本并作出任何修改，或者执行代码片段，notebook 文档都将会被更改。这就需要你打开一个新的 notebook 时做好备份。如果你需要恢复原始版本，你可以从 PYNQ Github 项目页面上下载全部笔记本。

在 Running 一栏下，则可以看到正在运行的项目。

- **访问板载文件**

在 PYNQ 板上，运行有一个文件共享服务：Samba。通过它，板子上的主目录可以作为网络驱动器访问，同时你可以将文件在板子和电脑间传递。

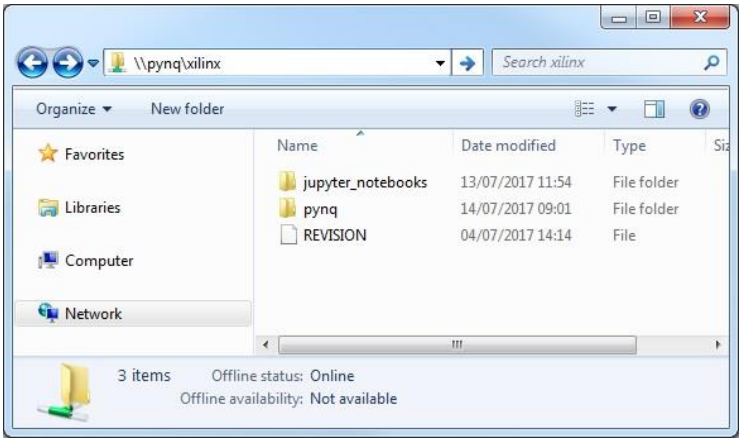
在 Windows 下，访问 PYNQ 主目录你可以键入：

```
\\pynq\xilinx               # If connected to a Network/Router with DHCP
\\192.168.2.99\xilinx       # If connected to a Computer with a Static IP
```

或者在 Linux 下：

```
smb://pynq/xilinx            # If connected to a Network/Router with DHCP
smb://192.168.2.99/xilinx    # If connected to a Computer with a Static IP
```

然后会跳出下图：

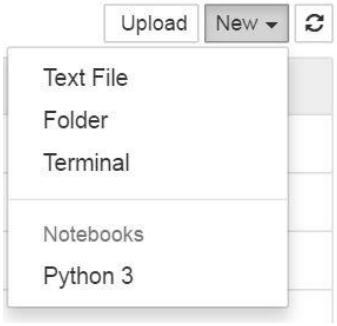


Samba 服务器的用户名和密码都是 xilinx。

注意：如果必要，请修改主机名/IP 地址。

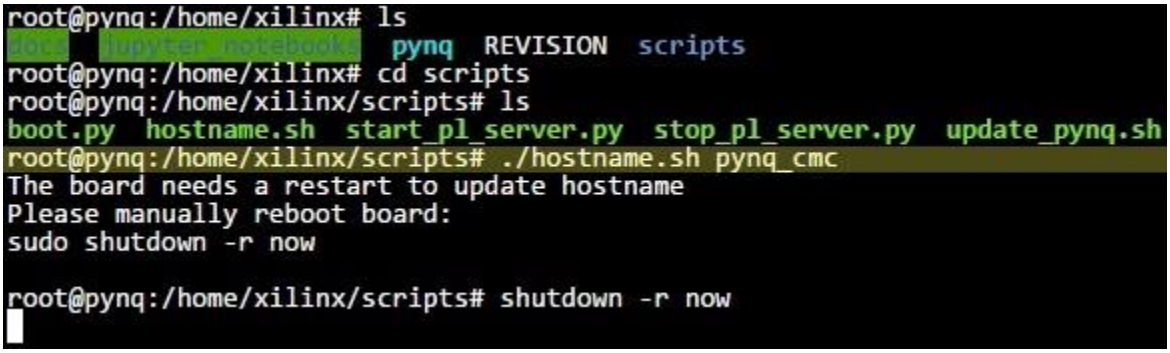
- **更改 hostname**

如果你连接在一个其它 PYNQ-Z1 开发板可能已经连接入的网络下，建议你立即更改你的主机名称。通常，这种情况在工作或者校园环境下会比较常见。PYNQ 的默认 hostname 是 pynq，终端被内嵌在 Jupyter 中。在 Jupyter 的主页 pynq:9090 界面中打开 New>>Terminal，你将以 root 权限在浏览器中打开一个终端：



在 Terminal 里输入以下指令更改 hostname（使用你自己希望给板子设置的主机名来替换 NEW_HOST_NAME 的位置）：

```
sudo /home/xilinx/scripts/hostname.sh NEW_HOST_NAME
```



然后需要重启 PYNQ 才能生效（使用新的主机名重新连接）：

```
sudo shutdown -r now
```

注意：如果你以 root 权限登录，则不需要使用 sudo。但是如果使用 xilinx 进行登录，sudo 必须添加在这些命令之前。如果你不能访问你的板子，浏览下面的步骤以通过 micro USB 线来打开终端。

- **通过 USB 接口连接电脑终端**

如果你需要修改板载设置，但是无法访问通过 Jupyter 访问终端，你可以借助 USB 接口，通过电脑终端控制 PYNQ。此时我们需要安装一个终端工具，比如 PuTTY 或者 Tera Term。为了打开终端，你需要知道开发板所在 COM 端口。

在 Windows 上，你可以在控制面板打开 Windows 设备管理器进行查看

- 打开设备管理器，展开端口项
- 找到 USB 串口所在 COM 端口，例如 COM5

一旦你知道了 COM 端口，打开 PuTTY 并使用下列设置：

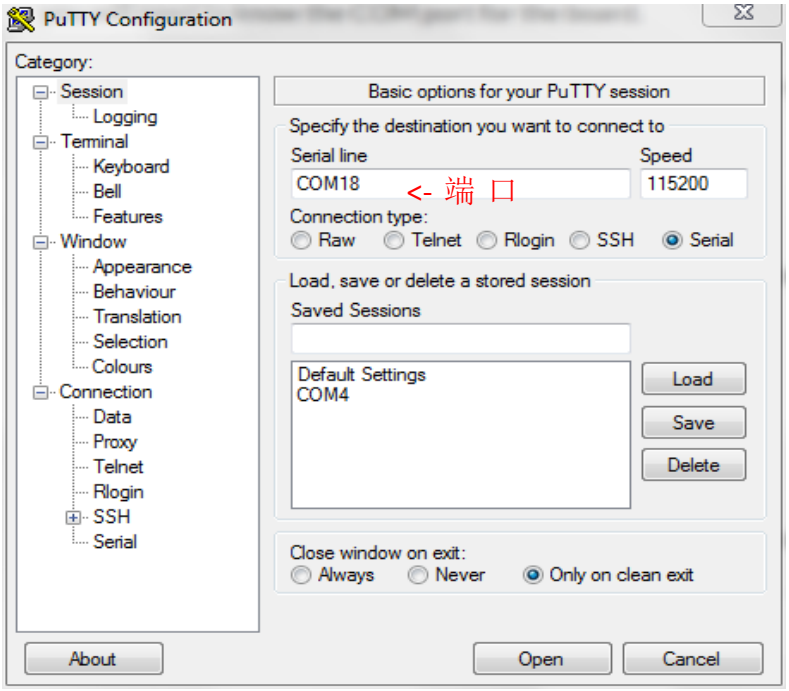
- 选择 Serial
- 输入 COM 端口号
- 输入波特率
- 然后点击 Open 启动

在终端窗口中按 Enter（回车）以确保你能看到命令行：

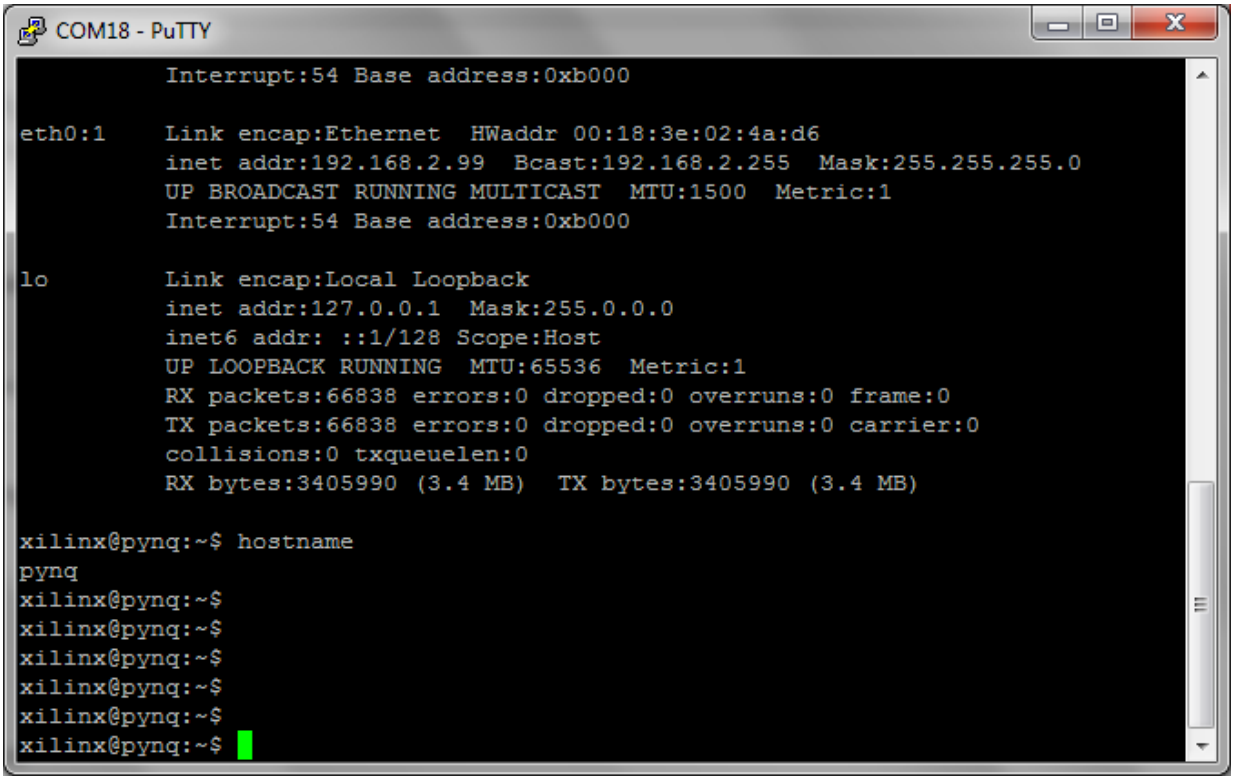
```
xilinnx@pynq:/home/xilinx#
```

完整的终端设置如下：

- 115200 baud
- 8 data bits
- 1 stop bit
- No Parity
- No Flow Control



在终端中按回车键，出现 xilinx@pynq:~$，即可输入指令控制 pynq。比如输入 hostname 查看名称，输入 ifconfig 查看 IP 地址等。



- **设置代理**

如果你的开发板连接到的是使用代理的网络中，你需要在板上设置代理服务器。按照上方教程打开终端，并输入下列命令并将“my_http_proxy:8080”和“my_https_proxy:8080”更改为你自己的设置：

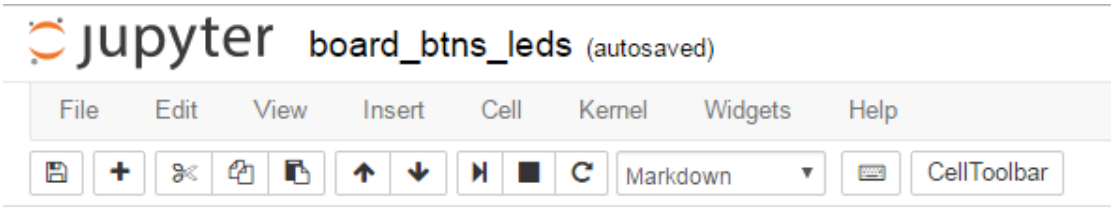
```
set http_proxy=my_http_proxy:8080
set https_proxy=my_https_proxy:8080
```

6. PYNQ 入门例程实践

下面就以四个实验作为例子，简单介绍如何使用 Jupyter 在线编程工具，以及如何用 python 语言对 PYNQ 编程。如需更多资料及示例项目，可以查看：www.pynq.io

实验一：按键控制 LED

打开 base>board 文件夹中的 board_btn_leds.ipynb 文件。点击工具栏的 run 图标或者选择 Cell->Run 运行代码。



这个项目中，按下 PYNQ 开发板上的按键 0 可改变彩色 LED 的颜色，按键 1 可开启从右到左的流水灯，按键 2 可开启从左到右的流水灯，按键 3 结束运行。

代码如下：

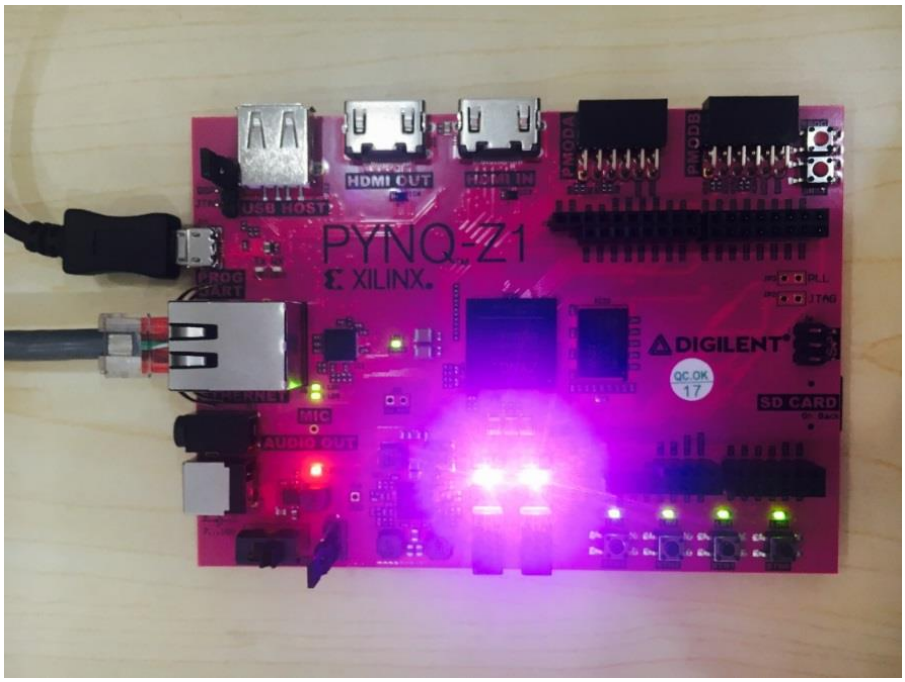
#导入 PYNQ 开发板的支持文件

```
from time import sleep
from pynq.overlays.base import BaseOverlay

base = BaseOverlay("base.bit")

Delay1 = 0.3
Delay2 = 0.1
color = 0
```

#定义寄存器

```
rgbled_position = [4,5]

for led in base.leds:
    led.on()
while (base.buttons[3].read()==0):
    if (base.buttons[0].read()==1):
```

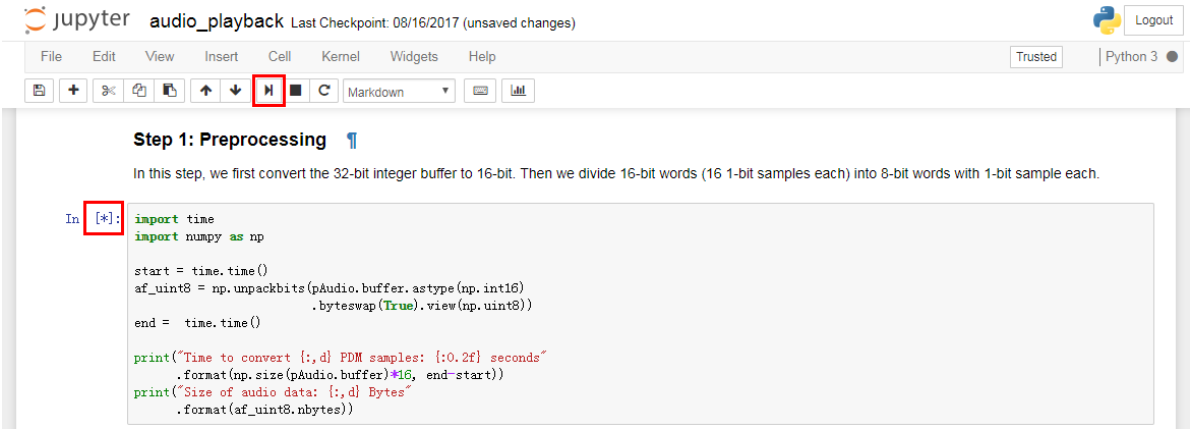
```
            color = (color+1) % 8
            for led in rgbled_position:
                base.rgbleds[led].write(color)
                base.rgbleds[led].write(color)
            sleep(Delay1)

        elif (base.buttons[1].read()==1):
            for led in base.leds:
                led.off()
            sleep(Delay2)
            for led in base.leds:
                led.toggle()
                sleep(Delay2)

        elif (base.buttons[2].read()==1):
            for led in reversed(base.leds):
                led.off()
            sleep(Delay2)
            for led in reversed(base.leds):
                led.toggle()
                sleep(Delay2)

print('End of this demo ...')
for led in base.leds:
    led.off()
for led in rgbled_position:
    base.rgbleds[led].off()
```

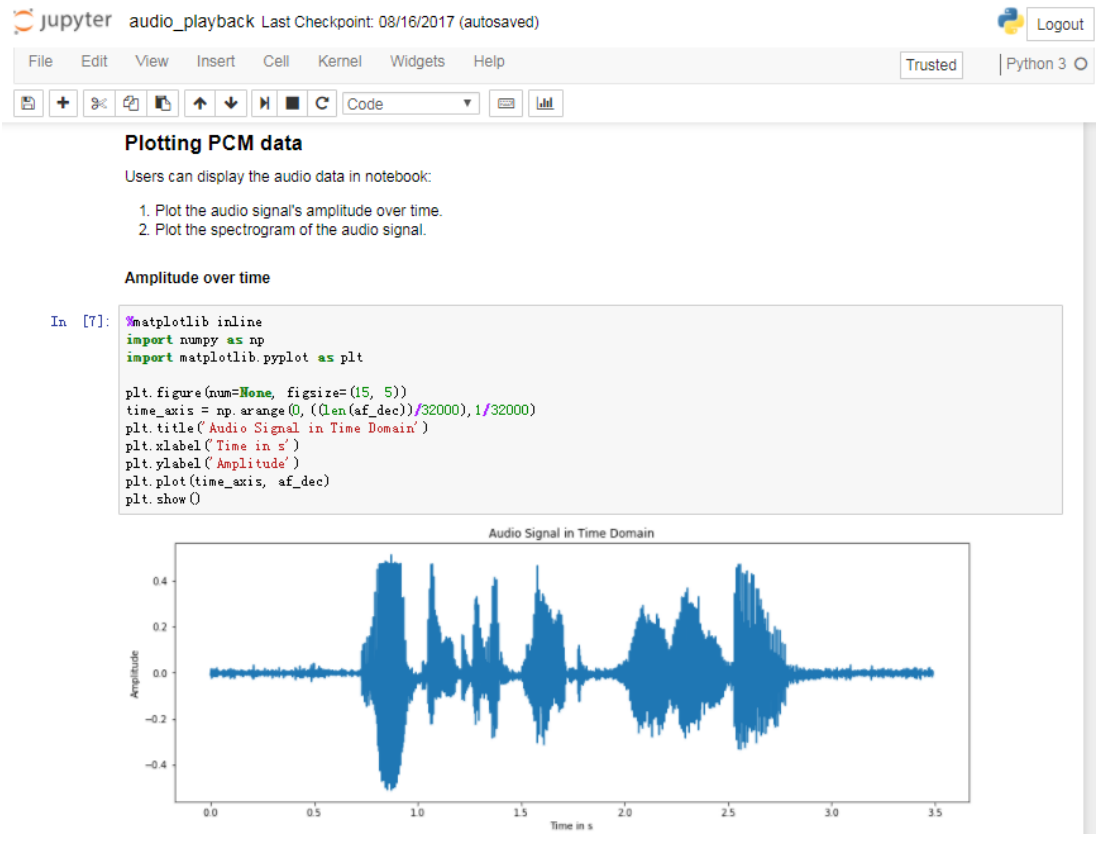


实验二：录音及音频处理

进入 base>audio 文件夹，点击 audio_playback.ipynb

代码被分成一个个代码块，每一段代码前解释了这一段的功能。点击工具栏的 run 图标或者选择 Cell->Run 依次运行下面的代码块。在运行中的代码块会被标为[*]，如图，此时应该等待运行结束，不能进行下一步操作。运行结束后，[*]会变成数字，表示这是第几段被运行的代码，以记录顺序。





Python 代码如下：

```
#创建一个新的音频对象
from pynq.overlays.base import BaseOverlay
base = BaseOverlay("base.bit")
pAudio = base.audio
# 录制一个三秒的音频文件并保存
pAudio.record(3)
pAudio.save("Recording_1.pdm")
# 播放刚才录制的音频文件
pAudio.play()

# 加载一个预先录制好的音频文件并播放
pAudio.load("/home/xilinx/pynq/lib/tests/pynq_welcome.pdm")
pAudio.play()

#预处理
#首先将 32 位整型缓冲器转换位 16 位。然后将 16 位字分为每个带一位采样的 8 位字
import time
import numpy as np

start = time.time()
af_uint8 = np.unpackbits(pAudio.buffer.astype(np.int16)
                         .byteswap(True).view(np.uint8))
end =  time.time()

print("Time to convert {:,d} PDM samples: {:0.2f} seconds"
      .format(np.size(pAudio.buffer)*16, end-start))
print("Size of audio data: {:,d} Bytes"
      .format(af_uint8.nbytes))

#通过抽取的方式将 PDM 信号转换为 PCM 格式，采样率从 3MHz 降为 32kHz。
import time
from scipy import signal

start = time.time()
af_dec = signal.decimate(af_uint8,8,zero_phase=True)
af_dec = signal.decimate(af_dec,6,zero_phase=True)
af_dec = signal.decimate(af_dec,2,zero_phase=True)
af_dec = (af_dec[10:-10]-af_dec[10:-10].mean())
end = time.time()
print("Time to convert {:,d} Bytes: {:0.2f} seconds"
      .format(af_uint8.nbytes, end-start))
print("Size of audio data: {:,d} Bytes"
      .format(af_dec.nbytes))
del af_uint8

#在浏览器中播放转换完的音频信息
from IPython.display import Audio as IPAudio
IPAudio(af_dec, rate=32000)
```

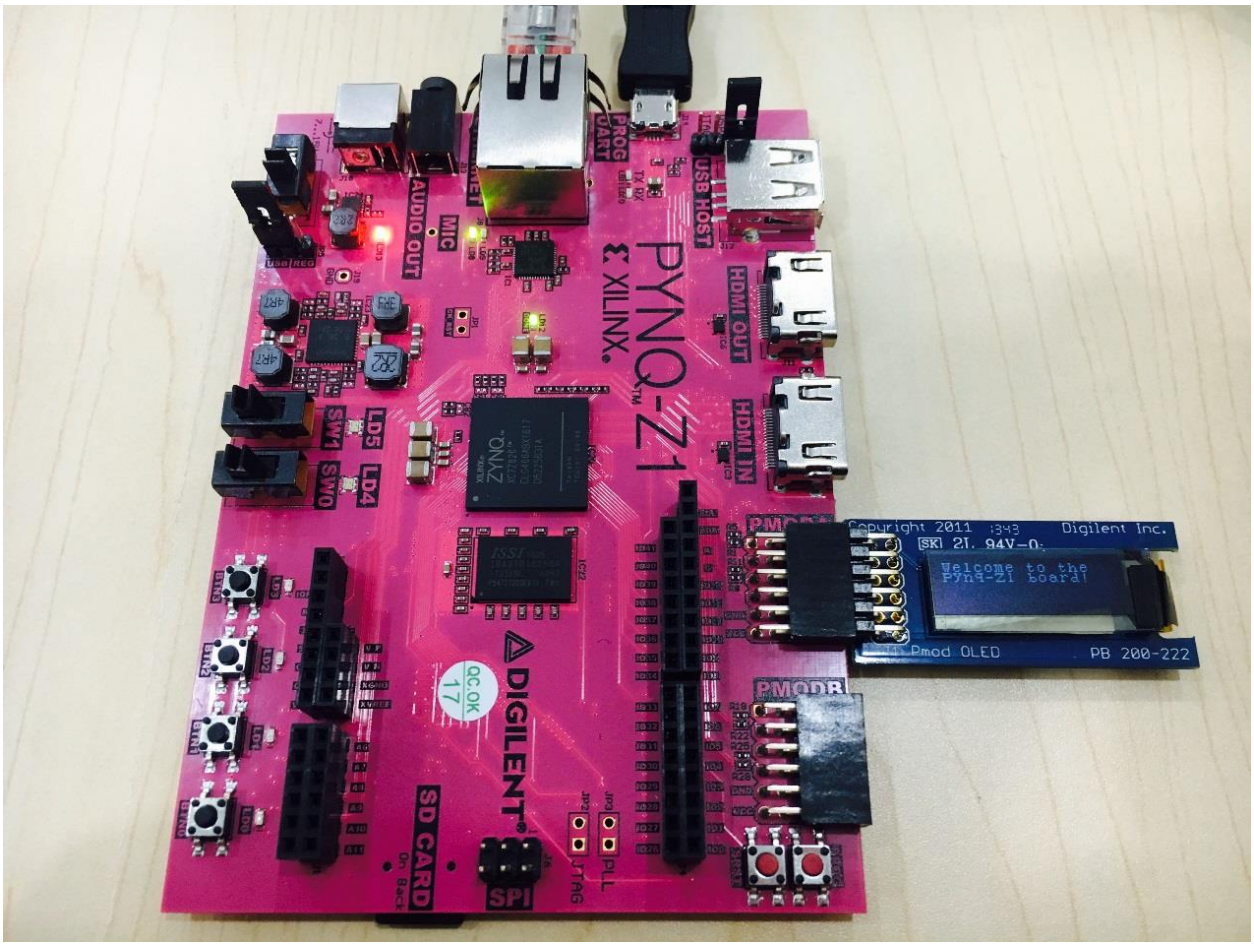
```
#绘制 PCM 数据
# 信号随时间推移的振幅
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

plt.figure(num=None, figsize=(15, 5))
time_axis = np.arange(0,((len(af_dec))/32000),1/32000)
plt.title('Audio Signal in Time Domain')
plt.xlabel('Time in s')
plt.ylabel('Amplitude')
plt.plot(time_axis, af_dec)
plt.show()

#绘制信号的频谱（按振幅大小）
from scipy.fftpack import fft

yf = fft(af_dec)
yf_2 = yf[1:len(yf)//2]
xf = np.linspace(0.0, 32000//2, len(yf_2))

plt.figure(num=None, figsize=(15, 5))
plt.plot(xf, abs(yf_2))
plt.title('Magnitudes of Audio Signal Frequency Components')
plt.xlabel('Frequency in Hz')
plt.ylabel('Magnitude')
plt.show()

#绘制传统的随时间推移的信号频谱图
import matplotlib

np.seterr(divide='ignore',invalid='ignore')
matplotlib.style.use("classic")
plt.figure(num=None, figsize=(15, 4))
plt.title('Audio Signal Spectogram')
plt.xlabel('Time in s')
plt.ylabel('Frequency in Hz')
_ = plt.specgram(af_dec, Fs=32000)
```

实验三：使用 PmodOLED

base>pmod 文件夹里提供了多种 Pmod 设备的程序，在这里我们以 PmodOLED 为例。

在 PYNQ 处于关闭状态时，将 PmodOLED 插入 PYNQ 的 JA 接口，如图。然后再上电，等待四个黄绿色 LED 亮起，说明已启动。

然后打开 pynq:9090，找到 base>pmod 文件夹下的 pmod_oled.ipynb 文件并运行。



代码如下：

#导入 io 定义

```
from pynq.overlays.base import BaseOverlay
from pynq.lib import Pmod_OLED

base = BaseOverlay("base.bit")
```

#初始化，连接 PmodOLED 到 PmodA

```
pmod_oled = Pmod_OLED(base.PMODA)
```

#清屏并输出字符

```
pmod_oled.clear()
pmod_oled.write('Welcome to the\nPynq-Z1 board!')
```

#清屏，输出新的字符（可以尝试输出其他信息）

```
pmod_oled.clear()
pmod_oled.write('Python and Zynq\nproductivity &  performance')
```

#读取 ip 地址并输出

```
def get_ip_address():
    ipaddr_slist = !hostname -I
    ipaddr = (ipaddr_slist.s).split(" ")[0]
    return str(ipaddr)

pmod_oled.clear()
pmod_oled.write(get_ip_address())
```

实验四：面部识别

将 PYNQ 开发板的 HDMI IN 连接到电脑的 HDMI 接口，PYNQ 开发板的 HDMI OUT 连接到显示器。

打开 base>video 文件夹下的 opencv_face_detect_hdmi.ipynb。按照提示依次运行代码。

```
from pynq.overlays.base import BaseOverlay
from pynq.lib.video import *

base = BaseOverlay("base.bit")
hdmi_in = base.video.hdmi_in
hdmi_out = base.video.hdmi_out
```

#初始化

```
hdmi_in.configure(PIXEL_RGB)
hdmi_out.configure(hdmi_in.mode, PIXEL_RGB)

hdmi_in.start()
hdmi_out.start()
```

#输入待检测的图片（截屏）

```
import PIL.Image
frame = hdmi_in.readframe()
img = PIL.Image.fromarray(frame)
img.save("/home/xilinx/jupyter_notebooks/base/video/data/face_detect.jpg")

img
```

#使用 opencv 进行面部检测

```
import cv2
import numpy as np

frame = hdmi_in.readframe()

face_cascade = cv2.CascadeClassifier(
    '/home/xilinx/jupyter_notebooks/base/video/data/'
    'haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier(
    '/home/xilinx/jupyter_notebooks/base/video/data/'
    'haarcascade_eye.xml')
```

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.3, 5)

for (x,y,w,h) in faces:
    cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)
    roi_gray = gray[y:y+h, x:x+w]
    roi_color = frame[y:y+h, x:x+w]

    eyes = eye_cascade.detectMultiScale(roi_gray)
    for (ex,ey,ew,eh) in eyes:
        cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
```

#将 OpenCV 面部识别结果输出到显示器

```
hdmi_out.writeframe(frame)
```

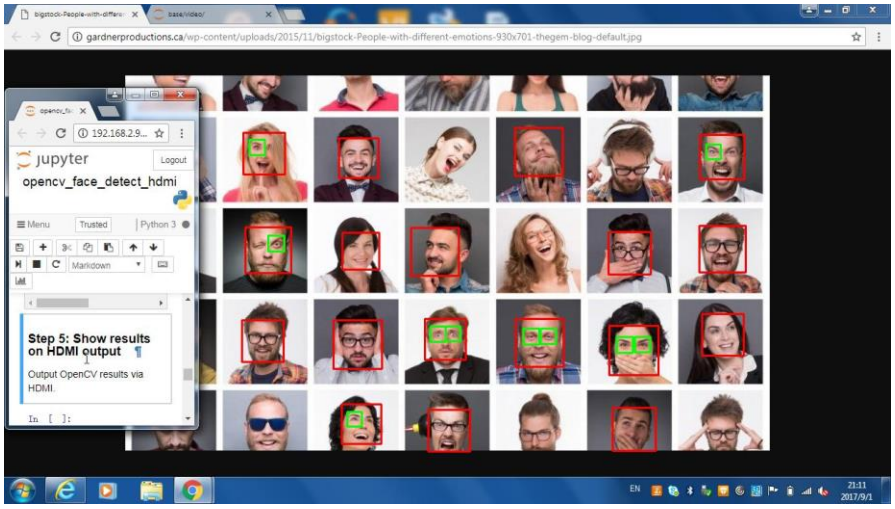
#在 Jupyter 中显示 OpenCV 面部识别结果并保存为 JPEG 格式图片

```
img = PIL.Image.fromarray(frame)
img.save("/home/xilinx/jupyter_notebooks/base/video/data/face_detect.jpg")

img
```

#关闭 HDMI

```
hdmi_out.stop()
hdmi_in.stop()
del hdmi_in, hdmi_out
```

PYNQ 面部识别输出结果：



有意思的是，当用手机拍摄 PYNQ 识别的结果时，手机也自动对图片进行了面部识别。可以看到，蓝色框是 PYNQ 识别的结果，黄色框是 Iphone 识别的结果。这个项目使用的算法似乎还是略胜一筹。