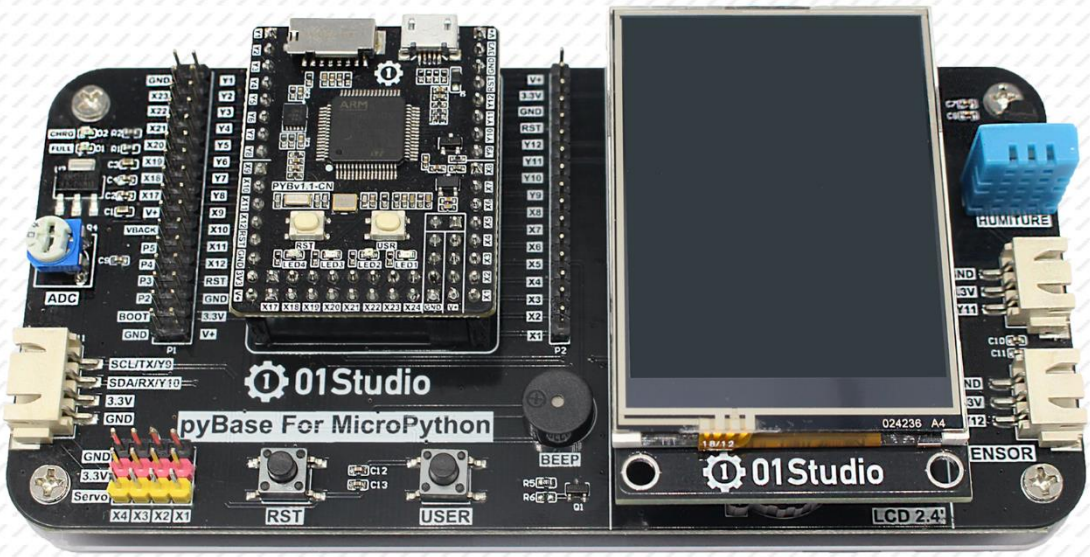
MicroPython 从0到1

用python做嵌入式编程

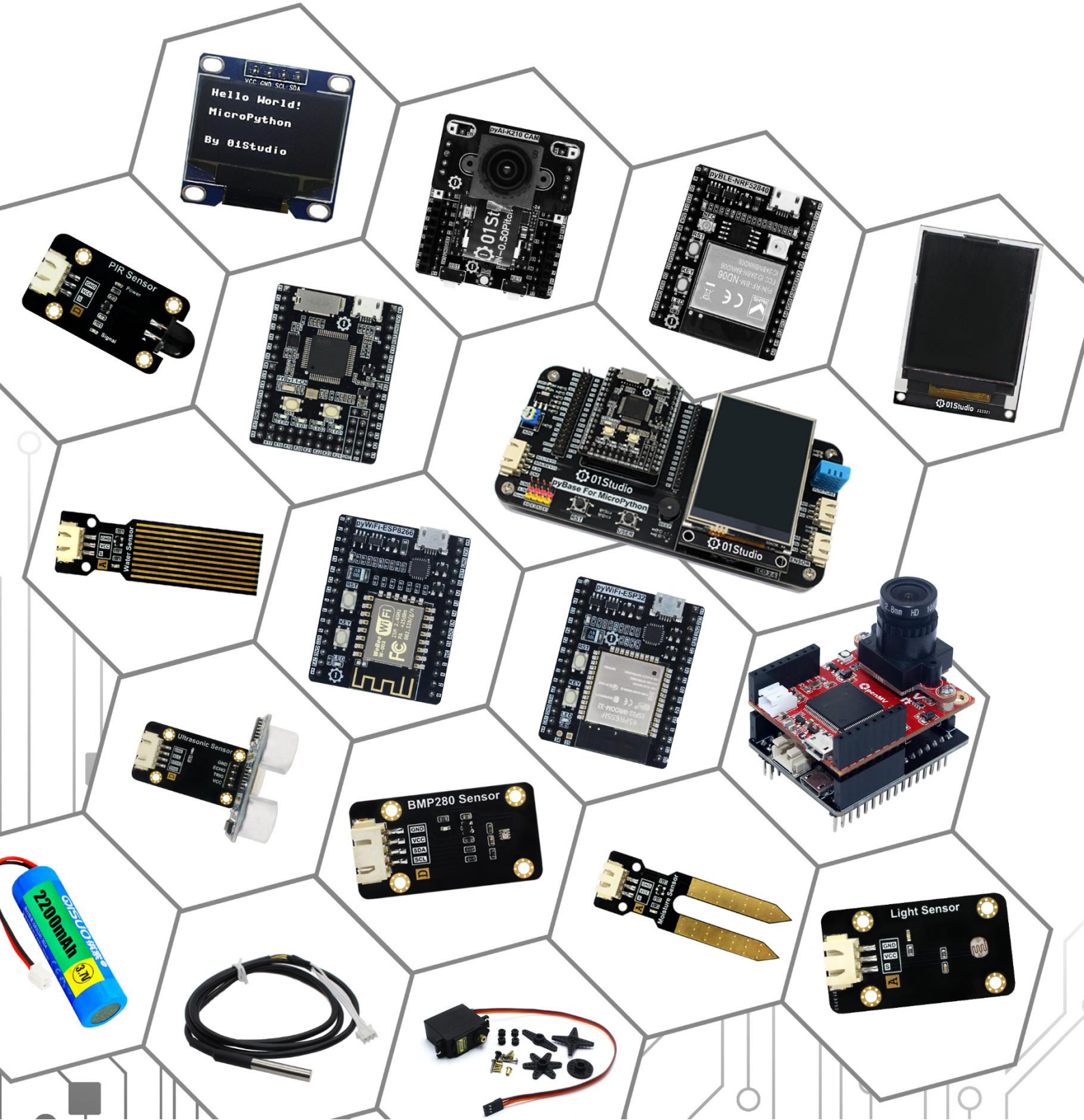
01Studio团队 编著



01Studio
-让编程变得简单有趣-

MicroPython

产品家族



前言

为什么学习 MicroPython？

单片机嵌入式编程经历了汇编、C 语言的发展历程，可以说是一次编程革命，其背后的原因是单片机的速度越来越快，集成度越来越高。而这一趋势并没停止，摩尔定律仍然适用。在未来，单片机上很可能直接跑机器语言。

在 2014 年，MicroPython 在英国诞生了，对于电子爱好者来说无疑拉开了新时代的序幕，用 python 这个每年用户量不断增长的编程语言来开发嵌入式，加上无数开源的函数模块，让嵌入式开发变得从未如此的简单。

MicroPython 致力于兼容 Python。因此，我们在学习完 MicroPython 后除了可以开发有趣的电子产品外，还可以继续深入使用 Python 语言去开发后台、人工智能等领域。

为什么要写《MicroPython 从 0 到 1》教程？

对于初学者，常常需要浪费大量时间来搭建开发环境和安装应用软件，以及需要从不同渠道寻找开发资料。MicroPython 作为新兴的东西，市面上的资料更是参差不齐，被搞得头昏目眩。

“让编程变得简单有趣”作为我们 01Studio 团队的使命，我们一直在寻找最优的学习方法。力求以最简单易懂的方式来讲述 MicroPython 的学习方法，从开发环境快速建立、基础实验、传感器实验到项目进阶实验。《MicroPython 从 0 到 1》诞生了。相比于传统的出版图书，我们会以更平易近人的口吻讲解实验，配上精美的插图，每一行代码都是自己的亲身经历，目的就是让大家更好的入门 MicroPython，将精力放在编程和开发中去。

为什么要打造 MicroPython 学习套件？

MicroPython 在中国是一个新兴的东西，前途无限，但是网上的学习模块套件参差不齐，大多是复制官方开源的开发板来设计，用过的就知道，外国的电路设计跟国内的风格很不同，甚至常常让初学者钻牛角尖。为此，01Studio 团队特意打造的中国风的 MicroPython 开发套件，《MicroPython 从 0 到 1》上的例程也是基于此板开发的，每个例程都能直接跑起。通过一系列系统学习，你甚至可

以用它来完成你的比赛或项目。

为什么要打造 01Studio 社区论坛？

早在数年前我们打造了 WeBee 品牌，专注物联网开发，到现在已经服务了数万名学生、教师和工程师等相关人员。早期依靠着群来维护，但随着开发者的数量增加，我们发现仅依靠群或者技术支持人员已经不能满足需求。

社区论坛是很好的学习交流地方，在这里你可以学习到前人的经验，可以通过搜索内容来解决问题。为什么全世界很火的开源硬件都是由外国人做起来，我们认为很重要的一个点是源于开源和分享精神。大部分开发者都会想着从论坛或网站解决自身问题而不愿意奉献，而我们希望 01Studio 社区是一个大家乐于发表文章和分享心得的地方。我们始终始终坚持开源原则，包括书籍内容、所有例程代码和部分硬件模块的开源。

期待你加入 01Studio 社区大家庭，成为我们的一份子，一起成长。

【社区链接：www.01Studio.org】

【微信公众号：01Studio 社区】

01Studio

2019.4 于深圳

版本说明

版本	日期	作者	更新内容
v1.0	2019-6-1	Jackey	1.第 1 版正式发布
v1.1	2019-7-8	Jackey	1.增加 STM32 平台拓展实验
v2.0B	2019-8-20	Jackey	1.增加 ESP8266/ESP32 平台教程
v3.0	2019-10-31	Jackey	1.增加 OpenMV4 平台教程 2.STM32 平台加入串口通信实验 3.ESP 平台更换 MQTT 助手
v3.1	2019-11-31	Jackey	1.STM32 平台新增以太网模块教程 2.OpenMV 平台新增多路舵机模块、WiFi 模块、摄像头模块等教程。
v4.0	2020-2-15	Jackey	1.增加 K210 平台教程
v4.1	2020-4-24	Jackey	1.K210 平台新增 WiFi 模块应用 2.增加 openmv4 Plus 介绍
v5.0	2020-6-15	Jackey	1.增加 NRF52840 平台教程
v5.1	2020-8-5	Jackey	1.增加 LVGL 教程（基于 ESP32）
v5.2	2020-10-16	Jackey	1.修正部分错误 2.增加 pyboard Mini 和 pyBoard plus 产品介绍

版权声明

《MicroPython 从 0 到 1》由 01Studio 团队打造，已于深圳市版权局注册备案，任何单位或个人引用相关文字、图片或相关内容请注明出处【01Studio】，否则我们将保留追究相关法律责任的权利。

目录

第一部分 MicroPython 基础知识 13

第 1 章 MicroPython 简介 13
1.1 MicroPython 是什么 13
1.2 MicroPython 支持的微控制器平台 14
1.3 MicroPython 相关学习资料 15
1.3.1 01Studio 社区 15
1.3.2 MicroPython 文档（中文） 16
1.3.3 MicroPython 官方网站 17
1.4 MicroPython 开发套件介绍 18
1.4.1 STM32 平台 19
1.4.2 ESP8266 平台 26
1.4.3 ESP32 平台 29
1.4.4 NRF52840 平台 34
1.4.5 OpenMV4 平台 37
1.4.6 K210 平台 43
1.4.7 pyBase 46
1.4.8 IOT/通讯模块 48
1.4.9 传感器模块 51
1.4.10 拓展配件 58
第 2 章 Python 基础知识 67
2.1 原始数据类型和运算符 67
2.2 变量和集合 72
2.3 流程控制和迭代器 78
2.4 函数 82
2.5 类 85
2.6 模块 87
2.7 高级用法 88

第二部分 基于 STM32 平台 90

第 3 章 开发环境快速建立 91
3.1 基于 Windows 91
3.1.1 安装 python3 91
3.1.2 安装开发软件 Mu 94
3.1.3 开发套件使用 97
3.1.4 附录 122
3.2 基于 Mac OS 131
3.2.1 安装 python3 131
3.2.2 安装开发软件 Mu 132
3.2.3 开发套件使用 133

3.3 基于 Linux 140
3.3.1 开发软件 Mu 安装 140
3.3.2 串口终端 minicom 140
第 4 章 基础实验 143
4.1 点亮第一个 LED 灯 144
4.2 流水灯 148
4.3 按键 155
4.4 GPIO 159
4.5 外部中断 163
4.6 I2C 总线（OLED 显示屏） 167
4.7 RTC 实时时钟 174
4.8 ADC 180
4.9 DAC 185
4.10 三轴加速度计 193
4.11 UART（串口通信） 198
第 5 章 传感器实验 205
5.1 温度传感器 DS18B20 206
5.2 温湿度传感器 DHT11 212
5.3 人体感应传感器 217
5.4 光敏传感器 223
5.5 土壤湿度传感器 228
5.6 水位传感器 234
5.7 大气压强传感器 241
5.8 超声波传感器 248
第 6 章 拓展实验 255
6.1 舵机 256
6.2 RGB 灯带 266
6.3 LCD 液晶显示屏 272
6.4 以太网模块 283
6.4.1 连接网络 286
6.4.2 Socket 通信 291
6.4.3 MQTT 通信 303

第三部分 基于 ESP8266 平台 316

第 7 章 开发环境快速建立 317
7.1 基于 Windows 317
7.1.1 安装 python3 317
7.1.2 安装开发软件 Mu 317
7.1.3 开发套件使用 320
7.2 基于 Mac OS 333
7.2.1 安装 python3 333
7.2.2 安装开发软件 Mu 335
7.2.3 开发套件使用 338

7.3 基于 Linux 343
第 8 章 基础实验 344
8.1 点亮第一个 LED 345
8.2 按键 350
8.3 外部中断 355
8.4 定时器 360
8.5 I2C 总线（OLED 显示屏） 364
8.6 RTC 实时时钟 371
8.7 ADC 377
8.8 PWM 382
第 9 章 传感器实验 388
9.1 温度传感器 DS18B20 389
9.2 温湿度传感器 DHT11 397
第 10 章 WiFi 应用 403
10.1 连接无线路由器 404
10.2 Socket 通信 410
10.3 MQTT 通信 422
10.4 WebREPL 437

第四部分 基于 ESP32 平台 441

第 11 章 开发环境快速建立 442
11.1 基于 Windows 442
11.1.1 安装 python3 442
11.1.2 安装开发软件 Mu 442
11.1.3 开发套件使用 442
11.2 基于 Mac OS 455
11.1 基于 Linux 455
第 12 章 基础实验 456
12.1 点亮第一个 LED 457
12.2 按键 462
12.3 外部中断 467
12.4 定时器 472
12.5 I2C 总线（OLED 显示屏） 476
12.6 RTC 实时时钟 483
12.7 ADC 489
12.8 PWM 494
第 13 章 传感器实验 500
13.1 温度传感器 DS18B20 501
13.2 温湿度传感器 DHT11 509
13.3 人体感应传感器 515
13.4 光敏传感器 521
13.5 土壤湿度传感器 527
13.6 水位传感器 533

13.7 大气压强传感器 540
13.8 超声波传感器 547
第 14 章 拓展实验 554
14.1 舵机 555
14.2 RGB 灯带 561
第 15 章 WiFi 应用 566
15.1 连接无线路由器 567
15.2 Socket 通信 573
15.3 MQTT 通信 585
15.4 WebREPL 600
第 16 章 LVGL（LittleVGL） 601
16.1 LVGL（LittleVGL）简介 602
16.2 LCD 和触摸屏校准 603
16.3 小部件实验 611
16.3.1 Basc object（基础对象） 612
16.3.2 Bar（进度条） 616
16.3.3 Button（按钮） 620
16.3.4 Button Matrix（矩阵键盘） 624
16.3.5 Calendar（日历） 629
16.3.6 CheckBox（复选框） 633
16.3.7 Chart（图表） 636
16.3.8 Container（容器） 640
16.3.9 Drawdown List（下拉列表） 644
16.3.10 Gauge（计量器） 648
16.3.11 Keyboard（键盘） 652
16.3.12 Label（标签） 656
16.3.13 Line（线） 660
16.3.14 List（菜单） 664
16.3.15 Message box（消息框） 668
16.3.16 Page（页） 672
16.3.17 Roller（滚筒） 676
16.3.18 Slider（滑动条） 680
16.3.19 Spinner（旋转加载条） 684
16.3.20 Swtich（开关） 688
16.3.21 Table（表格） 693
16.3.22 Tabview（选项卡视图） 698
16.3.23 Textarea（文本区域） 702
16.3.24 Window（窗口） 705
16.4 项目应用 709
16.4.1 网络时钟 709

第五部分 基于 NRF52840 平台 718

第 17 章 开发环境快速建立 719

17.1 基于 Windows 719
17.1.1 安装 python3 719
17.1.2 安装开发软件 Mu 722
17.1.3 开发套件使用 725
17.2 基于 Mac OS 745
17.2.1 安装 python3 745
17.2.2 安装开发软件 Mu 746
17.2.3 开发套件使用 746
17.3 基于 Linux 747
17.3.1 开发软件 Mu 安装 747
第 18 章 基础实验 748
18.1 点亮第一个 LED 749
18.2 按键 756
18.3 ADC 761
18.4 PWM 767
18.5 I2C 总线（OLED 显示屏） 773
18.6 UART（串口通讯） 781
第 19 章 传感器实验 788
19.1 温度传感器 DS18B20 789
19.2 温湿度传感器 DHT11 797
19.3 人体感应传感器 803
19.4 光敏传感器 809
19.5 土壤湿度传感器 815
19.6 水位传感器 821
19.7 大气压强传感器 828
19.8 超声波传感器 835
第 20 章 拓展实验 842
20.1 舵机 843
20.2 RGB 彩灯（Neopixel） 849
20.3 LCD 856
20.3.1 1.77 寸 LCD 856
第 21 章 BLE 应用 865
21.1 蓝牙广播 866
21.2 设备信息 871
21.3 无线数据收发 876
21.4 彩灯控制 889

第六部分 基于 OpenMV4 平台 895

第 22 章 开发环境快速建立 896
22.1 基于 Windows 896
22.1.1 安装 python3 896
22.1.2 安装开发软件 OpenMV IDE 896
22.1.3 开发套件使用 898

22.2 基于 Mac OS 913
22.2.1 安装 python3 913
22.2.2 安装开发软件 OpenMV IDE 913
22.3 基于 Linux（树莓派） 914
第 23 章 基础实验 915
23.1 点亮第一个 LED 916
23.2 流水灯 921
23.3 GPIO（按键） 926
23.4 外部中断 931
23.5 定时器 936
23.6 I2C 总线（OLED 显示屏） 939
23.7 RTC 实时时钟 946
23.8 ADC 952
23.9 DAC 958
第 24 章 机器视觉 966
24.1 摄像头应用 966
24.2 画图 972
24.3 特征检测 978
24.3.1 边缘检测 979
24.3.2 圆形识别 983
24.3.3 线段识别 987
24.3.4 直线识别 991
24.3.5 矩形识别 995
24.3.6 特征点识别 999
24.3.7 快速线性回归（巡线） 1006
24.4 颜色追踪 1012
24.4.1 RGB565 彩色自动追踪 1013
24.4.2 机器人巡线（实线巡线） 1020
24.4.3 单个颜色识别 1028
24.5 人脸检测 1034
24.5.1 人脸检测 1035
24.5.2 人脸追踪 1041
24.6 眼球追踪 1050
24.6.1 眼球检测 1051
24.6.2 瞳孔追踪 1056
24.7 图片拍摄 1062
24.7.1 普通拍摄 1063
24.7.2 人脸检测拍摄 1065
24.7.3 移动物体抓拍 1070
24.8 视频录制 1076
24.8.1 普通 GIF 视频录制 1077
24.8.2 人脸检测 GIF 视频录制 1082
24.8.3 移动物体 GIF 视频录制 1087
24.9 图像滤波 1093

24.9.1 颜色二值化滤波 1094
24.9.2 锐化滤波 1099
24.9.3 图像翻转 1103
第 25 章 机器学习 1108
25.1 物体识别 1109
25.2 笑脸识别 1118
25.3 数字识别 1124
第 26 章 拓展模块 1129
26.1 LCD 显示屏 1130
26.2 舵机 1134
26.3 多路舵机模块 1144
26.4 WiFi 模块 1149
26.4.1 连接无线路由器 1150
26.4.2 Socket 通信 1154
26.4.3 MQTT 通信 1165
26.4.4 无线视频传输 1178
26.5 摄像头模块 1181
26.5.1 MT9V034 全局快门 1182
26.5.2 FLIR Lepton 红外热成像 1188
26.6 摄像头镜头 1193
26.6.1 广角镜头 1194
26.6.2 长焦镜头 1196
26.6.3 无畸变镜头 1198
26.6.4 手动调焦镜头 1200
第 27 章 项目应用 1202
27.1 照相机 1202

第七部分 基于 K210 平台 1208

第 28 章 开发环境快速建立 1209
28.1 基于 Windows 1209
28.1.1 安装 python3 1209
28.1.2 安装开发软件 MaixPy IDE 1209
28.1.3 开发套件使用 1211
28.2 基于 Mac OS 1231
28.2.1 安装 python3 1231
28.2.2 安装开发软件 MaixPy IDE 1231
28.3 基于 Linux 1232
28.3.1 安装 MaixPy IDE 1232
第 29 章 基础实验 1233
29.1 点亮第一个 LED 1234
29.2 流水灯 1240
29.3 按键 1246
29.4 外部中断 1251

29.5 定时器 1257
29.6 PWM 1262
29.7 I2C 总线（OLED 显示屏） 1269
29.8 UART（串口通信） 1276
29.9 thread（线程） 1283
第 30 章 机器视觉 1286
30.1 LCD 1286
30.2 摄像头应用 1295
30.3 画图 1303
30.4 颜色识别 1309
30.5 二维码识别 1314
30.6 人脸检测 1319
30.7 物体识别 1327
30.8 图片拍摄 1333
30.9 视频录制 1338
第 31 章 机器听觉 1344
31.1 声音频率识别（FFT） 1344
31.2 声源定位 1352
第 32 章 拓展模块 1357
32.1 电阻触摸屏 1357
32.2 舵机 1363
32.3 RGB 灯带 1370
32.4 WiFi 模块 1376
32.4.1 连接无线路由器 1377
32.4.2 Socket 通信 1382
32.4.3 MQTT 通信 1395
32.5 摄像头镜头 1410
32.5.1 广角镜头 1411
32.5.2 长焦镜头 1413
32.5.3 无畸变镜头 1415
32.5.4 手动调焦镜头 1417
第 33 章 项目应用 1419
33.1 照相机 1419
33.2 视频播放器 1425
33.3 NES 游戏机 1431

第一部分　MicroPython 基础知识

第1章　MicroPython 简介

1.1　MicroPython 是什么

第一次接触 MicroPyhon 的时候，我就想这是个什么玩意，从字面意思来看，就是 Micro 加 Python。难道是阉割版的 Python？阉割后可以在微控制器上面跑？当然你也可以这么理解，我们来看看官方的说明：

“MicroPython 是 Python 3 编程语言的精简高效实现，包括 Python 标准库的一小部分，并且经过优化，可以在 Microcontrollers（微控制器）和有限的环境中运行。

MicroPython 包含许多高级功能，如交互式提示，任意精度整数，闭包，列表理解，生成器，异常处理等。然而它非常紧凑，可以在 256k 的代码空间和 16k 的 RAM 内运行。

MicroPyhon 旨在尽可能与普通 Python 兼容，以便您轻松地将代码从电脑传输到微控制器或者嵌入式系统。”

看完官方说明后，大家应该有所了解，Micropyton 是指在微控制器上使用 Python 语言进行编程，学习过单片机和嵌入式开发的小伙伴应该都知道早期的单片机使用汇编语言来编程，随着微处理器的发展，后来逐步被 C 所取代，现在的微处理器集成度越来越高了，那么我们现在可以使用 Python 语言来开发了。

Python 的强大之处是封装了大量的库，开发者直接调用库函数则可以高效地完成大量复杂的开发工作。MicroPython 保留了这一特性，常用功能都封装到库中了，以及一些常用的传感器和组件都编写了专门的驱动，通过调用相关函数，就可以直接控制 LED、按键、伺服电机、PWM、AD/DA、UART、SPI、IIC 以及 DS18B20 温度传感器等等。以往需要花费数天编写才能实现的硬件功能代码，现在基于 MicroPython 开发只要十几分钟甚至几行代码就可以解决。真可谓：“人生苦短，我用 Python 和 MicroPython”。

1.2 MicroPython 支持的微控制器平台

MicroPython 到目前为止已经可以在多种嵌入式硬件平台上运行：STM32、ESP8266、ESP32、CC3200、K210 等等。由于项目的开源特性，很多开发者在尝试将其移植到更多平台上。

MicroPython 最早支持的硬件平台是 STM32，开发板名称叫 pyboard。使用的芯片型号是：STM32F405RGT6，该芯片具备 1MB flash 和 196k SRAM，168MHZ 主频。也是目前学习资料最全的平台，本书主要是围绕 SMT32 平台来进行编写。

除此之外，上海乐鑫的 WIFI 芯片 ESP8266/ESP32 也非常成熟。用户使用 MicroPython 可以快速开发物联网相关应用，实现 WIFI 无线连接。

除此之外，不少优秀的开源项目也是基于 MicroPythno 衍生出来的，如机器视觉界 Arduino 之称的 OpenMV、人工智能芯片 K210 等。随着社区的日益成熟，MicroPython 必定将嵌入式编程推向新的高度。

目前 01Studio 开发板产品主要包含五大平台，开发起来有所区别，用户使用本教程课根据自己的开发板来选择性阅读。也就是**第一部分 MicroPython 基础知识 + 第 X 部分 自己的开发板平台。**

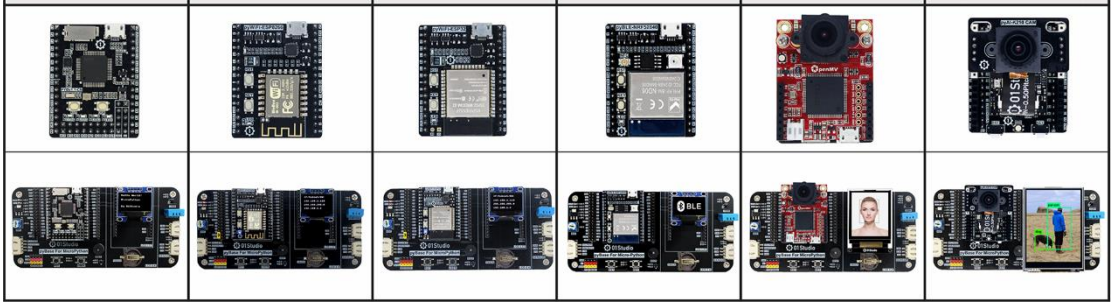
STM32 平台	ESP8266 平台	ESP32 平台	NFR52840 平台	OpenMV4 平台	K210 平台
pyBoard v1.1-CN	pyWiFi-ESP8266	pyWIFI-ESP32	pyBLE-NRF52840	pyAI-OpenMV4	pyAI-K210
					

图 1-1 01Studio MicroPython 系列开发平台

1.3 MicroPython 相关学习资料

1.3.1 01Studio 社区

【社区网址：www.01Studio.org】

01Studio 社区是 MicroPython 开发者交流的社区论坛，我们以极简风格设计，开发者在学习过程中遇到问题可以到论坛搜索或者发帖提问，以提高学习效率。01Studio 团队也会在社区定期发布学习资源。



图 1-2 01Studio 社区

1.3.2 MicroPython 文档（中文）

限于篇幅，本教程部分实验只介绍关键的函数和模块应用，如果在学习过程中希望深入了解 MicroPython 函数和模块，请查阅:

MicroPython 文档（中文） 【网址：docs.micropython.01studio.org】

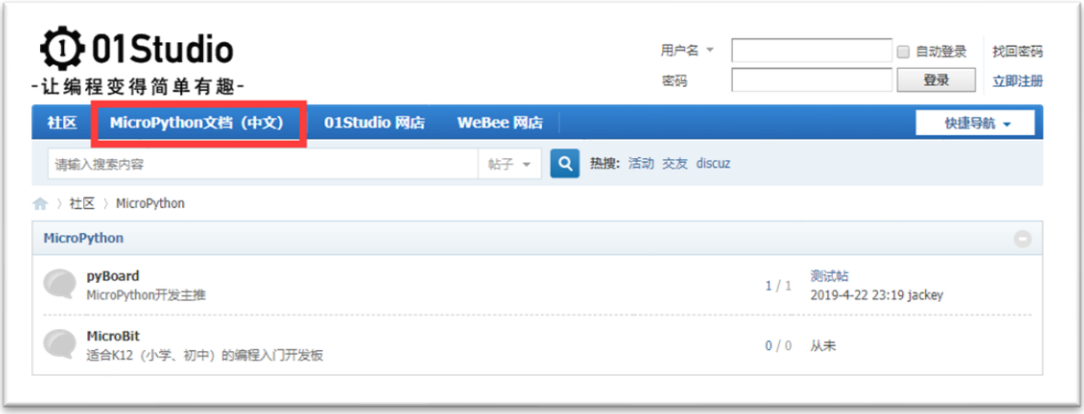


图 1-3 文档在社区的位置

该文档是 01Studio 团队在维护的官方文档中文翻译版，我们力求保持与官方文档保持实时同步，降低开发者的学习门槛。



图 1-4 MciroPython 文档（中文）

1.3.3 MicroPython 官方网站

【网址：www.micropython.org】

英文版官网有官方文档（DOCS）和英文论坛，适合比较英语比较好的小伙伴。

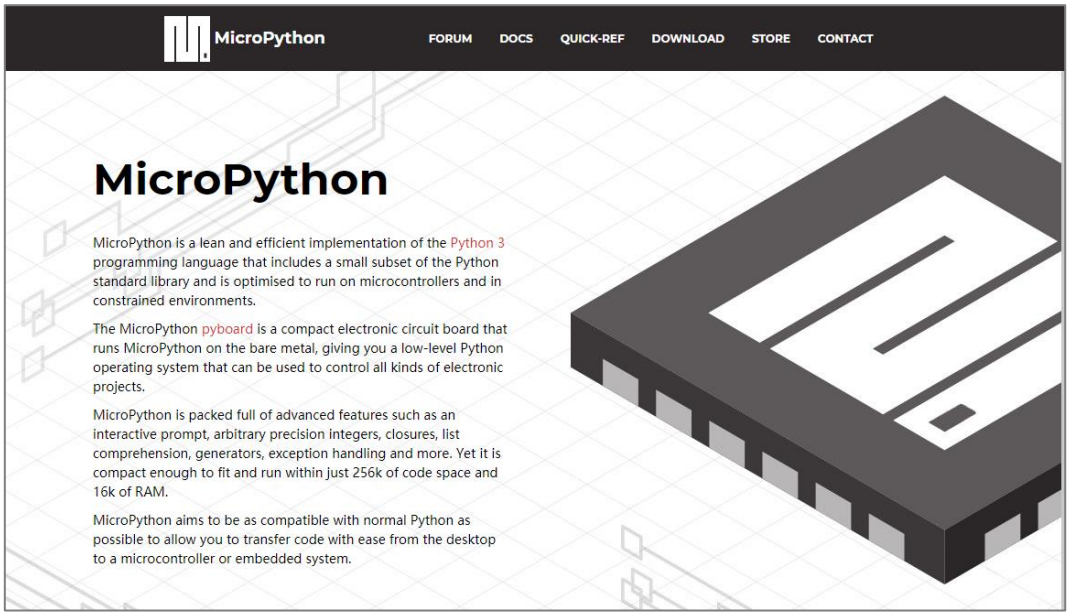


图 1-5 MicroPython 官网

1.4 MicroPython 开发套件介绍

为了让广大电子爱好者更方便地学习 MicroPython，01Studio 团队打造了一系列本土化的高性价比学习套件和周边模块。当前已支持 STM32 平台、ESP8266 平台、ESP32 平台、NRF52840 平台、OpenMV 平台、K210 平台以及周边传感器模块和配件外设。我们的开发平台采用核心板+底板形式设计，保留了 MicroPyhon 官方的兼容性，同时使开发者可以更好的连接外设，进行更多扩展性实验。

《MicroPython 从 0 到 1》上的例程也是基于本学习平台开发的，我们承诺资源会不断更新，保证所有代码程序能直接跑起。毫不夸张地说：你甚至可以将本教材的例程和实践应用在自己的产品研发和项目开发中去。

1.4.1 STM32 平台

1.4.1.1 pyBoard v1.1-CN

pyBaord v1.1-CN 主控芯片使用 STM32F405RGT6，即基于 STM32 平台，是 01Studio 在兼容官方 pyBoard v1.1 的情况下，对部分功能进行了改进，CN 代表 China 中国版，具体改进如下：

（1） 按键和 LED 重新排列，让开发者使用更直观；

（2） 增加锂电池输入接口（XH-2.54 2P 接口），位于板子背部；

（3） 改进丝印方式，提高了清晰度。

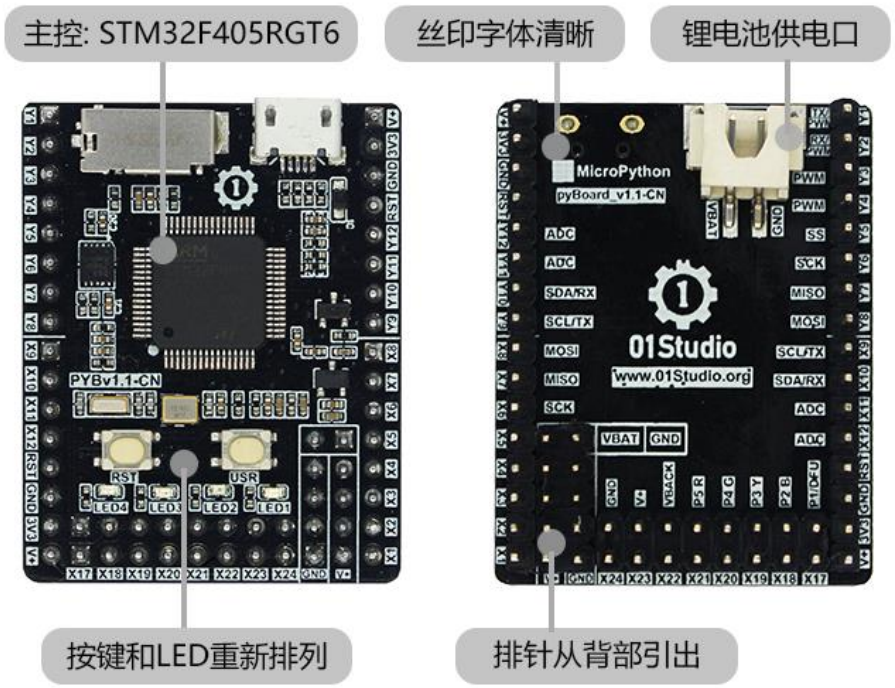


图 1-6 pyBoard v1.1-CN

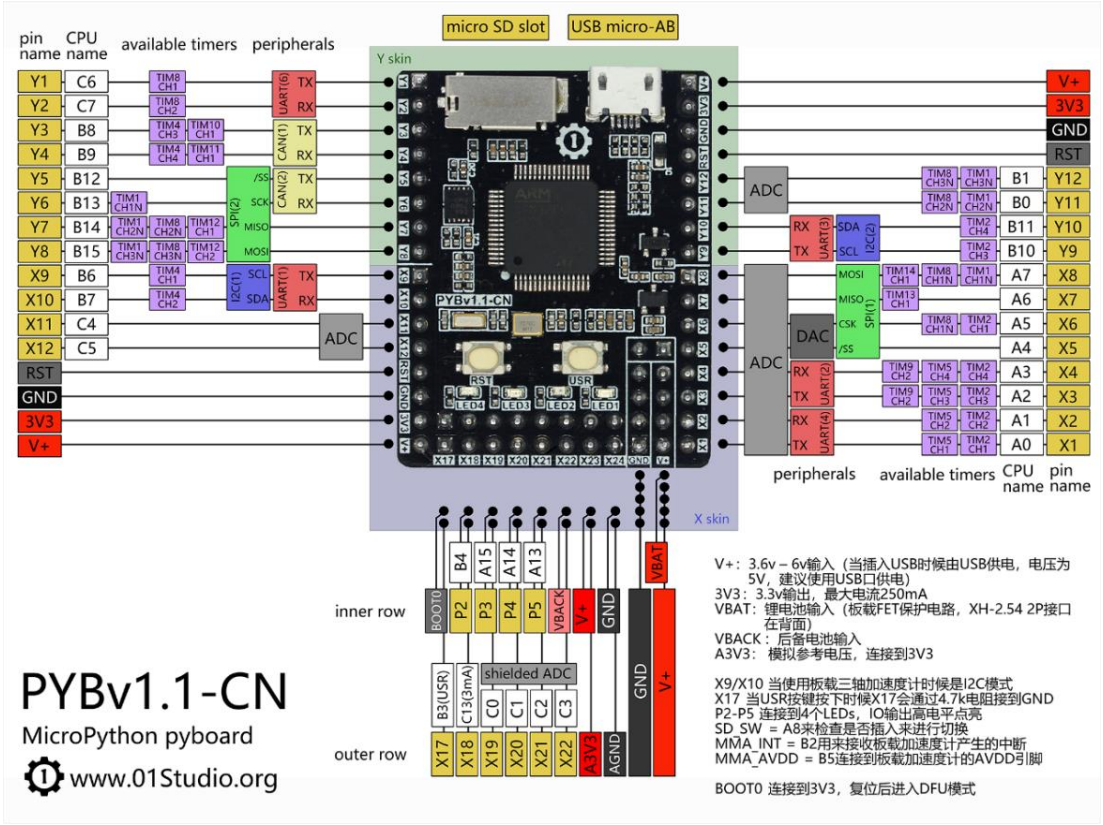


图 1-7 pyBoard v1.1-CN 引脚图

功能参数	
V+	3.6v – 6v 输入（当插入 USB 时候由 USB 供电，电压为 5V，建议使用 USB 口供电）
3V3	3.3v 输出，最大电流 250mA
VBAT	锂电池输入（板载 FET 保护电路，XH-2.54 2P 接口在背面）
VBACK	后备电池输入
A3V3	模拟参考电压，连接到 3V3
X9/X10	当使用板载三轴加速度计时候是 I2C 模式
X17	当 USR 按键按下时候 X17 会通过 4.7k 电阻接到 GND
P2-P5	连接到 4 个 LEDs，IO 输出高电平点亮
SD_SW	A8 来检查是否插入 SD 卡来进行切换
MMA_INT	B2 用来接收板载加速度计产生的中断

MMA_AVDD	B5 连接到板载加速度计的 AVDD 引脚
BOOT0	连接到 3V3，复位后进入 DFU 模式

表 1-1　pyBoard v1.1-CN 参数

1.4.1.2　pyBoard Mini

pyBaord Mini 主控芯片使用 STM32F411CEU6，即基于 STM32 平台，是 pyBoard 的 Mini 版本，精简了部分功能，提高了性价比：

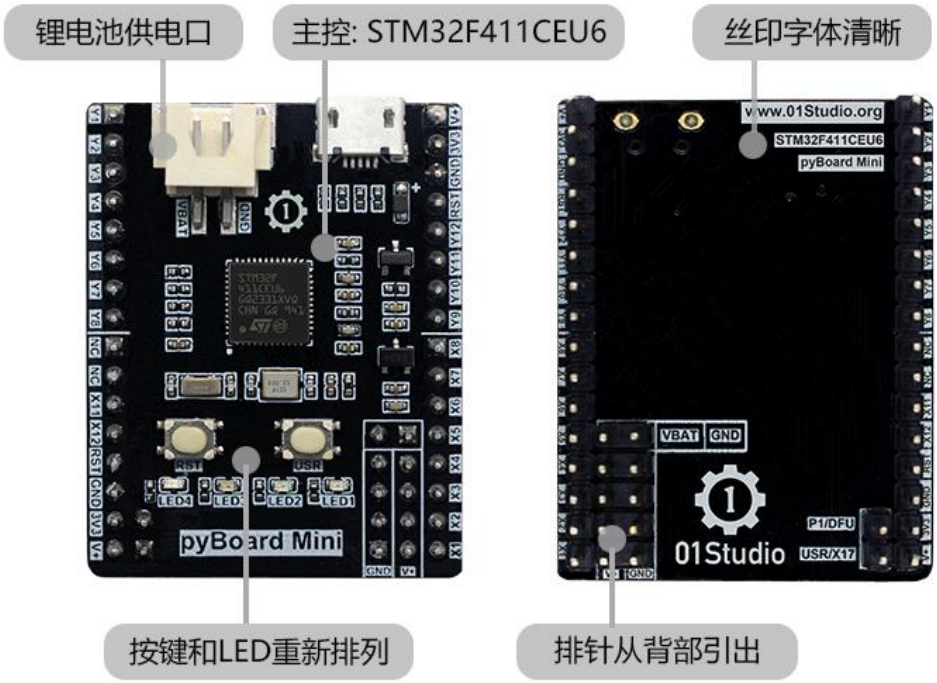


图 1-8 pyBoard Mini

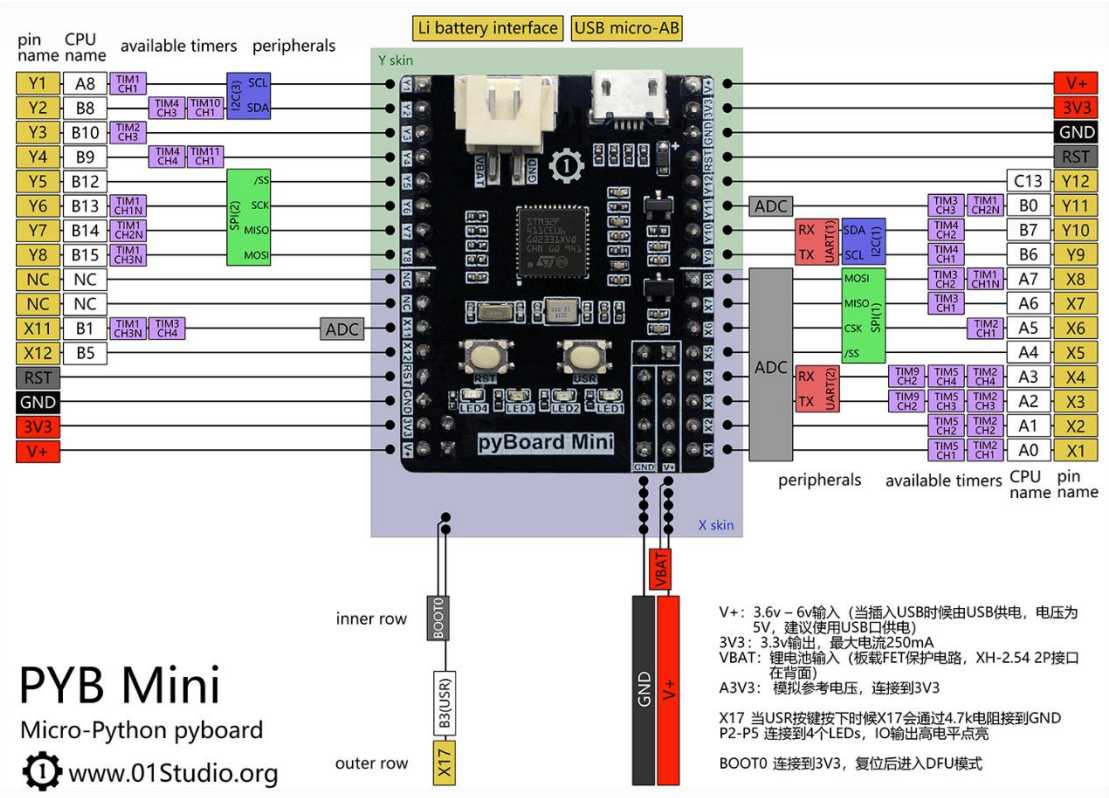


图 1-9 pyBoard Mini 引脚图

功能参数	
V+	3.6v – 6v 输入（当插入 USB 时候由 USB 供电，电压为 5V，建议使用 USB 口供电）
3V3	3.3v 输出，最大电流 250mA
VBAT	锂电池输入（板载 FET 保护电路，XH-2.54 2P 接口在背面）
A3V3	模拟参考电压，连接到 3V3
X17	当 USR 按键按下时候 X17 会通过 4.7k 电阻接到 GND
P2-P5	连接到 4 个 LEDs，IO 输出高电平点亮
BOOT0	连接到 3V3，复位后进入 DFU 模式

表 1-2 pyBoard Mini 功能参数

1.4.1.3 pyBoard Plus

pyBaord Plus 主控芯片使用 STM32F407VGT6，即基于 STM32 平台，是 pyBoard 的 Plus 加强版本，相对于 pyboard 增加了 32 个通用 GPIO 和拓展功能：

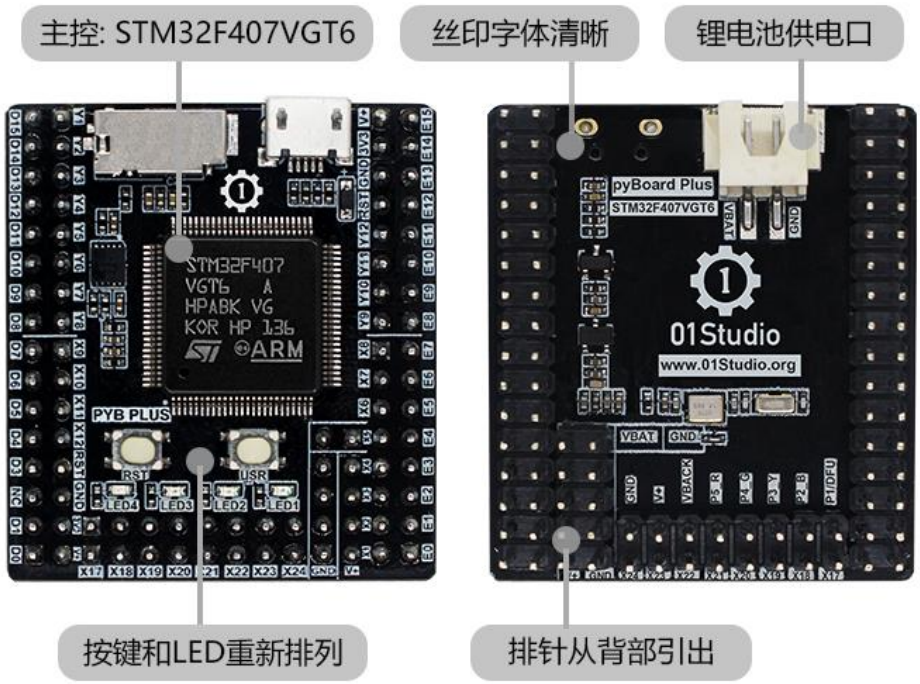


图 1-10 pyBoard Plus

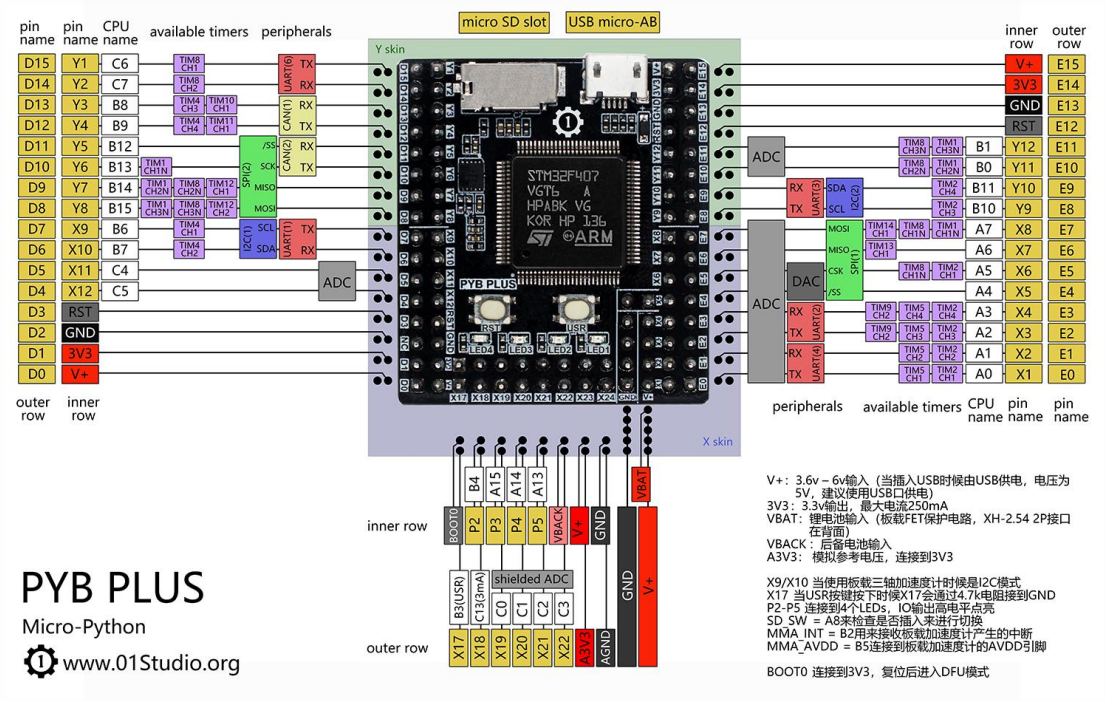


图 1-11 pyBoard Plus 引脚图

功能参数	
V+	3.6v – 6v 输入（当插入 USB 时候由 USB 供电，电压为 5V，建议使用 USB 口供电）
3V3	3.3v 输出，最大电流 250mA
VBAT	锂电池输入（板载 FET 保护电路，XH-2.54 2P 接口在背面）
A3V3	模拟参考电压，连接到 3V3
X9/X10	当使用板载三轴加速度计时候是 I2C 模式
X17	当 USR 按键按下时候 X17 会通过 4.7k 电阻接到 GND
P2-P5	连接到 4 个 LEDs，IO 输出高电平点亮
SD_SW	A8 来检查是否插入 SD 卡来进行切换
MMA_AVDD	B5 连接到板载加速度计的 AVDD 引脚
BOOT0	连接到 3V3，复位后进入 DFU 模式

表 1-3 pyBoard Plus 功能参数

1.4.1.4 pyBoard 开发套件

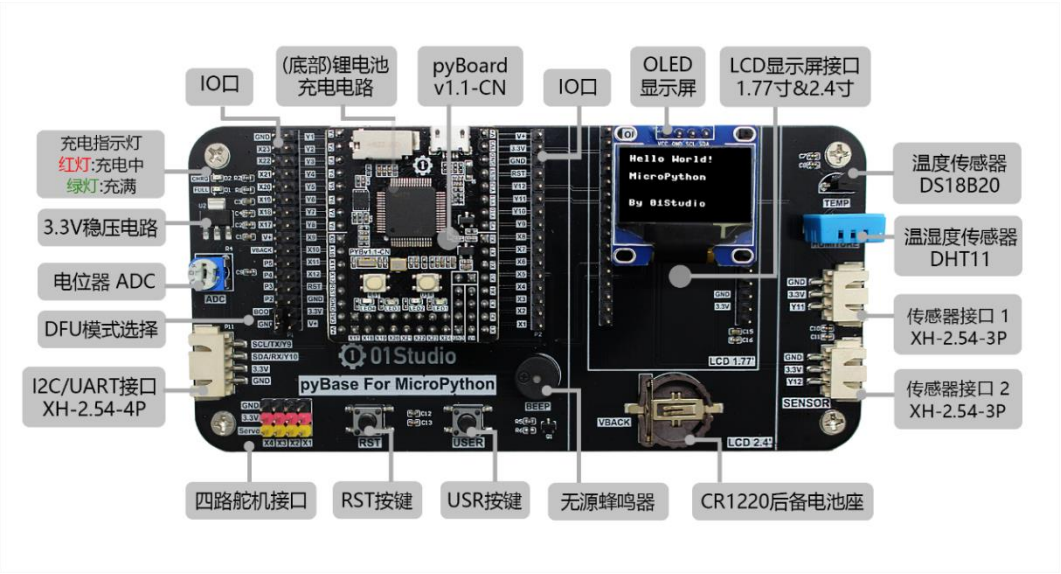


图 1-12 pyBoard 开发套件

主要配置	
核心板	pyBoard v1.1-CN
底板	pyBase
显示屏	0.9 寸 OLED 显示屏

表 1-4

1.4.2 ESP8266 平台

1.4.2.1 pyWiFi-ESP8266

pyWiFi-ESP8266 是由 01Studio 设计研发，基于 ESP8266 平台，接口兼容 MicroPython 的 pyBoard，主要特点如下：

- ✧ 兼容 pyBoard 接口
- ✧ 自动下载电路
- ✧ 板载锂电池输入接口和充电电路
- ✧ 按键和 LED 排列整齐，丝印清晰
- ✧ 全 IO 引出

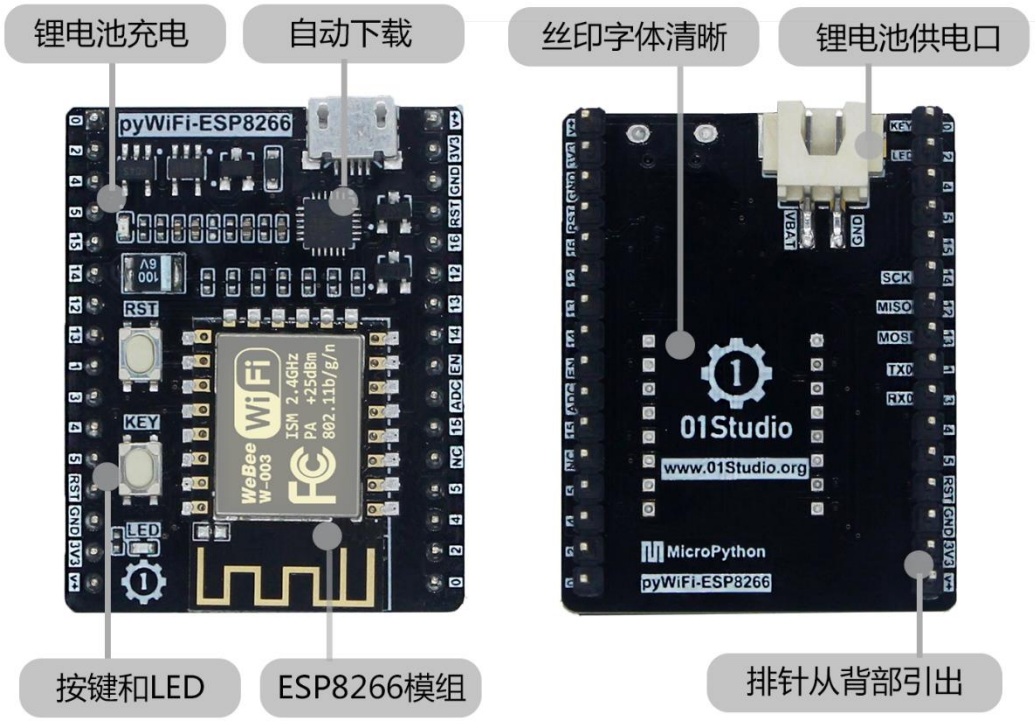


图 1-13 pyWiFi-ESP8266

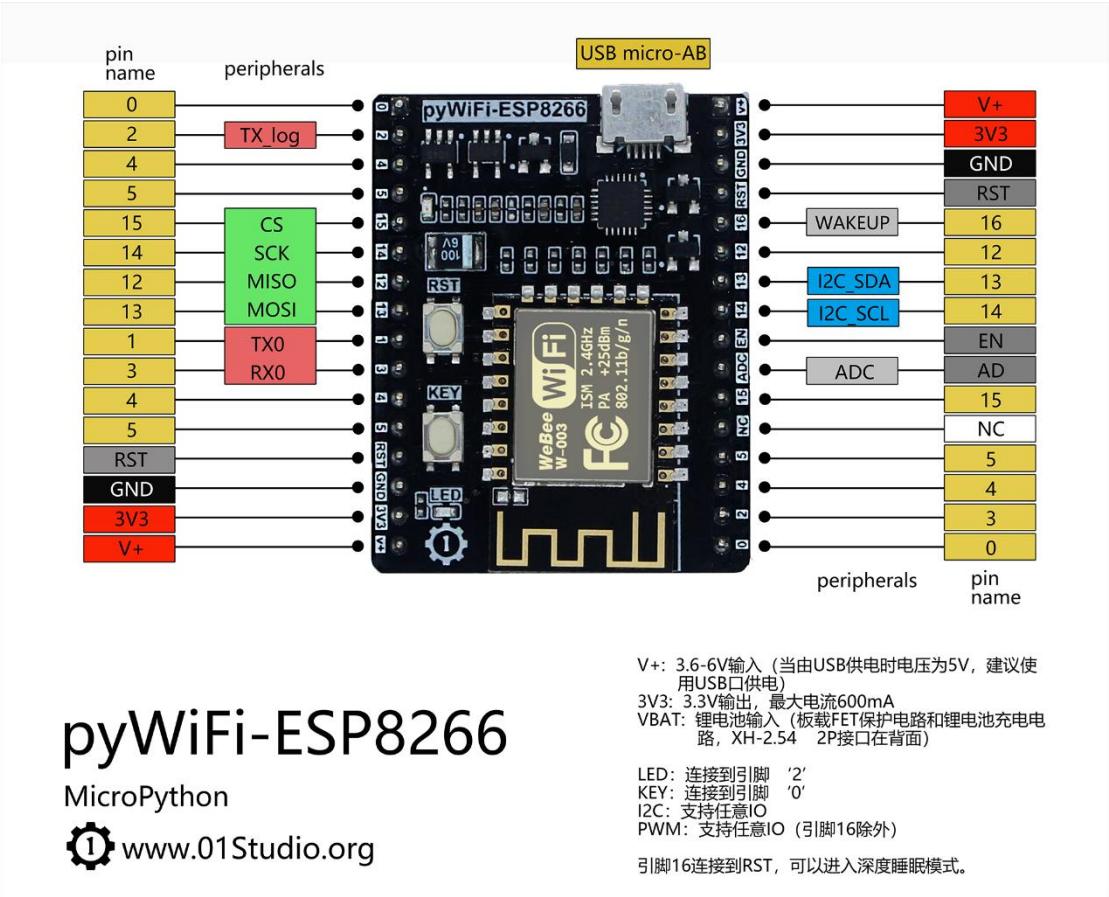


图 1-14 pyWiFi-ESP8266 引脚图

功能参数	
V+	3.6v – 6v 输入（当插入 USB 时候由 USB 供电，电压为 5V，建议使用 USB 口供电）
3V3	3.3v 输出，最大电流 600mA
VBAT	锂电池输入（板载 FET 保护电路和锂电池充电电路，XH-2.54 2P 接口在背面）
LED	连接到引脚 2
KEY	连接到引脚 0
I2C	支持任意 IO
PWM	支持任意 IO（引脚 16 除外）
SLEEP	引脚连接到 RST，可以进入深度睡眠唤醒模式

表 1-5 pyWiFi-ESP8266 功能参数表

1.4.2.2 pyWiFi-ESP8266 开发套件

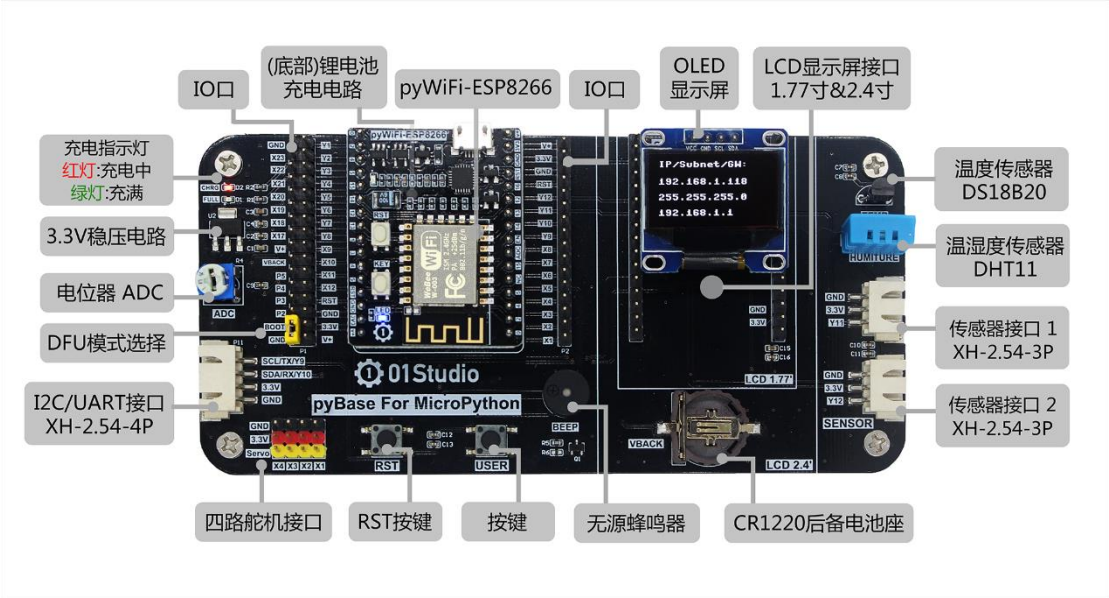


图 1-15 pyWiFi-ESP8266 开发套件

主要配置	
核心板	pyWiFi-ESP8266
底板	pyBase
显示屏	0.9 寸 OLED 显示屏

表 1-6

1.4.3 ESP32 平台

1.4.3.1 pyWiFi-ESP32

pyWiFi-ESP32 是由 01Studio 设计研发，基于 ESP32 平台，接口兼容 MicroPython 的 pyBoard，主要特点如下：

- ✧ 兼容 pyBoard 接口
- ✧ 自动下载电路
- ✧ 板载锂电池输入接口和充电电路
- ✧ 按键和 LED 排列整齐，丝印清晰
- ✧ 全 IO 引出

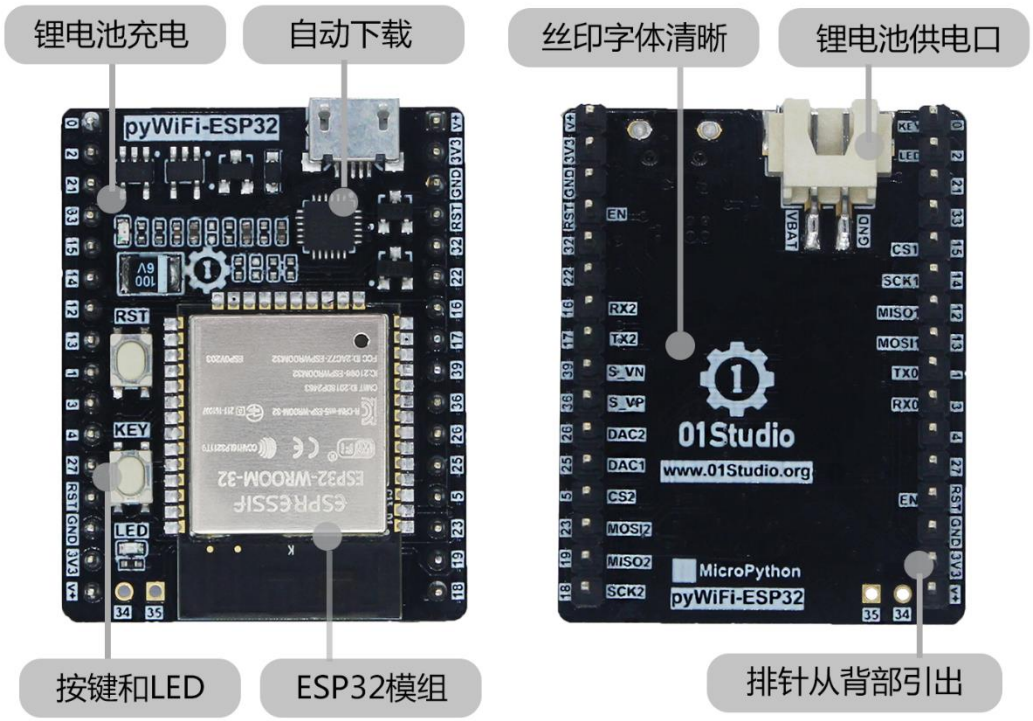


图 1-16 pyWiFi-ESP32

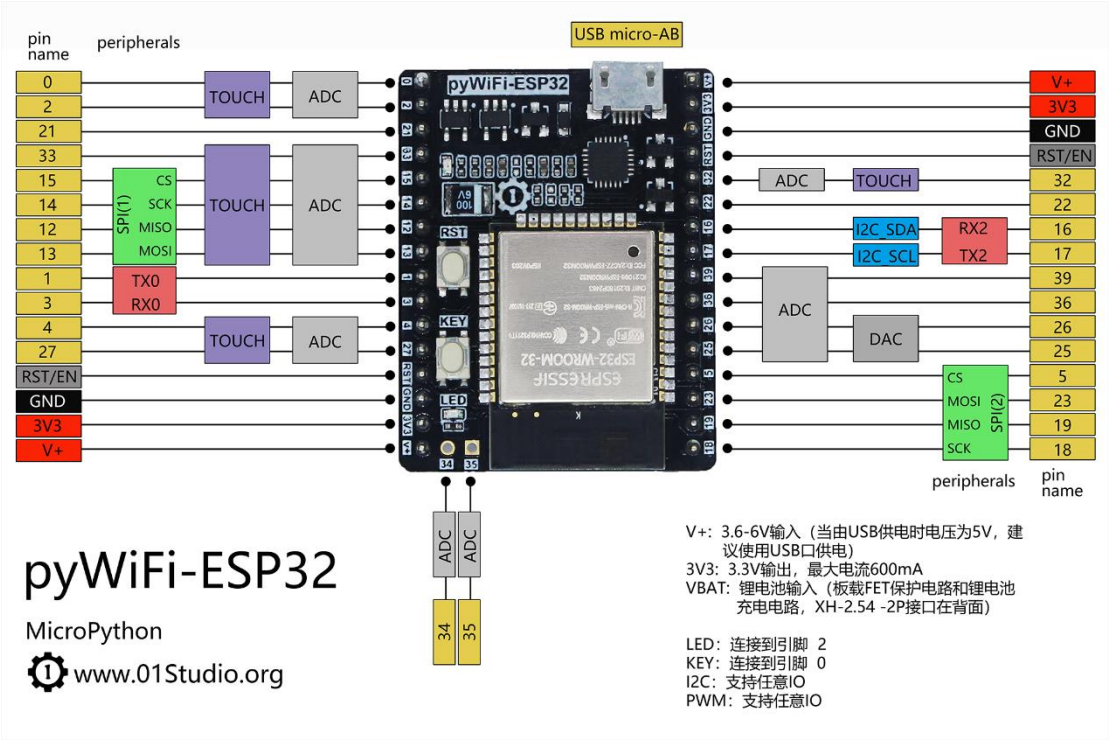


图 1-17 pyWiFi-ESP32 引脚图

功能参数	
V+	3.6v – 6v 输入（当插入 USB 时候由 USB 供电，电压为 5V，建议使用 USB 口供电）
3V3	3.3v 输出，最大电流 600mA
VBAT	锂电池输入（板载 FET 保护电路和锂电池充电电路，XH-2.54 2P 接口在背面）
LED	连接到引脚 2
KEY	连接到引脚 0
I2C	支持任意 IO
PWM	支持任意 IO

表 1-7　pyWiFi-ESP32 功能参数表

1.4.3.2 pyWiFi-ESP32P

pyWiFi-ESP32P 是由 01Studio 设计研发，基于 ESP32 平台，接口兼容 MicroPython 的 pyBoard，**相对于 pyWiFi-ESP32 内置了 8MB PSRAM，GPIO 16,17 被占用。**主要特点如下：

- ✧ 兼容 pyBoard 接口
- ✧ 自动下载电路
- ✧ 板载锂电池输入接口和充电电路
- ✧ 按键和 LED 排列整齐，丝印清晰
- ✧ 全 IO 引出

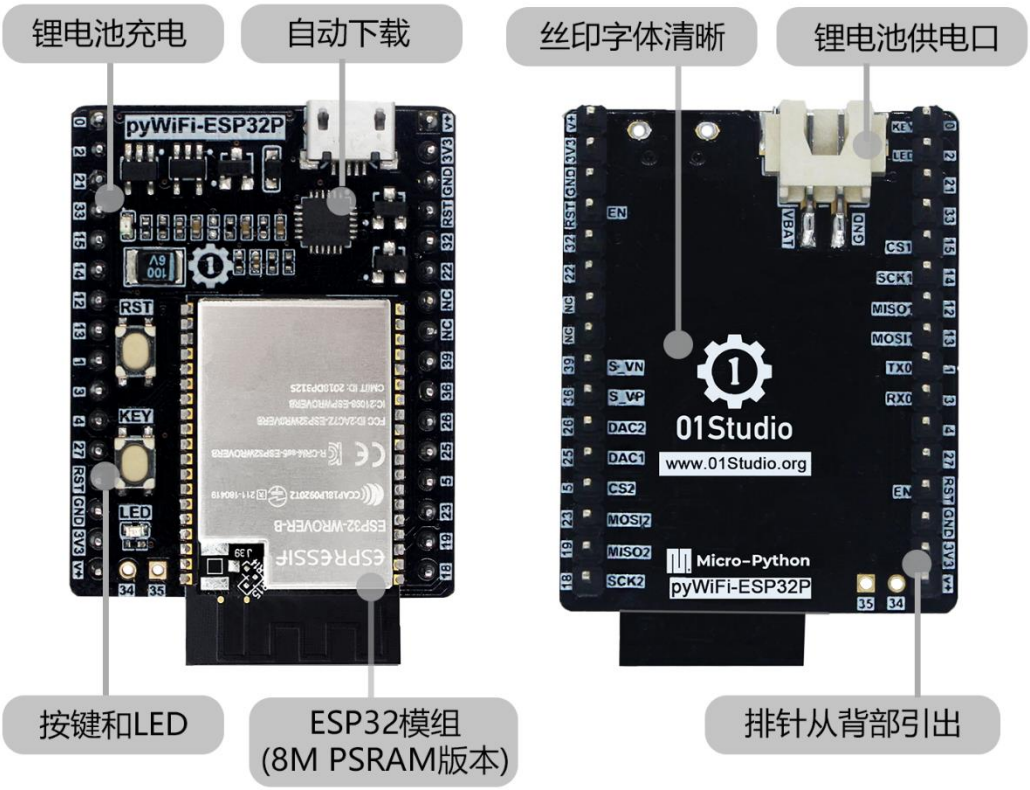


图 1-18 pyWiFi-ESP32P

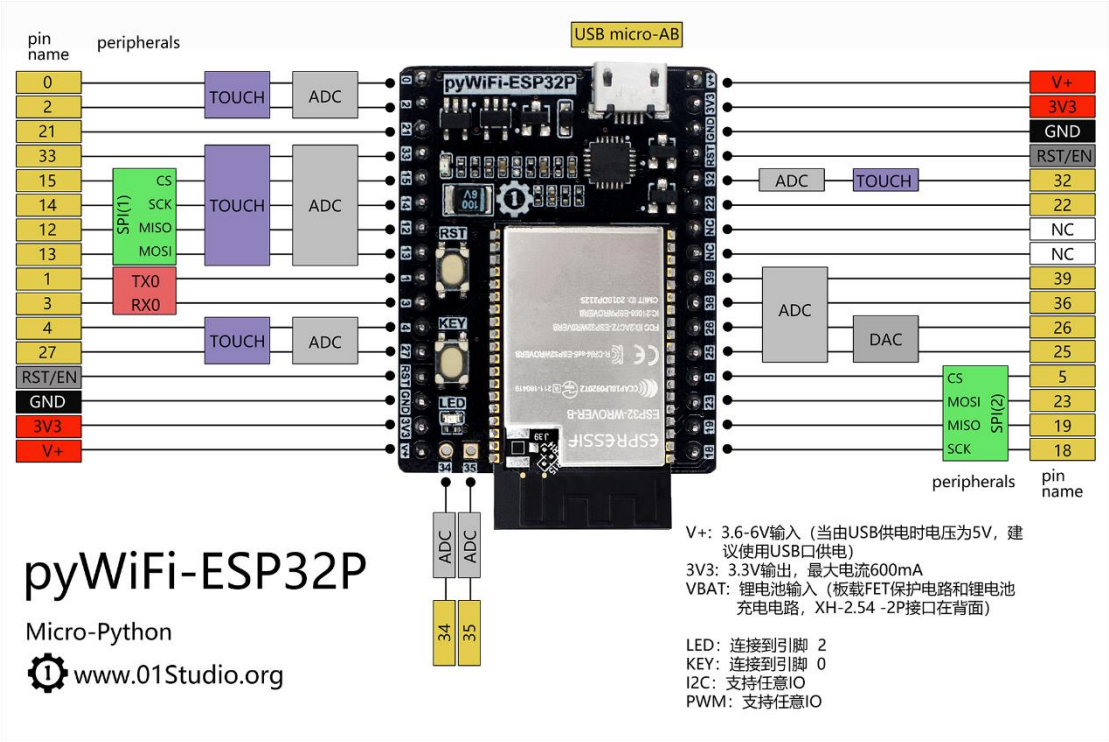


图 1-19 pyWiFi-ESP32P 引脚图

功能参数	
V+	3.6v – 6v 输入（当插入 USB 时候由 USB 供电，电压为 5V，建议使用 USB 口供电）
3V3	3.3v 输出，最大电流 600mA
VBAT	锂电池输入（板载 FET 保护电路和锂电池充电电路，XH-2.54 2P 接口在背面）
LED	连接到引脚 2
KEY	连接到引脚 0
I2C	支持任意 IO
PWM	支持任意 IO
拓展内存	8MB PSRAM

表 1-8 pyWiFi-ESP32P 功能参数表

1.4.3.3 pyWiFi-ESP32 开发套件

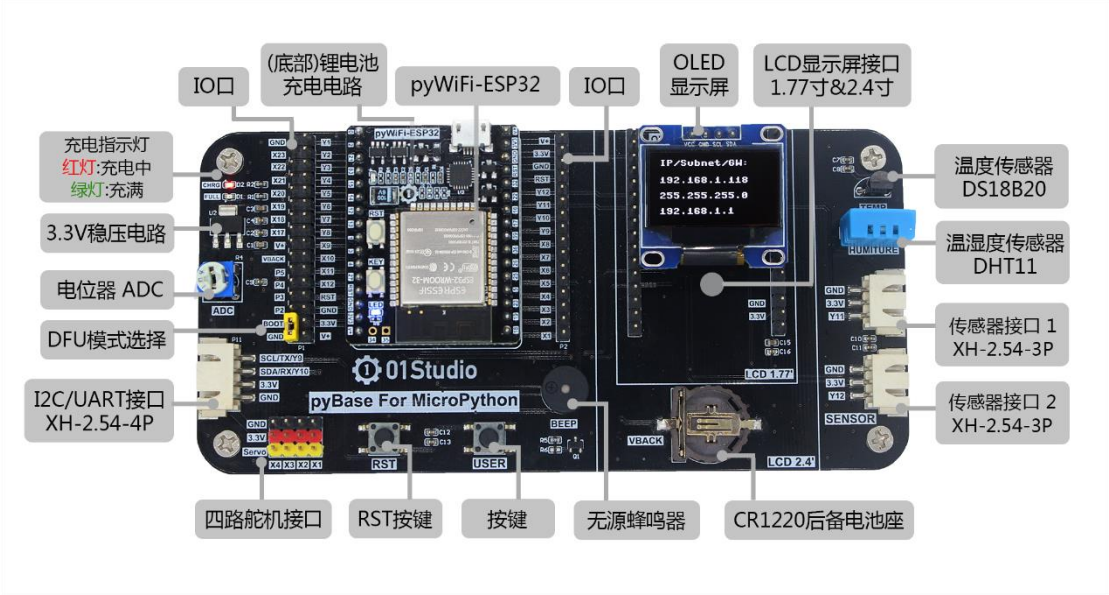


图 1-20 pyWiFi-ESP32 开发套件

主要配置	
核心板	pyWiFi-ESP32
底板	pyBase
显示屏	0.9 寸 OLED 显示屏

表 1-9

1.4.4 NRF52840 平台

1.4.4.1 pyBLE-NRF52840

pyBLE-NRF52840 是由 01Studio 设计研发，基于 Nordic 的 NRF52840 平台，接口兼容 MicroPython 的 pyBoard，主要特点如下：

- ✧ 兼容 pyBoard 接口
- ✧ 板载锂电池输入接口和充电电路
- ✧ 按键和 LED 排列整齐，丝印清晰
- ✧ 全 IO 引出

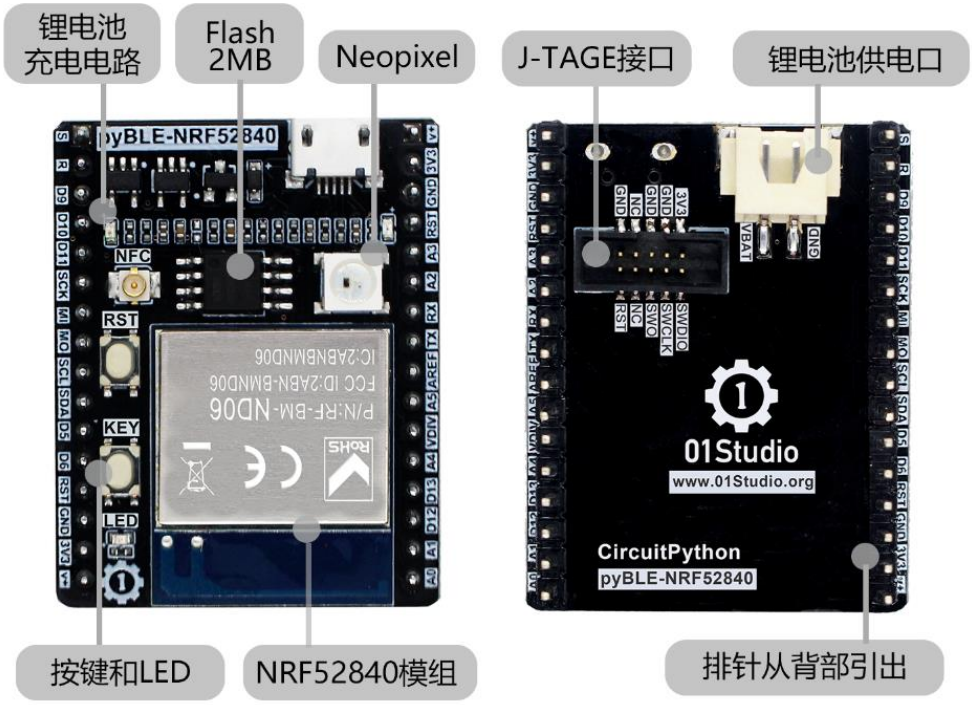


图 1-21 pyBLE-NRF52840

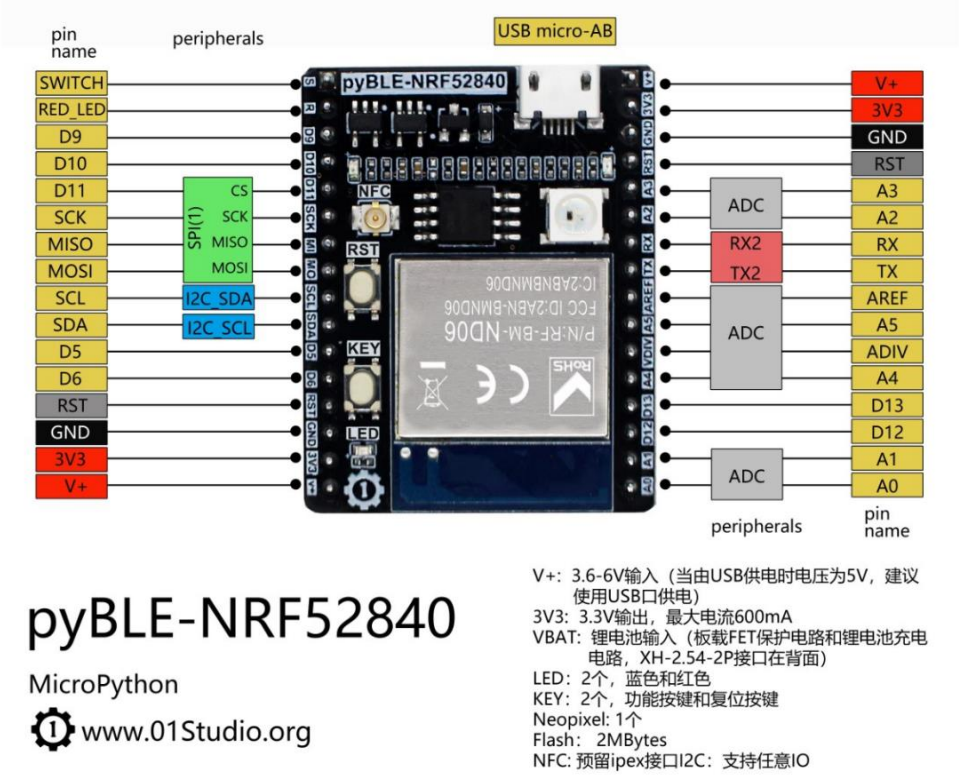


图 1-22 pyBLE-NRF52840 引脚图

功能参数	
V+	3.6v – 6v 输入（当插入 USB 时候由 USB 供电，电压为 5V，建议使用 USB 口供电）
3V3	3.3v 输出，最大电流 600mA
VBAT	锂电池输入（板载 FET 保护电路和锂电池充电电路，XH-2.54 2P 接口在背面）
LED	3 个，红色和蓝色连接 GPIO；黄色 LED 为充电指示灯
KEY	2 个，功能按键和复位按键
Neopixel	1 个 RGB 彩灯
Flash	板载 2MBytes
NFC	预留 IPEX 接口
J-tage 接口	预留烧录固件用

表 1-10 pyBLE-NRF52840 功能参数表

1.4.4.2 pyBLE-NRF52840 开发套件

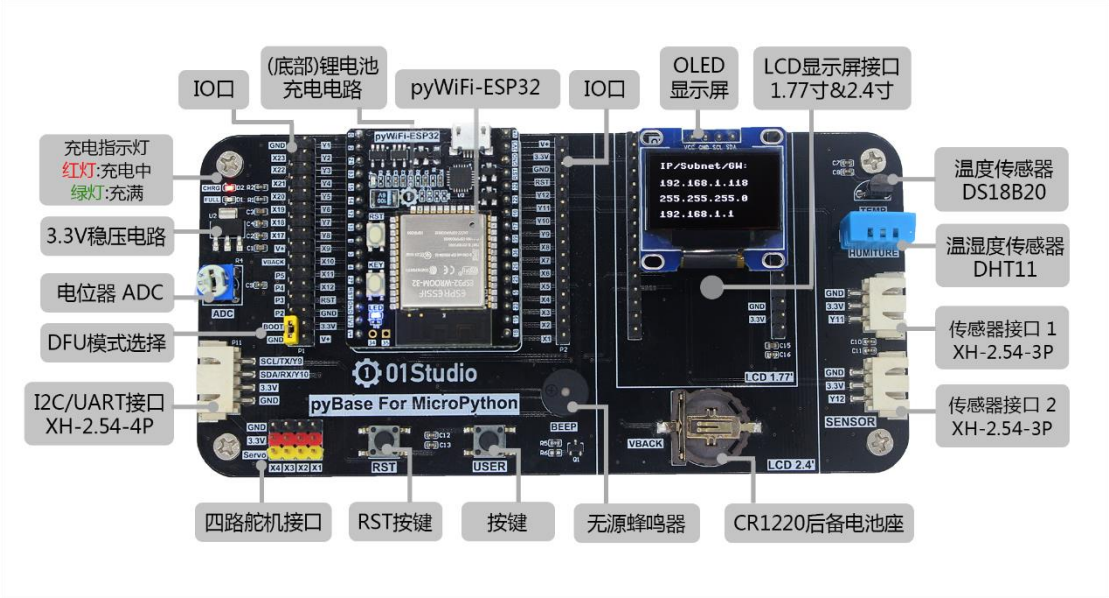


图 1-23 pyBLE-NRF52840 开发套件

主要配置	
核心板	pyBLE-NRF52840
底板	pyBase
显示屏	0.9 寸 OLED 显示屏

表 1-11

1.4.5 OpenMV4 平台

1.4.5.1 pyAI-OpenMV4

pyAI-OpenMV4 主控芯片使用 STM32H743VIT6，所以也是可以说是 STM32 平台，实现原理跟 pyBoard 有点类似。这是 01Studio 在保留官方 OpenMV4 不变的基础上增加转接板实现，具体改进如下：

（1） 兼容 pyBoard 接口；

（2） 板载锂电池输入接口和充电电路；

（3） 引出复位和功能按键，布局合理，丝印清晰。



图 1-24 pyAI-OpenMV4

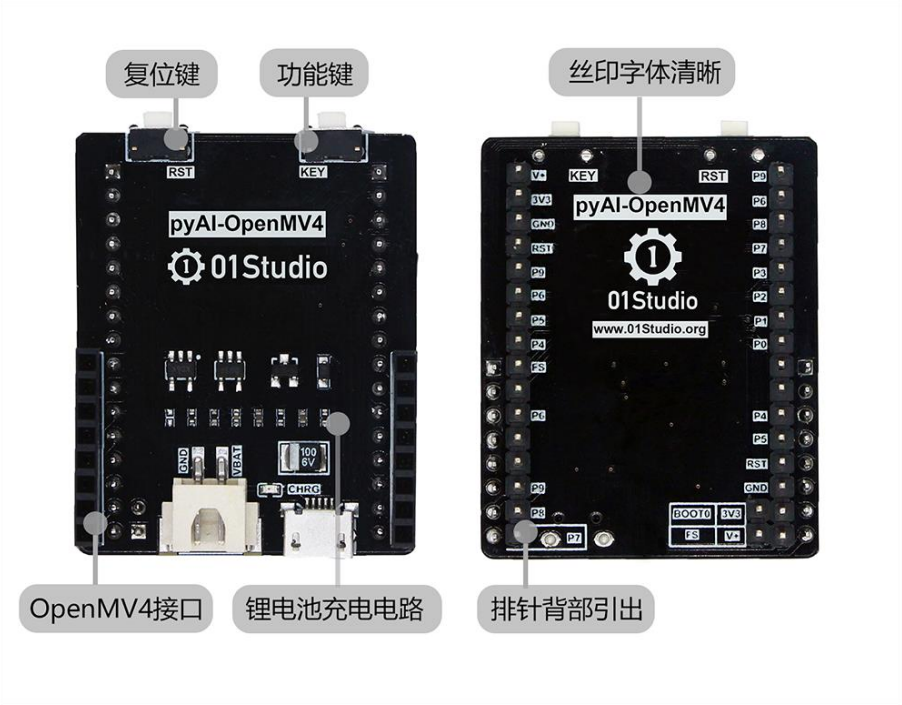


图 1-25 pyAI-OpenMV4 转接板

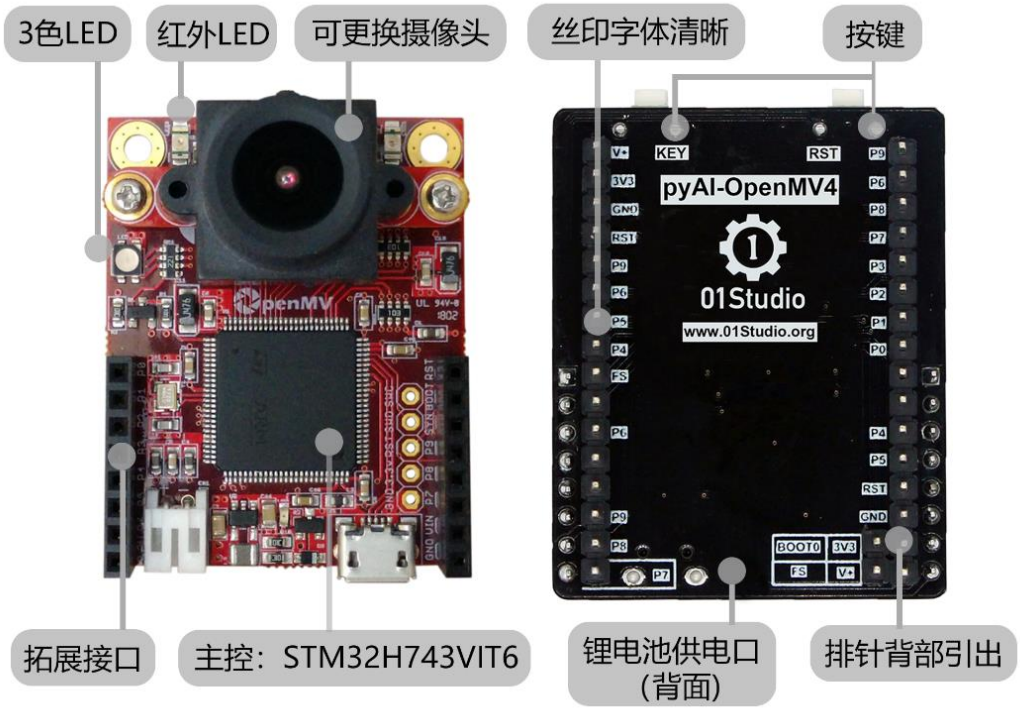


图 1-26　pyAI-OpenMV4（正反面）

功能参数

V+	3.6v – 6v 输入（当插入 USB 时候由 USB 供电，电压为 5V，建议使用 USB 口供电）
3V3	3.3v 输出，最大电流 500mA
VBAT	锂电池输入（板载 FET 保护电路和充电电路）
LED	LED1-红灯，LED2-绿灯，LED3-蓝灯，LED4-红外
按键	RST-复位，KEY-引脚“P9”
SD 卡	最大支持 32GB（SDHC）
BOOT0	如果连接到 3V3，复位后进入 DFU 模式

表 1-12　pyAI-OpenMV4 参数

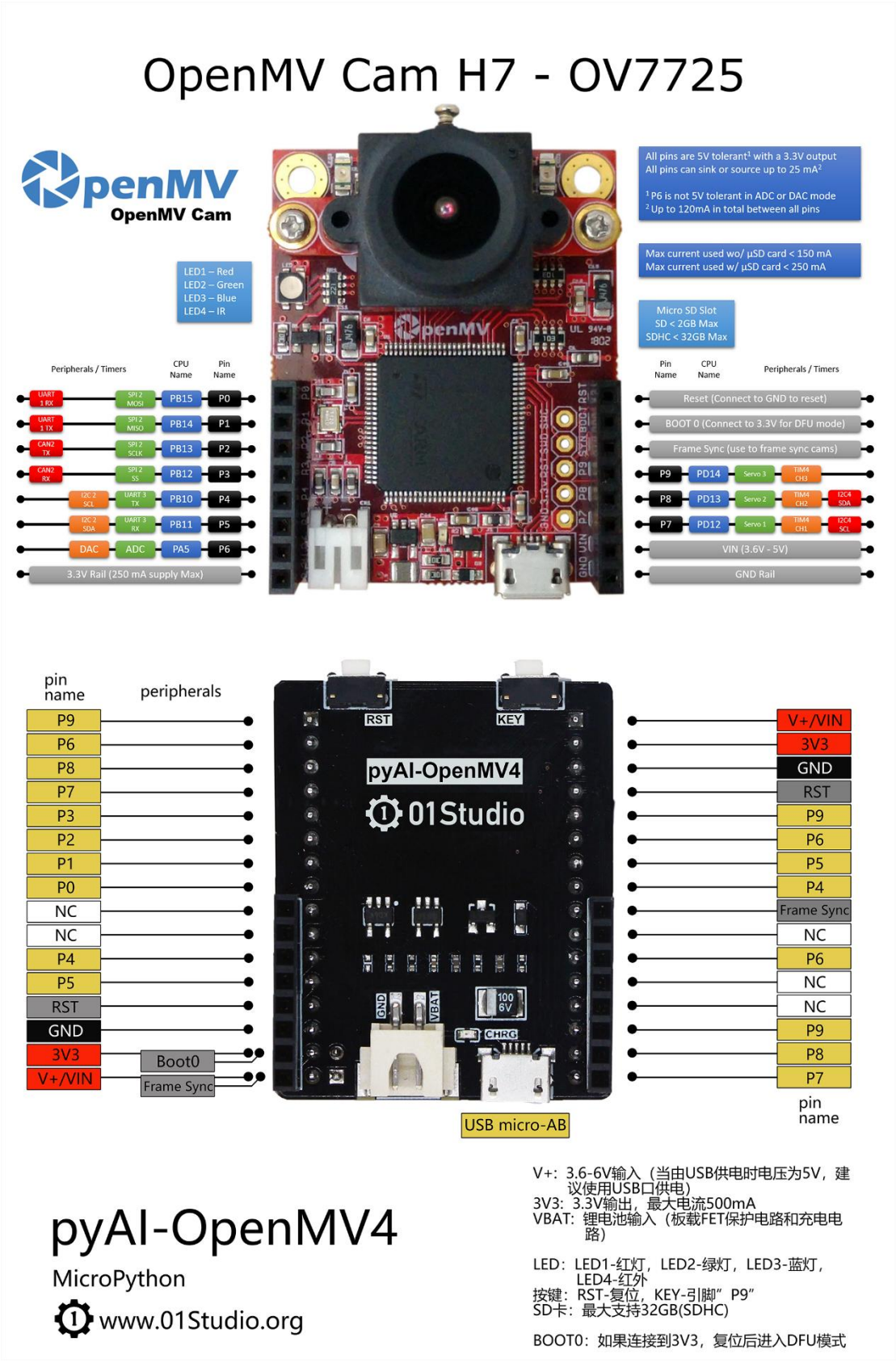


图 1-27　pyAI-OpenMV4 引脚图

1.4.5.2 pyAI-OpenMV4 Plus

pyAI-OpenMV4 Plus 相对于 pyAI-OpenMV4 性能有所提升，主控芯片使用 STM32H743IIK6，外挂 32MB SDRAM 和 32MB Flash。

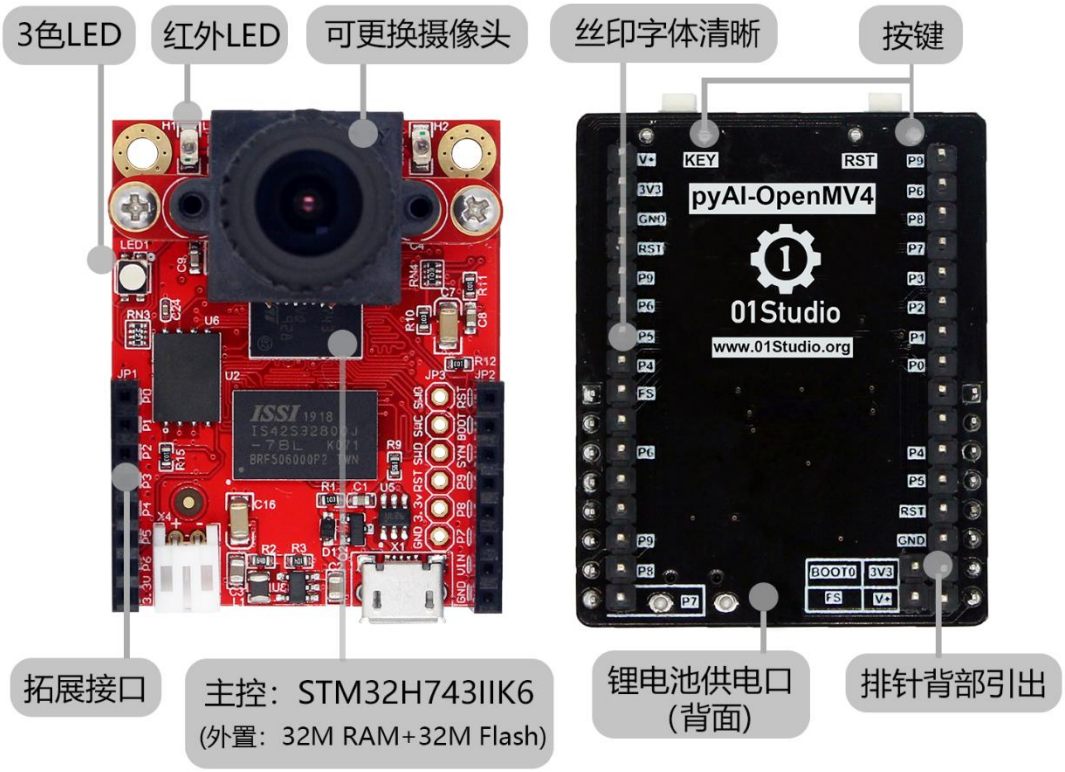


图 1-28 pyAI-OpenMV4 Plus

VS	pyAI-OpenMV4	PyAI-OpenMV4 Plus
		
标配摄像镜头	ov7725(30万像素)	ov5640(500万像素)
RAM	1MB	1MB+32MB SDRAM
FLASH	2M	2M+32MB QSPI Flash

图 1-29 OpenMV4 与 OpenMV4 Plus 主要参数对比

1.4.5.3 pyAI-OpenMV4 开发套件

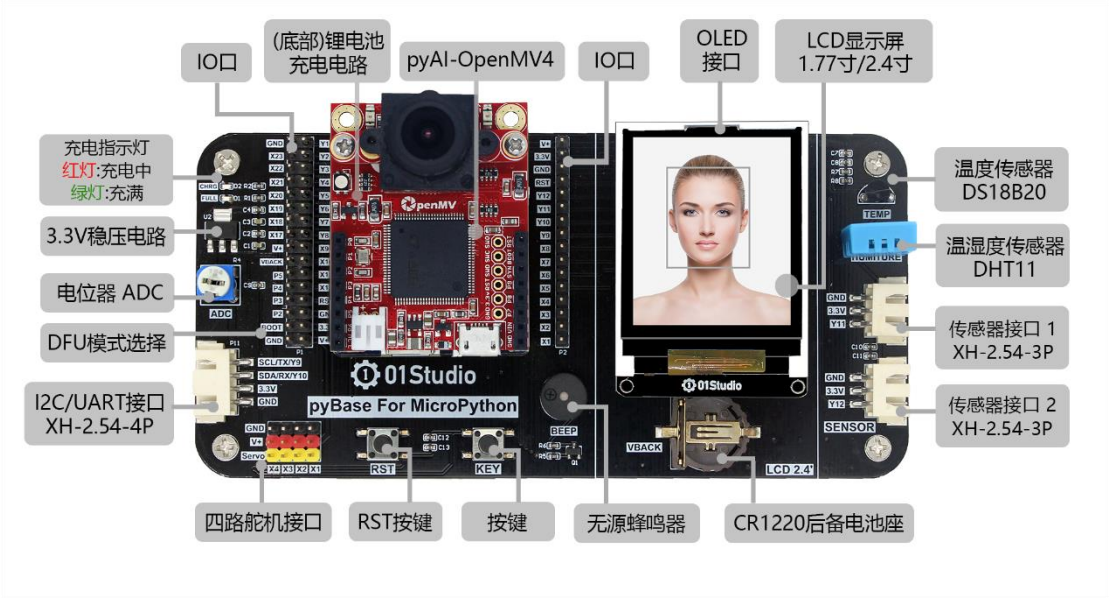


图 1-30 pyAI-OpenMV4 开发套件

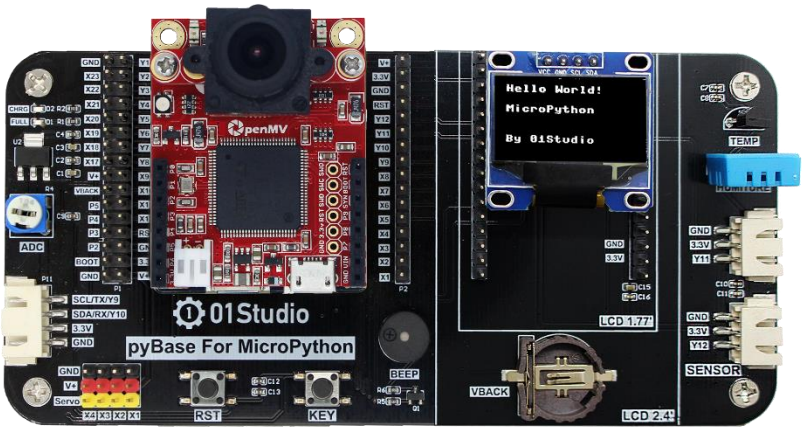


图 1-31 配 OLED 显示屏

主要配置	
核心板	pyAI-OpenMV4
底板	pyBase
显示屏	1.77 寸 LCD / 0.9 寸 OLED 显示屏

表 1-13

1.4.6 K210 平台

1.4.6.1 pyAI-K210

pyAI-K210 是由 01Studio 设计研发，基于嘉楠科技边缘计算芯片 K210（RSIC-V 架构，64 位双核）方案的一款开发板，其接口兼容 MicroPython 的 pyBoard，主要特点如下：

（1） 兼容 pyBoard 接口;

（2） 板载锂电池输入接口和充电电路;

（3） 引出复位和功能按键，布局合理，丝印清晰。

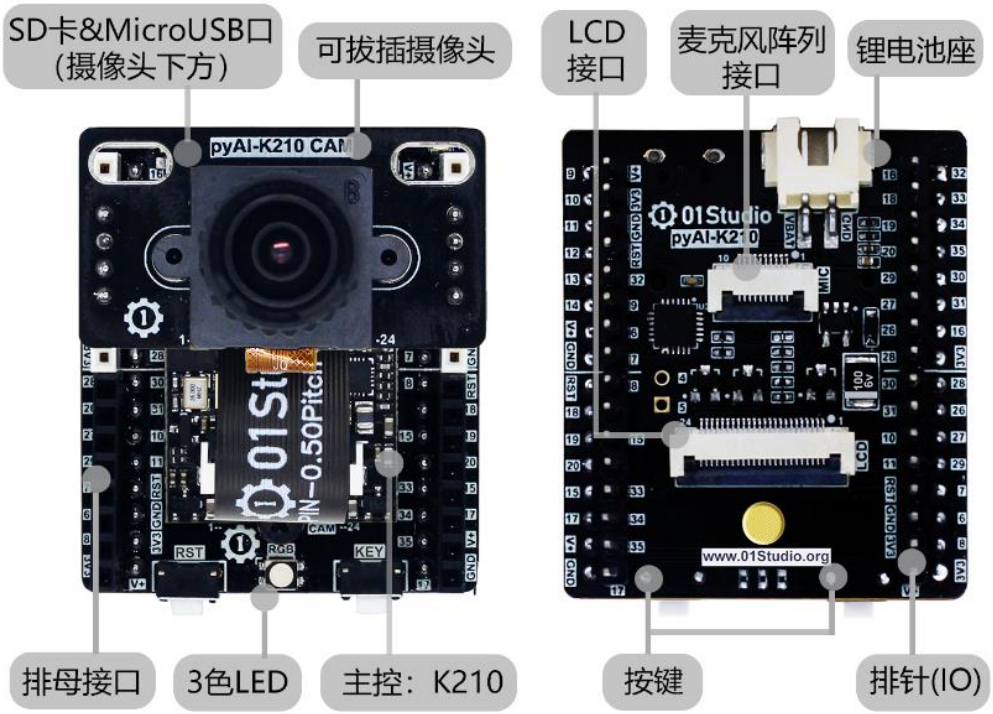


图 1-32 pyAI-K210（正反面）

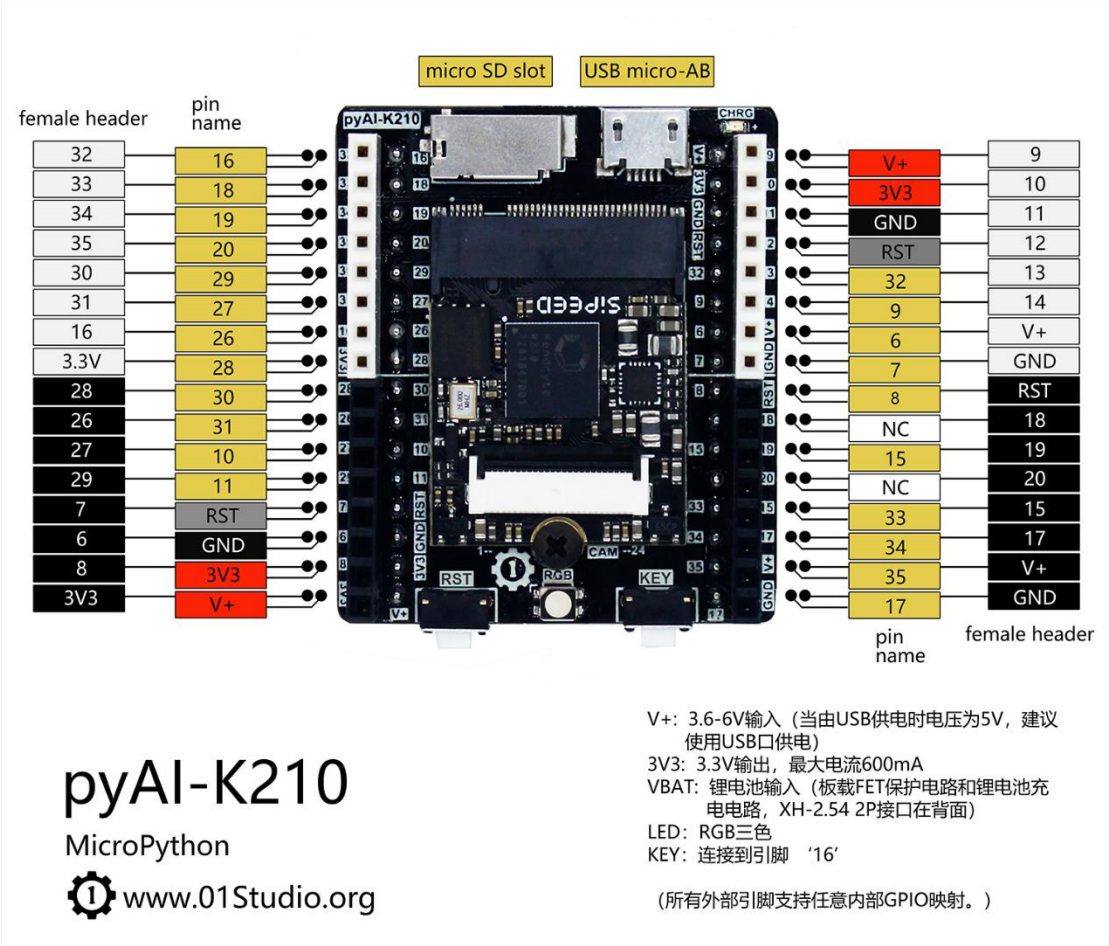


图 1-33　pyAI-K210 引脚图

功能参数	
V+	3.6V – 6V 输入（当插入 USB 时候由 USB 供电，电压为 5V，建议使用 USB 口供电）
3V3	3.3V 输出，最大电流 600mA
VBAT	锂电池输入（板载 FET 保护电路和充电电路）
LED	RGB 三色
按键	RST-复位；KEY-引脚 “16”
SD 卡	最大支持 32GB（SDHC）
其它	所有外部引脚支持任意 GPIO 内部映射

表 1-14 pyAI-K210 功能参数表

1.4.6.2 pyAI-K210 开发套件

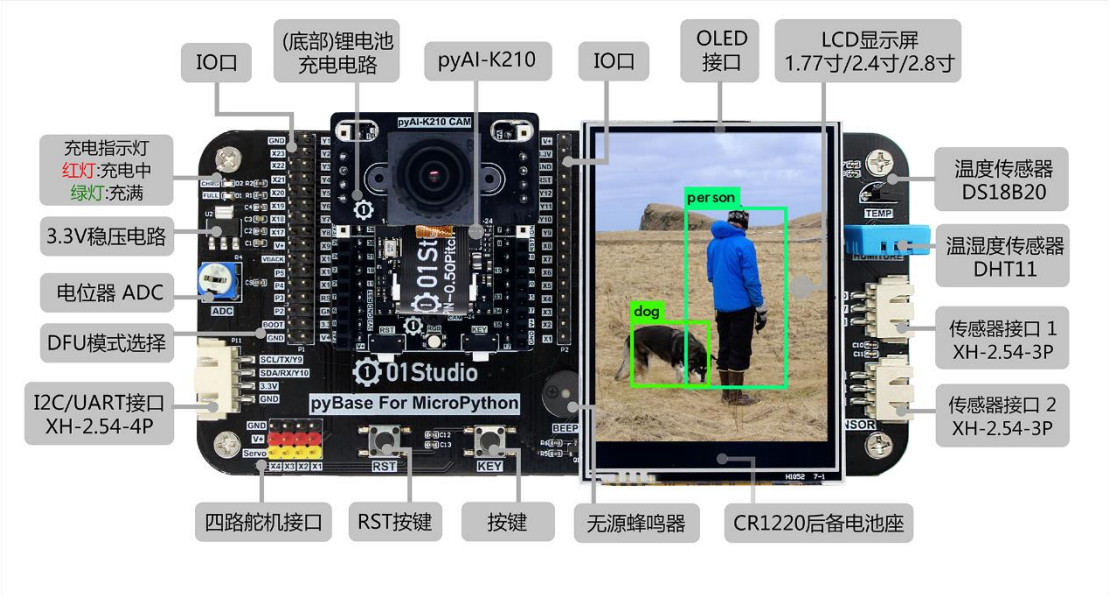


图 1-34 pyAI-K210 开发套件功能描述

主要配置	
核心板	pyAI-K210（配摄像头）
底板	pyBase
显示屏	2.8 寸 LCD / 0.9 寸 OLED 显示屏

表 1-15

1.4.7 pyBase

pyBase 是 01Studio 针对各 micropython 开发平台量身定制的底板，可以使用它可以做更多的 MicroPython 实验，pyBase 同时设计了外设接口，扩展性非常强。以下是详细的功能说明：

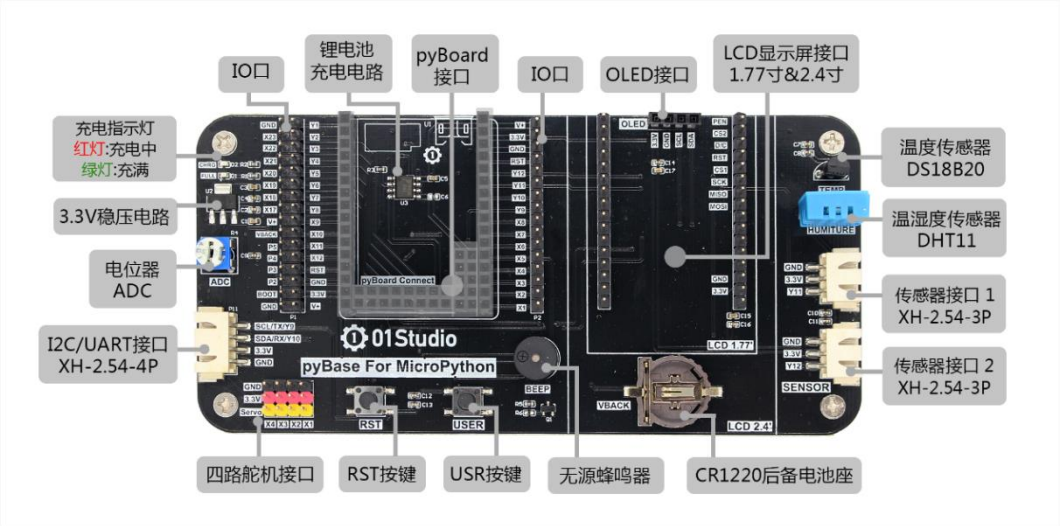


图 1-35 pyBase 功能说明

功能参数	
pyboard 接口	兼容 pyboard v1.1-CN 以及官方的 pyboard
2.54mm 排针	引出 pyboard 全部接口
按键 2 个	引出 RST 和 USR 按键
电位器	ADC 输入
无源蜂鸣器	DAC 输出
纽扣电池座	安装 CR1220 纽扣电池
Servo 接口	舵机接口 X1-X4 (连接舵机需要外接隔离电路)
OLED 接口	4P/0.96 寸/I2C/OLED 显示屏
1.77 寸 LCD 接口	适用于 01Studio 1.77 寸 LCD 显示屏
2.4 寸 LCD 接口	适用于 01Studio 2.4 寸 LCD 显示屏（带电阻触摸）
温度传感器	DS18B20
温湿度传感器	DHT11
Sensor 接口 1	3P 防呆接口，用于外接传感器

Sensor 接口 1	3P 防呆接口，用于外接传感器
通讯接口	4P 防呆接口，用于外接 UART/I2C 设备
锂电池充电电路	对接入的锂电池进行充电（红灯->充电，绿灯->充满）

表 1-16

1.4.8 IOT/通讯模块

pyIOT 开源项目由 01Studio 发起，旨在为市面上成熟的串口物联网模块开发 MicroPython 库，让用户可以使用 python 编程快速实现各类物联网相关应用。

1.4.8.1 pyIOT-BLE TLS01

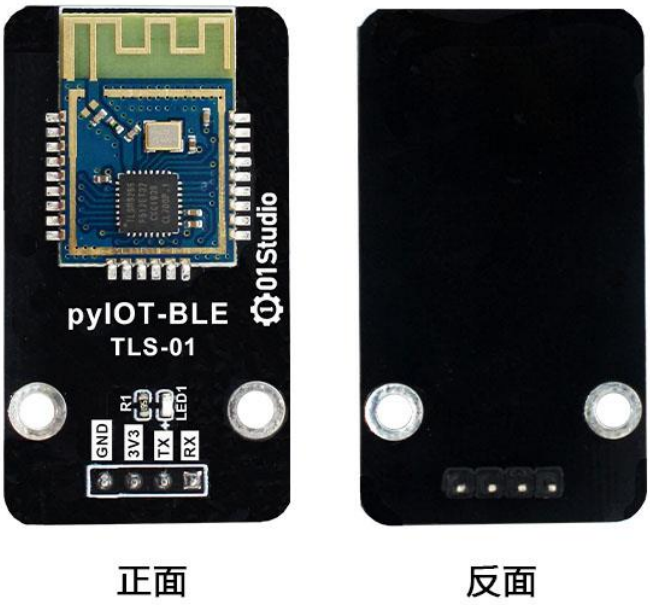
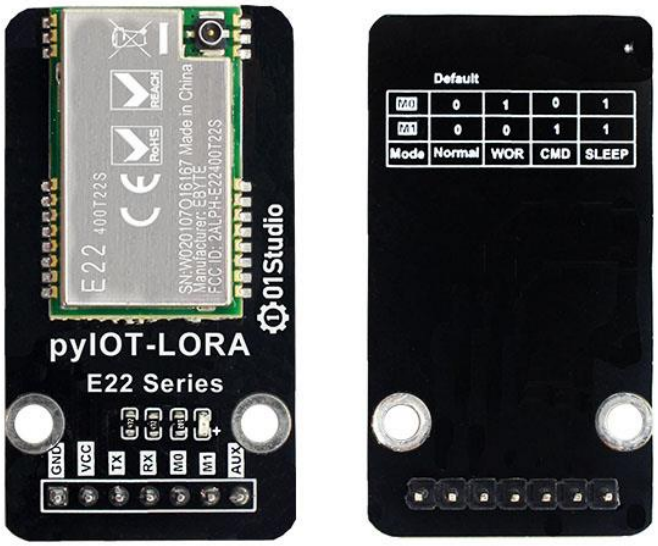


图 1-36

功能参数	
蓝牙主控	TLSR8266
控制方式	UART (串口)
蓝牙版本	BLE 4.0 (从机)
功耗	工作电流：<8.8mA，广播电流：<1mA
引脚	GND, 3.3V, TX, RX
模块尺寸	4.5*2.5cm

表 1-17

1.4.8.2 pyIOT-LORA E22



正面　　反面

图 1-37

功能参数	
工作频段	410~493MHz (433M)
射频芯片	SX1268
发射功率	22dBm
通讯距离	最大：5km
通讯接口	UART (串口)
天线	IPEX
发射电流	110mA
供电电压	2.3-5.2V
工作温度	-40℃ - 85℃
波特率	1200 – 115200 bps (默认 9600)
空中速率	0.3k – 42.5kbps
接收长度	1024 字节（自动分包）
模块尺寸	4.2*2.5cm

表 1-18

1.4.8.3 pyIOT-GPS

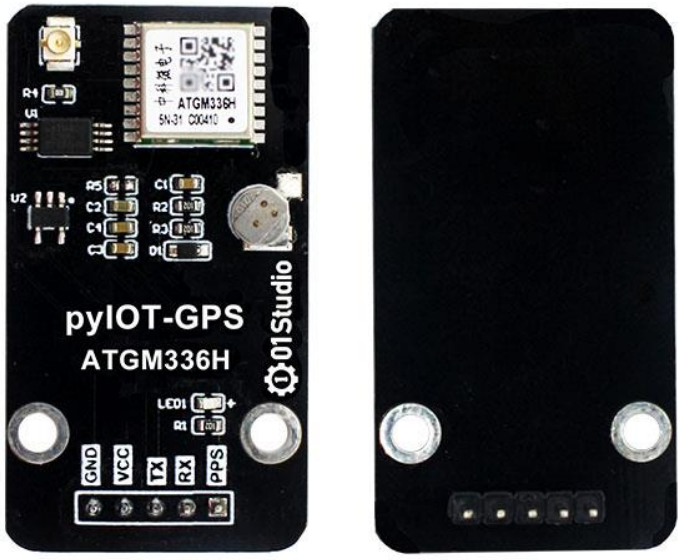


图 1-38

功能参数	
主控芯片	ATGM336H-5N
卫星信号	GPS/BDS/GLONASS
波特率	9600bps
工作电压	3.3V-5V
串口电平	3.3V 或 5V (自适应)
定位精度	2.5m (开阔地点)
功耗	工作：<25mA，待机：<10uA (@3.3V)
模块尺寸	4.2*2.5cm

表 1-19

1.4.9 传感器模块

1.4.9.1 人体感应传感器模块

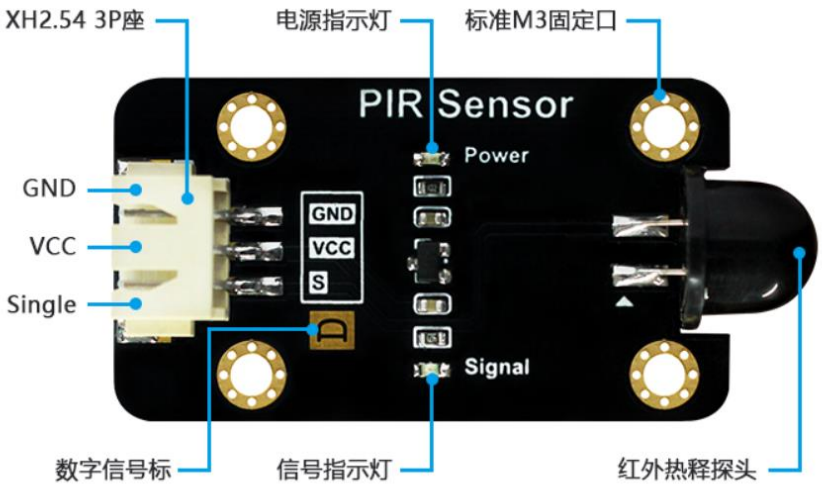


图 1-39 人体感应传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 65℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	数字信号： 检测到人体：高电平 3.3V；持续 3-8 秒 未检测到人体：低电平 0V。
感应角度	100°
感应距离	8 米
模块尺寸	4.5*2.5cm

表 1-20

1.4.9.2 光敏传感器模块

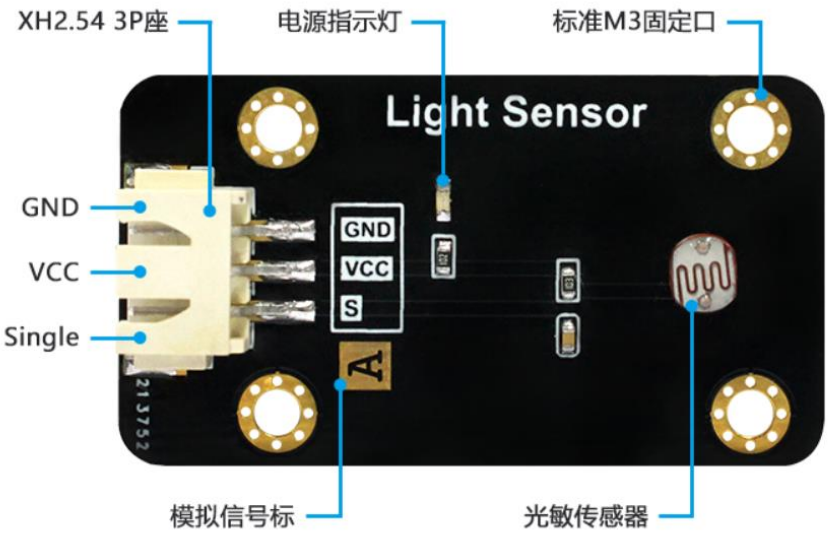


图 1-40 光敏传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	模拟信号：0-3.3V （VCC=3.3V 时）
模块尺寸	4.5*2.5cm

表 1-21

1.4.9.3 土壤湿度传感器模块

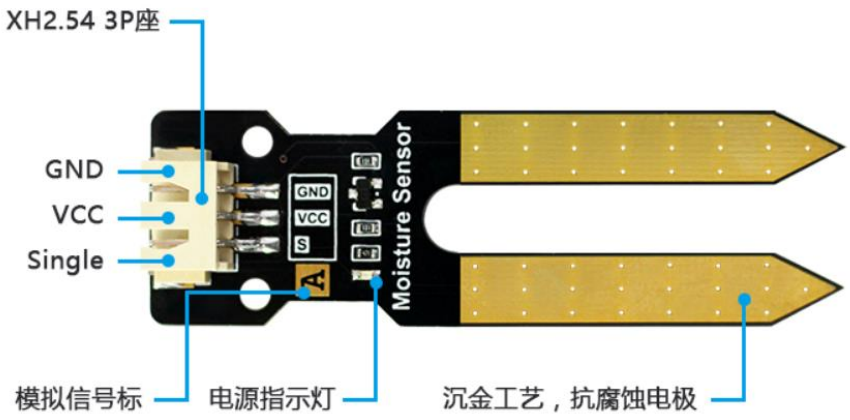


图 1-41 土壤湿度传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-40 至 85℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	模拟信号：0-3.3V （VCC=3.3V 时）
模块尺寸	6.6*2.0cm

表 1-22

1.4.9.4 水位传感器模块

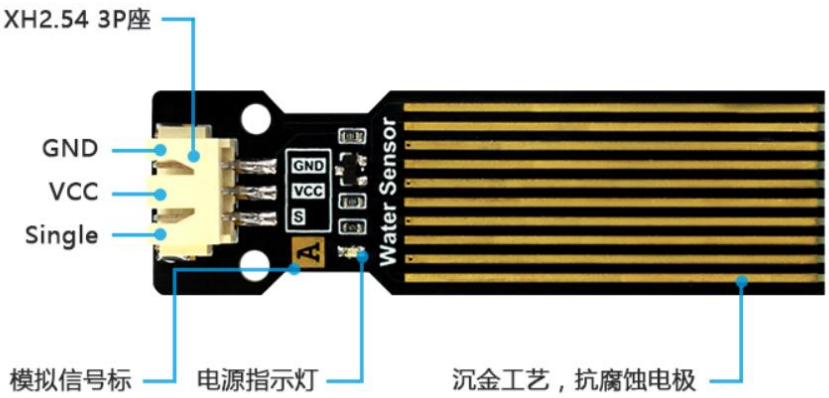


图 1-42 水位传感器模块

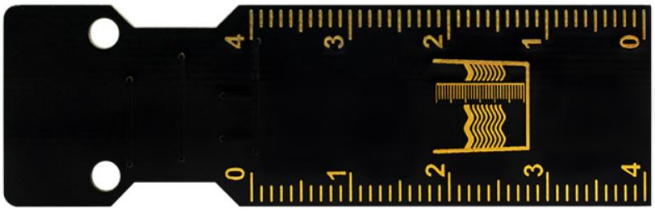


图 1-43 背面

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	0 至 60℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	模拟信号：0-3.3V （VCC=3.3V 时）
模块尺寸	6.6*2.0cm
注意事项	水位传感器接口旁元件不能接触水，否则容易短路。所以测量时候要注意液体水位的高度。

表 1-23

1.4.9.5 大气压强传感器模块

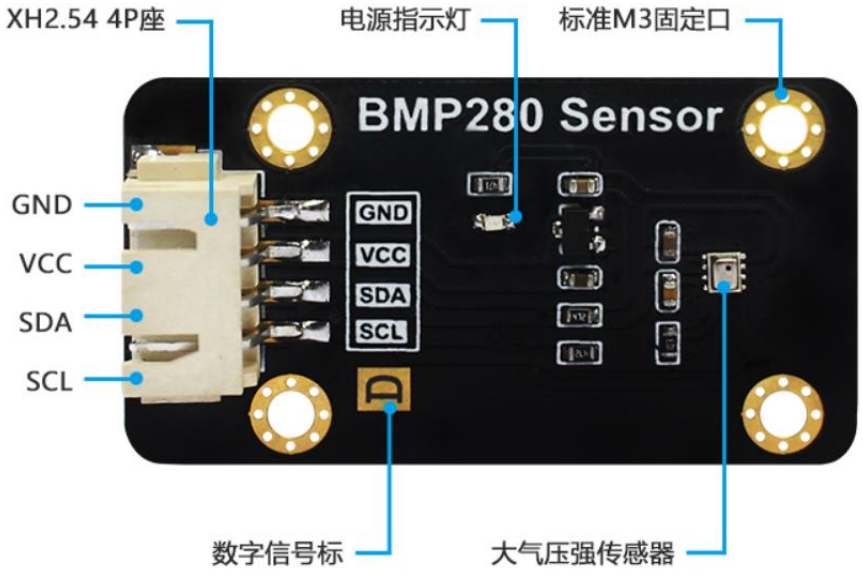


图 1-44 大气压强传感器模块

功能参数	
传感器型号	BMP280
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85℃
接口定义	XH2.54 防呆接口（4Pin）【GND、VCC、SDA、SCL】
通讯信号	I2C 总线
I2C 地址	0x76（BMP280 的 SDO 默认下拉）； 当 BMP280 的 SDO 引脚上拉时，I2C 地址为：0x77
模块尺寸	4.5*2.5cm

表 1-24

1.4.9.6 超声波模块

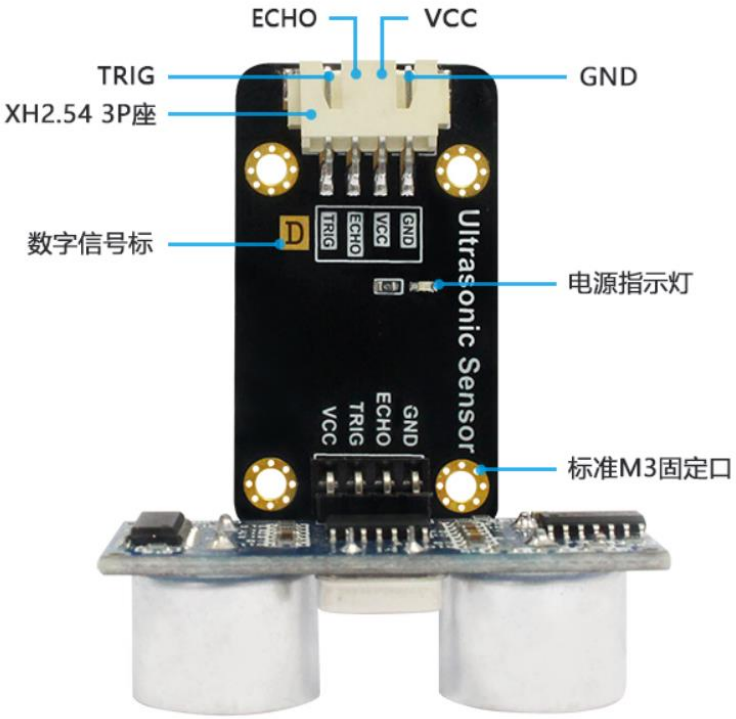


图 1-45 超声波模块

功能参数	
传感器型号	HCSR04
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85℃
接口定义	XH2.54 防呆接口（4Pin） 【GND、VCC、ECHO、TRIG】
通讯信号	IO 数字接口
测量距离	2-450 cm
测量精度	0.5 cm
整体尺寸	5.5*4.5*3.0 cm

表 1-25

1.4.9.7 温度传感器模块（金属探头封装）



图 1-46 金属探头温度传感器

功能参数	
供电电压	3.3-5V
工作温度	-55 至 125℃
接口定义	XH2.54 防呆接口（3Pin） 【GND、VCC、Single】
通讯信号	单总线
测量范围	-55 至 125℃
测量精度	-0.5℃
探头尺寸	5*0.6cm
引线长度	1 米

表 1-26

1.4.10 拓展配件

1.4.10.1 OLED 显示屏



图 1-47 OLED 显示屏

功能参数	
供电电压	3.3V
屏幕尺寸	0.9 寸
颜色参数	黑底白字
通讯方式	I2C 总线
接口定义	2.54mm 排针（4Pin） 【VCC、GND、SCL、SDA】
整体尺寸	2.8*2.8cm

表 1-27

1.4.10.2　1.77 寸 LCD 显示屏

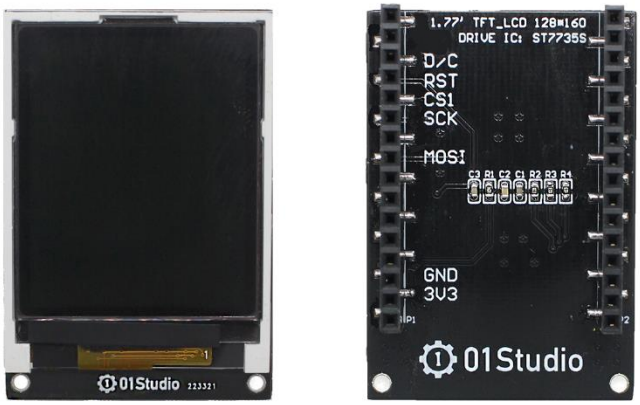


图 1-48 1.77 寸 LCD 显示屏

功能参数	
供电电压	3.3V
屏幕尺寸	1.77 寸
分辨率	128*160
颜色参数	TFT 彩色
驱动芯片	ST7735S
通讯方式	SPI 总线
接口定义	2.54mm 排母（兼容 pyboard v1.1 接口）
整体尺寸	5.0*3.5 cm

表 1-28

1.4.10.3　2.4 寸 LCD 显示屏（电阻触摸）

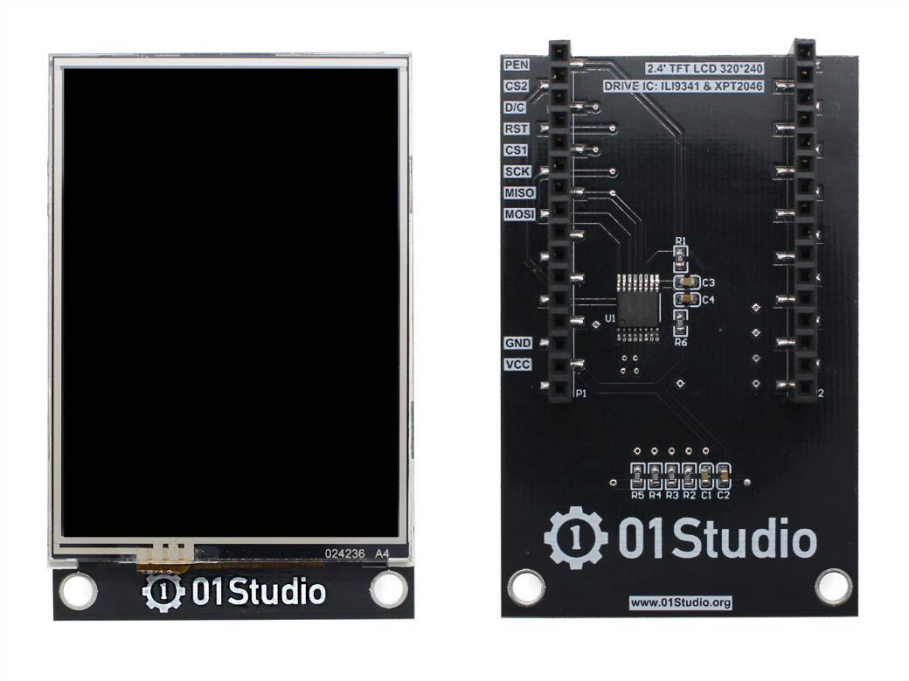


图 1-49　2.4 寸 LCD 显示屏（电阻触摸）

功能参数	
供电电压	3.3V
屏幕尺寸	2.4 寸
分辨率	240*320
颜色参数	TFT 彩色
驱动芯片	ILI9341+XPT2046
触摸方式	电阻屏
通讯方式	SPI 总线
接口定义	2.54mm 排母（兼容 pyboard v1.1 接口）
整体尺寸	6.6*4.2 cm

表 1-29

1.4.10.4　2.8 寸 LCD 显示屏（电阻触摸）

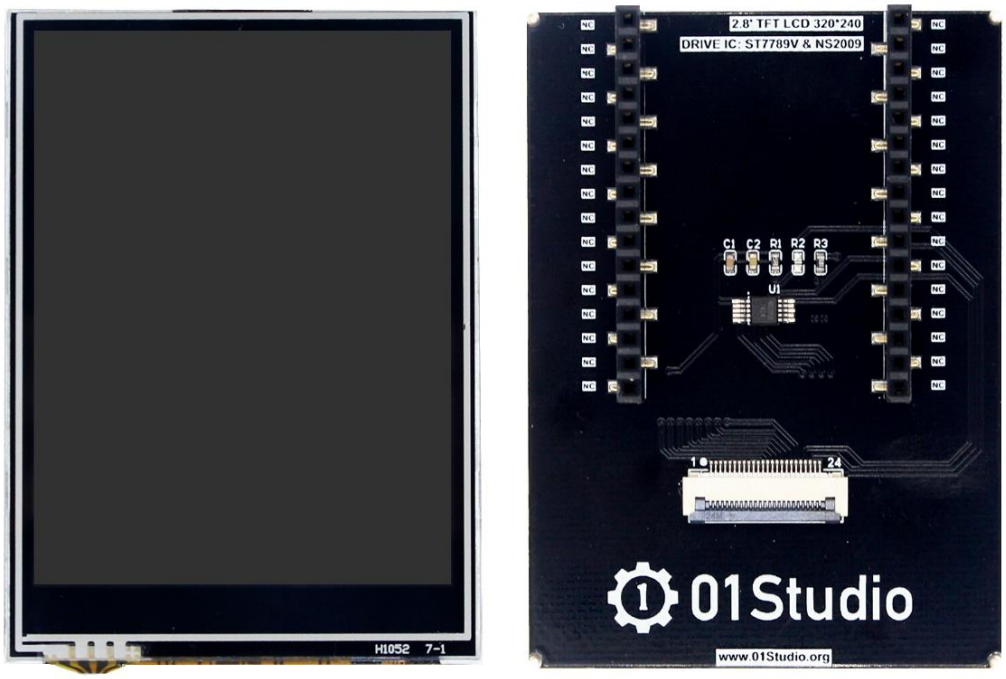


图 1-50　2.4 寸 LCD 显示屏（电阻触摸）

功能参数	
供电电压	3.3V
屏幕尺寸	2.8 寸
分辨率	240*320
颜色参数	TFT 彩色
驱动芯片	ST7789V+NS2009
触摸方式	电阻屏
通讯方式	8 位并口总线
接口定义	24P-0.5mm FPC 座
整体尺寸	7*5 cm

表 1-30

1.4.10.5　3.2 寸 LCD 显示屏（电阻触摸）

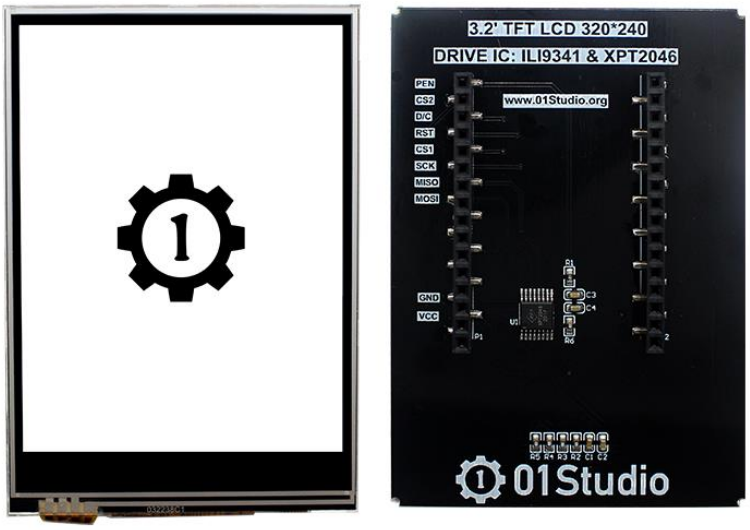


图 1-51　3.2 寸 LCD 显示屏（电阻触摸）

功能参数	
供电电压	3.3V
屏幕尺寸	3.2 寸
分辨率	240*320
颜色参数	TFT 彩色
驱动芯片	ILI9341+XPT2046
触摸方式	电阻屏
通讯方式	SPI 总线
接口定义	2.54mm 排母（兼容 pyboard v1.1 接口）
整体尺寸	7.7*5.5 cm

表 1-31

1.4.10.6 W5500 以太网模块

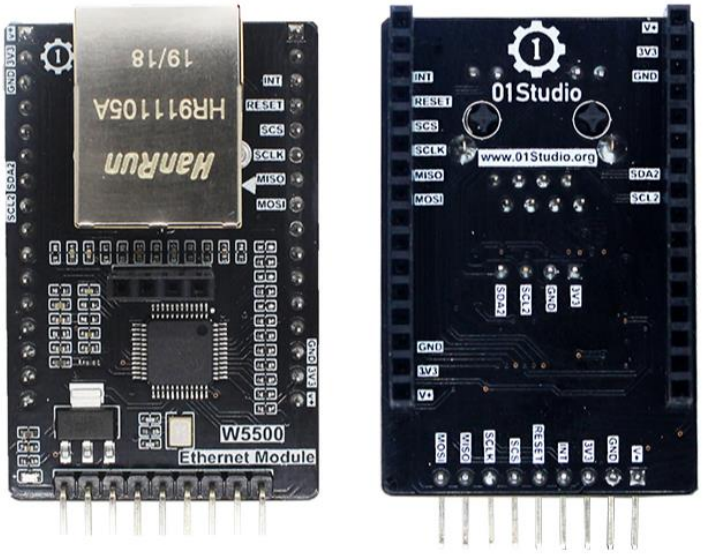


图 1-52 W5500 以太网模块

功能参数	
供电电压	3.3V 或 5V
主控芯片	W5500
控制方式	SPI 总线
接口定义	1、2.54mm 排母（兼容 pyboard v1.1 接口）； 2、2.54mm 排针
其它功能	支持 OLED 扩展接口
整体尺寸	6.6*4.2 cm

表 1-32

1.4.10.7 舵机

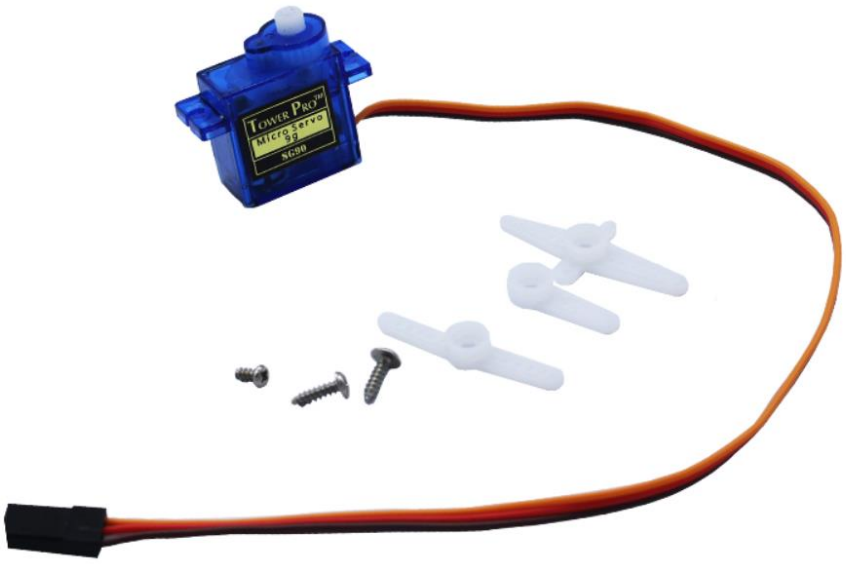


图 1-53 舵机

功能参数	
尺寸	32*30*1.1 mm
重量	15g
扭力	1.6kg/cm
接口	XH2.54 接口（3Pin） 【GND（黑）、VCC（红）、Single（橙）】
工作电压	3.5-6V
工作温度	-30℃-60 ℃
转动角度	180° 和 360° 连续旋转。
应用场景	固定翼、直升机 KT、机器人、机械臂、航模等

表 1-33

1.4.10.8 RGB 灯带

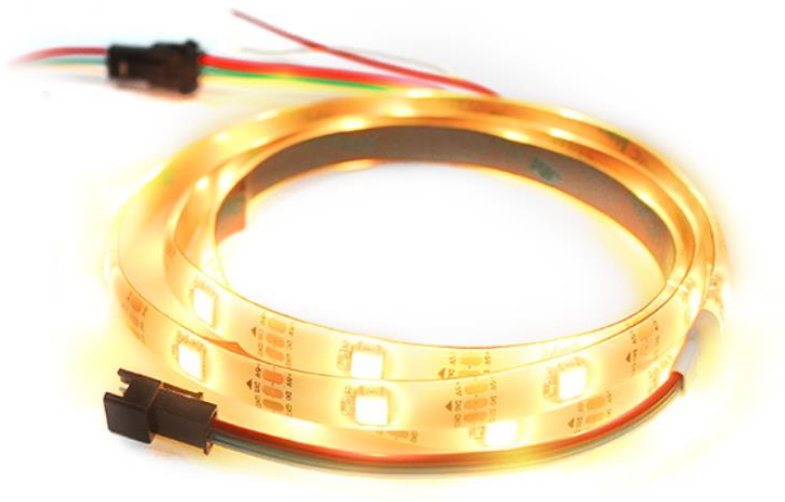


图 1-54 RGB 灯带

功能参数	
灯带长度	1 米
灯珠数量	30
颜色	RGB 全彩
接口定义	XH2.54 防呆接口（3Pin） 【GND、VCC、Single】
驱动 IC	WS2812B
供电电压	3.3-5V
功率	每颗灯珠最高 0.3W
应用场景	流水灯/呼吸灯/节日灯饰布置等

表 1-34

1.4.10.9 USB 转串口 TTL

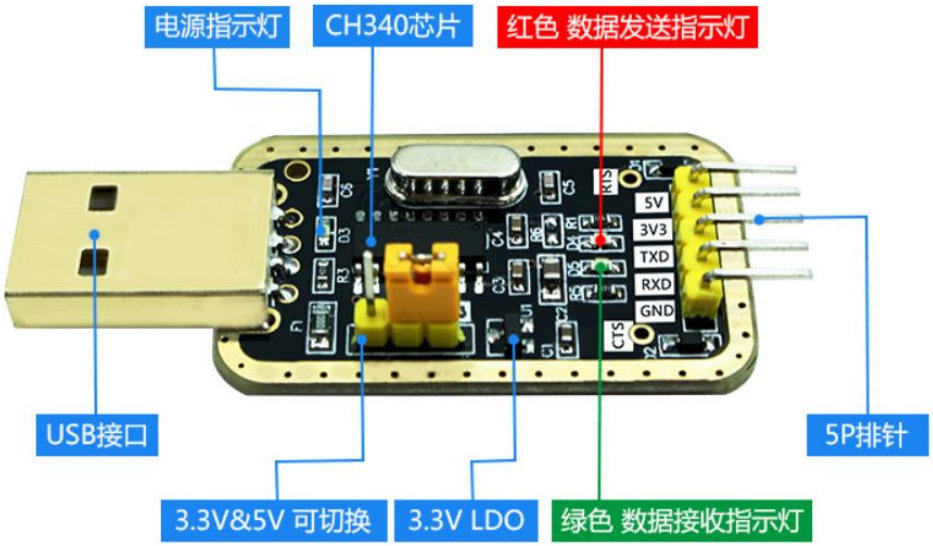


图 1-55 USB 转串口 TTL

功能参数	
驱动芯片	CH340
对外供电	5V 和 3.3V
支持电平	3.3V 和 5V 可切换
引脚接口	2.54MM 排针

表 1-35

第2章　Python 基础知识

由于 python 语言是学习 micropython 的基础，为了照顾没有 pyhton 基础的朋友，我们一直在思考应该给大家提供怎么样的教程。Python 相关的书籍和学习资料相信大家能在网上找到不少，所以这里整理给大家的经典的 python3 快速学习资料，你甚至可以使用它来当作 python 字典查阅！

【原著：Louie Dinh，翻译：Geoff Liu】

```
# 用井字符开头的是单行注释

""" 多行字符串用三个引号
    包裹，也常被用来做多
    行注释
"""
```

2.1　原始数据类型和运算符

```
####################################################
## 1. 原始数据类型和运算符
####################################################

# 整数
3  # => 3

# 算术没有什么出乎意料的
1 + 1  # => 2
8 - 1  # => 7
10 * 2  # => 20
```

```
# 但是除法例外，会自动转换成浮点数
35 / 5  # => 7.0
5 / 3  # => 1.6666666666666667

# 整数除法的结果都是向下取整
5 // 3     # => 1
5.0 // 3.0 # => 1.0 # 浮点数也可以
-5 // 3  # => -2
-5.0 // 3.0 # => -2.0

# 浮点数的运算结果也是浮点数
3 * 2.0 # => 6.0

# 模除
7 % 3 # => 1

# x 的 y 次方
2**4 # => 16

# 用括号决定优先级
(1 + 3) * 2  # => 8

# 布尔值
True
False

# 用 not 取非
not True  # => False
not False  # => True
```

```
# 逻辑运算符，注意 and 和 or 都是小写
True and False # => False
False or True # => True

# 整数也可以当作布尔值
0 and 2 # => 0
-5 or 0 # => -5
0 == False # => True
2 == True # => False
1 == True # => True

# 用==判断相等
1 == 1  # => True
2 == 1  # => False

# 用!=判断不等
1 != 1  # => False
2 != 1  # => True

# 比较大小
1 < 10  # => True
1 > 10  # => False
2 <= 2  # => True
2 >= 2  # => True

# 大小比较可以连起来!
1 < 2 < 3  # => True
2 < 3 < 2  # => False
```

```
# 字符串用单引双引都可以
"这是个字符串"
'这也是个字符串'

# 用加号连接字符串
"Hello " + "world!"  # => "Hello world!"

# 字符串可以被当作字符列表
"This is a string"[0]  # => 'T'

# 用.format 来格式化字符串
"{} can be {}".format("strings", "interpolated")

# 可以重复参数以节省时间
"{0} be nimble, {0} be quick, {0} jump over the {1}".format("Jack",
"candle stick")
# => "Jack be nimble, Jack be quick, Jack jump over the candle stick"

# 如果不想数参数，可以用关键字
"{name} wants to eat {food}".format(name="Bob", food="lasagna")
# => "Bob wants to eat lasagna"

# 如果你的 Python3 程序也要在 Python2.5 以下环境运行，也可以用老式的格式化语法
"%s can be %s the %s way" % ("strings", "interpolated", "old")

# None 是一个对象
None  # => None

# 当与 None 进行比较时不要用 ==，要用 is。is 是用来比较两个变量是否指向同一个对
象。
```

```
"etc" is None  # => False

None is None  # => True


# None，0，空字符串，空列表，空字典都算是 False

# 所有其他值都是 True

bool(0)  # => False

bool("")  # => False

bool([]) # => False

bool({}) # => False
```

2.2 变量和集合

```
####################################################
## 2. 变量和集合
####################################################

# print 是内置的打印函数
print("I'm Python. Nice to meet you!")

# 在给变量赋值前不用提前声明
# 传统的变量命名是小写，用下划线分隔单词
some_var = 5
some_var  # => 5

# 访问未赋值的变量会抛出异常
# 参考流程控制一段来学习异常处理
some_unknown_var  # 抛出 NameError

# 用列表(list)储存序列
li = []
# 创建列表时也可以同时赋给元素
other_li = [4, 5, 6]

# 用 append 在列表最后追加元素
li.append(1)    # li 现在是[1]
li.append(2)    # li 现在是[1, 2]
li.append(4)    # li 现在是[1, 2, 4]
li.append(3)    # li 现在是[1, 2, 4, 3]
# 用 pop 从列表尾部删除
```

```
li.pop()        # => 3 且 li 现在是[1, 2, 4]
# 把 3 再放回去
li.append(3)    # li 变回[1, 2, 4, 3]

# 列表存取跟数组一样
li[0]  # => 1
# 取出最后一个元素
li[-1]  # => 3

# 越界存取会造成 IndexError
li[4]  # 抛出 IndexError

# 列表有切割语法
li[1:3]  # => [2, 4]
# 取尾
li[2:]  # => [4, 3]
# 取头
li[:3]  # => [1, 2, 4]
# 隔一个取一个
li[::2]   # =>[1, 4]
# 倒排列表
li[::-1]   # => [3, 4, 2, 1]
# 可以用三个参数的任何组合来构建切割
# li[始:终:步伐]

# 用 del 删除任何一个元素
del li[2]   # li is now [1, 2, 3]

# 列表可以相加
# 注意：li 和 other_li 的值都不变
```

```
li + other_li   # => [1, 2, 3, 4, 5, 6]

# 用 extend 拼接列表
li.extend(other_li)   # li 现在是[1, 2, 3, 4, 5, 6]

# 用 in 测试列表是否包含值
1 in li   # => True

# 用 len 取列表长度
len(li)   # => 6


# 元组是不可改变的序列
tup = (1, 2, 3)
tup[0]   # => 1
tup[0] = 3  # 抛出 TypeError

# 列表允许的操作元组大都可以
len(tup)   # => 3
tup + (4, 5, 6)   # => (1, 2, 3, 4, 5, 6)
tup[:2]   # => (1, 2)
2 in tup   # => True

# 可以把元组合列表解包，赋值给变量
a, b, c = (1, 2, 3)     # 现在 a 是 1，b 是 2，c 是 3
# 元组周围的括号是可以省略的
d, e, f = 4, 5, 6
# 交换两个变量的值就这么简单
e, d = d, e     # 现在 d 是 5，e 是 4
```

```
# 用字典表达映射关系
empty_dict = {}
# 初始化的字典
filled_dict = {"one": 1, "two": 2, "three": 3}

# 用[]取值
filled_dict["one"]   # => 1



# 用 keys 获得所有的键。
# 因为 keys 返回一个可迭代对象，所以在这里把结果包在 list 里。我们下面会详细介绍可迭代。
# 注意：字典键的顺序是不定的，你得到的结果可能和以下不同。
list(filled_dict.keys())   # => ["three", "two", "one"]



# 用 values 获得所有的值。跟 keys 一样，要用 list 包起来，顺序也可能不同。
list(filled_dict.values())   # => [3, 2, 1]



# 用 in 测试一个字典是否包含一个键
"one" in filled_dict   # => True
1 in filled_dict   # => False

# 访问不存在的键会导致 KeyError
filled_dict["four"]   # KeyError

# 用 get 来避免 KeyError
filled_dict.get("one")   # => 1
```

```
filled_dict.get("four")   # => None
# 当键不存在的时候 get 方法可以返回默认值
filled_dict.get("one", 4)   # => 1
filled_dict.get("four", 4)   # => 4

# setdefault 方法只有当键不存在的时候插入新值
filled_dict.setdefault("five", 5)  # filled_dict["five"]设为 5
filled_dict.setdefault("five", 6)  # filled_dict["five"]还是 5

# 字典赋值
filled_dict.update({"four":4}) # => {"one": 1, "two": 2, "three": 3,
"four": 4}
filled_dict["four"] = 4  # 另一种赋值方法

# 用 del 删除
del filled_dict["one"]  # 从 filled_dict 中把 one 删除



# 用 set 表达集合
empty_set = set()
# 初始化一个集合，语法跟字典相似。
some_set = {1, 1, 2, 2, 3, 4}   # some_set 现在是{1, 2, 3, 4}

# 可以把集合赋值于变量
filled_set = some_set

# 为集合添加元素
filled_set.add(5)   # filled_set 现在是{1, 2, 3, 4, 5}

# & 取交集
```

```
other_set = {3, 4, 5, 6}

filled_set & other_set   # => {3, 4, 5}


# | 取并集

filled_set | other_set   # => {1, 2, 3, 4, 5, 6}


# - 取补集

{1, 2, 3, 4} - {2, 3, 5}   # => {1, 4}


# in 测试集合是否包含元素

2 in filled_set   # => True

10 in filled_set   # => False
```

2.3 流程控制和迭代器

```
####################################################
## 3. 流程控制和迭代器
####################################################

# 先随便定义一个变量
some_var = 5

# 这是个 if 语句。注意缩进在 Python 里是有意义的
# 印出"some_var 比 10 小"
if some_var > 10:
    print("some_var 比 10 大")
elif some_var < 10:    # elif 句是可选的
    print("some_var 比 10 小")
else:                  # else 也是可选的
    print("some_var 就是 10")



"""
用 for 循环语句遍历列表
打印:
    dog is a mammal
    cat is a mammal
    mouse is a mammal
"""
for animal in ["dog", "cat", "mouse"]:
    print("{} is a mammal".format(animal))

"""
```

```
"range(number)"返回数字列表从 0 到给的数字
打印:
    0
    1
    2
    3
"""
for i in range(4):
    print(i)

"""
while 循环直到条件不满足
打印:
    0
    1
    2
    3
"""
x = 0
while x < 4:
    print(x)
    x += 1  # x = x + 1 的简写

# 用 try/except 块处理异常状况
try:
    # 用 raise 抛出异常
    raise IndexError("This is an index error")
except IndexError as e:
    pass    # pass 是无操作，但是应该在这里处理错误
except (TypeError, NameError):
```

```
    pass    # 可以同时处理不同类的错误
else:   # else 语句是可选的，必须在所有的 except 之后
    print("All good!")   # 只有当 try 运行完没有错误的时候这句才会运行



# Python 提供一个叫做可迭代(iterable)的基本抽象。一个可迭代对象是可以被当作序列
# 的对象。比如说上面 range 返回的对象就是可迭代的。

filled_dict = {"one": 1, "two": 2, "three": 3}
our_iterable = filled_dict.keys()
print(our_iterable) # => dict_keys(['one', 'two', 'three'])，是一个实现可迭代接口的对象

# 可迭代对象可以遍历
for i in our_iterable:
    print(i)    # 打印 one, two, three

# 但是不可以随机访问
our_iterable[1]  # 抛出 TypeError

# 可迭代对象知道怎么生成迭代器
our_iterator = iter(our_iterable)

# 迭代器是一个可以记住遍历的位置的对象
# 用__next__可以取得下一个元素
our_iterator.__next__()  # => "one"

# 再一次调取__next__时会记得位置
our_iterator.__next__()  # => "two"
```

```
our_iterator.__next__()  # => "three"


# 当迭代器所有元素都取出后，会抛出 StopIteration

our_iterator.__next__() # 抛出 StopIteration


# 可以用 list 一次取出迭代器所有的元素

list(filled_dict.keys())  # => Returns ["one", "two", "three"]
```

2.4 函数

```
####################################################
## 4. 函数
####################################################

# 用 def 定义新函数
def add(x, y):
    print("x is {} and y is {}".format(x, y))
    return x + y    # 用 return 语句返回

# 调用函数
add(5, 6)   # => 印出"x is 5 and y is 6"并且返回 11

# 也可以用关键字参数来调用函数
add(y=6, x=5)   # 关键字参数可以用任何顺序


# 我们可以定义一个可变参数函数
def varargs(*args):
    return args

varargs(1, 2, 3)   # => (1, 2, 3)


# 我们也可以定义一个关键字可变参数函数
def keyword_args(**kwargs):
    return kwargs
```

```
# 我们来看看结果是什么：
keyword_args(big="foot", loch="ness")   # => {"big": "foot", "loch":
"ness"}



# 这两种可变参数可以混着用
def all_the_args(*args, **kwargs):
    print(args)
    print(kwargs)
"""
all_the_args(1, 2, a=3, b=4) prints:
    (1, 2)
    {"a": 3, "b": 4}
"""


# 调用可变参数函数时可以做跟上面相反的，用*展开序列，用**展开字典。
args = (1, 2, 3, 4)
kwargs = {"a": 3, "b": 4}
all_the_args(*args)   # 相当于 foo(1, 2, 3, 4)
all_the_args(**kwargs)   # 相当于 foo(a=3, b=4)
all_the_args(*args, **kwargs)   # 相当于 foo(1, 2, 3, 4, a=3, b=4)



# 函数作用域
x = 5

def setX(num):
    # 局部作用域的 x 和全局域的 x 是不同的
    x = num # => 43
    print (x) # => 43
```

```
def setGlobalX(num):
    global x
    print (x) # => 5
    x = num # 现在全局域的 x 被赋值
    print (x) # => 6

setX(43)
setGlobalX(6)

# 函数在 Python 是一等公民
def create_adder(x):
    def adder(y):
        return x + y
    return adder

add_10 = create_adder(10)
add_10(3)   # => 13

# 也有匿名函数
(lambda x: x > 2)(3)   # => True

# 内置的高阶函数
map(add_10, [1, 2, 3])   # => [11, 12, 13]
filter(lambda x: x > 5, [3, 4, 5, 6, 7])   # => [6, 7]

# 用列表推导式可以简化映射和过滤。列表推导式的返回值是另一个列表。
[add_10(i) for i in [1, 2, 3]]  # => [11, 12, 13]
[x for x in [3, 4, 5, 6, 7] if x > 5]   # => [6, 7]
```

2.5 类

```
####################################################
## 5. 类
####################################################


# 定义一个继承 object 的类
class Human(object):

    # 类属性，被所有此类的实例共用。
    species = "H. sapiens"

    # 构造方法，当实例被初始化时被调用。注意名字前后的双下划线，这是表明这个属
    # 性或方法对 Python 有特殊意义，但是允许用户自行定义。你自己取名时不应该用
这
    # 种格式。
    def __init__(self, name):
        # Assign the argument to the instance's name attribute
        self.name = name

    # 实例方法，第一个参数总是 self，就是这个实例对象
    def say(self, msg):
        return "{name}: {message}".format(name=self.name, message=msg)

    # 类方法，被所有此类的实例共用。第一个参数是这个类对象。
    @classmethod
    def get_species(cls):
        return cls.species
```

```
    # 静态方法。调用时没有实例或类的绑定。
    @staticmethod
    def grunt():
        return "*grunt*"


# 构造一个实例
i = Human(name="Ian")
print(i.say("hi"))     # 印出 "Ian: hi"

j = Human("Joel")
print(j.say("hello"))  # 印出 "Joel: hello"

# 调用一个类方法
i.get_species()   # => "H. sapiens"

# 改一个共用的类属性
Human.species = "H. neanderthalensis"
i.get_species()   # => "H. neanderthalensis"
j.get_species()   # => "H. neanderthalensis"

# 调用静态方法
Human.grunt()   # => "*grunt*"
```

2.6 模块

```
####################################################
## 6. 模块
####################################################

# 用 import 导入模块
import math
print(math.sqrt(16))  # => 4.0

# 也可以从模块中导入个别值
from math import ceil, floor
print(ceil(3.7))  # => 4.0
print(floor(3.7))   # => 3.0

# 可以导入一个模块中所有值
# 警告：不建议这么做
from math import *

# 如此缩写模块名字
import math as m
math.sqrt(16) == m.sqrt(16)   # => True

# Python 模块其实就是普通的 Python 文件。你可以自己写，然后导入，
# 模块的名字就是文件的名字。

# 你可以这样列出一个模块里所有的值
import math
dir(math)
```

2.7 高级用法

```
########################################################
## 7. 高级用法
########################################################

# 用生成器(generators)方便地写惰性运算
def double_numbers(iterable):
    for i in iterable:
        yield i + i

# 生成器只有在需要时才计算下一个值。它们每一次循环只生成一个值，而不是把所有的
# 值全部算好。
#
# range 的返回值也是一个生成器，不然一个 1 到 900000000 的列表会花很多时间和内
存。
#
# 如果你想用一个 Python 的关键字当作变量名，可以加一个下划线来区分。
range_ = range(1, 900000000)
# 当找到一个 >=30 的结果就会停
# 这意味着 `double_numbers` 不会生成大于 30 的数。
for i in double_numbers(range_):
    print(i)
    if i >= 30:
        break


# 装饰器(decorators)
# 这个例子中，beg 装饰 say
```

```
# beg 会先调用 say。如果返回的 say_please 为真，beg 会改变返回的字符串。
from functools import wraps



def beg(target_function):
    @wraps(target_function)
    def wrapper(*args, **kwargs):
        msg, say_please = target_function(*args, **kwargs)
        if say_please:
            return "{} {}".format(msg, "Please! I am poor :(")
        return msg

    return wrapper



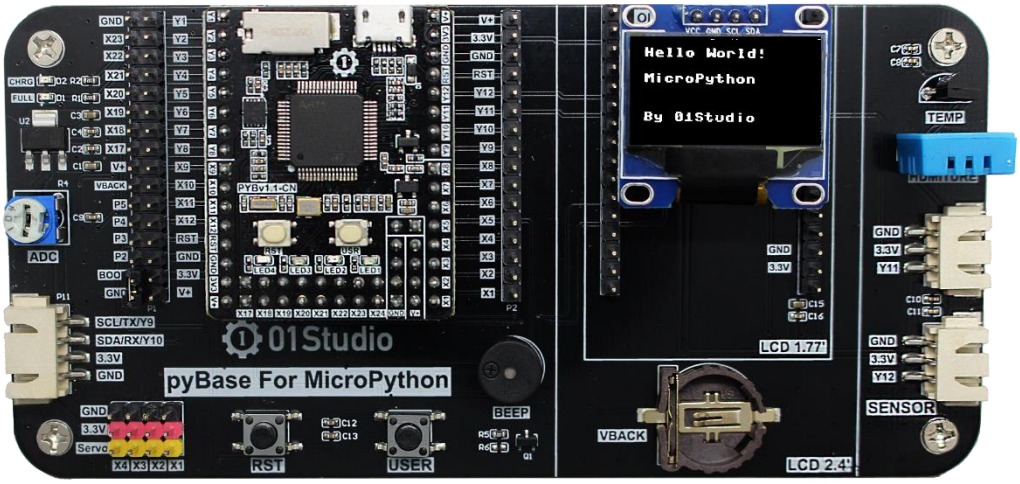
@beg
def say(say_please=False):
    msg = "Can you buy me a beer?"
    return msg, say_please



print(say())  # Can you buy me a beer?
print(say(say_please=True))  # Can you buy me a beer? Please! I am
poor :(
```

第二部分　基于 STM32 平台

我们以往做嵌入式开发的时候，一般都会用到 IDE 开发软件，比如 KEIL、IAR 等，有些还需要专门的烧录器，这为嵌入式开发增加了门槛。而使用 MicroPython 的话，用户只需要 1 根 MciroUSB 数据线（俗称的安卓手机数据线）就可以开发了。pyBoard 上使用了 STM32 自身 flash 的一定空间作为 U 盘跟电脑交互，所有文件存放在这个几十 KB 空间的里，这使得我们在不同的 PC 平台，Windows/Mac/Linux 上可以直接打开 U 盘文件来开发，你甚至可以用自带的文档编辑器来编写你的程序。我们将会对 Windows、MAC、Linux 不同开发环境的搭建进行讲解。



pyBoard 开发套件

第3章　开发环境快速建立

3.1　基于 Windows

Windows 是大部分开发人员的主流平台，毕竟微软凭借其 windows 操作系统在 PC 时代统治了几十年，大部分软件厂商还是愿意针对 windows 来开发桌面软件，但这也跟乔布斯英年早逝有关。扯远了，所以为了后面省事，我们还是推荐使用 Windows 作为首选的开发环境，推荐使用 Windows 10, 微软官方已经宣布停止对 win 7 更新和维护了，win10 功能强大，有些驱动还能自动联网安装，免去人工安装的麻烦。

3.1.1　安装 python3

说明：我们提供安装 python3 教程只是为了在 PC 上体验一下 python3 安装和使用过程，由于 micropython 固件集成了 python 库，所以电脑并非一定要安装，本节可以跳过。

python3 并不是 python2 的升级版，而是 python2 的一个分支，它们都发展了自己独特的功能和函数，市面上有些产品开发时候甚至不相互兼容，总体来说初学者使用 python3 较多。具体安装方法如下：

在 MicroPython 开发套件配套资料\开发工具\Windows\python3 安装包下找到 python-3.7.3-amd64.exe 文件打开（这个文件是针对 64bit 操作系统的用户，32bit 系统的用户自行到 python 官网下载相关版本安装：www.python.org ）。

打开后一直按下一步安装即可。安装路径不能有中文，推荐默认路径安装。安装过程中务必勾上 Add to PATH 选项。

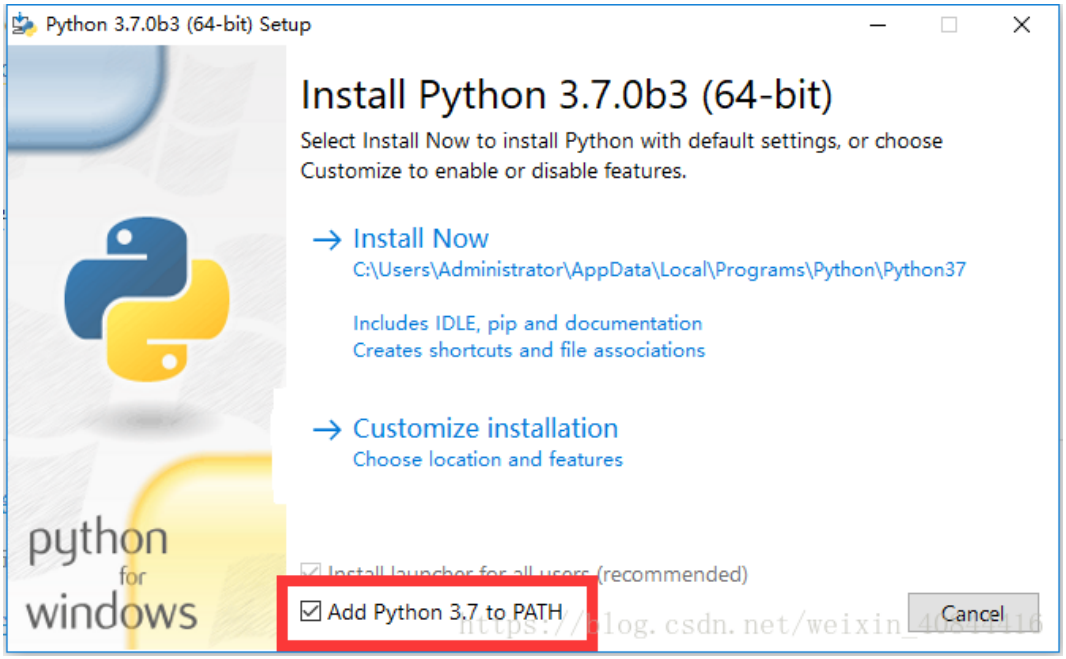


图 3-1 务必勾上 Add to PATH

安装完成后点击计算机的运行（快捷键是 windows+R），输入 cmd，弹出命令框；

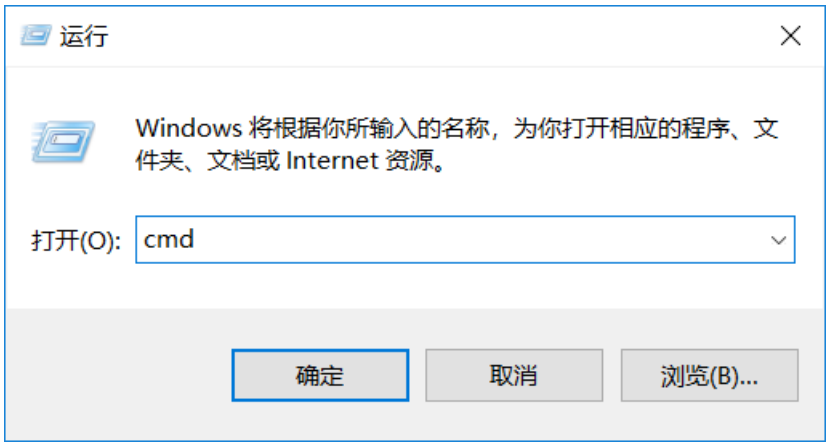


图 3-2 运行输入 cmd 进入命令

输入 python 按回车，出现 Python 3 版本信息即安装成功。

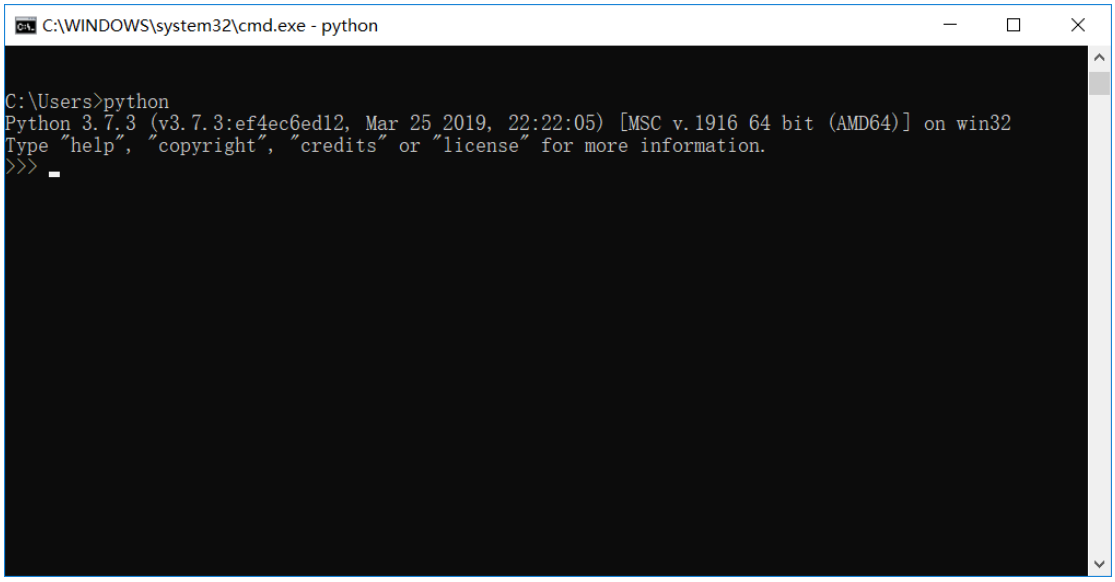


图 3-3 python 成功安装信息

此时已经进入 python 自带的编译器，我们在>>>后面输入 print(“Hello World!”)按回车，可见系统打印出来 Hello World! 需要退出的话直接关闭窗口即可。

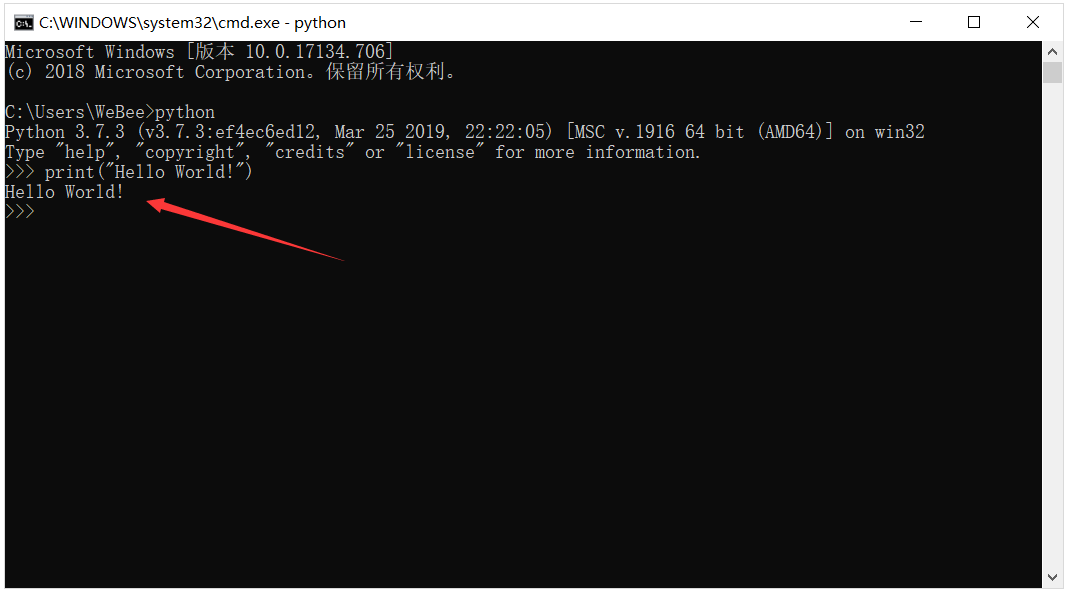


图 3-4 打印 Hello World 操作

至此，电脑已经成功安装 python3。

3.1.2 安装开发软件 Mu

开发软件是指我们用来写代码的工具，上一节那个命令框里面其实就是调用了自带的编程器，但是我想没有人希望一直在那黑框里面撸代码。 Python 拥有众多的编程器，如果你之前已经熟练掌握 python 或已经使用 python 开发，那么可以直接使用你原来习惯的开发软件来编程。如果你是初学者或者喜欢简单而快速应用，我们推荐使用 Mu。

Mu 是一款开源软件，以极简方式设计，对 MicroPython 的兼容性非常友善。而且支持 Windows、Mac OS、Linux、树莓派。由于开源，所以软件迭代速度非常快，功能日趋成熟。具体安装方法如下：

在 https://codewith.mu/en/download 下载地址，选择自己的开发平台进行下载安装即可：

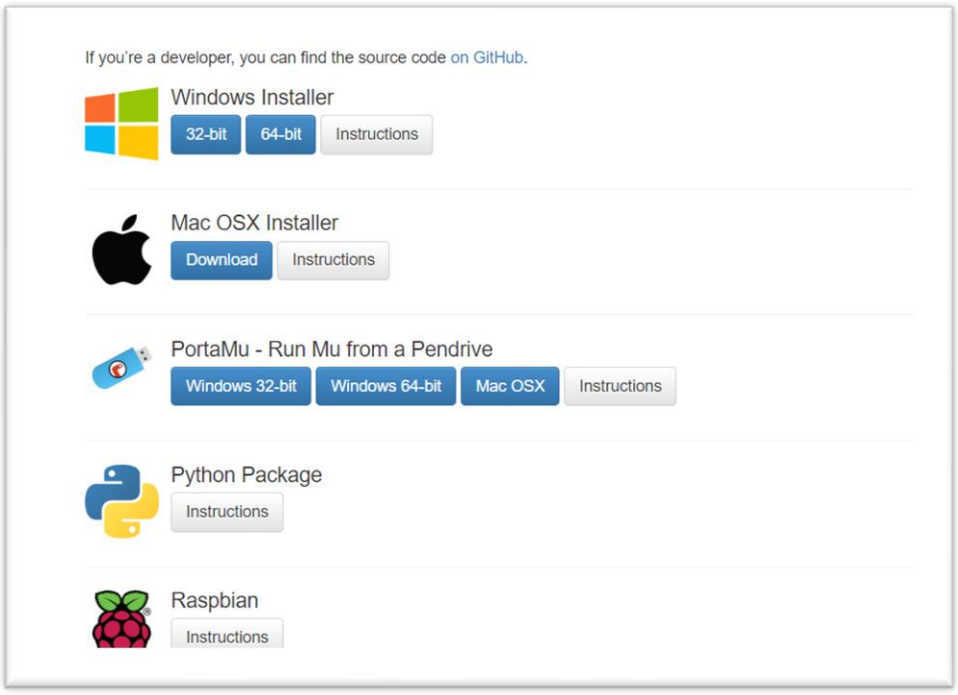


图 3-5 稳定版下载

由于 Mu 更新非常快，因此社区也同时提供最新版下载，给愿意尝鲜和给予软件反馈的用户，下载位置如下：

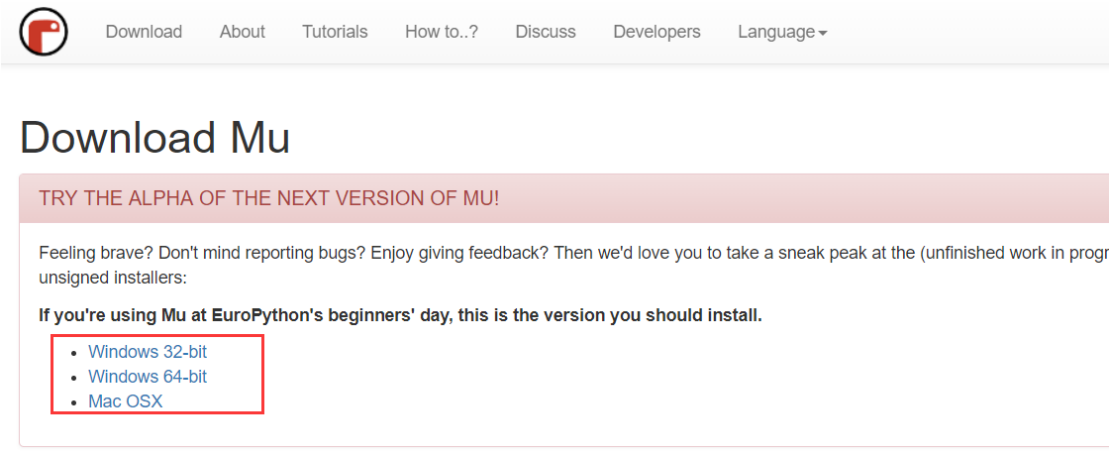


图 3-6 尝鲜版下载

我们推荐使用尝鲜版，能使用最新的功能。下载安装后，打开 Mu，可以看你的界面非常简洁：

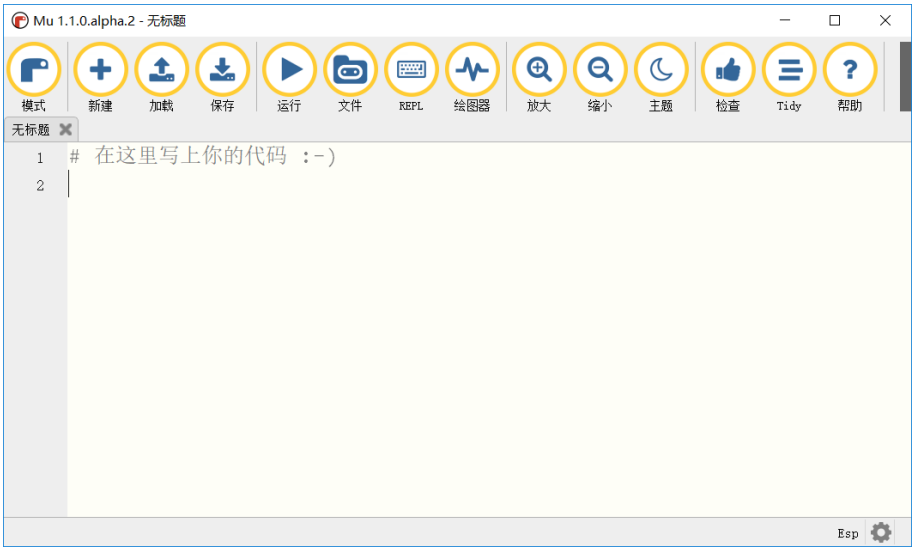


图 3-7 Mu 主界面

点击左上角“模式”，弹出对话框，可以看到支持 MicroBit、ESP 等一系列 MicroPython 产品。基于 pyBoard（STM32）系列产品选择【CircuitPython】（micropython）模式进行开发。

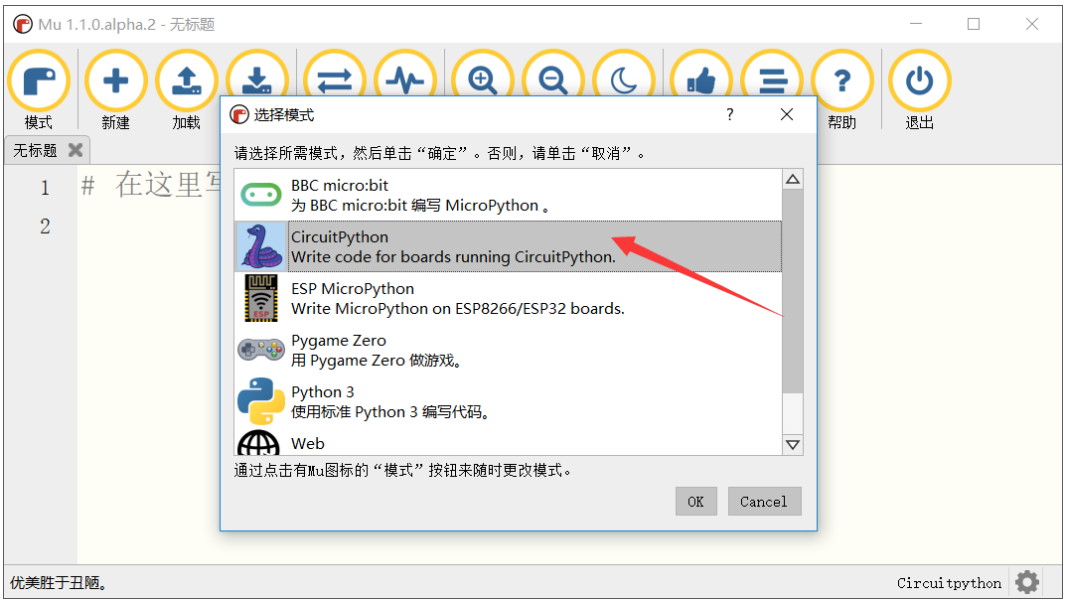


图 3-8 选择 CircuitPython

如果出现以下界面，直接点 OK 忽略即可。

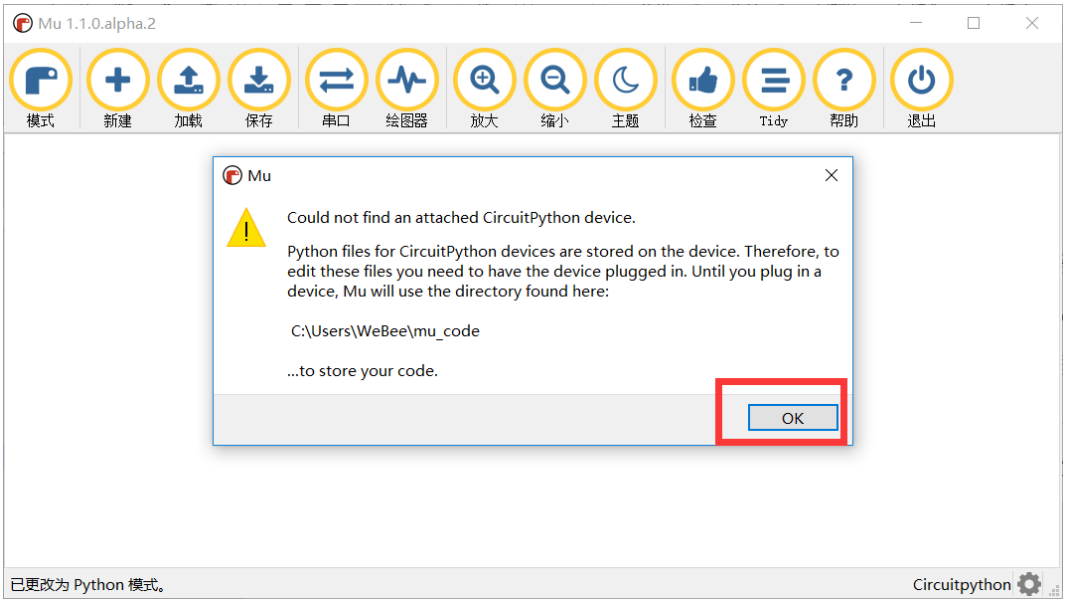


图 3-9 点击 OK 忽略提示

至此，Mu 安装完成。

3.1.3 开发套件使用

3.1.3.1 驱动安装

我们将 MicroPython 开发板通过 MicroUSB 数据线连接到电脑：

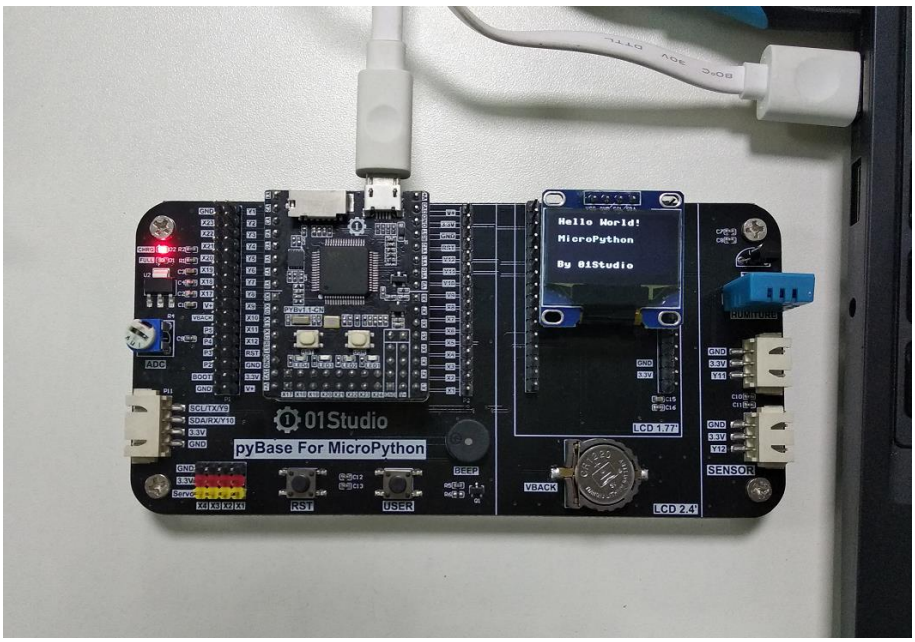


图 3-10 通过 MicroUSB 线连接到电脑

连接电脑后，会发现出现一个 U 盘，名字是 PYBFLASH，点击打开，看到以下文件。

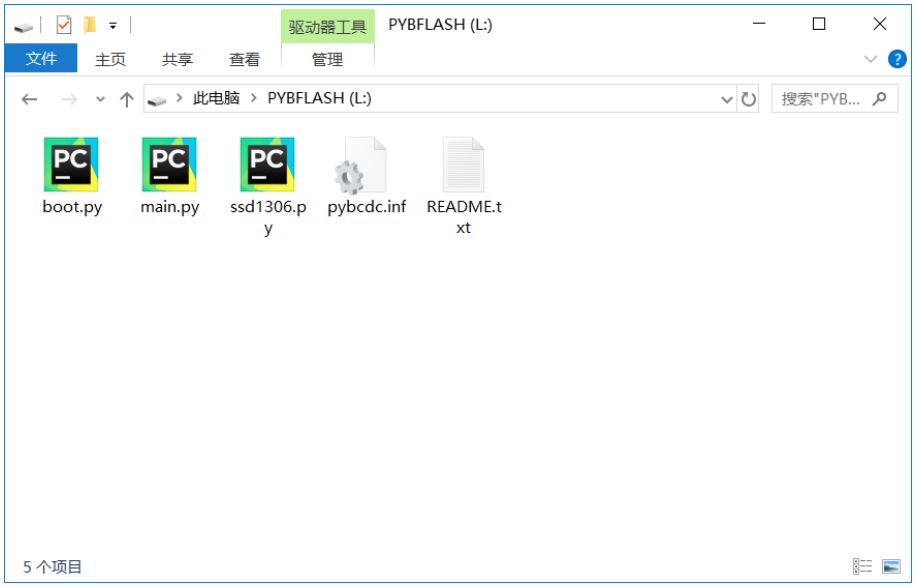


图 3-11 pyBoard 的系统文件

下面是对各个文件的简介，主要是前面 4 个文件构成 MicroPyhon 的文件系统：

1. **Boot.py：系统启动文件；**
2. **Main.py: 主函数代码文件；**
3. **Pybcdc.inf: USB 串口调试（REPL）驱动；**
4. **README.txt: 说明文件；**
5. ssd1306.py:这个是我们出厂配套的程序代码，用于 OLED 驱动。在后面章节会详细介绍，没有这个文件也没关系。

我们还需要安装一个 USB 转串口的驱动， 鼠标右键点击 “我的电脑”—属性—设备管理器：系统找到了 pyBoard 虚拟串口新设备（**部分 PC 可能不显示 pyboard，只要找到 COM 即可，不确定的可以拔插 pyboard，找到出现和消失的 COM 就是 pyboard 的串口 COM**），驱动在 U 盘里，我们安装一下。

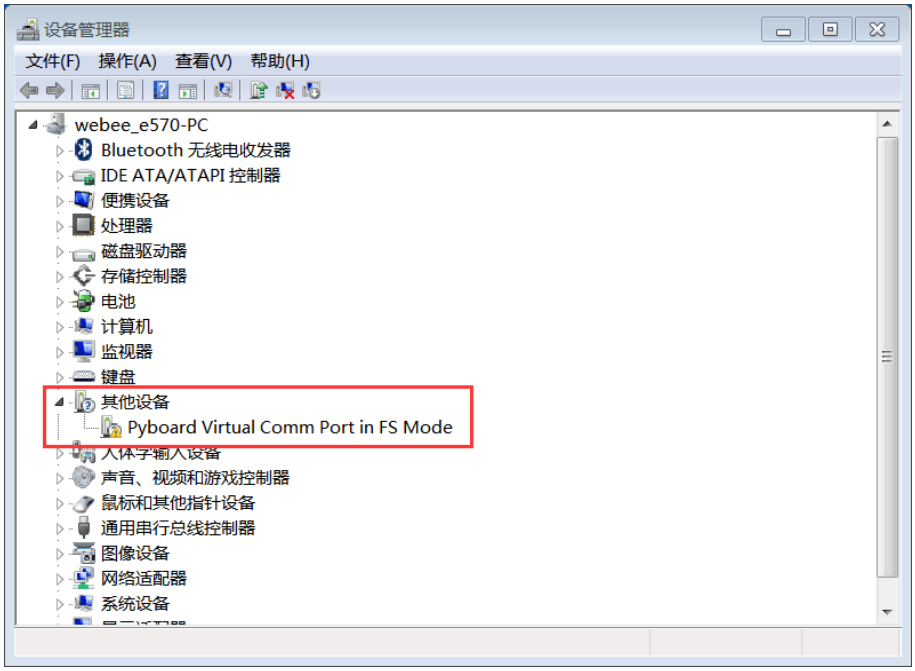


图 3-12 找到对应驱动

点击右键更新驱动软件：

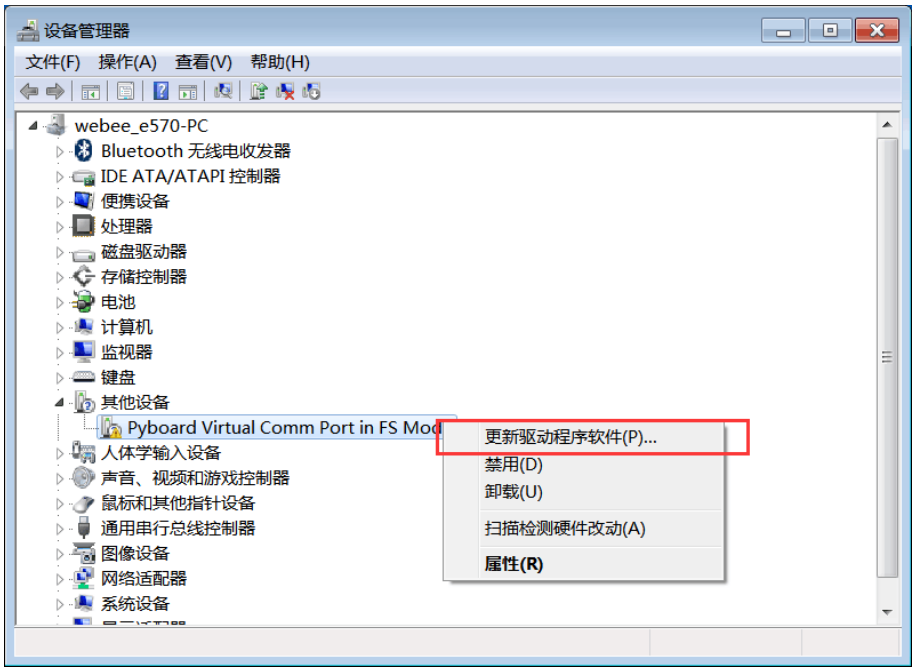


图 3-13 点击右键更新驱动

浏览计算机安装，选择路径是 PYBFLASH，也就是 U 盘盘符。

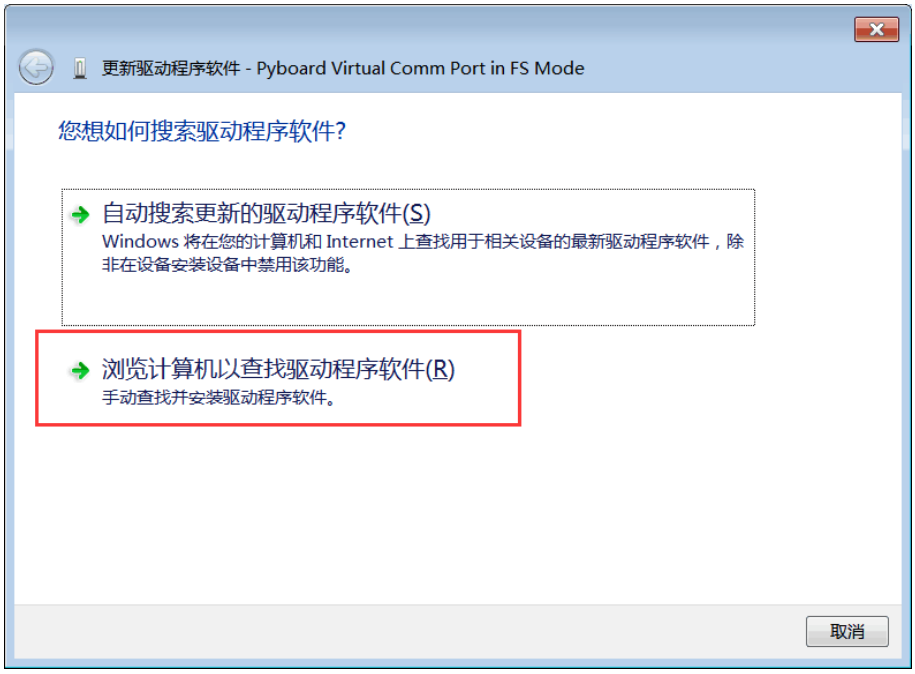


图 3-14

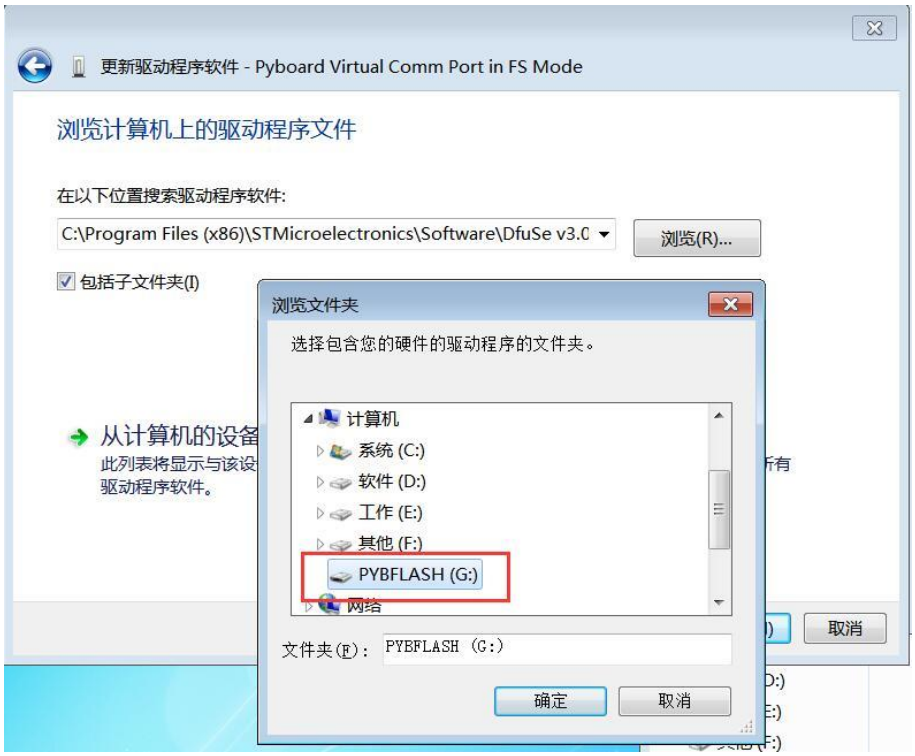


图 3-15 选择 PYBFLASH 盘符

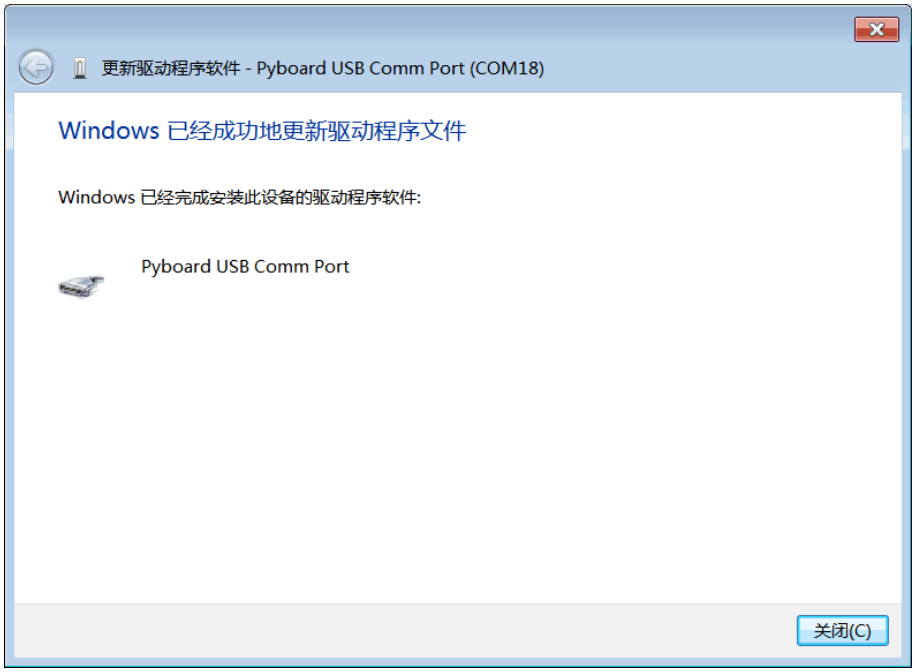


图 3-16 安装完成

安装完成后，我们看到，原来的叹号消失，出现了一个 COM18 串口号。这个串口号不同的电脑会显示不同，以后还会用到。

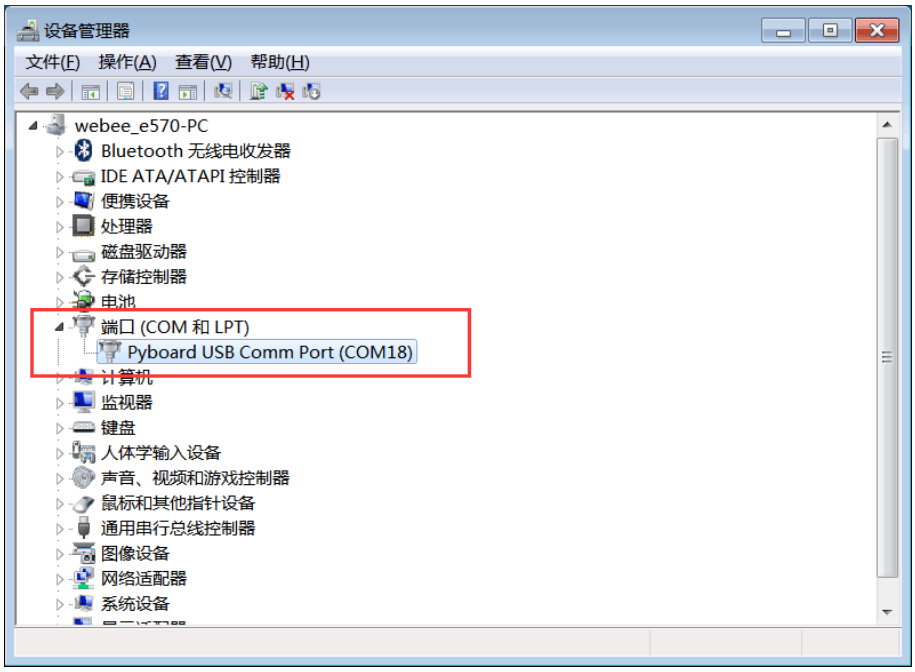


图 3-17 出现串口 COM 信息

注意事项：

部分用户的 Win7 系统由于使用的是 Ghost 版系统，Ghost 系统精简了很多系统文件，有可能导致驱动无法安装。我们不建议开发者安装 Ghost 的系统。这时候建议用户采取以下措施。

（1）使用 360 的系统重装功能，实测重装后驱动可以安装；

（2）升级成 Win10 系统。

我们更推荐方法（2），因为微软已经宣布停止对 Win7 进行更新，另外就是我们的教程都会基于 Win10 系统。

3.1.3.2 第一个实验

接下来我们使用最简单的方式来做一个点亮 LED4 蓝色灯的实验，你或许会惊喜的发现，只要两句代码就点亮了 LED 灯了，这就是 MicroPython 的神奇之处。大家暂时先不用理解代码意思，后面章节会有解释。这里主要是为了让大家了解一下 MicroPython 编程和下载程序的方法和原理。具体如下：

在 Mu 软件上方点击【加载】按钮，打开 U 盘里面的 main.py 文件，将代码修改成以下内容：

```
import pyb

pyb.LED(4).on()
```

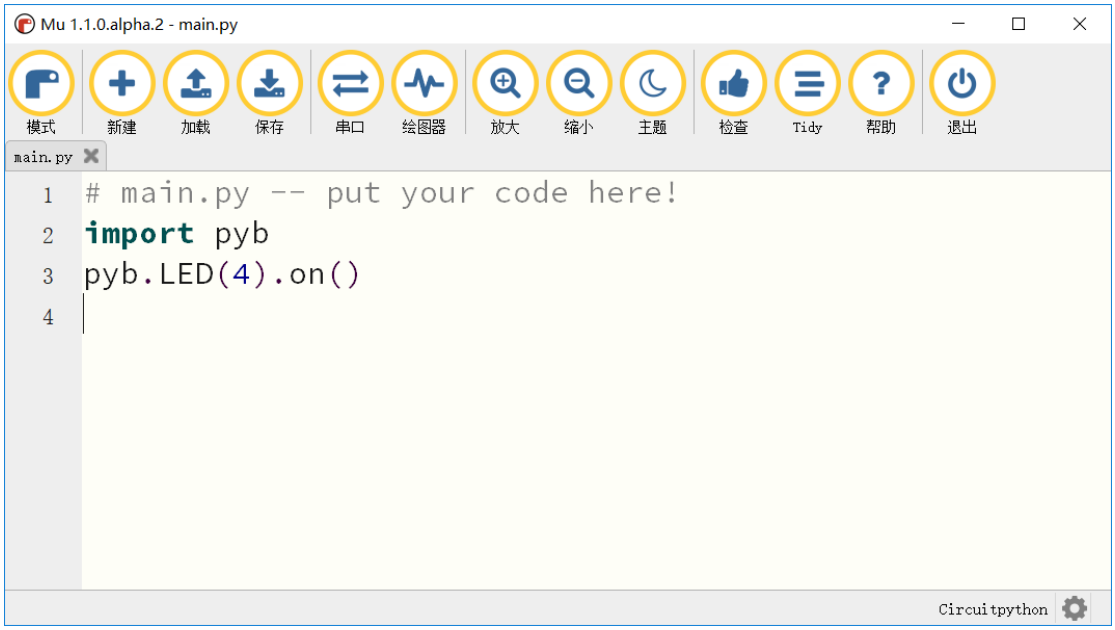


图 3-18 输入代码

按 ctrl + s 或点击 Mu 上方【保存】按钮保存文件，可以看到开发套件上的 LED1 红灯亮一会儿然后灭掉，这个灯的功能是检测到 U 盘文件发生变化，重新加载的过程。在 LED1 红灯熄灭前请勿进行复位或其他相关操作。

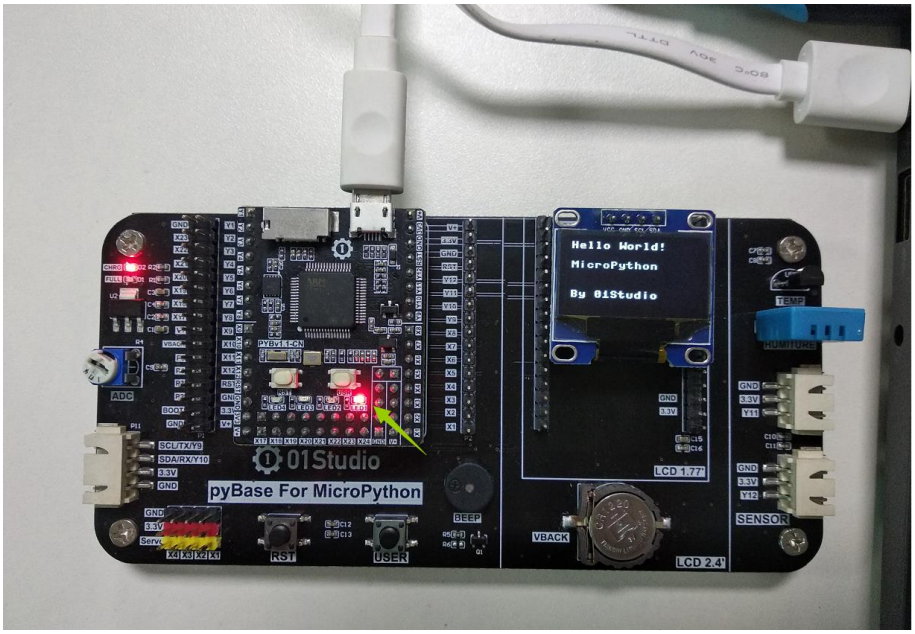


图 3-19 程序烧写过程红灯亮

待红灯灭掉后，说明程序已经烧写完成。按下 pyboard 或者 pyBase 上的 RST 复位按键，可见程序跑起来了，LED4 蓝灯被点亮。

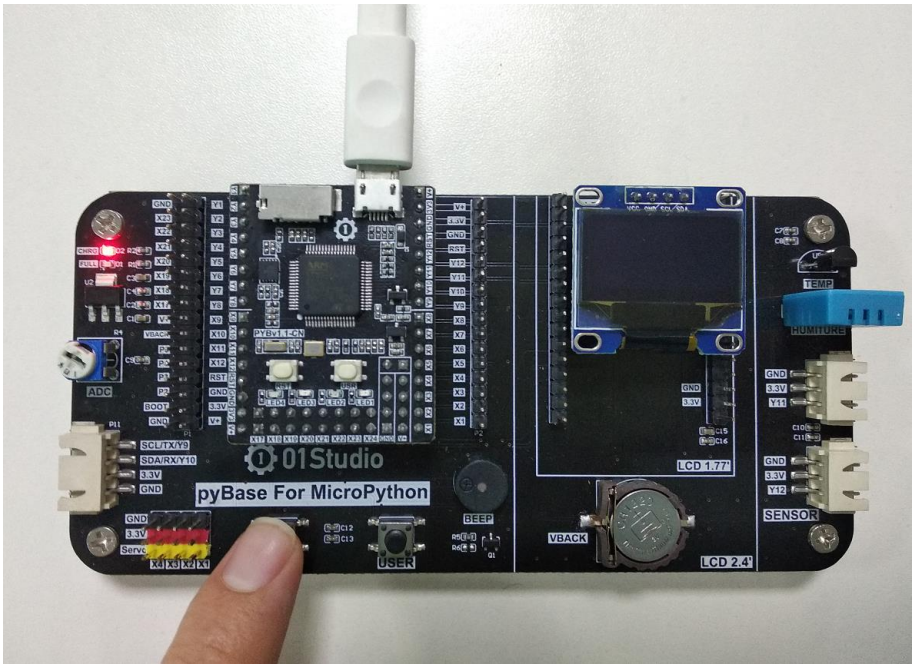


图 3-20 按下复位键

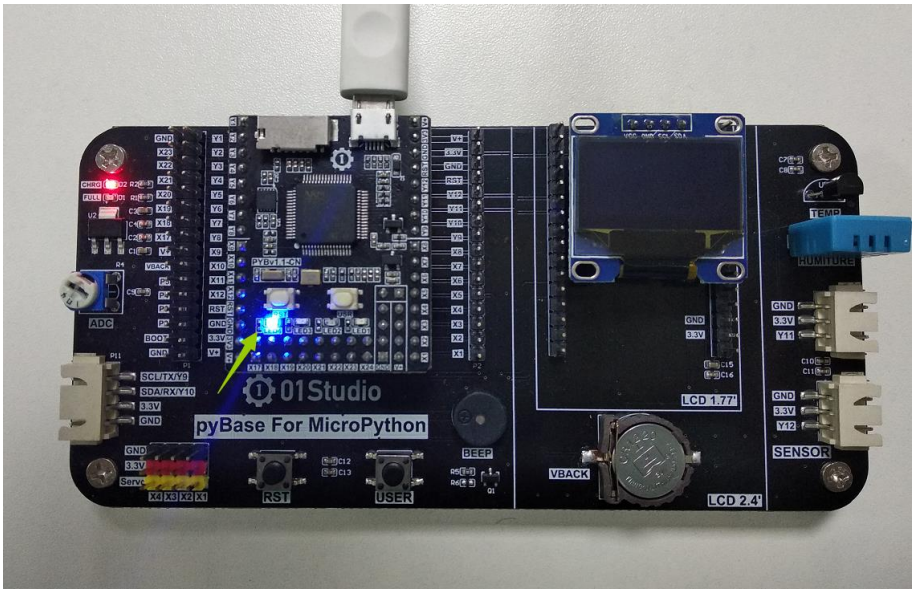


图 3-21 程序运行正常

总结：由以上简单的实验操作我们可以见到 micropython 最常见的编程方式，那就是直接修改 main.py 文件后保存，等待开发板烧录完成（LED1 红灯由亮到灭）后按下复位键，程序即正常运行。

3.1.3.3 REPL 串口交互调试

Pyboard 集成了交互解释器 REPL 【读取(Read)-运算(Eval)-输出(Print)-循环(Loop)】，开发者可以直接通过串口终端来调试 pyboard 或 micropython 开发套件。我们使用的软件是一款免费的串口终端软件 putty。

将开发板连接到电脑，从我的电脑—属性—设备管理器中找到当前的串口号，这里是 COM4。

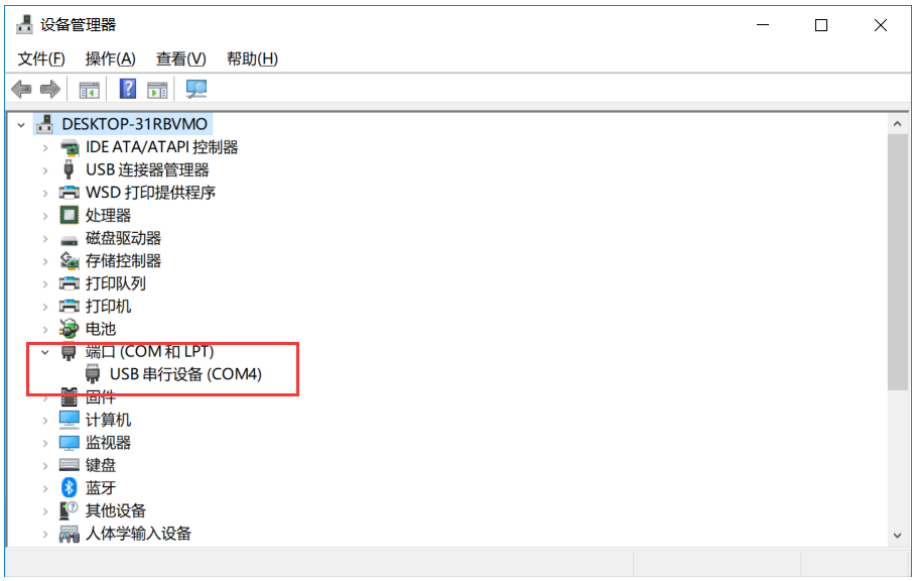


图 3-22 找到串口号

打开 MicroPython 开发套件配套资料\开发工具\串口终端工具\Putty.exe，选择左下角 Serial，配置信息如下：

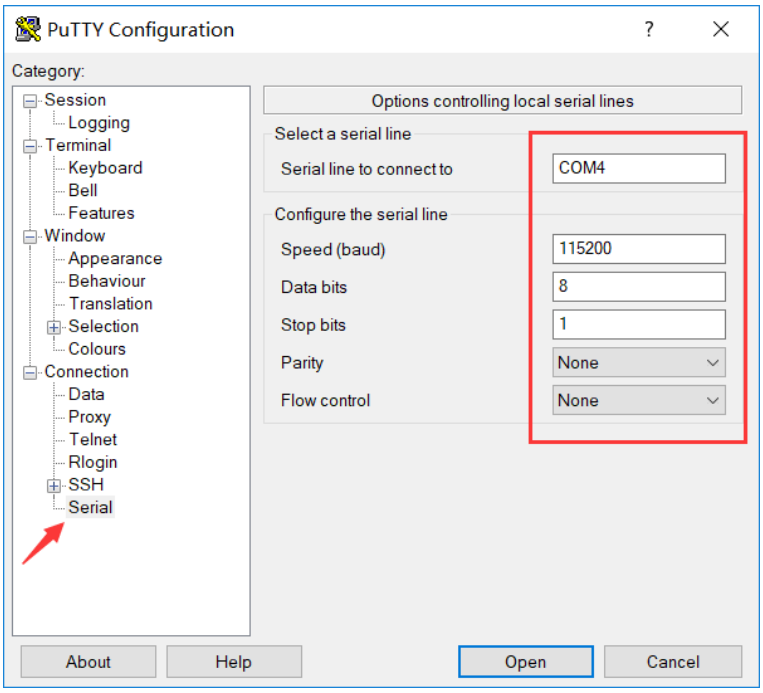


图 3-23 配置串行设备参数

配置好后不是点 open，而是点左边上方 Session，选择 Serial 后可看到刚刚的配置信息。串口号通常不会变化，我们在 Save Session 下方输入 COM4 或者自己喜欢的名称，点右边 Save，在空白框里面就出现 COM4 字样，以后可以直接使用。设置好后我们点击 Open。

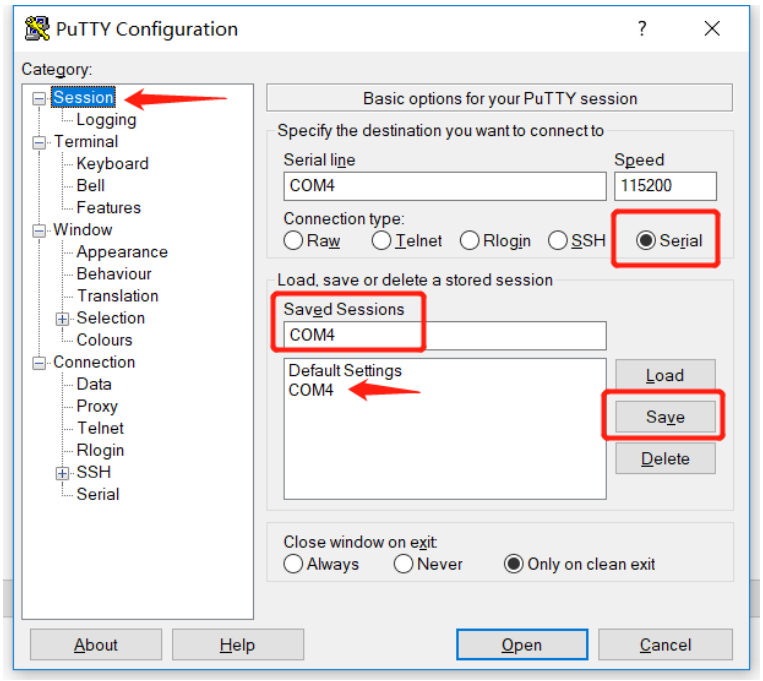


图 3-24

出现下面对话框，是不是似曾相识，没错，跟我们之前在计算机命令行执行 python 指令后出现类似的。上面主要是当前设备固件的版本号。

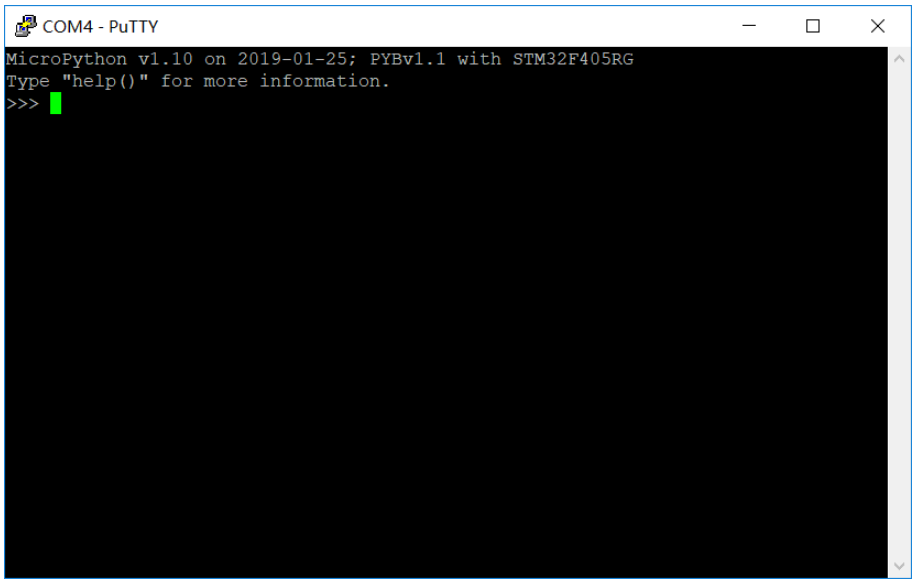


图 3-25

现在对话框相当于连接上了开发板上，由于 pyboard 集成了解析器。我们在这里可以进行调试和简单编程，接下来我们测试一下。在对话框输入下面代码，按回车，可以看到代码运行情况。

```
print(“Hello 01Studio!”)
```

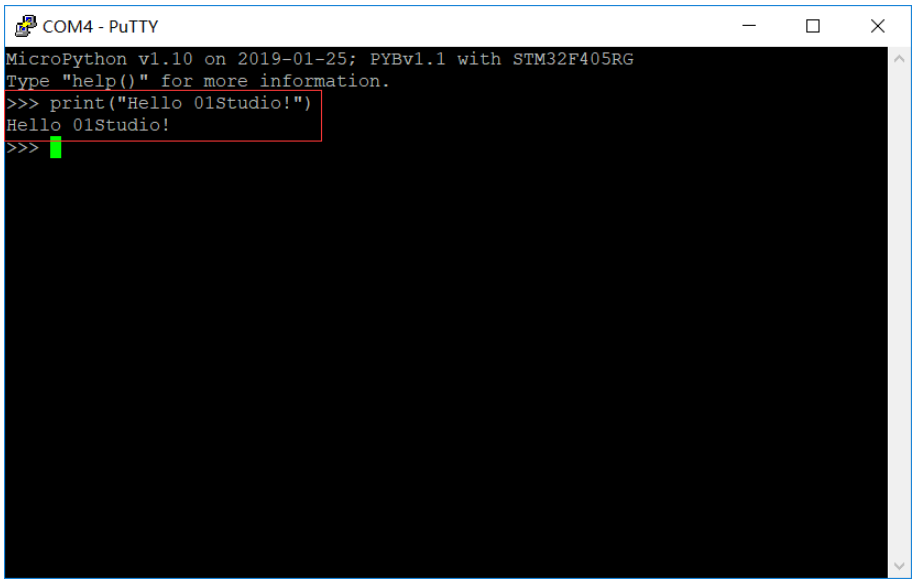


图 3-26

我们再输入 1+1 按回车，得到了计算结果 2。

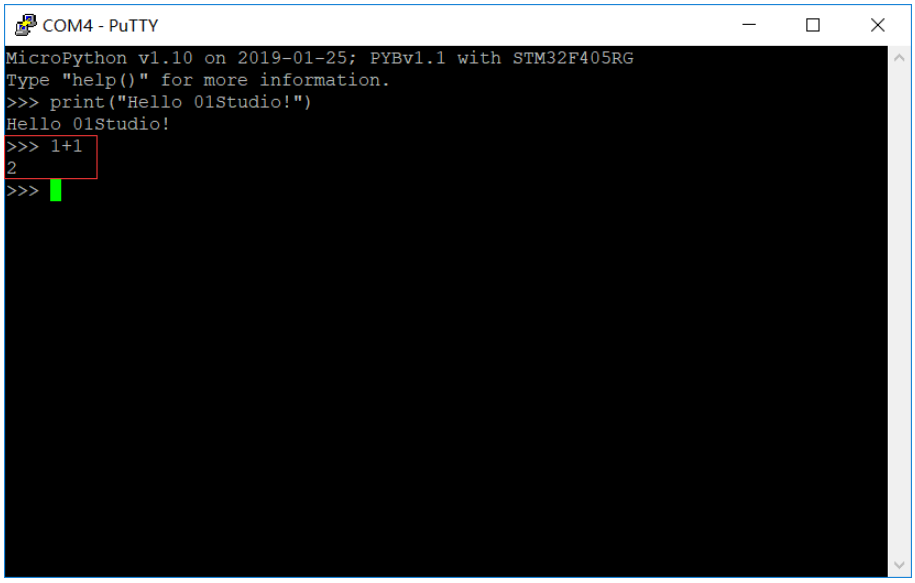


图 3-27

最后我们测试一下在 REPL 对话框上调试 MicroPython 开发板，这里先做测试，代码含义会在后面章节解释。

先输入 import pyb，按回车；再输入 pyb.LED(3).on()，按回车；可以看到开发板上 LED3 黄灯被点亮。

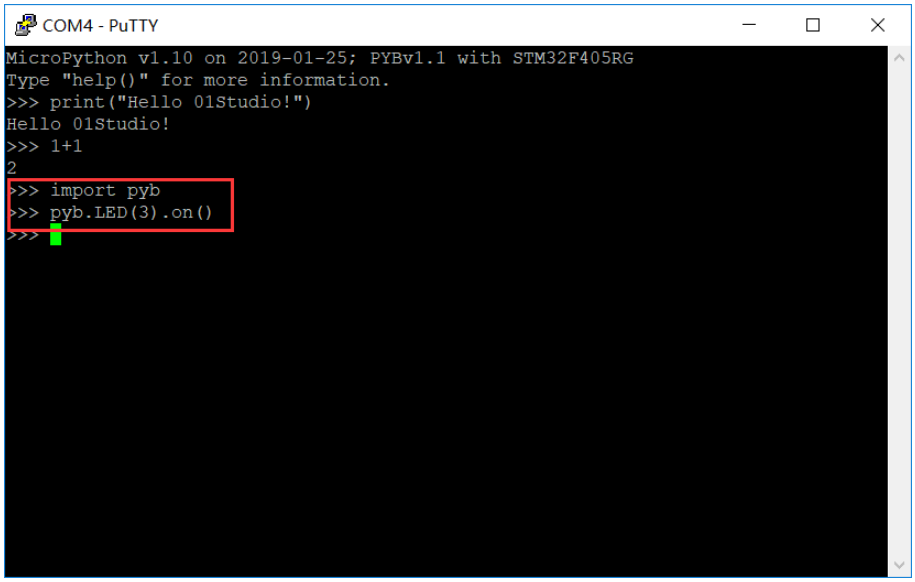


图 3-28

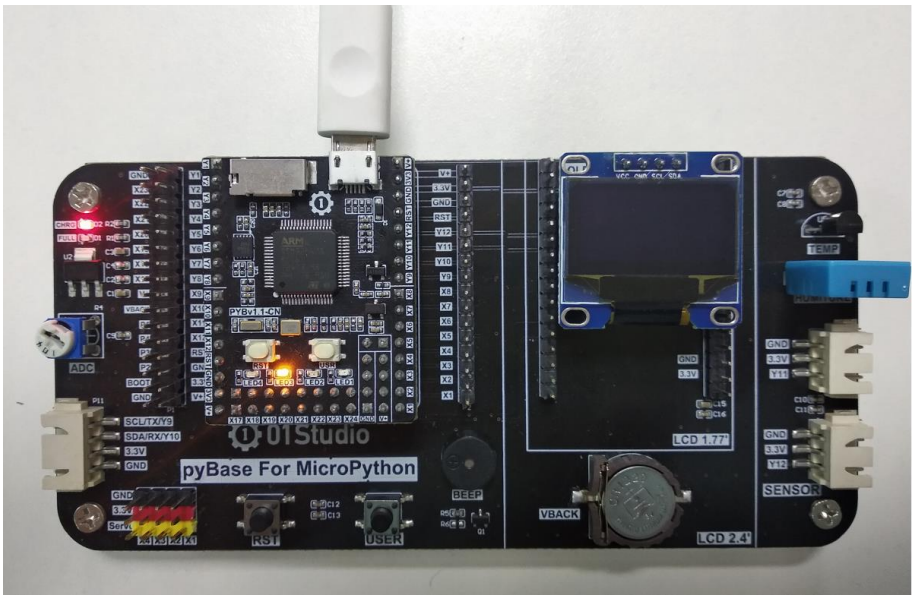


图 3-29 点亮 LED3-黄灯

温馨提示：putty 在 windows 操作系统下，当按下 pyboard 复位键的时候，由于系统重启过程中虚拟串口消失，会导致 putty 会死掉，需要重新开启。所以我们可以使用【Ctrl+D】组合键方式软件复位 pyBoard 设备。如下图所示：

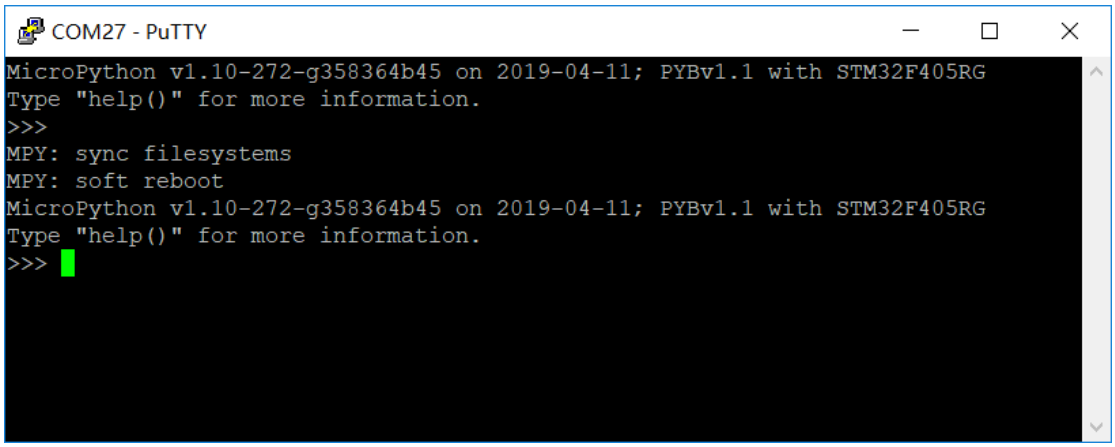


图 3-30 软件复位（soft reboot）

3.1.3.4 串口助手-超级终端

putty 在 windows 操作系统下，当按下 pyboard 复位键的时候，由于系统重启过程中虚拟串口消失，会导致 putty 会死掉，需要重新开启。（putty 是一个开源软件，我们只能默默期待后期的改进。）

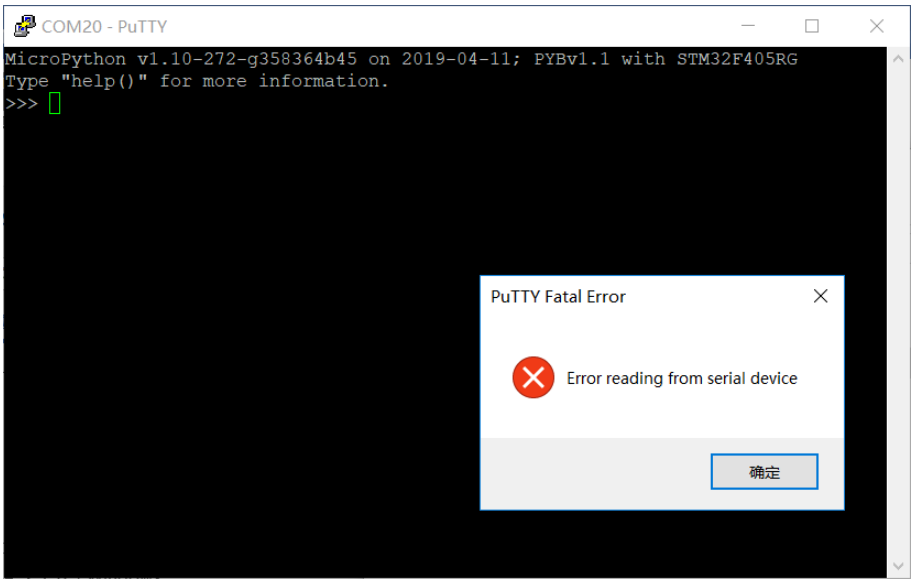


图 3-31 putty 软件挂掉

这时候可以使用 Windows 的自带串口工具，超级终端。早期使用过 Windows XP 的朋友可能知道超级终端这玩意，但在 Win7、Win10 系统已经已经被微软阉割了。不过我们可以在网上下载安装。在我们提供的 MicroPython 开发套件配套资料\开发工具\Windows\串口终端工具 文件夹下找到超级终端软件并打开：

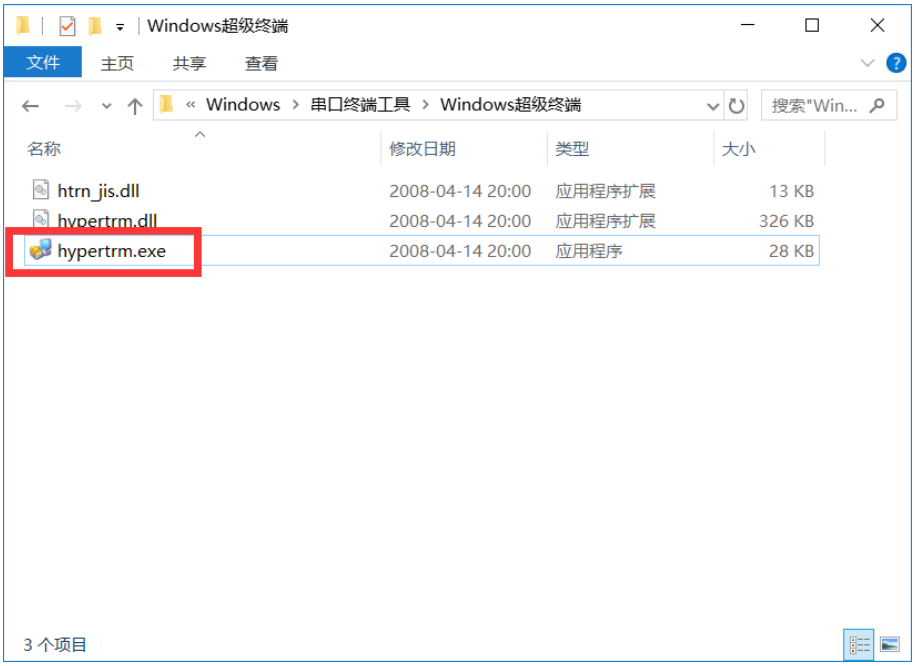


图 3-32 超级终端

打开后根据自己喜好设置名称，这里用“COM”，点“确定”。

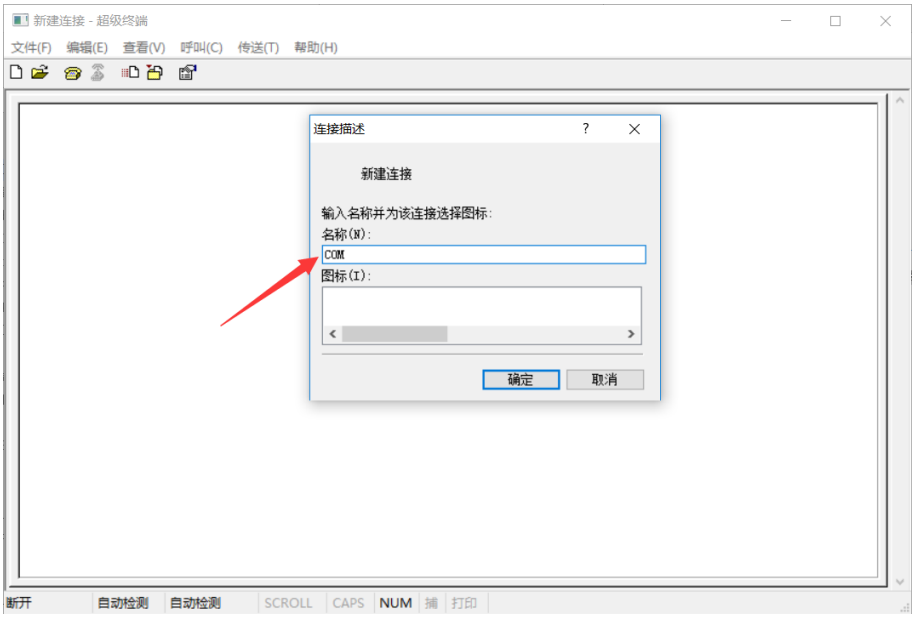


图 3-33 设置名称

点击“我的电脑”右键--属性--设备管理器中查看一下 COM 的端口号，这里是 COM20。

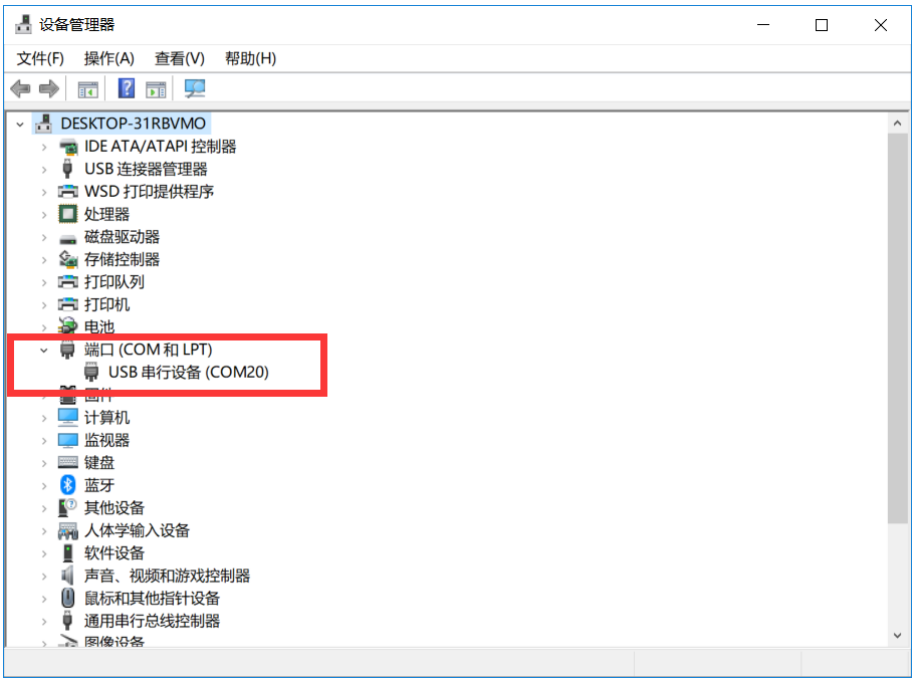


图 3-34 COM 端口号

回到超级终端，发现已经自动识别了 COM20，选择并确认。

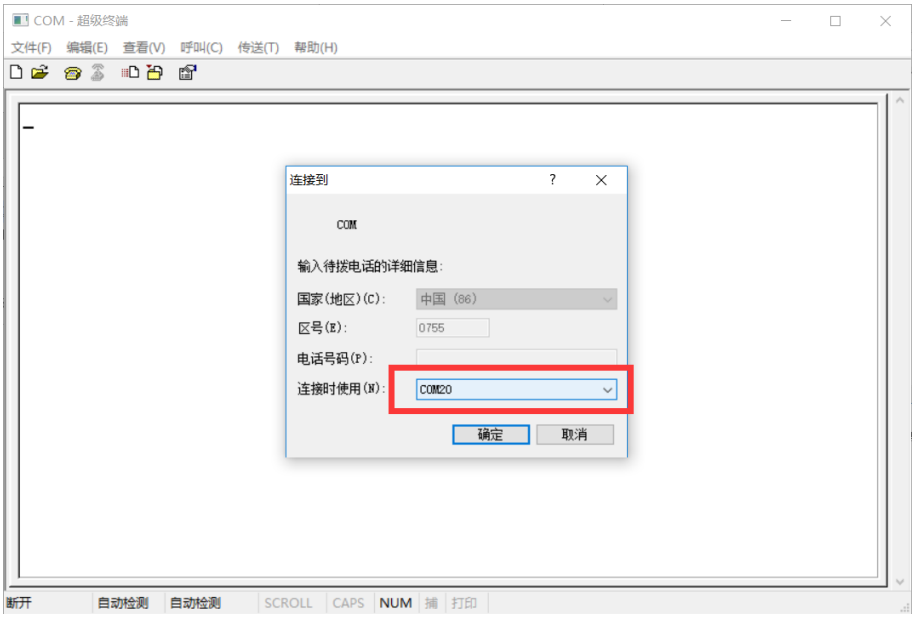


图 3-35 选择 COM

配置串口参数，注意数据流控选择“无”。点击“确认”：

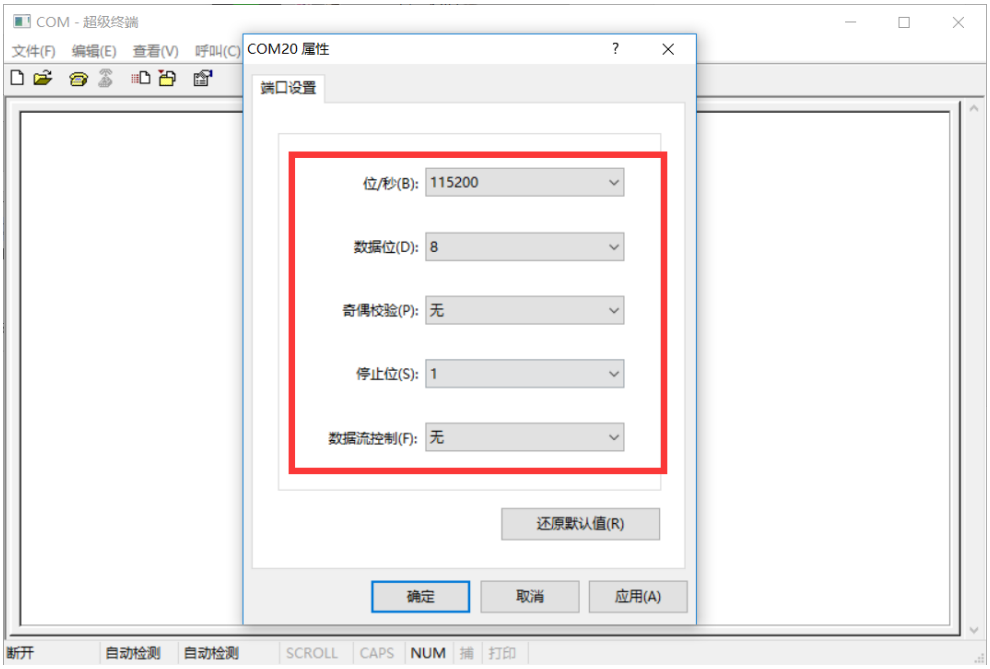


图 3-36 配置串口参数

点击确认后可以看到 pyboard 信息，成功进入 REPL 交互模式：

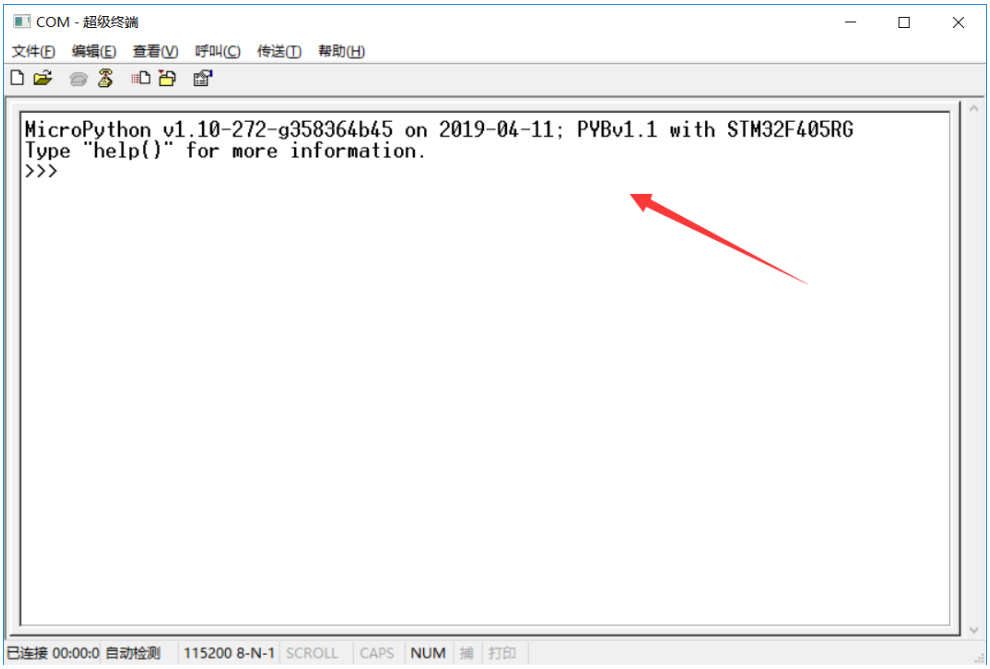


图 3-37 进入 REPL 交互

每次复位 pyboard 的后，软件不会崩溃，先点击“断开”，再点击“连接”即可重新进入 REPL 交互模式：

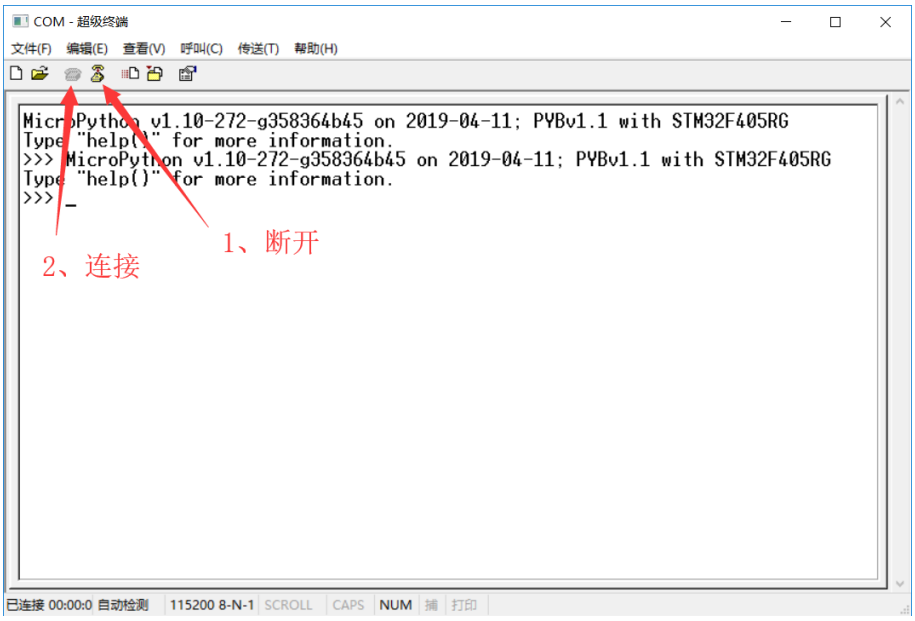


图 3-38

超级终端支持串口参数保存功能，可以通过保存当前信息，下次使用直接打开即可，无须重新配置。

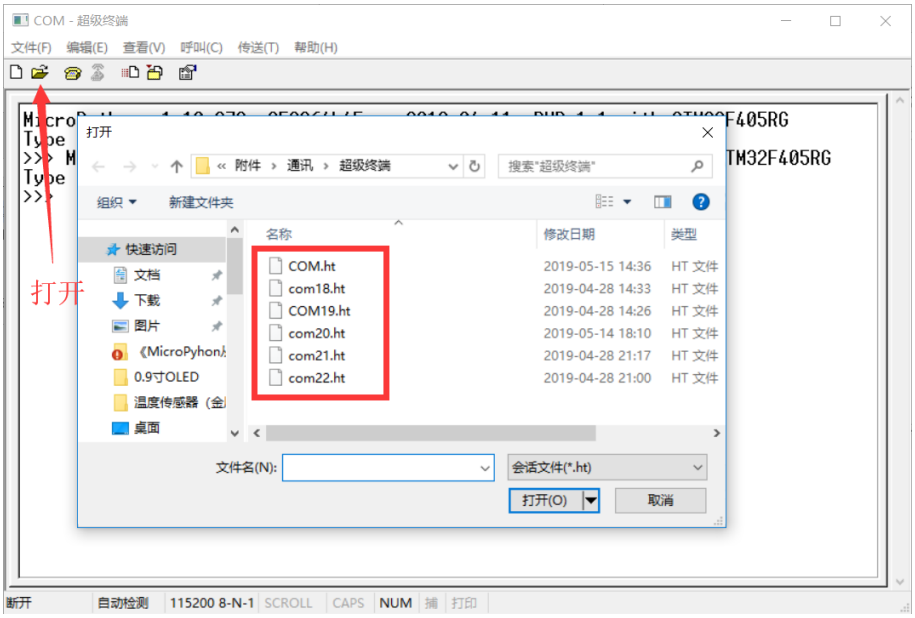


图 3-39

3.1.3.5 恢复出厂设置

当 pyboard 出现异常时，可以通过以下方式修改启动顺序或者恢复出厂设置，方法如下：

按着 pyBoard 上的 USER 键不放，再按一下 RST 键，LED 灯会持续交替闪烁，当闪烁达到你想要的模式时候，松开 user 键，LED 灯会快速闪烁，板子会接着重新启动。

模式 1：只有绿灯亮，正常模式：先启动``boot.py``然后``main.py``。

模式 2：只有橙色灯亮，安全模式：启动时候不运行任何脚本。(仅 1 次有效)

模式 3：绿灯和橙灯同时亮，文件系统重置：文件系统恢复出厂状态，然后以安全模式启动。

如果你的文件系统损坏了，可以通过模式 3 启动修复它。

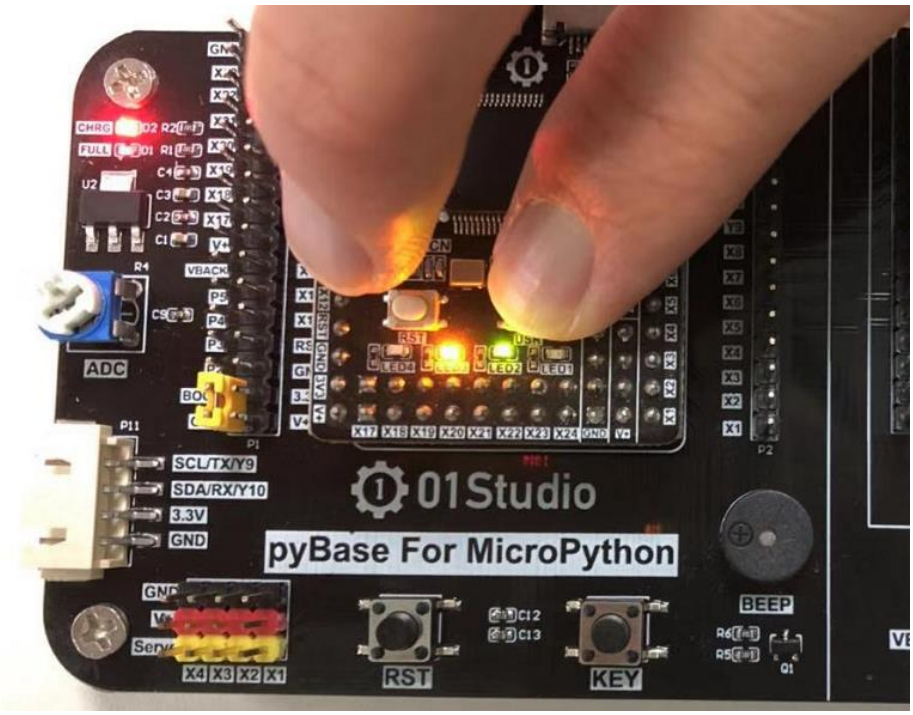


图 3-40 模式 3 启动

重新启动后红色 LED 灯会亮一会儿，待其熄灭后再使用。这时候打开 U 盘文件系统，可见是剩下 4 个原始文件：

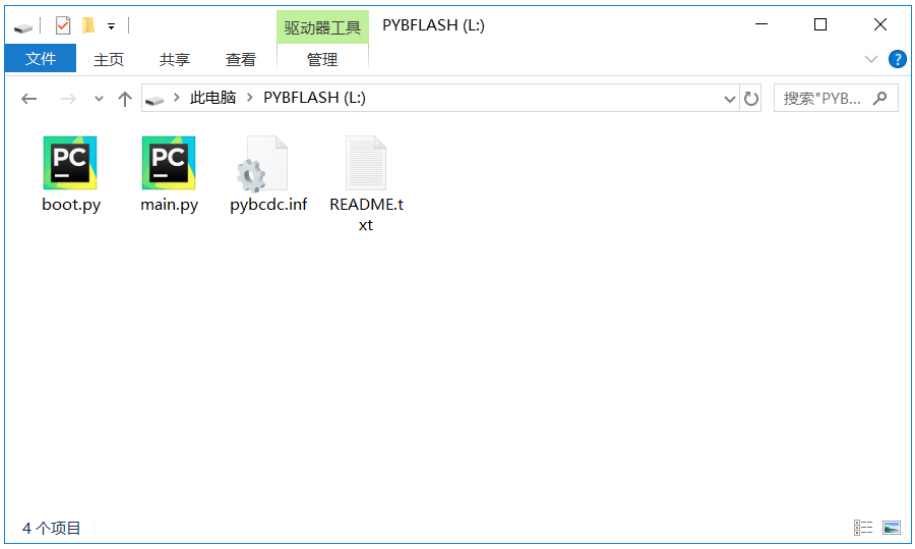


图 3-41 恢复出厂后的文件系统

打开 main.py，里面是空白的内容。

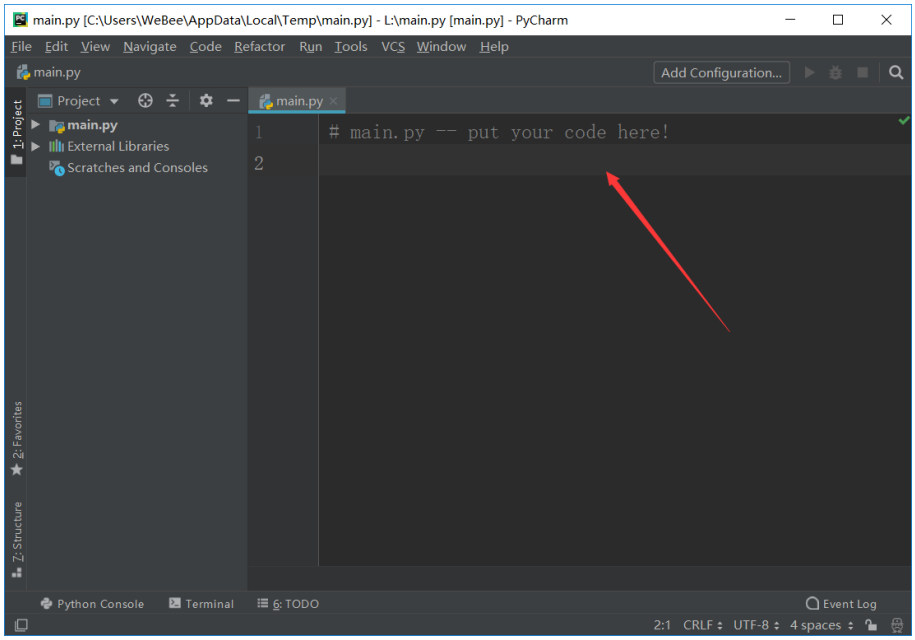


图 3-42 恢复出厂状态的 main.py 文件

3.1.3.6 固件更新

当 pyboard 上的固件意外丢失或者我们希望升级到较新版本固件时候，就需重新烧录 pyboard 的固件。Pyboard 上面是一个 M4 单片机，所以这个操作相当于给 M4 单片机重新烧录程序。

我们采用 DFU 的烧写方式，DFU 烧写方式优势是不需要 ST-LINK 一类仿真器，只需要安装一个软件即可。我们打开 MicroPython 开发套件配套资料\开发工具\Windows\固件更新工具\DfuSe Demo V3.0.5 目录下的 DfuSe 安装软件。DFU 软件安装完成后相当于电脑所需要的驱动也安装了。

将开发板上的默认连接的 BOOT--GND 跳线帽连接到 3.3V，让 pyboard 进入 DFU 烧写模式，如下图所示：

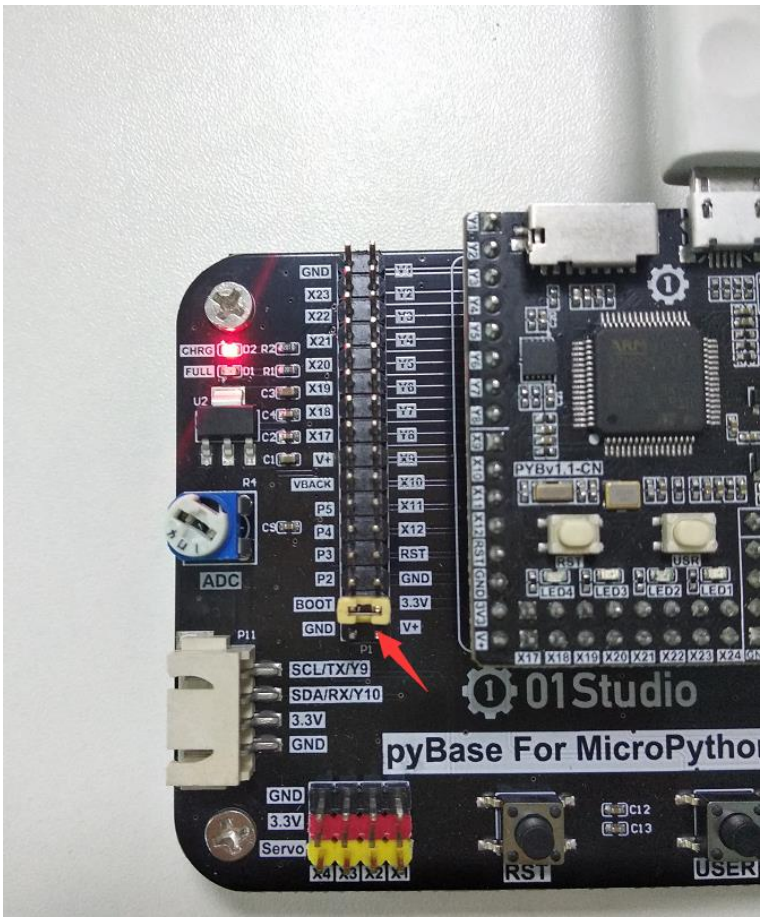


图 3-43 BOOT 跟 3.3V 通过跳线帽连起来

如果只有 pyBaord，则按下图方式用跳线帽或杜邦线将 DFU 与 3.3V 连接。



图 3-44 DFU 模式

打开刚刚安装的 DFU 软件，按下开发板 RST 复位键，可以见到 DFU 软件成功识别出 ST 的芯片。

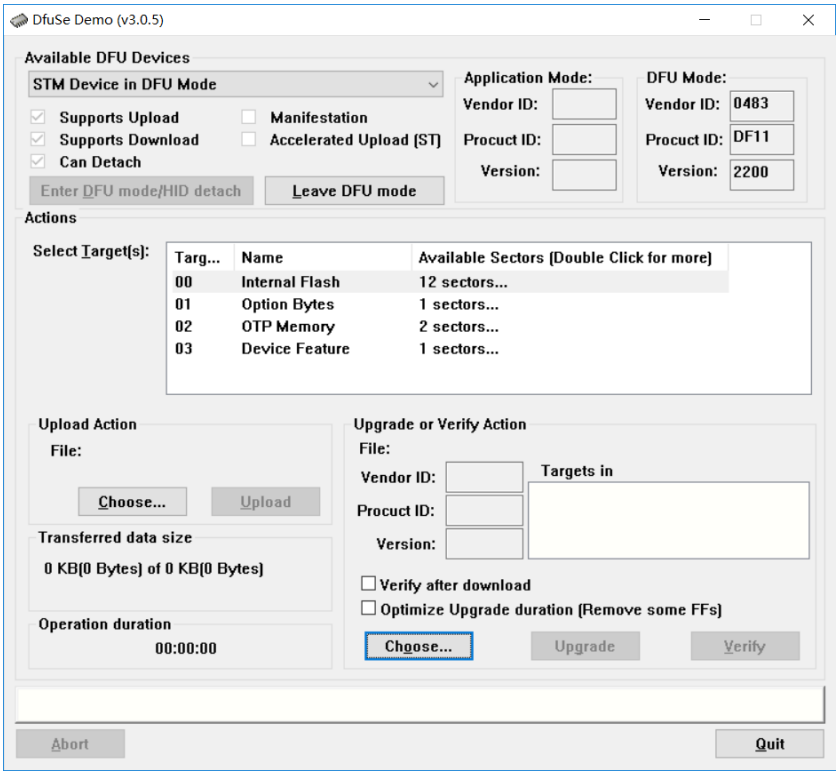


图 3-45

点击 choose，选择要升级的固件版本，在 MicroPython 开发板配套资料\相关固件\pyboard v1.1 目录下选择要升级的固件版本。

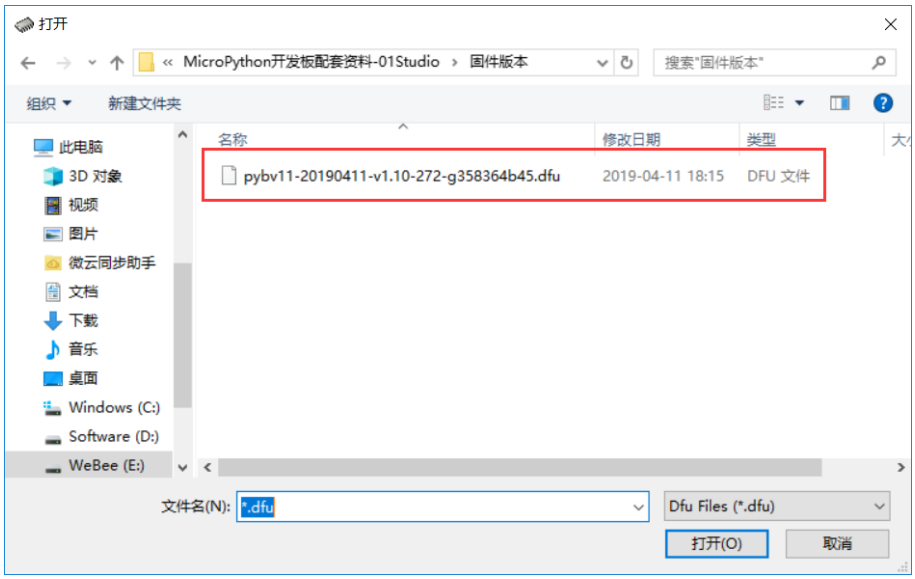


图 3-46 选择固件

勾选配置参数，点击 Upgrade 开始烧录。

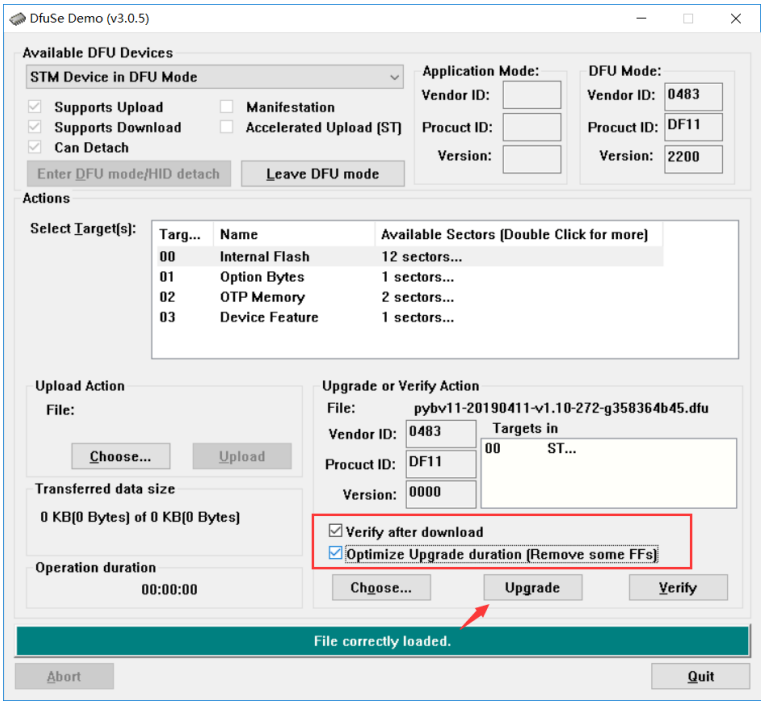


图 3-47

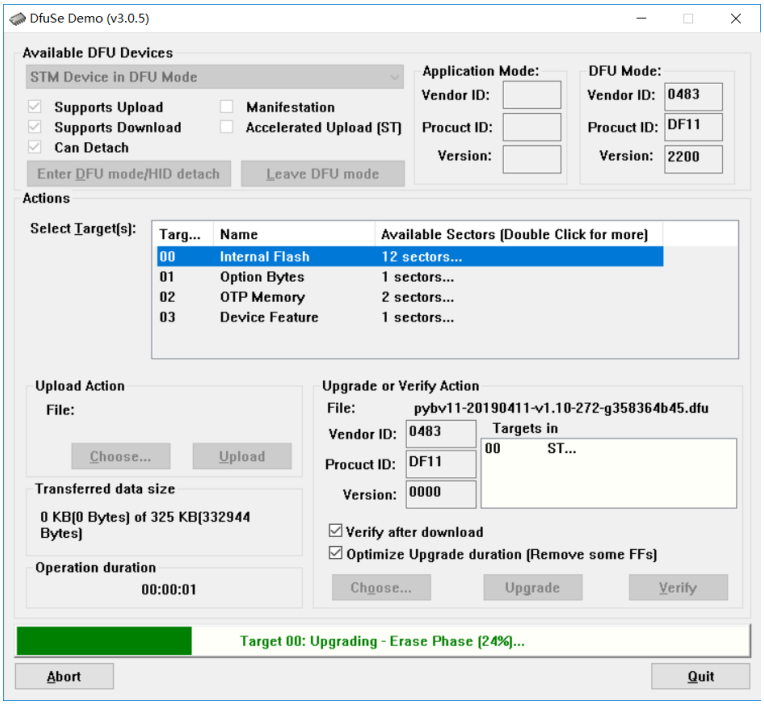


图 3-48 正在烧录

烧录完成后如下图所示：

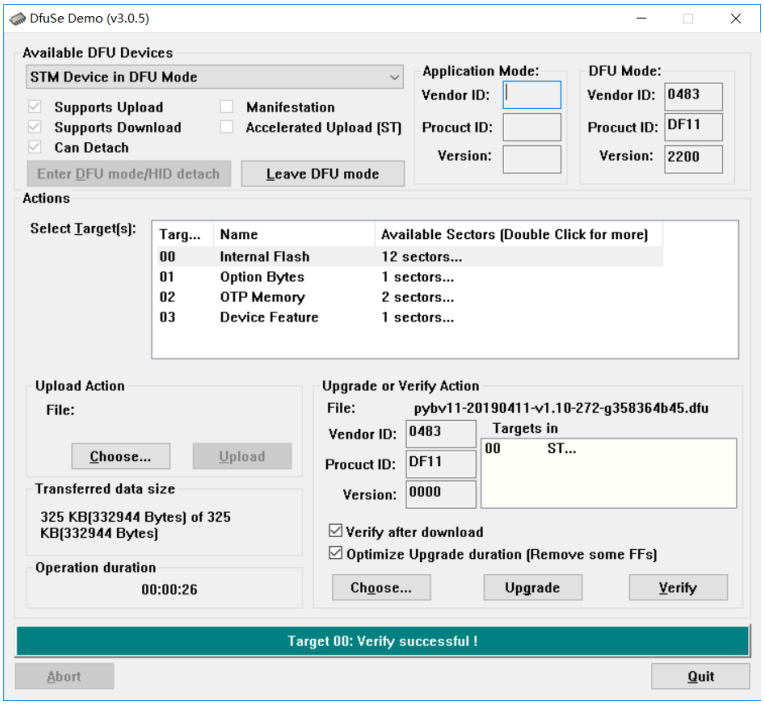


图 3-49 烧录完成

烧录完成后将 BOOT 跳线帽拔掉或者连接到开发板上的 GND 备用，重新按下复位键。可以见到系统检测到 pyboard 的 U 盘文件系统。说明固件更新完成。

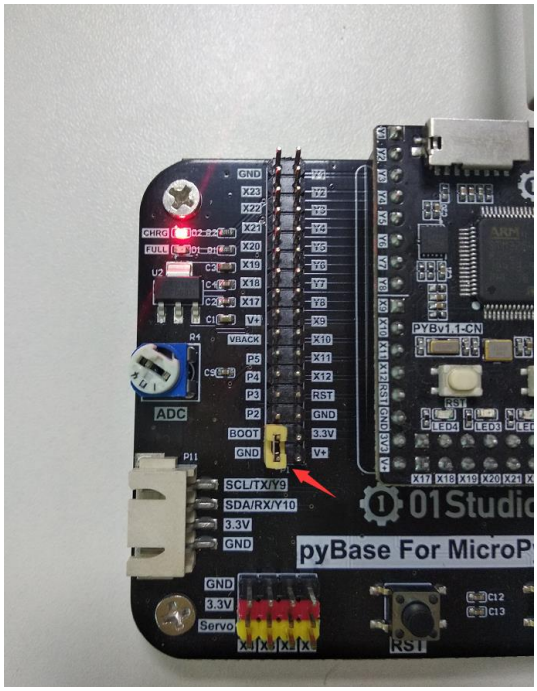


图 3-50



图 3-51

3.1.4 附录

3.1.4.1 编程软件 pyCharm 安装和使用

前面提到的 Mu 编程软件针对 MicroPython 开发比较友好。但考虑到有些用户可能是从事 python 相关工作，习惯了大型的编程软件，因此这里我们补充一项常用到的一款编程软件 pyCharm 的安装和使用教程。

pyCharm 的社区版是免费的，功能强大。很多 python 的开发者也是使用它具体安装方法如下：

在 https://www.jetbrains.com/pycharm/download 下载 Windows 社区版。

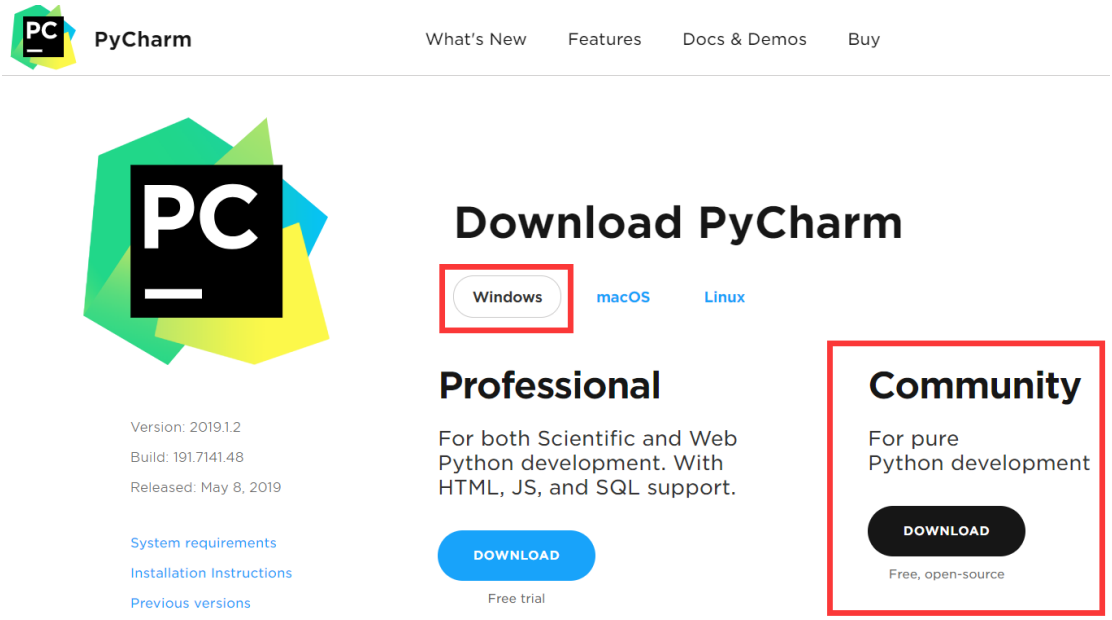


图 3-52 pyCharm 官方下载

下载完成后打开安装文件，选择自己喜欢的英文路径安装，出现配置界面，选择如下：根据自己的电脑配置选择 32bit 或者 64bit,环境变量 PATH 勾上，然后点击 NEXT，直至安装完成后重启电脑。

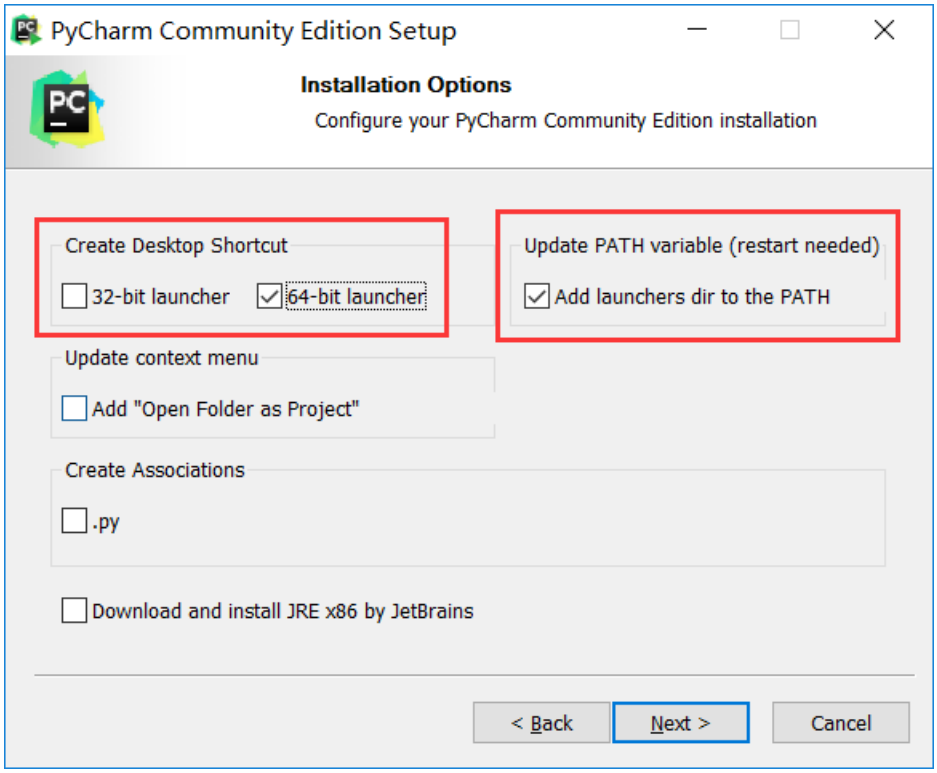


图 3-53 pyCharm 安装选项

安装完成重启后，我们打开 pyCharm，首次需要设置 pyCharm 的环境变量，环境变量这个词语看起来高大上，其实就是告诉 pyCharm 在编译 python 代码的时候，调用我们之前安装的 python3 应用作为解析器。否则是无法编译运行的。简单来说就是告诉 pyCharm 本电脑 python3 的安装路径，方法如下：

打开 pyCharm，点击 Configure—settings：

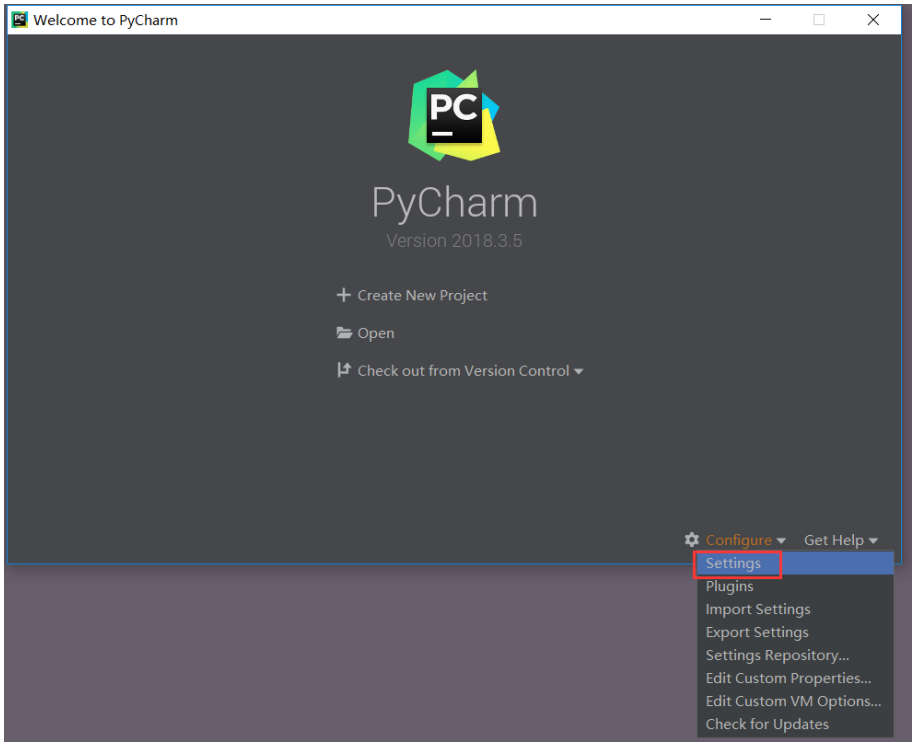


图 3-54

选择 Project Interpreter，点击右上角设置图标，选择 add:

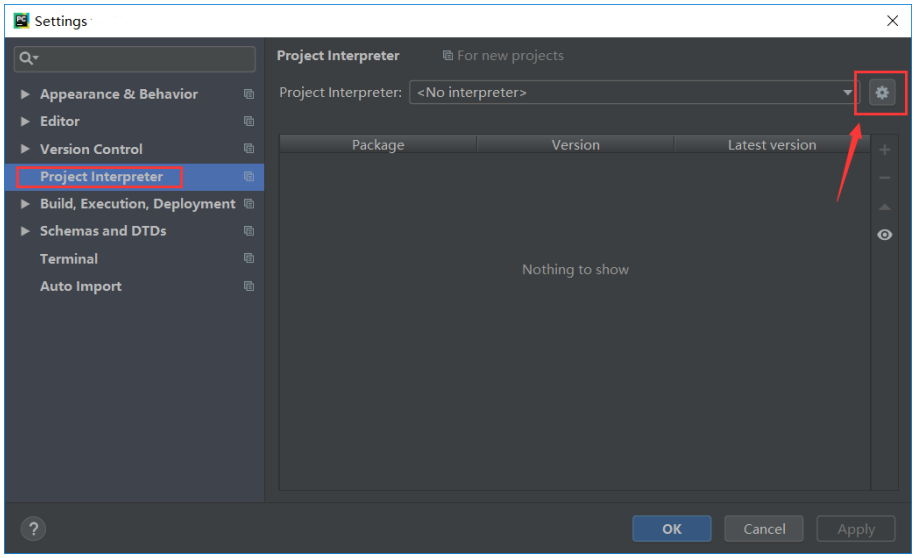


图 3-55 解析器配置

在弹出的新界面选择 Existing environment,路径选择之前安装 python3 的路径，选上 python.exe。另外注意勾上 make available to all projects，以后就不用重复配置。

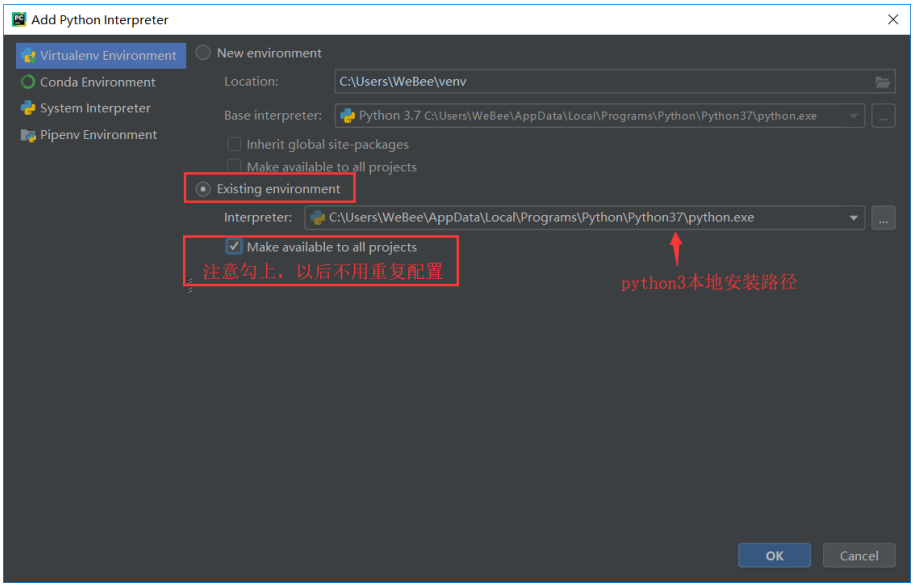


图 3-56 配置

关于 python3 安装路径，可以参考以下方法查找，在程序搜索“python.exe”(注意这个名称要一字不漏对上)， 点击右键-打开文件所在位置，即可找到 python3 安装路径。



图 3-57

将路径复制到 pyCharm 配置即可。

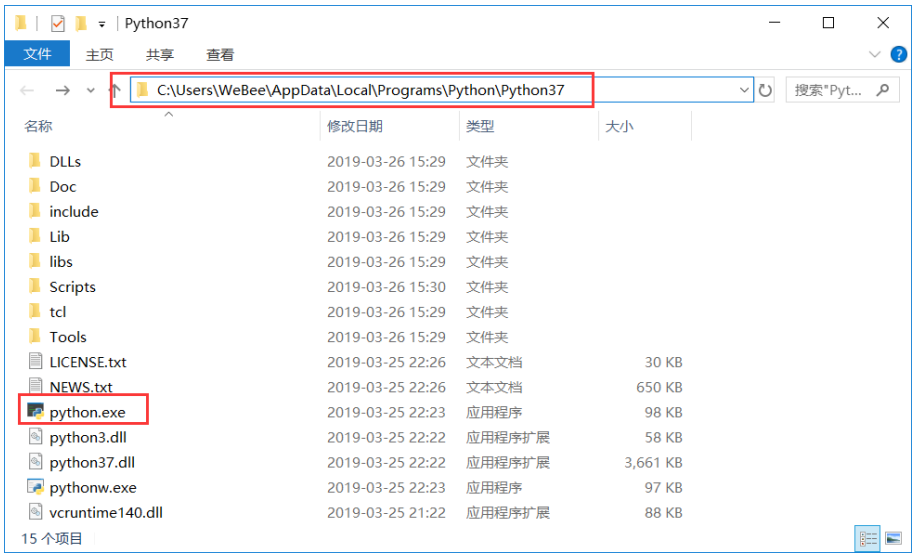


图 3-58 python3 安装路径

完成以上配置后，我们新建一个工程和文档测试一下。打开 pyCharm—Create New Project

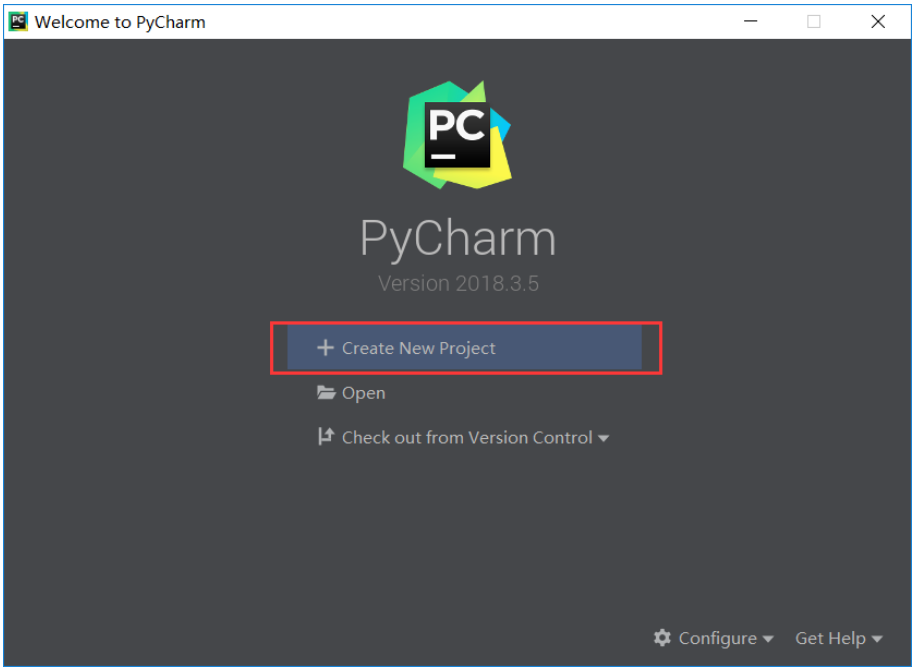


图 3-59 新建工程

在桌面或者自己喜欢的路径下新建一个名为 test 的工程

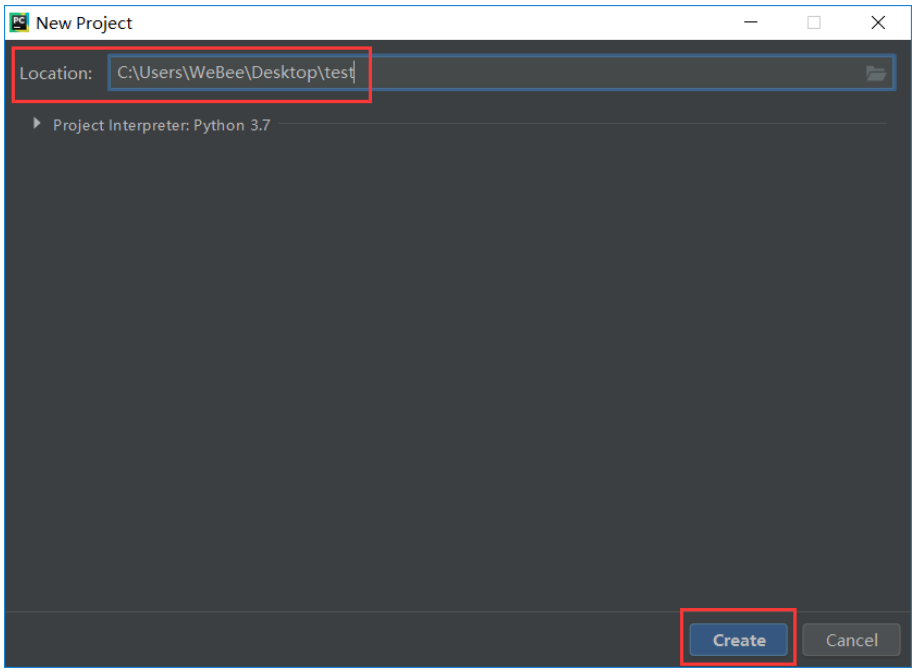


图 3-60 新建名为 test 的工程

接下来在 test 工程下新建 python 文件，在 project 下空白区域点击右键—new--Python File.

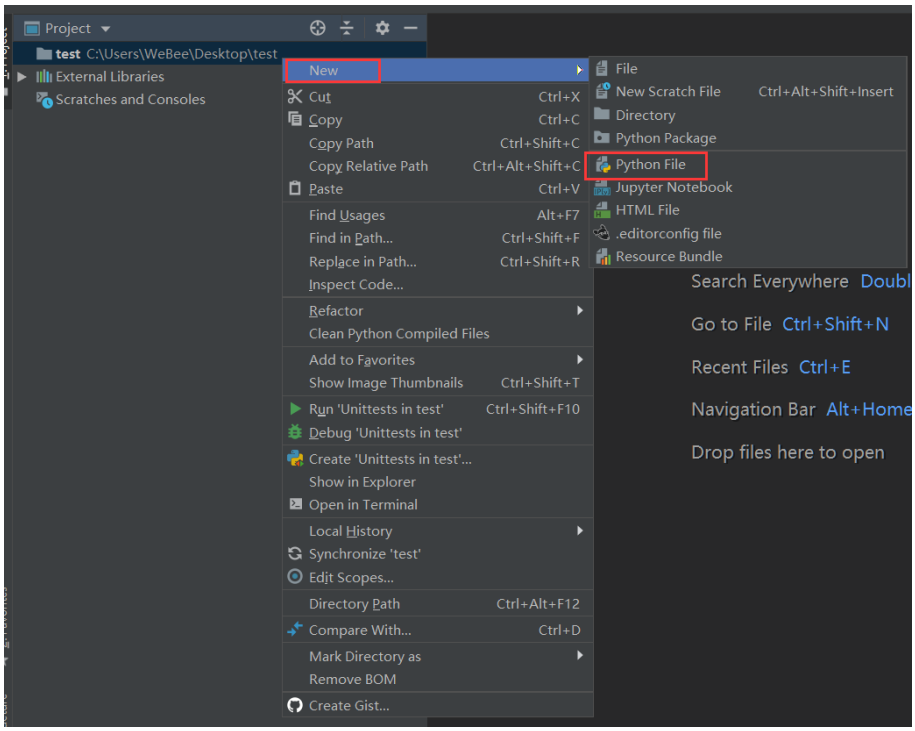


图 3-61 添加 python 文件

输入自己想要的 python file 名称，我们这里输入 test

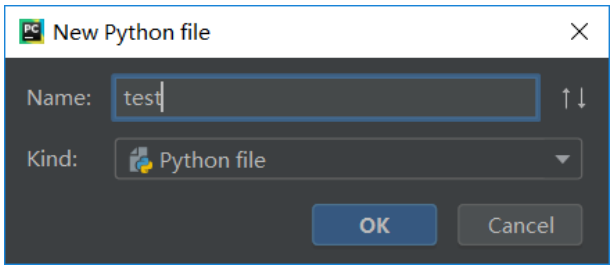


图 3-62

在编辑框输入 print(“Hello 01Studio!”)

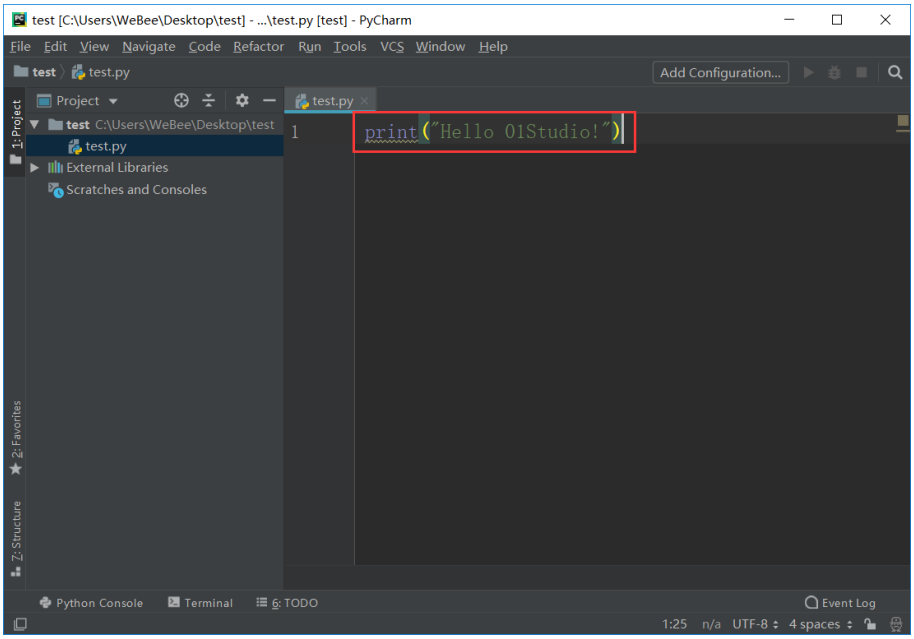


图 3-63

默认字体可能有点小，我们可以调整一下，点击 File—settings—Editor—Font，将 Size 调到 20。

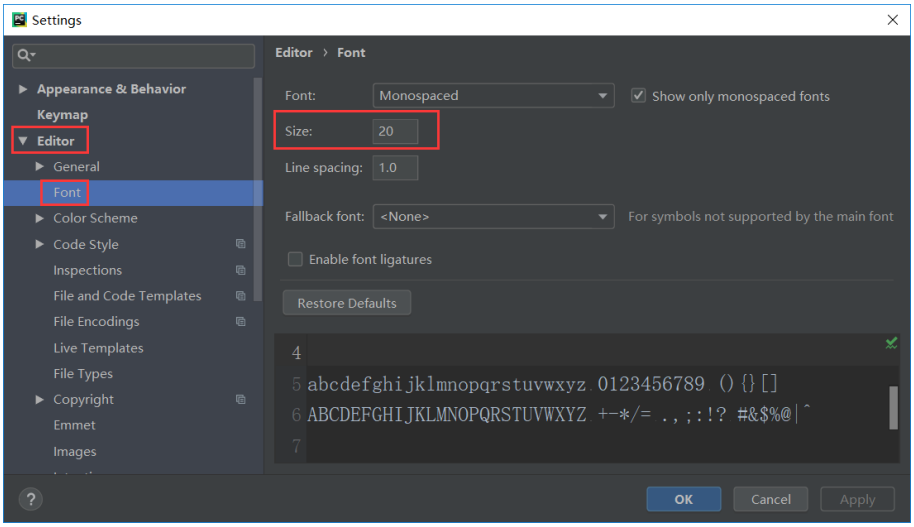


图 3-64 调整编辑框字体大小

点击 Run，选择 test (首次需要选择)

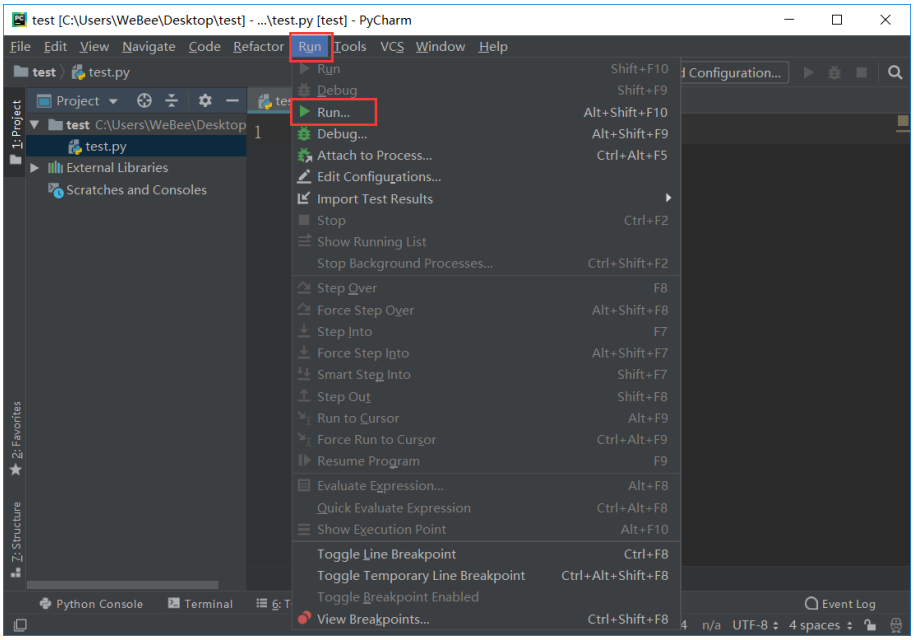


图 3-65 运行

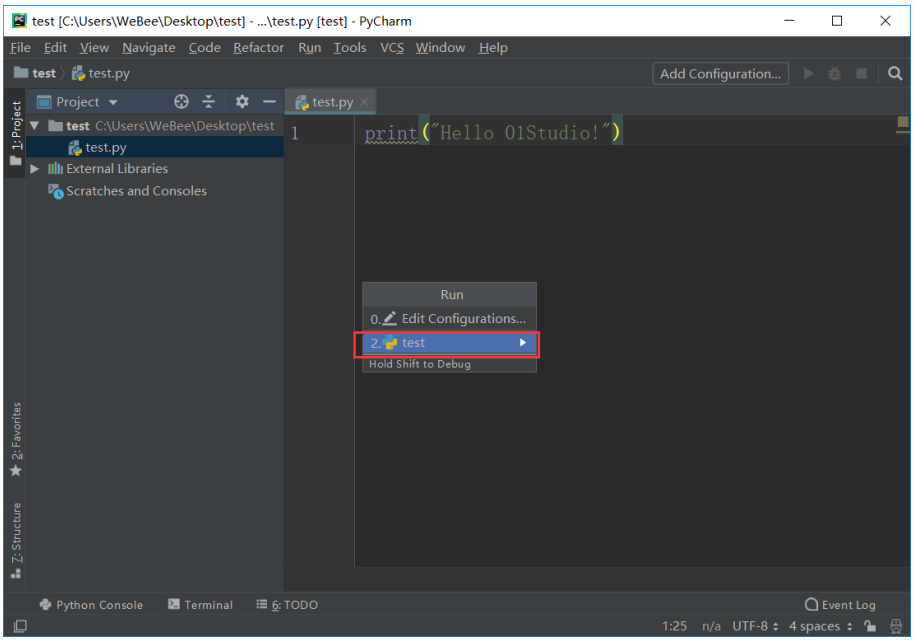


图 3-66 选择要运行的文件

我们看到输出框打印出来了 Hello 01Studio，说明运行成功。

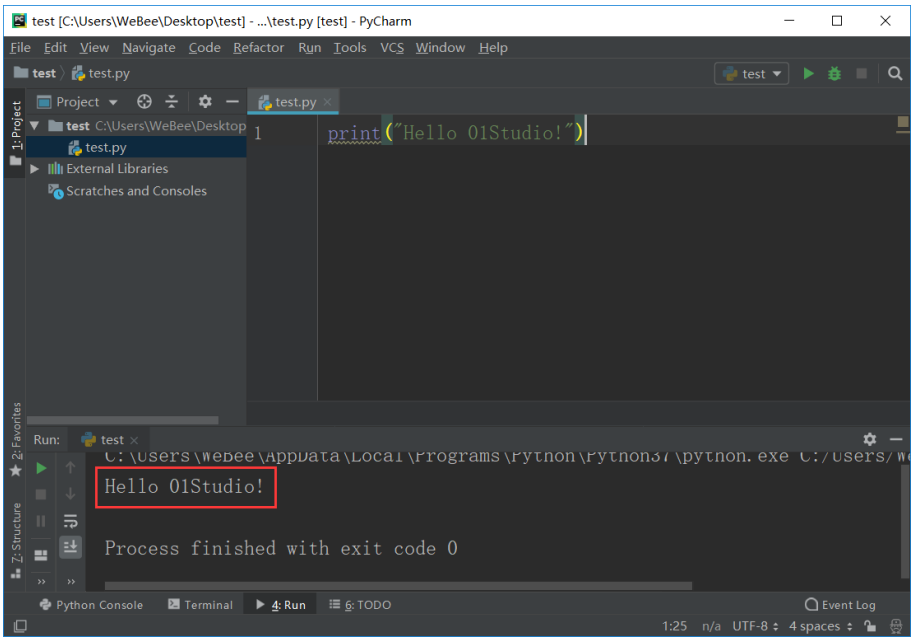


图 3-67 运行成功

至此，python 编译器安装和配置成功了。需要注意的是，在打开别人 python 项目的时候，有可能出现环境变量不匹配情况，重新按照以上方法设置一下即可。现在，你甚至可以用 pyCharm 来开发相关 python 应用了。

3.2 基于 Mac OS

还记得当年乔布斯从大信封里面拿出 macbook 么，苹果和微软在 PC 时代走了两个极端，乔布斯主导下的苹果要求软硬一体化，封闭，务求最佳的用户体验；而微软则主打开发，让操作系统兼容不同的 PC 厂家，使得其 windows 一度占据了 90%的市场份额。从商业角度，盖茨毫无疑问是赢家，就个人而言，我更认可乔布斯的方式，现在的 win10 和微软自家的笔记本，某程度上都有一体化的思想。我用了半天的 Macbook 和 MAC 系统，如果不是要做硬件发，基本上可以放弃十多年的 windows 了。

3.2.1 安装 python3

在 MAC 系统下，安装 python3 方法也很简单，在官网 www.python.org 下载 Mac 版 python3，根据自己系统情况选择 32bit 或者 64bit 下载安装。

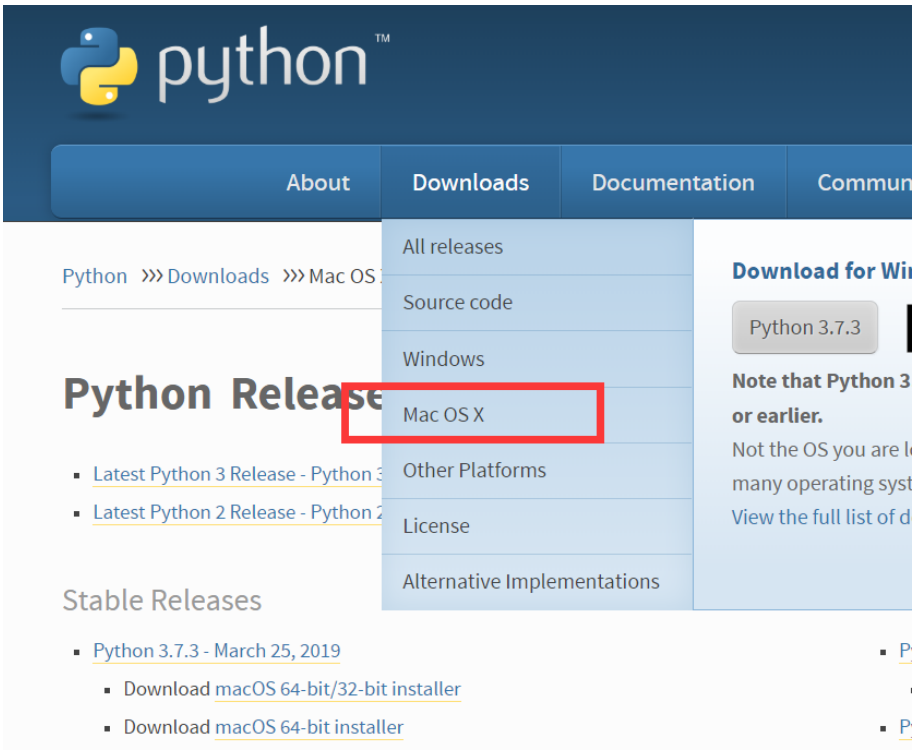


图 3-68

安装完成后在终端输入 python3，出现版本好的话说明安装成功。

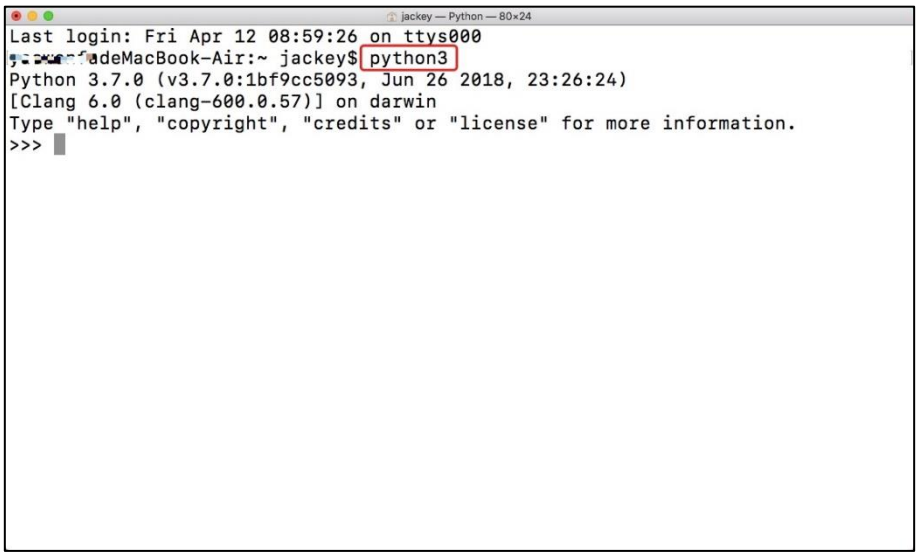


图 3-69 成功安装

3.2.2 安装开发软件 Mu

Mu 在 MAC OS 环境下的安装和使用方法和 Windows 一样，具体请参考：**3.1.2 安装开发软件 Mu** 章节内容，这里不再重复。

3.2.3 开发套件使用

安装完成后将 MicroPython 开发板连接到 Macbook，可以看到同样是出现盘符 PYBFLASH。打开可以看到文件系统，跟 Windows 下一样。使用 pyCharm 打开 main.py 即可开始编程。可参考“0 第一个实验”章节，这里不再重复。

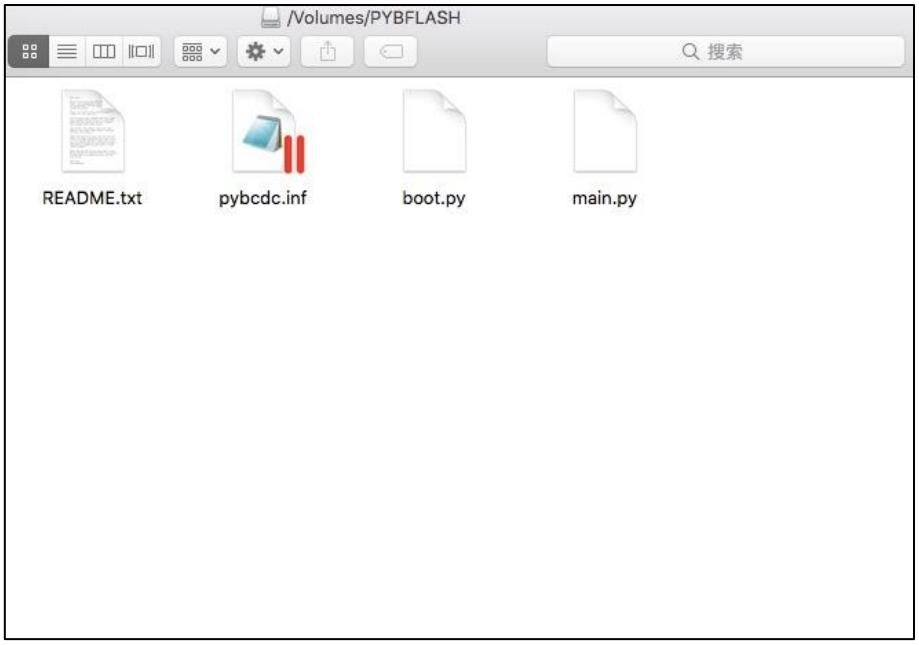


图 3-70

3.2.3.1 REPL 串口交互调试

Macbook 自带这个版本的串口驱动，一般不需要再安装，查看方法是点击左上角苹果图标—关于本机—系统报告—选择 USB。可以看到相关驱动如下图所示：



图 3-71 关于本机

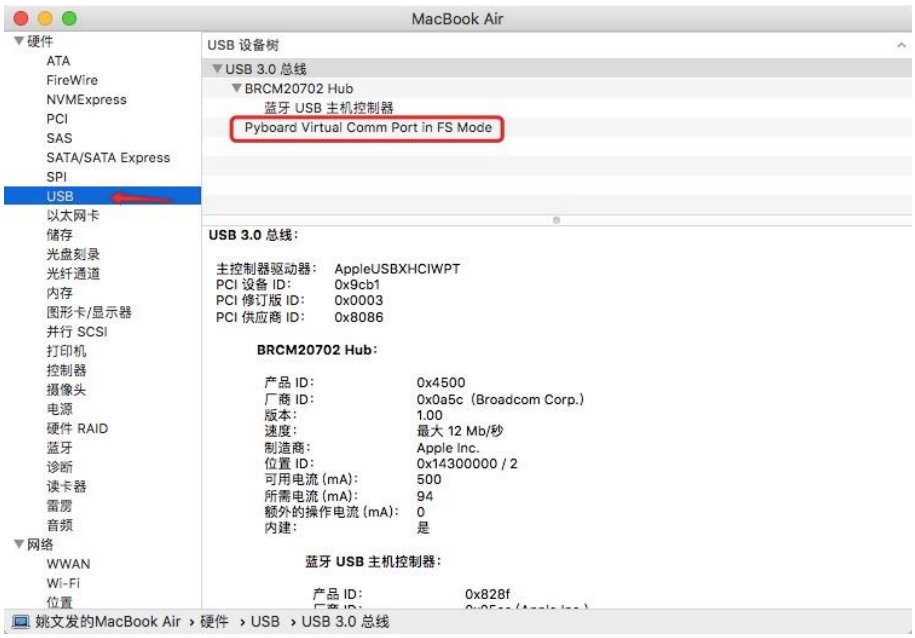


图 3-72 pyboard 虚拟串口驱动

检查驱动已经安装后，可以通过 screen 命令来打开 MAC 内置的串口终端来跟开发板进行交互，具体如下:

可以在终端输入 **screen /dev/tty.usbmodem，然后按键盘 TAB 键，自动补全系统的设备名称。**



图 3-73 按 tab 保全名称

波特率是 115200，在设备号后加空格，输入*115200，按回车。



图 3-74 完整命令



图 3-75 成功连接 pyboard

成功连接后可以按之前的方法简单测试一下开发板，先输入 import pyb，按回车；再输入 pyb.LED(3).on()，按回车；可以看到开发板上 LED3 黄灯被点亮。



图 3-76

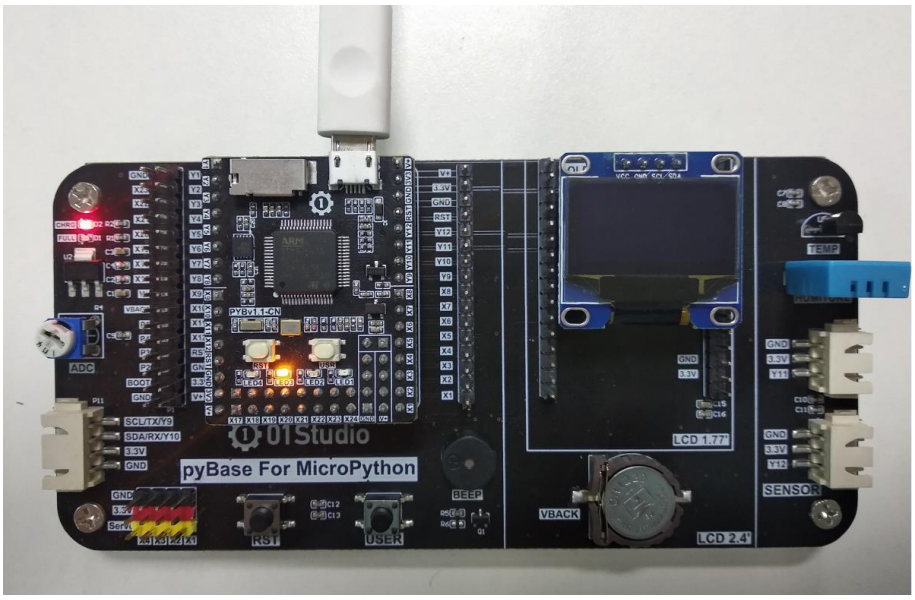


图 3-77 点亮 LED3-黄灯

想退出 screen，可以用下面方法：先按 control + A ，在按 control + \ ，会弹出询问是否关闭窗口，按键盘 y 即可退出。除此之外直接关闭窗口也可以退出。

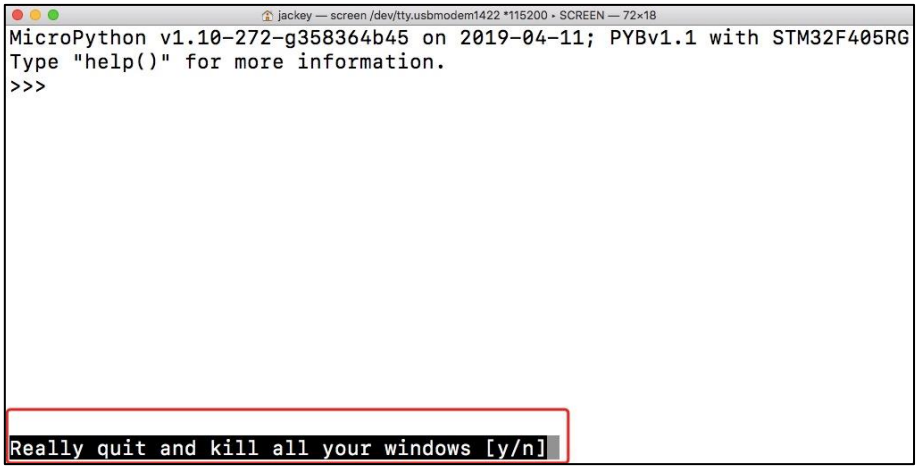


图 3-78 退出 screen 命令

3.2.3.2 串口终端 minicom

自带得 screen 命令来调试串口并不是那么方便，每次都要敲很长的命令，我们给大家推荐一款基于 MAC 的串口终端 minicom，可以通过配置记录串口，使用非常方便。

1. 安装 minicom：

   在终端输入：brew install minicom 进行安装，自动安装完成。

2. 找到设备名称

   在终端输入：ls /dev/tty.usb

3. 配置 minicom:

   在终端输入：minicom -s ，进入配置界面后选择第三项 Serial port setup 进入：

   （1） 设置 Serial Device 为：/dev/tty.usbmodem1422

   （2） 设置波特率和流控；

通过按 A、B…选择配置项，修改完后把光标移动到行尾，再按 Enter 键，否则配置不生效，然后回到上一层，选择 Save as dfl 保存退出。

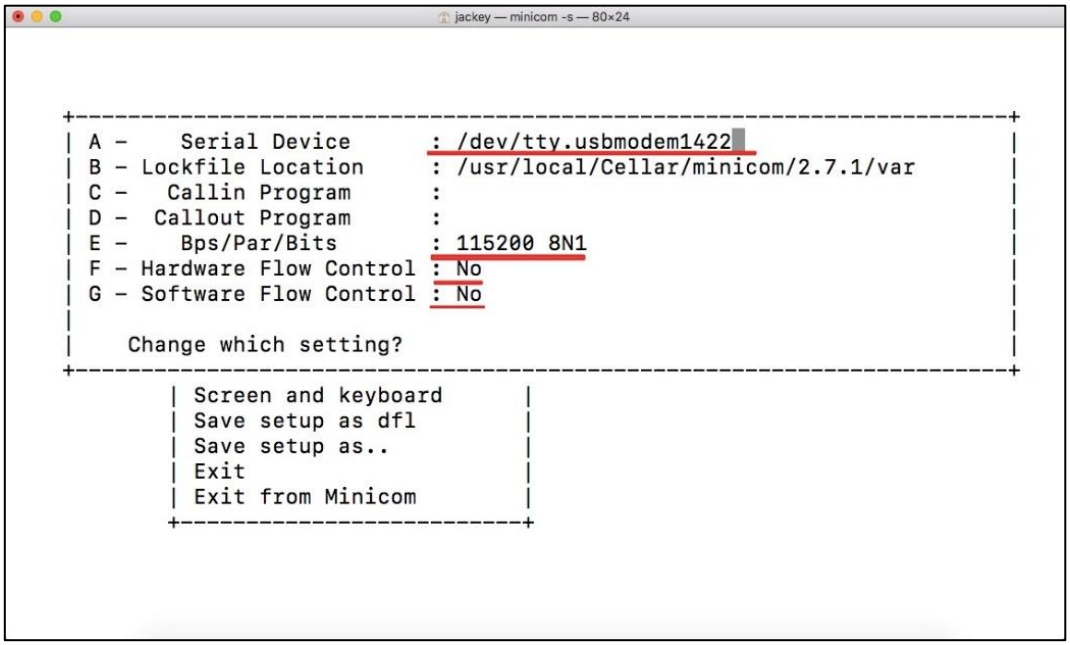


图 3-79 串口参数配置

4. 运行 minicom:

在终端输入：minicom ,按 Enter。即可打开刚刚配置好的串口。此时若连接了 MicroPython 开发板，发现可以直接进入了命令对话框。

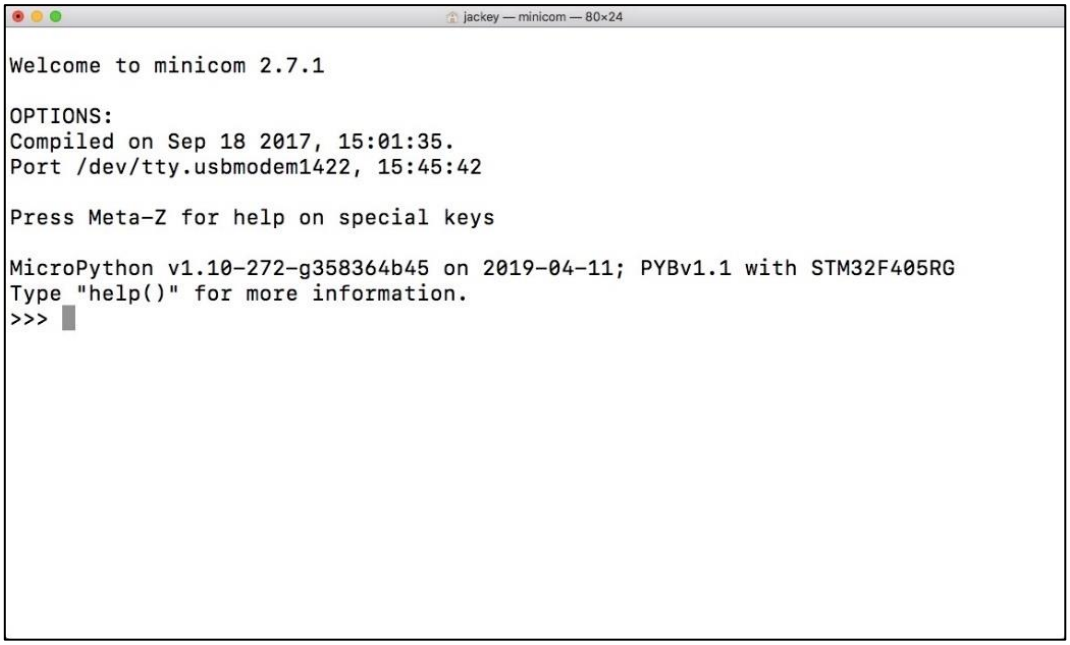


图 3-80 使用 minicom 进入 REPL

需要退出的话直接关闭窗口即可。

3.3 基于 Linux

Linux 系统为大多工程师热衷的开源操作系统，一般内部都集成了 python3 和 IDE，用户直接使用即可。本节教程同样适用于树莓派的 Linux 系统。

3.3.1 开发软件 Mu 安装

安装方法请参考：https://codewith.mu/en/howto/1.0/install_raspberry_pi

3.3.2 串口终端 minicom

Linux 下可以跟 MAC 一样使用 minicom 作为串口交互，安装和使用方法跟 MAC 类似，具体如下。

1. 安装 minicom：

在终端输入：sudo apt-get install minicom 进行安装，自动安装完成。

2. 找到设备名称

设备名称在 /dev 下，以 tty 开头，可以通过插入 micropython 开发板后所增加的设备号来确认设备名称。通过 ls /dev/tty*命令可以列出所有的设备名称。这里的名称是 ttyACM0。



图 3-81 pyBoard 串口设备名称

3. 配置 minicom:

在终端输入：minicom -s ，进入配置界面后选择第三项 Serial port setup 进入：

（3） 设置 Serial Device 为：/dev/ttyACM0

（4） 设置波特率和流控；

通过按 A、B...选择配置项，修改完后把光标移动到行尾，再按 Enter 键，否则配置不生效，然后回到上一层，选择 Save as dfl 保存退出。如出现无法保存的情况，则需要重启电脑再重复上面步骤即可。

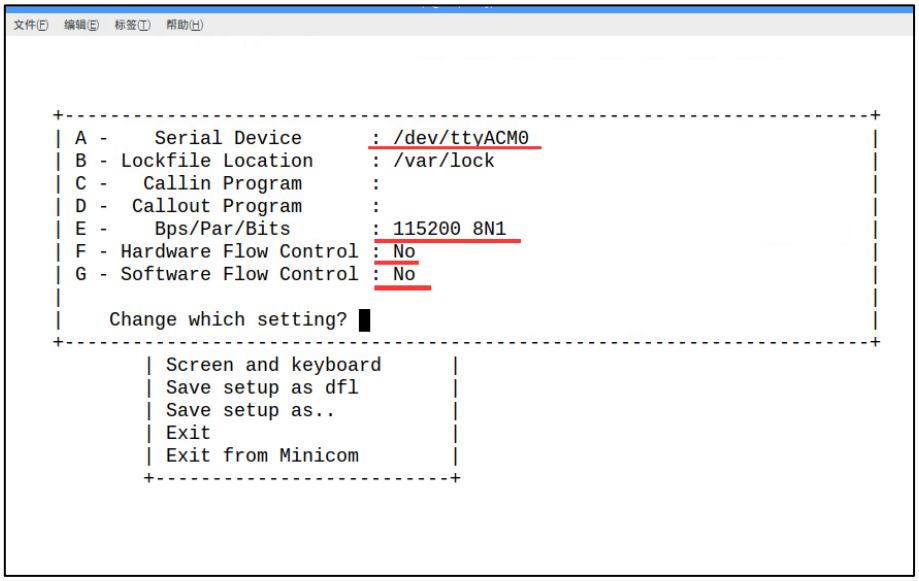


图 3-82 串口参数配置

4. 运行 minicom:

在终端输入：minicom ,按 Enter。即可打开刚刚配置好的串口。此时若连接了 MicroPython 开发板，发现可以直接进入了命令对话框。

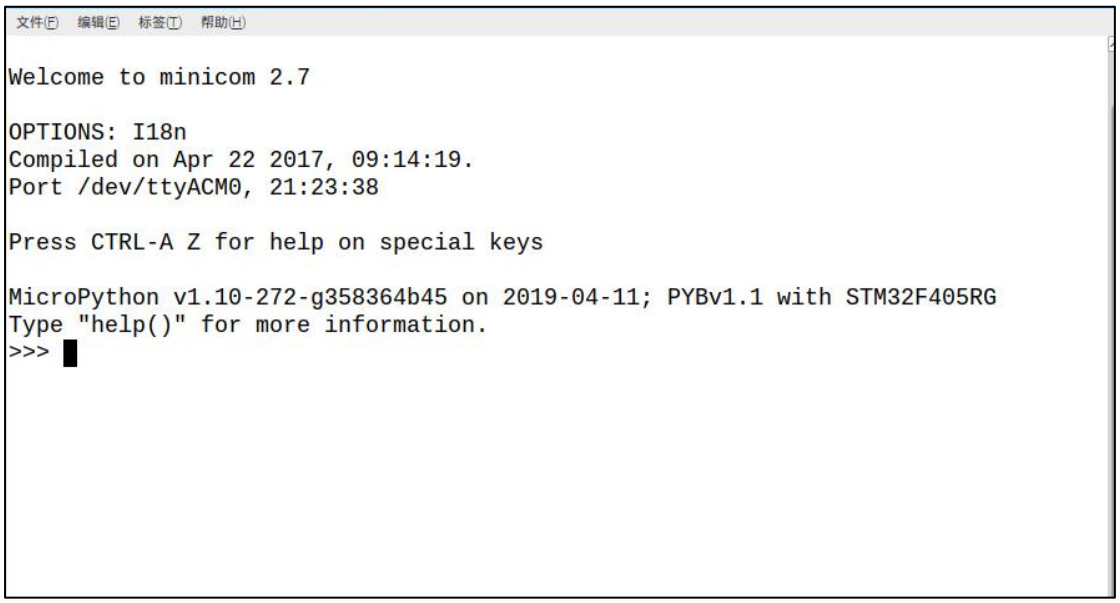


图 3-83 使用 minicom 进入 REPL

需要退出的话直接关闭窗口即可。

第4章　基础实验

MicroPython 更强调的是针对应用的学习，强大的底层库函数让我们可以直接关心功能的实现，也就是说我们只要理解和熟练相关的函数用法，就可以很好的玩转 MicroPython。它让我们可以做到不关心硬件和底层原理（当然有兴趣和能力的小伙伴可以深入研究）而直接跑起硬件。

基础实验是针对一些简单的实验学习讲解，可以让我们更快和更直接感受到 MicroPython 针对嵌入式开发的强大之处， 我后面的学习和开发夯实基础。

请先看实验讲解格式预览，每一节我们都会以以下形式讲解，图文并茂，务求达到快速理解和学习的作用：

（1） 前言：简单介绍这个实验；

（2） 实验平台：实验所用到的开发板、配件和接线说明；

（3） 实验目的：本实验要实现的功能；

（4） 实验讲解：对函数、代码、编程方法以及实验的详细讲解；

（5） 实验结果：记录程序下载到开发板上的图片示例；

（6） 总结:对实验进行总结。

4.1 点亮第一个 LED 灯

- **前言：**

相信大部分人开始学习嵌入式单片机编程都会从点亮 LED 开始，我们 MicroPython 的学习也不例外，通过点亮第一个 LED 能让你对编译环境和程序架构有一定的认识，为以后的学习和更大型的程序打下基础，增加信心。

- **实验平台：**

pyBoard v1.1-CN 或者 MicroPython 开发套件。

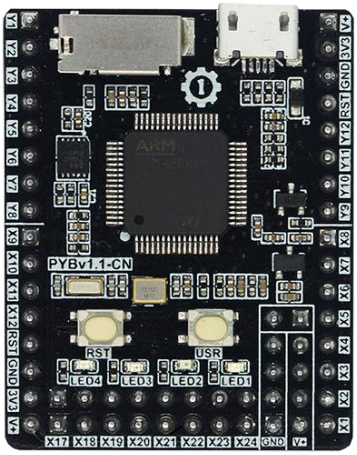


图 4-1 pyBoard v1.1-CN

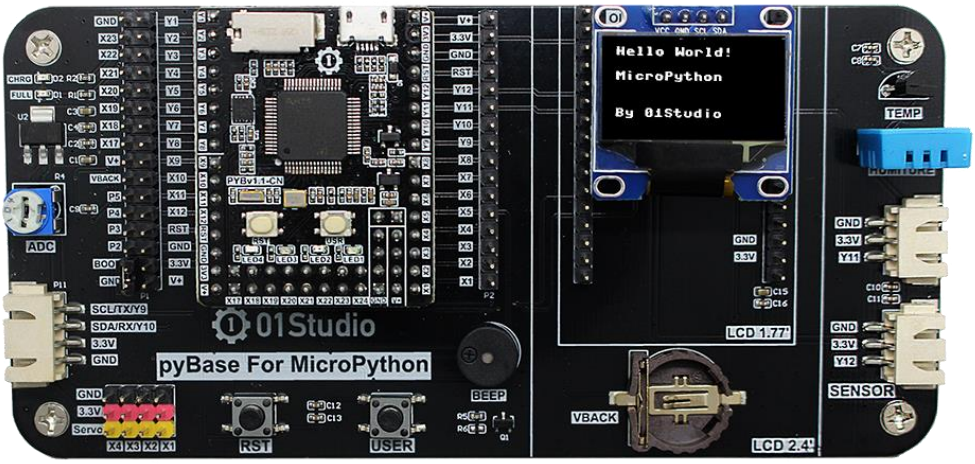


图 4-2 MicroPython 开发套件

- **实验目的：**

  点亮 LED4（蓝灯）。

- **实验讲解：**

pyboard 上总共有 4 个 LED，分别是 LED1(红色)、LED2(绿色)、LED3(黄色)、LED4(蓝色)；控制 LED 使用到 LED 对象。LED 的构造函数和使用方法如下表所示：

构造函数	说明
pyb.LED(id)	其中 id 是编号，1-4 分别对应 LED1 至 LED4
使用方法	**说明**
LED.off()	关闭 LED
LED.on()	打开 LED
LED.toggle()	打开/关闭 LED 状态切换（反转）
LED.intensity([value])	亮度调节，value 的值范围是 0-255.

表 4-1 LED 对象

上表对 MicroPython 的 LED 对象做了详细的说明，pyb 是大模块，LED 是 pyb 下面的其中一个小模块，在 python 编程里有两种方式引用相关模块:

方式 1 是：import pyb，然后通过 pyb.LED 来操作；

方式 2 是：from pyb import LED,意思是直接从 pyb 中引入 LED 模块，然后直接通过 LED 来操作。显然方式 2 会显得更直观和方便，本实验也是使用方式 2 来编程。代码编写流程如下：

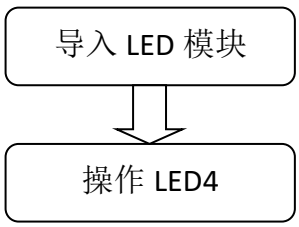


图 4-3 代码编写流程

以下是本实验示例代码：

```
'''
实验名称：点亮 LED(4)蓝灯
版本：v1.0
日期：2019.4
作者：01Studio
'''

from pyb import LED   #从 pyb 导入 LED 模块
LED(4).on()   #打开 LED4
```

打开 main.py 文件，将上面代码拷贝进去并保存。等程序加载完毕（红灯从亮到灭）后，按下 RST 复位键。便能看到实验结果。看到 LED4 蓝灯被成功点亮。

- **实验结果：**

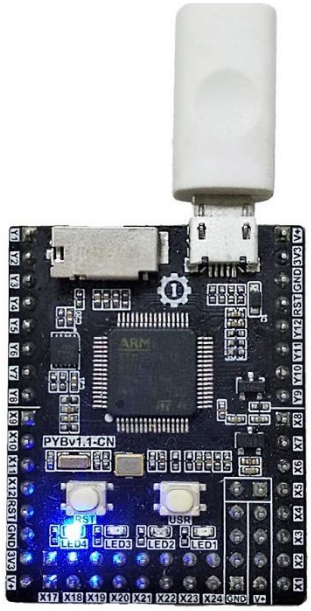


图 4-4 基于 pyBoard v1.1-CN

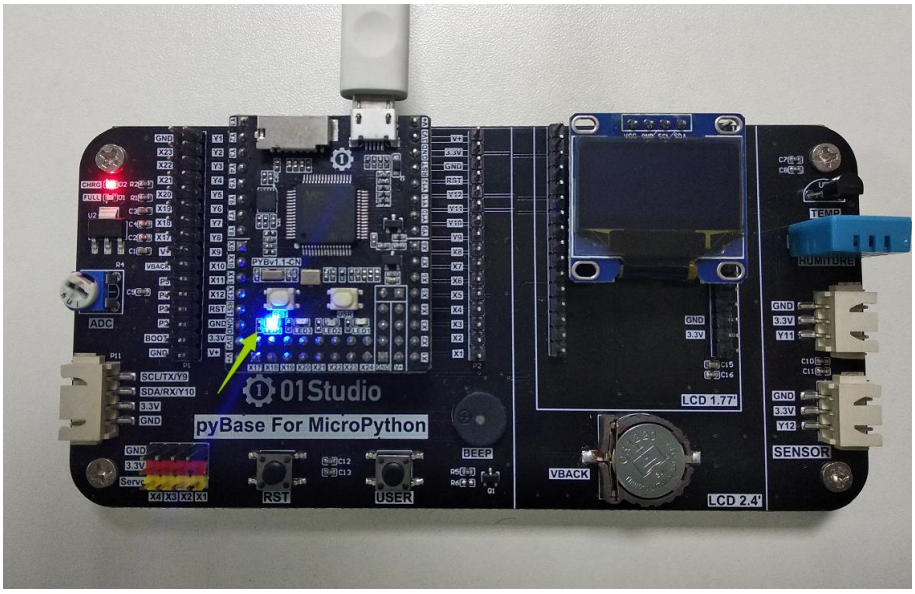


图 4-5 基于 MicroPython 开发套件

- **总结：**

从第一个实验我们可以看到，使用 MicroPython 来开发关键是要学会构造函数和其使用方法，便可完成对相关对象的操作，在强大的模块函数支持下，实验只用了简单的两行代码便实现了点亮 LED 灯。

4.2 流水灯

- **前言：**

通过上一节点亮 LED 灯的学习，我们已经对 micropython 的编程有了初步的了解，这一节来做一个功能稍微复杂一点的实验，流水灯。流水灯也叫跑马灯，也就是让几个 LED 来回亮灭，达到好像流水的效果。也是单片机开发学习的典型例子。

- **实验平台：**

pyBoard v1.1-CN 或者 MicroPython 开发套件。

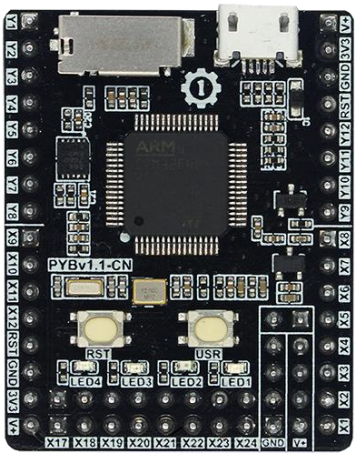


图 4-6 pyBoard v1.1-CN

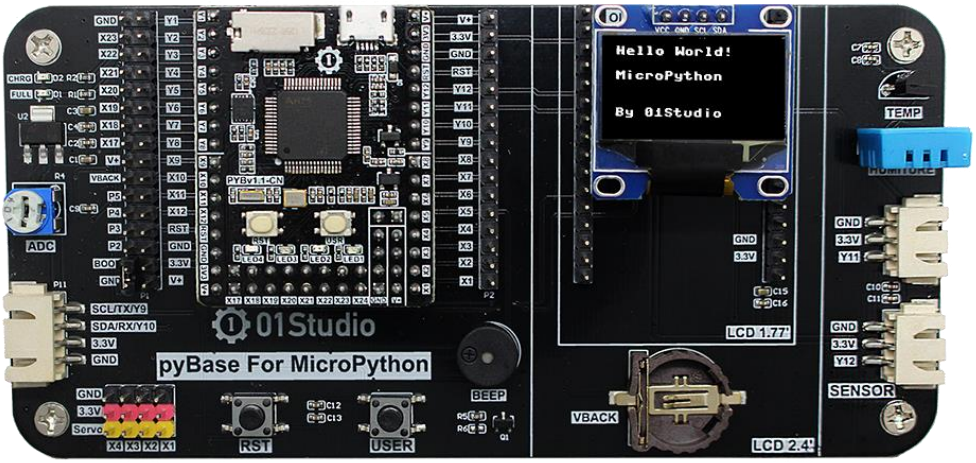


图 4-7 MicroPython 开发套件

- **实验目的：**

流水灯。让 LED2-LED4 循环亮灭，达到像流水一样的效果。

- **实验讲解：**

pyboard 上总共有 4 个 LED，分别是 LED1(红色)、LED2(绿色)、LED3(黄色)、LED4(蓝色)；控制 LED 使用到 LED 对象。上一章节我们已经学习过 LED 点亮，这里要实现固定时间来亮灭，需要用到 delay 延时的函数。具体如下：

LED 的构造函数和使用方法如下表所示：

构造函数	说明
pyb.LED(id)	其中 id 是编号，1-4 分别对应 LED1 至 LED4
使用方法	**说明**
LED.off()	关闭 LED
LED.on()	打开 LED
LED.toggle()	打开/关闭 LED 状态切换（反转）
LED.intensity([value])	亮度调节，value 的值范围是 0-255.

表 4-2　LED 对象

Delay 延时函数和使用方法如下表所示：

构造函数	说明
pyb.delay(ms)	毫秒级延时
pyb.udelay(us)	微秒级延时
使用方法	**说明**
pyb.delay(1000)	延时 1000ms
pyb.udelay(1000)	延时 1000us

表 4-3　延时对象

知道了函数的使用方法后，我们可以简单的梳理一下流程，首先导入 LED 和 Delay 模块，程序开始先让 LED2-LED4 灭掉，开启循环，依次点亮每个 LED，延时

1 秒，关闭 LED。流程如下：

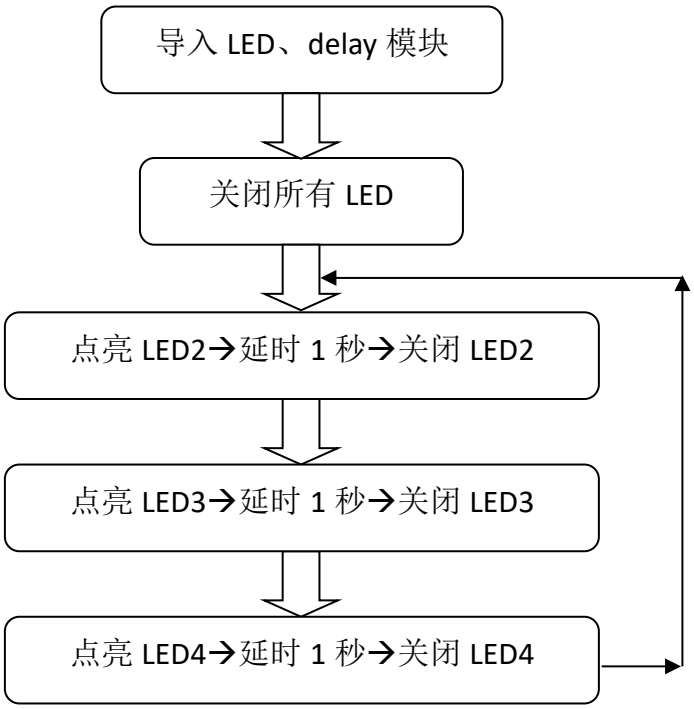


图 4-8 代码编写流程

以下是本实验参考程序代码：

```
'''
实验名称：流水灯
版本：v1.0
日期：2019.4
作者：01Studio
'''

from pyb import LED,delay   #从 pyb 导入 LED 模块

#关闭全部 LED
```

```
LED(2).off()
LED(3).off()
LED(4).off()

#while True 表示一直循环
while True:
    #LED2 亮 1 秒
    LED(2).on()
    pyb.delay(1000) #延时 1000ms,即 1 秒
    LED(2).off()

    #LED3 亮 1 秒
    LED(3).on()
    pyb.delay(1000)
    LED(3).off()

    #LED4 亮 1 秒
    LED(4).on()
    pyb.delay(1000)
    LED(4).off()
```

上述代码没错是完整的按照编程思路来编写，但可以见到有很多格式相似的地方，这显得代码非常冗余。我们可以通过 for 函数来编写程序，由于是对 LED2/3/4 的操作，因此我们可以用 for i in range(2,5): 语句来修改。参考代码如下：

```
'''
实验名称：流水灯
```

```
版本：v2.0
日期：2019.4
作者：01Studio
'''
from pyb import LED,delay   #从 pyb 导入 LED 模块

# 相当于 for i in [2, 3, 4]，LED(i).off()执行 3 次，分别是 LED 2，3，4
for i in range(2,5):
    LED(i).off()

while True:
    #使用 for 循环
    for i in range(2,5):
        LED(i).on()
        delay(1000) #延时 1000 毫秒，即 1 秒
        LED(i).off()
```

修改后的代码一下子变得简约美观。Python 作为高级语言，使用起来非常灵活，常常能通过几行代码就可以表达复杂的逻辑。我们在编程的过程中应该多思考如何优化代码，才能让自己的编程水平得到较快的提升。

（需要注意的是如果在 main.py 文件里使用了类似 while True：的死循环函数，pyboard 的 REPL 功能将失效。）

- **实验结果：**

  LED2、LED3、LED4 循环点亮，实现流水效果。

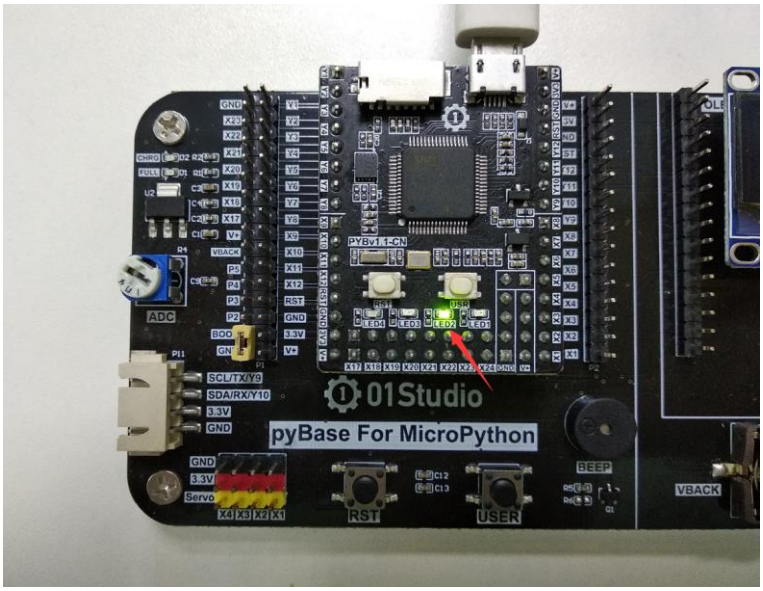

01Studio
pyBase For MicroPython
RST
USER
BEEP
VBACK
ADC
Servo

图 4-9

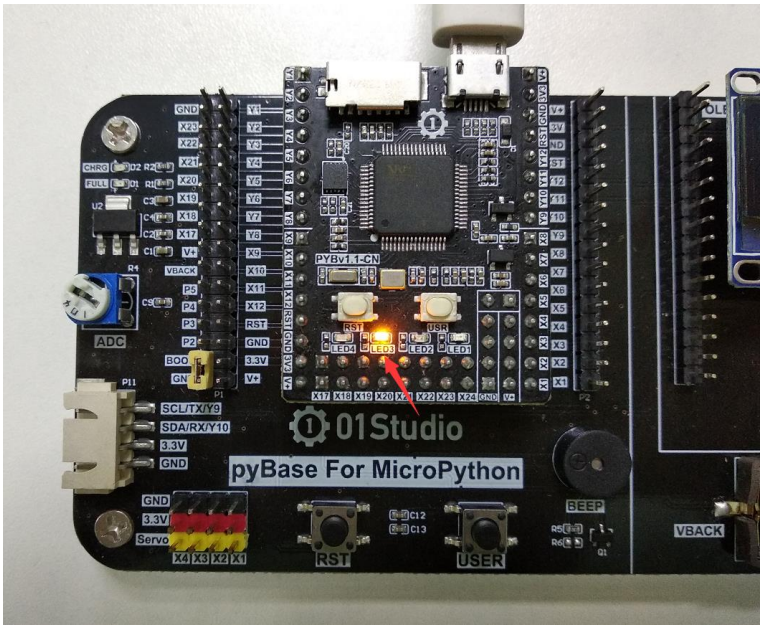

01Studio
pyBase For MicroPython
RST
USER
BEEP
VBACK
ADC
Servo

图 4-10

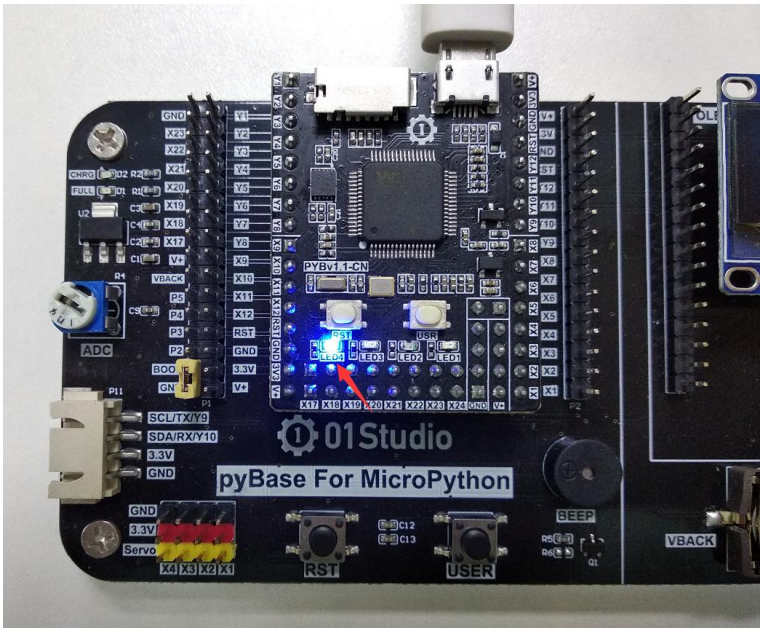


图 4-11

- **总结：**

本节实验相对于第一节相比增加了延时函数的应用，可见 MicroPython 编程功能增加就是各类函数的叠加，当开发复杂功能时候，因为底层函数已经封装好，所以面向应用的开发使用起来非常方便。除此之外我们也体验了 Python 代码的简洁和高效，使得程序更好的阅读。

4.3 按键

- **前言：**

按键是最简单也最常见的输入设备，很多产品都离不开按键，包括早期的 iphone，今天我们就来学习一下如何使用 MicroPython 来编写按键程序。有了按键输入功能，我们就可以做很多好玩的东西了。

- **实验平台：**

pyBoard v1.1-CN 或者 MicroPython 开发套件。

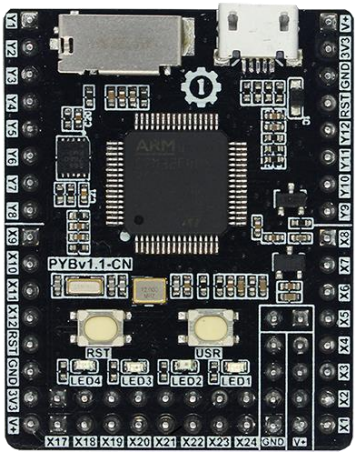


图 4-12 pyBoard v1.1-CN

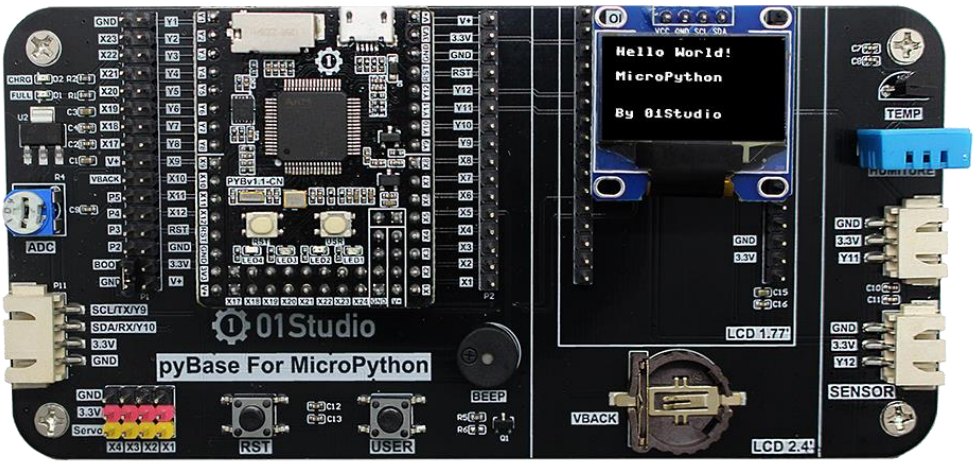


图 4-13 MicroPython 开发套件

- **实验目的：**

使用按键功能，通过检测按键被按下后，改变 LED(4)蓝灯的亮灭状态。

- **实验讲解：**

pyBoard 上有 2 个按键，RST 和 USER，RST 顾名思义是复位用的，所以真正自带可以用的就只有 1 个按键 USER。

让我们先来搞清楚 micropython 里面按键的构造函数和使用方法。

构造函数	说明
pyb.Switch()	Switch 代表了唯一的按键 USER
使用方法	**说明**
Switch.value()	读取按键状态，按下返回 True，松开返回 False
Switch.callback(fun)	当按键被按下时候执行函数 fun。

表 4-4　按键对象说明

跟之前的实验一样，首先导入 LED 和 Swtich 模块，先定义函数 fun1()用来处理 LED(4)反转。通过 callback 指令使按键被按下时候执行 fun1()函数，流程如下：

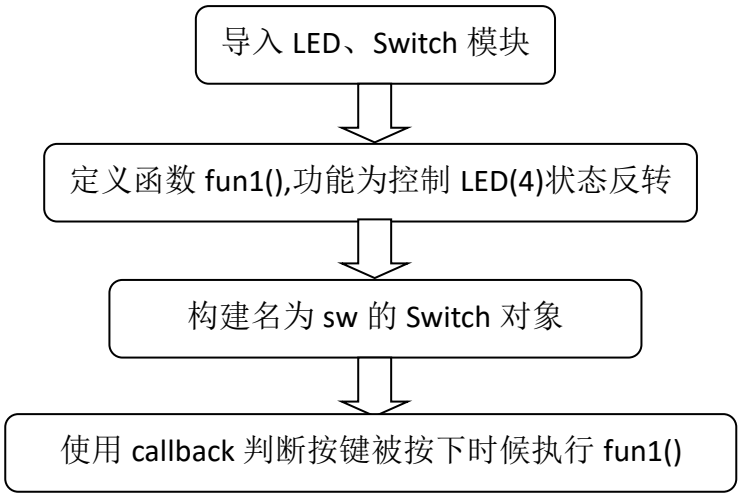


图 4-14 代码编写流程

参考程序代码如下：

```
'''
实验名称：按键
版本：v1.0
日期：2019.4
作者：01Studio
'''

from pyb import LED,Switch

def fun1():
    LED(4).toggle()

sw = Switch()     #定义按键对象名字为 sw
sw.callback(fun1) #当按键被按下时，执行函数 fun1(),即 LED(4)状态反转
```

上述代码按照编写流程来实现，实际上我们还可以再优化一下。这里的函数 fun1()要实现功能比较简单，我们可以使用 lambda 匿名函数表达式来优化代码（lambda 用法在 Python 基础知识“**第一部分　　2.4　　函数**”章节有介绍）。优化后代码如下：

```
'''
实验名称：按键
版本：v2.0
日期：2019.4
作者：01Studio
'''
```

```
from pyb import LED,Switch


sw = Switch()                               #定义按键对象名字为 sw
sw.callback(lambda:LED(4).toggle())    #当按键被按下时，LED(4)状态反转
```

我们看到只需要三行代码就实现了我们想要的功能。

- **实验结果：**

每次按下 pyBoard 上面的 USER 按键，可看到 LED（4）蓝灯的状态相应改变。

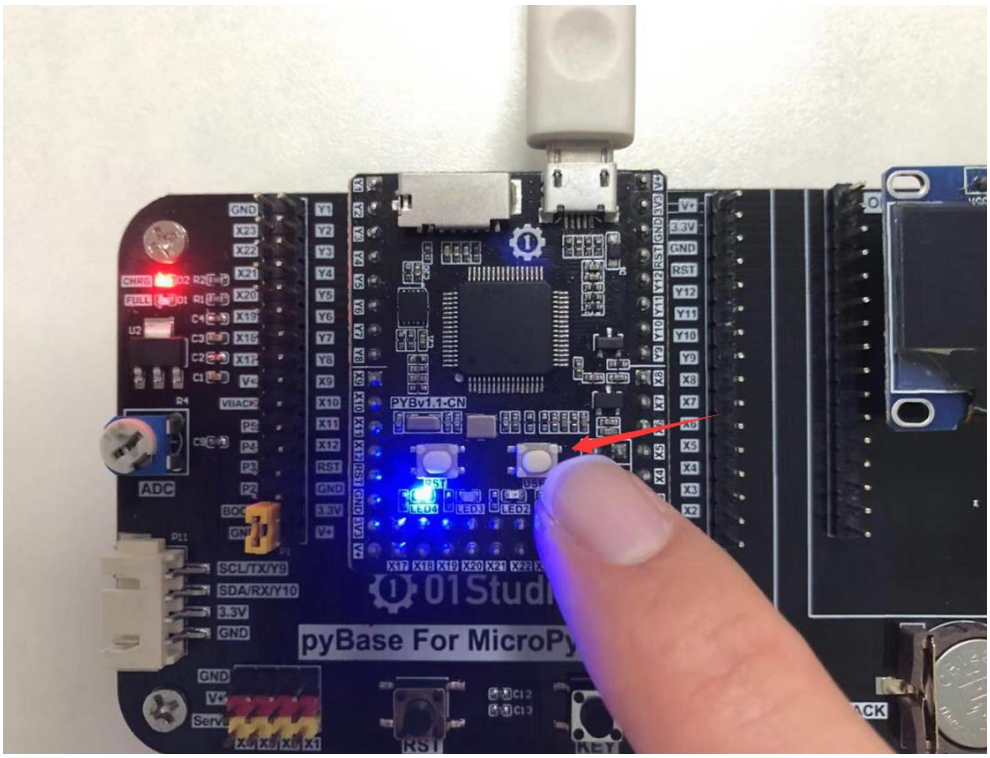


图 4-15 按键实验

- **总结：**

按键作为我们学习的第一个输入设备，有了输入设备我们就可以跟硬件做人机交互了，这对后面的学习非常有意义。按键在 MicroPython 下开发也显得很简单，我们同样可以根据自己的理解和经验来优化代码。

4.4 GPIO

- **前言：**

我们能看到 pyboard 和 MicroPython 开发套件上引出了非常多的引脚和 GPIO 口（General Purpose Input/Output，通用输入输出口），前两节的 LED 和按键实验背后原理都是使用 GPIO 来实现的，只是被提前封装好了。今天我们来做一下 GPIO 口的实验。

- **实验平台：**

pyBoard v1.1-CN 或者 MicroPython 开发套件。

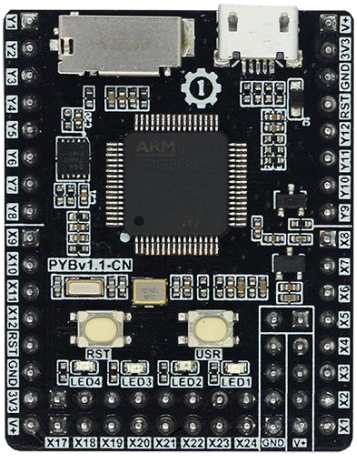


图 4-16 pyBoard v1.1-CN

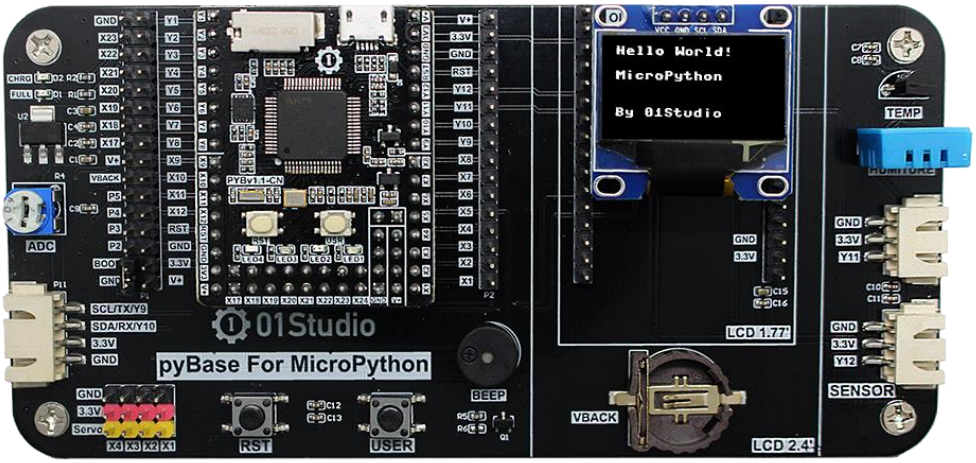


图 4-17 MicroPython 开发套件

- **实验目的：**

学会使用微处理器的 GPIO 的输入/输出，使用 GPIO 方式来控制 LED 和按键。

- **实验讲解：**

基本上 pyboard 的每个 IO 口都可以配置成特定的 GPIO 方式来进行应用。我们先来了解一下 GPIO 的构造函数和使用方法：

构造函数	说明
pyb.Pin(id,mode ,pull=Pin.PULL_NONE)	【id】 Pin 脚，如 X1,Y1 【mode】 Pin.IN: 输入 Pin.OUT_PP: 输出，带推挽 【pull】 Pin.PULL_NONE-没有上下拉电阻； Pin.PULL_UP-启用上拉电阻； Pin.PULL_DOWN-启用下拉电阻；
使用方法	**说明**
Pin.high()	引脚输出高电平
Pin.low()	引脚输出低电平
Pin.value()	在输入模式下获取引脚电平：返回 1 或者 0

表 4-5 GPIO 对象

可见 GPIO 对象使用非常简单，我们将 LED（4）即“P2”引脚配置成输出，将 USER 按键即“X17”引脚配置成输入，实现当检测到按键被按下时候点亮 LED（4），松开时关闭 LED（4）。代码编写流程如下：

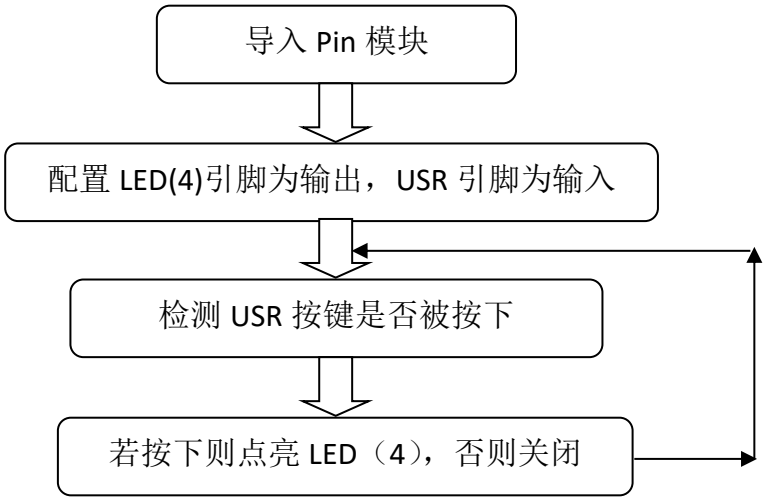


图 4-18 代码编写流程

参考程序代码如下：

```
'''
实验名称：GPIO
版本：v1.0
日期：2019-4-1
作者：01Studio
社区：www.01studio.org
'''

from pyb import Pin

#将 LED(4)-"B4"配置成推挽输出模式
p_out=Pin('B4',Pin.OUT_PP)

#将 USR 按键-"X17"配置为输入方式
p_in = Pin('X17', Pin.IN, Pin.PULL_UP)

while True:

```

```
if p_in.value()==0: #USR 被按下接地
    p_out.high()    #点亮 LED（4）蓝灯

else:
    p_out.low()     #关闭 LED（4）蓝灯
```

- **实验结果：**

程序启动后，按下 USER 按键不放，LED(4)蓝灯保持点亮，松手后熄灭

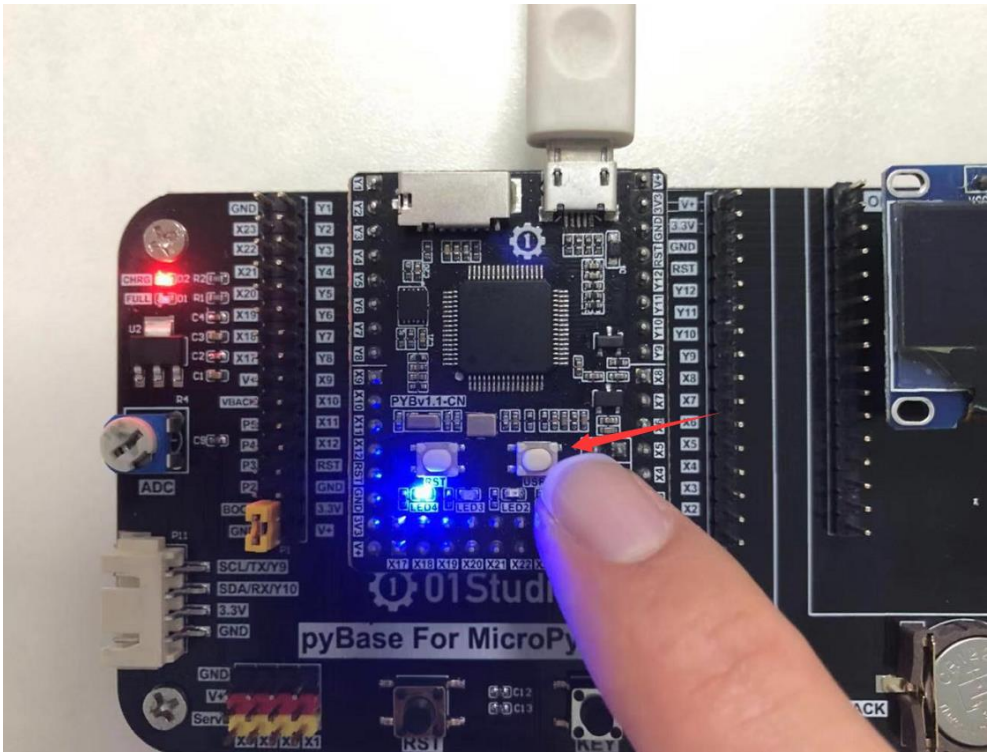


图 4-19 GPIO 实验结果

- **总结：**

GPIO 是非常通用的实验和功能，学会了 GPIO，就可以把开发板所有的引脚为自己所用，灵活性很强。

4.5 外部中断

- **前言：**

前面我们在做普通的 GPIO 时候，虽然能实现 IO 口输入输出功能，但代码是一直在检测 IO 输入口的变化，因此效率不高，特别是在一些特定的场合，比如某个按键，可能 1 天才按下一次去执行相关功能，这样我们就浪费大量时间来实时检测按键的情况。

为了解决这样的问题，我们引入外部中断概念，顾名思义，就是当按键被按下(产生中断)时，我们才去执行相关功能。中断的应用非常普遍，而 pyboard 上基本每个 IO 口都具备中断功能。

- **实验平台：**

MicroPython 开发套件。

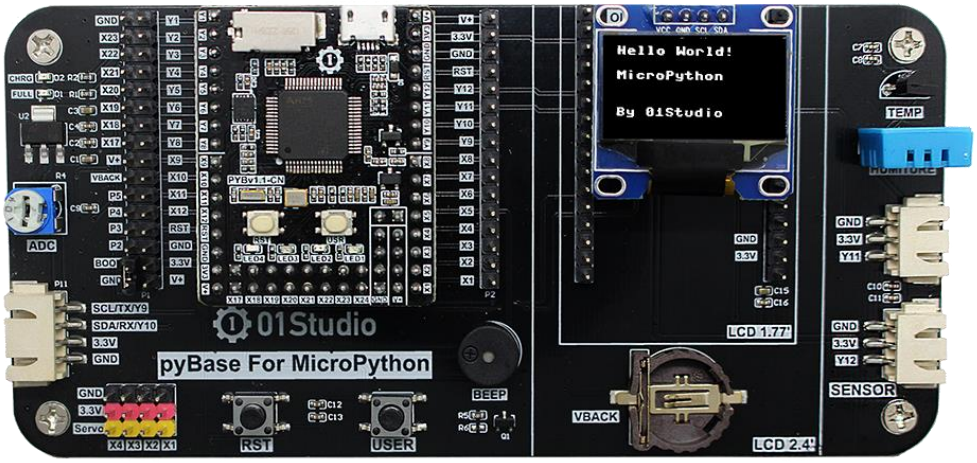


图 4-20 MicroPython 开发套件

- **实验目的：**

利用中断方式来检查 pyBase 按键 KEY 状态，被按键被按下（产生外部中断）后使 LED（4）蓝灯的状态反转。

- **实验讲解：**

pyBase 的按键 KEY 连接到 pyboard 的‘Y1’引脚，pyboard 一共有 22 个中断线，16 个来自 GPIO 引脚，剩下 6 个来源于内部。先来看看其构造函数：

构造函数	说明
pyb.ExtInt(pin,mode ,pull,callback)	【Pin】 Pin 脚，如 X1,Y1 【mode】 ExtInt.IRQ_RISING：上升沿触发 ExtInt.IRQ_FALLING：下降沿触发 ExtInt.IRQ_RISING_FALLING：上升或下降沿触发 【pull】 Pin.PULL_NONE-没有上下拉电阻； Pin.PULL_UP-启用上拉电阻； Pin.PULL_DOWN-启用下拉电阻； 【callback】中断后执行的回调函数

表 4-6 外部中断对象

我们先来了解一下上升沿和下降沿的概念，由于按键 KEY 引脚是通过按键接到 GND，也就是我们所说的低电平“0”，所以当按键被按下再松开时，引脚先获得下降沿，再获得上升沿，如下图所示：

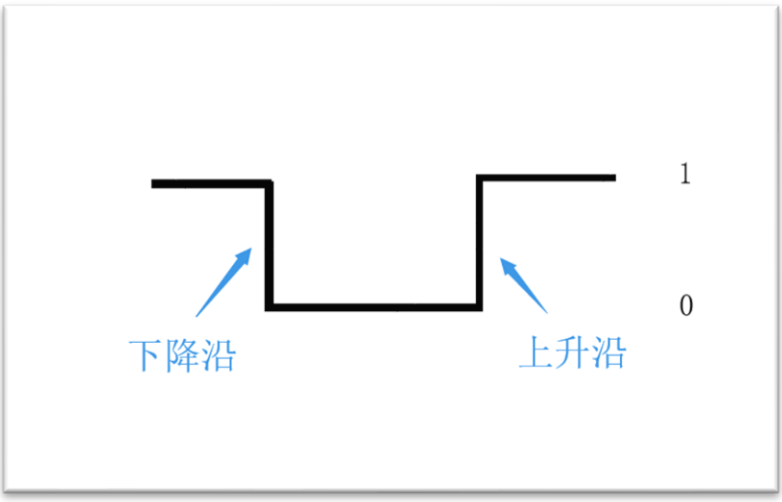


图 4-21 上升沿和下降沿

由此可见，我们可以选择下降沿方式触发外部中断，也就是当按键被按下的

时候立即产生中断。

编程思路中断跟按键章节类似，在初始化中断后，当系统检测到外部终端时候，执行 LED（4）状态反转的代码即可。

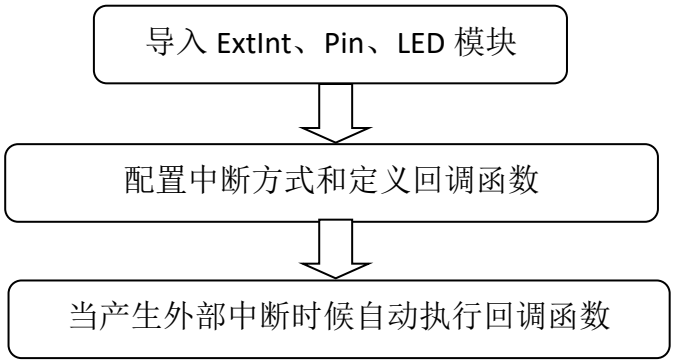


图 4-22 代码编写流程

参考程序代码如下：

```
'''
实验名称：外部中断
版本：v1.0
日期：2019-4-1
作者：01Studio
社区：www.01studio.org
'''

from pyb import Pin,ExtInt,LED

callback = lambda e: LED(4).toggle()
ext = ExtInt(Pin('Y1'), ExtInt.IRQ_FALLING, Pin.PULL_UP, callback)
#下降沿触发，打开上拉电阻
```

- **实验结果：**

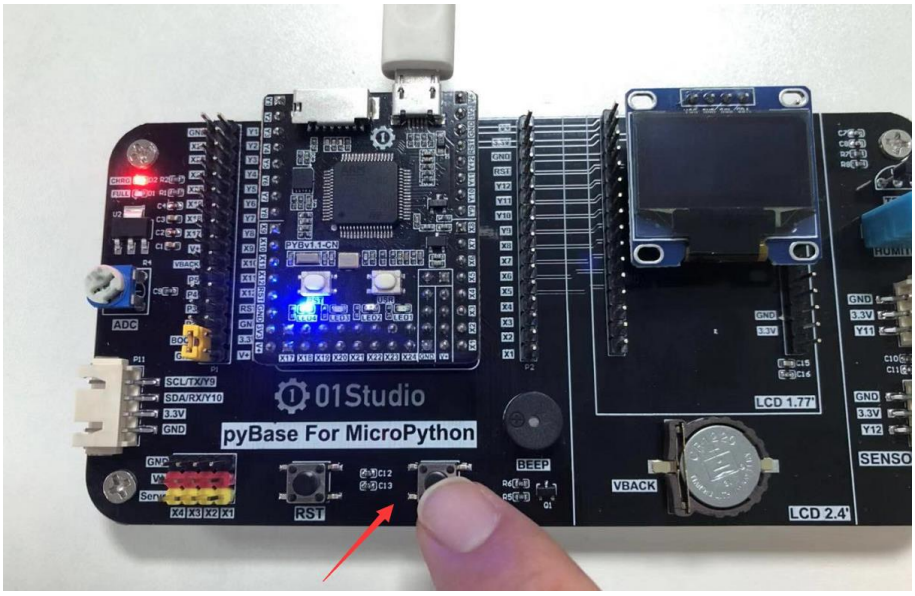


图 4-23 中断方式控制 LED（4）

- **总结：**

从参考代码来看，只是用了 3 行代码就实现了实验功能，而且相对于使用 while True 实时检测函数来看，代码的效率大大增强。外部中断的应用非常广，除了普通的按键输入和电平检测外，很大一部分输入设备，比如传感器也是通过外部中断方式来实时检测，这个在后面的章节会讲述。

4.6 I2C 总线（OLED 显示屏）

- **前言：**

在上一节学习了按键输入设备后，我们这一节先来学习输出设备 OLED 显示屏，其实之前的 LED 灯也算是输出设备，因为它们确切地告诉了我们硬件的状态。只是相对于只有亮灭的 LED 而言，显示屏可以显示更多的信息，体验更好。

本章节的 OLED 显示屏学习，实际上是在使用 I2C 的总线接口，pyBoard 是通过 I2C 总线与 OLED 显示屏通讯的。这章稍微复杂一点，我们把它放在了前面来学习，是因为学会了显示屏的使用，那么在后面的实验中可玩性就更强了。

- **实验平台：**

MicroPython 开发套件。

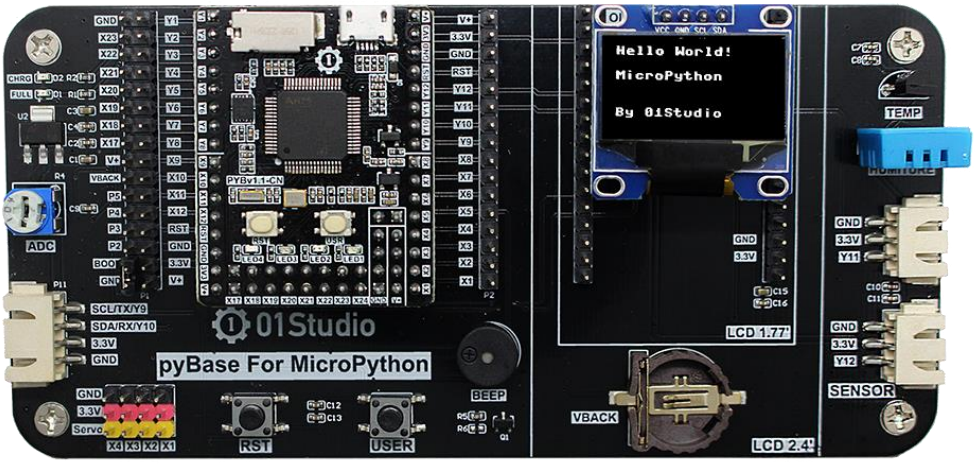


图 4-24 MicroPython 开发套件

- **实验目的：**

学习使用 MicroPython 的 I2C 总线通讯编程和 OLED 显示屏的使用。

- **实验讲解：**

什么是 I2C?

I2C 是用于设备之间通信的双线协议，在物理层面，它由 2 条线组成：SCL 和 SDA，分别是时钟线和数据线。也就是说不通设备间通过这两根线就可以进行通信。

什么是 OLED 显示屏？

OLED 的特性是自己发光，不像 TFT LCD 需要背光，因此可视度和亮度均高，其次是电压需求低且省电效率高，加上反应快、重量轻、厚度薄，构造简单，成本低等特点。简单来说跟传统液晶的区别就是里面像素的材料是由一个个发光二极管组成，因为密度不高导致像素分辨率低，所以早期一般用作户外 LED 广告牌。随着技术的成熟，使得集成度越来越高。小屏也可以制作出较高的分辨率。



图 4-25 I2C 接口的 OLED

在了解完 I2C 和 OLED 显示屏后，我们先来看看开发板的原理图，也就是 MicroPython 上的 OLED 接口是如何连线的。pyBoard 有 2 个自带 I2C 接口，但 1 个用于三轴加速度传感器（X9,X19），另外一个(Y9,Y10)保留了日后其他设备接入使用。

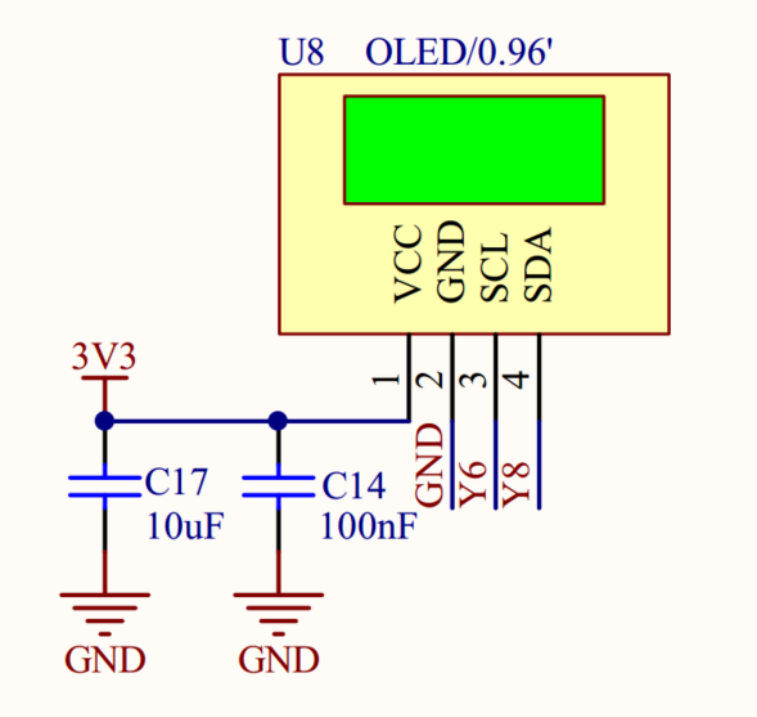


图 4-26 OLED 接口原理图

从上图可见连接到 OLED 的别是 Y8→SDA 和 Y6→SCL。尽管没有连接到自带的 I2C 接口，我们也可以通过软件模拟 I2C 方式来实现 OLED 的数据通信和使用。本例程将使用 MicroPython 的 Machine 模块来定义 Pin 口和 I2C 初始化。具体如下：

先来看看 Machine 模块里面 Pin 的用法：

构造函数	说明
machine.Pin(id,……)	id 就是指 pyboard 的引脚
使用方法	**说明**
machine.Pin(“Y8”)	建立 Pin(“Y8”对象)

表 4-7 Machine 的 Pin 对象

Machine 模块下 I2C 的用法：

构造函数	说明
machine.I2C(SDA,SCL，……)	自定义 I2C 接口。

使用方法	说明
machine.I2C.scan()	返回扫描到的 I2C 设备地址，从 0x08 至 0x77 之间
machine.I2C.write(buf)	写数据到 I2C 设备。

表 4-8 I2C 对象

定义好 I2C 后，还需要驱动一下 OLED。这里我们已经写好了 OLED 的库函数，在 ssd1306.py 文件里面。开发者只需要拷贝到 pyBoard 文件系统里面，然后在 main.py 里面调用函数即可。人生苦短，我们学会调用函数即可，也就是注重顶层的应用，想深入的小伙伴也可以自行研究 ssd1306.py 文件代码。OLED 显示屏的使用方法如下：

使用方法	说明
SSD1306_I2C(width,height,i2c,addr,……)	OELD 初始化： width:屏幕宽像素； height:屏幕高像素； i2c：已经定义的 I2C 总线对象 addr: I2C 设备地址
SSD1306_I2C.text(“string”,width,height)	将“string”在写在指定位置
SSD1306_I2C.show()	显示内容
SSD1306_I2C.fill(RGB)	清屏：RGB=0 表示黑色，RGB=1 表示白色

表 4-9 OLED 显示屏对象

学习了 Pin、I2C、OLED 对象用法后我们通过流程图来理顺一下思路：

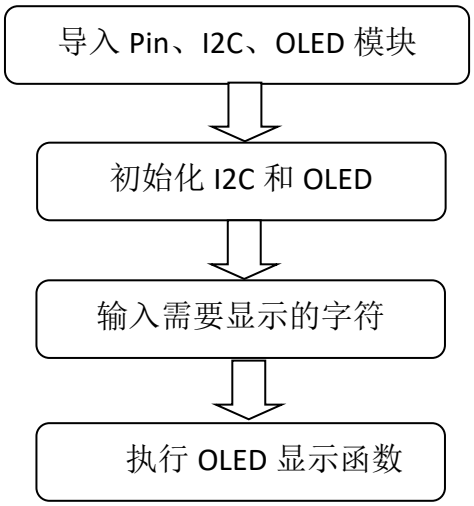


图 4-27 代码编写流程

参考代码如下：

```
'''
实验名称：OLED 显示屏（I2C 总线）
版本：v1.0
日期：2019.4
作者：01Studio
'''

from machine import I2C,Pin          #从 machine 模块导入 I2C、Pin 子模块
from ssd1306 import SSD1306_I2C     #从 ssd1306 模块中导入 SSD1306_I2C 子模块

#pyBoard I2C 初始化：sda--> Y8, scl --> Y6
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))

#OLED 显示屏初始化：128*64 分辨率, I2C 地址是 0x3c
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
oled.text("Hello World!", 0,  0)      #写入第 1 行内容
oled.text("MicroPython",  0, 20)      #写入第 2 行内容
oled.text("By 01Studio",  0, 50)      #写入第 3 行内容
```

```
oled.show()    #OLED 执行显示
```

上述代码中 OLED 的 I2C 地址是 0x3C,不同厂家的产品地址可能预设不一样，具体参考厂家的说明书。或者也可以通过 I2C.scan()来获取设备地址。

另外记得将我们提供的示例代码中的 ssd1306.py 文件拷贝到 pyboard 文件系统下，跟 main.py 保持同一个路径。

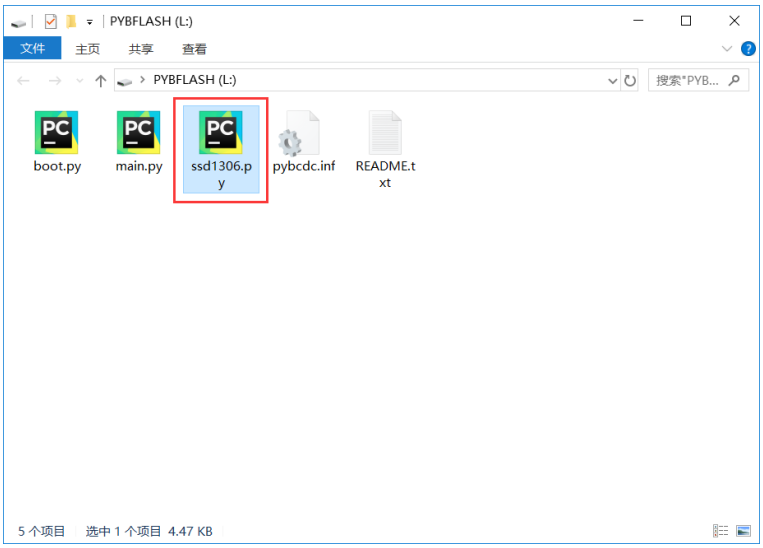


图 4-28 拷贝 ssd1306.py 文件

- **实验结果：**

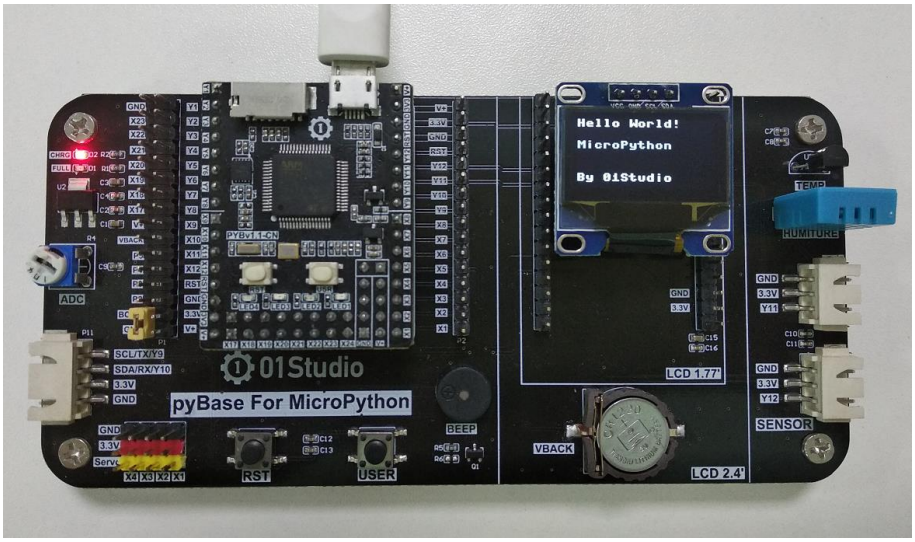


图 4-29 OLED 显示实验

- **总结：**

这一节我们学会了驱动 OLED 显示屏，换着以往如果从使用单片机从 0 开发的话你需要了解 I2C 总线原理，了解 OLED 显示屏的使用手册，编程 I2C 代码，有经验的嵌入式工程师搞不好也要弄个几天。现在基本半个小时解决问题。当然前提是别人已经给你搭好桥了，有了强大的底层驱动代码支持，我们只做好应用就好。

这一节学习的意义不仅在完成实验。在学习完 OLED 显示屏实验后，接下来我们的实验都可以使用这个 OLED 来跟用户交互了。这大大提高了实验的可观性。

4.7 RTC 实时时钟

- **前言：**

时钟可以说我们日常最常用的东西了，手表、电脑、手机等等无时无刻不显示当前的时间。

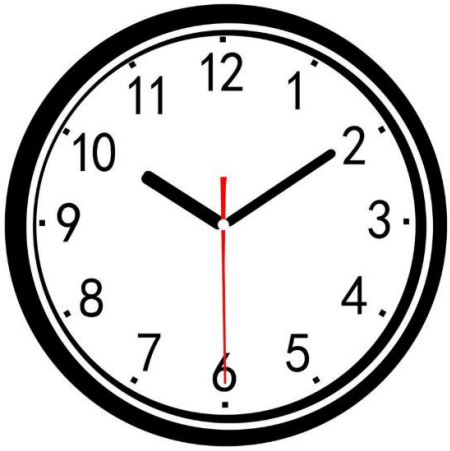


图 4-30 时钟

可以说每一个电子爱好者心中都希望拥有属于自己制作的一个电子时钟，接下来我们就用 MicroPython 开发板来制作一个属于自己的电子时钟。

- **实验平台：**

MicroPython 开发套件。

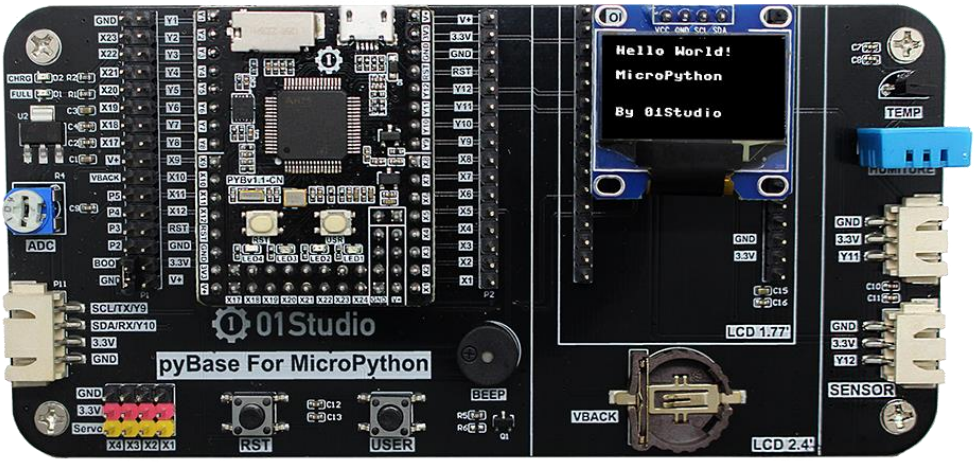


图 4-31 MicroPython 开发套件

- **实验目的：**

学习 RTC 编程和制作电子时钟，使用 OLED 显示。

- **实验讲解：**

实验的原理是读取 RTC 数据，然后通过 OLED 显示。毫无疑问，强大的 pyBoard 已经集成了内置时钟函数模块。具体如下：

构造函数	说明
pyb.RTC()	RTC()是内置的实时时钟函数模块
使用方法	说明
RTC().datetime((2019,4,1,1,0,0,0,0))	设置日期时间：顺序分别是年，月，日，星期，时，分，秒，次级秒。 星期：1-7 表示周一至周日 次级秒：255-0 倒着计数
RTC().datetime()	获取当前日期时间。

表 4-10　RTC 实时时钟对象

从上表可以看到 RTC（）的使用方法，我们需要做的就是先设定时间，然后再获取当前芯片里的时间，通过 OLED 显示屏显示，如此循环。在循环里，如果一直获取日期时间数据会造成资源浪费，所以可以每隔第一段时间获取一次数据，又由于肉眼需要看到至少每秒刷新一次即可，这里每隔 300ms 获取一次数据，所以具体流程如下：

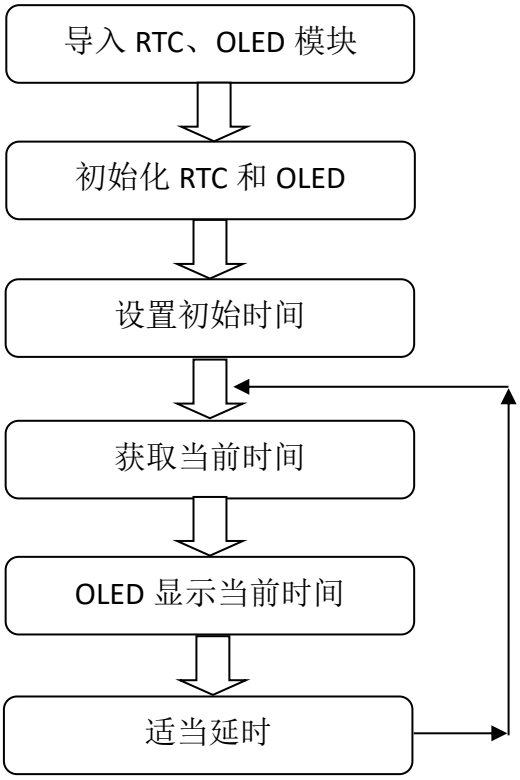


图 4-32 代码编写流程

参考代码如下：

```
'''
实验名称：RTC 实时时钟
版本：v1.0
日期：2019.4
作者：01Studio
'''

# 导入相关模块
from pyb import RTC
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C

# 定义星期和时间（时分秒）显示字符列表
```

```
week = ['Mon', 'Tues', 'Wed', 'Thur', 'Fri', 'Sat', 'Sun']
time = ['', '', '']

# 初始化所有相关对象
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
rtc = RTC()

# 首次上电配置时间，按顺序分别是：年，月，日，星期，时，分，秒，次秒级；这里做
#了一个简单的判断，检查到当前年份不对就修改当前时间，开发者可以根据自己实际情况
#来修改。
if rtc.datetime()[0] != 2019:
    rtc.datetime((2019, 4, 1, 1, 0, 0, 0, 0))

while True:

    datetime = rtc.datetime()  # 获取当前时间

    oled.fill(0)  # 清屏显示黑色背景
    oled.text('01Studio', 0, 0)    # 首行显示 01Studio
    oled.text('RTC Clock', 0, 15)  # 次行显示实验名称

    # 显示日期，字符串可以直接用“+”来连接
    oled.text(str(datetime[0]) + '-' + str(datetime[1]) + '-' +
str(datetime[2]) + ' ' + week[(datetime[3] - 1)], 0, 40)

    # 显示时间需要判断时、分、秒的值否小于 10，如果小于 10，则在显示前面补“0”以
    #达
    # 到较佳的显示效果
    for i in range(4, 7):
```

```
        if datetime[i] < 10:
            time[i - 4] = "0"
        else:
            time[i - 4] = ""

    # 显示时间
    oled.text(time[0] + str(datetime[4]) + ':' + time[1] +
str(datetime[5]) + ':' + time[2] + str(datetime[6]), 0, 55)

    oled.show()
    pyb.delay(300)  #延时 300ms
```

由于实验要用到 OLED 显示屏，所以同样别忘了将示例代码该实验文件夹下的 ssd1306.py 文件复制到 pyboard 的文件系统里面。

- **实验结果：**

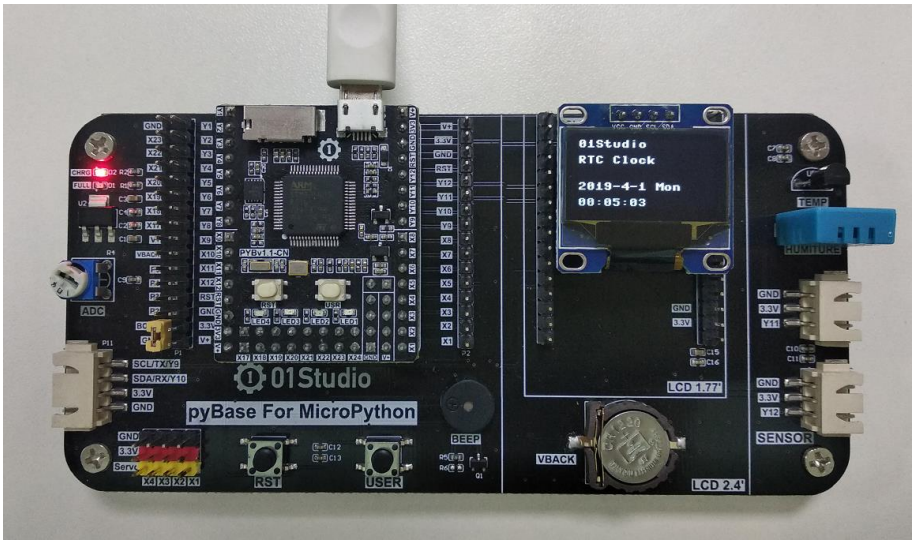


图 4-33　RTC 实时时钟实验

细心的用户或者会发现开发板的右下方有个小的纽扣电池，这个小电池的作用是后备电池，装上后开发板断电后时钟会继续跑，重新上电后时间是最新的。如果没有这电池，那么开发板断电后时钟无法继续运行。

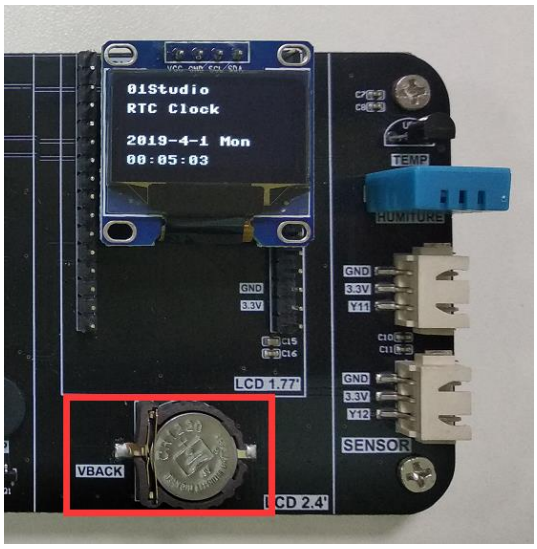


图 4-34 后备电池（VBACK）

- **总结：**

RTC 实时时钟的可玩性很强，我们还可以根据自己的风格来设定数字显示位置，以及加上一些属于自己的字符标识。打造自己的电子时钟。

4.8 ADC

- **前言：**

ADC(analog to digital conversion) 模拟数字转换。意思就是将模拟信号转化成数字信号，由于单片机只能识别二级制数字，所以外界模拟信号常常会通过 ADC 转换成其可以识别的数字信息。常见的应用就是将变化的电压转成数字信号。

- **实验平台：**

MicroPython 开发套件

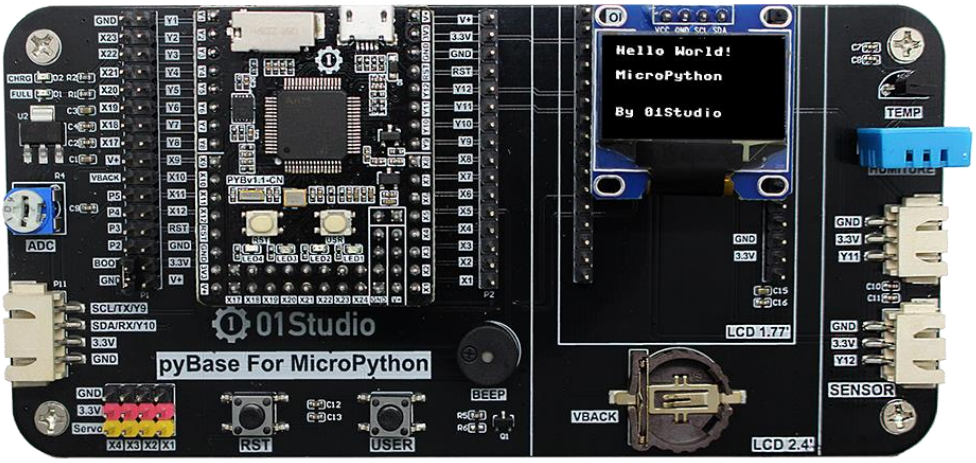


图 4-35 MicroPython 开发套件

- **实验目的：**

通过编程调用 MicroPython 的内置 ADC 函数，实现测量 0-3.3V 电压，并显示到屏幕上。

- **实验讲解：**

pyBoard 的 X7 引脚连接到了 pyBase 的电位器，通过电位器的调节可以使得 X7 引脚上的电压变化范围实现从 0-3.3V。

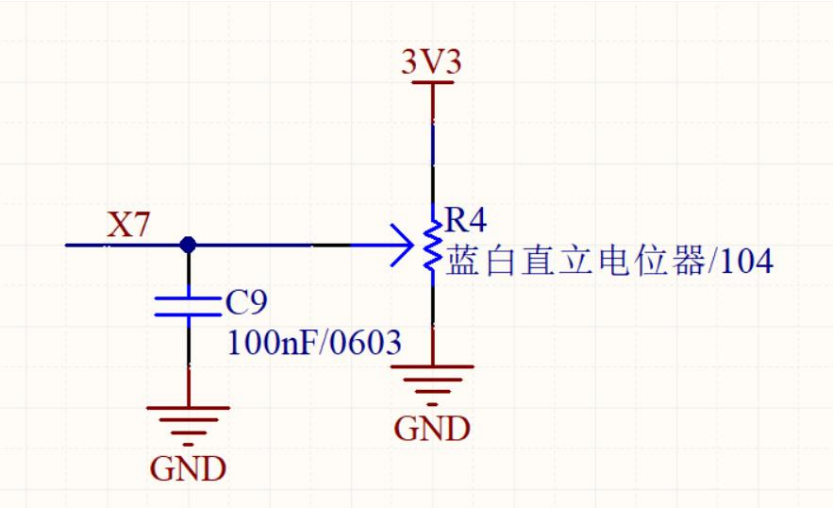


图 4-36 电位器原理图

在了解了原理图后，我们来看看内置 ADC 模块的函数使用方法。

构造函数	说明
pyb.ADC(pin)	开启某个 Pin 引脚的 ADC 功能
使用方法	**说明**
ADC.read()	读取 AD 值，返回 0-4095（0v-3.3V）;自带 AD 的测量的精度是 12 位，2^12=4096。

表 4-11 ADC 对象

你没看错，就这么简单。两句函数就可以获得 AD 数值，让我们来理顺一下编程逻辑。先导入相关模块，然后初始化模块。在循环中不断读取 ADC 的值，转化成电压值后在 OLED 上面显示，每隔 1 秒读取一次，具体如下：

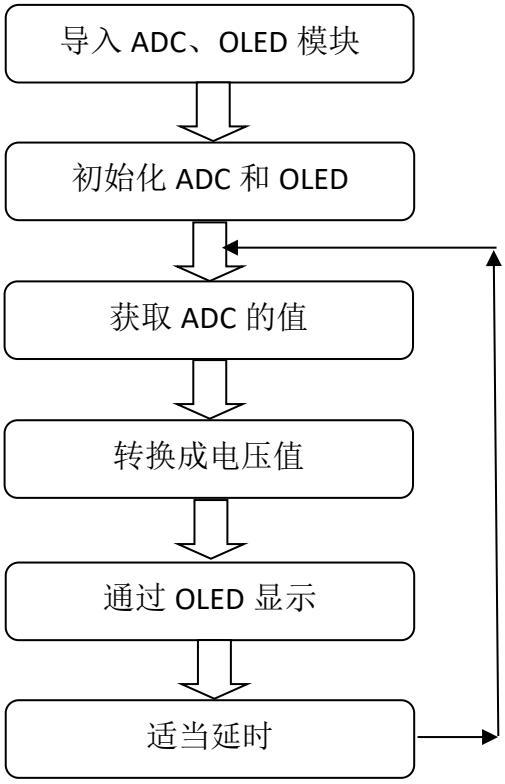


图 4-37 代码编写流程

实验参考代码：

```
'''
实验名称：ADC-电压测量
版本：v1.0
日期：2019.4
作者：01Studio
说明：通过对 ADC 数据采集，转化成电压在显示屏上显示。ADC 精度 12 位，电压 0-
3.3V。
'''

#导入相关模块
import pyb
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C
```

```
#初始化相关模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
adc = pyb.ADC('X7') #Pin='X7'

while True:

   oled.fill(0)  # 清屏显示黑色背景
   oled.text('01Studio', 0, 0)  # 首行显示 01Studio
   oled.text('ADC', 0, 15)      # 次行显示实验名称

   #获取 ADC 数值
   oled.text(str(adc.read()),0,40)
   oled.text('(4095)',40,40)

   #计算电压值，获得的数据 0-4095 相当于 0-3V，（'%.2f'%）表示保留 2 位小数
   oled.text(str('%.2f'%(adc.read()/4095*3.3)),0,55)
   oled.text('V',40,55)

   oled.show()
   pyb.delay(1000)
```

- **实验结果：**

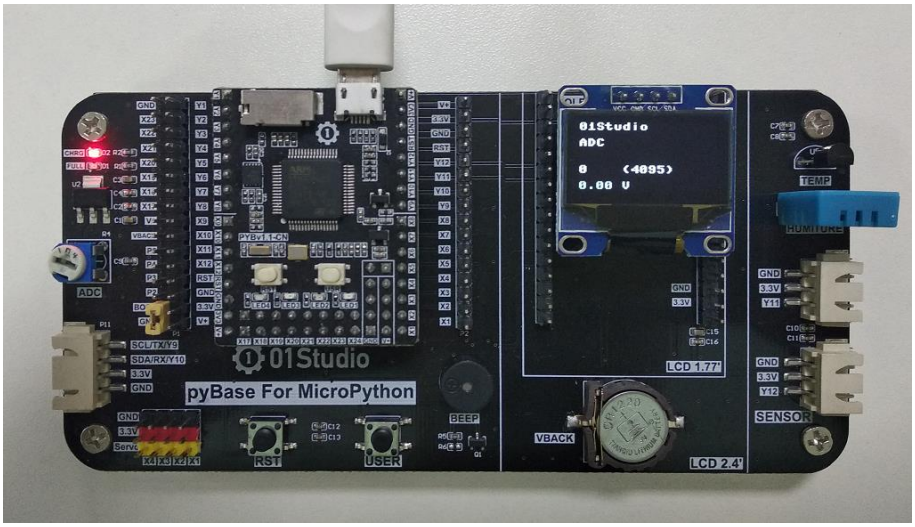


图 4-38 电位器顺时钟拧到尽头是 0V

通过调节电位器，可以发现电压在不断变化。

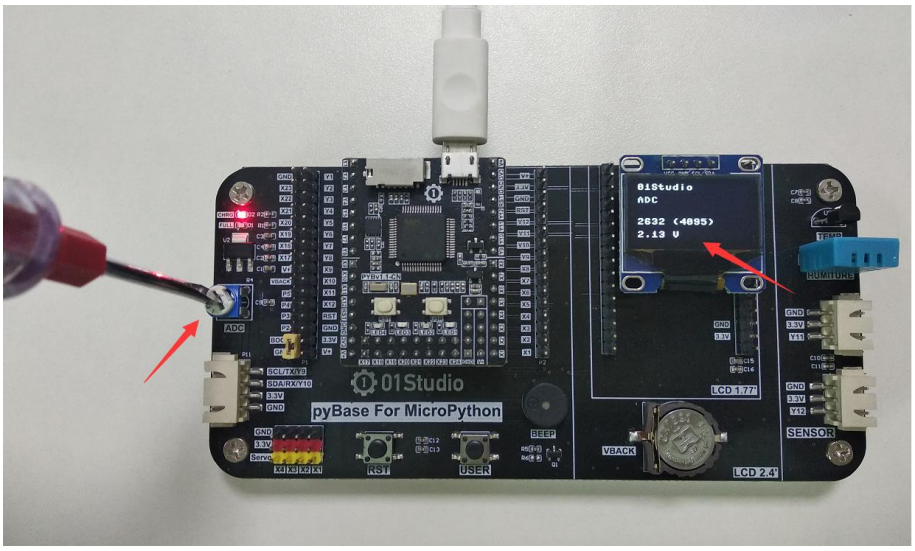


图 4-39 调节电位器来改变电压输入

- **总结：**

这一节我们学习了 ADC 的应用。主要用于电压的检测，pyboard 上有很多 ADC 引脚，这意味着可以实现多路同时检测。有兴趣的小伙伴可以自行研究。

4.9 DAC

- **前言：**

DAC（**Digital-to-Analog Converter**）数字模拟转换器。我们前面做的 ADC 实验，是将模拟信号转化为了数字信号。这一节做的实验刚好是反转，我们将特定的数字信号通过 DAC 输出。输出的形式多种，如特定电压、信号波形（正弦波、方波、三角波等），有条件的用户可以用示波器来观看输出。本章节利用开发板上的无源蜂鸣器，通过 DAC 输出不同的频率来改变其音色。

- **实验平台：**

MicroPython 开发套件。

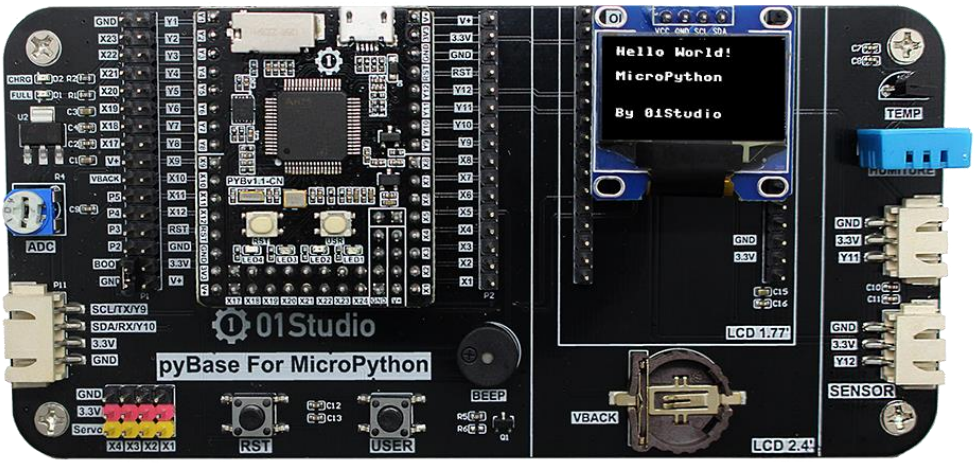


图 4-40 MicroPython 开发套件

- **实验目的：**

通过 DAC 输出不同频率的方波来驱动蜂鸣器。

- **实验讲解：**

蜂鸣器分有源蜂鸣器和无源蜂鸣器，有源蜂鸣器的使用方式非常简单，只需要接上电源，蜂鸣器就发声，断开电源就停止发声。而本实验用到的无源蜂鸣器，是需要给定指定的频率，才能发声的，而且可以通过改变频率来改变蜂鸣器的发声音色，以此来判定 pyBoard 的 DAC 输出频率是在变化的。

PyBoard 上有 2 个 DAC 引脚，分别是 X5 和 X6，其中 X5 连接了无源蜂鸣器。如下图所示：

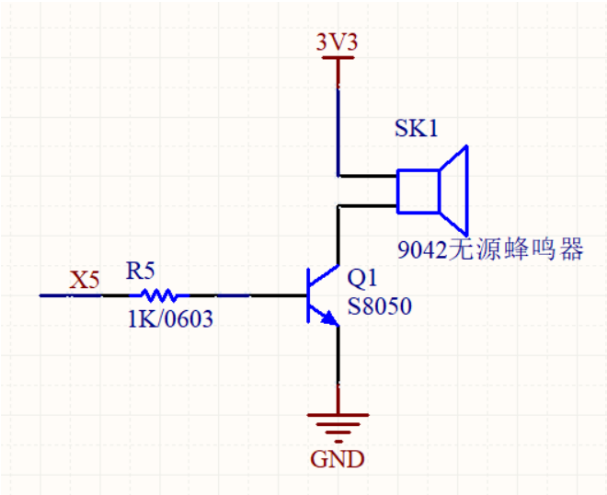


图 4-41 蜂鸣器原理图

在了解了原理图后，我们来看看内置 ADC 模块的函数使用方法。

构造函数	说明
pyb.DAC(port,bits=8)	Port 可以是 1 或者 2，分别对应 X5 和 X6；bits（精度）可以是 8 位或者 12 位。
使用方法	**说明**
DAC.noise(freq)	产生特定频率的噪声信号
DAC.triangle(freq)	产生特定频率三角波信号
DAC.write(value)	输出特定电压(0-3.3V)。value 的值位 0-2^bits-1（例：8bits 那么范围就是 0-255）
DAC.write_timed(data,freq,mode)	将 data(字节数组)的内容以特定频率输出；mode 分别是 DAC.NORMAL 或 DAC.CIRCULAR(循环模式)

表 4-12 DAC 对象

无源蜂鸣器我们可以用特定频率的方波来驱动，方波的原理很简单，就是一定频率的高低电平转换，使用 DAC.write(0)和 DAC.write(255)交替输出即可，

可以通过延时函数来控制输出变换的次数。实际过程中我们没必要用低效率的延时函数，因为看到 DAC 模块里面已经定义好了特定格式输出的函数 **DAC.write_timed(data,freq,mode)**，将 mode 设定成循环模式，就可以循环不断地输出指定频率的信号了。

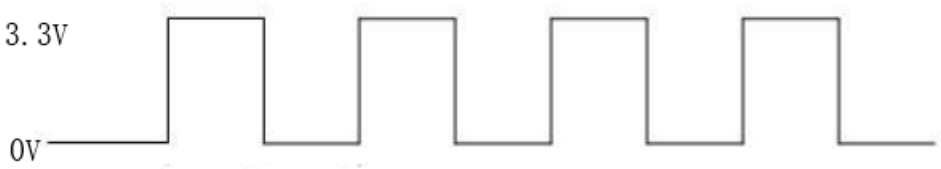


图 4-42 方波信号

结合我们前面的按键实验，我们可以通过 USER 按键按下来切换输出频率，并在 OLED 上直观体验。具体流程如下：

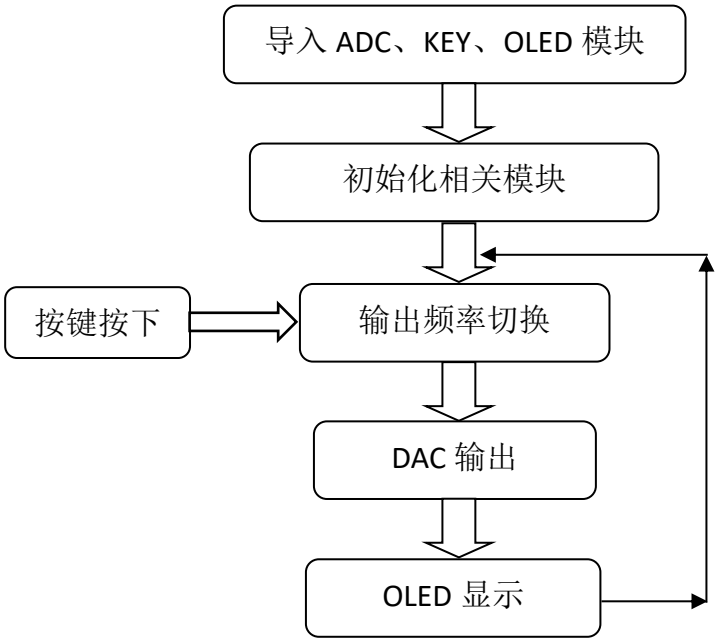


图 4-43 代码编程流程

参考代码如下：

```
'''
实验名称：DAC-蜂鸣器
版本：v1.0
日期：2019.4
作者：01Studio
说明：通过 USER 按键让 DAC 输出不同频率的方波来驱动蜂鸣器。
'''

#导入相关模块
from pyb import DAC,Switch
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C

#初始化相关模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

sw = Switch()     #定义按键对象名字为 sw
dac = DAC(1)     #定义 DAC 对象名字为 dac，输出引脚为 X5

#定义 4 组频率值：1Hz、200Hz、1000Hz、5000Hz
freq=[1,200,1000,5000]

# 定义 8 位精度下方波的值。0、255 分别对应输出 0V、3.3V。需要定义成字节数组。
buf = bytearray(2)
buf[0]=0
buf[1]=255

key_node = 0  #按键标志位
```

```
i = 0         #用于选择频率数组

################################################
#  按键和其回调函数
################################################
def key():
    global key_node
    key_node = 1

sw.callback(key)  #当按键被按下时，执行函数 key()

################################################
#  OLED 初始显示
################################################
oled.fill(0)  # 清屏显示黑色背景
oled.text('01Studio', 0, 0)  # 首行显示 01Studio
oled.text('DAC-Beep', 0, 15)  # 次行显示实验名称
oled.text('Pls Press USER', 0, 40)  # 显示当前频率
oled.show()

while True:

    if key_node==1: #按键被按下
        i = i+1
        if i == 4:
            i = 0
        key_node = 0 #清空按键标志位

        #DAC 输出指定频率
        dac.write_timed(buf, freq[i]*len(buf), mode=DAC.CIRCULAR)
```

```
#显示当前频率
oled.fill(0)  # 清屏显示黑色背景
oled.text('01Studio', 0, 0)  # 首行显示 01Studio
oled.text('DAC-Beep', 0, 15)  # 次行显示实验名称
oled.text(str(freq[i]) + 'Hz', 0, 40)  # 显示当前频率
oled.show()
```

以上代码中 DAC 的函数使用非常简单，但为了让实验能更直观，加入按键和 OLED 模块后需要更多的代码来支持，但编程逻辑不变。以上代码有几个关键的地方需要补充说明一下：

（1） 由于底层独特的操作方式，使得输入的数据 buf 必须是字节数组，所以需要加入 bytearray()来定义。

（2） Key_node 是全局变量，因此在 key()函数里面用该变量必须添加 global key_node 代码，否则会在函数里面新建一个样的变量。

（3） 变量 i 的作用是为了让 DAC 选择指定的频率数组 freq 输出

- **实验结果：**

将实验代码文件拷贝到 pyboard 文件系统，红灯从亮到灭更新完固件后按下 RST 按键重启。发现 OLED 显示让我们按下 USER 键。

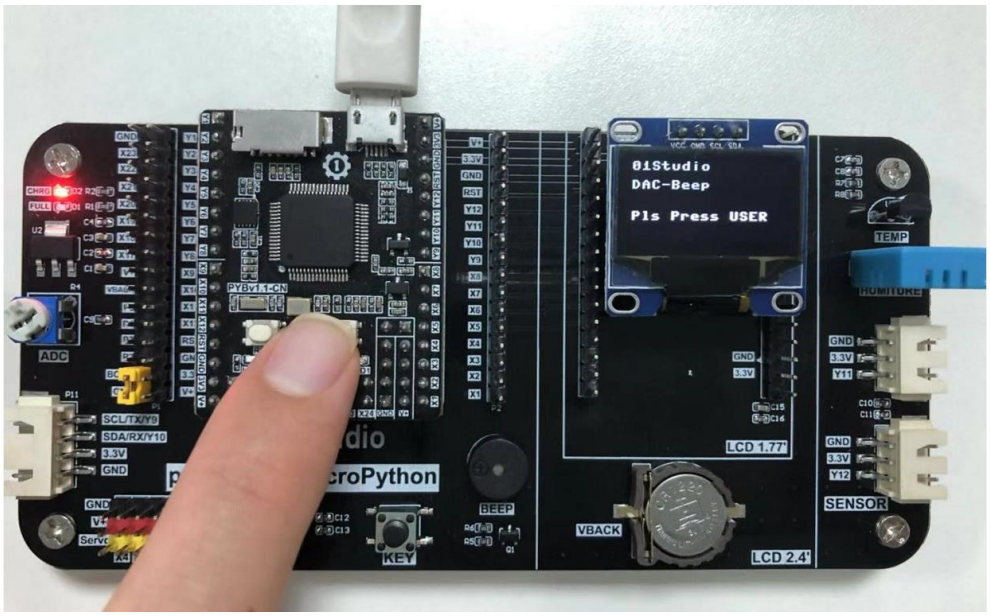


图 4-44 按下 USER 按键开始

每次按下后发现 OLED 上的频率示意发生变化，蜂鸣器的叫声也跟着变化。

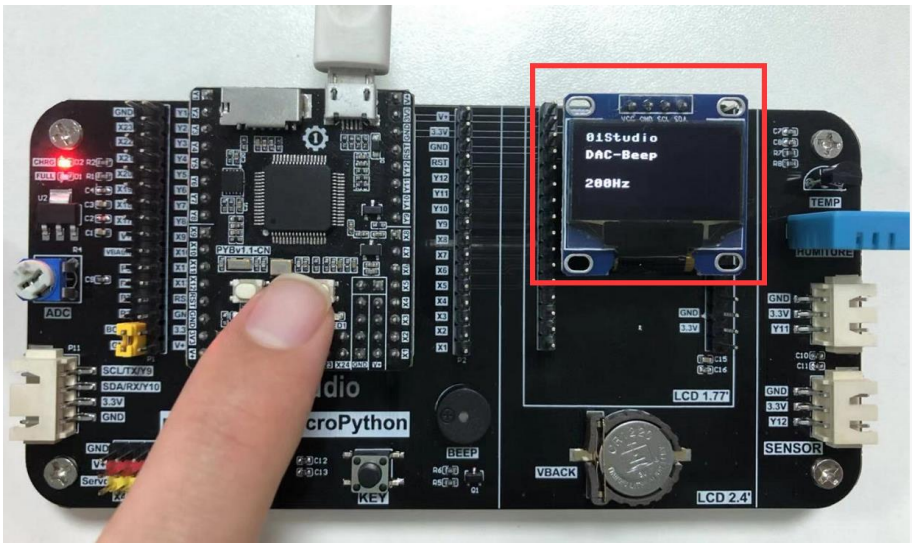


图 4-45 通过按键切换输出频率

有条件的朋友可以使用示波器连接 X5，观察信号波形的变化：

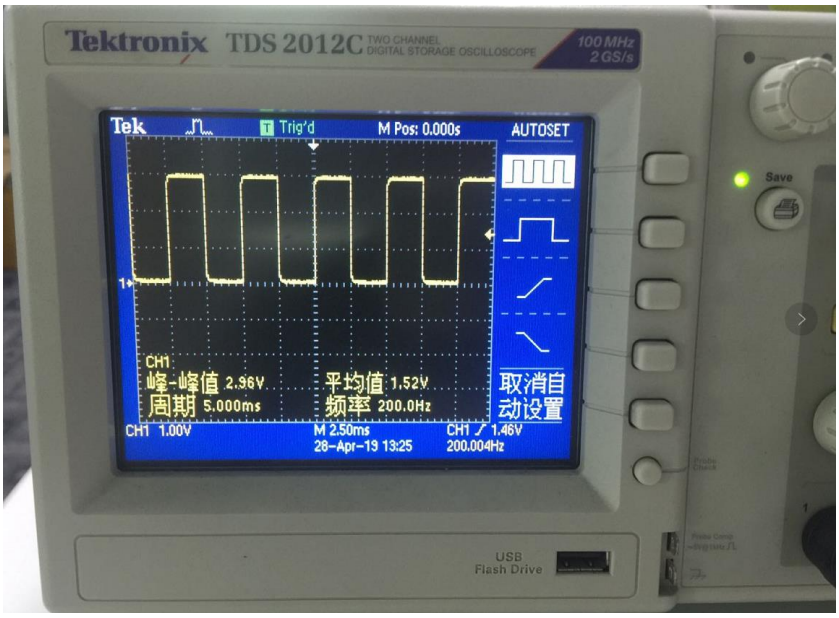


图 4-46 示波器显示

- **总结：**

到了这一节，我们发现实验对象函数使用方法非常简单，而随着要实现功能的复杂化让编程的代码数量变多，逻辑也略显复杂。这是好事，让我们可以将更多精力放在应用上，做出更多好玩的创意。而不需要过多的关注复杂的底层代码开发。

4.10 三轴加速度计

- **前言：**

三轴加速度传感器用途广泛，可以计算物体倾斜角度，加速度以及通过算法来计算步数，比如日常用的手环计步。Pyboard 上集成了一款三轴加速度计芯片（MMA7660），可以直接通过编程使用。

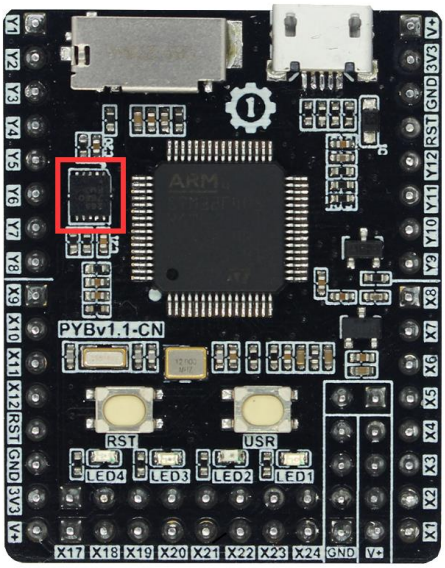


图 4-47 三轴加速度计芯片

- **实验平台：**

MicroPython 开发套件。

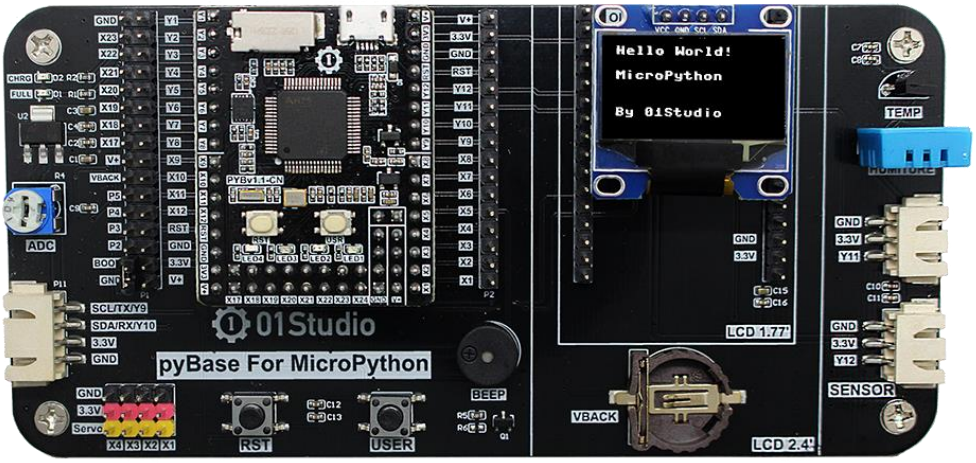


图 4-48 MicroPython 开发套件

- **实验目的：**

了解三轴加速传感器的工作原理，通过编程获取其各个方向的数值（X 轴、Y 轴、Z 轴）并在 OLED 上显示。

- **实验讲解：**

MMA7660 三轴加速度计的使用原理很简单，就是在 x、y、z 各个方向根据受力情况通过数据方式呈现。测量结果范围是-32 至 31，大约为-1.5g 至 1.5g（g 为重力加速度，9.8m/s^2）。我们只需要知道以上 3 个方向的值，便可以计算出各个方向的加速度。

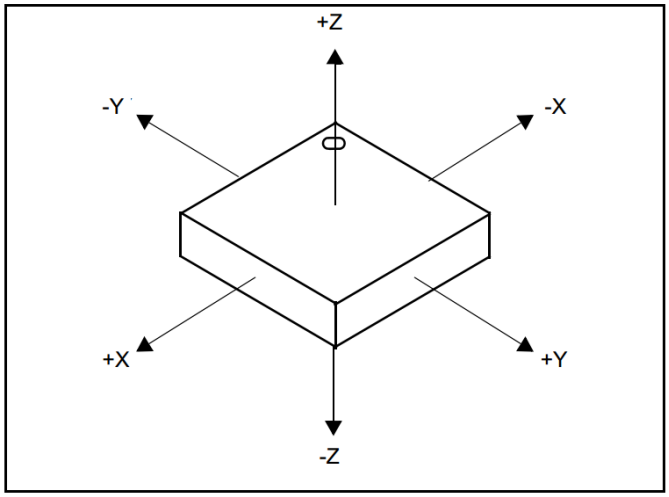


图 4-49 三轴加速度计原理

我们来了解一下加速度计的构造函数和使用方法：

构造函数	说明
pyb.Accel	加速度计构造
使用方法	**说明**
Accel.x()	获取 X 轴上的值
Accel.y()	获取 y 轴上的值
Accel.z()	获取 z 轴上的值

表 4-13

我们循环 500ms 采集一次，编程逻辑如下：

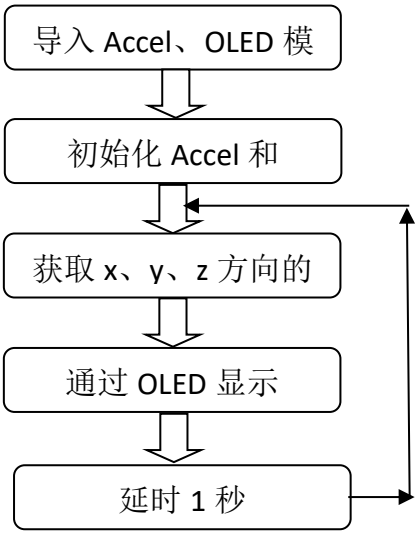


图 4-50 编程流程图

参考代码：

```
'''
实验名称：DAC-蜂鸣器
版本：v1.0
日期：2019.4
作者：01Studio
说明：通过编程获取其各个方向的数值（X 轴、Y 轴、Z 轴）并在 OLED 上显示。
'''

import pyb
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C

#初始化相关模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
```

```
accel = pyb.Accel()


while True:


   oled.fill(0) #清屏
   oled.text('01Studio', 0, 0)
   oled.text('Accel test:',0,15)

   #获取 x,y,z 的值并显示
   oled.text('X:'+str(accel.x()),0,40)
   oled.text('Y:'+str(accel.y()),44,40)
   oled.text('Z:'+str(accel.z()),88,40)
   oled.show()

   pyb.delay(1000) #延时 1s
```

- **实验结果：**

运行后见到 OLED 显示 X/Y/Z 的值，用户可以根据翻转、抖动开发板等方式来观察 X/Y/Z 的值的事实变化情况。

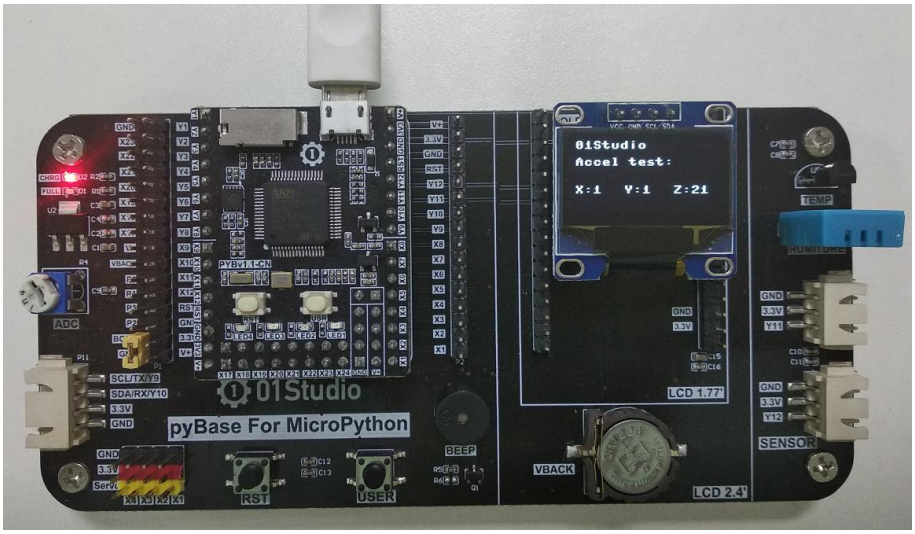


图 4-51

细心的用户或许会发现，当开发板静止在桌面水平面上，Z 的值始终保持在 20-21 左右。查看 MMA7660 芯片手册，看到 0.98g 也就是重力加速度的值约为 21，而板子平放静止的时候 Z 轴有个重力加速的，这不难理解。

6-bit result	Binary	2's Comp	g value	Angle X or Y		Angle Z
0	0	0	0.000g	0.00°		90.00°
1	1	1	0.047g	2.69°		87.31°
2	10	2	0.094g	5.38°		84.62°
3	11	3	0.141g	8.08°		81.92°
4	100	4	0.188g	10.81°		79.19°
5	101	5	0.234g	13.55°	Z-axis must be in the range	76.45°
6	110	6	0.281g	16.33°		73.67°
7	111	7	0.328g	19.16°		70.84°
8	1000	8	0.375g	22.02°		67.98°
9	1001	9	0.422g	24.95°		65.05°
10	1010	10	0.469g	27.95°		62.05°
11	1011	11	0.516g	31.04°		58.96°
12	1100	12	0.563g	34.23°		55.77°
13	1101	13	0.609g	37.54°		52.46°
14	1110	14	0.656g	41.01°		48.99°
15	1111	15	0.703g	44.68°		45.32°
16	10000	16	0.750g	48.59°		41.41°
17	10001	17	0.797g	52.83°		37.17°
18	10010	18	0.844g	57.54°		32.46°
19	10011	19	0.891g	62.95°		27.05°
20	10100	20	0.938g	69.64°		20.36°
21	10101	21	0.984g	79.86°		10.14°
22	10110	22	1.031g			

图 4-52 重力加速度对应的值约为 21

- **总结：**

在嵌入式编程中，有些时候我们需要学看芯片手册，手册不用全看，可以锻炼到的是学会从芯片手册中寻找关键信息，以理解实际产品的基本参数。

4.11 UART（串口通信）

- **前言：**

串口是非常常用的通信接口，有很多工控产品、无线透传模块都是使用串口来收发指令和传输数据，这样用户就可以在无须考虑底层实现原理的前提下将各类串口功能模块灵活应用起来。

- **实验平台：**

MicroPython 开发套件或 pyBoard v1.1-CN。

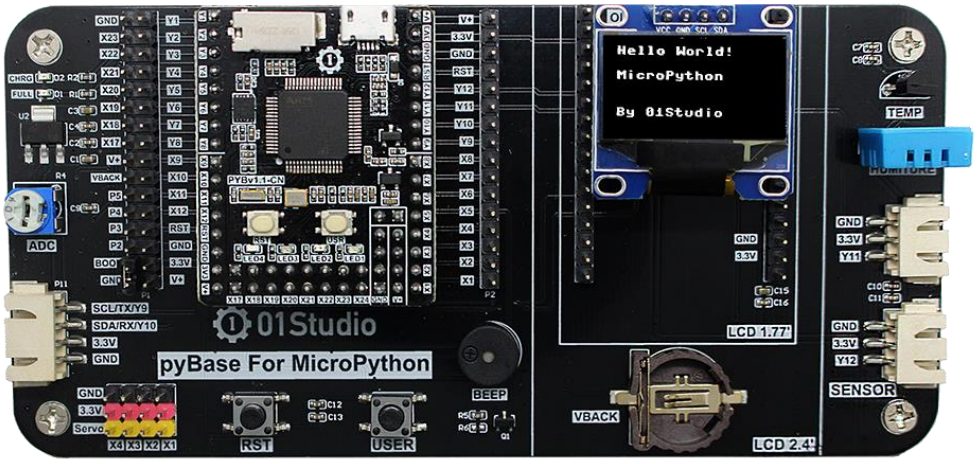


图 4-53 MicroPython 开发套件

- **实验目的：**

编程实现串口收发数据。

- **实验讲解：**

pyBoard v1.1-CN 一共有 6 个串口，编号是 1-6，其中串口 5 用于 REPL 调试，因此并没有开放使用。串口号和引脚关系如下

UART(1)	TX	X9
	RX	X10
UART(2)	TX	X3
	RX	X4

UART(3)	TX	Y9
	RX	Y10
UART(4)	TX	X1
	RX	X2
UART(6)	TX	Y1
	RX	Y2

表 4-14　pyBoard 串口引脚资源

我们来了解一下 pyb 串口对象的构造函数和使用方法：

构造函数
uart=pyb.UART.init(baudrate, bits=8, parity=None, stop=1, *, timeout=0, flow=0, timeout_char=0, read_buf_len=64)
创建 UART 对象。 【id】1-6（5 已用于 REPL，不能使用） 【baudrate】波特率，常用 115200、9600 【bits】数据位 【parity】校验；默认 None, 0(偶校验)，1(奇校验) 【stop】停止位，默认 1 【flow】流控 …
使用方法
uart.deinit()
关闭串口
uart.any()
返回等待读取的字节数据，0 表示没有
uart.read([*nbytes*])
【nbytes】读取字节数
UART.readline()
读行

UART.write(*buf*)
【buf】串口 TX 写数据
*更多使用说明请阅读官方文档： http://docs.micropython.01studio.org/zh_CN/latest/library/pyb.UART.html#pyb-uart

表 4-15　PYB 的 UART 对象

我们可以用一个 USB 转 TTL 工具，配合电脑上位机串口助手来跟 MicroPython 开发板模拟通信。

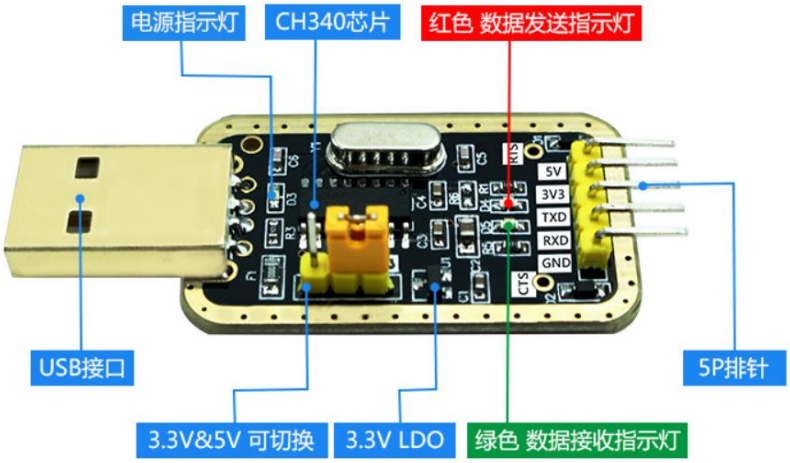


图 4-54 常用 USB 转串口工具

注意要使用 3.3V 电平的 USB 转串口 TTL 工具，本实验我们使用串口 3，也就是 Y9（TX）和 Y10（RX），接线示意图如下：



图 4-55 串口接线图

在本实验中我们可以先初始化串口，然后给串口发去一条信息，这样 PC 机的串口助手就会在接收区显示出来，然后进入循环，当检测到有数据可以接收时候就将数据接收并打印，并通过 REPL 打印显示。代码编写流程图如下：

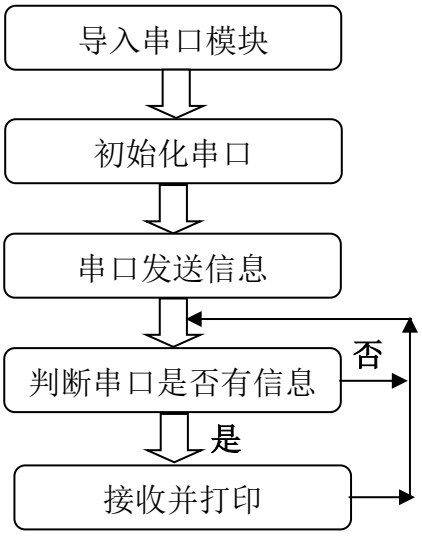


图 4-56 代码编程流程图

参考代码：

```
'''
实验名称：串口通信
版本：v1.0
日期：2019.10
作者：01Studio
说明：通过编程实现串口通信，跟电脑串口助手实现数据收发。
'''

#导入串口模块
from pyb import UART

uart=UART(3,115200) #设置串口号 3 和波特率,TX--Y9,RX--Y10
```

```
uart.write('Hello 01Studio!')#发送一条数据

while True:

    #判断有无收到信息
    if uart.any():
        text=uart.read(64) #默认单次最多接收 64 字节
        print(text) #通过 REPL 打印串口 3 接收的数据
```

- **实验结果：**

我们按照上述方式将 USB 转 TTL 的 TX 接到 Y10，RX 接到 Y9。GND 接一起，3.3V 可以选择接或不接。

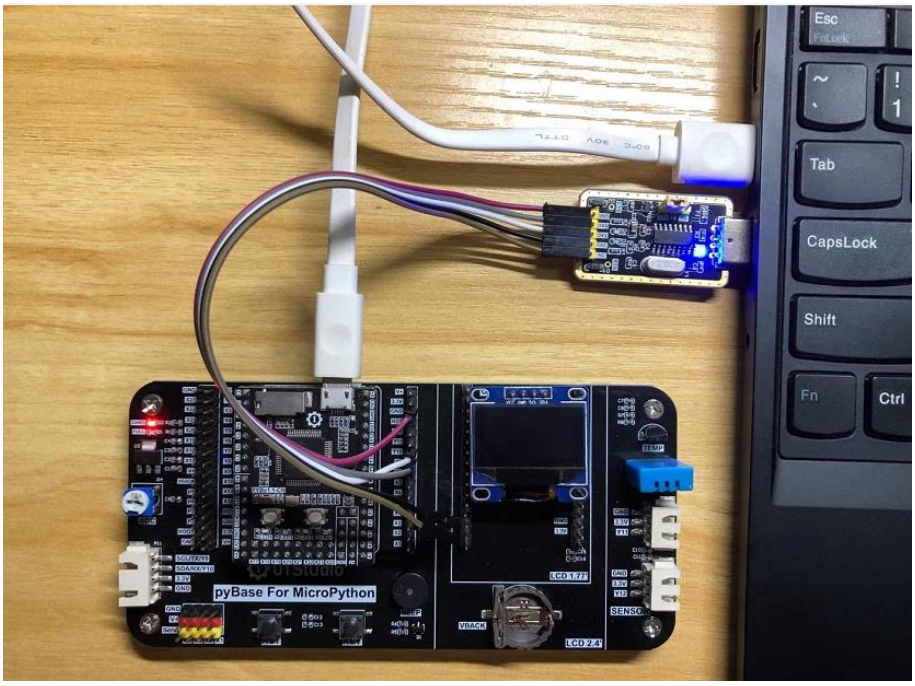


图 4-57 将 pyboard 和 usb ttl 工具同时接入电脑

这时候打开电脑的设备管理器，能看到 2 个 COM。写着 CH340 的是串口工具，另外一个则是 pyboard 的 REPL。如果 CH340 驱动没安装，则需要手动安装，

驱动在：配套资料包→开发工具→windows→串口终端→CH340 文件夹下。

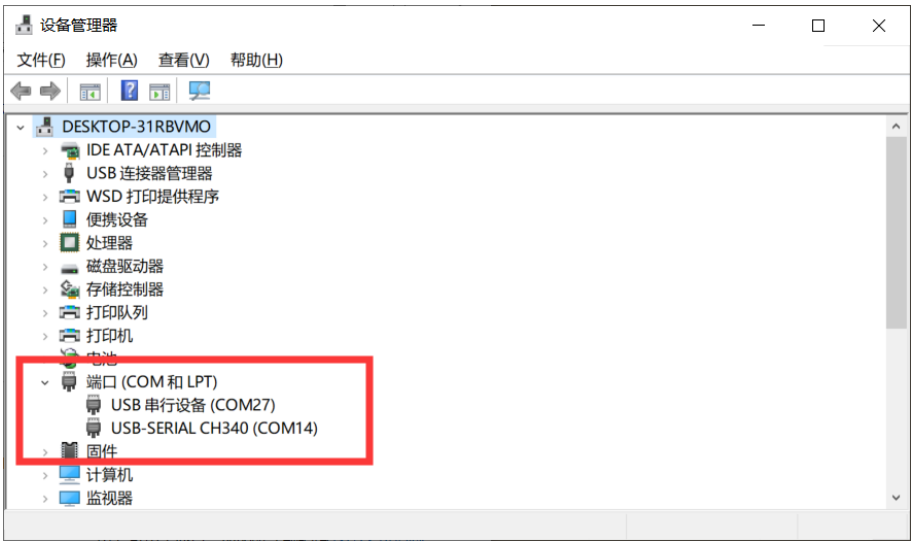


图 4-58

本实验要用到串口助手，打开配套资料包→开发工具→windows→串口终端工具下的【UartAssist.exe】软件。

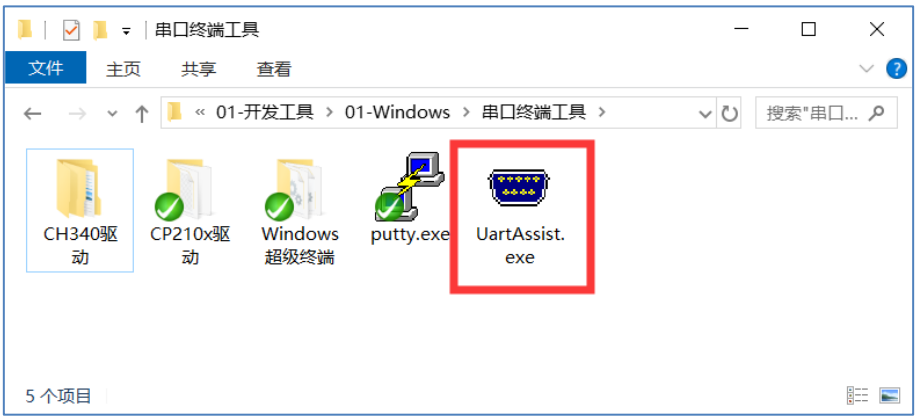


图 4-59

根据上图设备管理器里面的信息，将串口工具配置成 COM14，REPL 串口配置成 COM27（根据自己的串口号调整）。波特率 115200。运行程序，可以看到一开始串口助手收到 pyboard 上电发来的信息“Hello 01Studio!”。我们在串口助手的发送端输入“http://www.01studio.org”， 点击发送，可以看到 pyboard 在接收到该信息后在 REPL 里面打印了出来。如下图所示：

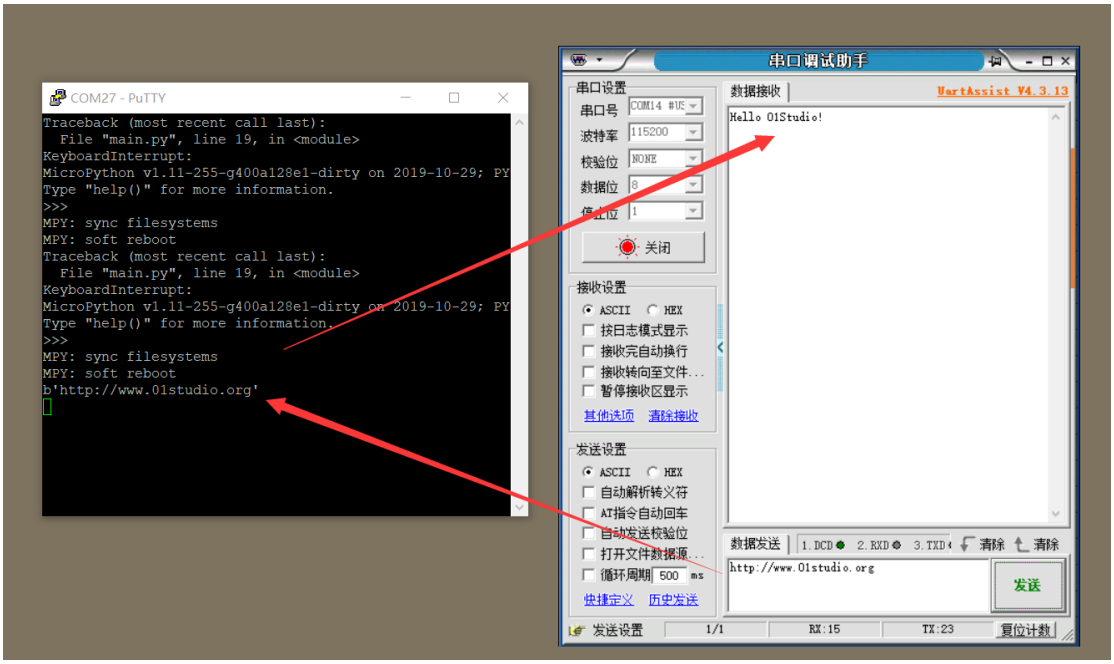


图 4-60 串口收发实验

- **总结：**

通过本节我们学会了串口收发应用，pyboard 的串口资源非常丰富，因此可以接非常多的串口外设。从而实验更多的功能。

第5章　传感器实验

基础实验让我们对 MicroPython 的操作和编程有了比较全面的了解，但其魅力绝不限于此。日常生活中我们会用到各式各样的外设或者传感器，还是那句，一个有经验的嵌入式开发工程师驱动一款未接触过的传感器的一般流程是：了解传感器原理、设计电路图、信号时序分析和编程。没个几天折腾不出来。

生活中有很多传感器已经是非常通用了，前人已经做好封装函数模块，我们直接调用函数即可。我们不需要将时间花在“怎么用”上，而更多的是考虑“用到什么地方”!

本章实验以最常见的传感器出发，讲述是如何通过 MicroPyhon 编程实现传感器应用的。实验还是基于我们的 MicroPython 开发板。而且实验例程将持续更新。

5.1 温度传感器 DS18B20

- **前言：**

相信没有电子爱好者不知道 DS18B20 的，DS18B20 是常用的数字温度传感器，其输出的是数字信号，具有体积小，硬件开销低，抗干扰能力强，精度高的特点。DS18B20 数字温度传感器接线方便，封装成后可应用于多种场合，如管道式，螺纹式，磁铁吸附式，不锈钢封装式，型号多种多样。

主要根据应用场合的不同而改变其外观。封装后的 DS18B20 可用于电缆沟测温，高炉水循环测温，锅炉测温，机房测温，农业大棚测温，洁净室测温，弹药库测温等各种非极限温度场合。耐磨耐碰，体积小，使用方便，封装形式多样，适用于各种狭小空间设备数字测温和控制领域。

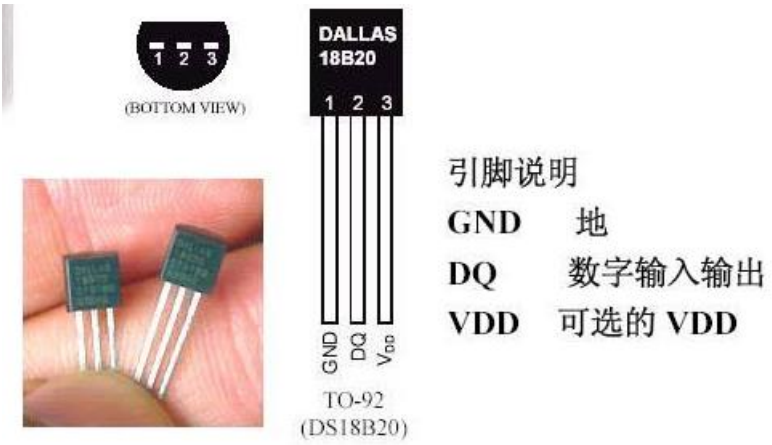


图 5-1 DS18B20 传感器



图 5-2 DS18B20 金属探头封装

- **实验平台：**

MicroPython 开发套件。

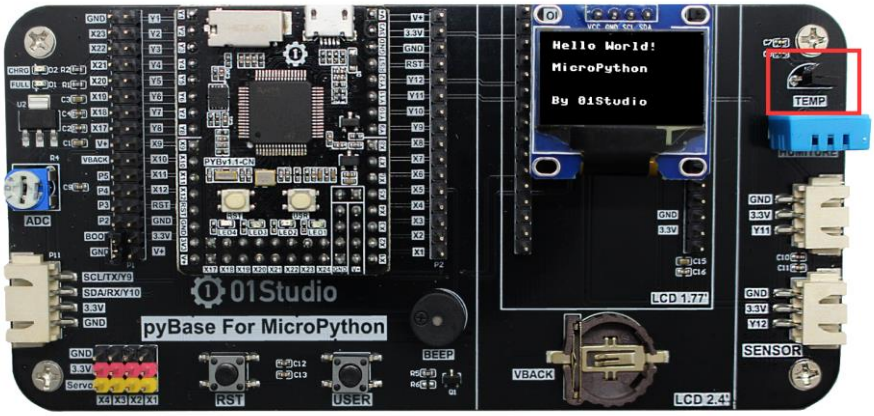


图 5-3　DS18B20 传感器在右上方

- **实验目的：**

通过编程采集温度数据，并在 OLED 上显示。

- **实验讲解：**

DS18B20 是单总线传感器，也就是说只占用 1 个 IO 口。我们来看看原理图：

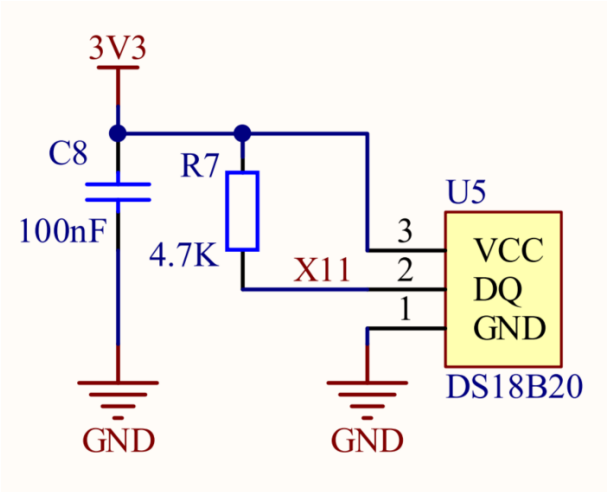


图 5-4 DS18B20 接线原理图

可以看到连接到 DS18B20 传感器是 pyBoard 的 X11 引脚。也就是说针对该引脚编写程序来驱动 DS18B20。那么我们需要自己来编写驱动么？如果你有兴趣的可以自己尝试一下。这部分我们 01Studio 已经收集整理和编写好了，当然也包括 DS18B20。单总线模块文件是：onewire.py，DS18B20 模块的文件是 DS18X20.py，将示例程序文件夹里面以上这两个文件拷贝到 pyboard 文件系统中。

单总线模块说明如下

构造函数	说明
OneWire(Pin)	构造函数，Pin 为相关引脚
使用方法	说明
ow=OneWire(Pin('X11'))	将 X11 引脚使能单总线，名字为 ow

表 5-1　单总线对象

DS18B20 模块说明如下：

构造函数	说明
DS18X20（ow）	构造函数，ow 为单总线对象名称
使用方法	说明
DS18X20(ow).scan()	扫描单总线上所有传感器地址，返回列表
DS18X20(ow).convert_temp()	温度转换，结果存放在地址列表中
DS18X20(ow).read_temp(list[0])	获取总线第一个 DS18B20 的温度值

表 5-2　DS18X20 对象

大部分场景下温度的变化不会太频繁，我们可以每隔 1 秒采集一次，显示精度为小数点后 2 位，基本满足大部分应用需求。编程逻辑如下：

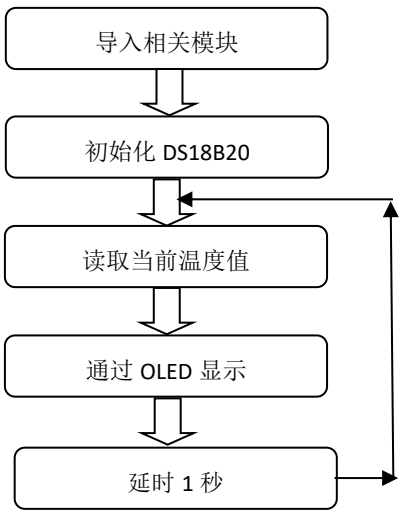


图 5-5 代码编写流程图

参考代码：

```
'''
实验名称：温度传感器 DS18B20
版本：v1.0
日期：2019.4
作者：01Studio
说明：通过编程采集温度数据，并在 OLED 上显示。。
'''

#引用相关模块
from pyb import delay
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C
from onewire import OneWire
from ds18x20 import DS18X20

#初始化相关模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6")) # 软件模拟 I2C
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化 DS18B20
ow= OneWire(Pin('X11')) #使能单总线
ds = DS18X20(ow)        #传感器是 DS18B20
rom = ds.scan()         #扫描单总线上的传感器地址，支持多个传感器同时连接

while True:

   ds.convert_temp()   #温度采集转换
   temp = ds.read_temp(rom[0]) #温度显示,rom[0]为第 1 个 DS18B20
```

```
#数据显示
oled.fill(0)   #清屏背景黑色
oled.text('MicroPython', 0, 0)
oled.text('Temp test:',0,20)
oled.text(str('%.2f'%temp)+' C',0,40) #显示 temp,保留 2 位小数
oled.show()

delay(1000)#延时 1 秒
```

- **实验结果：**

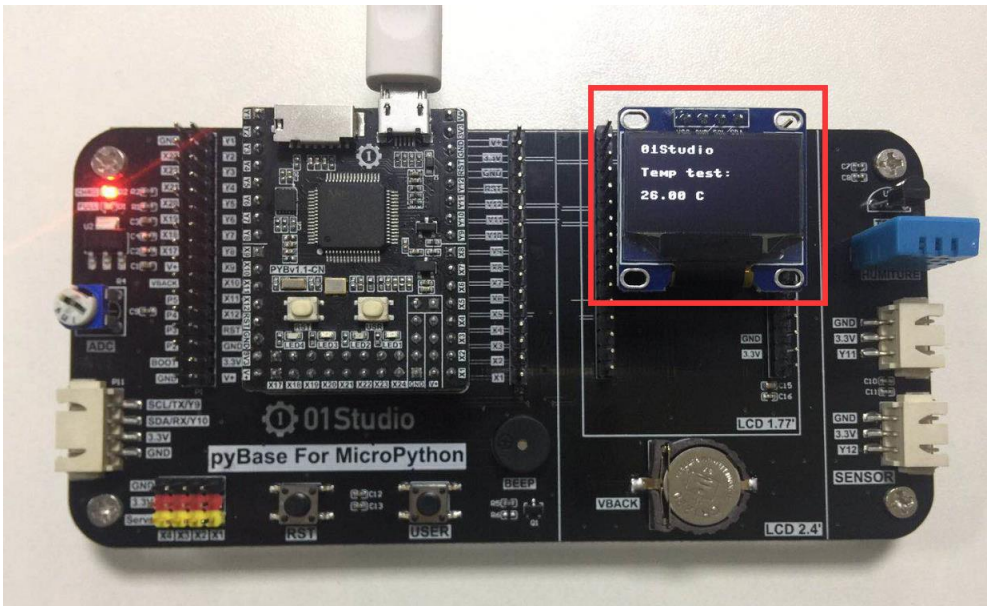


图 5-6 温度检测实验

MicroPython 开发板预留了外界传感器接口，只要接线正确就可以进行更多的传感器实验。我们将带金属探头的 DS18B20 传感器接到中间的传感器母座，其连接的 IO 是“Y11”,所以要将原程序代码的 ow=OneWire(Pin('X19')) 改成 ow=OneWire(Pin('Y11'))，保存后复位。

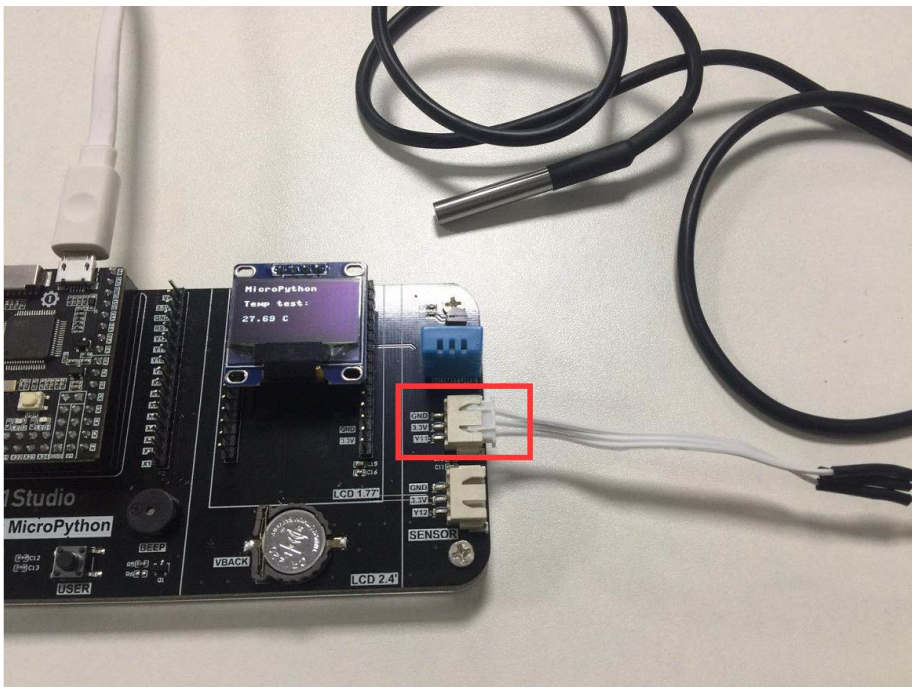


图 5-7 外接温度传感器

我们将金属探头放在冰水里面，可以见到测试到的温度是 3 点多℃。

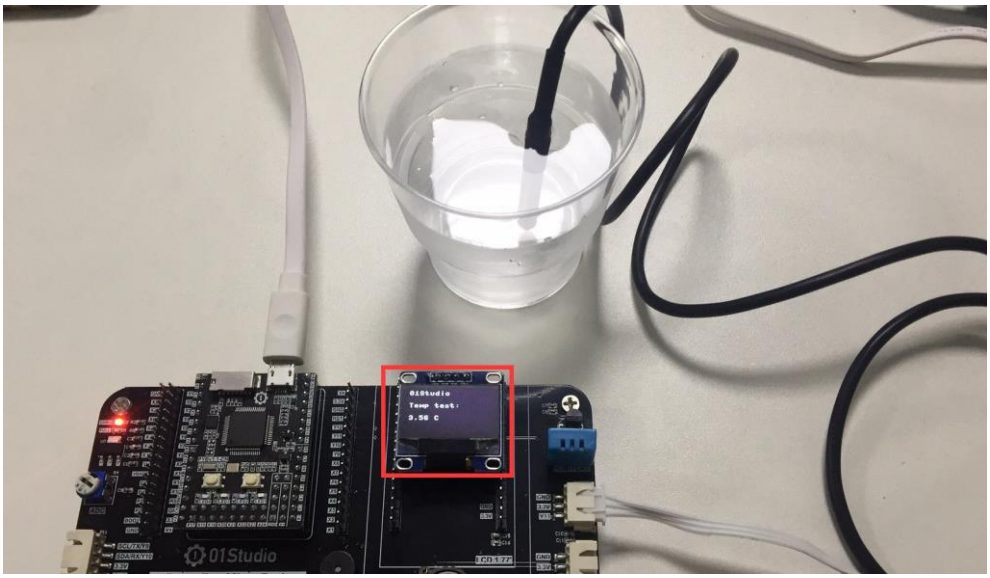


图 5-8 实现金属探头 DS18B20 测量

- **总结：**

DS18B20 作为我们第一个实验传感器，使用 MicroPython 编程非常容易就用起来了，而且精度和稳定性丝毫没有影响。温度传感器只是一个敲门砖，接下来我们将会学习更多的传感器应用。

5.2 温湿度传感器 DHT11

- **前言：**

温湿度也是我们日常非常常见的指标，我们使用的是 DHT11 数字温湿度传感器。这是一款含有已校准数字信号输出的温湿度复合传感器，它应用专用的数字模块采集技术和温湿度传感技术，确保产品具有极高的可靠性和卓越的长期稳定性。

DHT11 具有小体积、极低的功耗，信号传输距离可达 20 米以上，使其成为给类应用甚至最为苛刻的应用场合的最佳选择。产品为 4 针单排引脚封装，连接方便。

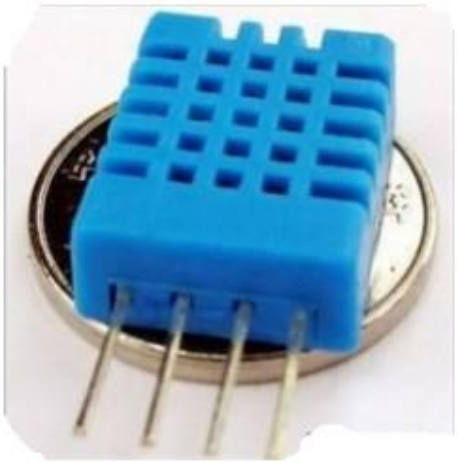


图 5-9 DHT11 温湿度传感器

- **实验平台：**

MicroPython 开发套件。

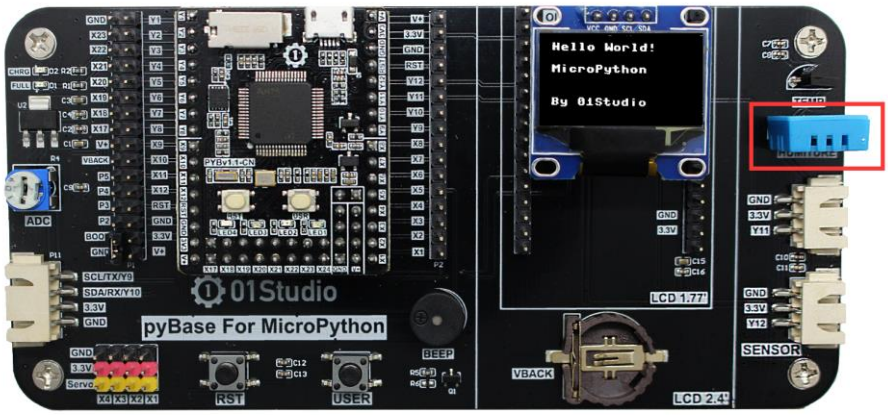


图 5-10 DHT11 传感器在右上方

- **实验目的：**

通过编程采集温湿度数据，并在 OLED 上显示。

- **实验讲解：**

DHT11 虽然有 4 个引脚，但其中第 3 个引脚是悬空的，也就是说 DHT11 也是单总线的传感器，只占用 1 个 IO 口。

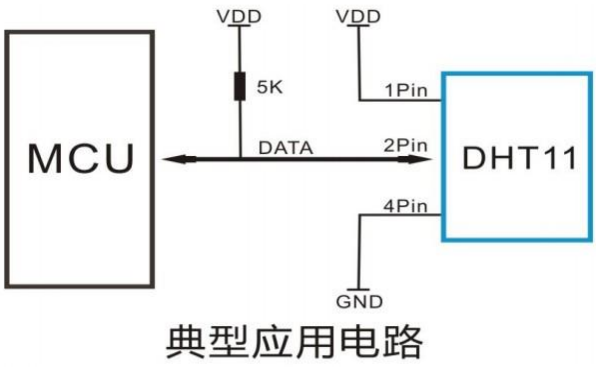


图 5-11 DHT11 应用

我们来看看 DHT11 在开发板上的接线图：

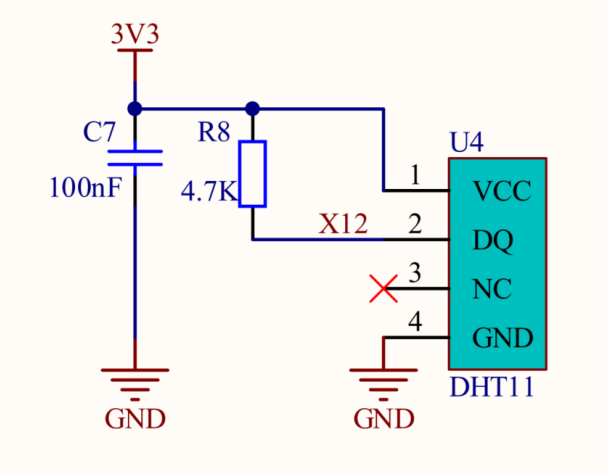


图 5-12 DHT11 接线图

可以看到 DHT11 连接到 pyboard 的‘X12’引脚，可以针对该引脚编程来驱动 DHT11 传感器，模块文件是配套示例程序下的 dht.py，需要拷贝到 pyboard 的文件系统。函数模块说明如下：

构造函数	说明
DHT11（Pin）	指定连接 DHT11 的引脚
使用方法	**说明**
DHT11().measure()	温湿度数据测量
DTH11().temperature()	获取温度值
DTH11().humidity()	获取湿度值

表 5-3

建议上电先延时 1 秒，让 DHT11 稳定后再开设读取。代码编写流程如下：

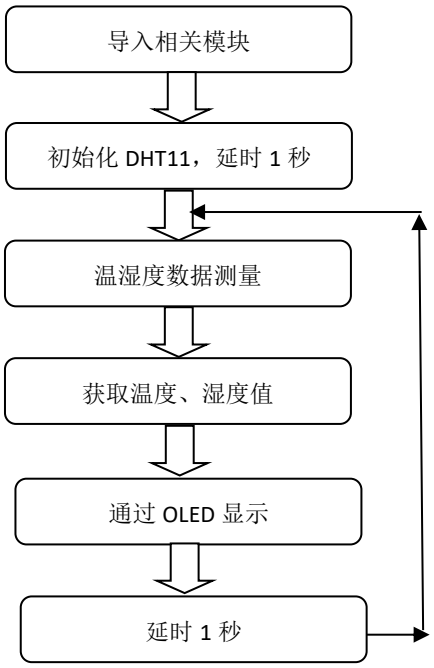


图 5-13 代码编程流程

参考代码如下：

```
'''
实验名称：温湿度传感器 DHT11
版本：v1.0
日期：2019.4
作者：01Studio
```

```
说明：通过编程采集温湿度数据，并在 OLED 上显示。。
'''

#引入相关模块
from pyb import delay
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C
from dht import DHT11

#初始化相关模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#创建 DTH11 对象 dt
dt=DHT11(Pin('X12')) #'X20'连接到开发板上的 DHT11
delay(1000)          #首次启动停顿 1 秒然传感器稳定

while True:

   dt.measure()         #温湿度采集
   te=dt.temperature()  #获取温度值
   dh=dt.humidity()     #获取湿度值

   oled.fill(0) #清屏背景黑色
   oled.text('01Studio', 0, 0)
   oled.text('DHT11 test:',0,15)

   #温度显示
   oled.text(str(te)+' C',0,40)
```

```
#湿度显示
oled.text(str(dh)+' %',48,40)


oled.show()


delay(1000)            #每隔 1 秒采集一次
```

- **实验结果：**

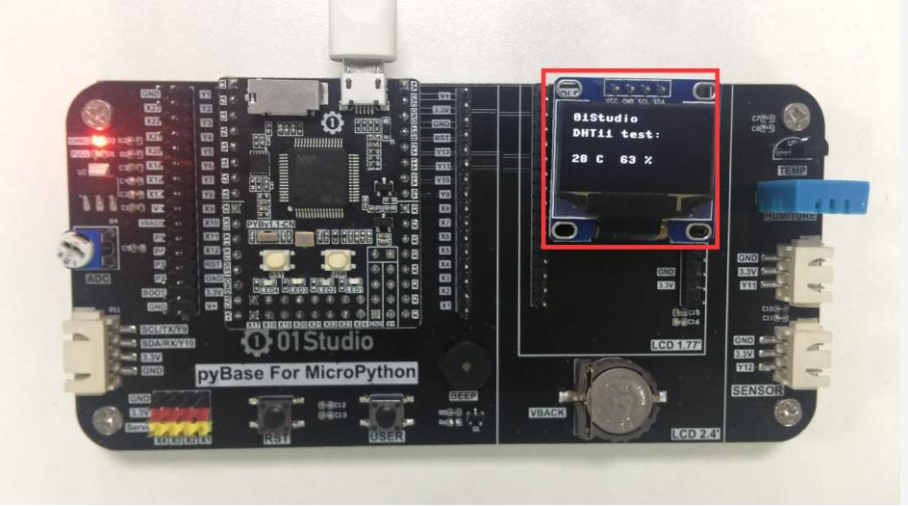


图 5-14 DHT11 实验结果

- **总结：**

通过本节学习我们学会了使用 MicroPython 来驱动 DTH11 温湿度传感器，DHT11 性价比比较高，是很适合学习使用的，但精度和响应速度有点低，需要更高要求应用的用户可以使用 DHT22 或者其他更高级的传感器。

5.3 人体感应传感器

- **前言：**

人体感应传感器，在室内安防应用非常普遍，其原理是由探测元件将探测到人体的红外辐射转变成微弱的电压信号，经过放大后输出。为了提高探测器的探测灵敏度以增大探测距离，一般在探测器的前方装设一个塑料的菲涅尔透镜，它和放大电路相配合，可将信号放大 70dB 以上，这样就可以测出 5~10 米范围内人的行动。

- **实验平台：**

MicroPython 开发套件和人体感应传感器模块。

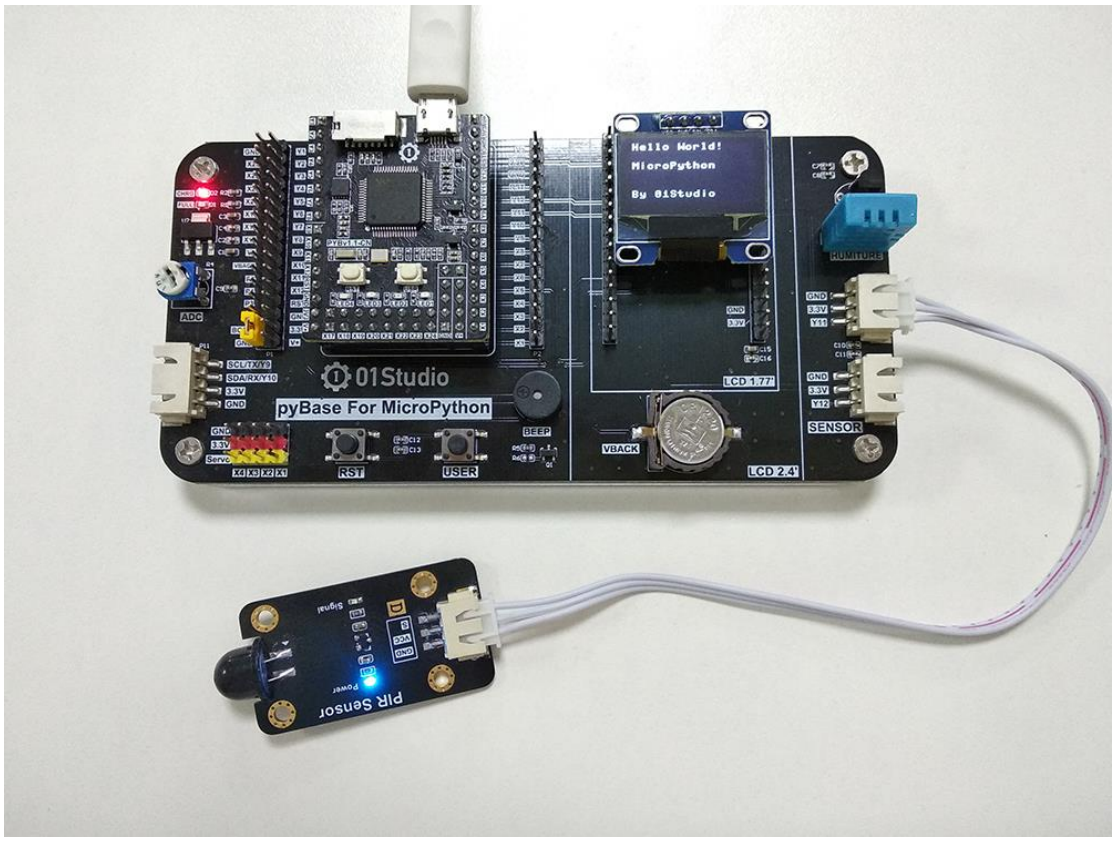


图 5-15 开发套件和传感器接线

- **实验目的：**

通过外部中断编程来检测人体感应模块，当有人出现时候 OLED 通过“Get People!!!”闪烁提示。

- **实验讲解：**

我们先来看看人体感应传感器的介绍：

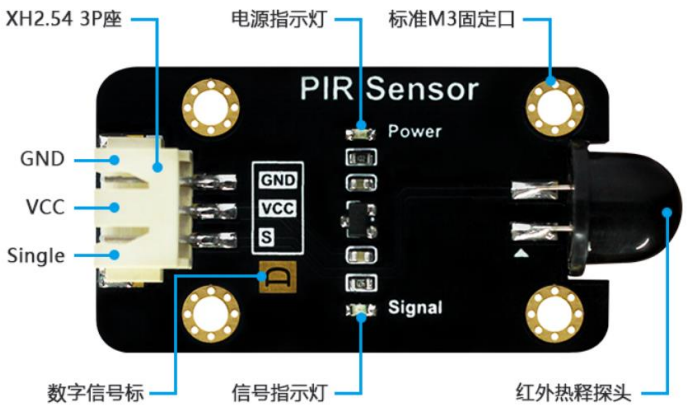


图 5-16 人体感应传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 65℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	数字信号： 检测到人体：高电平 3.3V；持续 3-8 秒 未检测到人体：低电平 0V。
感应角度	100°
感应距离	8 米
模块尺寸	4.5*2.5cm

表 5-4 人体感应传感器模块参数说明

从上表可以知，当检测到人体红外时候，传感器的输出信号为高电平并持续 3-5 秒。如下图所示：

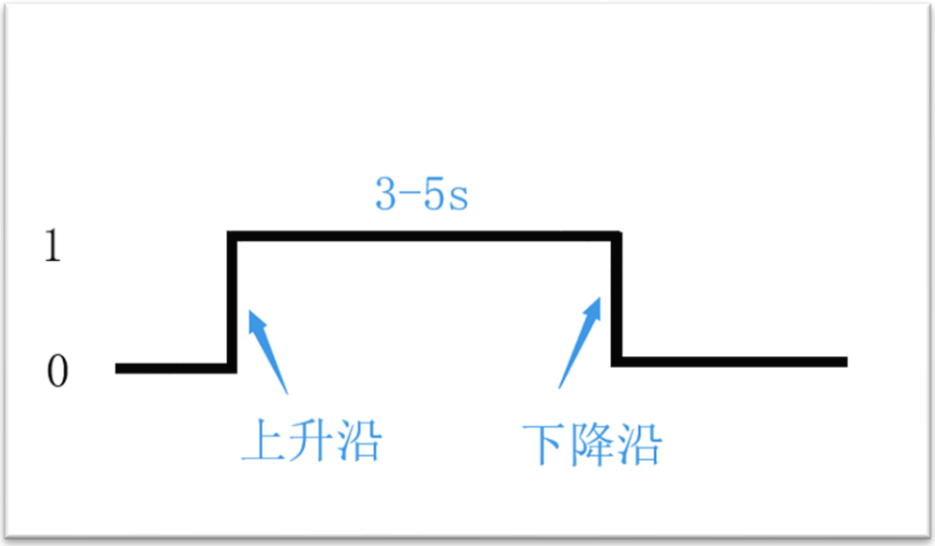


图 5-17 人体感应传感器电平变化

由此可见，可以使用外部中断结合上升沿的出发方式来编程实现相关功能。传感器的输出引脚连接 Sensor1 接口，即连接到 pybaord 的“Y11”引脚。编程方法可以是当“Y11”引脚产生中断时候，说明传感器检测到人体红外线，此时可以在 OLED 上显示相关信息。

关于外部中断的编程方法可以参考基础实验的外部中断章节，这里不再重复，具体的编程流程图如下：

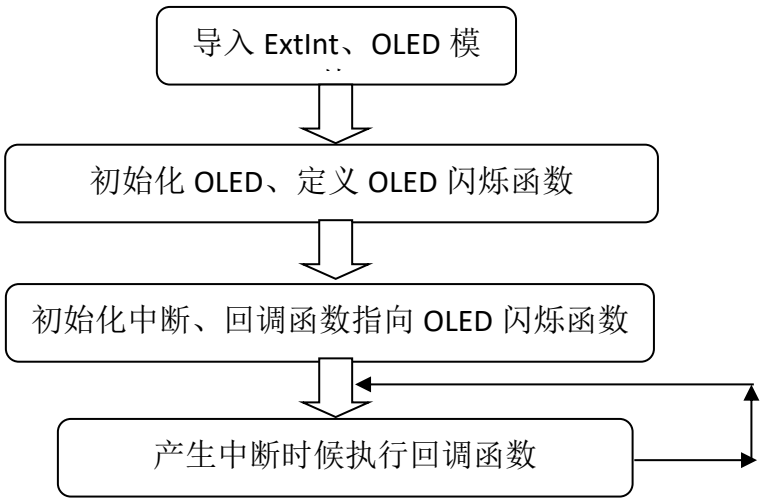


图 5-18 代码编写流程

参考例程代码如下：

```
'''
实验名称：人体感应传感器
版本：v1.0
日期：2019-5-1
作者：01Studio
社区：www.01studio.org
'''

#导入相关模块
from pyb import ExtInt
from machine import I2C,Pin
from ssd1306 import SSD1306_I2C

#初始化 OLED 模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#OLED 初始信息显示
oled.fill(0)  # 清屏背景黑色
oled.text("01Studio", 0, 0)  # 写入第 1 行内容
oled.text("Human body test:", 0, 15)  # 写入第 2 行内容
oled.show()  # OLED 执行显示

def Display(): #Get People 闪烁 5 次效果!

    for i in range(5):
        oled.fill(0)  # 清屏背景黑色
        oled.text("01Studio", 0, 0)  # 写入第 1 行内容
```

```
        oled.text("Human body test:", 0, 15)  # 写入第 2 行内容
        oled.text("Get People!!!", 0, 40)  # 写入第 3 行内容
        oled.show()  # OLED 执行显示
        pyb.delay(500)

        oled.fill(0)  # 清屏背景黑色
        oled.text("01Studio", 0, 0)  # 写入第 1 行内容
        oled.text("Human body test:", 0, 15)  # 写入第 2 行内容
        oled.text("            ", 0, 40)  # 写入第 3 行内容
        oled.show()  # OLED 执行显示
        pyb.delay(500)

callback=lambda e: Display()   #回调函数指向 Display()函数
#上升沿触发，打开上拉电阻
ext = ExtInt(Pin('Y11'), ExtInt.IRQ_RISING, Pin.PULL_UP, callback)
```

- **实验结果：**

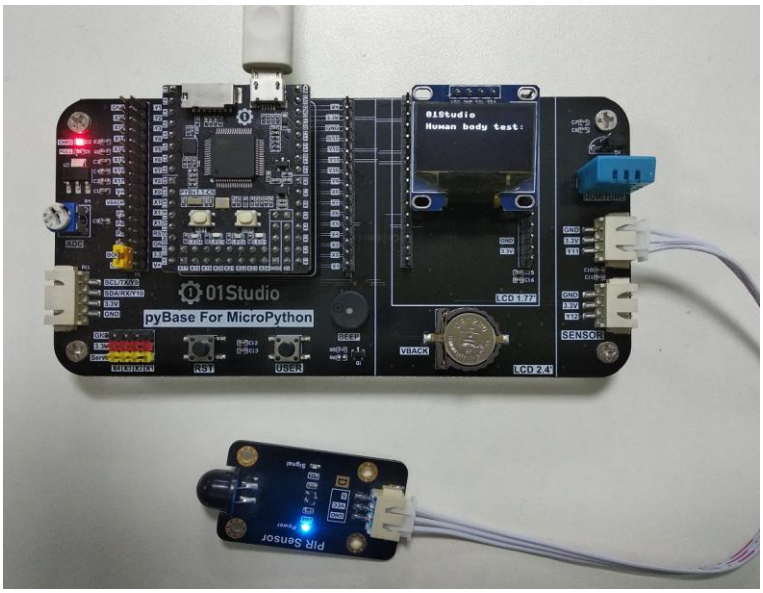


图 5-19 没有检测到到人体

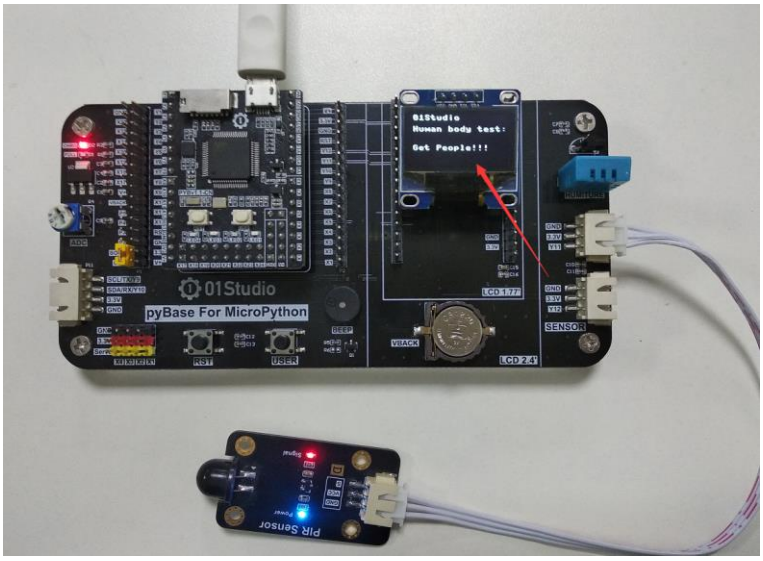


图 5-20 检测到人体，OLED 闪烁

- **总结：**

本节通过简单的中断方式便实现了对人体感应传感器的检测，人体感应传感器的应用非常广泛，特别是在安防领域，结合其它硬件模块可以实现当发现有人入侵时，执行发出警报、实现远程提醒等功能。

5.4 光敏传感器

- **前言：**

光敏传感器实际是一个款可以检测光照强度的传感器，可以应用于我们日常生活中植物光照强度、室内光线检测、以及某些场合的亮度检测。传感器的原理就是将外界模拟变化的信号转变成数字信号（电压值）让单片机出来，处理方式就是常用的 ADC（模拟数字转换）。

- **实验平台：**

MicroPython 开发套件和光敏传感器模块。

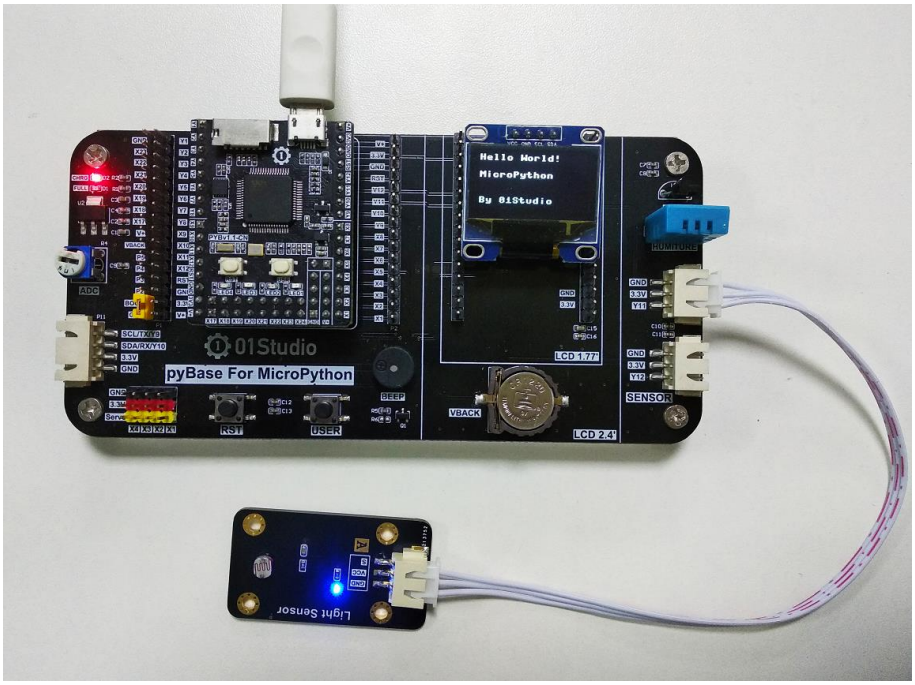


图 5-21 开发套件和光敏传感器接线

- **实验目的：**

采集当前环境的光照强度并在 OLED 显示，显示方式为：Bright-强，Normal-中等，Weak-弱。

- **实验讲解：**

我们先来看看光敏传感器模块的介绍：

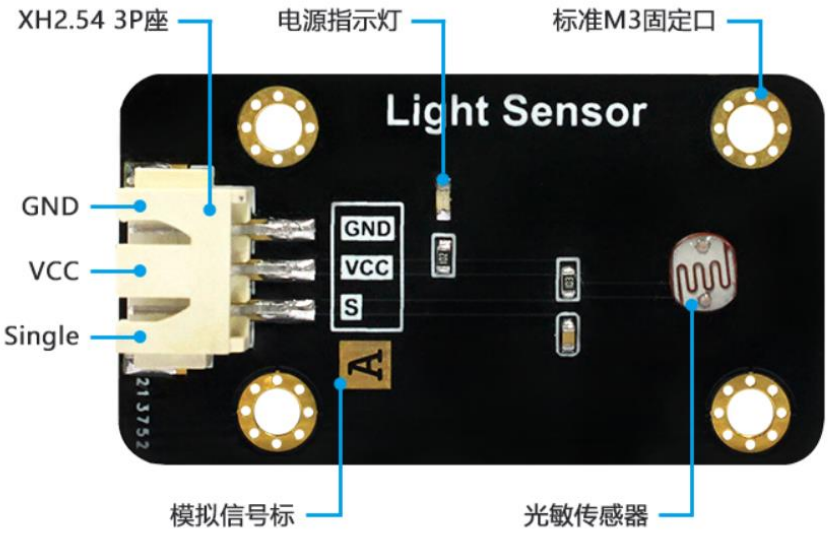


图 5-22 光敏传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	模拟信号：0-3.3V （VCC=3.3V 时）
模块尺寸	4.5*2.5 cm

表 5-5

从上表可以看到，光敏传感器输出的是模拟信号：0-3.3V，这代表了外界的光照强度。接近 0V 时光线最强，接近 3.3V 时，光线最弱。因此我们可以使用基础实验-ADC 学习过的内容来编程。

光敏传感器接在传感器接口 1，对应的引脚是“Y11”。ADC 使用 12bit 精度，即最大值为 2^12-1=4095。然后我们将检测到的数值 0-4095 分成三段，分别代表光线强：【0-1365】，中等：【1365-2730】，弱：【2730-4095】。当然开发者可以根据自己实际情况来调整数值。编程流程图如下：

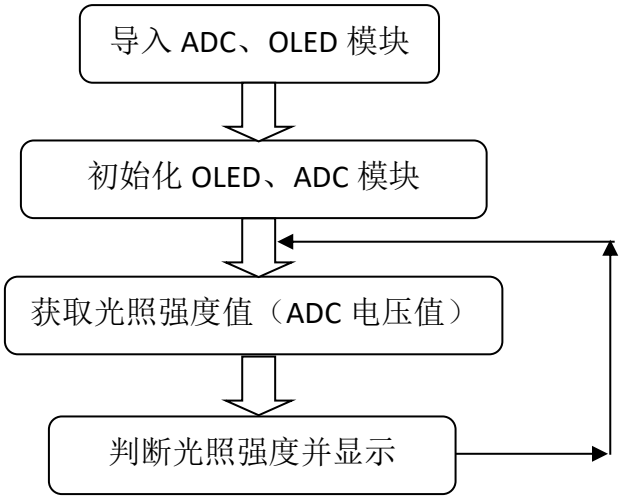


图 5-23 代码编写流程

参考程序代码如下：

```
'''
实验名称：光敏传感器
版本：v1.0
日期：2019.5
作者：01Studio 【www.01Studio.org】
说明：通过光敏传感器对外界环境光照强度测量并显示。
'''

#导入相关模块
import pyb
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C

#初始化相关模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
```

```
#初始化 ADC,Pin='Y11'
Light = pyb.ADC('Y11')

while True:

    oled.fill(0)  # 清屏显示黑色背景
    oled.text('01Studio', 0, 0)  # 首行显示 01Studio
    oled.text('Light test:', 0, 15)      # 次行显示实验名称

    value=Light.read() #获取 ADC 数值

    #显示数值
    oled.text(str(value)+' (4095)',0,40)
    #计算电压值，获得的数据 0-4095 相当于 0-3V，（'%.2f'%）表示保留 2 位小数
    oled.text(str('%.2f'%(value/4095*3.3))+' V',0,55)

    #判断光照强度，分 3 档显示。
    if 0 < value <=1365:
        oled.text('Bright', 60, 55)

    if 1365 < value <= 2730:
        oled.text('Normal', 60, 55)

    if 2730 < value <= 4095:
        oled.text('Weak  ', 60, 55)

    oled.show()
    pyb.delay(1000)
```

- **实验结果：**

用手遮挡住光敏传感器，可以见到 OLED 显示光线强势是弱 Weak：

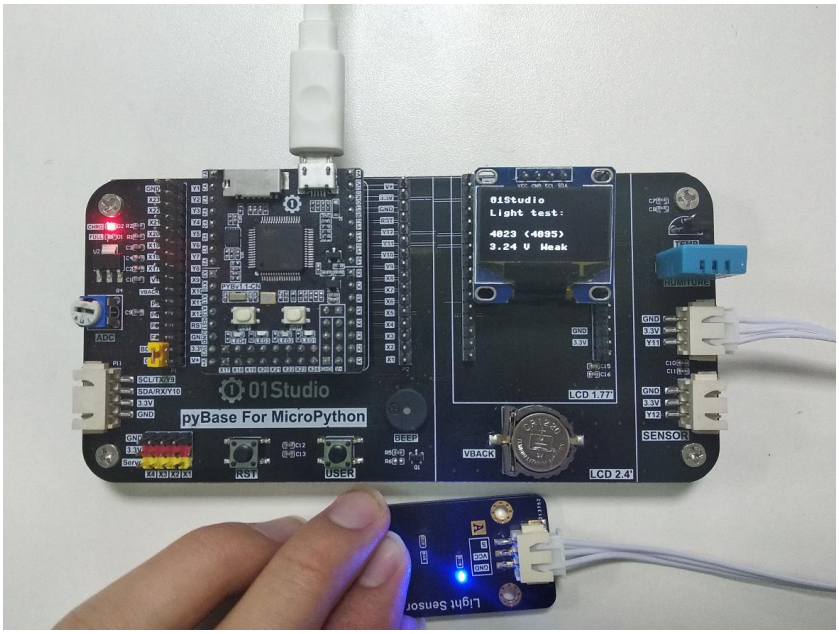


图 5-24 光线强度弱

打开手机手电筒，照在光敏传感器上，可以见到光线强度是强 Bright：

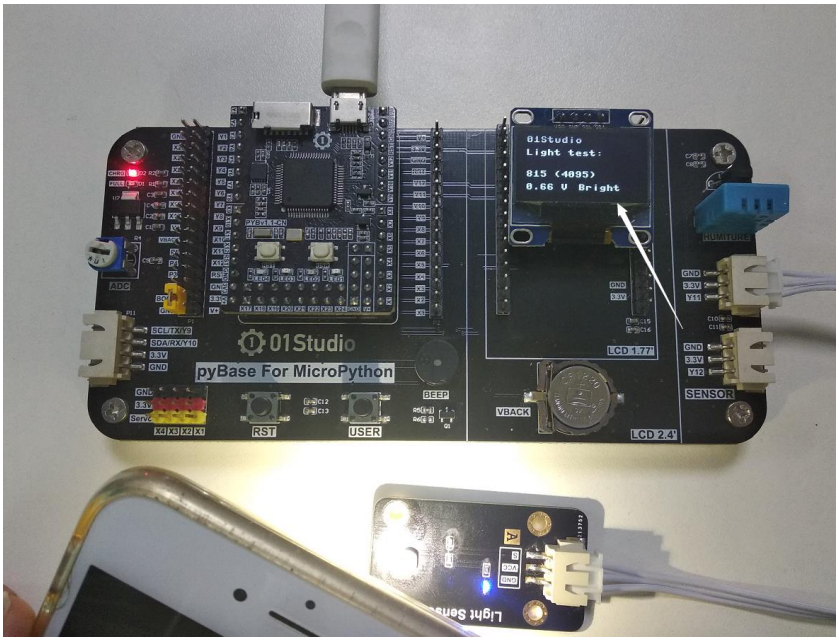


图 5-25 光线强度强

- **总结：**

从实验可以看到，光敏传感器背后的原理是对 ADC 的应用，实现了改功能后。我们可以自行扩展深入，制作自己喜欢的电子产品。

5.5 土壤湿度传感器

- **前言：**

土壤湿度传感器用于检测盆栽泥土的湿度，当泥土干枯时候，我们就需要给植物浇水了。这个用途非常广泛，如自动灌溉。

- **实验平台：**

MicroPython 开发套件和光敏传感器模块。

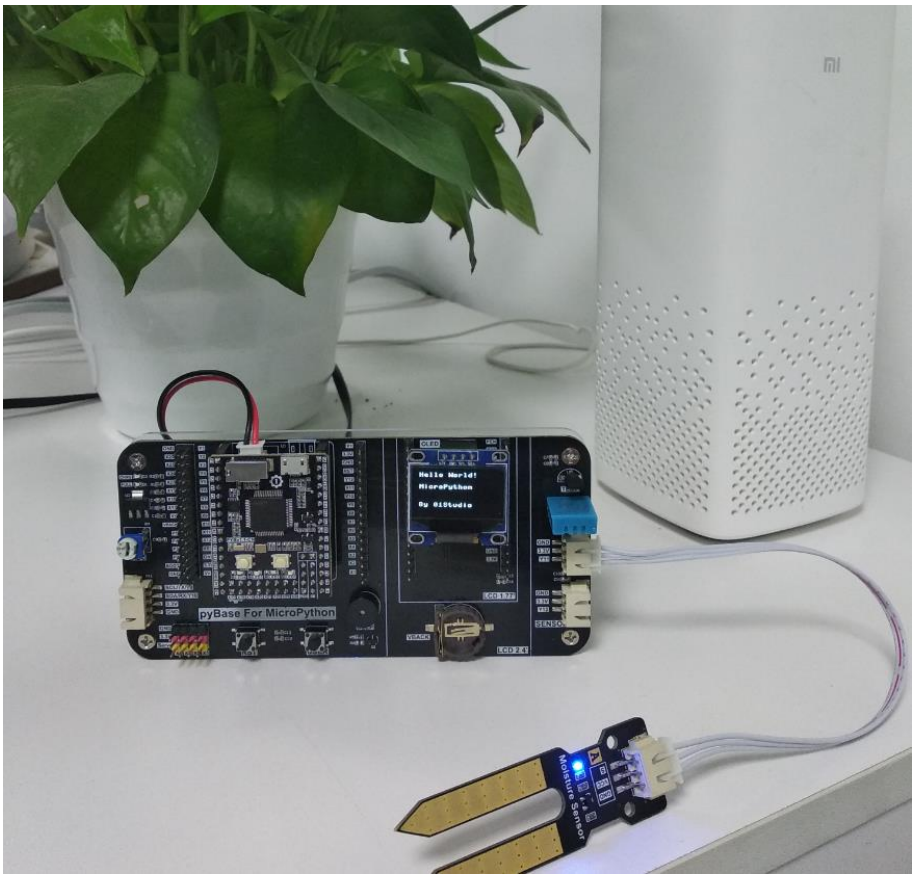


图 5-26 开发套件和传感器连接图

- **实验目的：**

采集盆栽土壤的的光照强度并在 OLED 显示，显示方式为：Dry-干，Normal-中等，Wet-湿。

- **实验讲解：**

我们先来看看土壤湿度传感器模块的介绍：

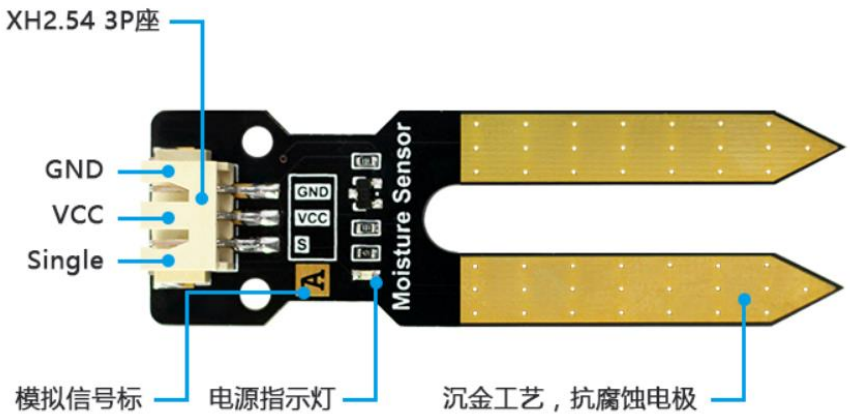


图 5-27 土壤湿度传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-40 至 85℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	模拟信号：0-3.3V （VCC=3.3V 时）
模块尺寸	6.6*2.0cm

表 5-6

从上表可以看到，土壤湿度传感器输出的是模拟信号：0-3.3V，这代表土壤的湿度情况。接近 0V 时湿度为干燥，接近 3.3V 时，湿度情况为湿润。因此我们可以使用基础实验-ADC 学习过的内容来编程。

土壤湿度传感器接在传感器接口 1，对应的引脚是“Y11”。ADC 使用 12bit 精度，即最大值为 2^12-1=4095。然后我们根据实际测试数据将检测到的数值 0-4095 分成三段，分别代表土壤干燥:【0-1247】，中等:【1247-2238】，湿润:【2238-4095】。当然开发者可以根据自己实际情况来调整数值。编程流程图如下：

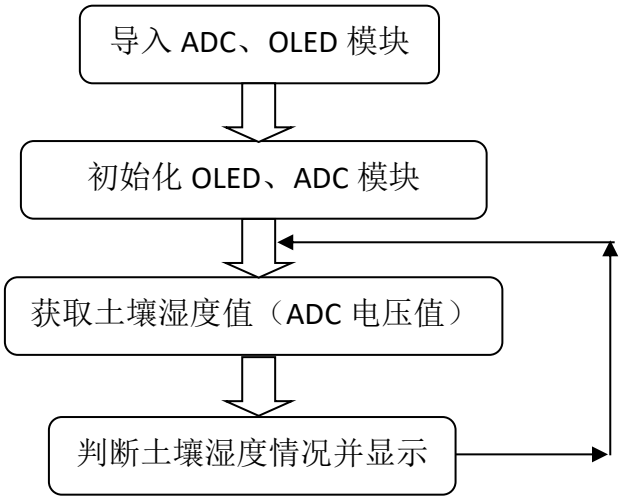


图 5-28 代码编写流程

参考程序代码如下：

```
'''
实验名称：土壤湿度传感器
版本：v1.0
日期：2019.5
作者：01Studio 【www.01Studio.org】
说明：通过土壤湿度传感器对土壤湿度测量并显示。
'''

#导入相关模块
import pyb
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C

#初始化相关模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化 ADC,Pin='Y11'
```

```
Soil = pyb.ADC('Y11')

while True:

    oled.fill(0)  # 清屏显示黑色背景
    oled.text('01Studio', 0, 0)  # 首行显示 01Studio
    oled.text('Soil test:', 0, 15)      # 次行显示实验名称

    value=Soil.read() #获取 ADC 数值

    #显示数值
    oled.text(str(value)+' (4095)',0,40)
    #计算电压值，获得的数据 0-4095 相当于 0-3V，（'%.2f'%）表示保留 2 位小数
    oled.text(str('%.2f'%(value/4095*3.3))+' V',0,55)

    #判断土壤湿度，分 3 档显示。
    if 0 <= value <=1247:
        oled.text('Dry', 60, 55)

    if 1247 < value <= 2238:
        oled.text('Normal', 60, 55)

    if 2238 < value <= 4095:
        oled.text('Wet  ', 60, 55)

    oled.show()
    pyb.delay(1000)
```

- **实验结果：**

将传感器插到干燥的土壤中，可以见到 OLED 显示干燥-Dry：

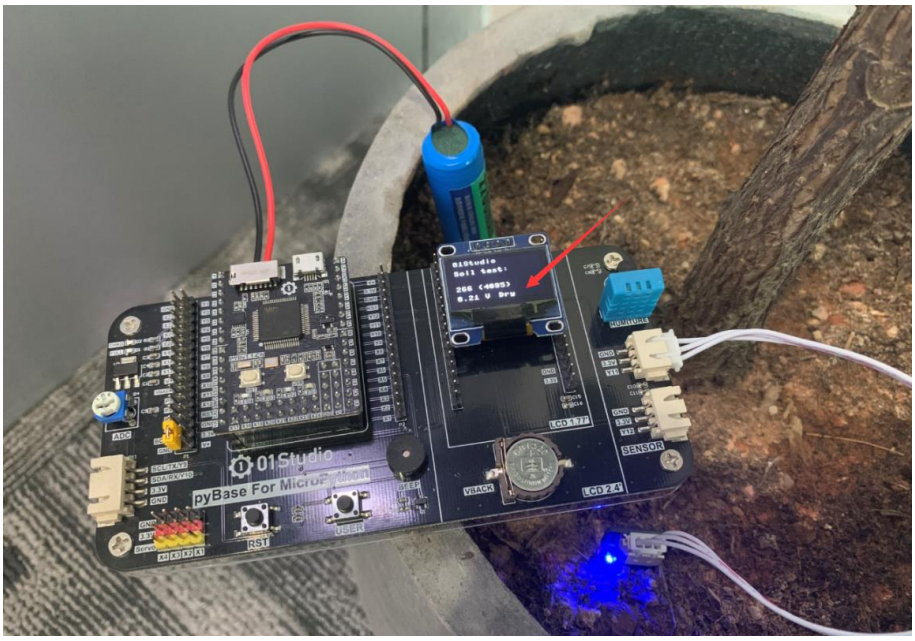


图 5-29 干燥的土壤

从传感器侧面往土壤浇水，让土壤变得湿润。**注意不要浇到传感器电路上！**

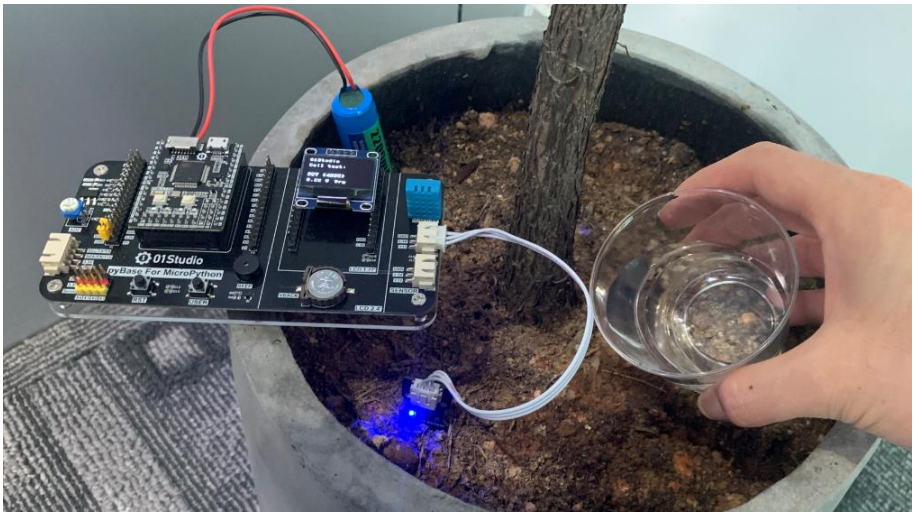


图 5-30 浇水

可以看到 OLED 显示湿润-Wet:



图 5-31 湿润的土壤

- **总结：**

从实验可以看到，土壤湿度传感器背后的原理是对 ADC 的应用，实现了该功能后。我们可以自行扩展深入，制作自己喜欢的电子产品。

5.6 水位传感器

- **前言：**

水位（液位）传感器是一款简单易用、小巧轻便、水位/水滴识别检测传感器，传感器通过一系列的暴露的平行导线线迹测量其水滴/水量大小使得传感器输出的模拟信号产生相应的变化，轻松完成水量到模拟信号的转换，从而判断水位。

- **实验平台：**

MicroPython 开发套件和水位传感器模块。

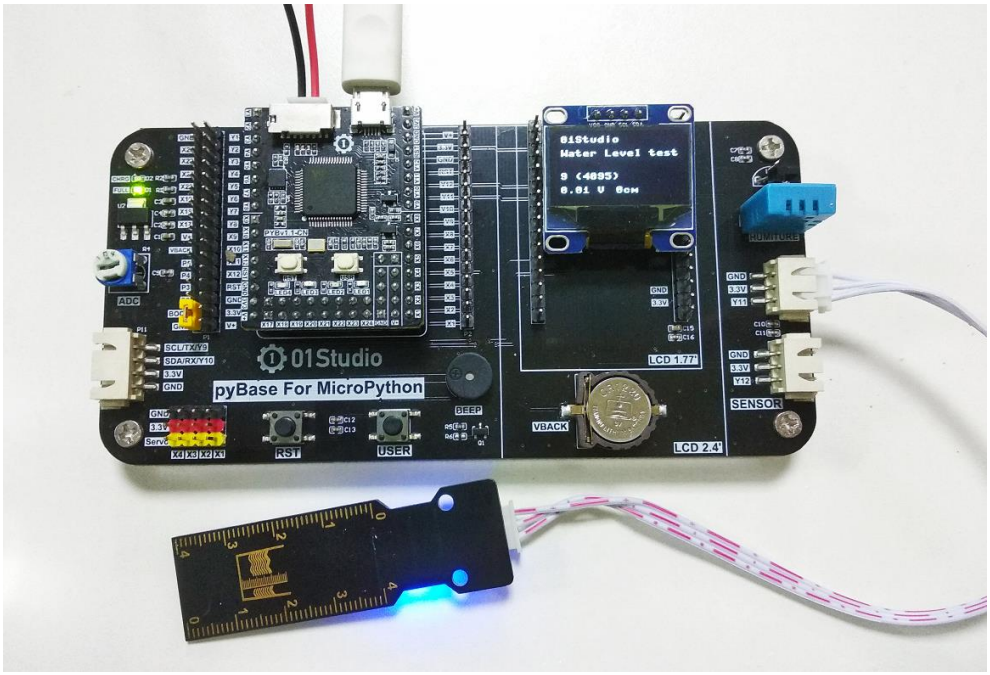


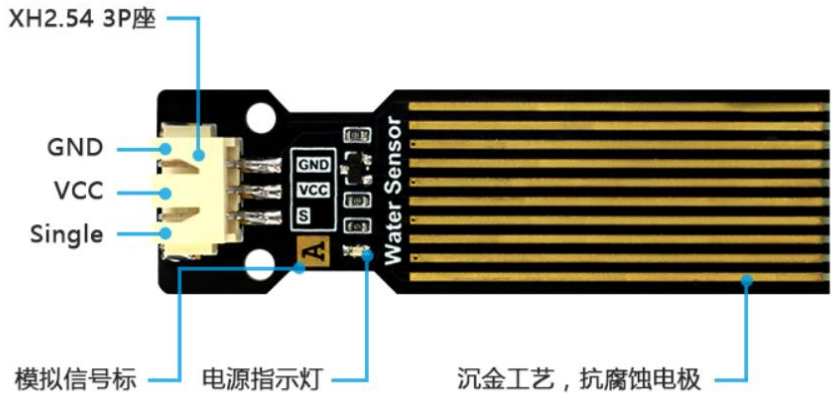
图 5-32 开发套件和水位传感器接线图

- **实验目的：**

测量水位高度并在 OLED 显示，显示方式为：0-4cm。

- **实验讲解：**

我们先来看看水位传感器的介绍：



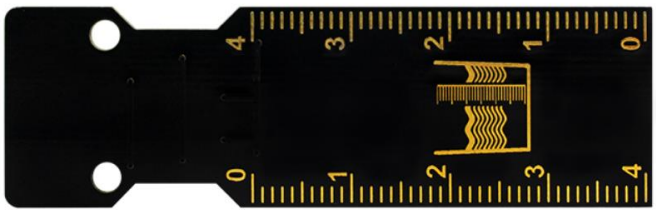


图 5-33 水位传感器模块正反面

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	0 至 60℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	模拟信号：0-3.3V （VCC=3.3V 时）
模块尺寸	6.6*2.0cm
注意事项	水位传感器接口旁元件不能接触水，否则容易短路。所以测量时候要注意液体水位的高度。

表 5-7

从上表可以看到，水位传感器的量程为 4cm，输出的是模拟信号：0-3.3V，这代表水位高度情况。接近 0V 时湿度为 0cm 或没有放进液体，电压升高，意味着水位升高。因此我们可以使用基础实验-ADC 学习过的内容来编程。

水位传感器接在传感器接口 1，对应的引脚是“Y11”。ADC 使用 12bit 精度，

即最大值为 2^12-1=4095。我们使用的是普通桶装水，根据实际测试数据将检测到的数值 0-4095 分成 5 档。分别代表水位高度 0cm：【0-600】，1cm：【600-900】，2cm：【900-1200】，3cm：【1200-1300】，4cm：【>1300】。

当然开发者可以根据自己测量液体不同或其它情况来调整数值。编程流程图如下：

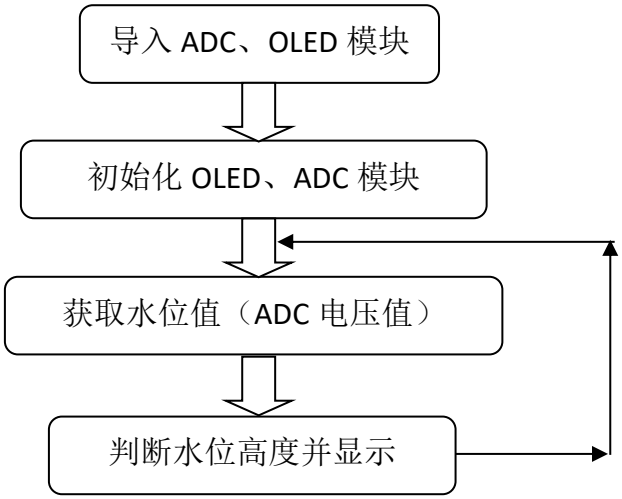


图 5-34 代码编写流程

参考程序代码如下：

```
'''
实验名称：水位传感器
版本：v1.0
日期：2019.5
作者：01Studio 【www.01Studio.org】
说明：通过水位传感器对水位测量并显示。
'''

#导入相关模块
import pyb
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C
```

```
#初始化相关模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化 ADC,Pin='Y11'
Water_level = pyb.ADC('Y11')

while True:

    oled.fill(0)  # 清屏显示黑色背景
    oled.text('01Studio', 0, 0)  # 首行显示 01Studio
    oled.text('Water Level test', 0, 15)      # 次行显示实验名称

    value=Water_level.read() #获取 ADC 数值

    #显示数值
    oled.text(str(value)+' (4095)',0,40)
    #计算电压值，获得的数据 0-4095 相当于 0-3V，（'%.2f'%）表示保留 2 位小数
    oled.text(str('%.2f'%(value/4095*3.3))+' V',0,55)

    #判断水位，分 5 档显示，0-4cm
    if 0 <= value <=600:
        oled.text('0cm', 60, 55)

    if 600 < value <= 900:
        oled.text('1cm', 60, 55)

    if 900 < value <= 1200:
        oled.text('2cm', 60, 55)
```

```
if 1200 < value <= 1300:
    oled.text('3cm', 60, 55)

if 1300 < value:
    oled.text('4cm', 60, 55)

oled.show()
pyb.delay(1000)
```

- **实验结果：**

  当没插入水中时，OLED 显示当前水位为 0cm:

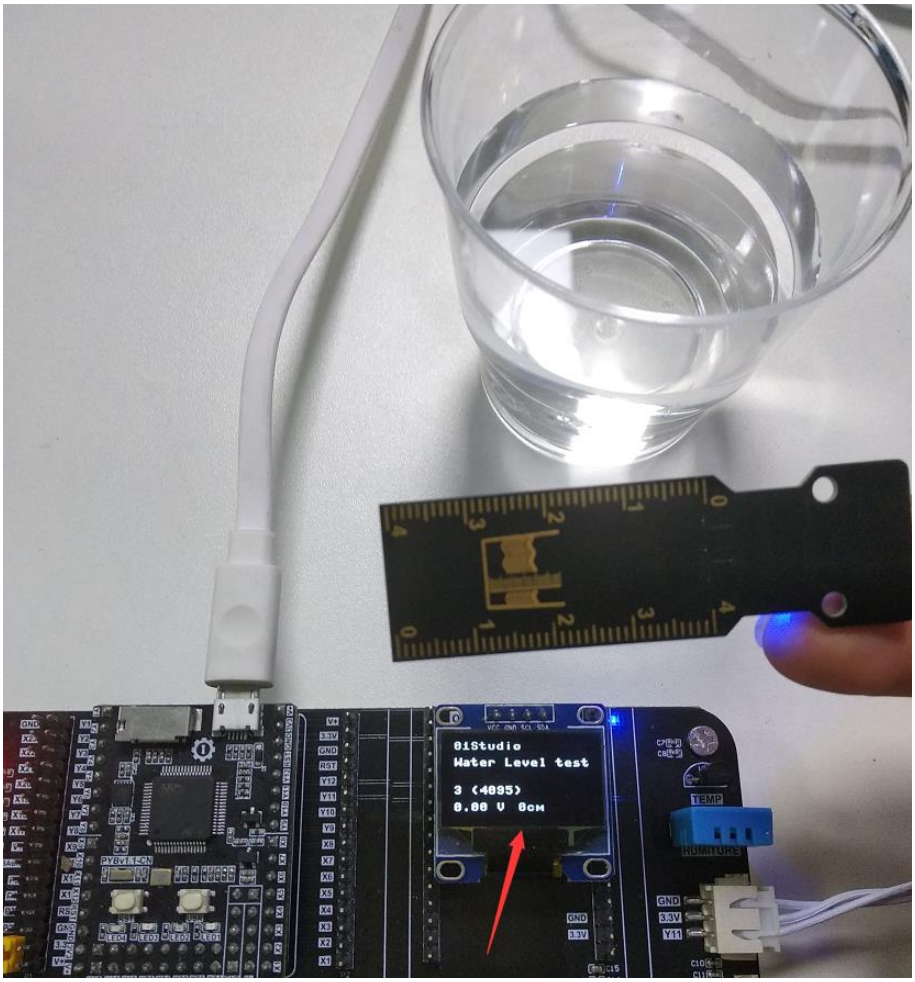


图 5-35

将水位传感器模块慢慢放入水中，可以看到 OLED 显示从 1cm-4cm 变化。

（注意量程是 4cm，请勿将超出的电路部分放置水中以免发生短路。）

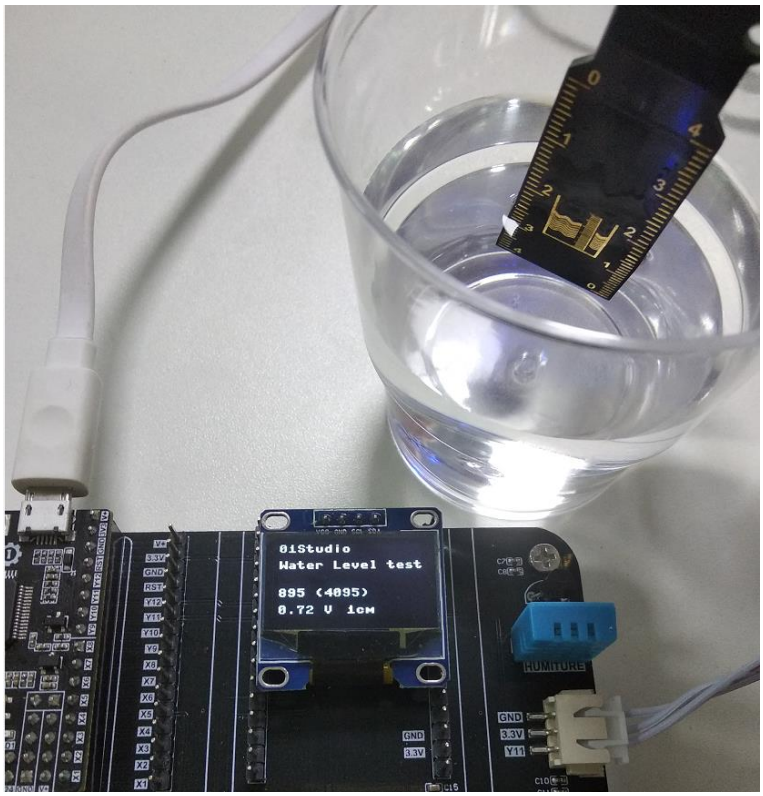


图 5-36 水位接近 1cm

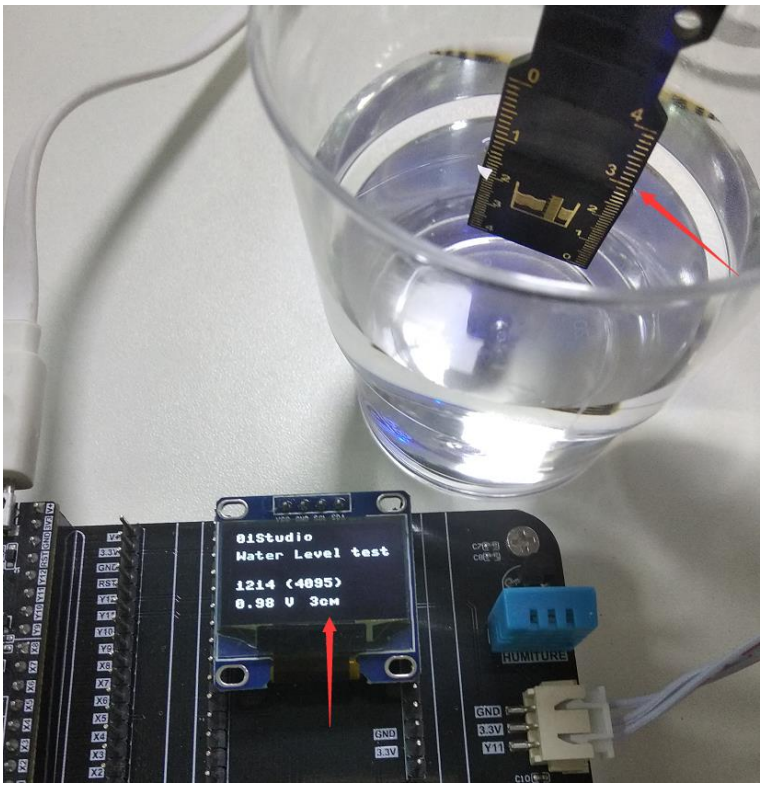


图 5-37 水位接近 3cm

- **总结：**

从实验可以见到，这款水位传感器出了可以用来测量水位（液位）高度外，还可以用来测量室外下雨情况和家里的淹水情况。有兴趣的小伙伴可以动手制作一下！

5.7 大气压强传感器

- **前言：**

本实验中大气压强传感器模块使用的是 BMP280，这是一款专为移动应用而设计的高精度大气压传感器， 传感器模块采用极其紧凑的封装，其小尺寸和低功耗可以应用在手机、 GPS 模块、 手表的电池供电设备中。 BMP280 传感器除了可以测量大气压强，还可以测量温度（温度精度不高）以及通过计算公式来换算出海拔高度。

- **实验平台：**

MicroPython 开发套件和大气压强传感器模块，传感器连接到 I2C/UART 扩展接口。

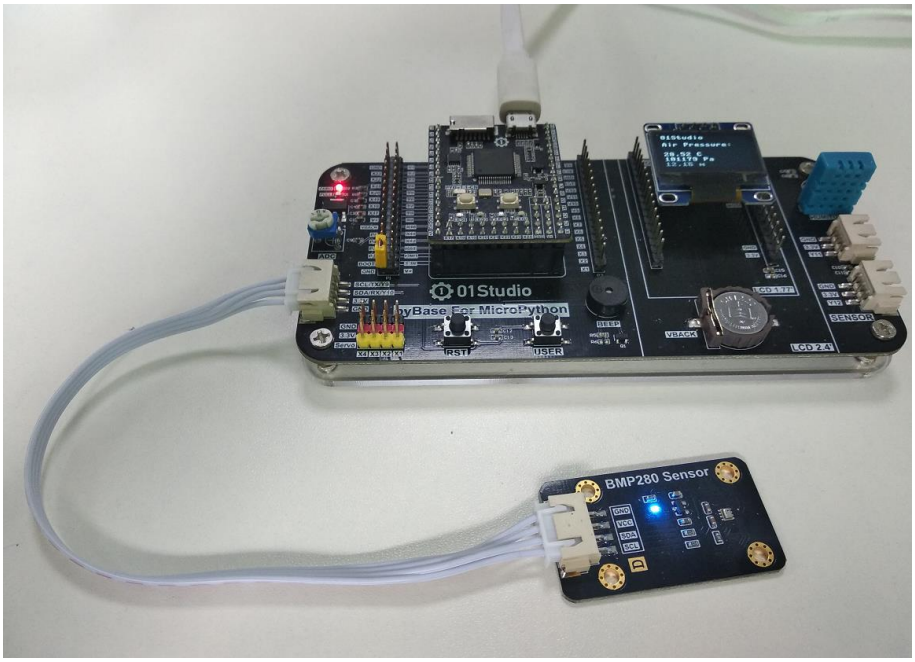


图 5-38 开发套件和大气压强传感器接线方法

- **实验目的：**

编程测量当前环境的大气压强、温度信息，将大气压强通过公式计算出当前海拔高度，并在 OLED 显示。打造自己的气压计！

- **实验讲解：**

我们先来看看大气压强传感器模块的介绍：

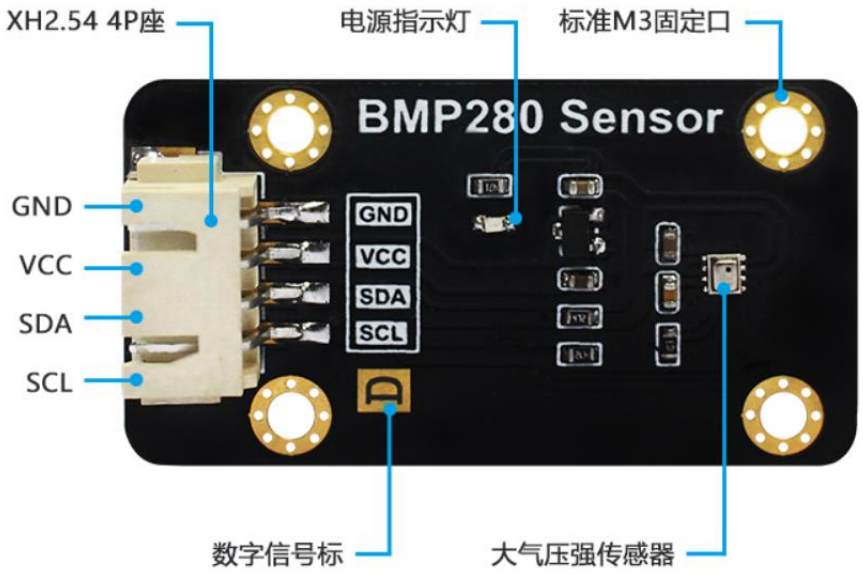


图 5-39 大气压强传感器模块

功能参数	
传感器型号	BMP280
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85℃
接口定义	XH2.54 防呆接口（4Pin）【GND、VCC、SDA、SCL】
通讯信号	I2C 总线
I2C 地址	0x76（BMP280 的 SDO 默认下拉）； 当 BMP280 的 SDO 引脚上拉时，I2C 地址为：0x77
模块尺寸	4.5*2.5cm

表 5-8

从上面介绍可以看到，BMP280 是一款通过 I2C 接口驱动的传感器。连接到我们的 I2C（2）外扩接口上。我们通过前面学习的 I2C 接口使用的方式，即可以对该模块实现数据通讯。

标准大气压是指把温度为 0℃、纬度 45 度海平面（海拔为 0 米）上的气压

称为 1 个大气压，其数值为 101325 帕斯卡（Pa）。

大气压同海拔高度的关系：P=P0×（1-H/44300)^5.256

因此计算高度公式为：H=44300*(1- (P/P0)^(1/5.256))

式中：H 为海拔高度，P0=大气压（0℃，101325Pa）

从上面公式可以看到，高度是通过大气压强换算出来的，从物理学的角度我们可以知道，高度越高的地方，空气越稀薄，大气压强越低。通过气压的变化我们就可以计算出海拔高度；但是这存在特定条件，那就是温度为 0℃的时候，而温度越高的地方，空气越稀薄，大气压强就越低。因此高度数据理论上需要做温度补偿，因此本实验的高度值换算存在轻微误差。有兴趣的小伙伴可以自行深入研究。

MicroPython 的强大之处是其有丰富的模块和函数库，一旦模块建立了，那么后来者使用起来就非常简单，无须再去做底层驱动的开发。从而实现面向对象的编程，而当有需要的时候又可以去改底层代码，可以说是进可攻退可守，非常灵活。在这里我们直接调用已经编写好的驱动文件 bmp280.py，该文件实现了对大气压、温度、高度的测量和计算。用户直接使用即可，具体如下：

构造函数	说明
bmp280.BMP280(I2C)	定义传感器的 I2C 接口。
使用方法	**说明**
getTemp()	获取温度数据
getPress()	获取气压数据
getAlititude()	获取高度数据

表 5-9　大气压强传感器 BMP280 对象

打开示例程序的 bmp280.py 文件，找到 getAltitude() 的定义函数，可以看到计算公式跟前面分析的一样，这里保留了 2 位小数：

```
    # Calculating absolute altitude
    def getAltitude(self):
        return '%.2f'%(44330*(1-(self.getPress()/101325)**(1/5.256)))
```

理解了传感器原理和模块使用方法后，我们可以整理出编程思路，流程图如下：

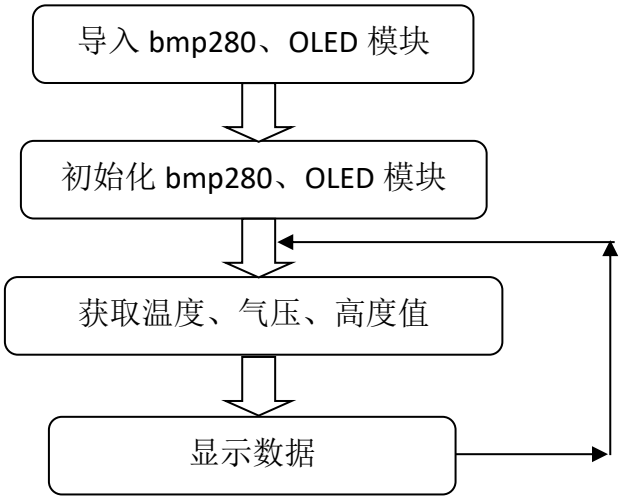


图 5-40 代码编写流程

参考程序代码如下：

```
'''
实验名称：大气压强传感器
版本：v1.0
日期：2019.5.1
作者：01Studio 【www.01Studio.org】
说明：测量 BMP280 温度、气压和计算海拔值，并在 OLED 上显示。。
'''
```

```
import pyb
import bmp280
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C

#初始化 OLED
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化 BMP280，I2C 接口 2
BMP = bmp280.BMP280(I2C(2))

while True:

    oled.fill(0)  # 清屏,背景黑色
    oled.text('01Studio', 0, 0)
    oled.text('Air Pressure:', 0, 15)

    # 温度显示
    oled.text(str(BMP.getTemp()) + ' C', 0, 35)
    # 湿度显示
    oled.text(str(BMP.getPress()) + ' Pa', 0, 45)
    # 海拔显示
    oled.text(str(BMP.getAltitude()) + ' m', 0, 55)

    oled.show()

    pyb.delay(1000)  # 每隔 1 秒采集一次
```

- **实验结果：**

程序烧写完复位后可以见到实验结果：温度：28.36℃，气压：101160 Pa，高度：13.74 M。 该数据的实测地点深圳市，楼层 5 楼，可见数据还是相当合理。

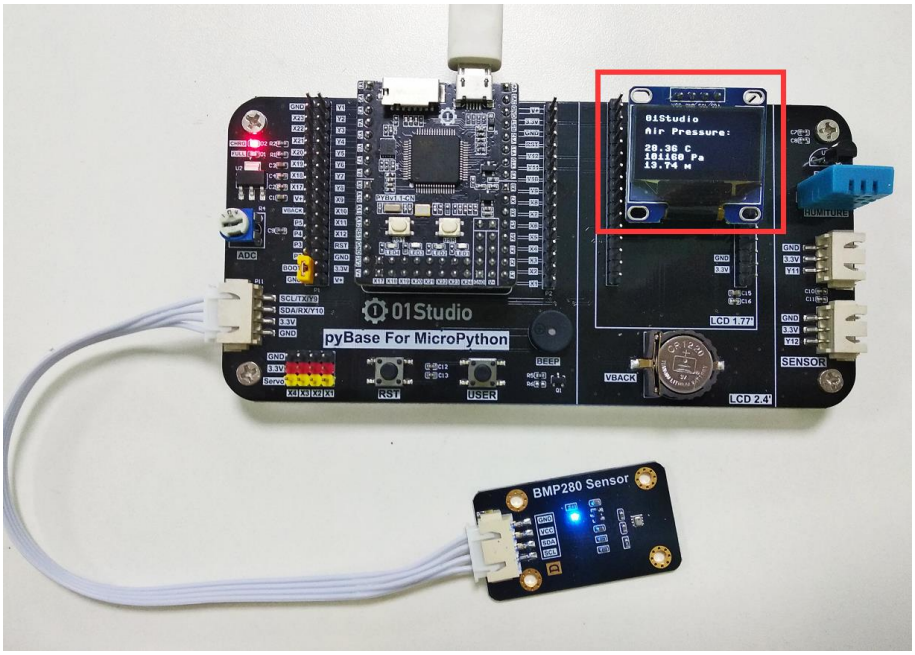


图 5-41 气压测量

开发板接上锂电池供电，尝试上下楼，会发现气压和海拔数值相应的变化。

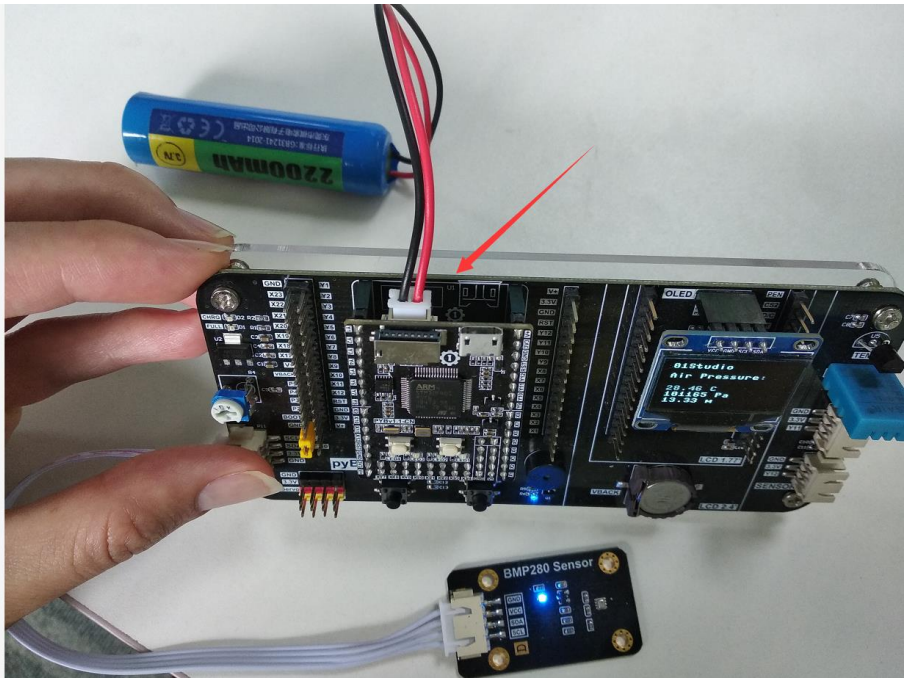


图 5-42 锂电池供电

- **总结：**

本节实现了大气压强传感器模块 BMP280 的应用，而且实验结果的精度很高，有兴趣的小伙伴可以结合自己情况，打造一个属于自己的气压计，然后到户外暴走！

5.8 超声波传感器

- **前言：**

超声波传感器是一款测量距离的传感器。其原理是利用声波在遇到障碍物反射接收结合声波在空气中传播的速度计算的得出。在测量、避障小车，无人驾驶等领域都有相关应用。

- **实验平台：**

MicroPython 开发套件和超声波传感器模块，传感器模块连接到 I2C/UART 扩展接口。

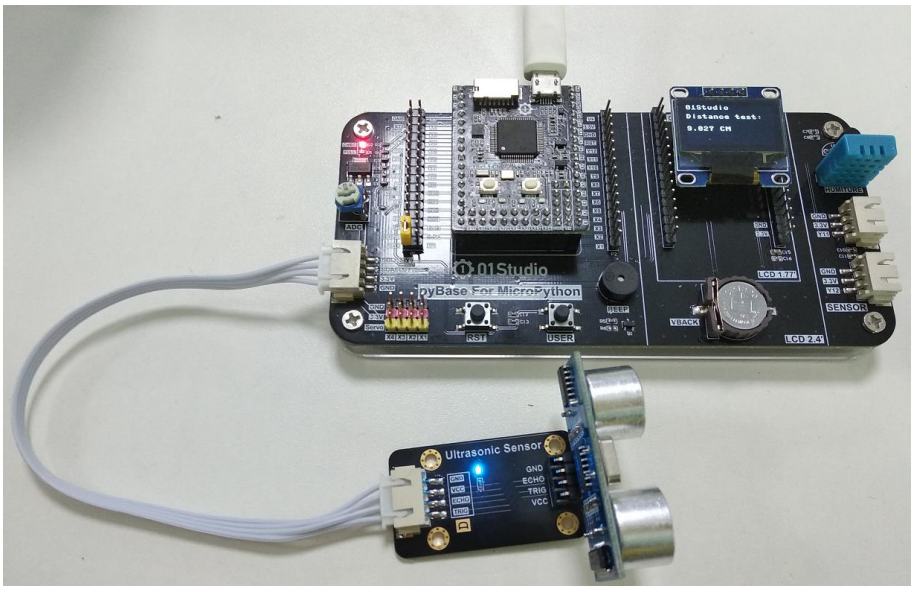


图 5-43 开发套件与传感器连接

- **实验目的：**

测量距离并在 OLED 上显示相关数据。

- **实验讲解：**

我们先来看看超声波传感器模块的介绍：

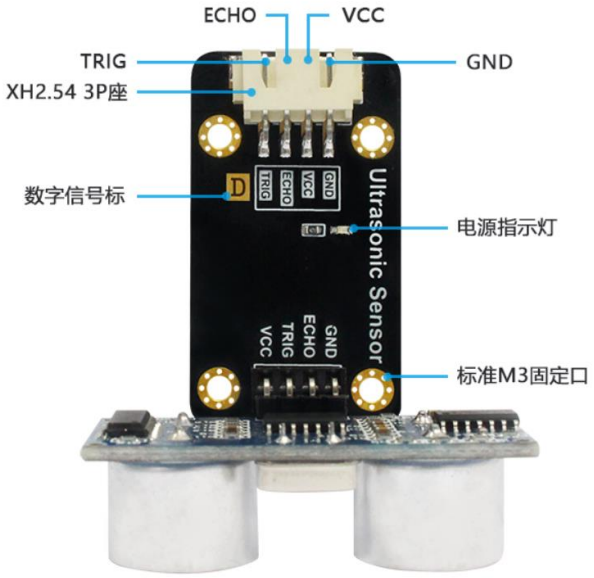


图 5-44 超声波模块

功能参数	
传感器型号	HCSR04
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85℃
接口定义	XH2.54 防呆接口（4Pin） 【GND、VCC、ECHO、TRIG】
通讯信号	IO 数字接口
测量距离	2-450 cm
测量精度	0.5 cm
整体尺寸	5.5*4.5*3.0 cm

表 5-10

超声波传感器模块使用两个 IO 口分别控制超声波发送和接收，工作原理如下：

1. 给超声波模块接入电源和地；
2. 给脉冲触发引脚（trig）输入一个长为 20us 的高电平方波；

3. 输入方波后，模块会自动发射 8 个 40KHz 的声波，与此同时回波引脚（echo）端的电平会由 0 变为 1；（此时应该启动定时器计时）
4. 当超声波返回被模块接收到时，回波引 脚端的电平会由 1 变为 0；（此时应该停止定时器计数），定时器记下的这个时间即为超声波由发射到返回的总时长；
5. 根据声音在空气中的速度为 340 米/秒，即可计算出所测的距离。

要学习和应用传感器，学会看懂传感器的时序图是很关键的，所以我们来看一下超声波传感器 HCSR04 的时序触发图。

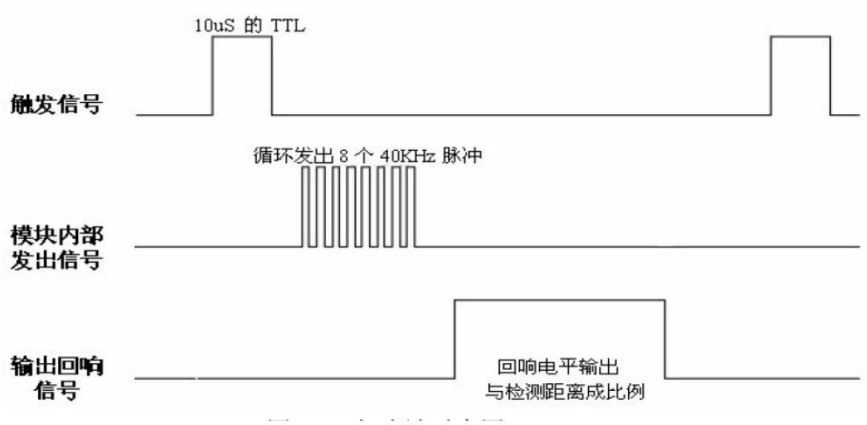


图 5-45 超声波传感器 HCSR04 时序图

以上普及了超声波传感器的原理，我们已经将其集成在 HCSR04.py 文件，如想了解代码原理可以打开 HCSR04.py 文件查看代码实现原理。使用 MicroPython 开发的用户只需要直接使用即可。使用方法如下：

构造函数	说明
HCSR04.HCSR04(trig,echo)	定义连接传感器的 trig 和 echo 引脚
使用方法	说明
getDistance()	获取距离数据

表 5-11 超声波传感器 HCSR04 对象

从上表看到，超声波传感器的使用方法非常简单，使用 2 个代码就完成对距离信息的采集，代码编写流程如下：

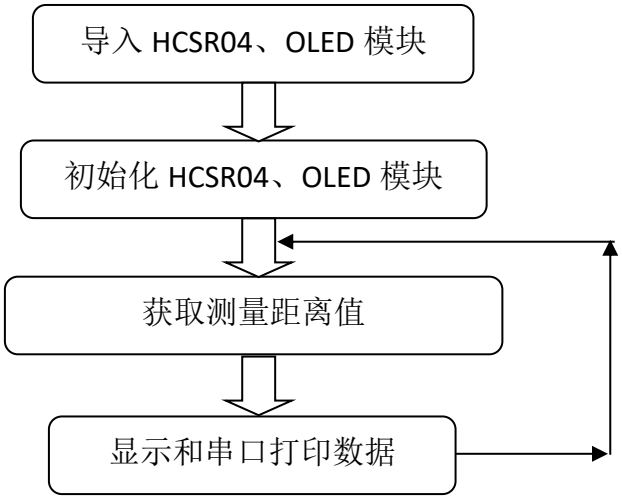


图 5-46 代码编写流程

主程序 main.py 参考代码如下：

```
'''
实验名称：超声波传感器
版本：v1.0
日期：2019.5.1
作者：01Studio 【www.01Studio.org】
说明：通过超声波传感器测距，并在 OLED 上显示。
'''

from HCSR04 import HCSR04     #子文件夹下的调用方式
from pyb import Pin,delay
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C

#初始化 OLED
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
```

```
#初始化接口 trig='Y9',echo='Y10'
trig = Pin('Y9',Pin.OUT_PP)
echo = Pin('Y10',Pin.IN)
HC=HCSR04(trig,echo)

while True:

    oled.fill(0)  # 清屏,背景黑色
    oled.text('01Studio', 0, 0)
    oled.text('Distance test:', 0, 15)

    Distance = HC.getDistance() #测量距离

    # OLED 显示距离
    oled.text(str(Distance) + ' CM', 0, 35)

    oled.show()

    #串口打印
    print(str(Distance)+' CM')

    delay(1000) #每秒采集 1 次
```

- **实验结果：**

在超声波传感器 30cm 外放置障碍物，可以看到 OLED 显示屏实时显示距离数据约为 30cm。

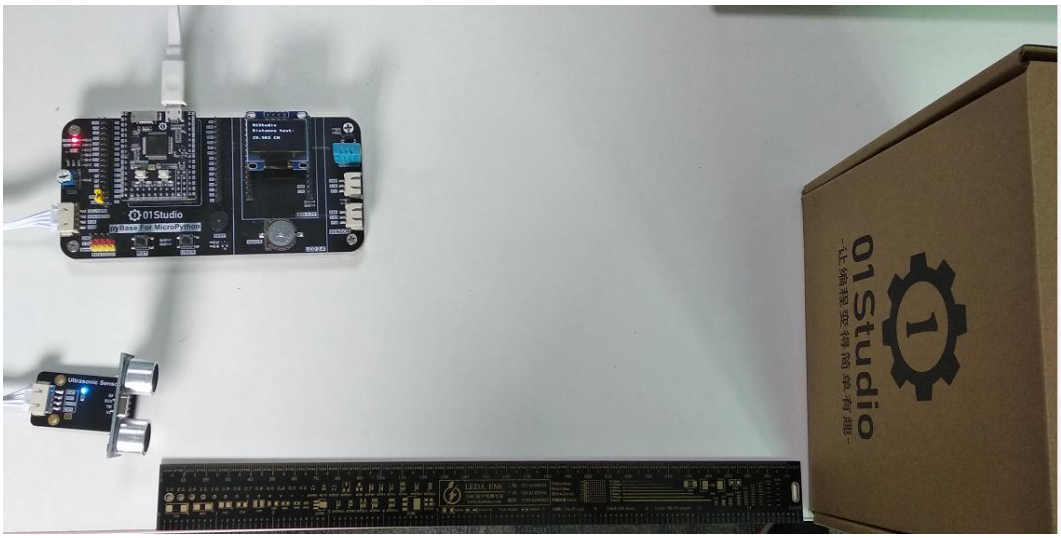


图 5-47 障碍物距离为 30cm

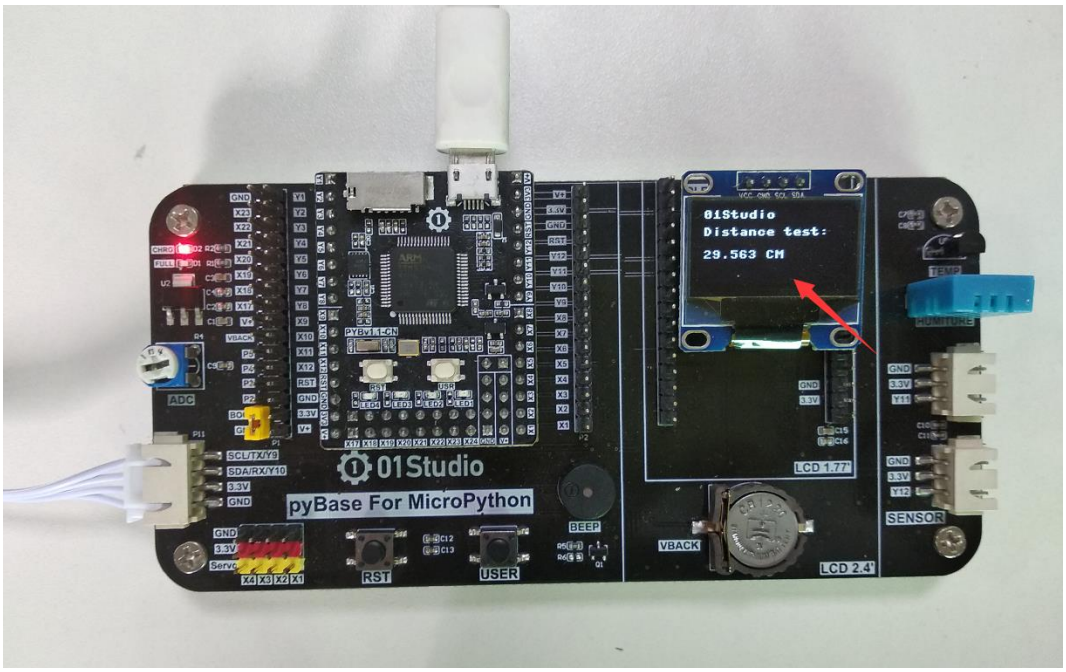


图 5-48 实验结果约为 30cm

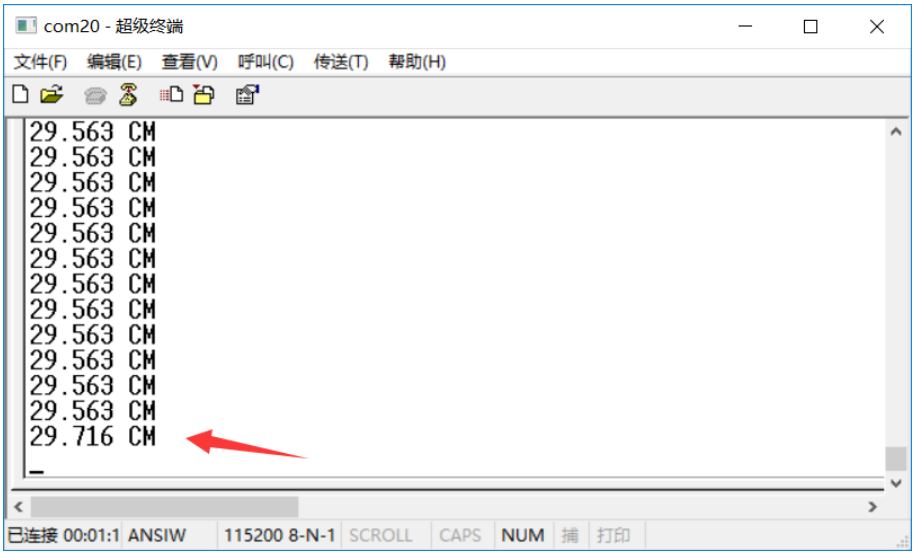


图 5-49 超级终端同步打印实验结果

- **总结：**

出去 OLED 显示代码，我们实际上只用了 2 行代码：初始化和调用测量函数就实现了对超神波传感器测距的应用。这让我们再一次感受到了 MicroPython 的魅力。赶快动作制作自己的避障小车和其他好玩的创作吧。

第6章　拓展实验

在学习了 MicroPython 前面内容后，相信我们已经对其编程方法有了一定的认识，那么除了较为简单的基础实验外，我们还可以使用 MicroPython 开发板连接非常多的外接设备，如常见的舵机、LCD 等。

这些外设的编程难度分两块：如果是仅仅面向应用的话，那么我们可以使用非常简单的方式就用起来；由于代码是完全开源，所以如果想深入学习，则可以看一下驱动代码的实验原理，甚至自行编写驱动程序。从而一起丰富 MicroPython 的周边的生态产品。

6.1 舵机

- **前言：**

舵机又叫伺服电机，是一个可以旋转特定角度的电机，可转动角度通常是 90°、180°和 360°（360°可以连续旋转）。我们看到的机器人身上就有非常多的舵机，它们抬手或者摇头的动作往往是通过舵机完成，因此机器人身上的舵机越多，意味着动作越灵活。

- **实验平台：**

MicroPython 开发套件和舵机。

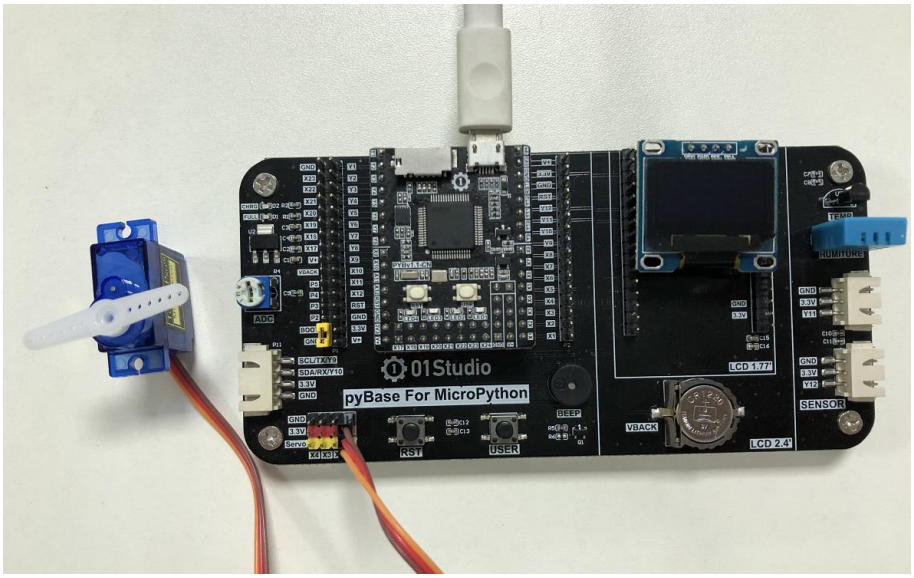


图 6-1 舵机接线

通常情况下：**黑色表示 GND，红色表示 VCC，橙色表示信号线。**

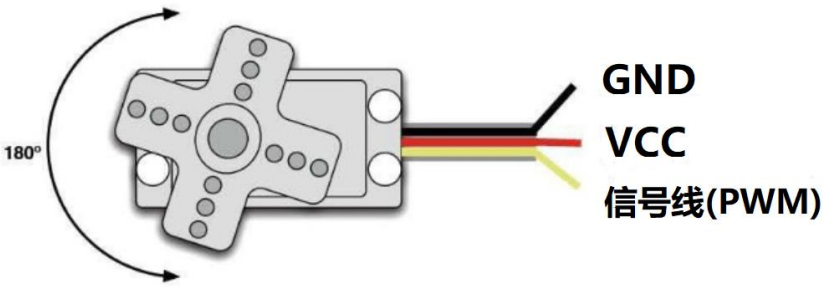


图 6-2 舵机示意图

- **实验目的：**

通过编程实现对舵机的控制。

- **实验讲解：**

伺服电机对象通过 3 线（信号，电源，地）控制，pyboard 和 MicroPyhon 开发板上有 4 个位置可以插这些电机，分别对应 X1-X4 引脚。

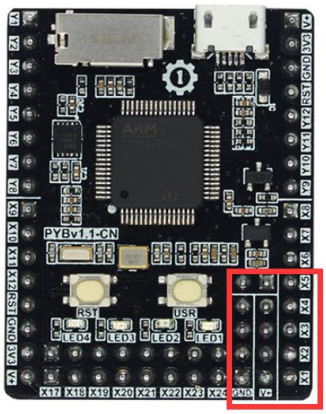


图 6-3 pyBoard v1.1-CN4 路舵机接口

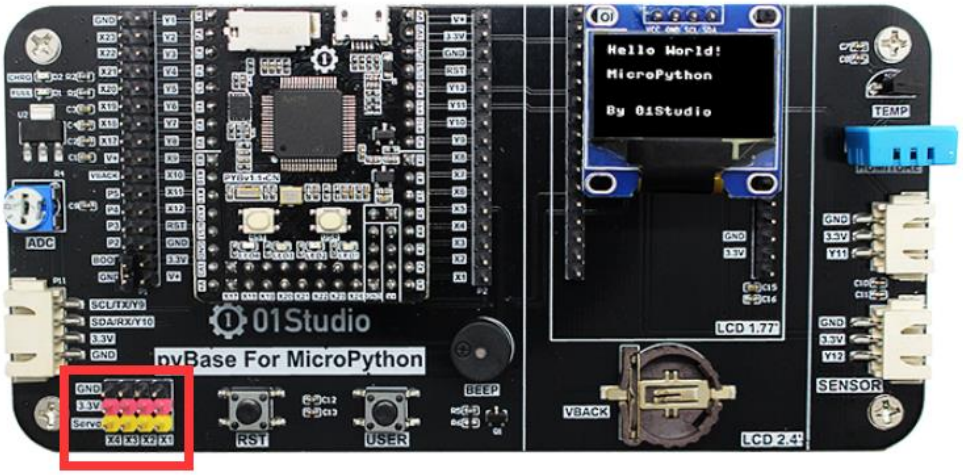


图 6-4 MicroPython 开发板 4 路舵机接口

180° 舵机的控制一般需要一个 20ms 左右的时基脉冲，该脉冲的高电平部分一般为 0.5ms-2.5ms 范围内的角度控制脉冲部分，总间隔为 2ms。以 180 度角度伺服为例，在 MicroPython 编程对应的控制关系是从-90° 至 90° ，示例图如下：

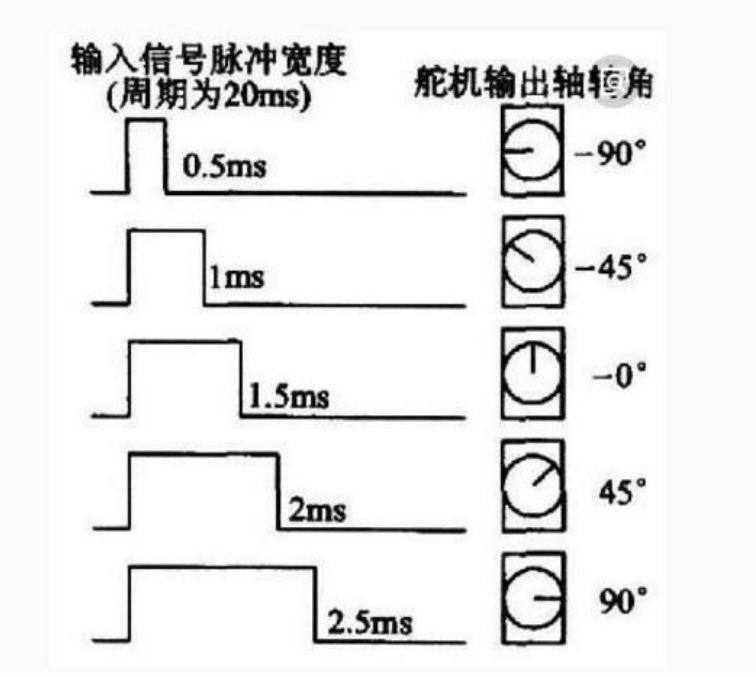


图 6-5 PWM 与角度关系

而对于 360° 连续旋转舵机，上面的脉冲表则对应从正向最大速度旋转到反向最大速度旋转的过程。

pyBorad 固件自带了舵机（Servo）的对象模块，我们可以直接调用，下面是 Servo 对象说明。

构造函数
`s=pyb.Servo(id)`
构建舵机对象。id 是 1-4，对应引脚 X1-X4
使用方法
`s.angle (angle,time=0)`
角度控制。angle:角度[-90 至 90]；time:旋转到指定角度的时间，单位 ms，默认 0 表示最快
`s.speed(speed,time=0)`
速度控制（针对 360 度连续旋转舵机）。speed:速度[-100 至 100]，time：开始启动到指定速度的时间，单位 ms，默认 0 表示以最快

表 6-1 舵机对象

可以看到 MicroPython 让舵机控制变得非常简单，代码编程流程图如下：

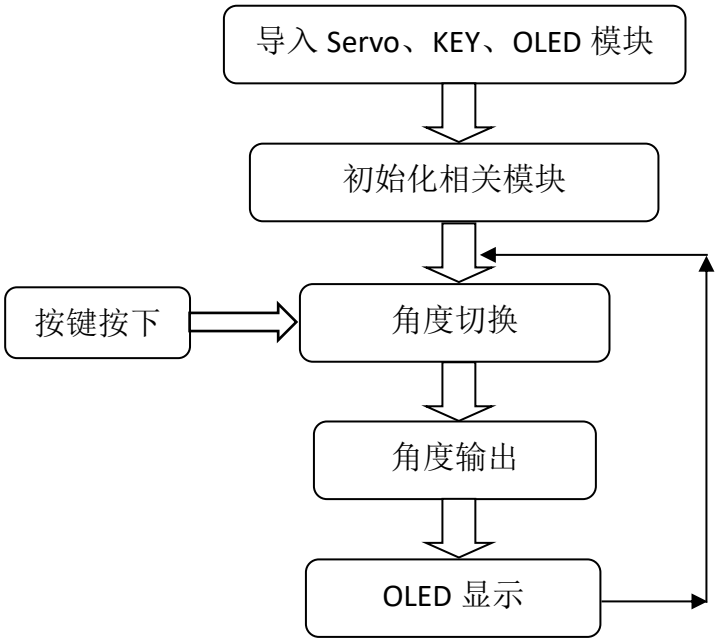


图 6-6 编程流程图

参考代码如下：

```
'''
实验名称：舵机(Servo)
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过 USER 按键让让舵机旋转不同角度。
'''

#导入相关模块
from pyb import Servo,Switch
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C
```

```
#初始化相关模块
i2c = I2C(sda=Pin("Y8"), scl=Pin("Y6"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

sw = Switch()     #定义按键对象名字为 sw
s1 = Servo(1)     #构建舵机对象 s1，输出引脚为 X1

#定义 5 组角度：-90、-45、0、45、90
angle=[-90,-45,0,45,90]

key_node = 0  #按键标志位
i = 0         #用于选择角度

################################################
#  按键和其回调函数
################################################
def key():
    global key_node
    key_node = 1

sw.callback(key)  #当按键被按下时，执行函数 key()

################################################
#  OLED 初始显示
################################################
oled.fill(0)  # 清屏显示黑色背景
oled.text('01Studio', 0, 0)  # 首行显示 01Studio
oled.text('Servo-180', 0, 15)  # 次行显示实验名称
oled.text(str(angle[i]) , 0, 35)  # 显示当前角度
```

```
oled.text('Pls Press USER', 0, 55)  #提示按按键
oled.show()

#X1 指定角度,启动时 i=0，默认-90°
s1.angle(angle[i])

while True:

    if key_node==1: #按键被按下
        i = i+1
        if i == 5:
            i = 0
        key_node = 0 #清空按键标志位

        #X1 指定角度
        s1.angle(angle[i])

        #显示当前频率
        oled.fill(0)  # 清屏显示黑色背景
        oled.text('01Studio', 0, 0)  # 首行显示 01Studio
        oled.text('Servo-180', 0, 15)  # 次行显示实验名称
        oled.text(str(angle[i]) , 0, 35)  # 显示当前角度
        oled.text('Pls Press USER', 0, 55)  #提示按按键
        oled.show()
```

- **实验结果：**

将 180°舵机插到 X1 三线接口，通过按键按下实现舵机的不同角度，同时在 OLED 上显示角度值。

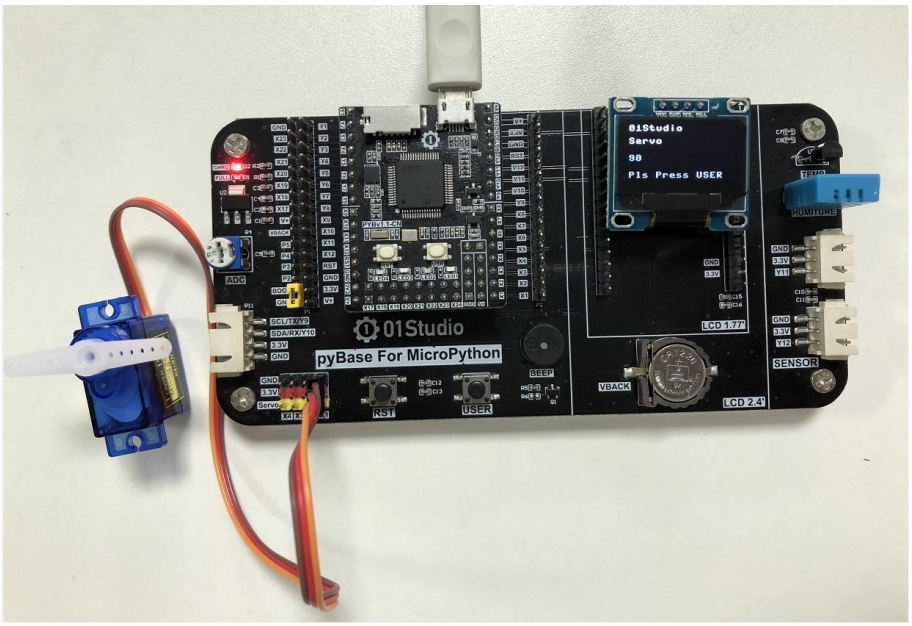


图 6-7 舵机角度 90°

- **实验拓展：**

我们刚刚实现了 180° 舵机的角度控制，现在来做一下 360° 连续旋转舵机的实验，360° 连续旋转舵机可以实现直流减速电机功能，用在小车或者航模上。

实验的基础条件和方式不变，我们只需要将角度参数【-90 至 90】改成速度【-100 至 100】即可。修改后的程序代码如下：

```
'''
实验名称：舵机(Servo)-360° 连续旋转
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过 USER 按键让舵机实现不同速度旋转。
'''

#导入相关模块
from pyb import Servo,Switch
from machine import Pin,I2C
from ssd1306 import SSD1306_I2C
```

```
#初始化相关模块
i2c = I2C(sda=Pin("X12"), scl=Pin("X11"))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

sw = Switch()     #定义按键对象名字为 sw
s1 = Servo(1)     #构建舵机对象 s1，输出引脚为 X1

#定义 5 组速度值
speed=[-100,-50,0,50,100]

key_node = 0  #按键标志位
i = 0         #用于选择角度

################################################
#  按键和其回调函数
################################################
def key():
    global key_node
    key_node = 1

sw.callback(key)  #当按键被按下时，执行函数 key()

################################################
#  OLED 初始显示
################################################
oled.fill(0)  # 清屏显示黑色背景
oled.text('01Studio', 0, 0)  # 首行显示 01Studio
oled.text('Servo-360', 0, 15)  # 次行显示实验名称
oled.text(str(speed[i]) , 0, 35)  # 显示当前速度值
```

```
oled.text('Pls Press USER', 0, 55)  #提示按按键
oled.show()

#X1 指定角度,启动时 i=0，默认-90°
s1.speed(speed[i])

while True:

    if key_node==1: #按键被按下
        i = i+1
        if i == 5:
            i = 0
        key_node = 0 #清空按键标志位

        #X1 指定角度
        s1.speed(speed[i])

        #显示当前频率
        oled.fill(0)  # 清屏显示黑色背景
        oled.text('01Studio', 0, 0)  # 首行显示 01Studio
        oled.text('Servo-360', 0, 15)  # 次行显示实验名称
        oled.text(str(speed[i]) , 0, 35)  # 显示当前速度值
        oled.text('Pls Press USER', 0, 55)  #提示按按键
        oled.show()
```

实验结果如下：

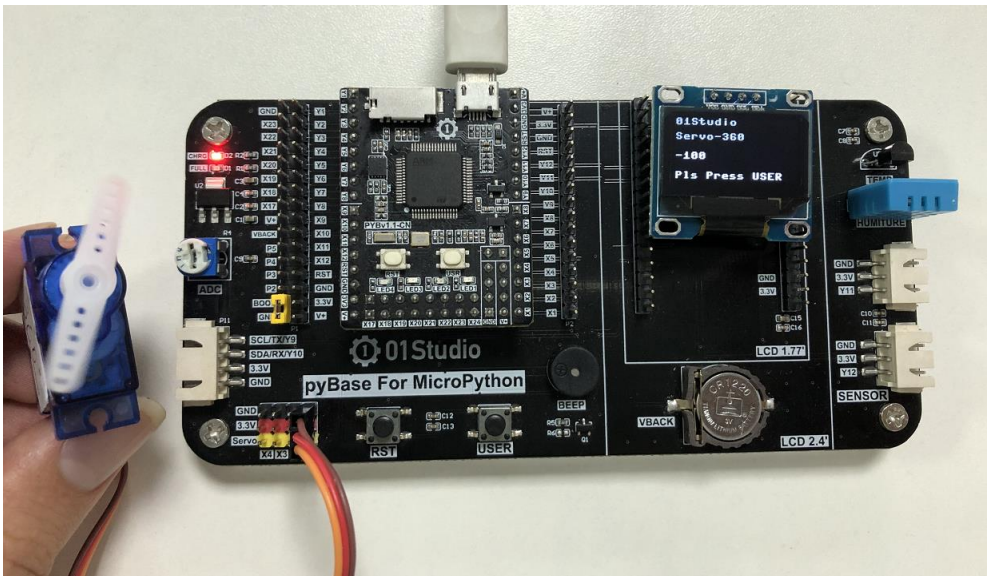


图 6-8　360°连续旋转舵机

- **总结：**

通过本节我们学会了使用不同类型的舵机，通过多路电机的组合使用，可以实现模型、航模、小车、机器人的实验。

6.2 RGB 灯带

- **前言：**

RGB 灯带是彩灯的一种，我们看到长长的灯带以及 LED 彩色显示屏，都是采用一个个灯珠组合而成。在本章节我们将通过编程实现 RGB 灯带的控制，可以应用到家居布置、节日气氛的场景中去。

- **实验平台：**

MicroPython 开发套件和 RGB 灯带。连接到“Y11”接口。

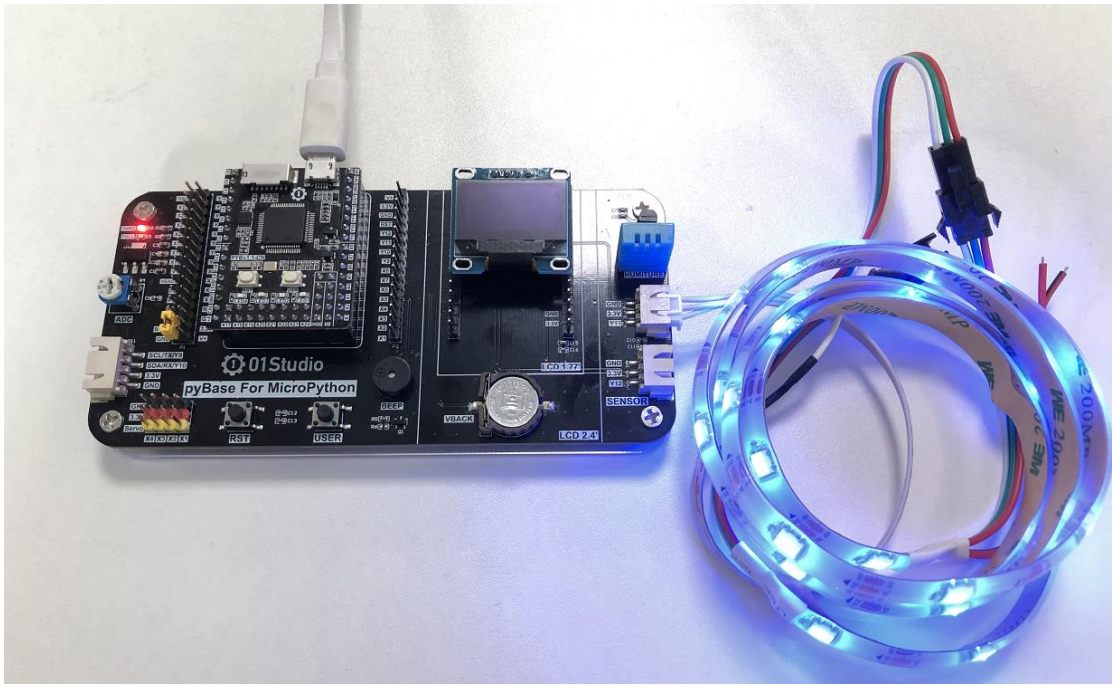


图 6-9 RGB 灯带接线方式

- **实验目的：**

通过编程实现灯带循环显示红色（RED）、绿色（GREEN）和蓝色（BLUE）。

- **实验讲解：**

先来介绍一下本实验用到的 RGB 灯带。

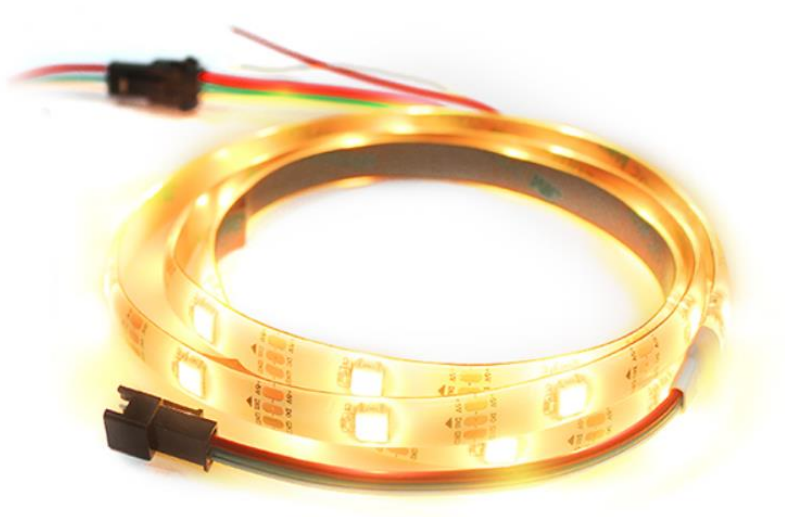


图 6-10 RGB 灯带

该灯带为 3 线接口，长 1 米，有 30 颗灯珠，每个灯珠里面都有一个驱动 IC，型号是 WS2812B，单总线驱动。每个灯珠首尾相连。灯带末端预留了接口可以串联更多的灯带。也就是说只需要通过 1 个 GPIO 口就可以控制了数十上百的灯珠了。

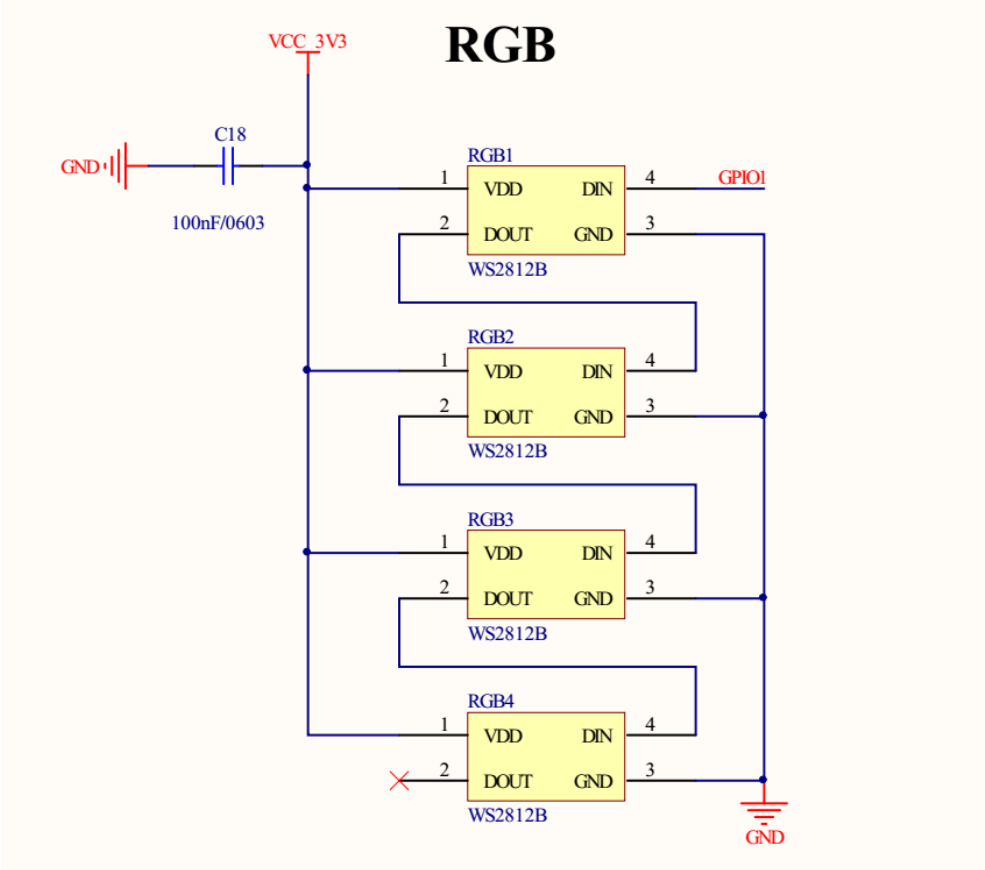


图 6-11 灯珠串联接线方式

了解了接线方式后，我们知道颜色是由最基本的三种颜色的不同亮度混合出新颜色。这 3 个最基本的颜色顺序分别是红，绿，蓝（RGB）。这里每个颜色的亮度级别从 0-255,0 表示没有，255 表示最亮。如（255,0,0）则表示红色最亮。

GPIO 口就是将这些数据逐一发送给灯带。本实验灯带有 30 个灯珠，则连续发送 30 次，通过 SPI 方式发送。可见在灯珠数量多的时候，传输数据量是非常大的，这些我们通过驱动文件 ws2812.py 实现，这使得用户可以轻松地使用 MicroPython 编程实现相关功能，对象说明如下：

构造函数
`strip=WS2812(spi_bus, led_count)`
构建灯带对象。spi_bus:灯带控制引脚（MOSI），led_count:灯珠数量；
使用方法
`strip.show(data)`
灯带显示。data:颜色数据。例：1 个灯珠显示红色为 data=[(255,0,0)]; 2 个灯珠显示绿色为 data=[(0,255,0),(0,255,0)];如此类推。

表 6-2　灯带对象

编程流程图如下：

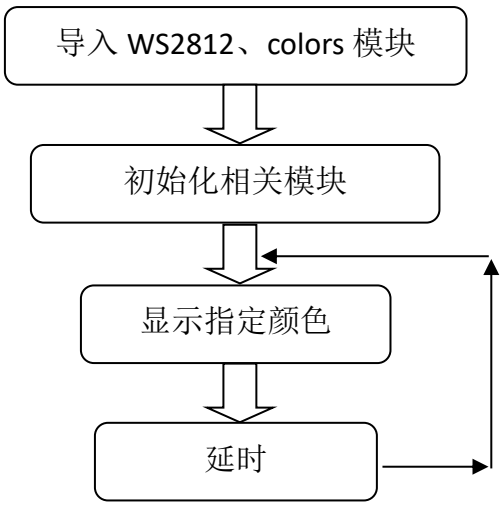


图 6-12 编程流程图

main.py 参考程序代码如下：

```
'''
实验名称：RGB 灯带
版本：v1.0
日期：2019.7
作者：01Studio
说明：RGB 灯带控制。
'''

from ws2812 import WS2812
from colors import *
from machine import Pin
import pyb

#定义灯带连接引脚，Y11 接口
LED = Pin('Y11',Pin.OUT,value=0)

#构建 RGB 灯带对象,定义控制引脚和灯珠数量
strip = WS2812(spi_bus=LED, led_count=30)

#灯带填色函数,灯珠数量为 led_count
def fill_color(color):
    data=[]
    for i in range (strip.led_count):
        data.append(color)
    return data

#清空 RGB 灯带颜色
strip.show(fill_color(EMPTY))
```

```
while True:
    strip.show(fill_color(RED))
    pyb.delay(1000)

    strip.show(fill_color(GREEN))
    pyb.delay(1000)

    strip.show(fill_color(BLUE))
    pyb.delay(1000)
```

我们简单讲解一下本实验代码：

主文件的颜色可以直接调用是因为在 colors.py 文件中包含了颜色的定义，用户可以根据自己的实际情况来补充。

```
#colors define
EMPTY=(0,0,0)
RED=(255,0,0)
GREEN=(0,255,0)
BLUE=(0,0,255)
WHITE=(255,255,255)
```

另外我们定义了一个颜色填充函数，里面是一个循环，append 的作用是在后面添加数据，返回的结果 data[]是对灯带各个灯珠颜色的定义。

```
#灯带填色函数,灯珠数量为 led_count
def fill_color(color):
    data=[]
    for i in range (strip.led_count):
        data.append(color)
    return data
```

- **实验结果：**

将程序下载到 MicroPython 开发板，可以看到灯带的颜色在红、绿、蓝三色中循环变化。

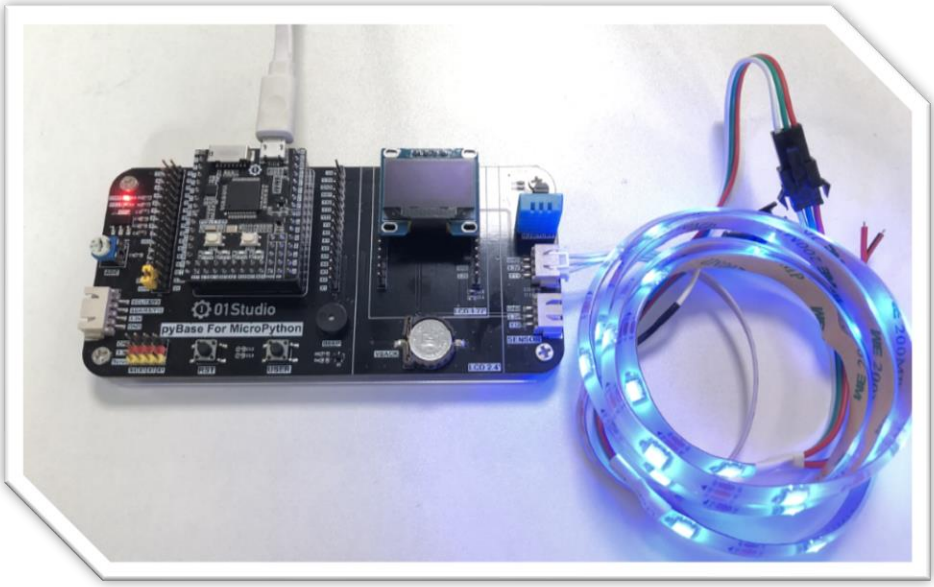


图 6-13 实验结果

- **总结：**

RGB 灯带的单总线特点使得我们可以轻易的增加和减少灯带的数量而无需过多的修改程序。但需要注意的是如果灯带数量过多，那么需要外接电源供电，否则会因为电流不足而影响使用。

6.3 LCD 液晶显示屏

- **前言：**

LCD 液晶显示屏是非常常见的一个外接显示设备，跟前面的 OLED 显示屏相比，LCD 会更常用一些，我们看到的手持设备、小型电器，很多都用到 LCD，部分配合触摸屏应用，能实现非常多的功能。

- **实验平台：**

MicroPython 开发套件、2.4 寸 LCD（带触摸）和 MicroSD 卡。

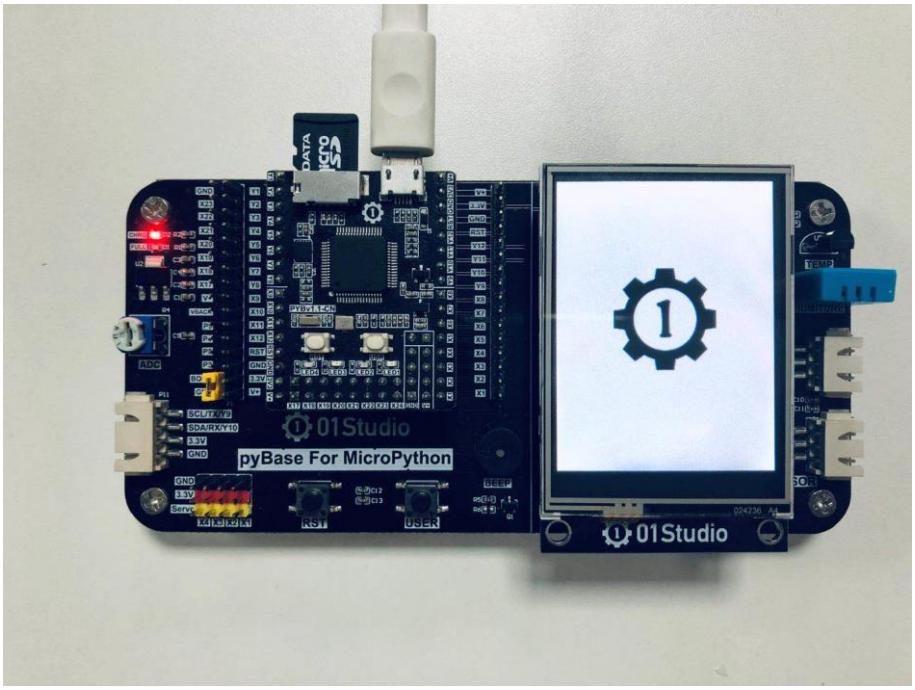


图 6-14 2.4 寸 LCD

- **实验目的：**

通过 MicorPython 编程方式实现 LCD 的各种显示功能，包括画点、线、矩形、圆形、显示英文、显示图片等。

- **实验讲解：**

我们先来看看实验所使用 LCD 的参数介绍：

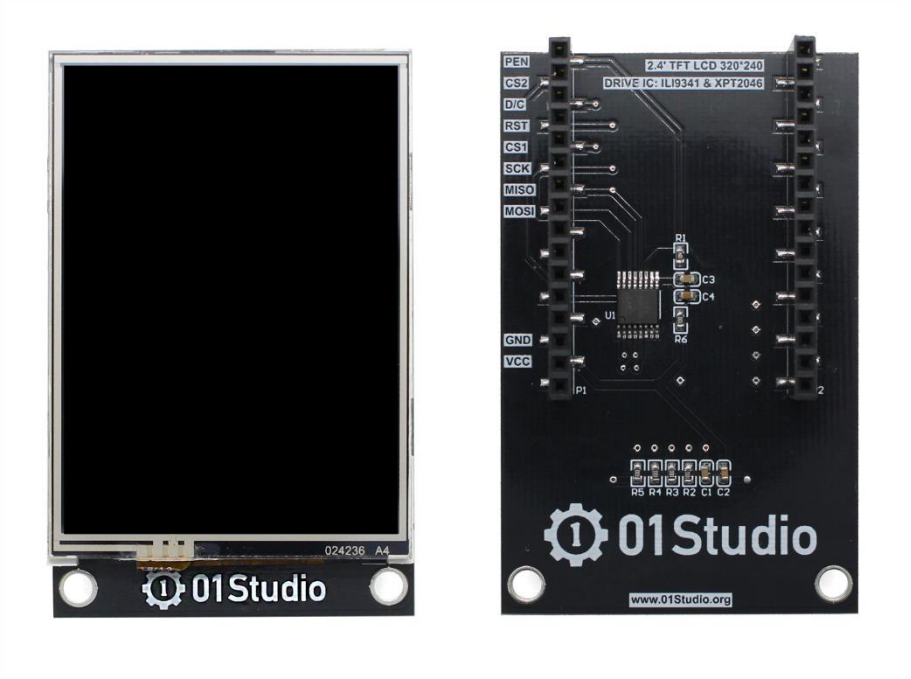


图 6-15　2.4 寸 LCD 显示屏（带触摸）

功能参数	
供电电压	3.3V
屏幕尺寸	2.4 寸
颜色参数	TFT 彩色
驱动芯片	ILI9341+XPT2046
触摸方式	电阻屏
通讯方式	SPI 总线
接口定义	2.54mm 排母（兼容 pyboard v1.1 接口）
整体尺寸	6.6*4.2 cm

表 6-3　LCD 功能参数

2.4 寸 LCD 跟 pyboard 的的引脚连线如下表，触摸的 PEN 和 CS2 在本实验暂没用到，从下表和图可以看到 LCD 实际连接到 pyboard 的 Y1-Y8, SPI(2)接口。

pyboard	LCD	LCD 引脚说明
Y1	PEN	触摸屏中断
Y2	CS2	触摸屏片选
Y3	D/C	数据/命令选择
Y4	RST	复位
Y5	CS1	LCD 片选
Y6	SCK	SPI 接口
Y7	MISO	SPI 接口
Y8	MOSI	SPI 接口
3.3V	VCC	电源（3.3V 供电）
GND	GND	地

图 6-16 LCD 引脚连接和说明

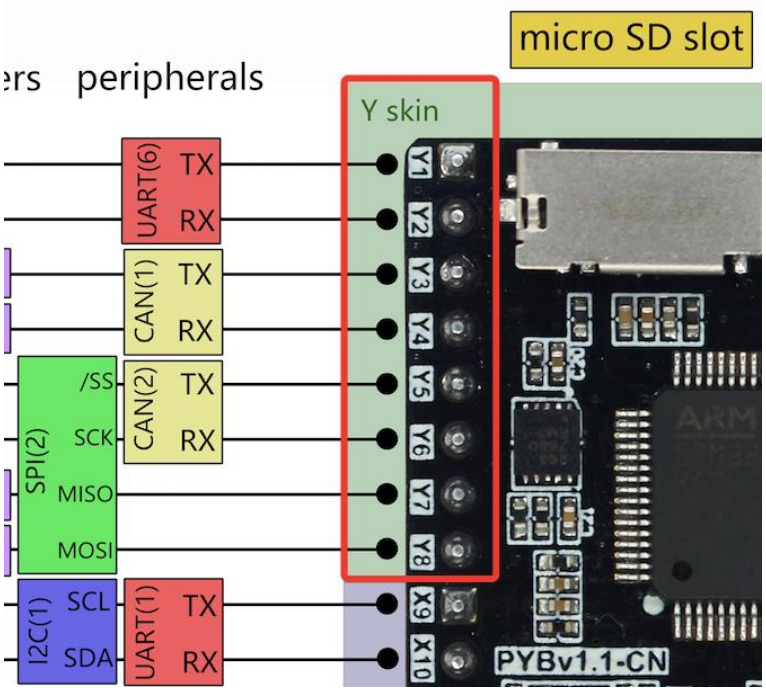


图 6-17 LCD 连接到 Y1-Y8

实验用的 LCD 是 2.4 寸，驱动是常见的 ILI9341，使用 SPI 接口跟 pyboard 通信，按以往嵌入式 C 语言开发，我们需要对 ILI9341 进行编程实现驱动，然后再建立各种描点、划线、以及显示图片函数。

使用 MicroPython 其实也需要做以上工作，但由于可读性和移植性强的特点，我们只需要搞清各个对象函数使如何使用即可。总的来说和之前一样，有构造函数和功能函数。构造函数解决的是初始化问题，告诉 pyboard 外设是怎么接线，是什么样的；而功能函数解决的则是使用问题，我们基于自己的需求直接调用相关功能函数，实现自己的功能即可！

我们管这些函数的集合叫驱动，这里我们打开配例程，先来看示例程序里面 LCD 的驱动程序，下图是简要的说明：

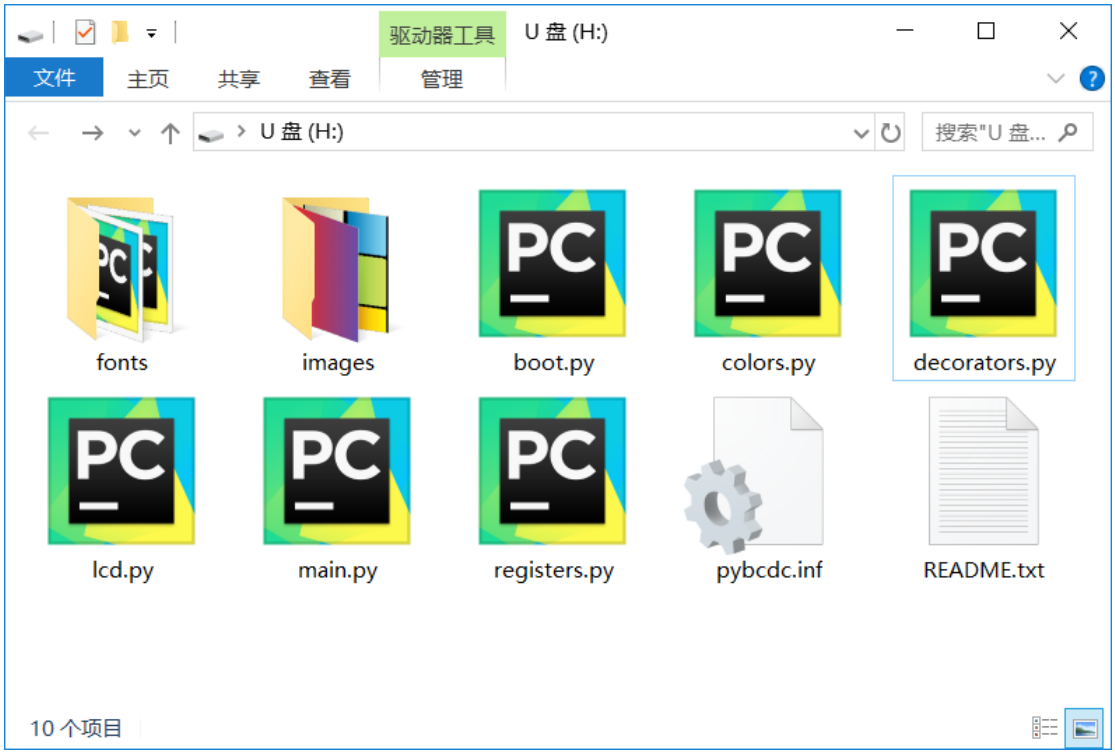


图 6-18 LCD 驱动程序

主要驱动文件说明	
lcd.py	主要驱动程序和功能函数定义
colors.py	颜色定义
fonts.py	字体库
images	图片存放处

表 6-4 驱动文件说明

由于本实验驱动文件比较大，所以在 SD 卡上进行，我们可以将以上文件拷贝到 SD 卡，pyboard 上电检测到 SD 卡会自动从 SD 卡开始运行。

打开 lcd.py，我们可以看到定义的类和函数，常用功能函数整理如下，如需了解全部函数，可以自行阅读 lcd.py 代码：

构造函数
d=LCD(rstPin='Y4', csxPin='Y5', dcxPin='Y3',port=2, portrait=True)
构建 LCD 对象 d。rstPin:复位，csxPin:片选，dcxPin：数据/指令，port=2: SPI(2)，portrait:True(竖屏)，False(横屏)
c = d.initCh(color, bgcolor, font)
构建字符对象 c。color:颜色，bgcolor:字符背景颜色，font:字体
使用方法
d. fillMonocolor (color)
填充：color:颜色
d.drawPixel(x, y, color, pixels=4)
画点，x:横坐标，y:纵坐标，color:颜色，pixels:像素[1-4]
d.drawVline(x, y, length, color, width=1)
画垂直线，x:横坐标，y:纵坐标，length:长度，color:颜色，width:线宽[1-10]
d.drawHline(x, y, length, color, width=1)
画水平线，x:横坐标，y:纵坐标，length:长度，color:颜色，width:线宽[1-10]
drawLine(self, x, y, x1, y1, color)
画线段，x,y:起始坐标，x1,y1:终点坐标，color:颜色
d.drawRect(x, y, width, height, color, border=1, fillcolor=None)
画矩形，x,y:起始坐标，width:宽度，length:长度，color:颜色，border:边宽，fillcolor:填充颜色，None 为不填充
d.drawCircle(x, y, radius, color, border=1, degrees=360, startangle=0):
画圆。x, y:圆心，radius:半径，color:颜色，border:边宽，degrees:圆弧度数，startangle:起始角度

c.printChar(char, x, y,)
写单个字符，char:字符，x,y:起始坐标
c.printLn(string, x, y,)
写字符串，string:字符串，x,y:起始坐标
d.renderBmp(filename, pos=None)
显示图片，filename: images 文件夹下的 BMP 文件名称；pos：起始坐标，格式（x,y）。***注意事项：BMP 文件格式是 16 位(R5,G6,B5)格式。分辨率小于 240*320。**

表 6-5　LCD 对象

有了上面的对象构造函数和使用说明，编程可以说是信手拈来了，我们在使用中将以上功能都跑一遍，顺便计算一下刷屏速度，先看看编程流程图：

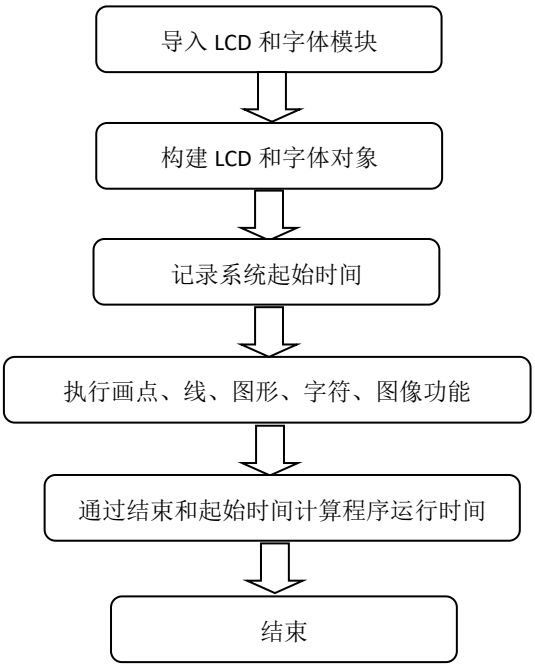


图 6-19 编程流程图

参考 main.py 函数代码如下：

```
'''
实验名称：LCD 液晶显示屏
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过编程实现 LCD 的各种显示功能，包括画点、线、矩形、圆形、显示英文、显示图片等。
'''

from lcd import *
from fonts.arial_14 import Arial_14

starttime = pyb.micros()//1000 #测试刷屏起始时间

########################
    #构建对象并初始化#
########################

#构建 LCD 对象
d = LCD() # or d = LCD(portrait=False) for landscape

#构建字符对象
c = d.initCh(color=BLACK, bgcolor=WHITE, font=Arial_14)

########################
    #函数使用方法#
########################

#填充
d.fillMonocolor(WHITE)
```

```
#画点
d.drawPixel(5, 5, RED, pixels=4)

#画垂直线
d.drawVline(5, 10, 10, RED, width=1)

#画水平线
d.drawHline(5, 25, 30, RED, width=1)

#画线段
d.drawLine(5, 30, 25, 50, RED)

#画矩形
d.drawRect(5, 60, 30, 20, RED, border=1)

#画圆
d.drawCircle(30, 120, 20, RED, border=1, degrees=360, startangle=0)

#写字符
c.printChar('@', 20, 160)

#写字符串行
c.printLn('Hello 01Studio', 20, 200)

pyb.delay(3000) #延时 3 秒方便观察

#显示图片
d.renderBmp("01Studio.bmp", (0, 0))
```

```
# 测试刷屏速度(减去 3000ms 延时)
print('executed in:', (pyb.micros()//1000-starttime-3000)/1000,
'seconds')
```

- **实验结果：**

将示例程序所有文件拷贝到 SD 卡（pyboard 自带文件系统容量不足。）复位后自动从 SD 卡运行。可以看到 LCD 一次执行相关显示功能，等待 3 秒后显示 01Stuido LOGO。

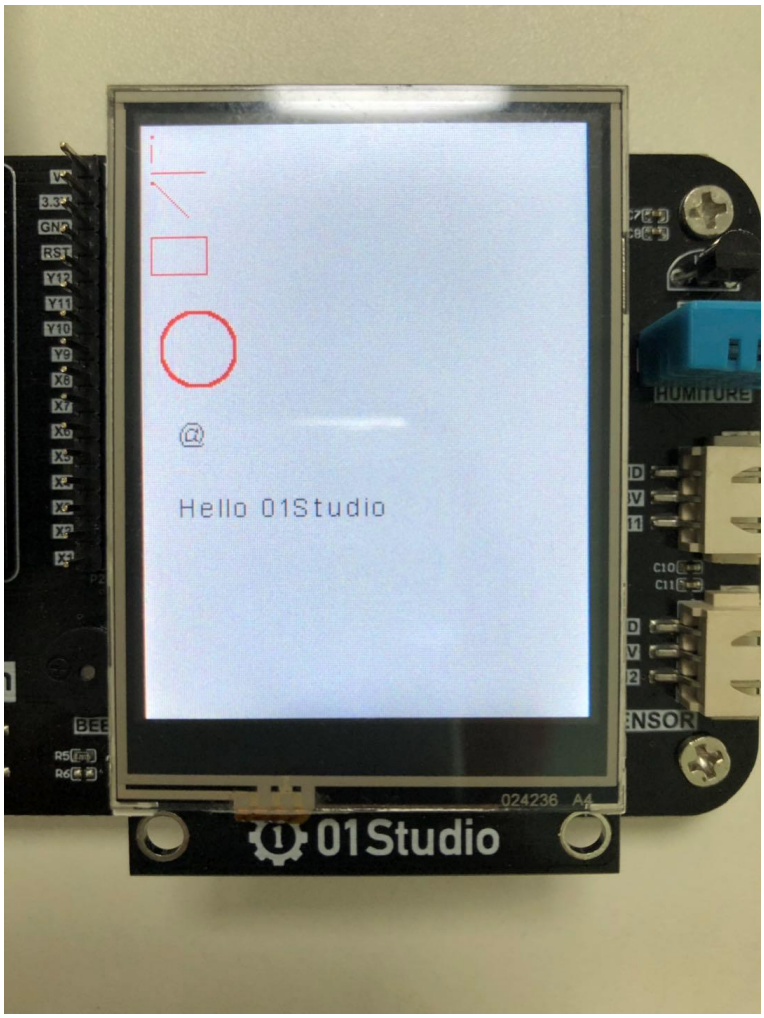


图 6-20

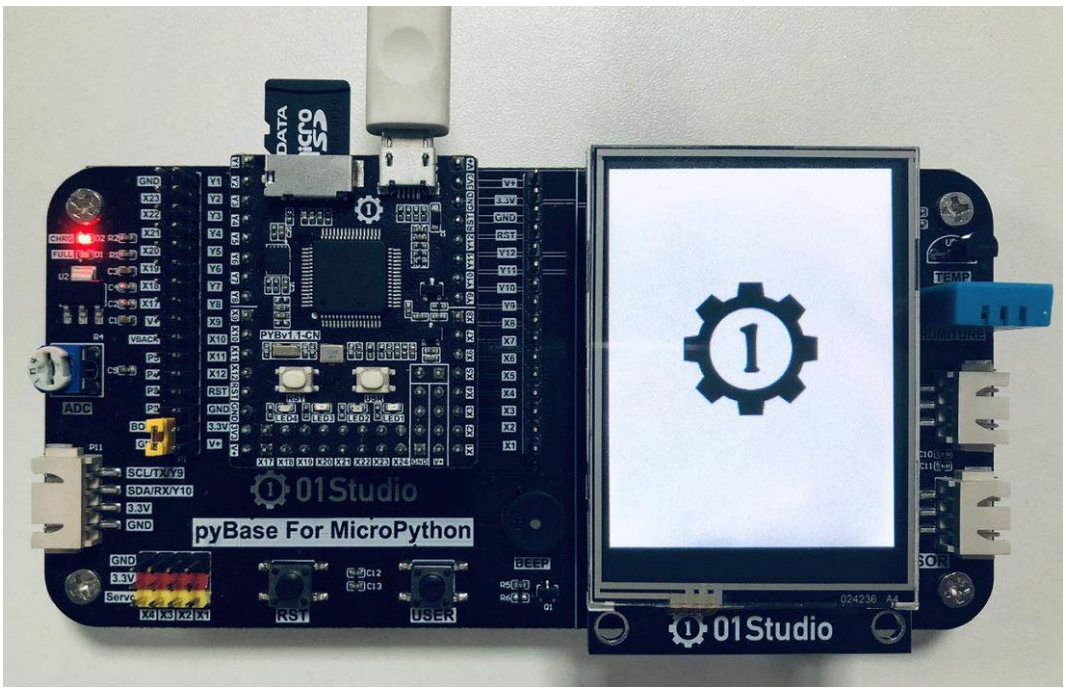


图 6-21 显示 LOGO

程序结束后在串口打印实际运行耗时。

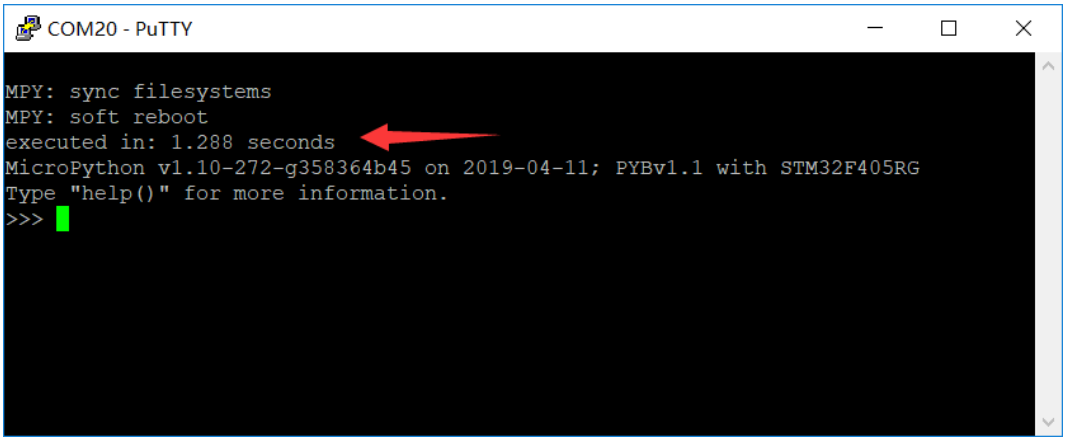
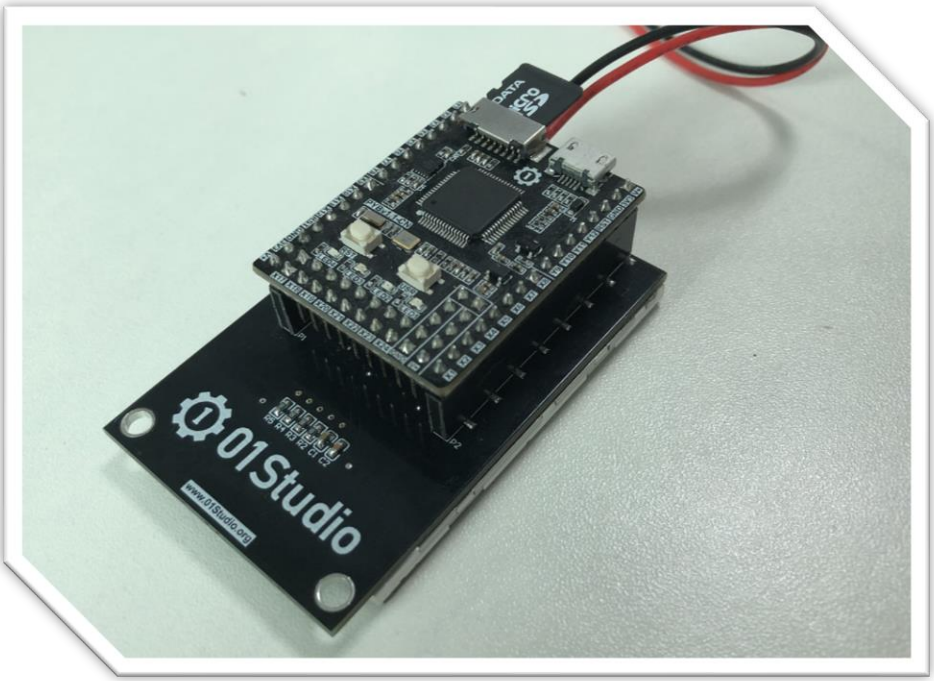


图 6-22 串口显示刷屏耗时

由于我们 LCD 接口兼容 pyboard，因此也可以直接将 pyboard v1.1-CN 插到 LCD 背面，使用锂电池供电。如下图所示：



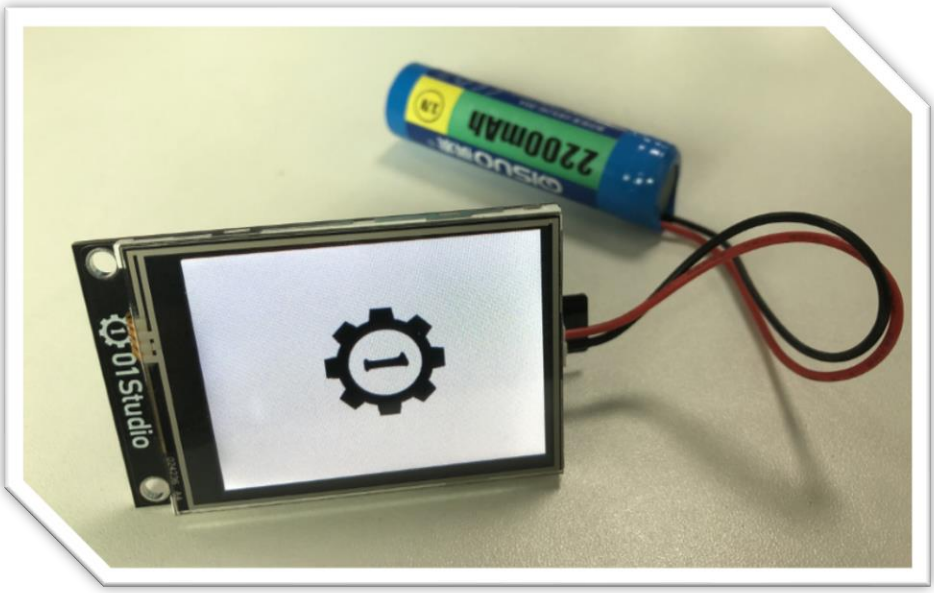


图 6-23　LCD 直接连接 pyboard v1.1-CN

- **总结：**

本实验用到的 LCD 和其驱动芯片 ILI9341 是常见的产品，可以看到 MicroPython 编程只关注屏幕的驱动和分辨率，对于不用尺寸和型号的 LCD 来说，移植性非常强，有兴趣的朋友可以自行驱动其它系列的 LCD。

6.4 以太网模块

我们先来看看 01Studio W5500 以太网模块参数介绍：

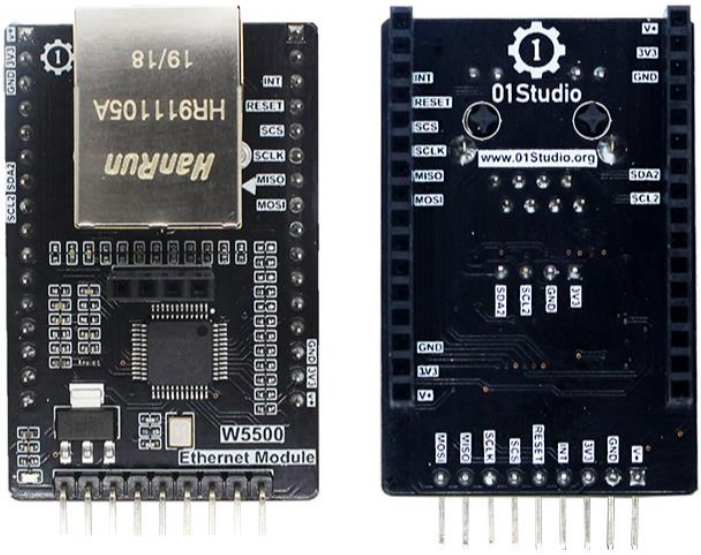


图 6-24 W5500 以太网模块

功能参数	
供电电压	3.3V 或 5V
主控芯片	W5500
控制方式	SPI 总线
接口定义	1、2.54mm 排母（兼容 pyboard v1.1 接口）； 2、2.54mm 排针
其它功能	支持 OLED 扩展接口
整体尺寸	6.6*4.2 cm

表 6-6

pyBoard 推荐引脚接口如下：

pyboard	W5500 以太网模块	引脚说明
Y3	INT	中断
Y4	RST	复位
Y5	CS	片选
Y6	SCK	SPI 接口
Y7	MISO	SPI 接口
Y8	MOSI	SPI 接口
3.3V	VCC	电源（3.3V 供电）
GND	GND	地
V+	V+	电源（3.6-6V 输入）

表 6-7　推荐接线

pyBoard v1.1-CN 可以通过连接 W5500 以太网模块实现网线连接路由器上网。需要先给 pyBoard v1.1-CN 烧录带 network 的 pyboard 固件，路径在：**零一科技（01Studio）MicroPython 开发套件配套资料\03-相关固件\01-pyboard v1.1\网络版，**烧写方法参考前面开发环境建立—固件更新章节内容。

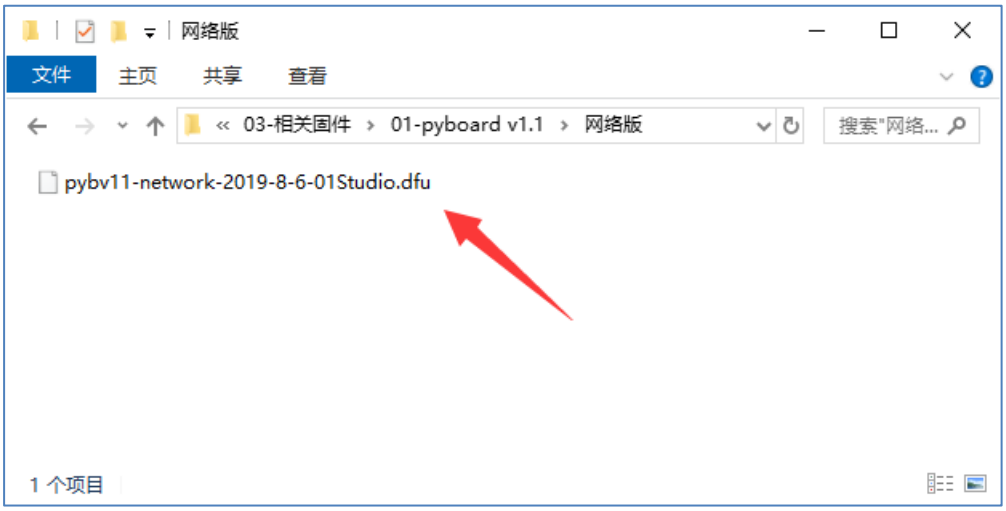


图 6-25 网络版固件

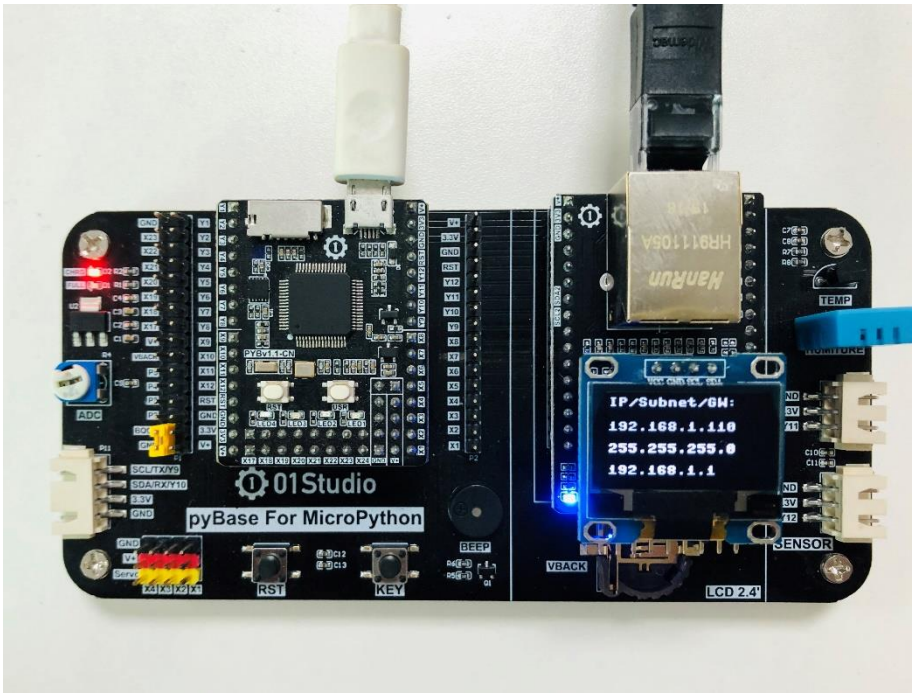

IP/Subnet/GW:
192.168.1.110
255.255.255.0
192.168.1.1
01Studio
pyBase For MicroPython
RST
KEY
BEEP
VBACK
LCD 2.4'
SENSOR
TEMP
ADC

图 6-26 以太网连接

6.4.1 连接网络

- **前言：**

万物互联是趋势，pyBaord 不具备 WiFi 功能，但其强大的库基础了 W5500 以太网模块函数代码，我们可以轻松通过 MicroPython 编程实现 pyboard 连接互联网。

- **实验平台：**

pyBoard 开发套件和 W5500 以太网模块。（可以选配 OLED）

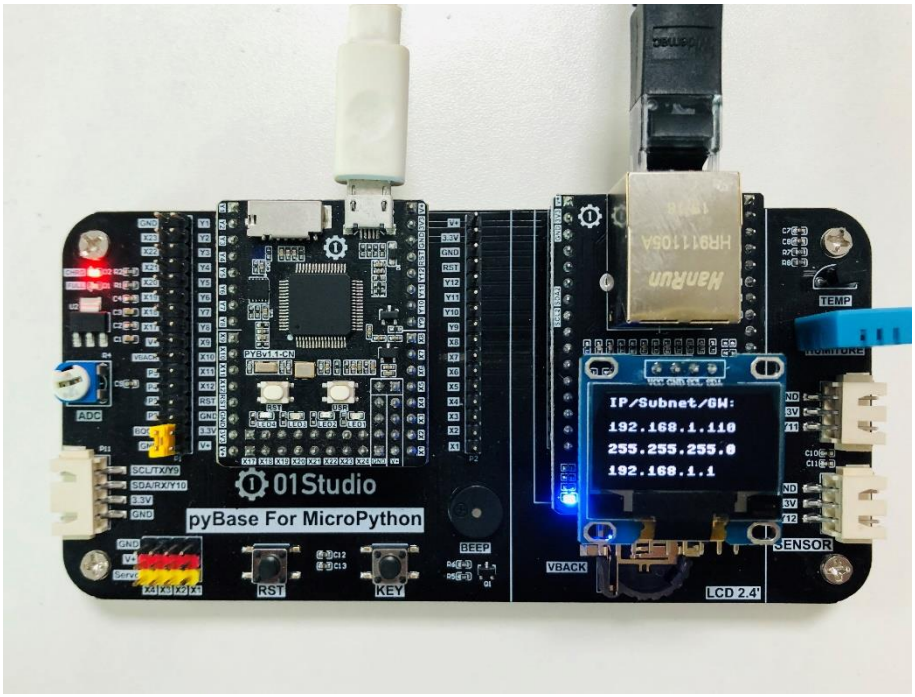


图 6-27 pyBoard 开发套件

- **实验目的：**

编程实现连接路由器，将 IP 地址等相关信息通过串口终端打印。

- **实验讲解：**

连接路由器上网是我们每天都做的事情，日常生活中我们只需要知道路由器的账号和密码，就能使用电脑或者手机连接到无线路由器，然后上网冲浪。

pybaord 带 network 固件已经集成了 network 模块，开发者使用内置的 network 模块函数可以非常方便地连接上路由器。我们先来看看 network 模块的

构造函数和使用方法。

构造函数
nic = network.WIZNET5K(spi, pin_cs, pin_rst)
构建 W5500 以太网模块连接对象。 【spi】SPI 引脚； 【pin_cs】模块片选引脚 【pin_rst】模块中断引脚
使用方法
nic. active(True)
激活网络。True：开启，False:关闭。
nic.isconnected()
检查设备是否已经连接上。返回 Ture:已连接；False：未连接。
wlan.ifconfig([ip,subnet,gateway,dns])
设备信息配置。 【手动分配 IP】ip：IP 地址；subnet:子网掩码；gateway:网关地址；dns:DNS 【自动分配 IP】'dhcp' **（如果参数为空，则返回当前连接信息。）**

图 6-28　W5500 以太网模块对象

从上表可以看到 MicroPython 通过模块封装，让联网变得非常简单。代码编写流程如下：

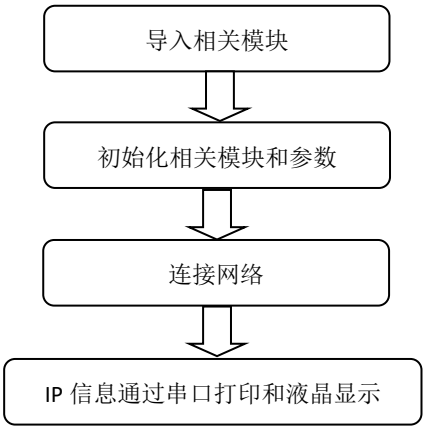


图 6-29 代码编写流程图

实验参考代码如下：

```
'''
实验名称：W5500 以太网模块连接网络
版本：v1.0
日期：2019.11
作者：01Studio
说明：通过 Socket 编程实现 pyBoard+W5500 以太网模块连接网络。
'''

import network,usocket,pyb
from machine import I2C,Pin
from ssd1306 import SSD1306_I2C

#初始化相关模块，OLED 引脚换成了 Y9、Y10
i2c = I2C(sda=Pin('Y10'), scl=Pin('Y9'))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#以太网模块初始化
nic = network.WIZNET5K(pyb.SPI(2), 'Y5', 'Y4')
```

```
nic.active(True)
nic.ifconfig('dhcp')


#判断网络是否连接成功
if nic.isconnected():


    print(nic.ifconfig()) #打印 IP 信息


    #OLED 数据显示
    oled.fill(0)   #清屏背景黑色
    oled.text('IP/Subnet/GW:',0,0)
    oled.text(nic.ifconfig()[0], 0, 20)
    oled.text(nic.ifconfig()[1],0,38)
    oled.text(nic.ifconfig()[2],0,56)
    oled.show()
```

上面代码在 main.py 文件中，我们也可以新建一个 net.py 文件，然后通过模块引用方式来供 main.py 调用，以提高程序的灵活性。

- **实验结果：**

运行程序，可以观察到连接成功后串口终端打印 IP 等信息。

```
('192.168.1.111', '255.255.255.0', '192.168.1.1', '192.168.1.1')
MicroPython v1.11-255-g400a128e1-dirty on 2019-10-29; PYBv1.1 with STM32F405RG
Type "help()" for more information.
>>>
```

图 6-30

液晶屏显示：

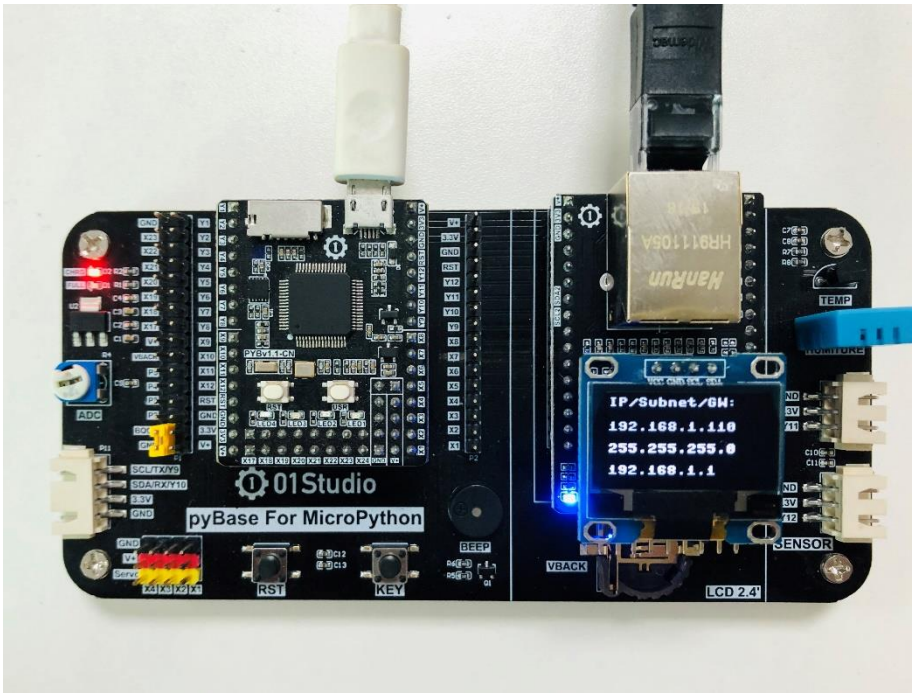


图 6-31 连接网络成功

- **总结：**

本节是联网应用的基础，成功连接到路由器的实验后，后面就可以做 socket 和 MQTT 等相关网络通信的应用了。

6.4.2 Socket 通信

- **前言：**

上一节我们学习了如何通过 MicroPython 编程实现 pyBoard+W5500 模块连接路由器。这一节我们则来学习一下 Socket 通信实验。Socket 几乎是整个互联网通信的基础。

- **实验平台：**

pyBoard 开发套件和 W5500 以太网模块。

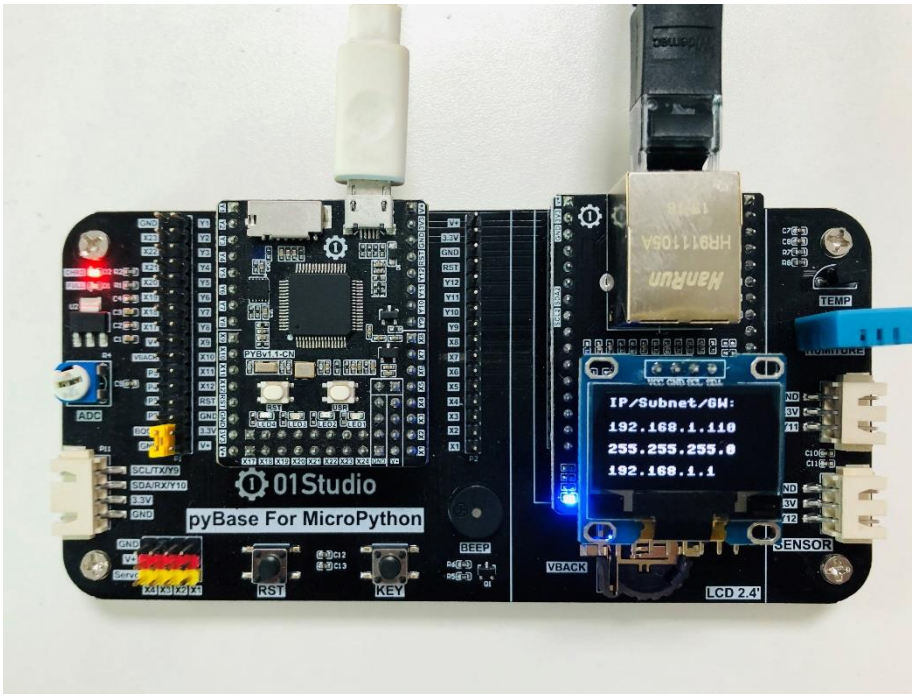


图 6-32 pyBoard 开发套件

- **实验目的：**

通过 Socket 编程实现 pyBoard 与电脑服务器助手建立连接，相互收发数据。

- **实验讲解：**

Socket 我们听得非常多了，但由于网络工程是一门系统工程，涉及的知识非常广，概念也很多，任何一个知识点都能找出一堆厚厚的的书，因此我们经常会混淆。在这里，我们尝试以最容易理解的方式来讲述 Socket，如果需要全面了解，可以自行查阅相关资料学习。

我们先来看看网络层级模型图，这是构成网络通信的基础：

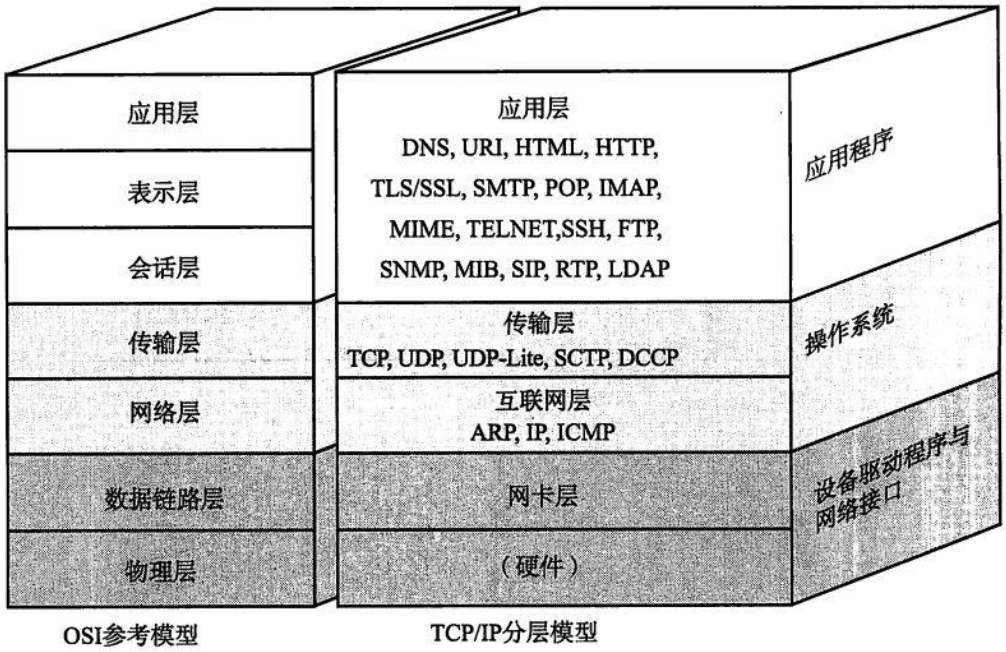


图 6-33 网络层级模型

我们看看 TCP/IP 模型的传输层和应用层，传输层比较熟悉的概念是 TCP 和 UDP，UPD 协议基本就没有对 IP 层的数据进行任何的处理了。而 TCP 协议还加入了更加复杂的传输控制，比如滑动的数据发送窗口（Slice Window），以及接收确认和重发机制，以达到数据的可靠传送。应用层中网页常用的则是 HTTP。那么我们先来解析一下这 TCP 和 HTTP 两者的关系。

我们知道网络通信是最基础是依赖于 IP 和端口的，HTTP 一般情况下默认使用端口 80。举个简单的例子：我们逛淘宝，浏览器会向淘宝网的网址（本质是 IP）和端口发起请求，而淘宝网收到请求后响应，向我们手机返回相关网页数据信息，实现了网页交互的过程。而这里就会引出一个多人连接的问题，很多人访问淘宝网，实际上接收到网页信息后就断开连接，否则淘宝网的服务器是无法支撑这么多人长时间的连接的，哪怕能支持，也非常占资源。

也就是应用层的 HTTP 通过传输层进行数据通信时，TCP 会遇到同时为多个应用程序进程提供并发服务的问题。多个 TCP 连接或多个应用程序进程可能需要通过同一个 TCP 协议端口传输数据。为了区别不同的应用程序进程和连接，许多计算机操作系统为应用程序与 TCP / IP 协议交互提供了套接字(Socket)接口。应用层可

以和传输层通过 Socket 接口，区分来自不同应用程序进程或网络连接的通信，实现数据传输的并发服务。

简单来说，Socket 抽象层介于传输层和应用层之间，跟 TCP/IP 并没有必然的联系。Socket 编程接口在设计的时候，就希望也能适应其他的网络协议。

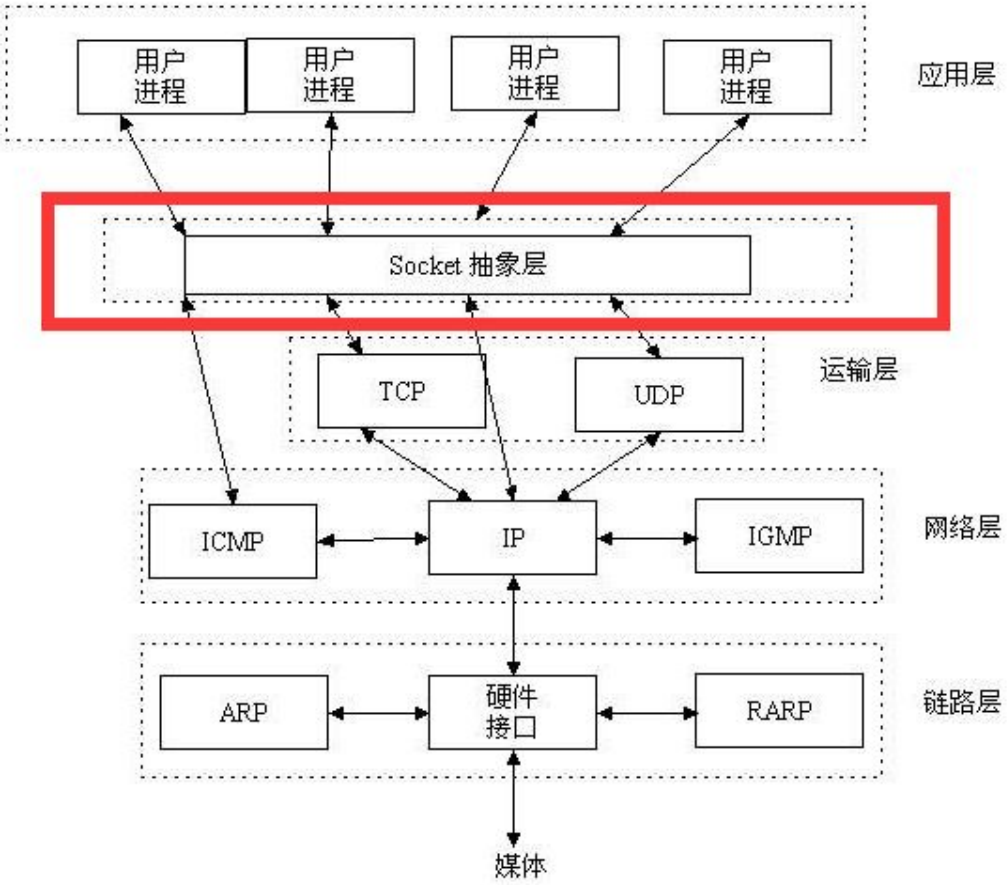


图 6-34　Socket 抽象层

套接字（socket）是通信的基石，是支持 TCP/IP 协议的网络通信的基本操作单元。它是网络通信过程中端点的抽象表示，包含进行网络通信必须的五种信息：**连接使用的协议（通常是 TCP 或 UDP），本地主机的 IP 地址，本地进程的协议端口，远地主机的 IP 地址，远地进程的协议端口。**

所以，socket 的出现只是可以更方便的使用 TCP/IP 协议栈而已，简单理解就是其对 TCP/IP 进行了抽象，形成了几个最基本的函数接口。比如 create，listen，accept，connect，read 和 write 等等。以下是通讯流程：

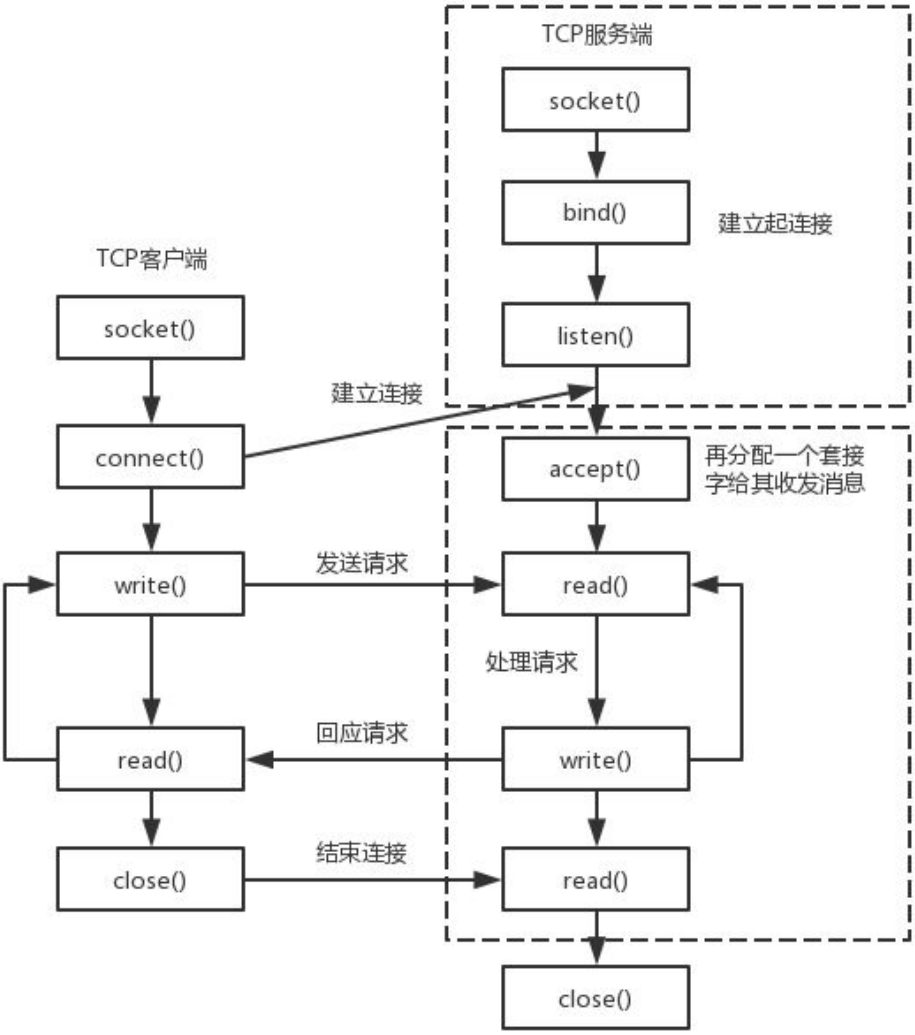


图 6-35　Socket 通信过程

从上图可以看到，建了 Socket 通信需要一个服务器端和一个客户端，以本实验为例，pyBoard 作为客户端，电脑使用网络调试助手作为服务器端，双方使用 TCP 协议传输。对于客户端，则需要知道电脑端的 IP 和端口即可建立连接。（端口可以自定义，范围在 0~65535，注意不占用常用的 80 等端口即可。）

以上的内容，简单来说就是如果用户面向应用来说，那么 OpenMV 只需要知道**通讯协议是 TCP 或 UDP、服务器的 IP 和端口号**这 3 个信息，即可向服务器发起连接和发送信息。就这么简单。

MicroPython 已经封装好相关模块 usocket,跟传统的 socket 大部分兼容，两者均可使用，本实验使用 usocket，对象如下介绍：

构造函数
s=usocket.socekt(af=AF_INET, type=SOCK_STREAM,proto=IPPROTO_TCP)
构建 usocket 对象。 af: AF_INET→IPV4，AF_INET6 → IPV6； type: SCOK_STREAM→TCP，SOCK_DGRAM→UDP； proto: IPPROTO_TCP→TCP 协议，IPPROTO_UDP→UDP 协议。 （如果要构建 TCP 连接，可以使用默认参数配置，即不输入任何参数。）
使用方法
addr=usocket.getaddrinfo('www.01studio.org', 80)[0][-1]
获取 Socket 通信格式地址。返回：('47.91.208.161',80)
s.connect(address)
创建连接。address:地址格式为 IP+端口。例：('192.168.1.115',10000)
s.send(bytes)
发送。bytes：发送内容格式为字节
s.recv(bufsize)
接收数据。bufsize：单次最大接收字节个数。
s.bind(address)
绑定，用于服务器角色
s.listen([backlog])
监听，用于服务器角色。backlog:允许连接个数，必须大于 0。
s.accept()
接受连接，用于服务器角色。
*其它更多用法请阅读 MicroPython 文档：(搜索:usocket) http://docs.micropython.01studio.org/zh_CN/latest/library/usocket.html

表 6-8　Socket 对象

本实验中 pyBoard 属于客户端，因此只用到客户端的函数即可。实验代码编

写流程如下：

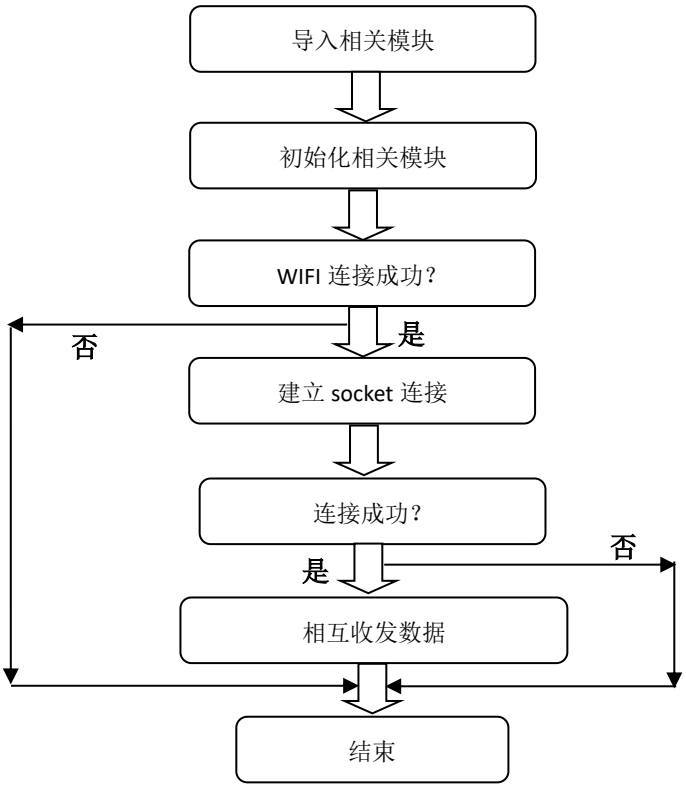


图 6-36 代码编写流程图

实验参考代码：

```
'''
实验名称：W5500 以太网模块 Socket 通信
版本：v1.0
日期：2019.11
作者：01Studio
说明：通过 Socket 编程实现 pyBoard+W5500 以太网模块与电脑服务器助手建立 TCP 连接，相互收发数据。
'''

import network,usocket,pyb
from machine import I2C,Pin
from ssd1306 import SSD1306_I2C
```

```
#初始化相关模块
i2c = I2C(sda=Pin('Y10'), scl=Pin('Y9'))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#socket 数据接收中断标志位
socket_node = 0

#初始化以太网模块
nic = network.WIZNET5K(pyb.SPI(2), pyb.Pin.board.Y5, pyb.Pin.board.Y4)
nic.active(True)
nic.ifconfig('dhcp')

#判断网络是否连接成功
if nic.isconnected():

    print(nic.ifconfig()) #打印 IP 信息
    #OLED 数据显示
    oled.fill(0)   #清屏背景黑色
    oled.text('IP/Subnet/GW:',0,0)
    oled.text(nic.ifconfig()[0], 0, 20)
    oled.text(nic.ifconfig()[1],0,38)
    oled.text(nic.ifconfig()[2],0,56)
    oled.show()

    #创建 socket 连接 TCP 类似，连接成功后发送“Hello 01Studio!”给服务器。
    s=usocket.socket()
    addr=('192.168.1.116',10000) #服务器 IP 和端口
    s.connect(addr)
    s.send('Hello 01Studio!')
```

```
#开启定时器，周期 100ms，执行 socket 通信接收任务
def fun(tim):
    global socket_node
    socket_node = 1
    pyb.LED(3).toggle()

tim = pyb.Timer(1,freq=10)
tim.callback(fun)

while True:
    if socket_node:
        text=s.recv(128) #单次最多接收 128 字节
        if text == '':
            pass

        else: #打印接收到的信息为字节，可以通过 decode('utf-8')转成字符串
            print(text)
            s.send('I got:'+text.decode('utf-8'))

        socket_node=0
```

网络连接代码在上一节已经讲解，这里不再重复，程序在连接成功后建了 Socket 连接，连接成功发送‘Hello 01Studio!’信息到服务器。另外 OpenMV 定时器设定了了每 100ms 处理从服务器接收到的数据。将接收到数据通过串口打印和重新发送给服务器。

- **实验结果：**

先在电脑端打开网络调试助手并建立服务器，软件路径在： **零一科技（01Studio）MicroPython 开发套件配套资料\01-开发工具\01-Windows\网络调试助手** 下的 NetAssist.exe ，直接双击打开即可！

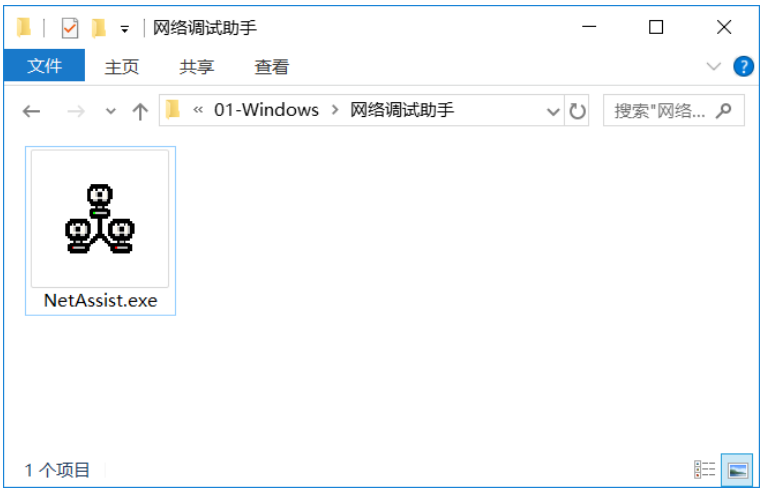


图 6-37 网络调试助手

以下是新建服务器的方法，打开网络调试助手后在左上角协议类型选择 TCP Server；中间的本地 IP 地址是自动识别的，不要修改，这个就是服务器的 IP 地址。然后端口写 10000（0-65535 都可以。），点击连接，成功后红点亮。如下图：



图 6-38 TCP 服务器配置

在时候服务器已经在监听状态！用户需要根据自己的实际情况自己的服务器 IP 地址+端口。即修改上面的代码以下部分内容。（服务器 IP 和端口可以在网络调试助手找到。）

```
addr=('192.168.1.115',10000) #服务器 IP 和端口
```

下载程序，开发板成功连接网络后，发起了 socket 连接，连接成功可以可以看到网络调试助手收到了开发板发来的信息。在下方列表多了一个连接对象，点击选中：

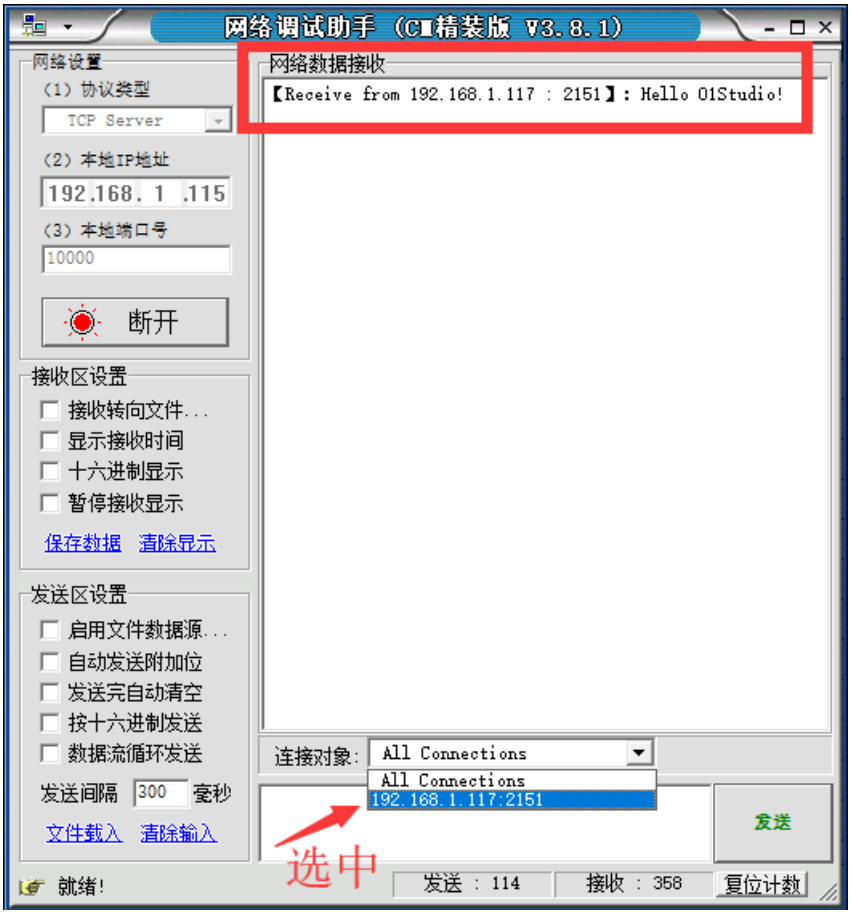


图 6-39 选中连接对象

选中后我们在发送框输入信息“Hi”，点击发送，可以看到开发板的 REPL 打印出来信息 Hi。为字节数据。另外由于程序将收到的信息发回给服务器，所以在网络调试助手中也接收到开发板返回的信息：I got:Hi。

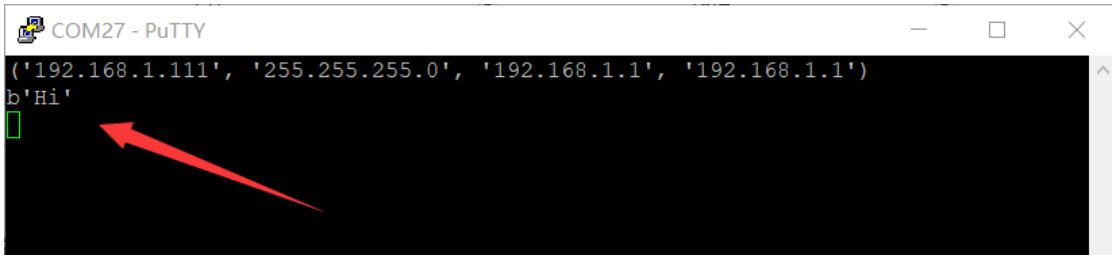


图 6-40

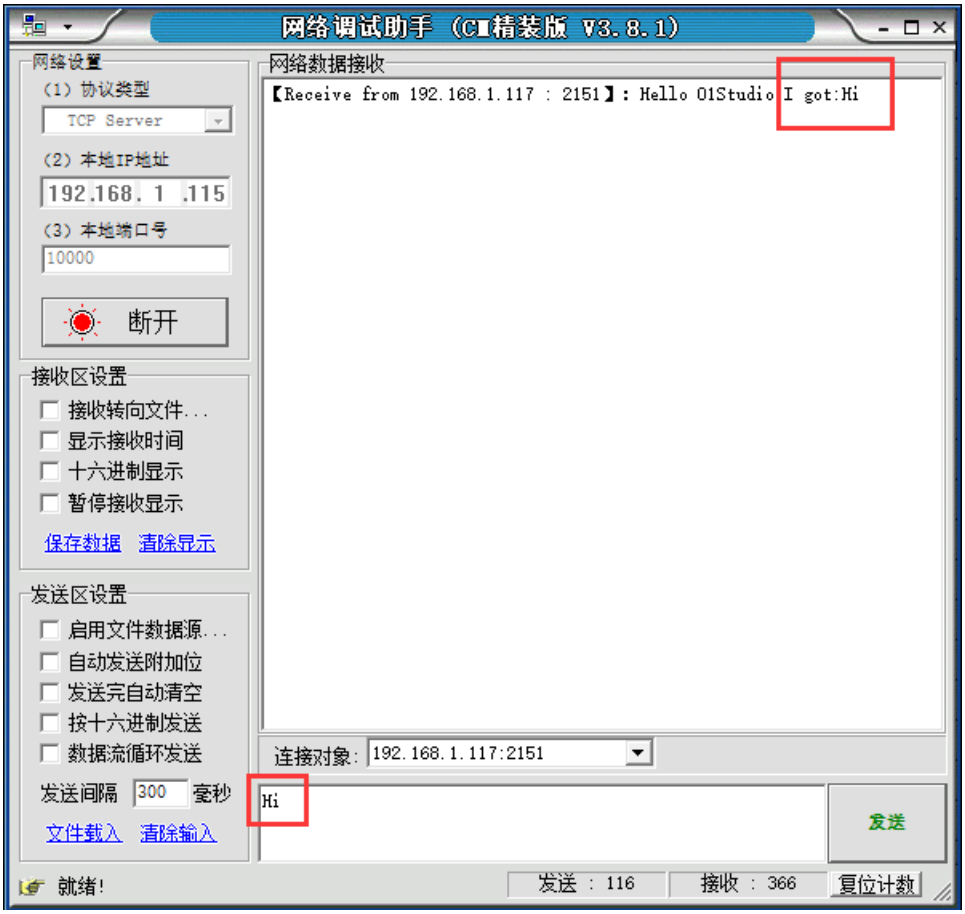


图 6-41 服务器发送数据

- **总结：**

通过本节学习，我们了解了 socket 通信原理以及使用 MicroPython 进行 socket 编程并且通信的实验。得益于优秀的封装，让我们可以直接面向 socket 对象编程就可以快速实现 socket 通信，从而开发更多的网络应用，例如将前面采集到的传感器数据发送到服务器。

6.4.3 MQTT 通信

- **前言：**

上一节，我们学习了 Socket 通信，当服务器和客户端建立起连接时，就可以相互通信了。在互联网应用大多使用 WebSocket 接口来传输数据。而在物联网应用中，常常出现这样的情况：海量的传感器，需要时刻保持在线，传输数据量非常低，有着大量用户使用。如果仍然使用 socket 作为通信，那么服务器的压力和通讯框架的设计随着数量的上升将变得异常复杂！

那么有无一个框架协议来解决这个问题呢，答案是有的。那就是 MQTT(消息队列遥测传输)。

- **实验平台：**

pyBoard 开发套件和 W5500 以太网模块。

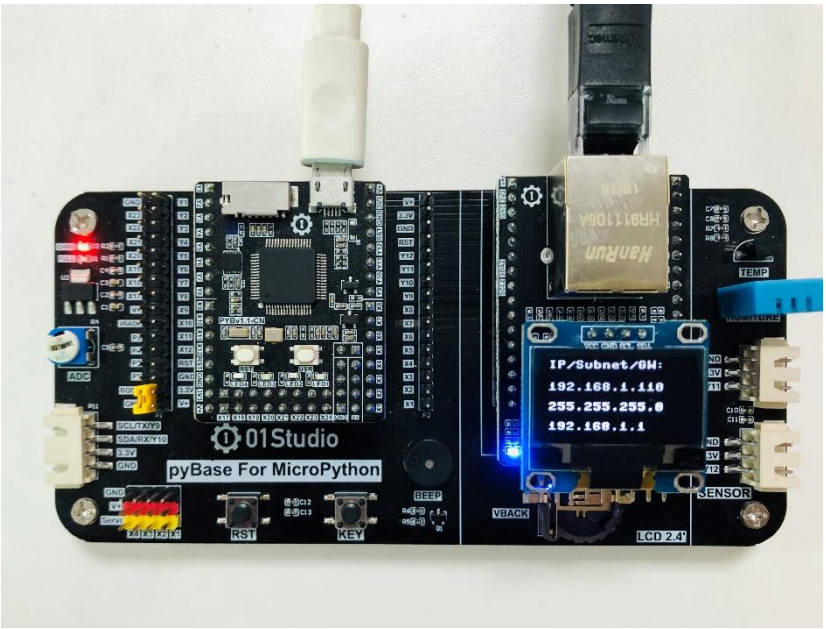


图 6-42 pyBoard 开发套件

- **实验目的：**

通过编程让 pyBoard 实现 MQTT 协议信息的发布和订阅（接收）。

- **实验讲解：**

MQTT 是 IBM 于 1999 年提出的，和 HTTP 一样属于应用层，它工作在 TCP/IP

协议族上，通常还会调用 socket 接口。是一个基于客户端-服务器的消息发布/订阅传输协议。其特点是协议是轻量、简单、开放和易于实现的，这些特点使它适用范围非常广泛。在很多情况下，包括受限的环境中，如：机器与机器（M2M）通信和物联网（IoT）。其在，通过卫星链路通信传感器、偶尔拨号的医疗设备、智能家居、及一些小型化设备中已广泛使用。

总结下来 MQTT 有如下特性/优势：

- **异步消息协议**
- **面向长连接**
- **双向数据传输**
- **协议轻量级**
- **被动数据获取**

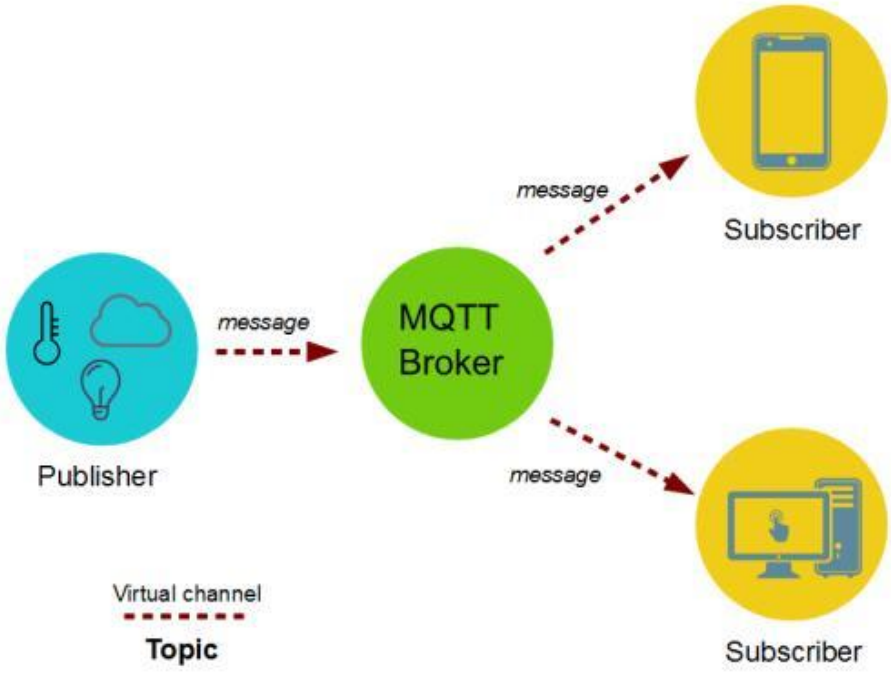


图 6-43 MQTT 通信流程

从上图可以看到，MQTT 通信的角色有两个，分别是服务器和客户端。服务器只负责中转数据，不做存储；客户端可以是信息发送者或订阅者，也可以同时是两者。具体如下图：

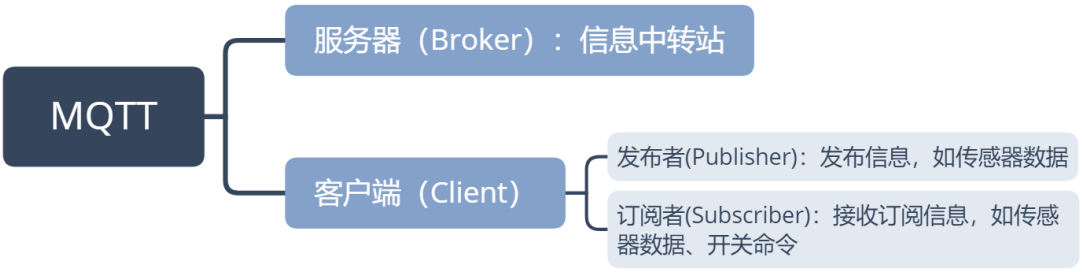


图 6-44　MQTT 角色说明

确定了角色后是如何传输数据呢？下表示 MQTT 最基本的数据帧格式，例如温度传感器发布主题“Temperature”编号,消息是“25”（表示温度）。那么所有订阅了这个主题编号的客户端（手机应用）就会收到相关信息，从而实现通信。如下表所示：

MQTT 数据帧格式	
Topic ID (主题编号)	Message (消息)
Temperature	25

表 6-9　MQTT 数据格式

由于特殊的发布/订阅机制，服务器不需要存储数据（当然也可以在服务器的设备上建立一个客户端来订阅保存信息），因此非常适合海量设备的传输。

人生苦短，而 MicroPython 已经封装好了 MQTT 客户端的库文件。让我们的应用变得简单美妙。OpenMV 的 MQTT 模块文件为例程文件夹里面的 mqtt.py 文件。使用方法如下：

构造函数
client=mqtt. MQTTClient (client_id, server, port)
构建 MQTT 客户端对象。 client_id: 客户端 ID，具有唯一性； server: 服务器地址，可以是 IP 或者网址； port:服务器端口。（默认是 1883，服务器通常采用的端口，也可以自定义。）
使用方法
`client.connect()`

连接到服务器。
client.publish(TOPIC,message)
发布。TOPIC：主题编号；message：信息内容，例：'Hello 01Studio!'
client.subscribe(TOPIC)
订阅。TOPIC：主题编号。
client.set_callback(callback)
设置回调函数。callback：订阅后如果接收到信息，就执行相名称的回调函数。
client.check_msg()
检查订阅信息。如收到信息就执行设置过的回调函数 callback。

表 6-10　MQTT 客户端对象

由于客户端分为发布者和订阅者角色，因此为了方便大家更好理解，本实验分开两个案例来编程，分别为发布者和订阅者。再结合 MQTT 网络调试助手来测试。代表编写流程图如下：

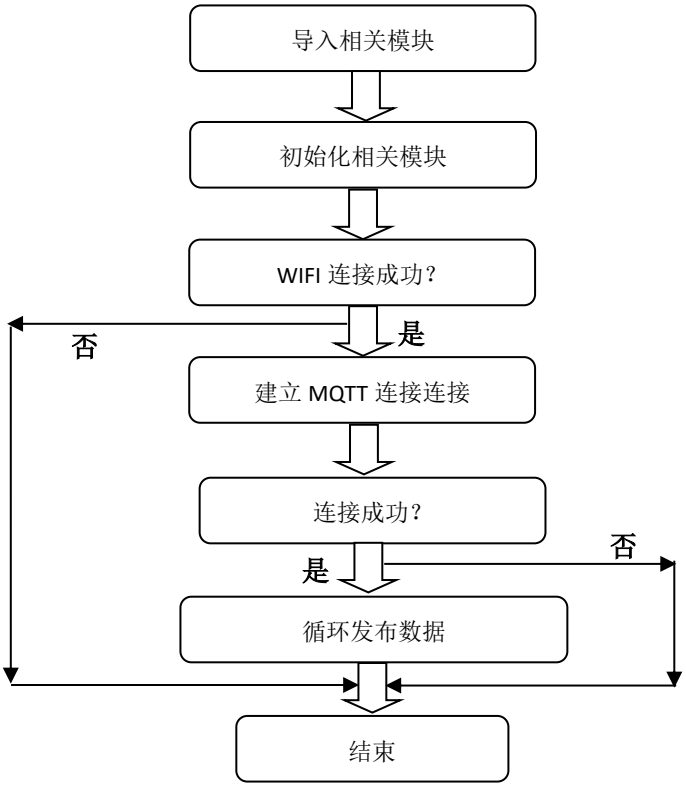


图 6-45 发布者代码编写流程

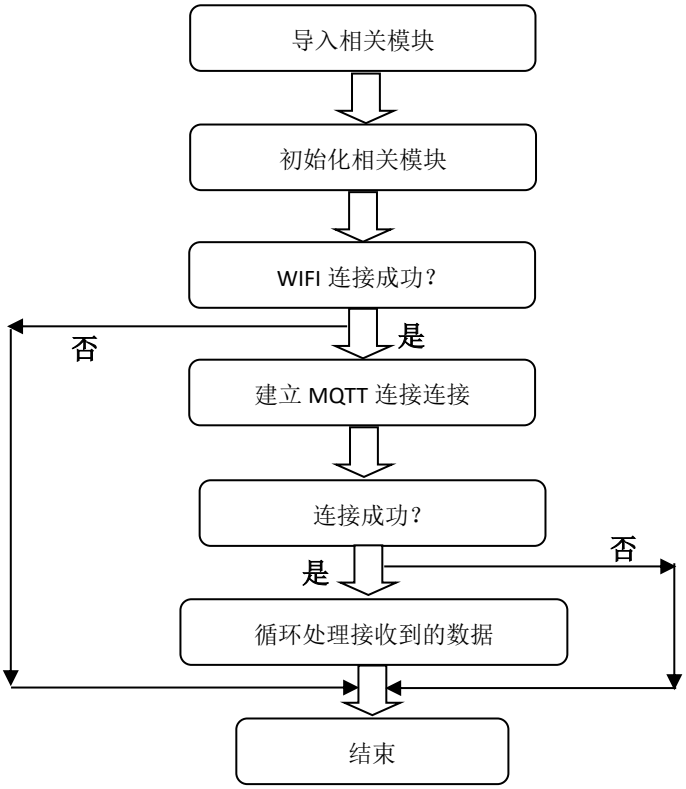


图 6-46 订阅者代码编写流程

发布者（publish）参考代码：

```
'''
实验名称：W5500 以太网模块 MQTT 通信
版本：v1.0
日期：2019.11
作者：01Studio
说明：通过 Socket 编程实现 pyBoard+W5500 以太网模 MQTT 通信　发布者（publish）。
'''

import network,pyb,time
from machine import I2C,Pin
from ssd1306 import SSD1306_I2C
from simple import MQTTClient
```

```
#初始化 OLED
i2c = I2C(sda=Pin('Y10'), scl=Pin('Y9'))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化以太网模块
nic = network.WIZNET5K(pyb.SPI(2), pyb.Pin.board.Y5, pyb.Pin.board.Y4)
nic.active(True)
nic.ifconfig('dhcp')

#判断网络是否连接成功
if nic.isconnected():

    print(nic.ifconfig()) #打印 IP 信息

    #OLED 数据显示
    oled.fill(0)   #清屏背景黑色
    oled.text('IP/Subnet/GW:',0,0)
    oled.text(nic.ifconfig()[0], 0, 20)
    oled.text(nic.ifconfig()[1],0,38)
    oled.text(nic.ifconfig()[2],0,56)
    oled.show()

    SERVER = 'mqtt.p2hp.com'
    PORT = 1883
    CLIENT_ID = '01Studio-pyBoard' # 客户端 ID
    TOPIC = '/public/01Studio/1' # TOPIC 名称

    client = MQTTClient(CLIENT_ID, SERVER, PORT)
    client.connect()
```

```
    while (True):
        client.publish(TOPIC, "Hello 01Studio!") #发布消息
        time.sleep_ms(1000) #延时 1 秒
```

订阅者（subscribe）参考代码：

```
'''
实验名称：W5500 以太网模块 MQTT 通信
版本：v1.0
日期：2019.11
作者：01Studio
说明：通过 Socket 编程实现 pyBoard+W5500 以太网模 MQTT 通信  订阅者
（subscribe）。
'''

import network,pyb,time
from machine import I2C,Pin
from ssd1306 import SSD1306_I2C
from simple import MQTTClient

#初始化 OLED
i2c = I2C(sda=Pin('Y10'), scl=Pin('Y9'))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化以太网模块
nic = network.WIZNET5K(pyb.SPI(2), pyb.Pin.board.Y5, pyb.Pin.board.Y4)
nic.active(True)
nic.ifconfig('dhcp')
```

```
#设置 MQTT 回调函数,有信息时候执行
def MQTT_callback(topic, msg):
    print('topic: {}'.format(topic))
    print('msg: {}'.format(msg))



#判断网络是否连接成功
if nic.isconnected():

    print(nic.ifconfig()) #打印 IP 信息

    #OLED 数据显示
    oled.fill(0)   #清屏背景黑色
    oled.text('IP/Subnet/GW:',0,0)
    oled.text(nic.ifconfig()[0], 0, 20)
    oled.text(nic.ifconfig()[1],0,38)
    oled.text(nic.ifconfig()[2],0,56)
    oled.show()

    SERVER = 'mqtt.p2hp.com'
    PORT = 1883
    CLIENT_ID = '01Studio-pyBoard' # 客户端 ID
    TOPIC = '/public/01Studio/1' # TOPIC 名称

    client = MQTTClient(CLIENT_ID, SERVER, PORT)
    client.set_callback(MQTT_callback)  #配置回调函数
    client.connect()
    client.subscribe(TOPIC) #订阅主题

    while (True):
```

```
        client.check_msg() #检测是否收到信息，收到则执行回调函数打印。
        time.sleep_ms(300) #接收间隔
```

从以上代码可以看到发布者和订阅者的编程方式相近，另外本实验需要一个 MQTT 服务器（Broker），这里使用的是跟我们将用到的 MQTT 在线网络助手同一个服务器和端口。

```
    SERVER = 'mqtt.p2hp.com'
    PORT = 1883
```

- **实验结果：**

我们将 01Studio 示例程序中的 simple.py 文件拷贝到文件系统，复位开发板。

为了方便测试，我们可以使用 MQTT 网络助手进行调试。这里推荐一个在线 MQTT 网络调试助手：http://mqtt.p2hp.com/websocket/

打开上面网址，即可看到 MQTT 在线调试助手。可以配置基本信息，这里默认即可，点击连接。

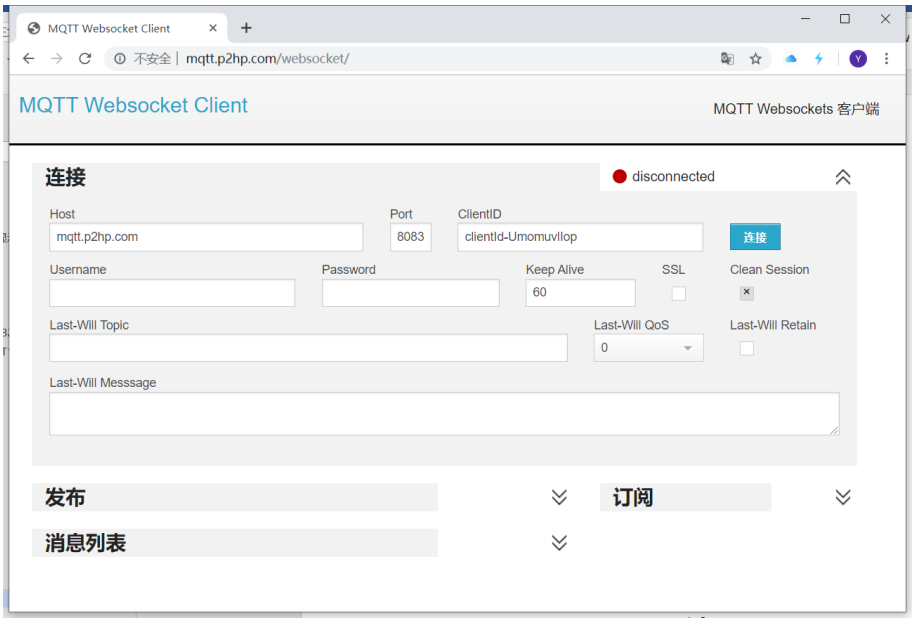


图 6-47 MQTT 助手

连接成功可以看到显示 Connected。

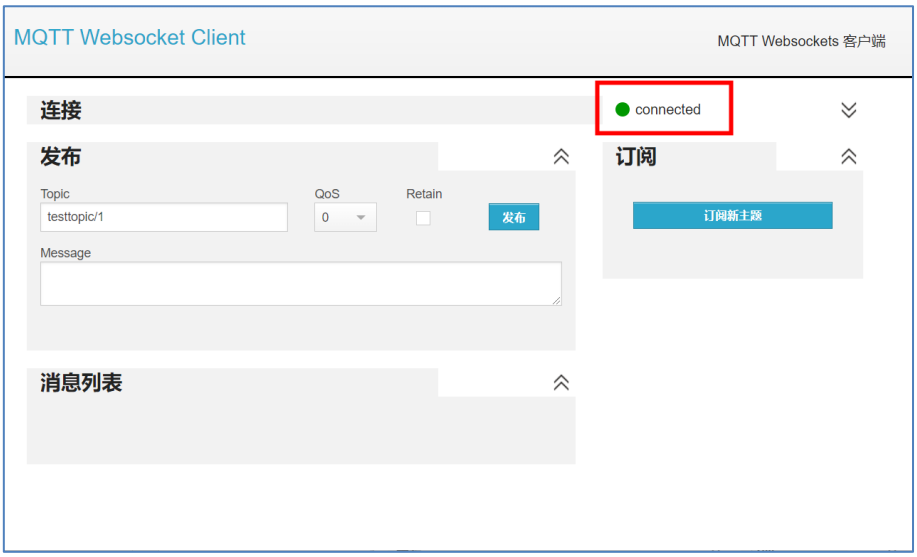


图 6-48 MQTT 助手连接成功

测试“发布者”代码：

我们先测试“发布者”代码。将“发布者”代码下载到开发板，然后在 MQTT 助手中订阅主题修改为：'/public/01Studio/1'（跟代码发布的主题一致），QOS 选择 0 即可。然后点击订阅主题。

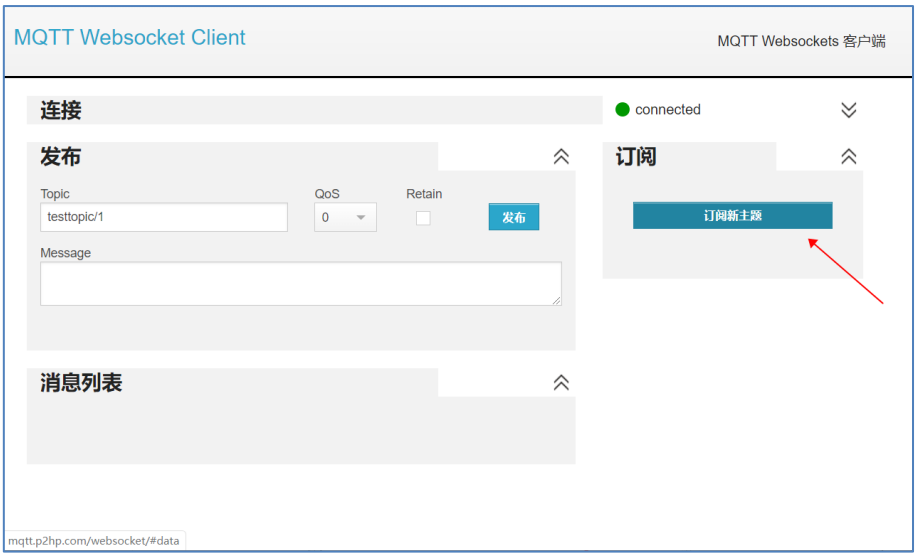


图 6-49 订阅主题

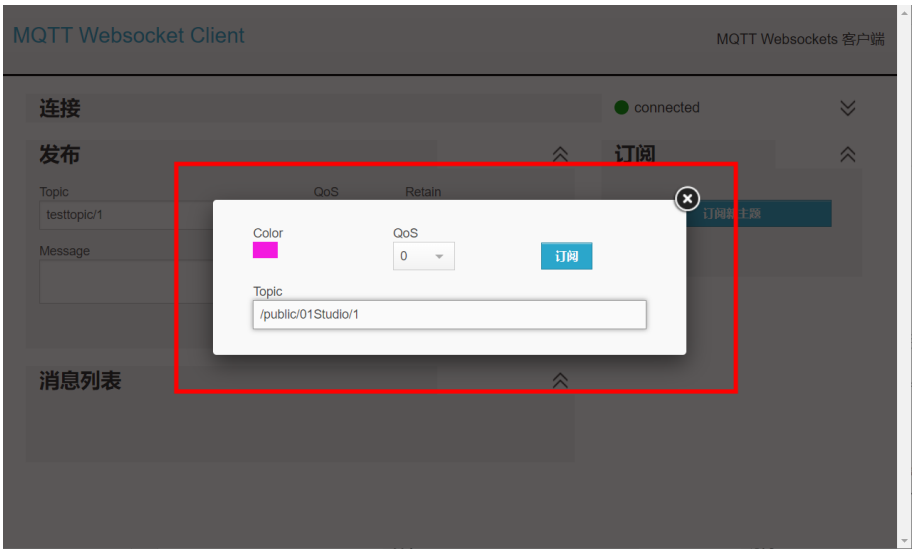


图 6-50

订阅后可以看到接收框收到来自开发板发布的信息。

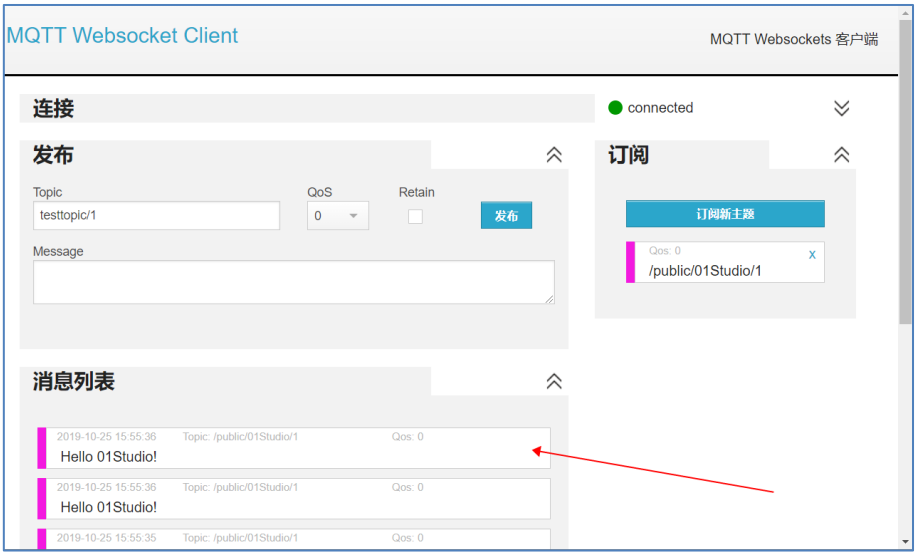


图 6-51 MQTT 助手收到开发板发布的信息

测试“订阅者”代码：

“订阅者”代码测试方法跟“发布者”相反。将“订阅者”代码下载到开发板，然后在电脑 MQTT 助手中发布主题修改为：'/public/01Studio/1'（跟代码发布的主题一致。）在下方空白框输入“Hello 01Studio!”。

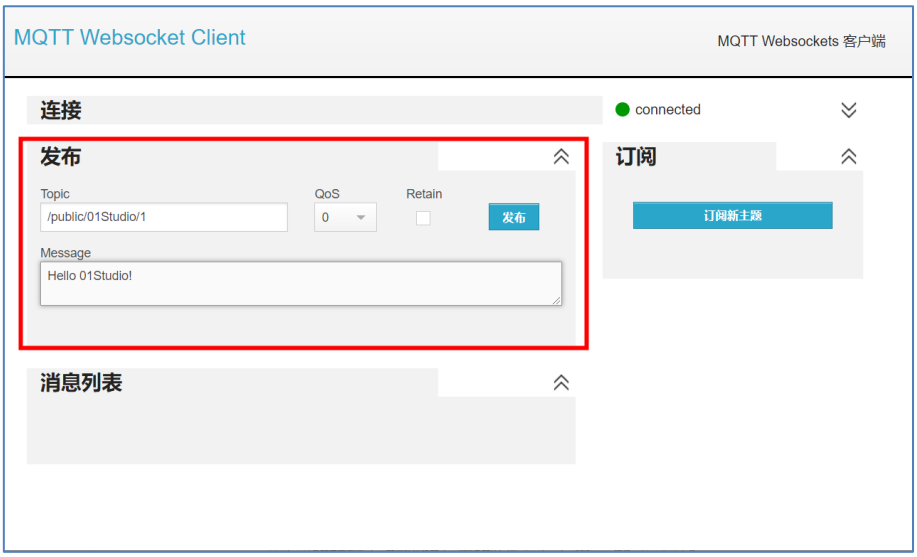


图 6-52 输入即将发布的主题和信息

将“订阅者”代码下载到开发板，成功连接后回到 MQTT 助手点击【发布】发布信息，可以在开发板的 REPL 看到接收到的订阅信息“Hello 01Studio!”被打印出来了（数据格式为字节数据）。

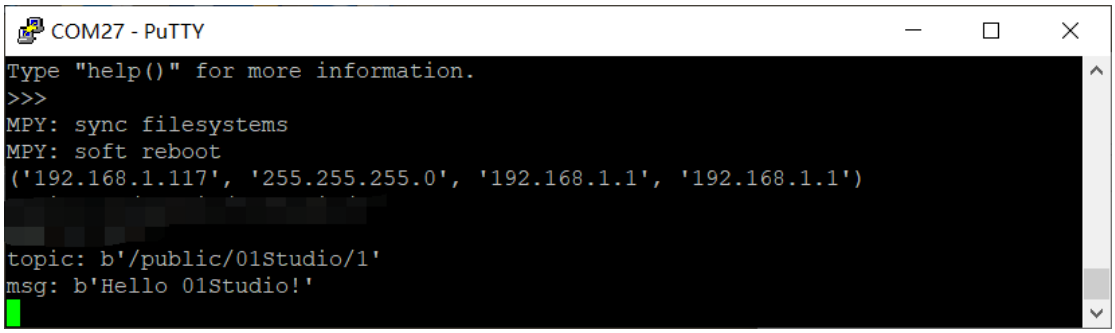


图 6-53 开发板接收到相关信息并打印

当然你也可以在同一个 MQTT 在线助手下测试发布和订阅功能，只需要将主题设置一致即可，如下图所示：

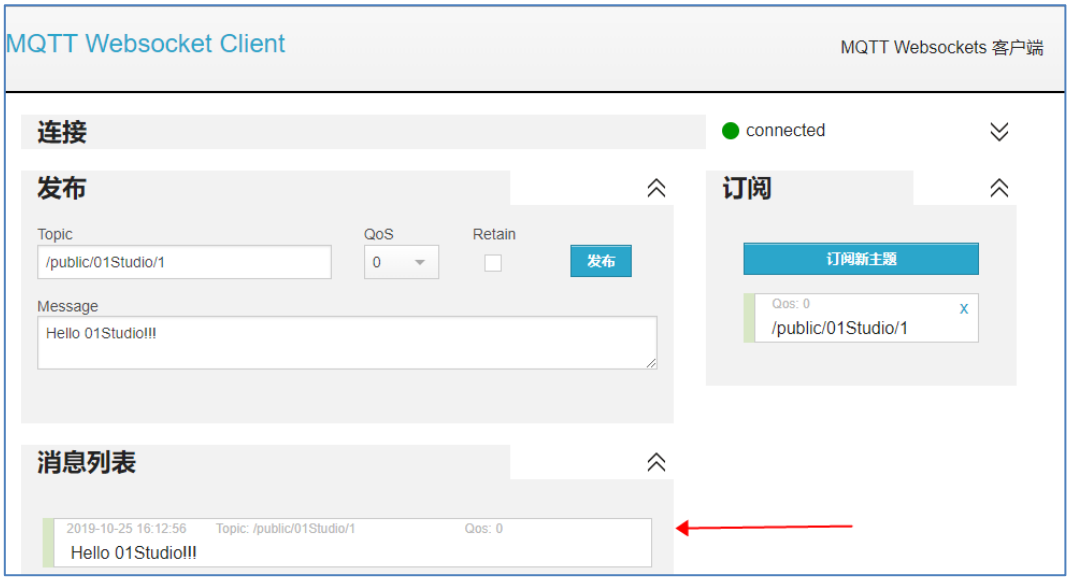


图 6-54 客户端同时发布和订阅信息

- **总结：**

通过本节我们了解了 MQTT 通信原理以及成功实现通信。目前市面上大部分物联网云平台支持 MQTT，原理大同小异。大家可以基于不同平台协议来开发，实现自己的物联网设备远程连接。

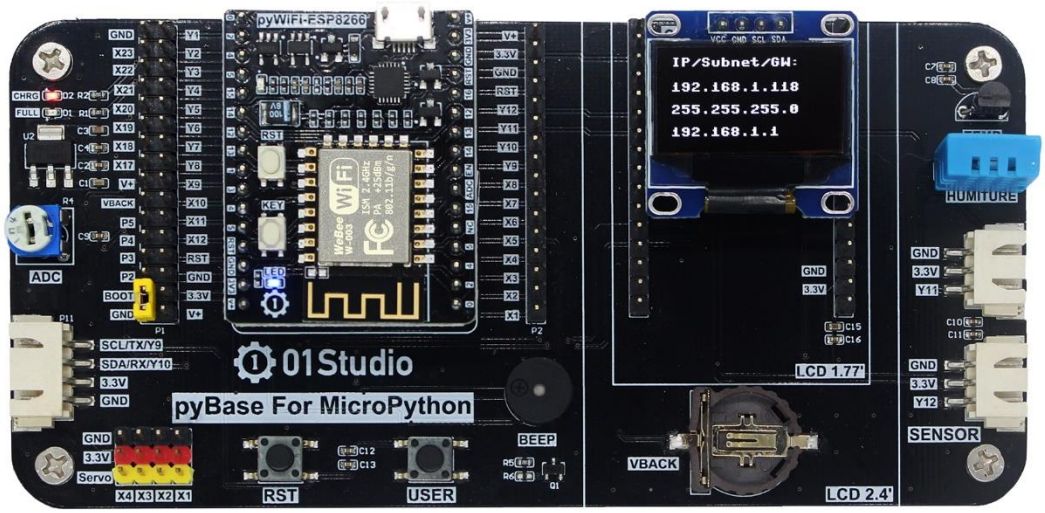
第三部分　基于 ESP8266 平台

ESP8266 相信大部分朋友都听过了，这是上海乐鑫公司推出的一款集成度非常高的 WiFi 无线 SOC，该芯片的特点是集成度非常高，只需要很少的周边外围元器件就能工作，因此市面上的 ESP8266 WIFI 模块也是非常成熟，用户拿到手即可开发。另外极高的性价比为 WIFI 嵌入式系统提供无限可能性。

传统的 ESP8266 开发方式是基于嵌入式 C 语言体系，这为一部分想快速应用物联网的用户增加了门槛，而 MicroPython 对 ESP8266 的兼容性已经非常全面了，我们使用 MicroPython 进行开发，可以快速实现基础功能和联网，非常高效！

如果你已经学习了前面基于 STM32 平台部分内容，那么恭喜你，你需要的只是搭建一下开发环境就可以快速开发了。如果没有学习过前面内容，也没关系，本部分也是会从 0 到 1，带你一步步学会 MicroPython 基于 ESP8266 的编程应用。

本部分内容包含开发环境快速建立、基础实验和 WIFI 项目，由于 ESP 主要是实现物联网 WIFI 无线连接，因此我们会重点讲解 WIFI 联网实验内容。



pyWiFi-ESP8266 开发套件

第7章　开发环境快速建立

7.1　基于 Windows

7.1.1　安装 python3

说明：我们提供安装 python3 教程只是为了在 PC 上体验一下 python3 安装和使用过程，由于 micropython 固件集成了 python 库，所以电脑并非一定要安装，本节可以跳过。

请阅读 第二部分　　3.1.1　　安装 python3 章节内容，这里不再重复。

7.1.2　安装开发软件 Mu

开发软件是指我们用来写代码的工具，上一节那个命令框里面其实就是调用了自带的编程器，但是我想没有人希望一直在那黑框里面撸代码。 Python 拥有众多的编程器，如果你之前已经熟练掌握 python 或已经使用 python 开发，那么可以直接使用你原来习惯的开发软件来编程。如果你是初学者或者喜欢简单而快速应用，我们推荐使用 Mu。

Mu 是一款开源软件，以极简方式设计，对 MicroPython 的兼容性非常友善。而且支持 Windows、Mac OS、Linux、树莓派。由于开源，所以软件迭代速度非常快，功能日趋成熟。具体安装方法如下：

在 https://codewith.mu/en/download 下载地址，选择自己的开发平台进行下载安装即可：

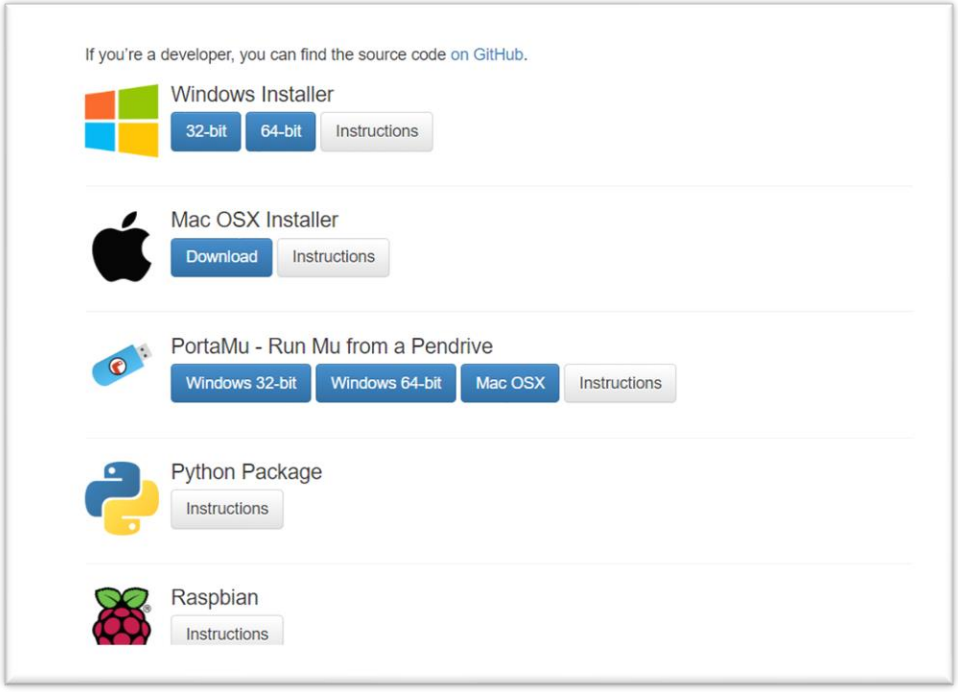


图 7-1 稳定版下载

由于 Mu 更新非常快，因此社区也同时提供最新版下载，给愿意尝鲜和给予软件反馈的用户，下载位置如下：

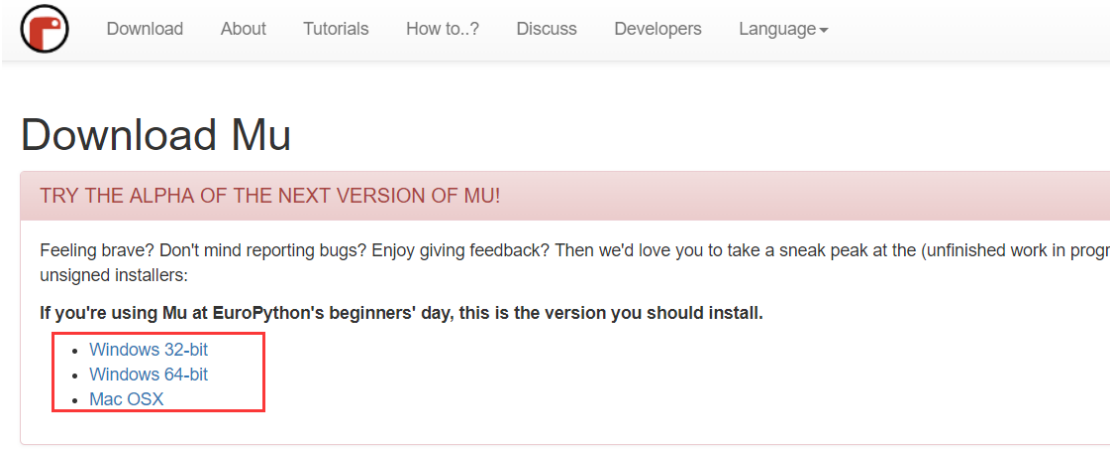


图 7-2 尝鲜版下载

我们推荐使用尝鲜版，能使用最新的功能。下载安装后，打开 Mu，可以看你的界面非常简洁：

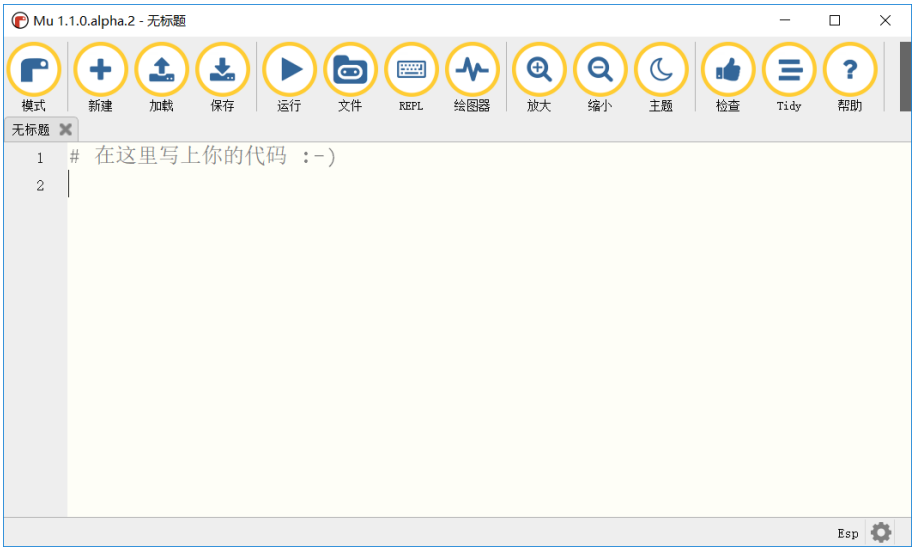


图 7-3　Mu 主界面

点击左上角“模式”，弹出对话框，可以看到支持 MicroBit、ESP 等一系列 MicroPython 产品。通常情况下在 USB 线插入能自动识别设备硬件类型。基于 pyWiFi 系列产品选择【ESP MicroPython】进行开发。

至此，Mu 安装完成。

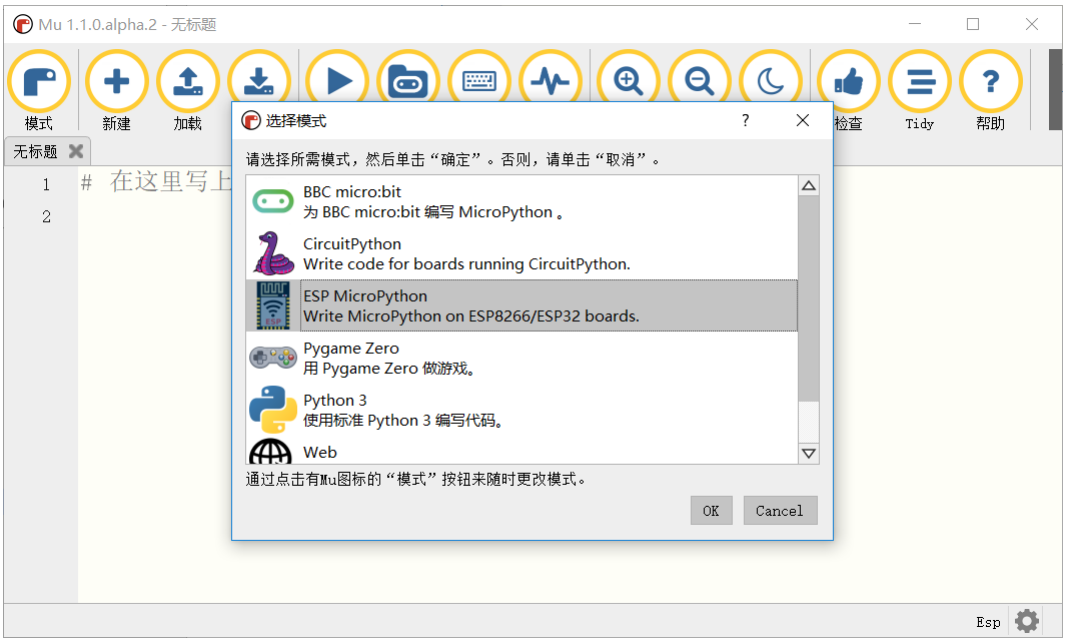


图 7-4

7.1.3 开发套件使用

7.1.3.1 驱动安装

主要是安装 USB 转串口驱动。我们将 pyWiFi-ESP8266 开发板通过 MicroUSB 数据线连接到电脑：

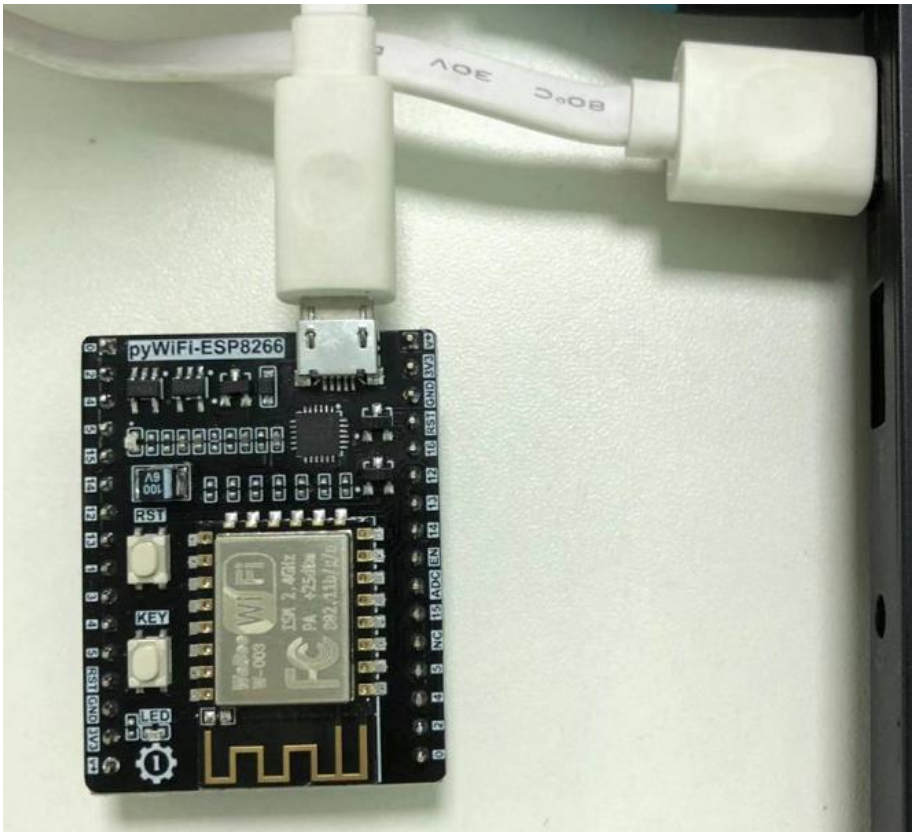


图 7-5 通过 MicroUSB 线连接到电脑

如果你的操作系统是 Win10,一般情况下能自动安装。鼠标右键点击“我的电脑”—属性—设备管理器： 出现串口号说明安装成功，如下图所示。

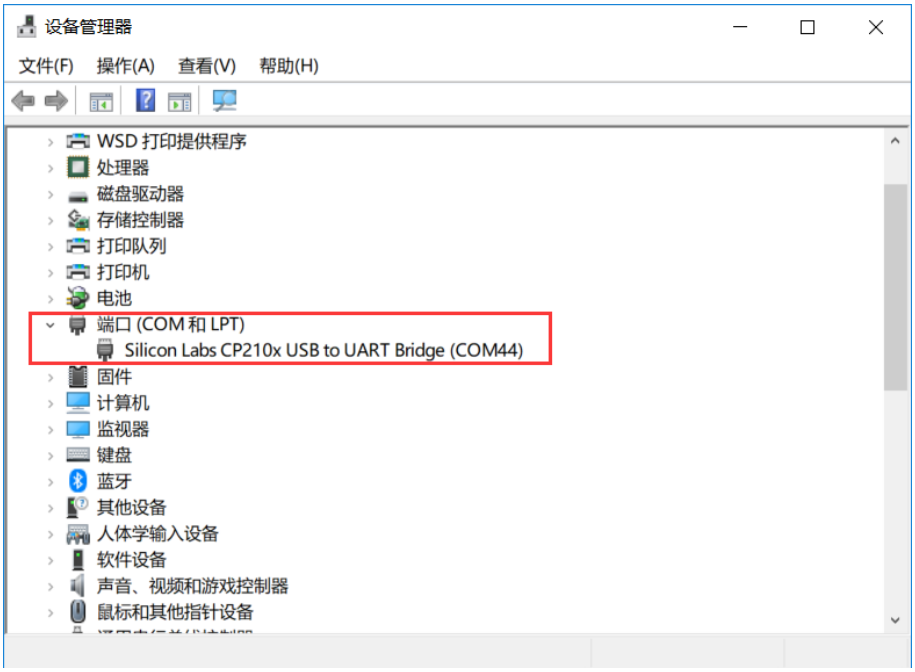


图 7-6 串口驱动成功安装

如果无法安装，请手动安装驱动，方法如下：

不能自动安装时候，设备会出现黄色叹号，这时候点击设备右键，选择“更新驱动程序”，选择“浏览计算机查找驱动”：

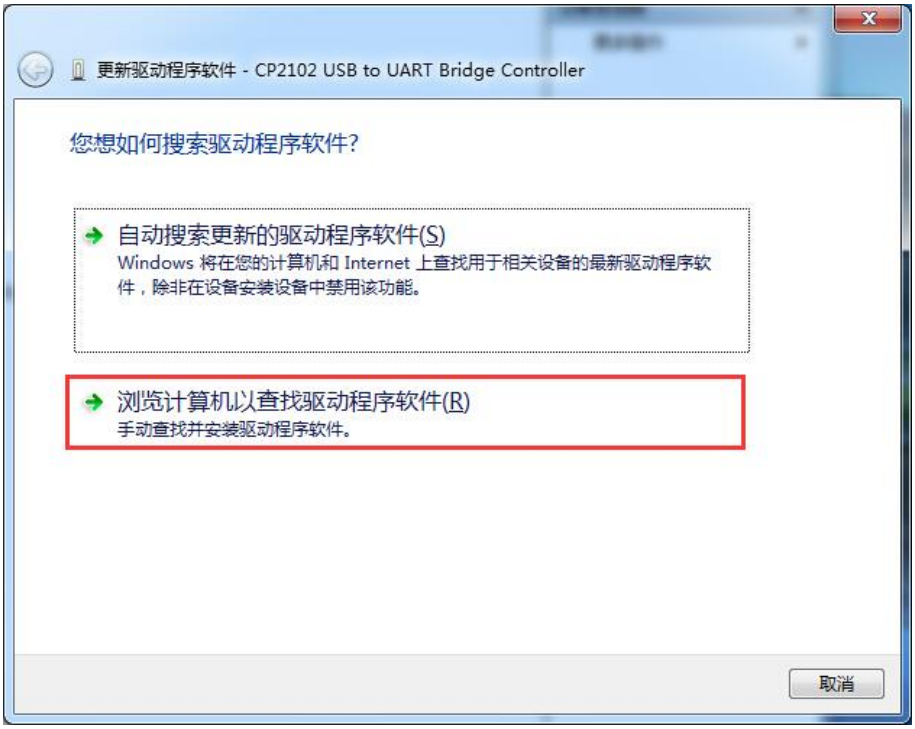


图 7-7

驱动路径选择：零一科技（01Studio）MicroPython 开发套件配套资料\01-开发工具\01-Windows\串口终端工具\CP210x 驱动，点击确认后即可自动安装：

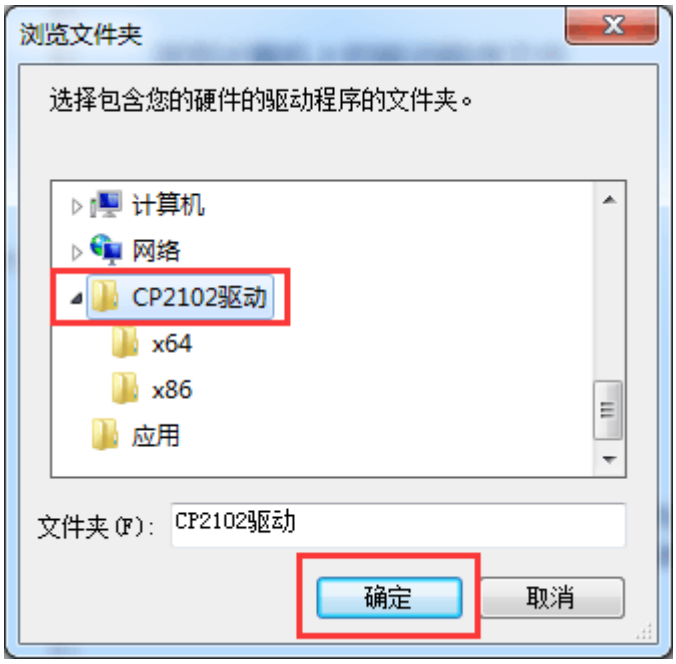


图 7-8

安装成功后如下图：

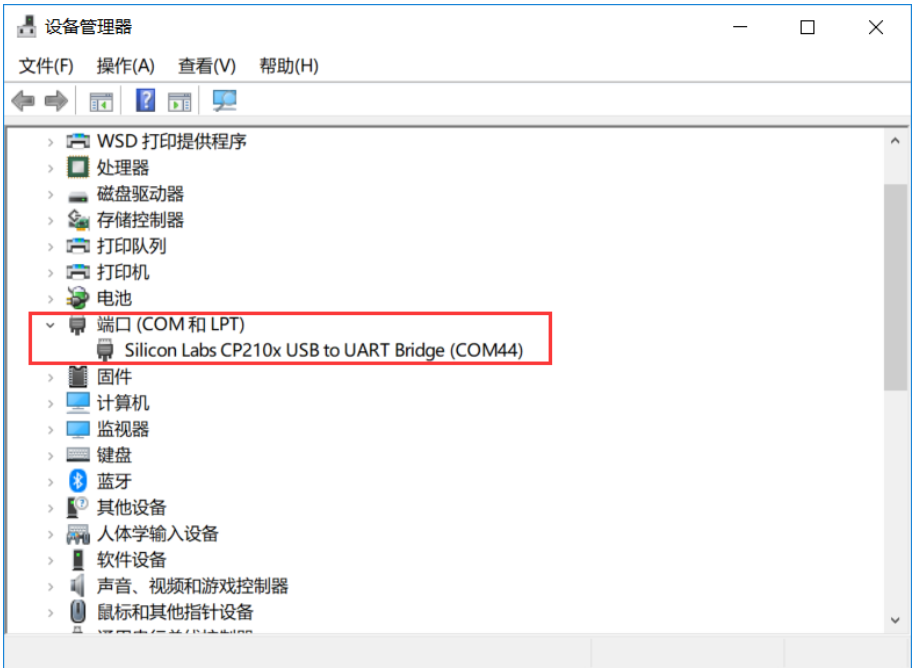


图 7-9 串口驱动安装成功

7.1.3.2 REPL 串口交互调试

请阅读 第二部分 3.1.3.3 章节 REPL 串口交互调试内容，这里不再重复。

需要补充的是 ESP826 在 Mu 软件下可以直接进入 REPL 串口调试，方法如下：

打开 Mu 软件，连接 pyWiFi-ESP8266 开发板，点击上方 REPL 按钮，软件下方即出现 REPL 调试界面：

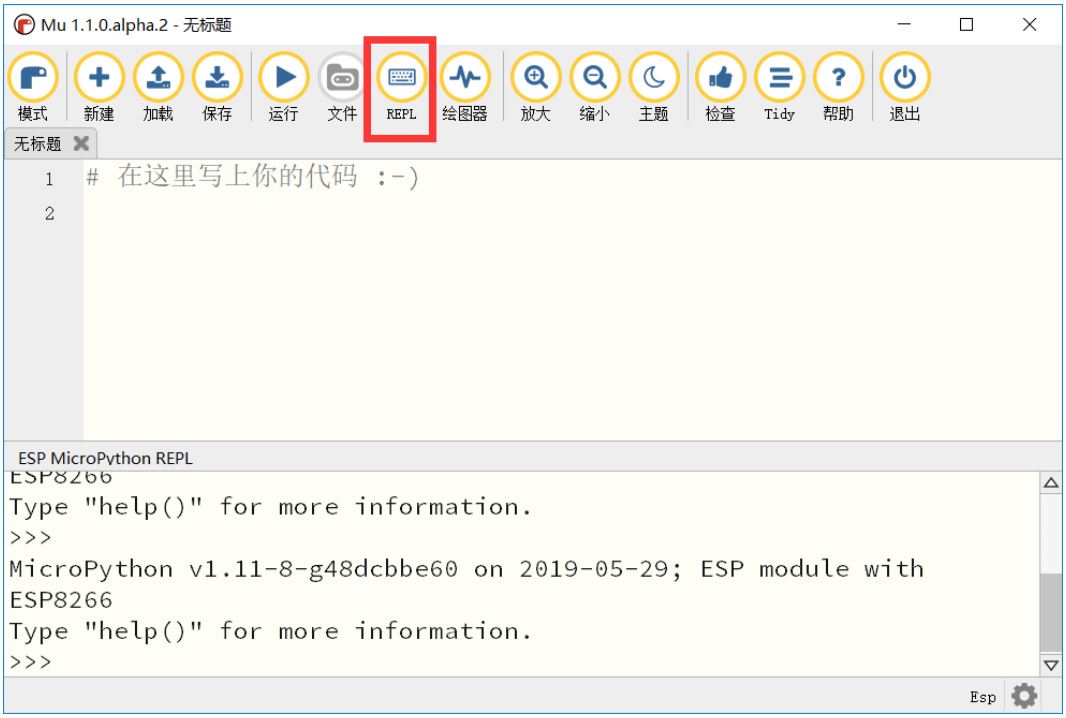


图 7-10

除此之外，还可以使用其它串口软件来进行 REPL 调试：

（1）Putty：位于 MicroPython 开发套件配套资料\开发工具\串口终端工具\Putty.exe；

（2）超级终端：位于 MicroPython 开发套件配套资料\开发工具\Windows\串口终端工具 文件夹下

7.1.3.3 文件系统

pyWiFi 里面内置了文件系统，可以简单理解成上电后运行的 python 文件，这个可以通过 Mu 非常方便地读取。

连接开发板，打开 Mu，先按一下开发板的复位键，然后点击上方“文件”按钮，可以在下方弹出的方框看到开发板里的文件和本地 PC 的文件。

（由于串口是共用的，因此有些时候可能打不开，这时候要先关掉 Mu 里所有文件或者 repl 窗口，按下开发板复位键，再点击打开。）

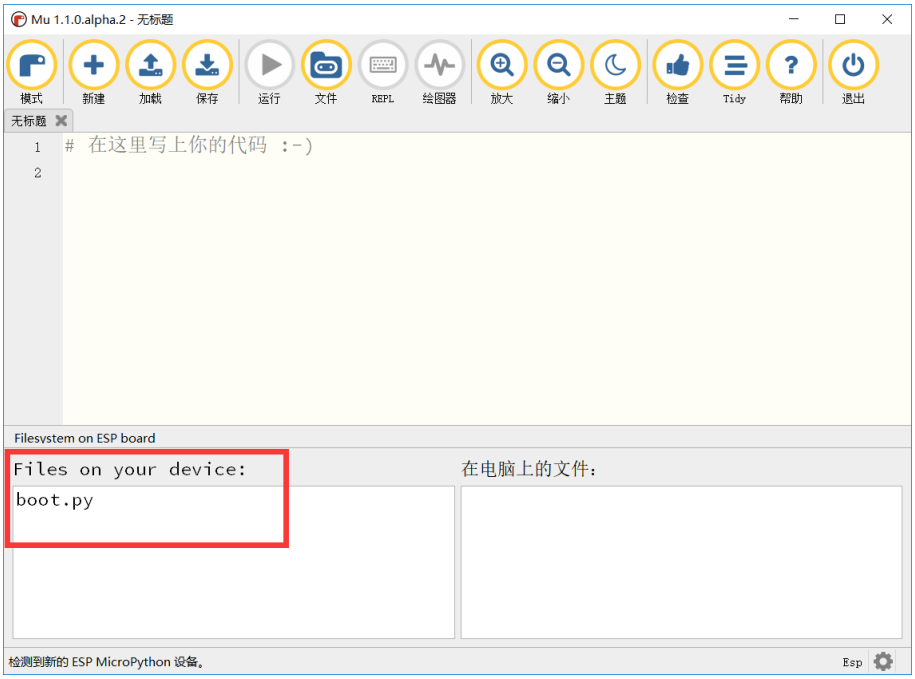


图 7-11

从上图可以看到开发板上的有个“boot.py”文件，这是系统启动后执行的第一个文件，通常做初始化工作。我们尝试将 boot.py 文件拖动到电脑的空白区，拖完后即完成从设备到电脑的文件复制，反过来就是从电脑复制文件到设备，使用非常方便直观。

复制完成后可以点击电脑端文件右键，选择在 Mu 中打开，即可查看和编辑该文件。**（新版固件可能没 boot.py 文件，不影响使用。）**

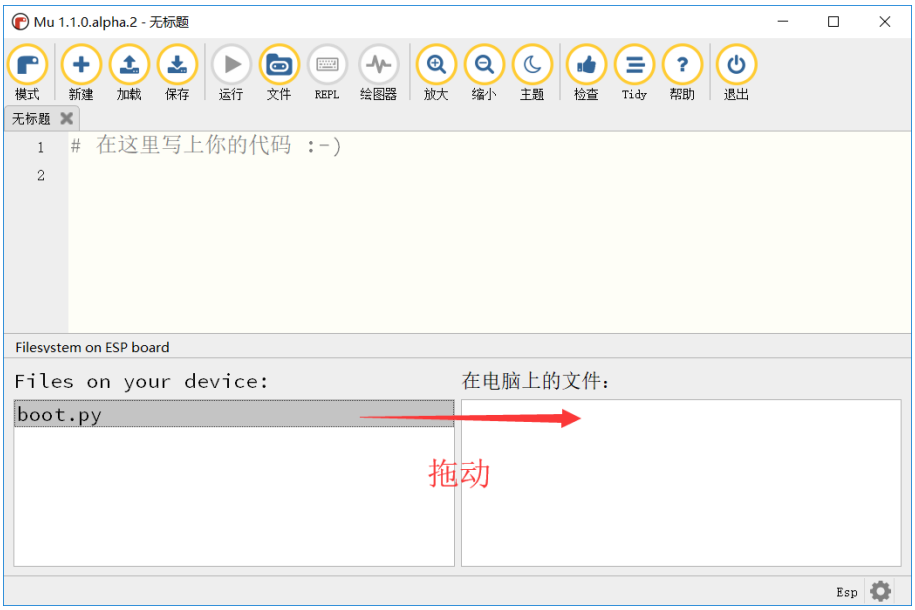


图 7-12

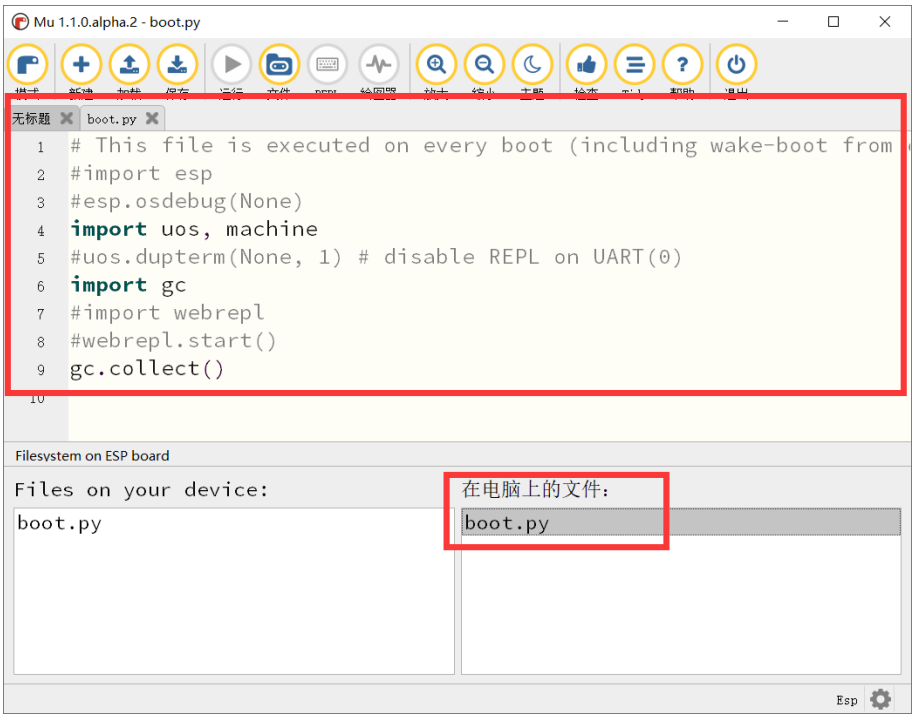


图 7-13 查看本地文件

那么本地文件都放在什么地方呢，可以在 Mu 中**双击**文件上方名称，即可在弹出的对话框中看到该文件的保存位置。

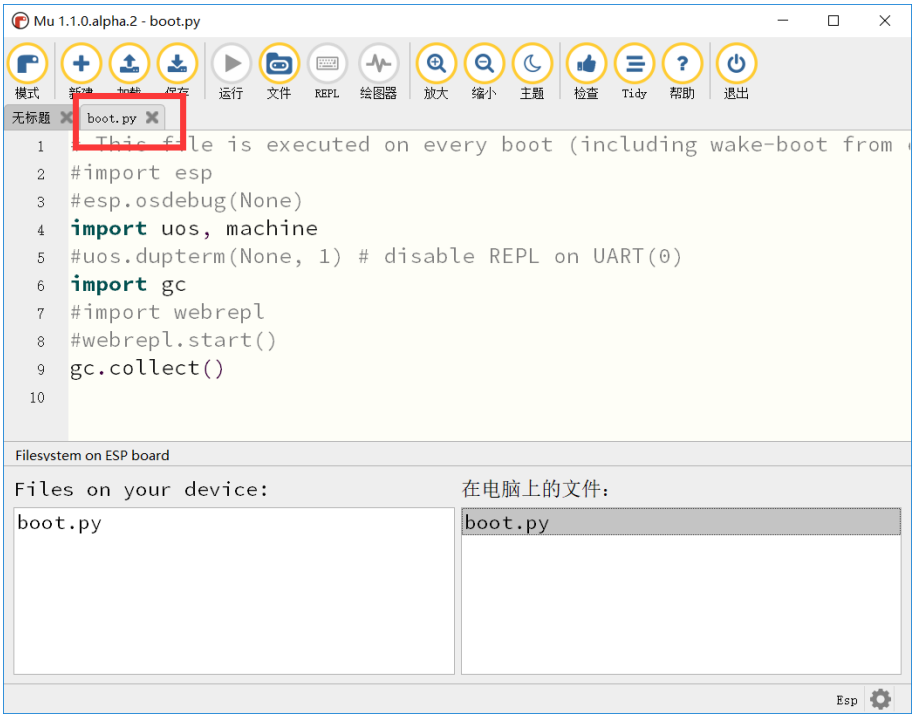


图 7-14 双击文件名

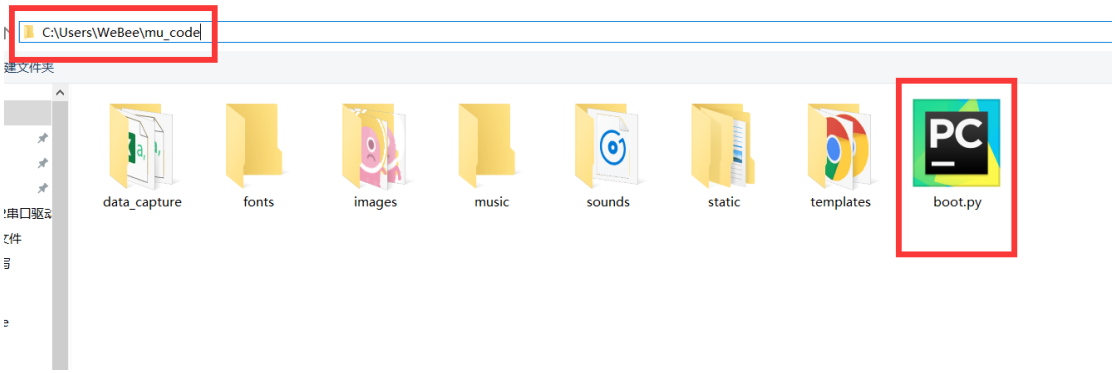


图 7-15 本地文件位置查看

我们在 Mu 中新建一个空白文件，保存名称为 main.py 到刚刚查看到的那个本地文件的路径：



图 7-16

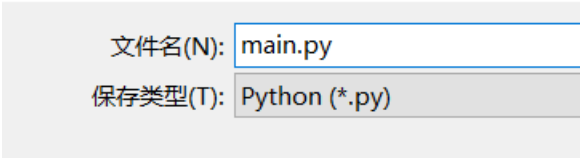


图 7-17

点击文件按钮关闭文件下方方框，再次点击刷新一下，可以发现出现了刚刚保存的 main.py 文件。将 main.py 文件拖动到左边设备栏，可以看到设备栏出现了 main.py 文件，说明复制成功。这就是使用 Mu 实现电脑跟设备文件相互传输的方法。



图 7-18

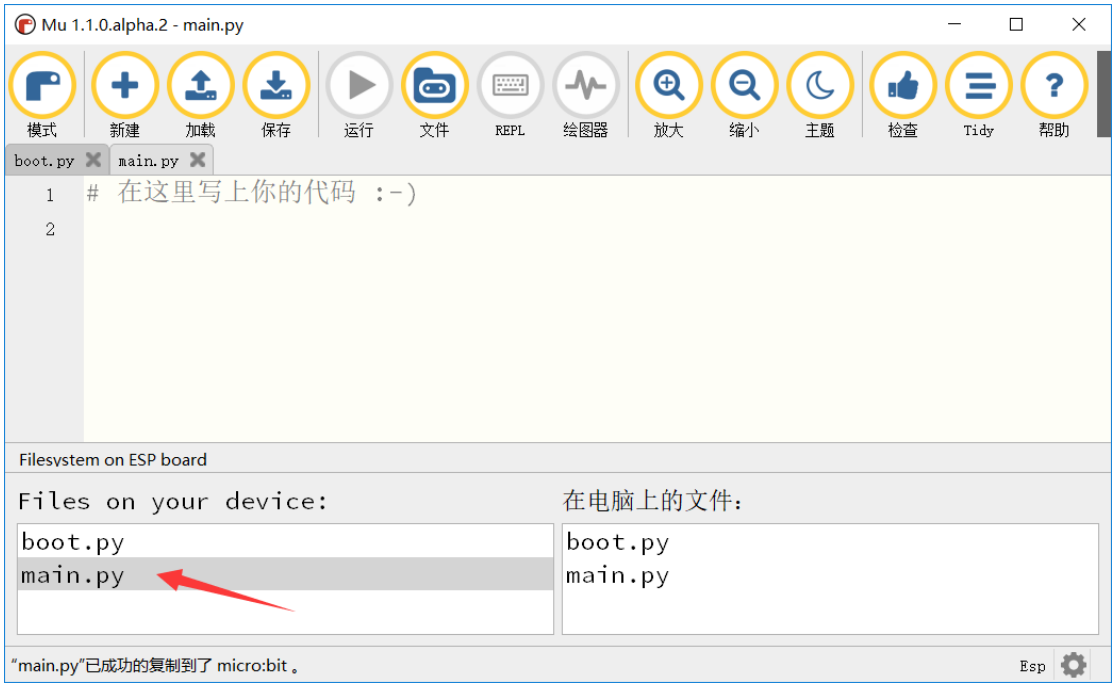


图 7-19 成功复制到设备

main.py 是主文件，是运行完 boot.py 后执行的第一个程序文件，我们主要也是在 main.py 文件里面编程，这在后面的实验会详细讲解。

7.1.3.4 固件更新

我们的开发板出厂已经烧录好了固件，固件更新是指重新烧写开发板的出厂文件或者是升级的固件，烧录使用上海乐鑫提供的官方软件烧录：

找到路径：**零一科技（01Studio）MicroPython 开发套件配套资料_latest\01-开发工具\01-Windows\固件更新工具\flash_download_tools_v3.6.5** 下的 flash_download_tools_v3.6.5.exe 软件，双击打开。

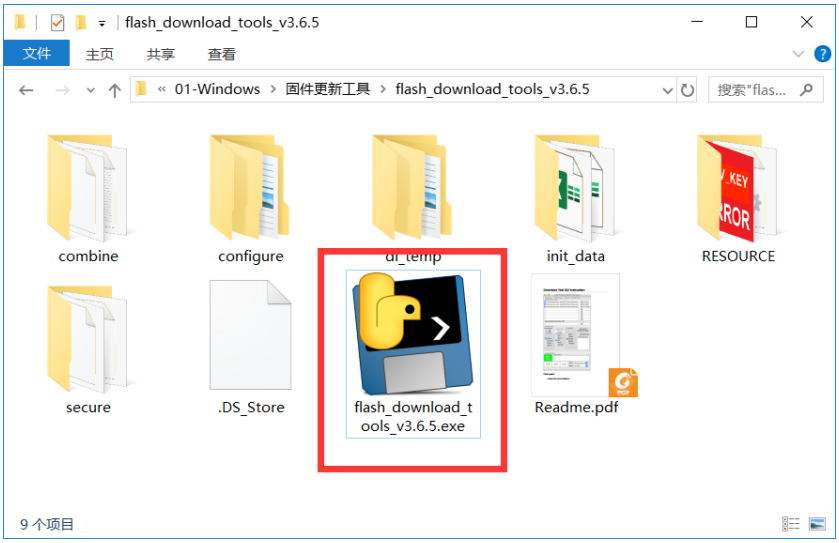


图 7-20

选择 ESP8266 Download tool :

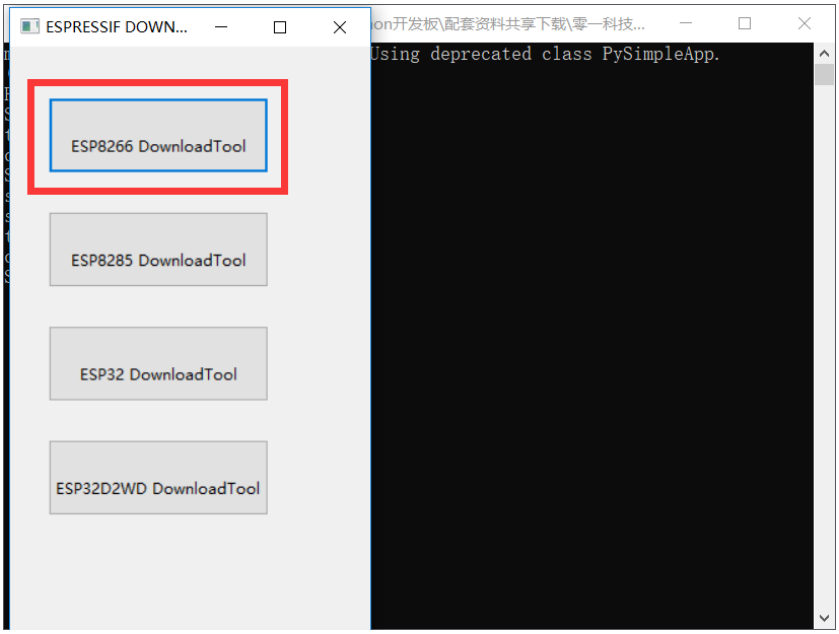


图 7-21 选择 ESP8266

选择 SPIDownload，在下图箭头位置点击，选择要烧录固件。固件位置位于：**零一科技（01Studio）MicroPython 开发套件配套资料\03-相关固件\01-pyWiFi-ESP8266** 目录下。

其它配置选项也请参考下图：（COM 串口是选择自己的串口，在设备管理器查询。）

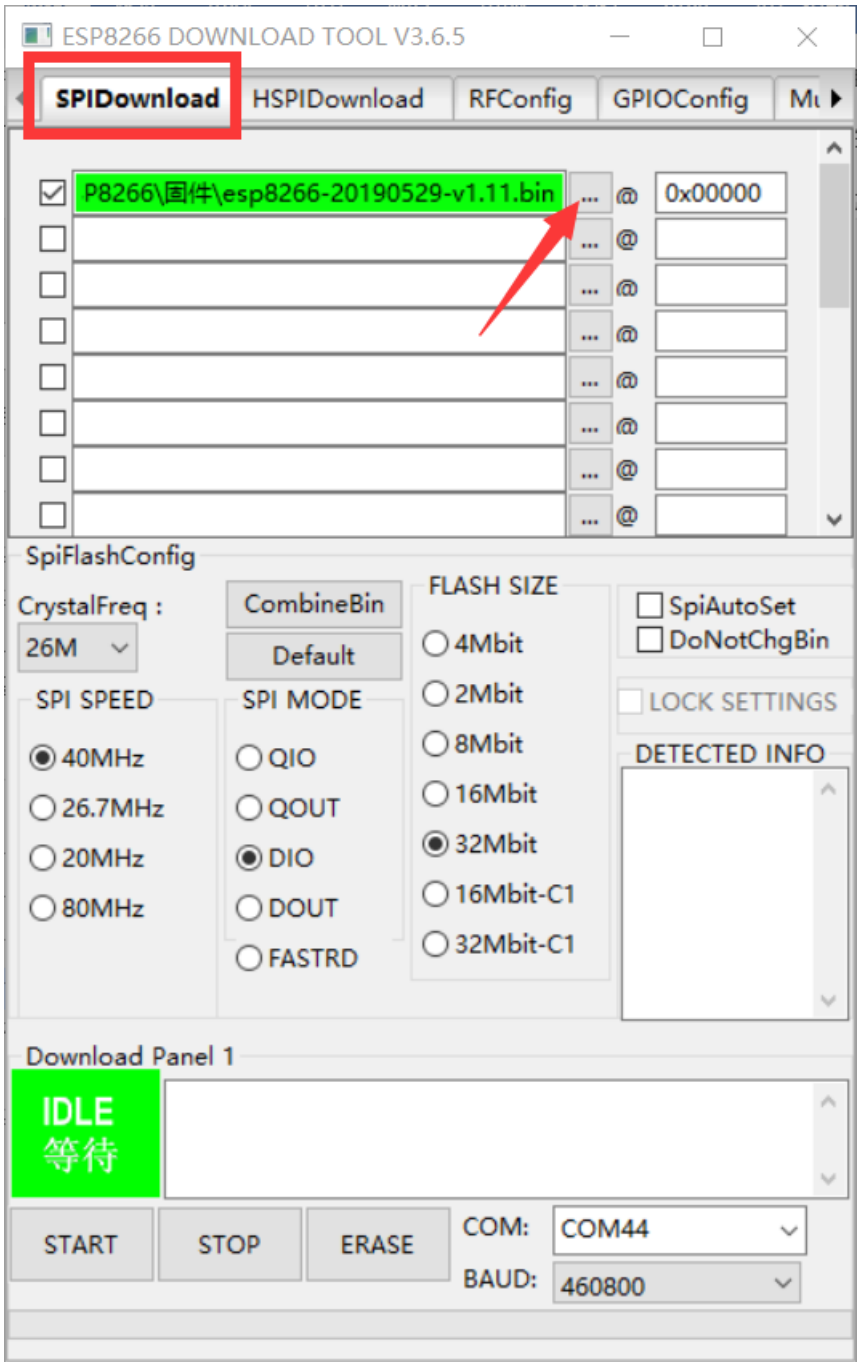


图 7-22 配置

配置好后，先点击“ERASE”按钮刷除模块里面内容。点击软件下方“ERASE”按钮，刷除成功后，左边绿色框出现完成字样。

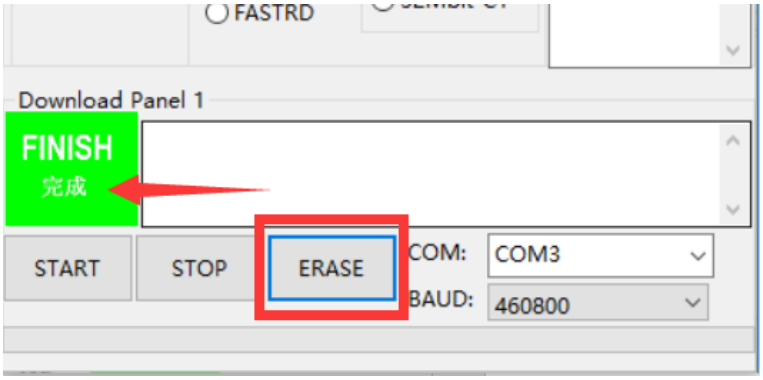


图 7-23 刷除 flash 成功

刷除成功后，点击“START”按钮开始烧录，烧录完成有左边绿色框出现“完成”字样。完成后记得点”stop”按钮或者关闭软件释放串口。

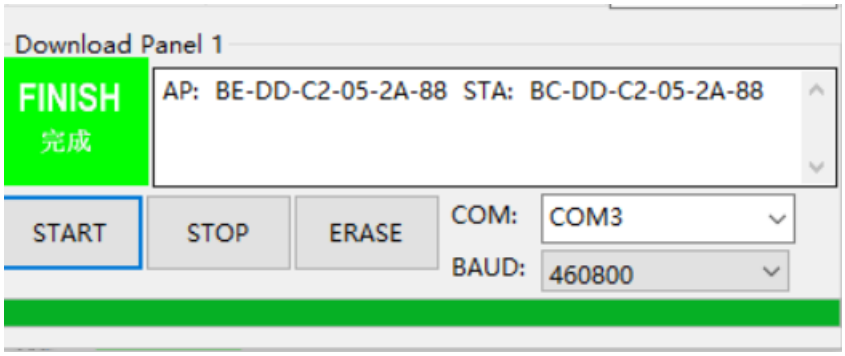


图 7-24 烧录成功

烧写完成后，打开 Mu 的 REPL 串口调试（也可以使用 putty 或其它串口调试工具），发现出现了熟悉的 MicroPython 调试模式。输入以下两句指令，可以检测固件的完整性。

```
import esp

esp.check_fw()
```

当系统返回 True 时候，说明固件是完整的。

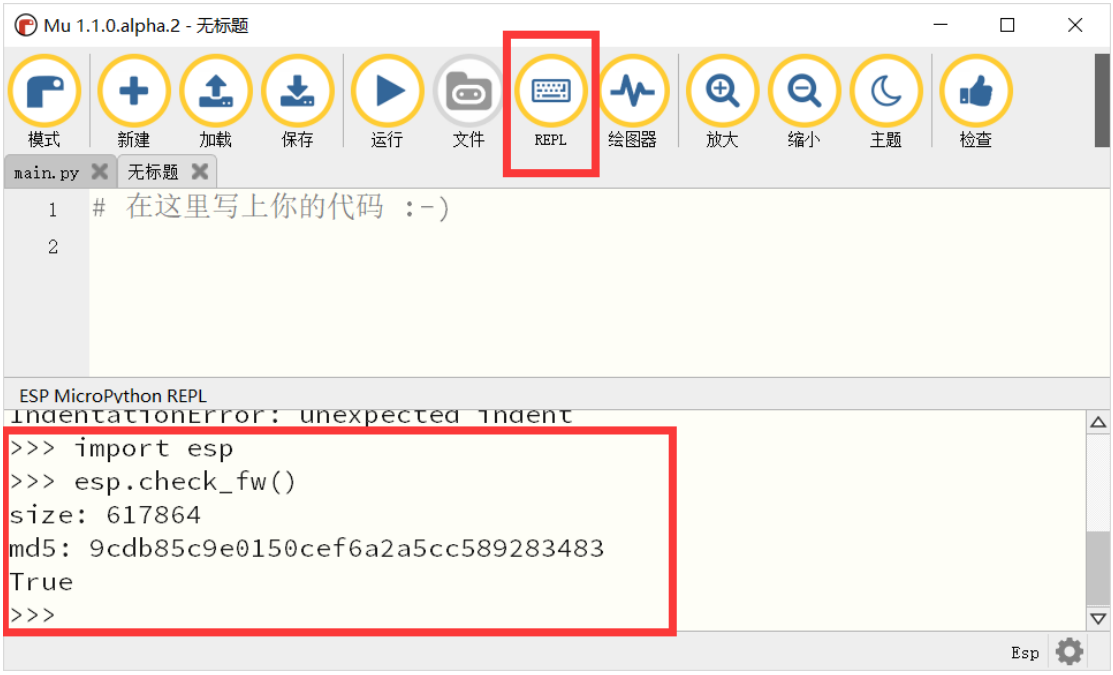


图 7-25 点击上方 REPL 按钮进入调试



图 7-26 固件完整性检测

7.2 基于 Mac OS

还记得当年乔布斯从大信封里面拿出 macbook 么，苹果和微软在 PC 时代走了两个极端，乔布斯主导下的苹果要求软硬一体化，封闭，务求最佳的用户体验；而微软则主打开发，让操作系统兼容不同的 PC 厂家，使得其 windows 一度占据了 90%的市场份额。从商业角度，盖茨毫无疑问是赢家，就个人而言，我更认可乔布斯的方式，现在的 win10 和微软自家的笔记本，某程度上都有一体化的思想。我用了半天的 Macbook 和 MAC 系统，如果不是要做硬件发，基本上可以放弃十多年的 windows 了。

7.2.1 安装 python3

在 MAC 系统下，安装 python3 方法也很简单，在官网 www.python.org 下载 Mac 版 python3，根据自己系统情况选择 32bit 或者 64bit 下载安装。

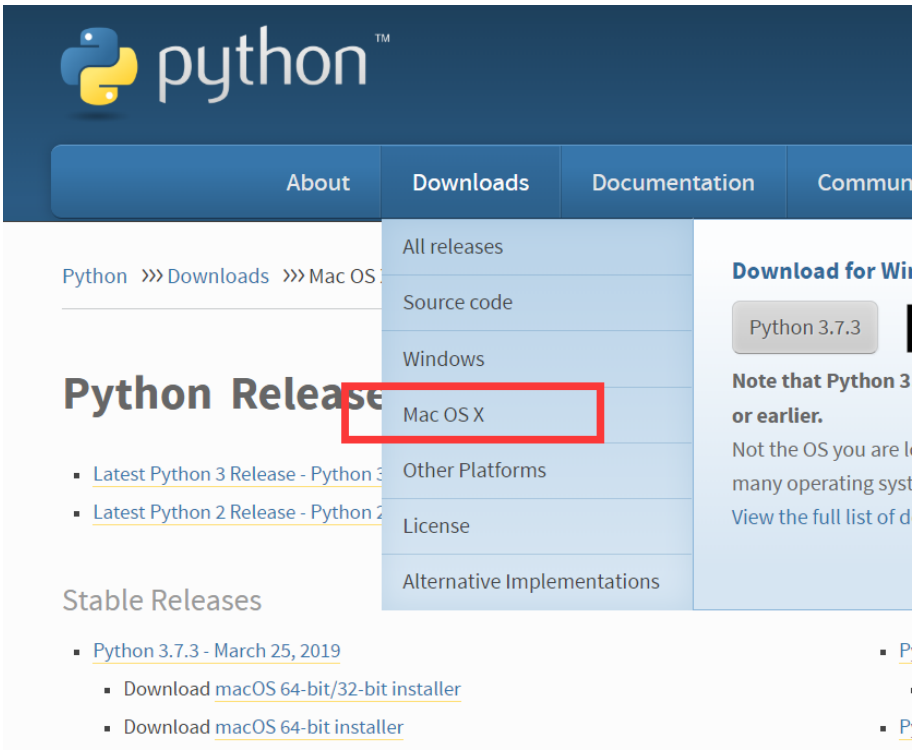


图 7-27

安装完成后在终端输入 python3，出现版本好的话说明安装成功。

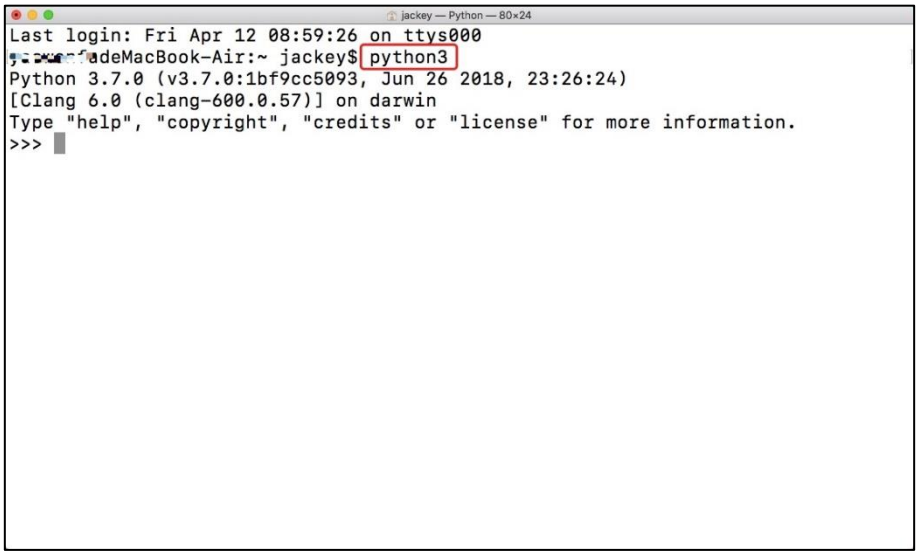


图 7-28 成功安装

7.2.2 安装开发软件 Mu

开发软件是指我们用来写代码的工具，上一节那个命令框里面其实就是调用了自带的编程器，但是我想没有人希望一直在那黑框里面撸代码。 Python 拥有众多的编程器，如果你之前已经熟练掌握 python 或已经使用 python 开发，那么可以直接使用你原来习惯的开发软件来编程。如果你是初学者或者喜欢简单而快速应用，我们推荐使用 Mu。

Mu 是一款开源软件，以极简方式设计，对 MicroPython 的兼容性非常友善。而且支持 Windows、Mac OS、Linux、树莓派。由于开源，所以软件迭代速度非常快，功能日趋成熟。具体安装方法如下：

在 https://codewith.mu/en/download 下载地址，选择自己的开发平台进行下载安装即可：

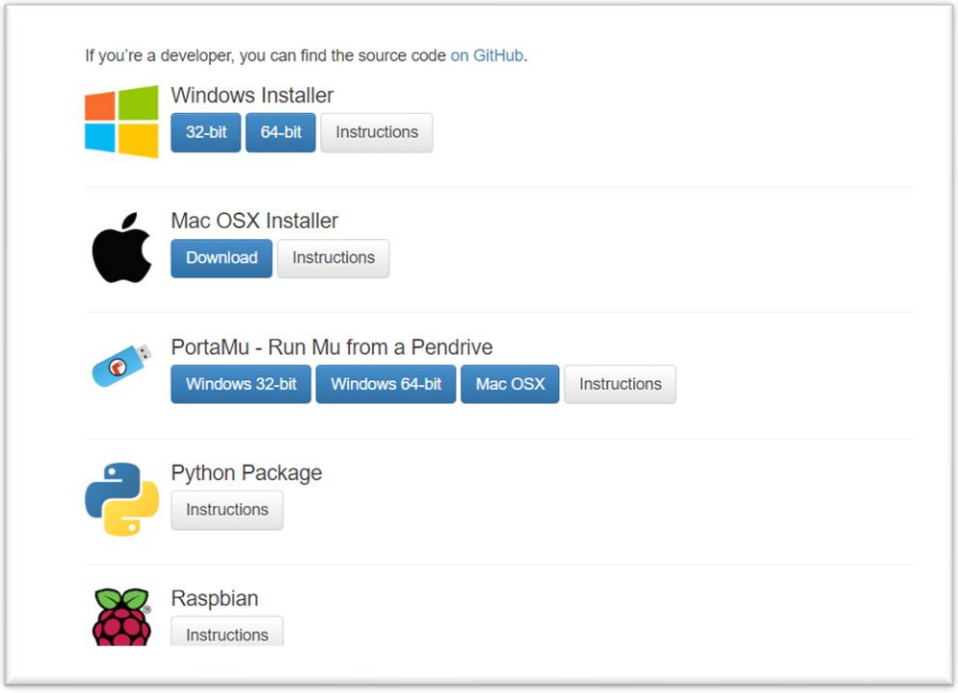


图 7-29 稳定版下载

由于 Mu 更新非常快，因此社区也同时提供最新版下载，给愿意尝鲜和给予软件反馈的用户，下载位置如下：

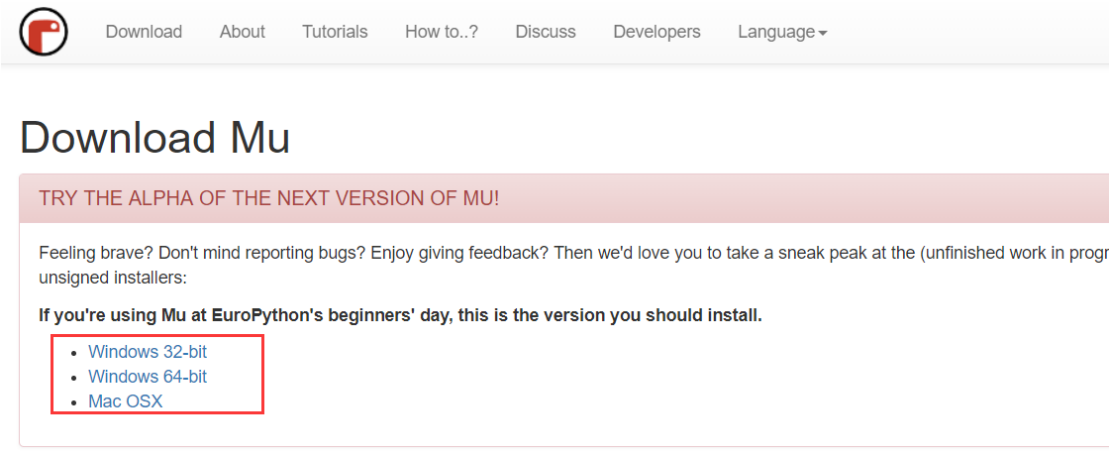


图 7-30 尝鲜版下载

我们推荐使用尝鲜版，能使用最新的功能。

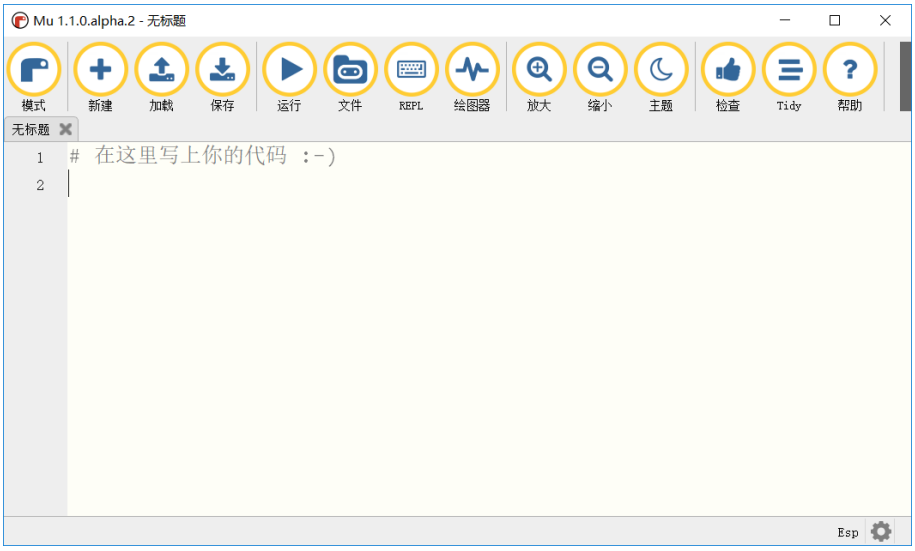


图 7-31 Mu 主界面

点击左上角“模式”，弹出对话框，可以看到支持 MicroBit、ESP 等一系列 MicroPython 产品。通常情况下在 USB 线插入能自动识别设备硬件类型。基于 pyWiFi 系列产品选择【ESP MicroPython】进行开发。

至此，Mu 安装完成。

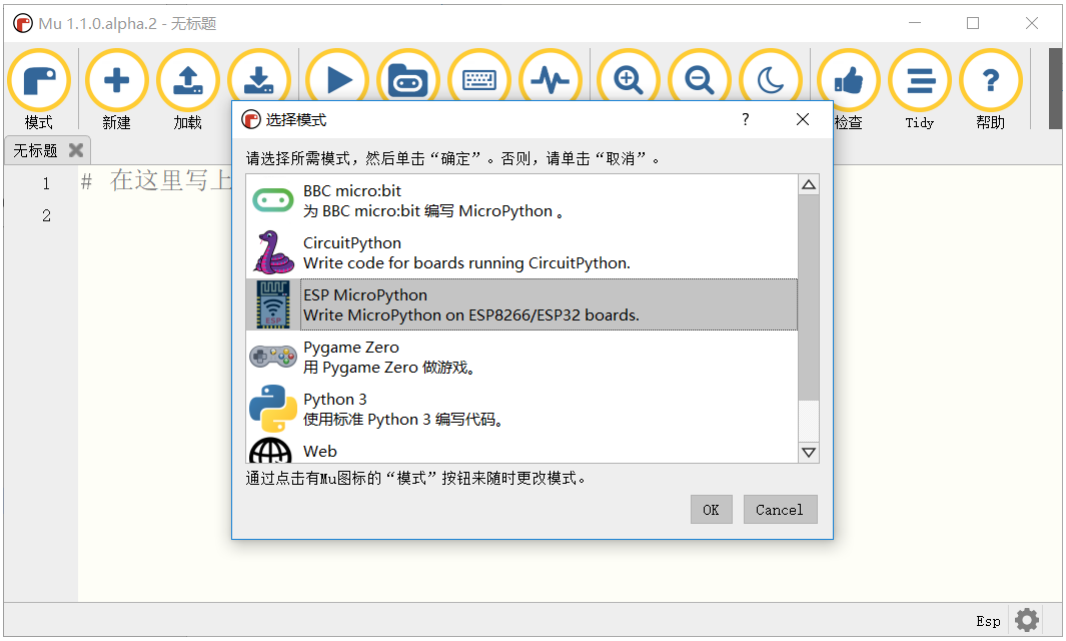

Mu 1.1.0.alpha.2 - 无标题
模式
新建
加载
检查
Tidy
帮助
无标题
1
2
在这里写上
选择模式
请选择所需模式，然后单击“确定”。否则，请单击“取消”。
BBC micro:bit
为 BBC micro:bit 编写 MicroPython。
CircuitPython
Write code for boards running CircuitPython.
ESP MicroPython
Write MicroPython on ESP8266/ESP32 boards.
Pygame Zero
用 Pygame Zero 做游戏。
Python 3
使用标准 Python 3 编写代码。
Web
通过点击有Mu图标的“模式”按钮来随时更改模式。
OK
Cancel
Esp

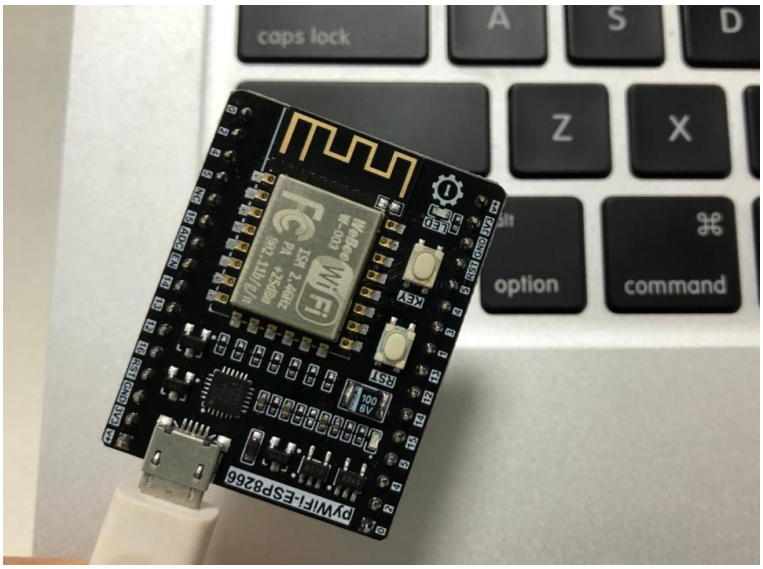
图 7-32

7.2.3 开发套件使用

7.2.3.1 驱动安装

Macbook 自带这个版本的串口驱动，一般不需要再安装，查看方法如下：

将 pyWiFi-ESP8266 通过 USB 线连接到 MacBook：



点击左上角苹果图标—关于本机—系统报告—选择 USB。可以看到相关驱动如下图所示：



图 7-33 关于本机



图 7-34　CP210X USB 转串口驱动

7.2.3.2 REPL 串口交互调试

基于 Mac OS 的 REPL 串口交互调试可以使用 Minicom 或 Mu 软件自带的 REPL 功能。

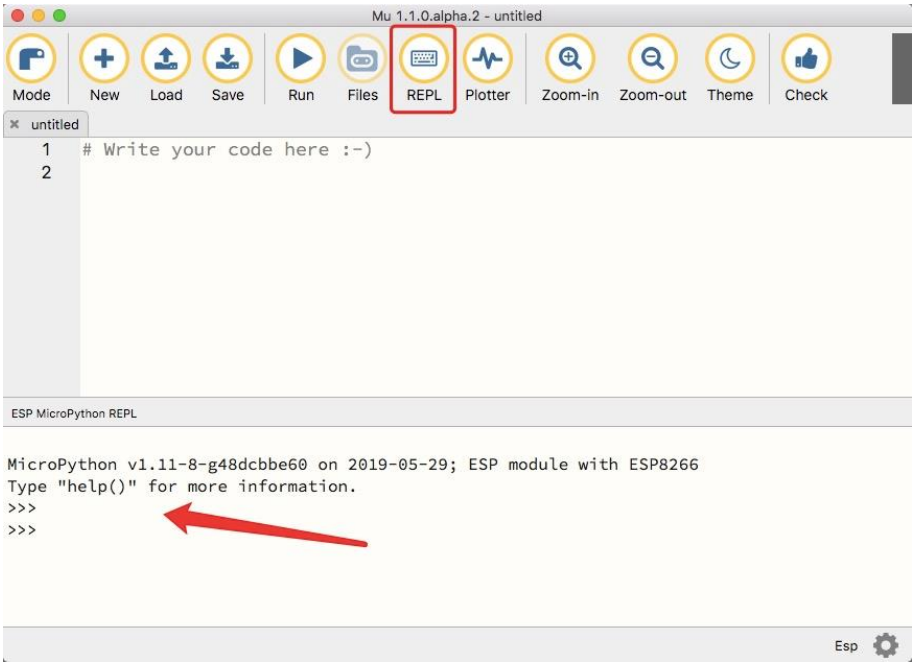


图 7-35 Mu 的 REPL

有关 Minicom 的使用方法请参阅：**第二部分 3.2.3.2 串口终端 minicom** 内容。

7.2.3.3 文件系统

这里是指实现电脑与开发板之间的文件传输，具体请参阅：**7.1.3.3 文件系统** 章节内容，这里不再重复

7.2.3.4 固件更新

01Studio 的开发板出厂已经烧录好了固件，固件更新是指重新烧写开发板的出厂文件或者是升级的固件，Mac OS 下没有跟 Windows 一样提供可视化 Flash 烧写软件，因此需要通过 esptool.py 工具结合终端命令来烧写固件。

打开终端，输入以下命令安装 esptool:

```
pip install esptool
```

安装完成后即可使用 esptool 工具来烧写固件，步骤如下：

第一步：刷除 Flash：

在终端输入下面命令按回车。

```
esptool.py --port /dev/tty.SLAB_USBtoUART erase_flash
```

刷除成功后出现以下信息

```
[redacted]MacBook-Air:~ jackey$ esptool.py --port /dev/tty.SLAB_USBtoUART erase_flash
esptool.py v2.6
Serial port /dev/tty.SLAB_USBtoUART
Connecting........_
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
MAC: b4:e6:2d:52:91:a6
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
Chip erase completed successfully in 2.5s
```

图 7-36 刷除成功提示

第二步：烧写固件：

先将固件放到 MacBook 桌面，终端需要通过“cd Desktop/”命令进入到桌面位置（目的是让终端当前位置和固件在同一路径下）：

在终端输入下面命令按回车（esp8266-20190529-v1.11.bin 是固件名称，可以根据自己的实际情况修改，输入时候可以通过 tab 键补全。）

```
esptool.py --port /dev/tty.SLAB_USBtoUART --baud 460800 write_flash --
flash_size=detect 0 esp8266-20190529-v1.11.bin
```

烧写成功如下图所示：

```
[redacted]MacBook-Air:Desktop jackey$ esptool.py --port /dev/tty.SLAB_USBtoUART --baud 460800 write_flash --flash_size=detect 0 esp8266-20190529-v1.11.bin
esptool.py v2.6
Serial port /dev/tty.SLAB_USBtoUART
Connecting........_
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
MAC: b4:e6:2d:52:91:a6
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 1MB
Flash params set to 0x0020
Compressed 617880 bytes to 402086...
Wrote 617880 bytes (402086 compressed) at 0x00000000 in 9.7 seconds (effective 510.9 kbit/s)...
Hash of data verified.
```

图 7-37 烧写成功

烧写完成后，打开 Mu 的 REPL 串口调试（也可以使用其它串口调试工具），发现出现了熟悉的 MicroPython 调试模式。输入以下两句指令，可以检测固件的完整性。

```
import esp

esp.check_fw()
```

当系统返回 True 时候，说明固件是完整的。

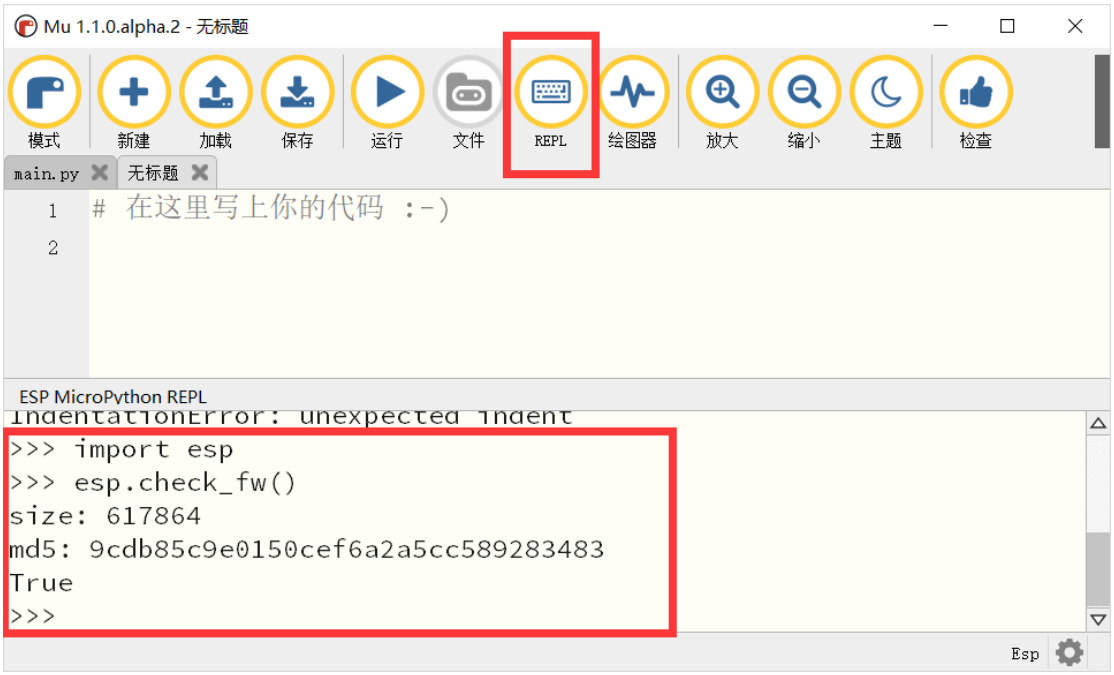


图 7-38 点击上方 REPL 按钮进入调试



图 7-39 固件完整性检测

7.3 基于 Linux

基于 Linux 平台开发环境搭建请阅读 **第二部分 3.3 基于 Linux** 章节内容，这里不再重复。

第8章　基础实验

MicroPython 更强调的是针对应用的学习，强大的底层库函数让我们可以直接关心功能的实现，也就是说我们只要理解和熟练相关的函数用法，就可以很好的玩转 MicroPython。它让我们可以做到不关心硬件和底层原理（当然有兴趣和能力的小伙伴可以深入研究）而直接跑起硬件。

基础实验是针对一些简单的实验学习讲解，可以让我们更快和更直接感受到 MicroPython 针对嵌入式开发的强大之处， 我后面的学习和开发夯实基础。

请先看实验讲解格式预览，每一节我们都会以以下形式讲解，图文并茂，务求达到快速理解和学习的作用：

（1） 前言：简单介绍这个实验；

（2） 实验平台：实验所用到的开发板、配件和接线说明；

（3） 实验目的：本实验要实现的功能；

（4） 实验讲解：对函数、代码、编程方法以及实验的详细讲解；

（5） 实验结果：记录程序下载到开发板上的图片示例；

（6） 总结:对实验进行总结。

8.1 点亮第一个 LED

- **前言：**

相信大部分人开始学习嵌入式单片机编程都会从点亮 LED 开始，我们 MicroPython 的学习也不例外，通过点亮第一个 LED 能让你对编译环境和程序架构有一定的认识，为以后的学习和更大型的程序打下基础，增加信心。

- **实验平台：**

pyWiFi-ESP8266。

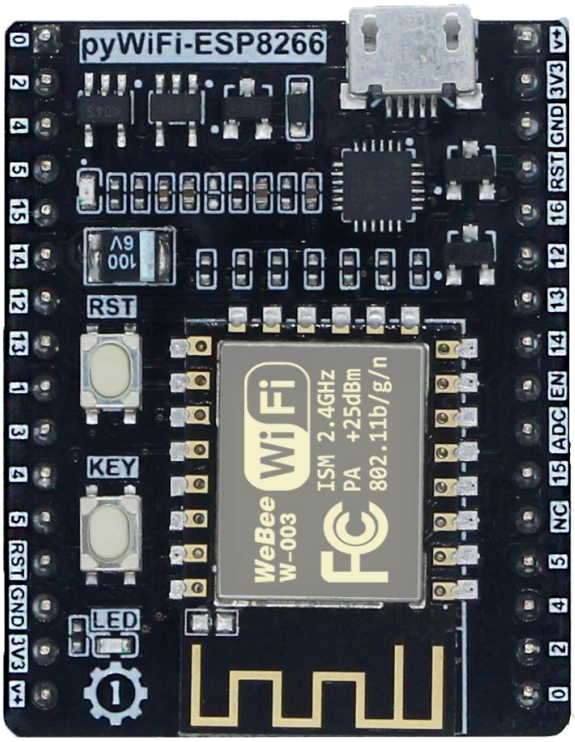


图 8-1　pyWiFi-ESP8266

- **实验目的：**

点亮 LED（蓝灯）。

- **实验讲解：**

pyWiFi-ESP8266 上有 1 个 LED（蓝色），控制 LED 使用 machine 中的 Pin 对象，其构造函数和使用方法如下：

构造函数
led=machine.Pin(id,mode,pull)
构建 led 对象。id:引脚编号；mode:输入输出方式；pull:上下拉电阻配置。
使用方法
led.value([x])
引脚电平值。输出状态：x=0 表示低电平，x=1 表示高电平；输入状态：无须参数，返回当前引脚值。
led.on()
使引脚输出高电平“1”。
led.off()
使引脚输出低电平“0”。

表 8-1　Pin 对象

上表对 MicroPython 的 machine 中 Pin 对象做了详细的说明，machine 是大模块，Pin 是 machine 下面的其中一个小模块，在 python 编程里有两种方式引用相关模块:

方式 1 是：import machine，然后通过 machine.Pin 来操作；

方式 2 是：from machine import Pin,意思是直接从 machine 中引入 Pin 模块，然后直接通过构建 led 对象来操作。显然方式 2 会显得更直观和方便，本实验也是使用方式 2 来编程。代码编写流程如下：

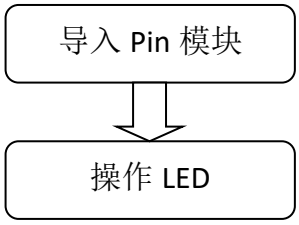


图 8-2 代码编写流程

LED 跟模块引脚 2 相连，通过输出低电平方式点亮。

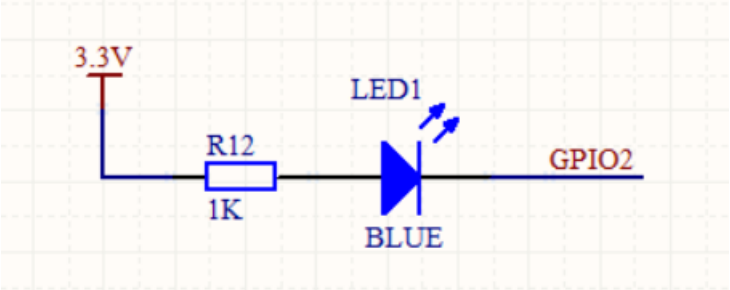


图 8-3 LED 接线图

参考代码如下 ：

```
'''
实验名称：点亮 LED 蓝灯
版本：v1.0
日期：2019.7
作者：01Studio
'''

from machine import Pin #导入 Pin 模块
led=Pin(2,Pin.OUT) #构建 led 对象，GPIO2,输出
led.value(0) #点亮 LED
```

运行程序有两个方法：

方法一：

编写好代码后点击 Mu 上方的“运行”按钮，可以直接观察到代码运行情况。这个方法不会将程序代码保存到 ESP8266 模块的 flash 里面。这注意是方便调试使用。

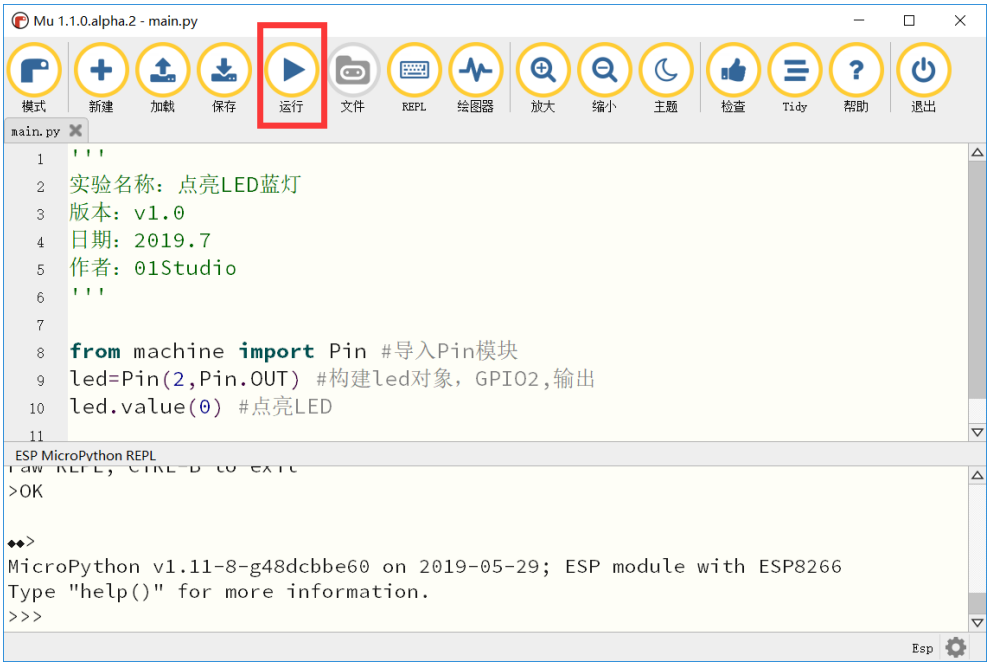


图 8-4 运行仿真

方法二：

将新建的文件保存名称为 “main.py”的 py 文件，使用 Mu 的文件功能，从本地拖动到设备。然后按下复位按键，设备运行相关代码，这个方式相当于将程序烧录到设备 flash。 可以脱机使用。 操作方法参考上一章文件系统章节

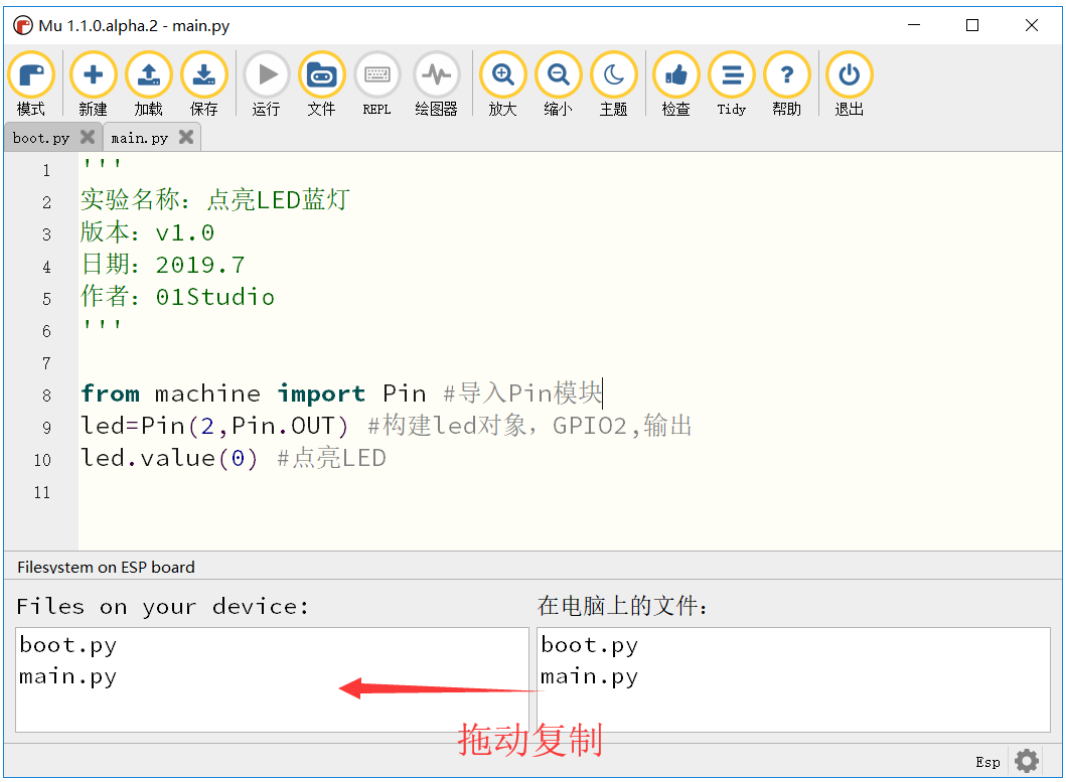


图 8-5 拷贝文件到设备

- **实验结果：**

下载程序，可以看到 LED（蓝灯）成功点亮。

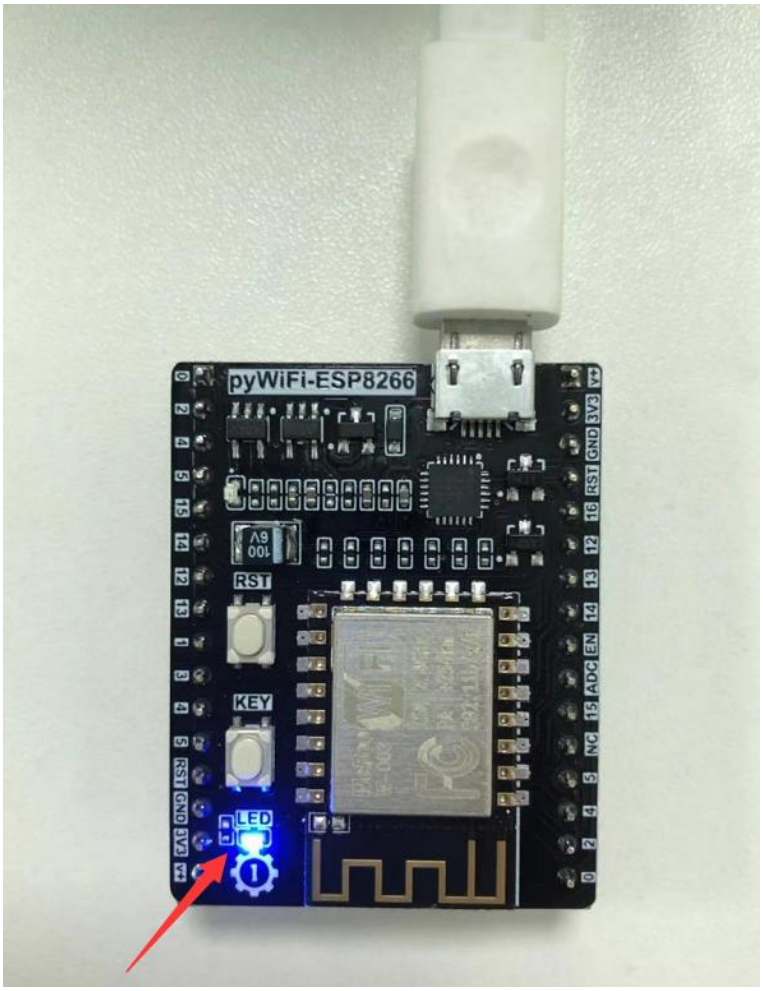


图 8-6　LED 点亮

- **总结：**

从第一个实验我们可以看到，使用 MicroPython 来开发关键是要学会构造函数和其使用方法，便可完成对相关对象的操作，在强大的模块函数支持下，实验只用了简单的两行代码便实现了点亮 LED 灯。

8.2 按键

- **前言：**

按键是最简单也最常见的输入设备，很多产品都离不开按键，包括早期的 iphone，今天我们就来学习一下如何使用 MicroPython 来编写按键程序。有了按键输入功能，我们就可以做很多好玩的东西了。

- **实验平台：**

pyWiFi-ESP8266。

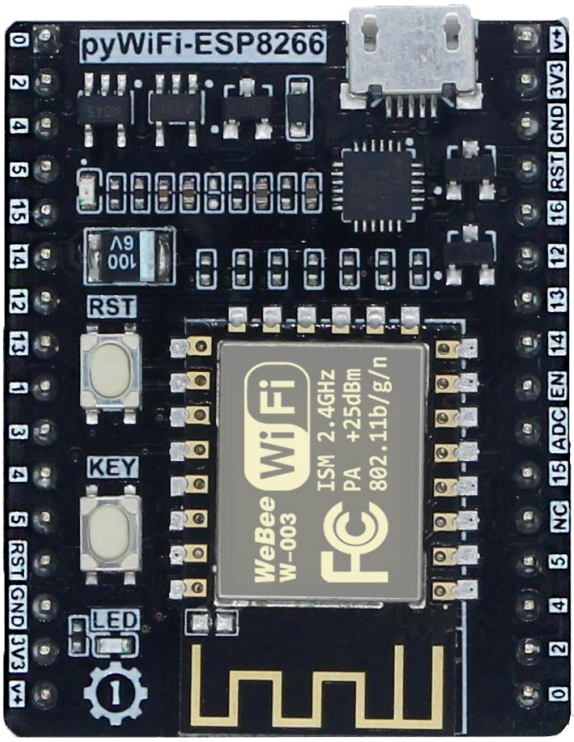


图 8-7　pyWiFi-ESP8266

- **实验目的：**

使用按键功能，通过检测按键被按下后，改变 LED（蓝灯）的亮灭状态。

- **实验讲解：**

pyWiFi-ESP8266 开发板上有 2 个按键，RST 和 KEY，RST 顾名思义是复位用的，所以真正自带可以用的就只有 1 个按键 KEY。

让我们先来搞清楚 micropython 里面 Pin 模块实现按键的构造函数和使用方法。

构造函数
KEY=machine.Pin(id,mode,pull)
构建按键对象。id:引脚编号；mode:输入输出方式；pull:上下拉电阻配置。
使用方法
KEY.value()
引脚电平值。输入状态：无须参数，返回当前引脚值 0 或者 1。

表 8-2　KEY 对象

可以看到跟上一节 LED 一样，只是输入/输出状态的一个改变。从下面原理图可以看到，我们只需要在开发板上电后判断 KEY 引脚的电平，当被按下时候引脚为低电平“0”。

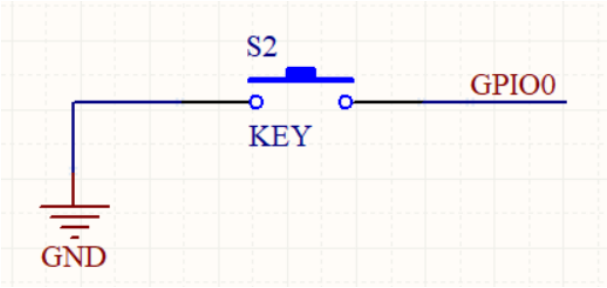


图 8-8 按键原理图

按键被按下时候可能会发生抖动，抖动如下图，有可能造成误判，因此我们需要使用延时函数来进行消抖：

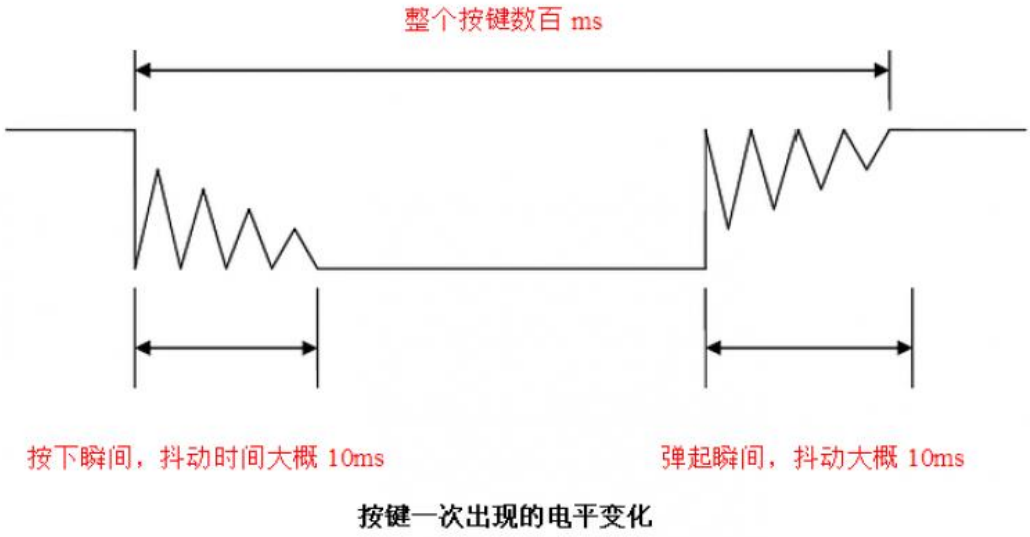


图 8-9 按键按下过程

常用的方法就是当检测按键值为 0 时，延时一段时间，大约 10ms，再判断按键引脚值仍然是 0，是的话说明按键被按下。延时使用 time 模块，使用方法如下：

```
import time

time.sleep(1)           # 睡眠 1 秒
time.sleep_ms(500)      # 睡眠 500 毫秒
time.sleep_us(10)       # 睡眠 10 微妙
start = time.ticks_ms() # 获取毫秒计时器开始值

delta = time.ticks_diff(time.ticks_ms(), start) # 计算从上电开始到当前时间的差值
```

编程流程图如下：

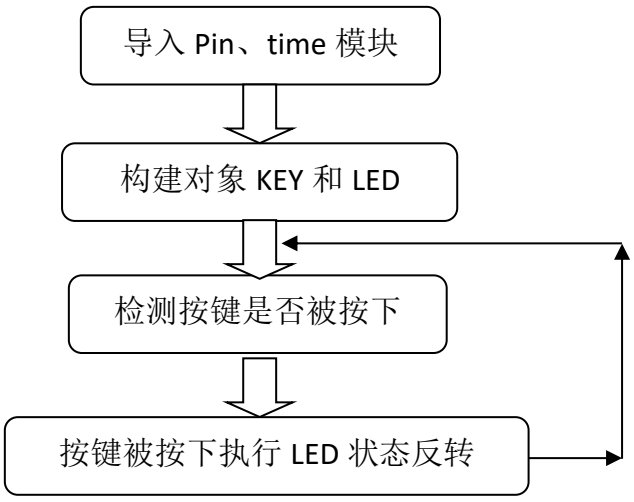


图 8-10 代码编写流程

实验参考代码如下：

```
'''
实验名称：按键
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过按键改变 LED 的亮灭状态
'''
from machine import Pin
import time

LED=Pin(2,Pin.OUT,value=1) #构建 LED 对象,开始熄灭
KEY=Pin(0,Pin.IN,Pin.PULL_UP) #构建 KEY 对象
state=1 #LED 引脚状态

while True:
    if KEY.value()==0:   #按键被按下
        time.sleep_ms(10) #消除抖动
        if KEY.value()==0: #确认按键被按下
            state=not state #改变 state 的值 0,1 切换
            LED.value(state) #LED 状态翻转
            while not KEY.value(): #检测按键是否松开
                pass
```

从上面代码可以看到，初始化各个对象后，进入循环，当检测到 KEY 的值为 0（按键被按下）时候，先做了 10ms 的延时，再次判断；

state 为 LED 状态的值，每次按键按下后通过使用 not 来改变。这里注意的是在 python 里使用‘not’而不是‘~’的方式。not 返回的是 True 和 False,即 0,1。而~ 是数值取反操作，会导致出错。

- **实验结果：**

将以上代码编写到 main.py 文件，拷贝到设备。复位后即可运行程序。可以看到每当按键 KEY 被按下后，LED 的亮灭状态发生改变。如下图所示：

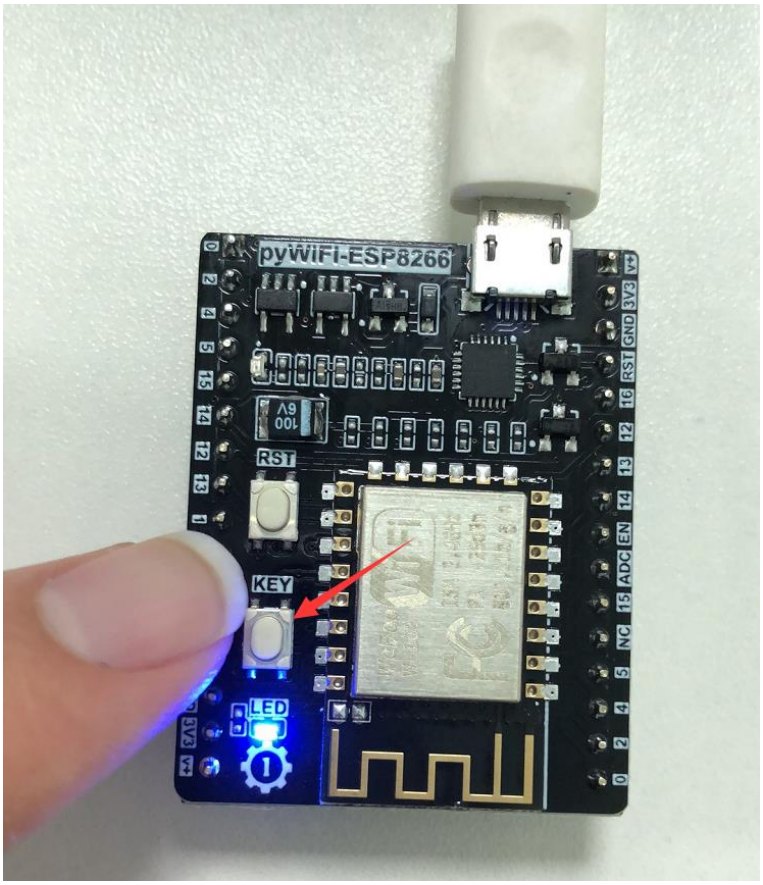


图 8-11

- **总结：**

按键作为我们学习的第一个输入设备，有了输入设备我们就可以跟硬件做人机交互了，这对后面的学习非常有意义。可以看到按键在 MicroPython 下开发也显得很简单。

8.3 外部中断

- **前言：**

前面我们在做普通的 GPIO 时候，虽然能实现 IO 口输入输出功能，但代码是一直在检测 IO 输入口的变化，因此效率不高，特别是在一些特定的场合，比如某个按键，可能 1 天才按下一次去执行相关功能，这样我们就浪费大量时间来实时检测按键的情况。

为了解决这样的问题，我们引入外部中断概念，顾名思义，就是当按键被按下(产生中断)时，我们才去执行相关功能。这大大节省了 CPU 的资源，因此中断的在实际项目的应用非常普遍。

- **实验平台：**

pyWiFi-ESP8266。

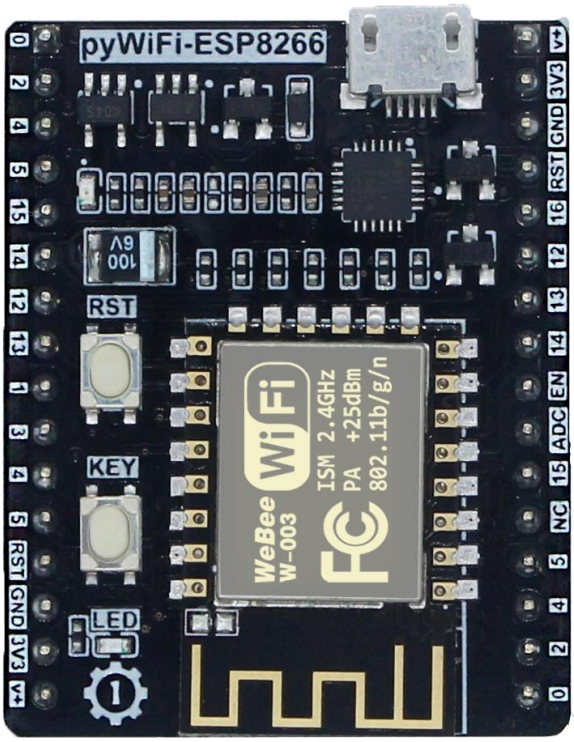


图 8-12　pyWiFi-ESP8266

- **实验目的：**

利用中断方式来检查按键 KEY 状态，被按键被按下（产生外部中断）后使 LED 的亮灭状态翻转。

- **实验讲解：**

外部中断也是通过 machine 模块的 Pin 子模块来配置，我们先来看看其配构造函数和使用方法：

构造函数
KEY=machine.Pin(id,mode,pull)
构建按键对象。id:引脚编号；mode:输入输出方式；pull:上下拉电阻配置。
使用方法
KEY.irq(handler,trigger)
配置中断模式。 handler:中断执行的回调函数； trigger: 触发中断的方式，共 4 种，分别是 Pin.IRQ_FALLING（下降沿触发）、Pin.IRQ_RISING（上升沿触发）、Pin.IRQ_LOW_LEVEL（低电平触发）、Pin.IRQ_HIGH_LEVEL（高电平触发）。

表 8-3 按键对象

上升沿和下降沿触发统称边沿触发。从上一节按键可以看到，按键被按下时一个引脚值从 1 到 0 变化的过程，边沿触发就是指这个过程。

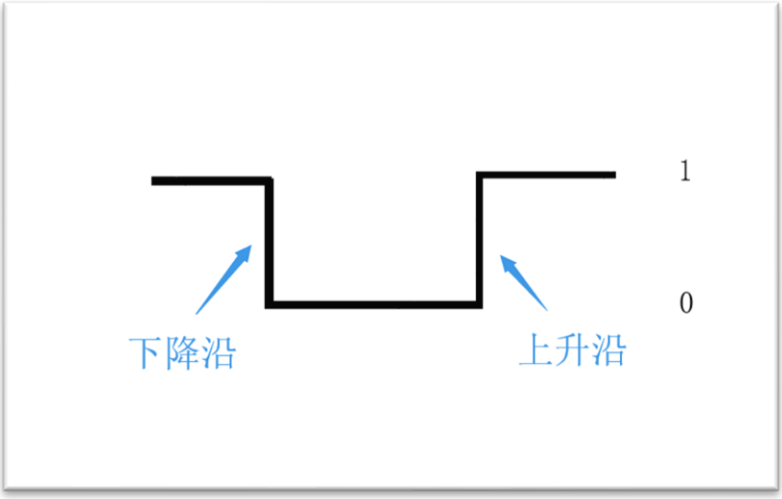


图 8-13 边沿触发

由此可见，我们可以选择下降沿方式触发外部中断，也就是当按键被按下的时候立即产生中断。

编程思路中断跟按键章节类似，在初始化中断后，当系统检测到外部终端时候，执行 LED 亮灭状态反转的代码即可。

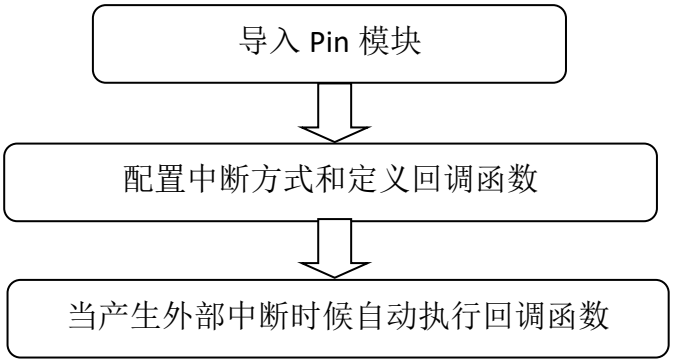


图 8-14 代码编写流程

参考程序代码如下：

```
'''
实验名称：外部中断
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过按键改变 LED 的亮灭状态（外部中断方式）
'''

from machine import Pin
import time

LED=Pin(2,Pin.OUT,value=1) #构建 LED 对象,开始熄灭
KEY=Pin(0,Pin.IN,Pin.PULL_UP) #构建 KEY 对象
state=1  #LED 引脚状态

#LED 状态翻转函数
```

```
def fun(KEY):
    global state
    time.sleep_ms(10) #消除抖动
    if KEY.value()==0: #确认按键被按下
        state = not state
        LED.value(state)

KEY.irq(fun,Pin.IRQ_FALLING) #定义中断，下降沿触发
```

以上代码中需要注意的地方：

1、state 是全局变量，因此在 fun 函数里面用该变量必须添加 global state 代码，否则会在函数里面新建一个样的变量造成冲突。

2、在定义回调函数 fun 的时候，需要将引脚对象 KEY 传递进去。

- **实验结果：**

将 main.py 文件拷贝到设备，复位。

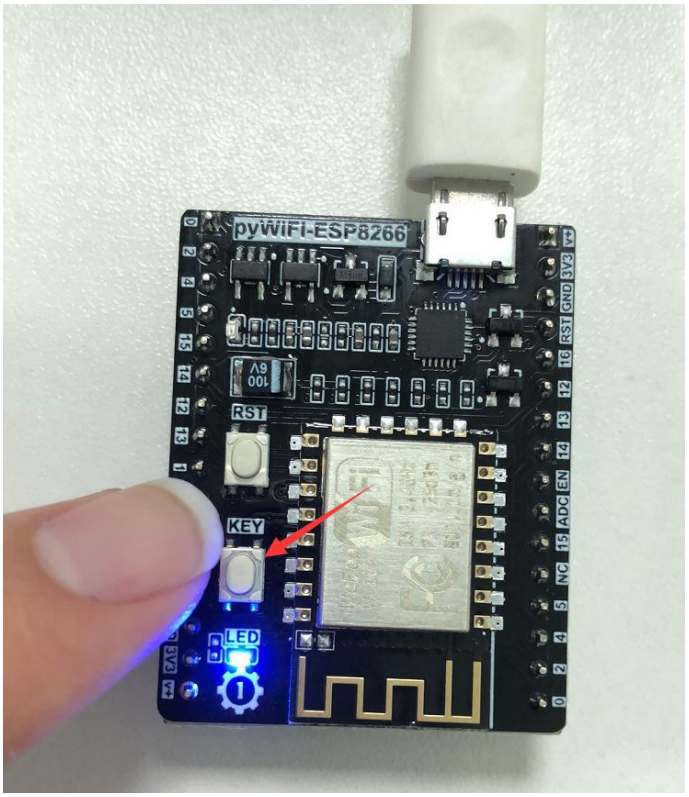


图 8-15 实验现象

- **总结：**

从参考代码来看，只是用了几行代码就实现了实验功能，而且相对于使用 while True 实时检测函数来看，代码的效率大大增强。外部中断的应用非常广，出来普通的按键输入和电平检测外，很大一部分输入设备，比如传感器也是通过外部中断方式来实时检测，这个在后面的章节会讲述。

8.4 定时器

- **前言：**

定时器，顾名思义就是用来计时的，我们常常会设定计时或闹钟，然后时间到了就告诉我们要做什么了。单片机也是这样，通过定时器可以完成各种预设好的任务。

- **实验平台：**

pyWiFi-ESP8266。

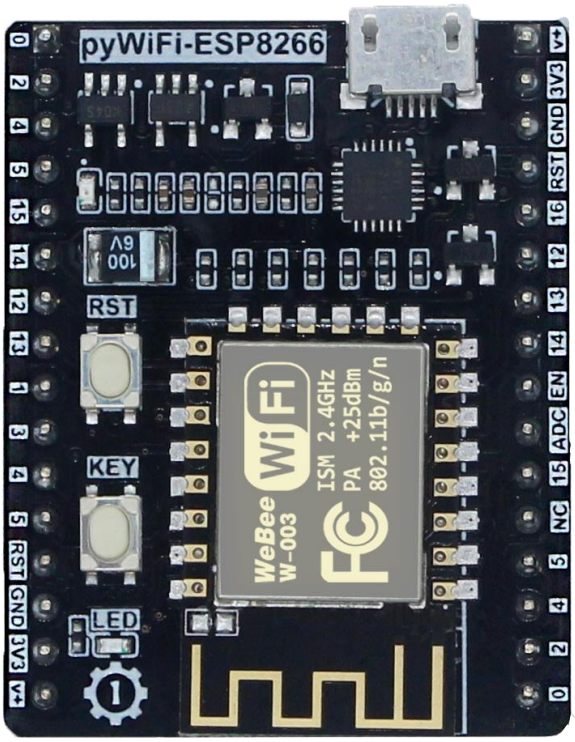


图 8-16 pyWiFi-ESP8266

- **实验目的：**

通过定时器让 LED 周期性每秒闪烁 1 次。

- **实验讲解：**

ESP8266 内置 RTOS（实时操作系统）定时器，在 machine 的 Timer 模块中。通过 MicroPython 可以轻松编程使用。我们也是只需要了解其构造对象函数和使用方法即可！

构造函数
tim=machine.Timer(-1)
构建定时器对象。RTOS 定时器编号为-1；
使用方法
tim.init(period,mode,callback)
定时器初始化。period:单位为 ms；mode：2 种工作模式，Timer.ONE_SHOT（执行一次）、Timer.PERIODIC（周期性）；callback:定时器中断后的回调函数。

表 8-4　定时器对象

定时器到了预设指定时间后，也会产生中断，因此跟外部中断的编程方式类似，代码编程流程图如下：

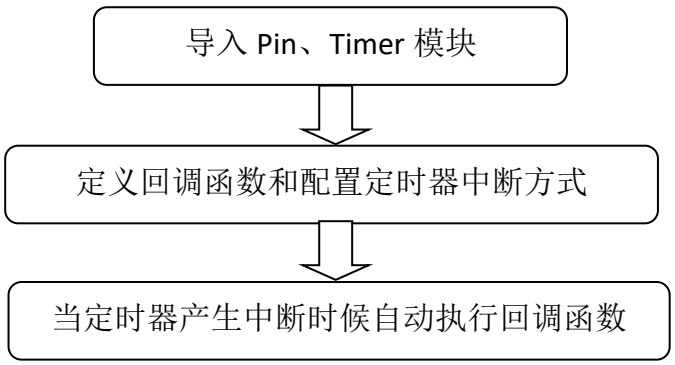


图 8-17 代码编写流程

参考代码如下：

```
'''
实验名称：定时器
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过定时器让 LED 周期性每秒闪烁 1 次
'''
from machine import Pin,Timer
```

```
led=Pin(2,Pin.OUT)
Counter = 0
Fun_Num = 0

def fun(tim):

    global Counter
    Counter = Counter + 1
    print(Counter)
    led.value(Counter%2)

#开启 RTOS 定时器，编号为-1
tim = Timer(-1)
tim.init(period=1000, mode=Timer.PERIODIC,callback=fun) #周期为 1000ms
```

- **实验结果：**

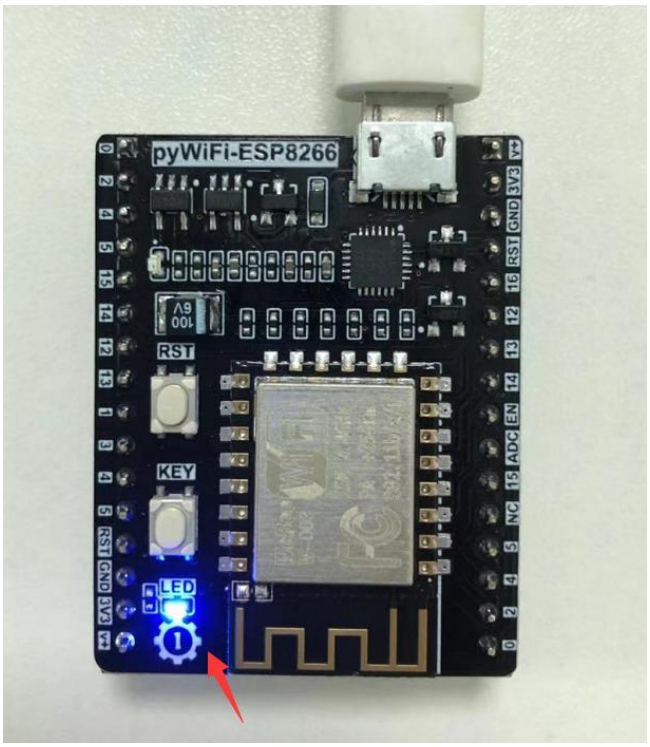


图 8-18　LED 以周期 1 秒的间隔闪烁

- **总结：**

本节实验介绍了 RTOS 定时器的使用方式，有用户可能会认为使用延时延时也可以实现这个功能，但相比于延时函数，定时器的好处就是不占用 CPU 资源。RTOS（实时操作系统）的原理就是利用多个定时器分别执行不同任务方式实现。

有兴趣的用户也可以多定义几个定时器对象 tim2,tim3，定时器标号还是-1，通过不同的周期和参数配置实现多任务的操作。

8.5 I2C 总线（OLED 显示屏）

- **前言：**

前面学习了按键输入设备后，这一节我们来学习输出设备 OLED 显示屏，其实之前的 LED 灯也算是输出设备，因为它们确切地告诉了我们硬件的状态。只是相对于只有亮灭的 LED 而言，显示屏可以显示更多的信息，体验更好。

本章节的OLED显示屏学习，实际上是在使用I2C的总线接口，pyWiFi-ESP8266是通过 I2C 总线与 OLED 显示屏通讯的。这章稍微复杂一点，但我们把它放在了前面来学习，是因为学会了显示屏的使用，那么在后面的实验中可玩性就更强了。

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。

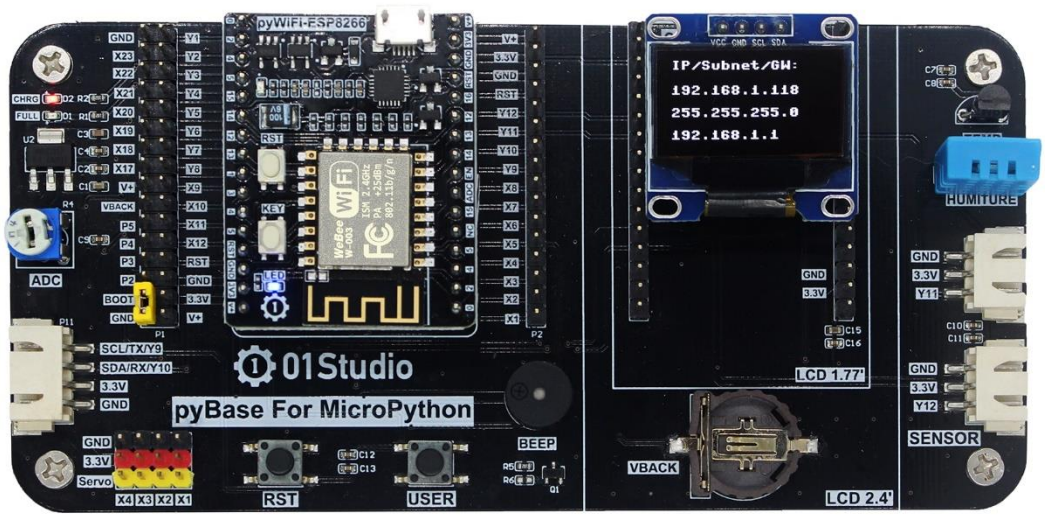


图 8-19 pyWiFi-ESP8266 开发套件

- **实验目的：**

学习使用 MicroPython 的 I2C 总线通讯编程和 OLED 显示屏的使用。

- **实验讲解：**

什么是 I2C?

I2C 是用于设备之间通信的双线协议，在物理层面，它由 2 条线组成：SCL 和 SDA，分别是时钟线和数据线。也就是说不通设备间通过这两根线就可以进行通信。

什么是 OLED 显示屏？

OLED 的特性是自己发光，不像 TFT LCD 需要背光，因此可视度和亮度均高，其次是电压需求低且省电效率高，加上反应快、重量轻、厚度薄，构造简单，成本低等特点。简单来说跟传统液晶的区别就是里面像素的材料是由一个个发光二极管组成，因为密度不高导致像素分辨率低，所以早期一般用作户外 LED 广告牌。随着技术的成熟，使得集成度越来越高。小屏也可以制作出较高的分辨率。



图 8-20　I2C 接口的 OLED

在了解完 I2C 和 OLED 显示屏后，我们先来看看 pyBase 开发板的原理图，也就是上面的 OLED 接口是如何连线的。

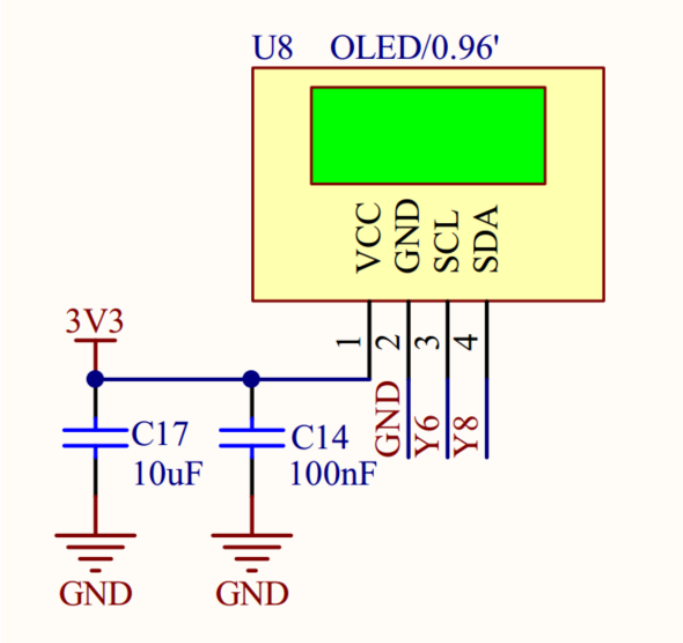


图 8-21　OLED 接口原理图

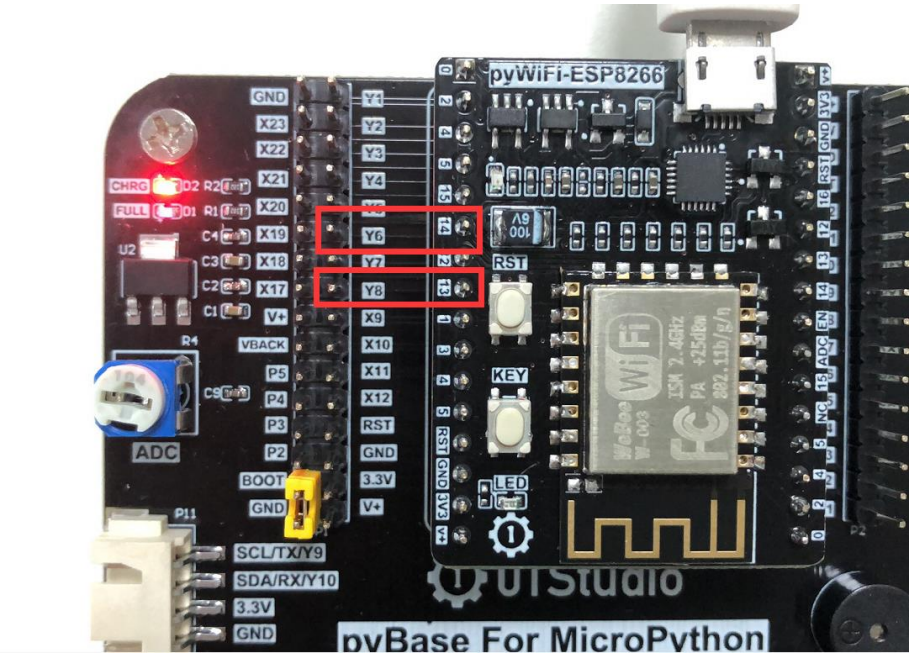


图 8-22　pyWiFi-ESP8266 和 pyBase 的连线图

从上图可见连接到 OLED 的别是 Y6→SCL 和 Y8→SDA。跟 pyWiFi-ESP8266 的 14、13 引脚相连。本实验将使用 MicroPython 的 Machine 模块来定义 Pin 口和 I2C 初始化。具体如下：

构造函数
i2c = machine.I2C(scl,sda)
构建 I2C 对象。scl:时钟引脚；sda:数据引脚。
使用方法
i2c.scan()
扫描 I2C 总线的设备。返回地址，如：0x3c；
i2c.readfrom(addr,nbytes)
从指定地址读数据。addr:指定设备地址；nbytes:读取字节数；
i2c.write(buf)
写数据。buf:数据内容；
*其它更多用法请阅读 MicroPython 文档： 中文文档链接：http://docs.micropython.01studio.org/

表 8-5　I2C 对象

定义好 I2C 后，还需要驱动一下 OLED。这里我们已经写好了 OLED 的库函数，在 ssd1306.py 文件里面。开发者只需要拷贝到 pyBoard 文件系统里面，然后在 main.py 里面调用函数即可。人生苦短，我们学会调用函数即可，也就是注重顶层的应用，想深入的小伙伴也可以自行研究 ssd1306.py 文件代码。OLED 显示屏对象介绍如下：

构造函数
oled = SSD1306_I2C(width, height, i2c, addr)
构 OLED 显示屏对象。width:屏幕宽像素；height: 屏幕高像素；i2c:定义好的 I2C 对象; addr:显示屏设备地址。
使用方法
`oled.text(string,x,y)`
将 string 字符写在指定为位置。string：字符；x:横坐标；y:纵坐标。
`oled.show()`
执行显示。
`oled.fill(RGB)`
清屏。RGB：0 表示黑色，1 表示白色。

图 8-23 OLED 对象

学习了 I2C、OLED 对象用法后我们通过编程流程图来理顺一下思路：

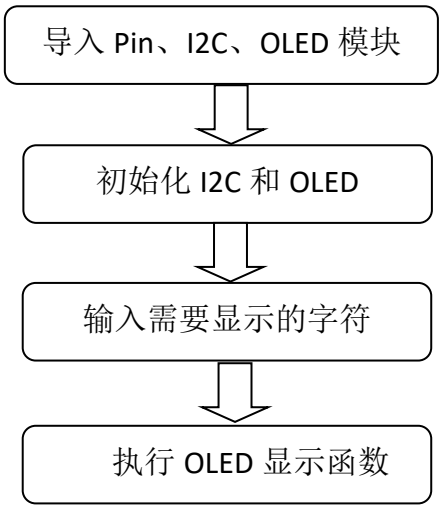


图 8-24 代码编写流程

main.py 文件参考代码如下：

```
'''
实验名称：I2C 总线(OLED 显示屏)
版本：v1.0
日期：2019.7
作者：01Studio
'''

from machine import I2C,Pin       #从 machine 模块导入 I2C、Pin 子模块
from ssd1306 import SSD1306_I2C   #从 ssd1306 模块中导入 SSD1306_I2C 子模块

i2c = I2C(sda=Pin(13), scl=Pin(14))  #I2C 初始化：sda--> 13, scl --> 14

#OLED 显示屏初始化：128*64 分辨率,OLED 的 I2C 地址是 0x3c
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

oled.text("Hello World!", 0,  0)      #写入第 1 行内容
oled.text("MicroPython",  0, 20)      #写入第 2 行内容
oled.text("By 01Studio",  0, 50)      #写入第 3 行内容

oled.show()   #OLED 执行显示
```

上述代码中 OLED 的 I2C 地址是 0x3C,不同厂家的产品地址可能预设不一样，具体参考厂家的说明书。或者也可以通过 I2C.scan()来获取设备地址。

另外记得将我们提供的示例代码中的 ssd1306.py 驱动文件拷贝到 pyWiFi-ESP8266 的文件系统下，跟 main.py 保持同一个路径。

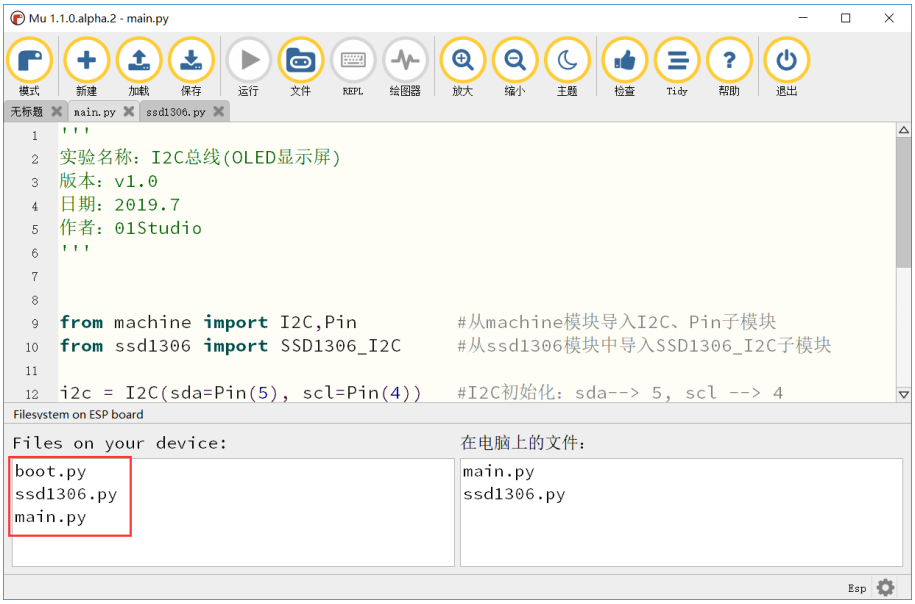


图 8-25 将 main.py、ssd1306.py 拷贝到设备

- **实验结果：**

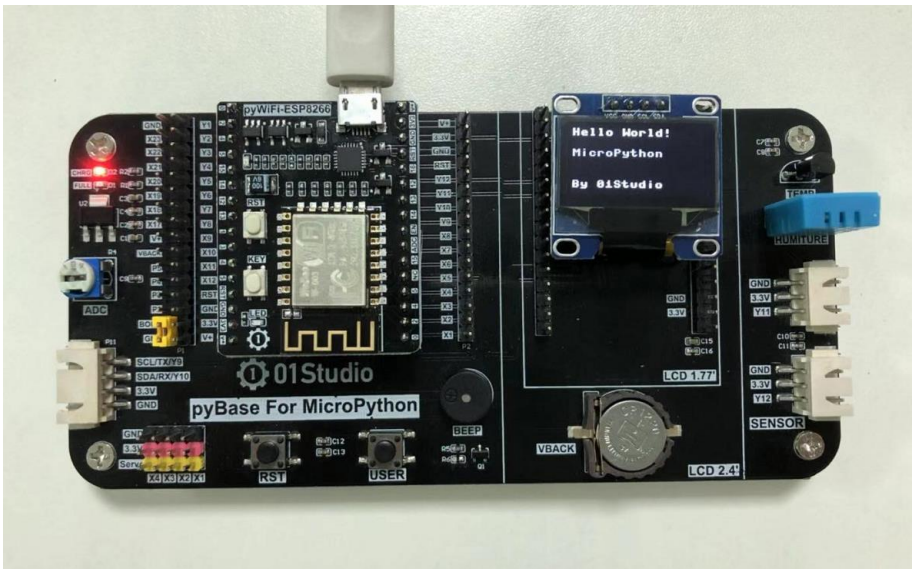


图 8-26 OLED 实验结果

- **总结：**

这一节我们学会了驱动 OLED 显示屏，换着以往如果从使用单片机从 0 开发的话你需要了解 I2C 总线原理，了解 OLED 显示屏的使用手册，编程 I2C 代码，有经验的嵌入式工程师搞不好也要弄个几天。现在基本半个小时解决问题。当然前提是别人已经给你搭好桥了，有了强大的底层驱动代码支持，我们只做好应用

就好。

这一节学习的意义不仅在完成实验。在学习完 OLED 显示屏实验后，接下来我们的实验都可以使用这个 OLED 来跟用户交互了，这大大提高了实验的可观性。

8.6 RTC 实时时钟

- **前言：**

时钟可以说我们日常最常用的东西了，手表、电脑、手机等等无时无刻不显示当前的时间。可以说每一个电子爱好者心中都希望拥有属于自己制作的一个电子时钟，接下来我们就用 MicroPython 开发板来制作一个属于自己的电子时钟。

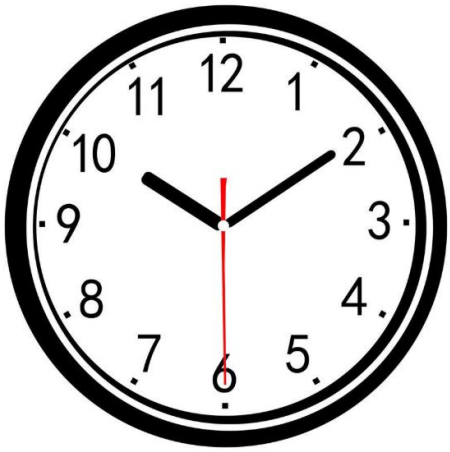


图 8-27 时钟

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。

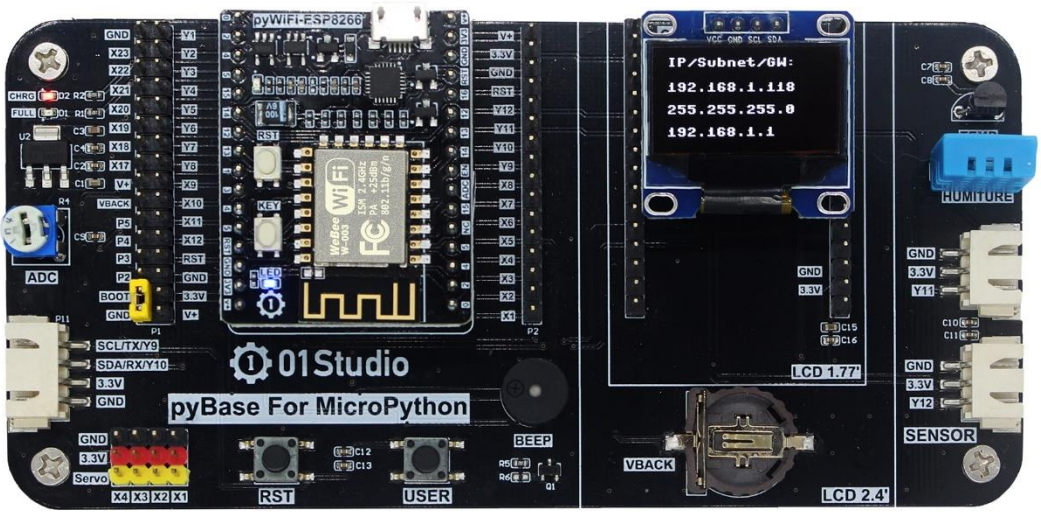


图 8-28 pyWiFi-ESP8266 开发套件

- **实验目的：**

  学习 RTC 编程和制作电子时钟，使用 OLED 显示。

- **实验讲解：**

实验的原理是读取 RTC 数据，然后通过 OLED 显示。毫无疑问，强大的 MicroPython 已经集成了内置时钟函数模块。位于 machine 的 RTC 模块中，具体介绍如下：

构造函数
rtc=machine.RTC()
构建 RTC 对象。
使用方法
`rtc.datetime((2019, 4, 1, 0, 0, 0, 0, 0))`
设置日期和时间。按顺序分别是：（年，月，日，星期，时，分，秒，微秒），其中星期使用 0-6 表示周一至周日。
`rtc.datetime()`
获取当前日期和时间。

表 8-6

从上表可以看到 RTC（）的使用方法，我们需要做的就是先设定时间，然后再获取当前芯片里的时间，通过 OLED 显示屏显示，如此循环。在循环里，如果一直获取日期时间数据会造成资源浪费，所以可以每隔第一段时间获取一次数据，又由于肉眼需要看到至少每秒刷新一次即可，这里每隔 300ms 获取一次数据，使用前面学习过的 RTOS 定时器来计时，具体编程流程如下：

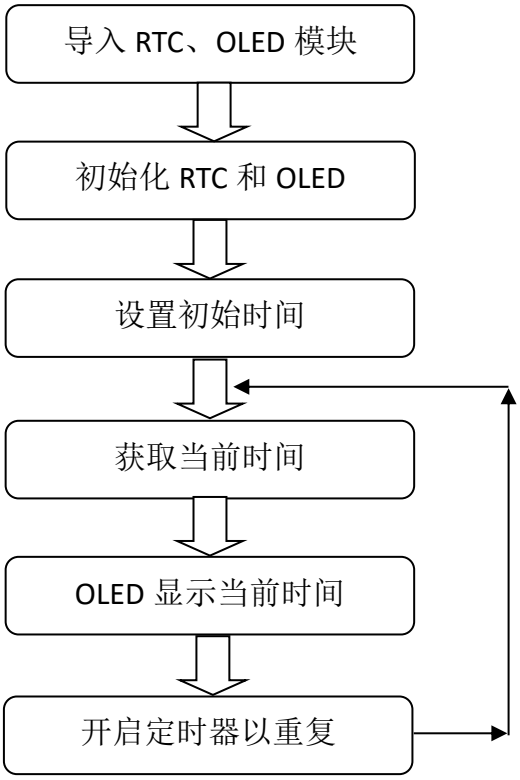


图 8-29 代码编写流程图

实验参考代码如下：

```
'''
实验名称：RTC 实时时钟
版本：v1.0
日期：2019.7
作者：01Studio
'''

# 导入相关模块
from machine import Pin, I2C, RTC,Timer
from ssd1306 import SSD1306_I2C

# 定义星期和时间（时分秒）显示字符列表
week = ['Mon', 'Tues', 'Wed', 'Thur', 'Fri', 'Sat', 'Sun']
```

```
time_list = ['', '', '']

# 初始化所有相关对象
i2c = I2C(sda=Pin(13), scl=Pin(14)) #I2C 初始化：sda-->13, scl --> 14
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
rtc = RTC()

# 首次上电配置时间，按顺序分别是：年，月，日，星期，时，分，秒，次秒级；这里做
#了一个简单的判断，检查到当前年份不对就修改当前时间，开发者可以根据自己实际情况
#来修改。
if rtc.datetime()[0] != 2019:
    rtc.datetime((2019, 4, 1, 0, 0, 0, 0, 0))

def RTC_Run(tim):
    datetime = rtc.datetime()  # 获取当前时间

    oled.fill(0)  # 清屏显示黑色背景
    oled.text('01Studio', 0, 0)    # 首行显示 01Studio
    oled.text('RTC Clock', 0, 15)  # 次行显示实验名称

    # 显示日期，字符串可以直接用“+”来连接
    oled.text(str(datetime[0]) + '-' + str(datetime[1]) + '-' +
str(datetime[2]) + ' ' + week[datetime[3]], 0, 40)

# 显示时间需要判断时、分、秒的值否小于 10，如果小于 10，则在显示前面补“0”以
# 达到较佳的显示效果
    for i in range(4, 7):
        if datetime[i] < 10:
            time_list[i - 4] = "0"
        else:
```

```
            time_list[i - 4] = ""


    # 显示时间
    oled.text(time_list[0] + str(datetime[4]) + ':' + time_list[1] +
str(datetime[5]) + ':' + time_list[2] + str(datetime[6]), 0, 55)


    oled.show()


#开启 RTOS 定时器
tim = Timer(-1)
tim.init(period=300, mode=Timer.PERIODIC, callback=RTC_Run) #周期 300ms
```

由于实验要用到 OLED 显示屏，所以同样别忘了将示例代码该实验文件夹下的 ssd1306.py 文件复制到 pyWiFi-ESP8266 的文件系统里面。

- **实验结果：**

烧录程序复位后，可以看到时钟开始跑起来。

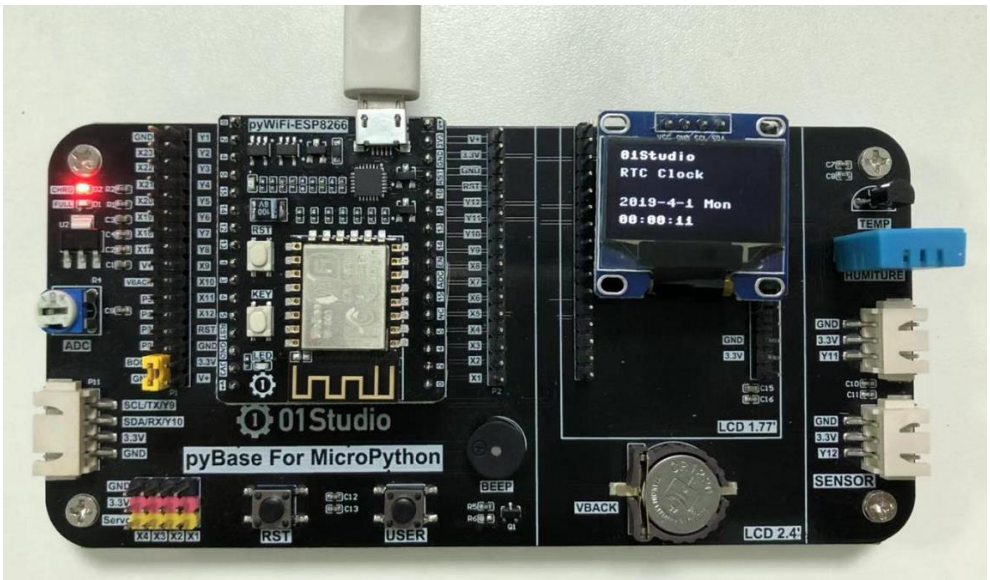


图 8-30

由于 ESP8266 没有后备电池引脚，所以不支持掉电保存。因此 pybase 上面的纽扣电池是不起作用的。

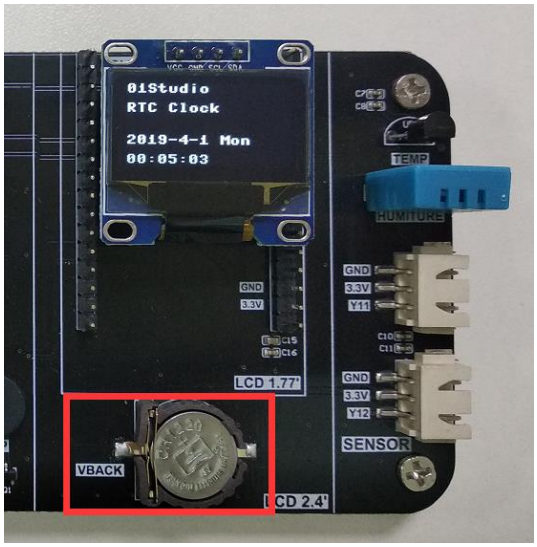


图 8-31 后备电池（VBACK）

- **总结：**

RTC 实时时钟的可玩性很强，我们还可以根据自己的风格来设定数字显示位置，以及加上一些属于自己的字符标识。打造自己的电子时钟。

8.7 ADC

- **前言：**

ADC(analog to digital conversion) 模拟数字转换。意思就是将模拟信号转化成数字信号，由于单片机只能识别二级制数字，所以外界模拟信号常常会通过 ADC 转换成其可以识别的数字信息。常见的应用就是将变化的电压转成数字信号。

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。

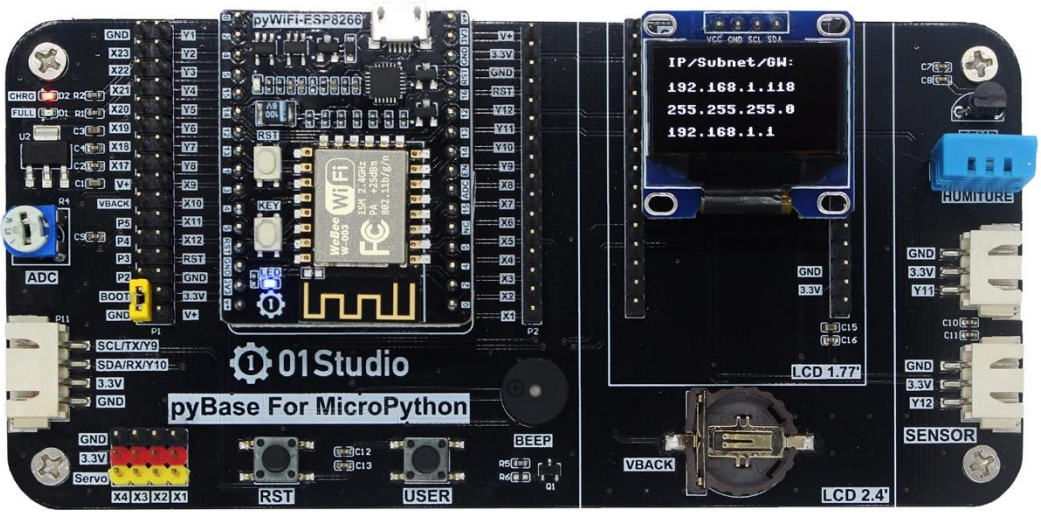


图 8-32 pyWiFi-ESP8266 开发套件

- **实验目的：**

通过编程调用 MicroPython 的内置 ADC 函数，实现测量 0-1V 电压，并显示到屏幕上。

- **实验讲解：**

pyBase 开发底板的 X7 引脚连接到了电位器，通过电位器的调节可以使得 X7 引脚上的电压变化范围实现从 0-3.3V。

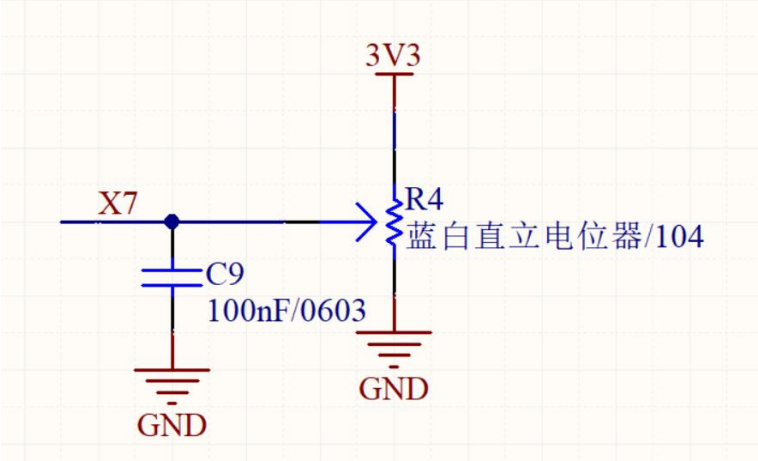


图 8-33 电位器原理图

但由于 ESP8266 的 ADC 输入引脚只能测量 0-1V 的量程，因此本实验的测量电压范围为 0-1V。我们来看看 ADC 模块的构造函数和使用方法。

构造函数
adc=machine.ADC(0)
构建 ADC 对象。0:ESP8266 只有 1 个 ADC
使用方法
adc.read()
获取 ADC 值。测量精度是 10 位，返回 0-1024（表示 0-1V）。

表 8-7 ADC 对象

你没看错，就这么简单。两句函数就可以获得 ADC 数值，让我们来理顺一下编程逻辑。先导入相关模块，然后初始化模块。在循环中不断读取 ADC 的值，转化成电压值后在 OLED 上面显示，每隔 300 毫秒读取一次，具体如下：

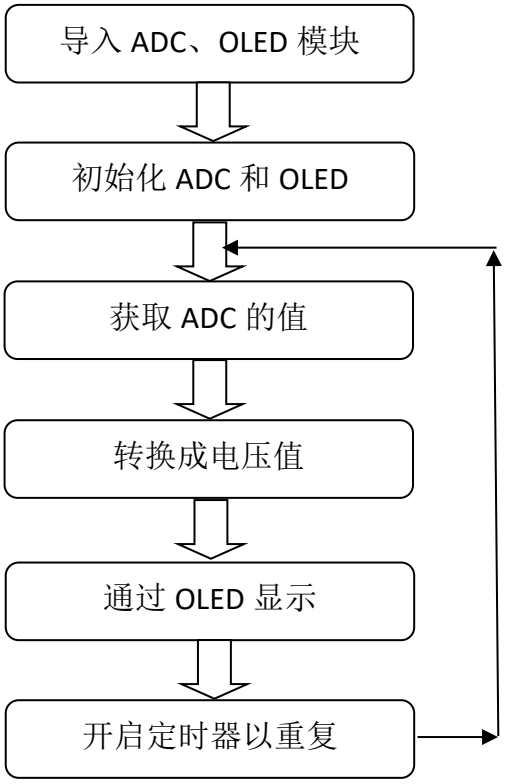


图 8-34 代码编写流程

实验参考代码：

```
'''
实验名称：ADC-电压测量
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过对 ADC 数据采集，转化成电压在显示屏上显示。ADC 精度 10 位，电压 0-1V。
'''

#导入相关模块
from machine import Pin,I2C,ADC,Timer
from ssd1306 import SSD1306_I2C
```

```
#初始化相关模块
i2c = I2C(sda=Pin(13), scl=Pin(14))  #I2C 初始化：sda--> 13, scl --> 14
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
adc = ADC(0) #Pin='AD'

def ADC_Test(tim):

    oled.fill(0)  # 清屏显示黑色背景
    oled.text('01Studio', 0, 0)  # 首行显示 01Studio
    oled.text('ADC', 0, 15)      # 次行显示实验名称

    #获取 ADC 数值
    oled.text(str(adc.read()),0,40)
    oled.text('(1024)',40,40)

    #计算电压值，获得的数据 0-1024 相当于 0-1V，（'%.2f'%）表示保留 2 位小数
    oled.text(str('%.2f'%(adc.read()/1024)),0,55)
    oled.text('V',40,55)

    oled.show()

#开启 RTOS 定时器
tim = Timer(-1)

#周期 300ms
tim.init(period=300, mode=Timer.PERIODIC, callback=ADC_Test)
```

- **实验结果：**

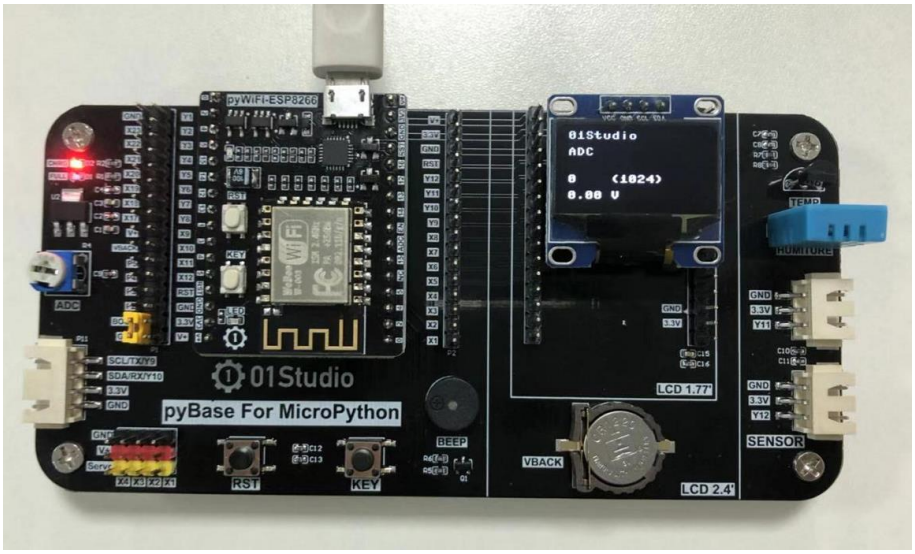


图 8-35 电位器顺时钟拧到尽头是 0V

通过调节电位器，可以发现电压在不断变化。由于 ESP8266 电压测量量程限制，当超出 1V 量程后，始终显示电压在 1V，测量值是 1024。

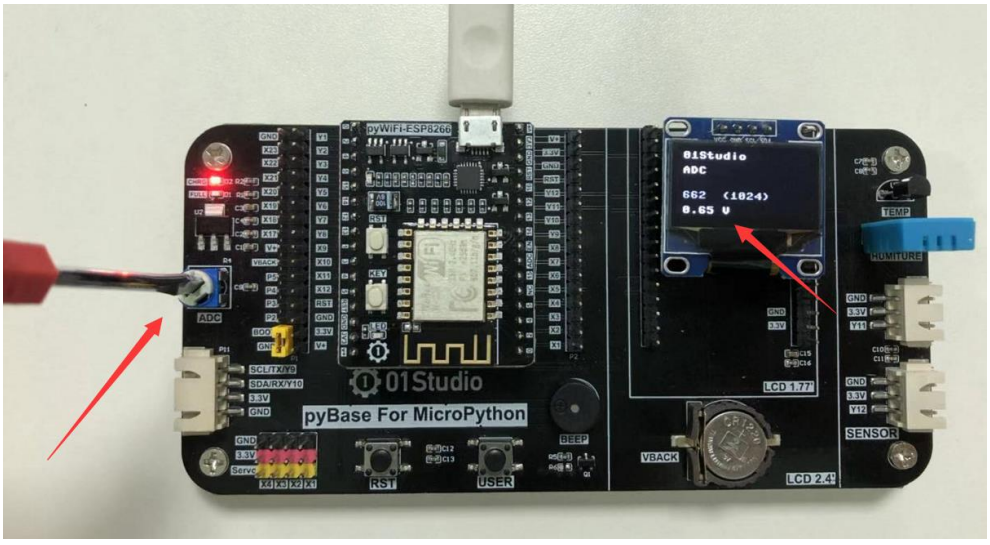


图 8-36 调节电位器来改变电压输入

- **总结：**

这一节我们学习了 ADC 的应用，主要用于电压的测量。

8.8 PWM

- **前言：**

上一节的 ADC 是信号输入，这节的 PWM 就是一个信号输出。PWM（脉冲宽度调制），主要用于输出不同频率、占空比（一个周期内高电平出现时间占总时间比例）的方波。以实现固定频率或平均电压输出。

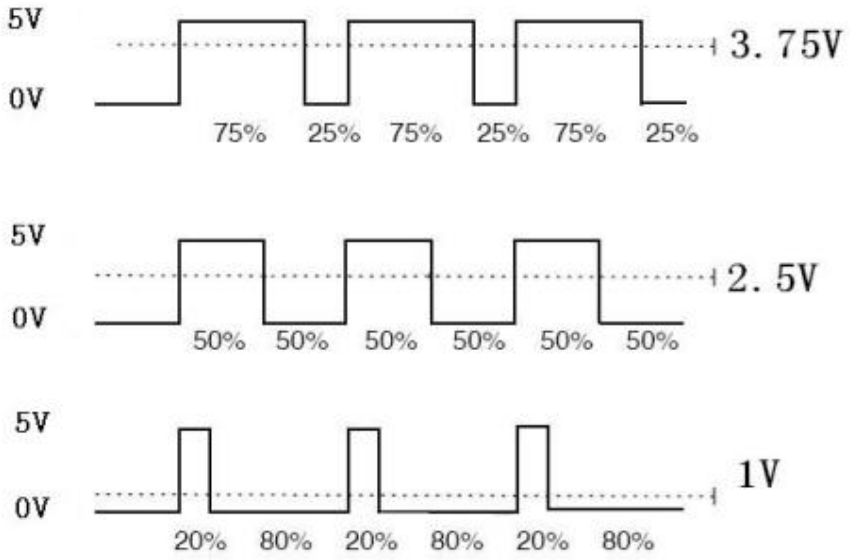


图 8-37 PWM 波形示例

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。

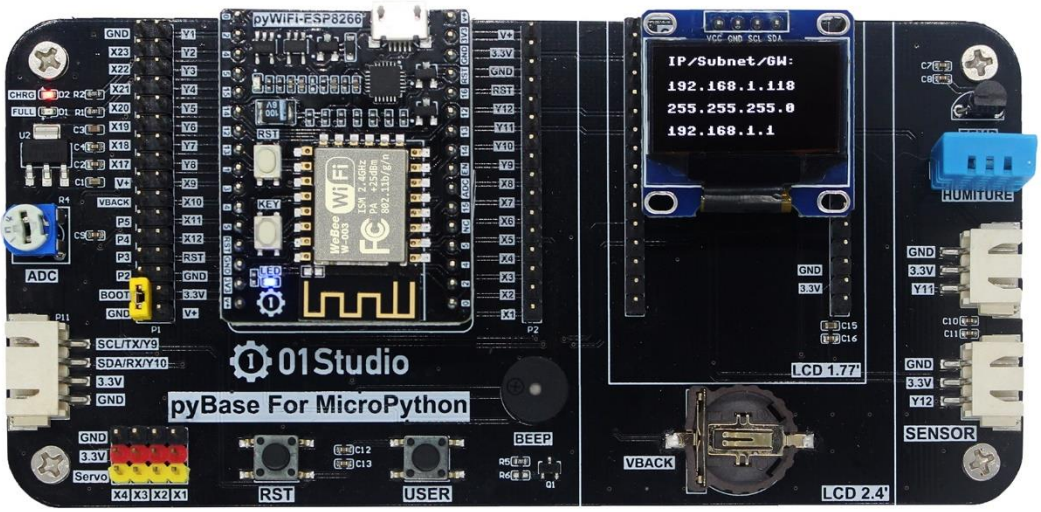


图 8-38 pyWiFi-ESP8266 开发套件

- **实验目的：**

通过不同频率的 PWM 信号输出，驱动无源蜂鸣器发出不同频率的声音。

- **实验讲解：**

蜂鸣器分有源蜂鸣器和无源蜂鸣器，有源蜂鸣器的使用方式非常简单，只需要接上电源，蜂鸣器就发声，断开电源就停止发声。而本实验用到的无源蜂鸣器，是需要给定指定的频率，才能发声的，而且可以通过改变频率来改变蜂鸣器的发声音色，以此来判定 pyWiFi-ESP8266 的 PWM 输出频率是在变化的。

pyBase 开发底板上的无源蜂鸣器连接到引脚 X5。如下图所示：

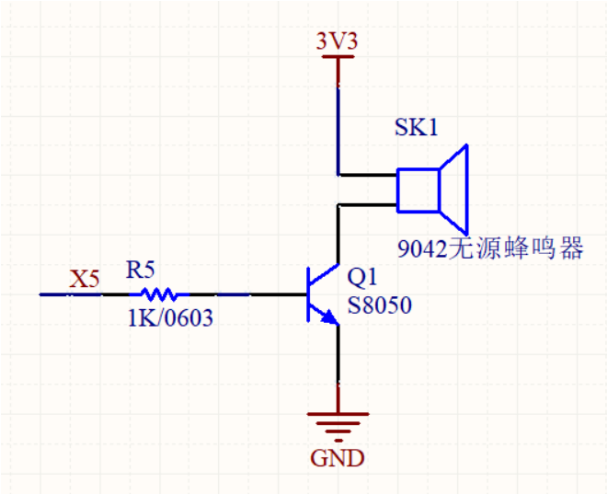


图 8-39 蜂鸣器原理图

PWM 可以通过 ESP8266 所有引脚输出除了 Pin(16). 所有通道都有 1 个特定的频率，从 1 到 1000 之间（单位是 Hz）。占空比的值为 0 至 1023 之间。在本实验中我们用到引脚 15。

先看看 PWM 模块对象：

构造函数
pwm15=machine.PWM(machine.Pin(id),freq,duty)
构建 PWM 对象。id:引脚编号；freq:频率值；duty:占空比；配置完后 PWM 自动生效。
使用方法
pwm15.freq(freq)
设置频率。freq:频率值在 1-1000 之间，freq 为空时表示获取当前频率值。

pwm15.duty(duty)
设置占空比。duty:占空比在 0-1023 之间，duty 为空时表示获取当前占空比值。
pwm15.deinit()
关闭 PWM。

表 8-8　PWM 对象

无源蜂鸣器我们可以用特定频率的方波来驱动，方波的原理很简单，就是一定频率的高低电平转换，可以简单理解成占空比为 50%的 PWM 输出。

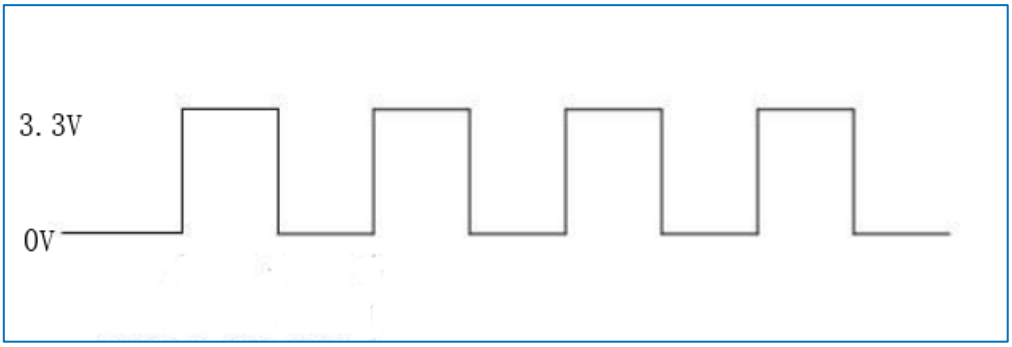


图 8-40　方波信号

结合上述讲解，总结出代码编写流程图如下：

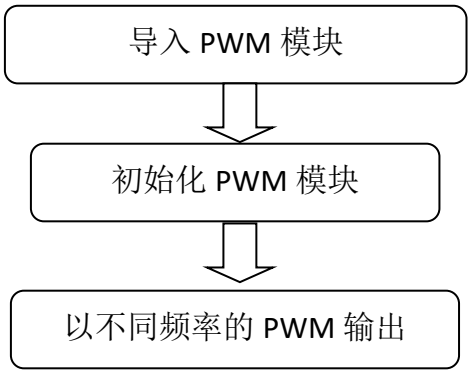


图 8-41 代码编写流程图

实验参考代码：

```
'''
实验名称：PWM
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过不同频率的 PWM 信号输出，驱动无源蜂鸣器发出不同频率的声音。
'''

from machine import Pin, PWM
import time

Beep = PWM(Pin(15), freq=0, duty=512) # 在同一语句下创建和配置 PWM

#蜂鸣器发出频率 200Hz 响声
Beep.freq(200)
time.sleep_ms(1000)

#蜂鸣器发出频率 400Hz 响声
Beep.freq(400)
time.sleep_ms(1000)

#蜂鸣器发出频率 600Hz 响声
Beep.freq(600)
time.sleep_ms(1000)



#蜂鸣器发出频率 800Hz 响声
Beep.freq(800)
time.sleep_ms(1000)
```

```
#蜂鸣器发出频率 1000Hz 响声
Beep.freq(1000)
time.sleep_ms(1000)

#停止
Beep.deinit()
```

- **实验结果：**

由于 pyBase 上的 X5 对应 pyWiFi-ESP8266 的引脚是 NC 悬空引脚。X6 对应 15 引脚。因此我们可以用一个跳线帽将 pybase 开发底板的 X5、X6 引脚短接。以实现引脚 15 连接到无源蜂鸣器。

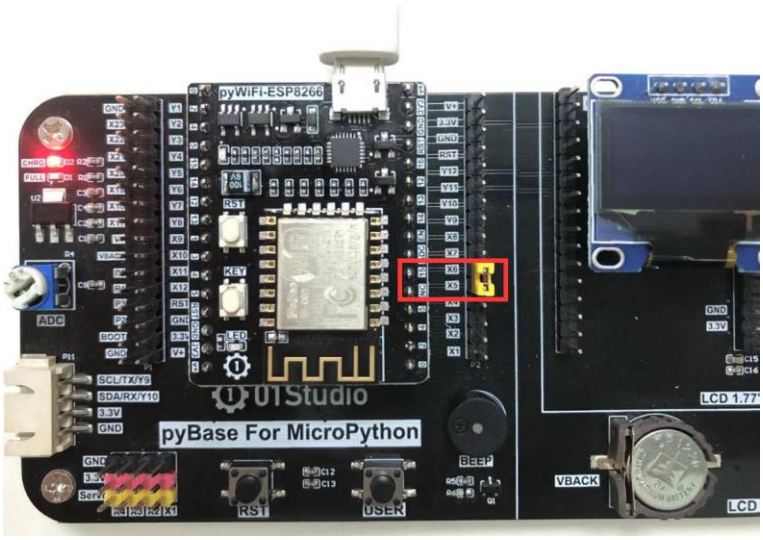


图 8-42 用跳线帽将 pyBase 的 X5、X6 短接

下载程序，复位后可以听到蜂鸣器依次发出不同频率的响声。

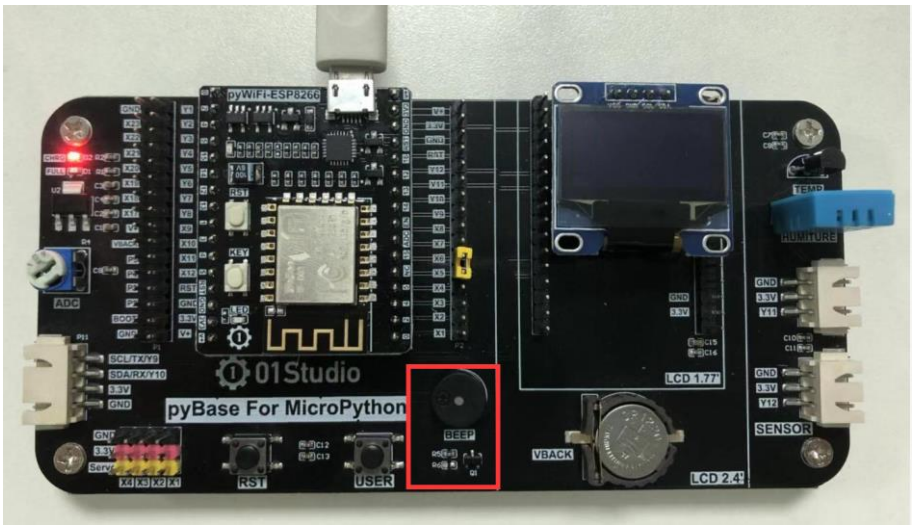


图 8-43 实验现象

有条件的朋友可以使用示波器测量 pyWiFi-ESP8266 的 15 引脚或 pyBase 的 X5 接口，观察信号波形的变化：

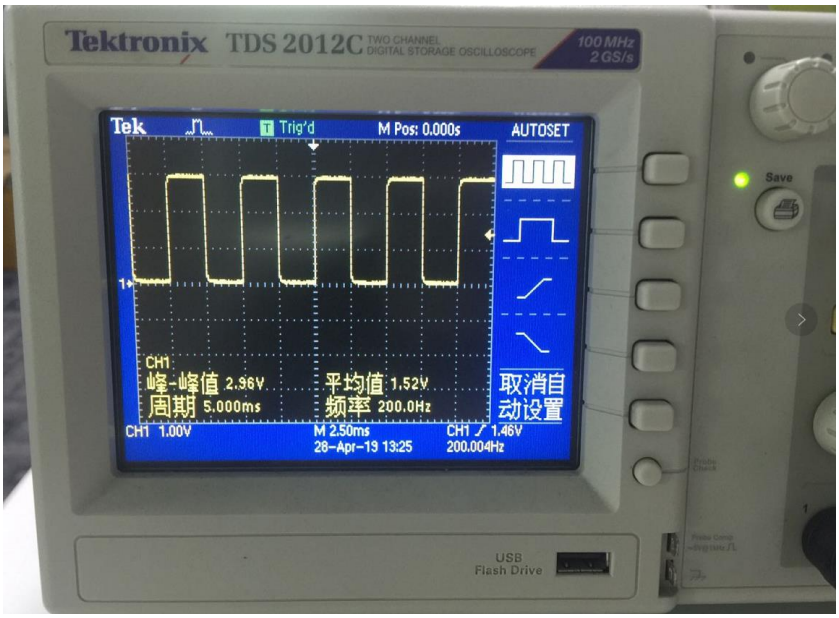


图 8-44 示波器显示波形

- **总结：**

到了这一节，我们发现实验中对象函数使用方法非常简单，这是好事，让我们可以将更多精力放在应用上，做出更多好玩的创意。而不需要过多的关注复杂的底层代码开发。而随着要实现功能的复杂化让编程的代码数量变多，逻辑也将略显复杂。

第9章　传感器实验

基础实验让我们对 MicroPython 的操作和编程有了比较全面的了解，但其魅力绝不限于此。日常生活中我们会用到各式各样的外设或者传感器，还是那句，一个有经验的嵌入式开发工程师驱动一款未接触过的传感器的一般流程是：了解传感器原理、设计电路图、信号时序分析和编程。没个几天折腾不出来。

生活中有很多传感器已经是非常通用了，前人已经做好封装函数模块，我们直接调用函数即可。我们不需要将时间花在“怎么用”上，而更多的是考虑“用到什么地方”!

本章实验以最常见的传感器出发，讲述是如何通过 MicroPyhon 编程实现传感器应用的。实验还是基于我们的 MicroPython 开发板。而且实验例程将持续更新。

9.1 温度传感器 DS18B20

- **前言：**

相信没有电子爱好者不知道 DS18B20 的，DS18B20 是常用的数字温度传感器，其输出的是数字信号，具有体积小，硬件开销低，抗干扰能力强，精度高的特点。DS18B20 数字温度传感器接线方便，封装成后可应用于多种场合，如管道式，螺纹式，磁铁吸附式，不锈钢封装式，型号多种多样。

主要根据应用场合的不同而改变其外观。封装后的 DS18B20 可用于电缆沟测温，高炉水循环测温，锅炉测温，机房测温，农业大棚测温，洁净室测温，弹药库测温等各种非极限温度场合。耐磨耐碰，体积小，使用方便，封装形式多样，适用于各种狭小空间设备数字测温和控制领域。

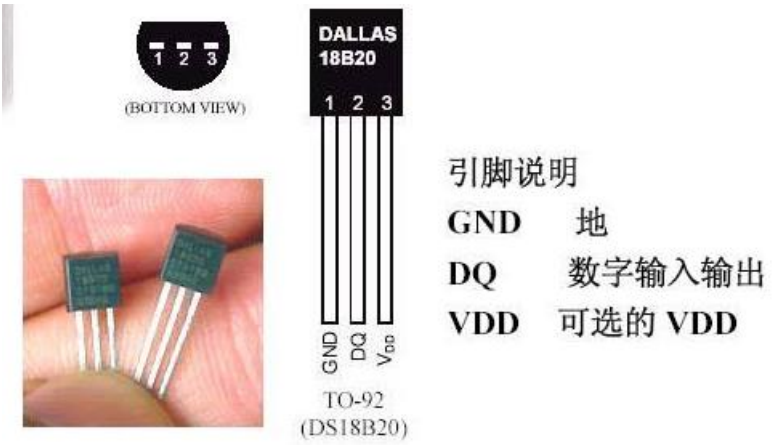


图 9-1 DS18B20 传感器



图 9-2 DS18B20 金属探头封装

- **实验平台：**

pyWiFi-ESP8266 开发套件。DS18B20 温度传感器位于 pyBase 右上方。

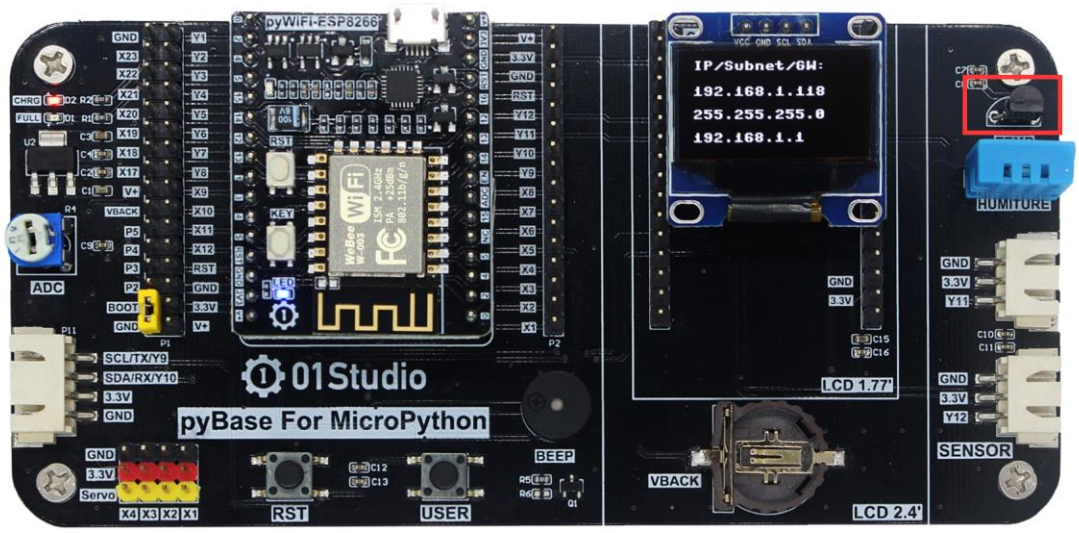


图 9-3　pyWiFi-ESP8266 开发套件

- **实验目的：**

通过编程采集温度数据，并在 OLED 上显示。

- **实验讲解：**

DS18B20 是单总线驱动（onewire）传感器，也就是说只占用 1 个 IO 口。我们来看看原理图：

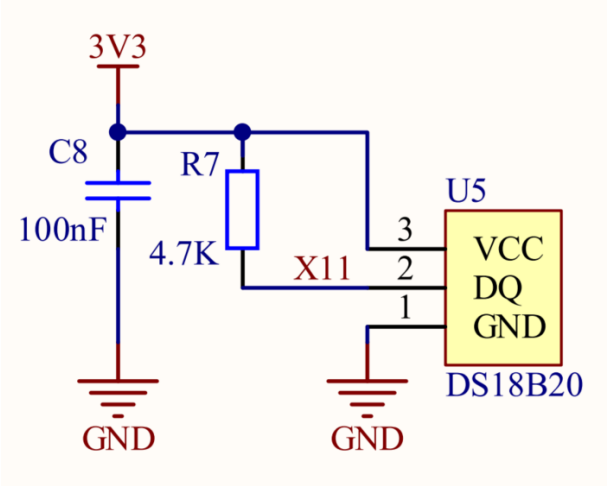


图 9-4 DS18B20 接线原理图

可以看到 DS18B20 传感器连接到了 pyBase 的 X11 引脚上。也就是说连接到 pyWiFi-ESP8266 的引脚 4 上。

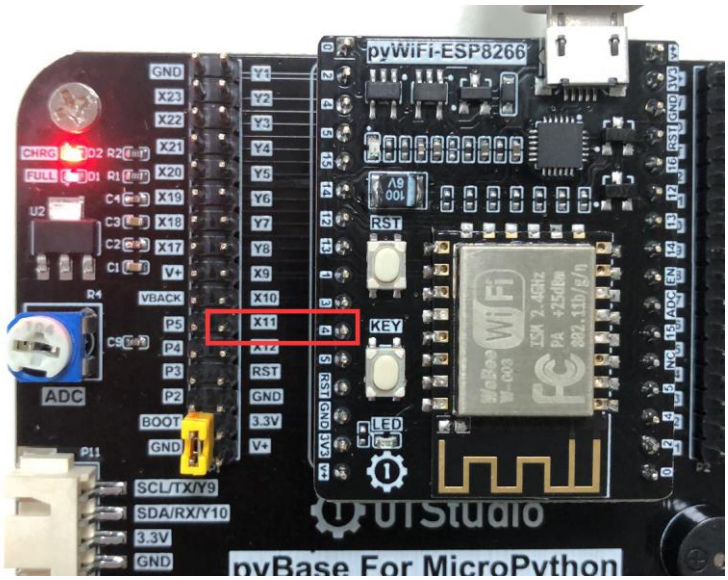


图 9-5 用跳线帽将 pyBase 的 X19、Y6 短接

也就是说我们需要针对引脚 4 编写程序来驱动 DS18B20。那么我们需要自己来编写驱动么？如果你有兴趣的可以自己尝试一下。这部分我们 01Studio 已经收集整理和编写好了，单总线模块文件是：onewire.py，DS18B20 模块的文件是 ds18x20.py。如果你学习过前面基于 STM32 平台应该不陌生。而对于 ESP8266,这两个模块已经集成到了初始化固件中，也就是说我们可以直接在 main.py 导入模块并调用即可！

单总线模块（onewire）说明如下：

构造函数
ow=onewire.OneWire(machine.Pin(id))
构建单总线对象。id:引脚编号；
使用方法
ow.scan()
扫描总线上的设备。返回设备地址，支持多设备同时挂载。
ow.reset()

总线设备复位。
ow.readbyte()
读 1 个字节。
ow.writebyte(0x12)
写入 1 个字节。
ow.write('123')
写入多个字节。
ow.select_rom(b'12345678')
根据 ROM 编号选择总线上指定设备。

表 9-1　单总线对象

ds18x20 模块说明如下：

构造函数
ds=ds18x20.DS18X20(ow)
构建 DS18B20 传感器对象。ow:定义好的单总线对象；
使用方法
ds.scan()
扫描总线上的设备。返回设备地址，支持多设备同时挂载。
ds.convert_temp()
温度转换。
ds.read_temp(rom)
获取温度值。rom：表示对应的设备号。

表 9-2　DS18B20 对象

大部分场景下温度的变化不会太频繁，我们可以每隔 1 秒采集一次，显示精度为小数点后 2 位，基本满足大部分应用需求。编程逻辑如下：

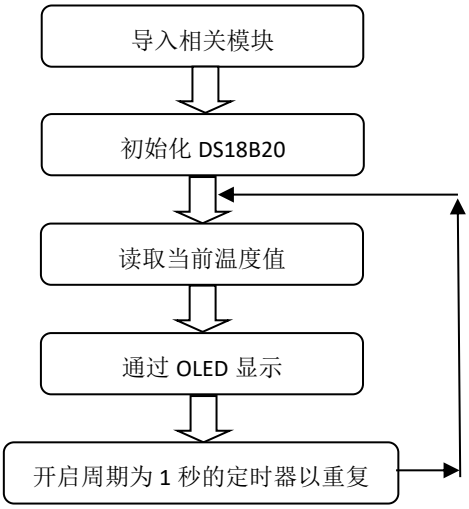


图 9-6 代码编写流程图

实验参考代码：

```
'''
实验名称：温度传感器 DS18B20
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过编程采集温度数据，并在 OLED 上显示。。
'''

#引用相关模块
from machine import Pin,I2C,Timer
from ssd1306 import SSD1306_I2C
import onewire,ds18x20

#初始化相关模块
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
```

```
#初始化 DS18B20
ow= onewire.OneWire(Pin(4))  #使能单总线
ds = ds18x20.DS18X20(ow)      #传感器是 DS18B20
rom = ds.scan()          #扫描单总线上的传感器地址，支持多个传感器同时连接

def temp_get(tim):
    ds.convert_temp()
    temp = ds.read_temp(rom[0]) #温度显示,rom[0]为第 1 个 DS18B20

    #OLED 数据显示
    oled.fill(0)   #清屏背景黑色
    oled.text('MicroPython', 0, 0)
    oled.text('Temp test:',0,20)
    oled.text(str('%.2f'%temp)+' C',0,40) #显示 temp,保留 2 位小数
    oled.show()

#开启 RTOS 定时器，编号为-1
tim = Timer(-1)
#定时器周期为 1000ms
tim.init(period=1000, mode=Timer.PERIODIC,callback=temp_get)
```

实验使用到 OLED 显示屏，记得同时将 ssd1306.py 文件拷贝到设备中。

- **实验结果：**

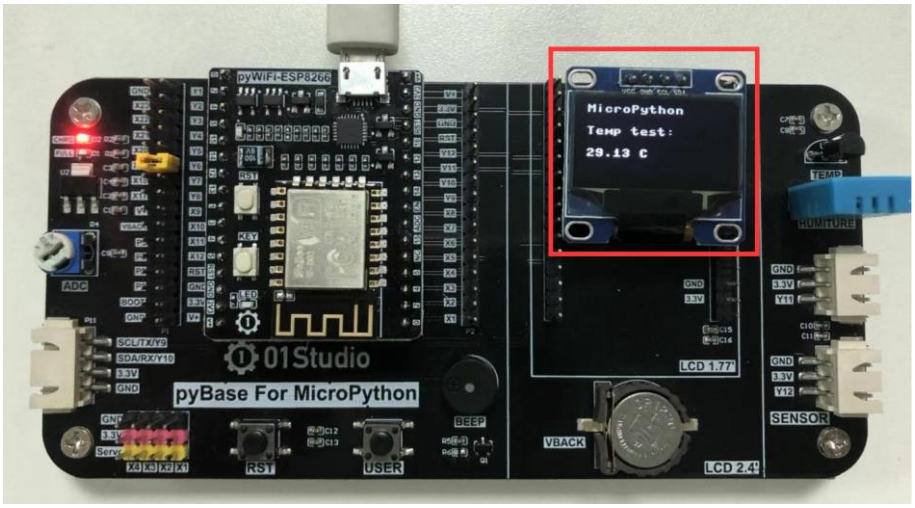


图 9-7 温度检测实验

- **实验拓展：**

pyBase 开发底板预留了外界传感器接口，只要接线正确就可以进行更多的传感器实验。我们将带金属探头的 DS18B20 传感器接到 pyBase 右侧上面的传感器母座，其连接到 pyBase 的“Y11”引脚,也就是对应 pyWiFi-ESP8266 的引脚 12。所以只要将原程序代码的 ow= onewire.OneWire(Pin(4)) 改成 ow= onewire.OneWire(Pin(12))，即可驱动外接的 DS18B20。

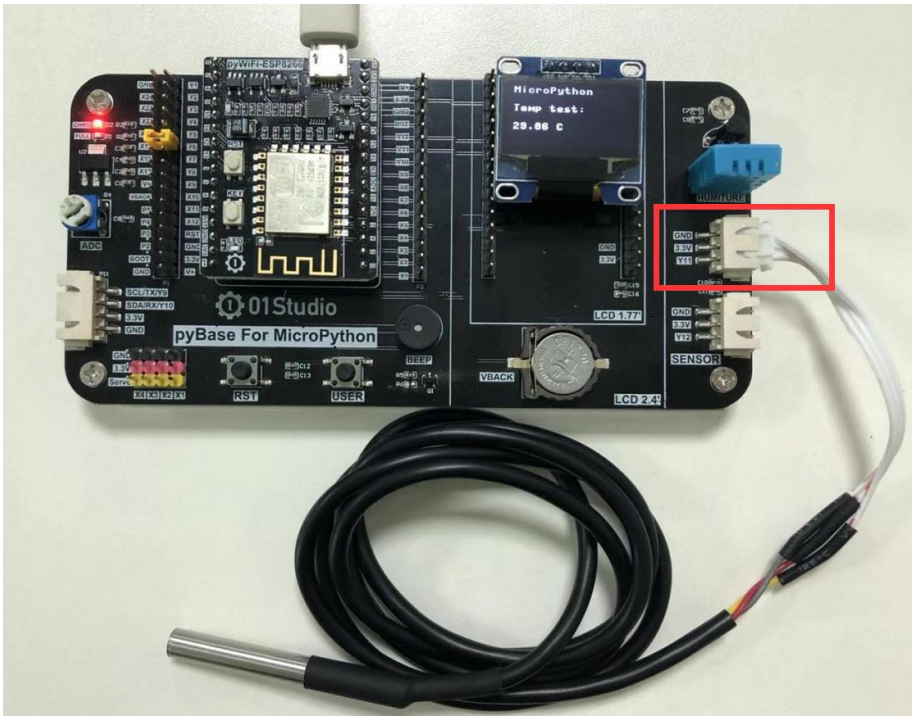


图 9-8 外接温度传感器

我们将金属探头放在水里面，可以见到测试到的温度。

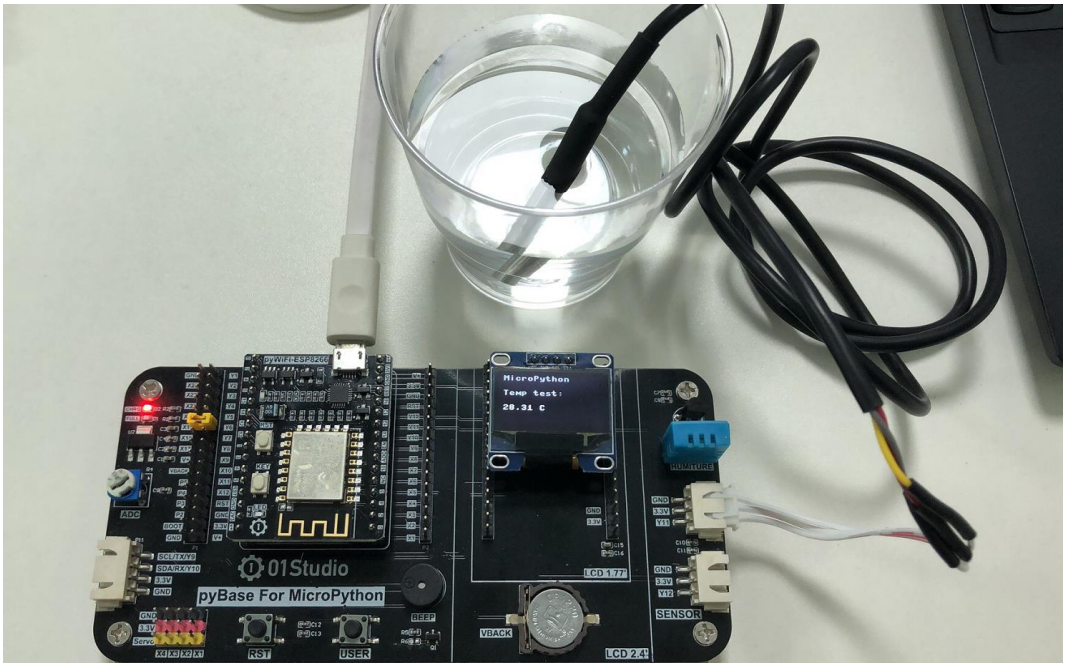


图 9-9 实现金属探头 DS18B20 测量

- **总结**

DS18B20 作为我们第一个实验传感器，使用 MicroPython 编程非常容易就用起来了，而且精度和稳定性丝毫没有影响。温度传感器只是一个敲门砖，接下来我们将会学习更多的传感器应用。

9.2 温湿度传感器 DHT11

- **前言：**

温湿度也是我们日常非常常见的指标，我们使用的是 DHT11 数字温湿度传感器。这是一款含有已校准数字信号输出的温湿度复合传感器，它应用专用的数字模块采集技术和温湿度传感技术，确保产品具有极高的可靠性和卓越的长期稳定性。

DHT11 具有小体积、极低的功耗，信号传输距离可达 20 米以上，使其成为给类应用甚至最为苛刻的应用场合的最佳选择。产品为 4 针单排引脚封装，连接方便。

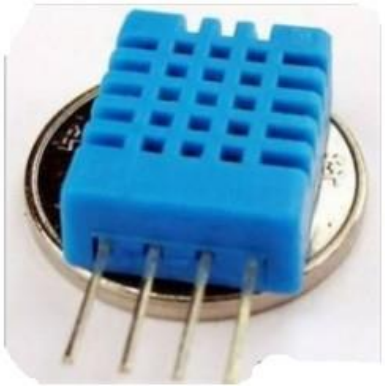


图 9-10 DHT11 温湿度传感器

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。DHT11 温湿度传感器位于右上方。

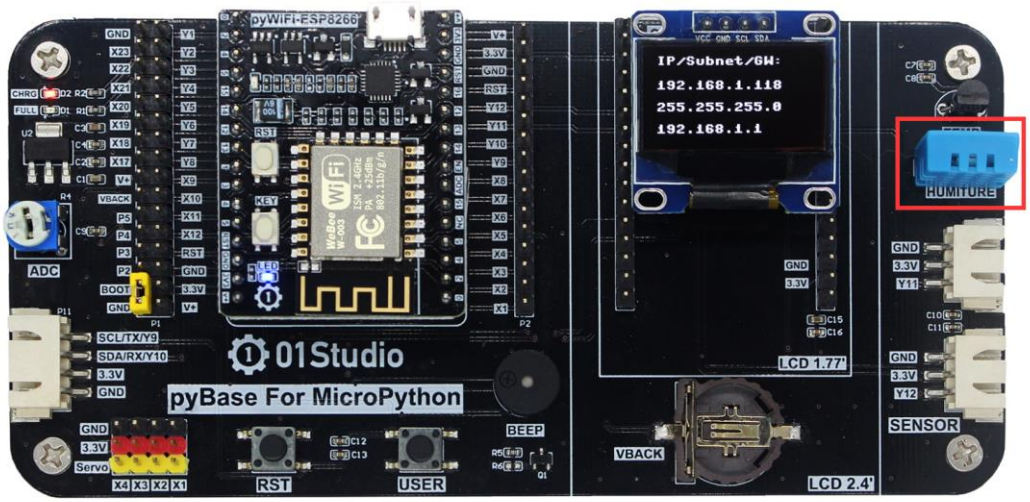


图 9-11 温湿度传感器 DHT11 位于右上方

- **实验目的：**

通过编程采集温湿度数据，并在 OLED 上显示。

- **实验讲解：**

DHT11 虽然有 4 个引脚，但其中第 3 个引脚是悬空的，也就是说 DHT11 也是单总线的传感器，只占用 1 个 IO 口。

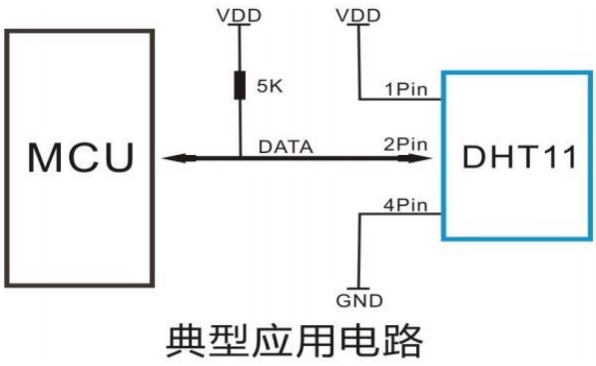


图 9-12　DHT11 应用电路

我们来看看 DHT11 在开发板上的接线图：

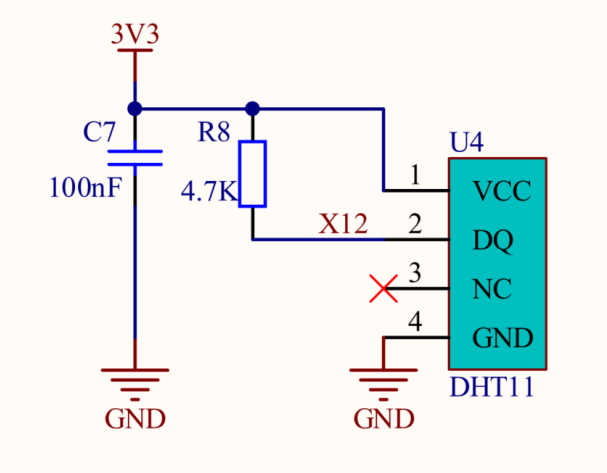


图 9-13　DHT11 接线图

可以看到 DHT11 连接到 pyBase 的‘X12’引脚，也就是连接到 pyWiFi-ESP8266 的引脚 5，如下图：

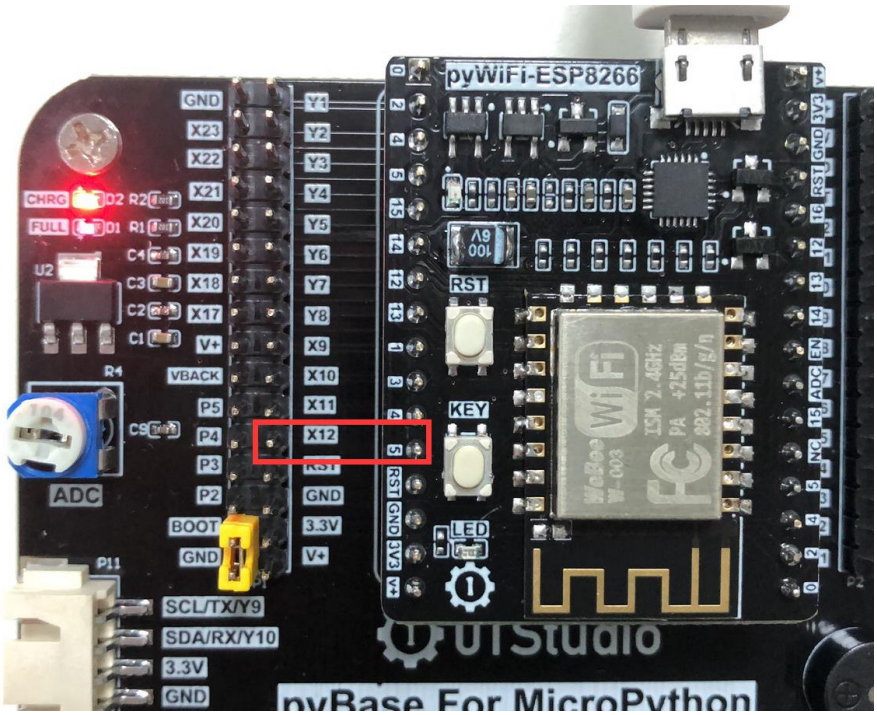


图 9-14 用跳线将 pyBase 的 X20、Y6 短接

也就是针对引脚 5 编程来驱动 DHT11 传感器，模块文件是 dht.py，如果你学习过前面基于 STM32 平台应该不陌生。而对于 ESP8266,这个模块已经集成到了初始化固件中，也就是说我们可以直接在 main.py 导入模块并调用即可。函数模块说明如下：

构造函数
d = dht.DHT11(machine.Pin(id))
构建 DHT11 传感器对象。id:传感器所连接的引脚；
使用方法
d.measure()
测量温湿度。
d.temperature()
获取温度值。
d.humidity()
获取湿度值。

表 9-3 DHT11 对象

建议上电先延时 1 秒，让 DHT11 稳定后再开设读取。代码编写流程如下：

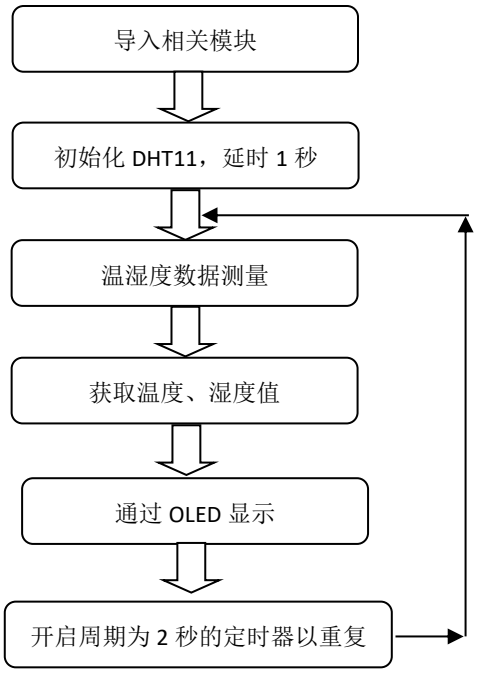


图 9-15 代码编写流程图

实验参考代码：

```
'''
实验名称：温湿度传感器 DHT11
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过编程采集温湿度数据，并在 OLED 上显示。。
'''

#引入相关模块
from machine import Pin,I2C,Timer
from ssd1306 import SSD1306_I2C
import dht,time
```

```
#初始化相关模块
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#创建 DTH11 对象
d = dht.DHT11(Pin(5)) #传感器连接到引脚 14
time.sleep(1)   #首次启动停顿 1 秒让传感器稳定

def dht_get(tim):

    d.measure()         #温湿度采集

   #OLED 显示温湿度
    oled.fill(0) #清屏背景黑色
    oled.text('01Studio', 0, 0)
    oled.text('DHT11 test:',0,15)
    oled.text(str(d.temperature() )+' C',0,40)   #温度显示
    oled.text(str(d.humidity())+' %',48,40)  #湿度显示
    oled.show()

#开启 RTOS 定时器，编号为-1，周期 2 秒
tim = Timer(-1)
tim.init(period=2000, mode=Timer.PERIODIC,callback=dht_get)
```

- **实验结果：**

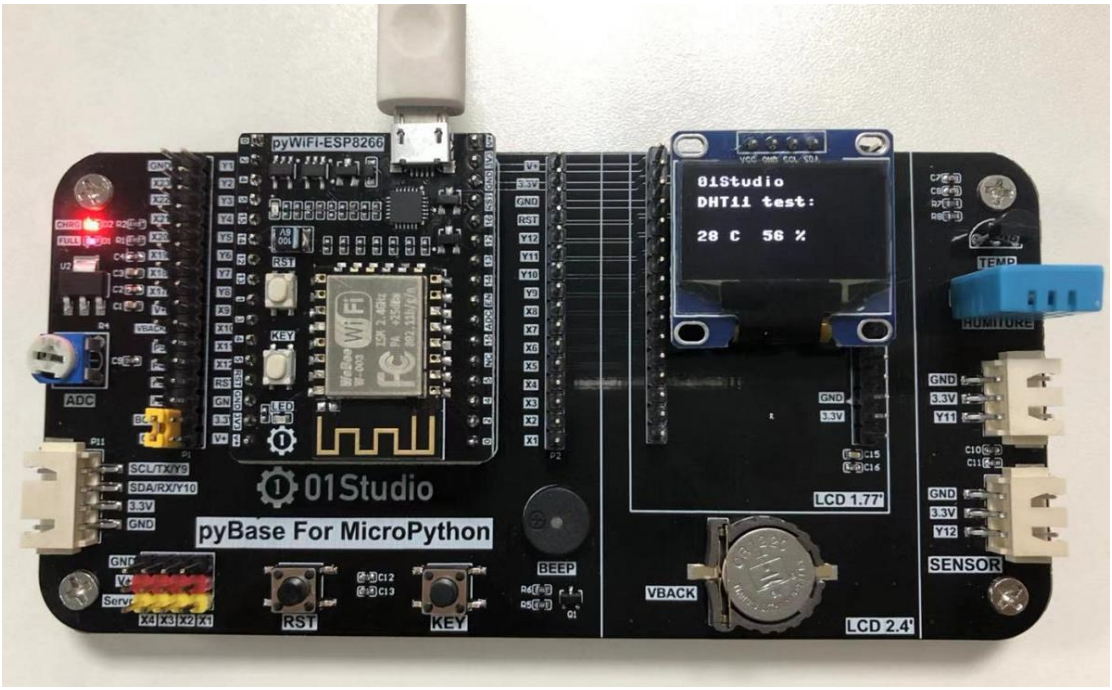


图 9-16　DHT11 实验结果

- **总结：**

通过本节学习我们学会了使用 MicroPython 来驱动 DTH11 温湿度传感器，DHT11 性价比比较高，是很适合学习使用的，但精度和响应速度有点低，需要更高要求应用的用户可以使用 DHT22 或者其他更高级的传感器。

第10章　WiFi 应用

通过前面的实验，我们已经对 ESP8266 有了一定的了解。从本章开始，将迎来非常重要实用的内容，那就是 WIFI 应用。ESP8266 就是为 WIFI 无线连接而生的。通过本章内容，我们可以看到基于 MicroPython 的 WIFI 开发是多么的简单而美妙。物联网的学习变得非常简单有趣！事不宜迟，马上开始学习。

10.1 连接无线路由器

- **前言：**

WIFI 是物联网中非常重要的角色，现在基本上家家户户都有 WIFI 网络了，通过 WIFI 接入到互联网，成了智能家居产品普遍的选择。而要想上网，首先需要连接上无线路由器。这一节我们就来学习如何通过 MicroPython 编程连上路由器。

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。

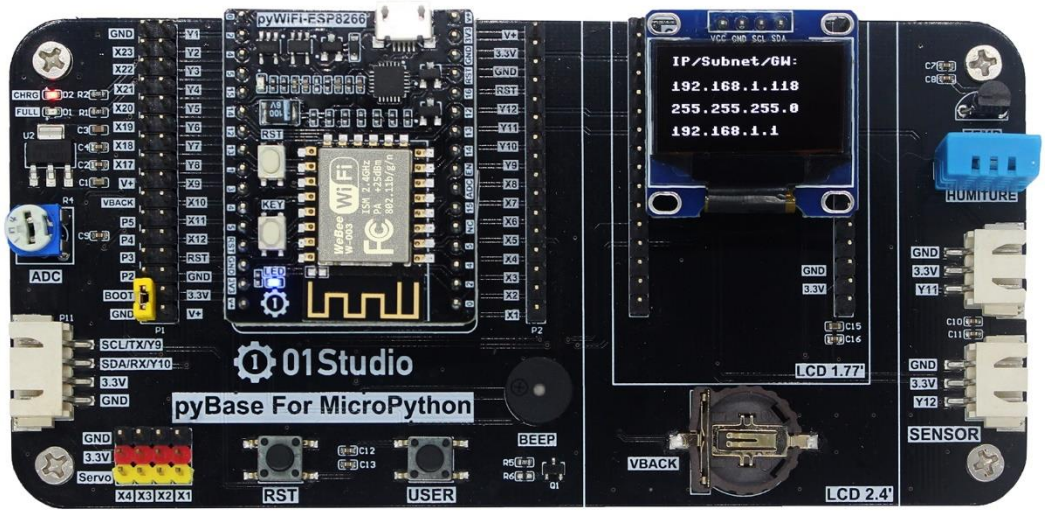


图 10-1 pyWiFi-ESP8266 开发套件

- **实验目的：**

编程实现连接路由器，将 IP 地址等相关信息通过 OLED 显示（只支持 2.4G 网络）。

- **实验讲解：**

连接路由器上网是我们每天都做的事情，日常生活中我们只需要知道路由器的账号和密码，就能使用电脑或者手机连接到无线路由器，然后上网冲浪。

MicroPython 已经集成了 network 模块，开发者使用内置的 network 模块函数可以非常方便地连接上路由器。但往往也有各种连接失败的情况，如密码不正

确等。这时候我们只需要再加上一些简单的判断机制，避免陷入连接失败的死循环即可！

我们先来看看 network 基于 WiFi（WLAN 模块）的构造函数和使用方法。

构造函数
wlan = network.WLAN(interface_id)
构建 WIFI 连接对象。interface_id:分为热点 network.AP_IF 和客户端 network.STA_IF 模式。
使用方法
wlan.active([is_active])
激活 wlan 接口。Ture：激活；False:关闭。
wlan.scan ()
扫描允许访问的 SSID。
wlan.isconnected()
检查设备是否已经连接上。返回 Ture:已连接；False：未连接。
wlan.connected(ssid,passwork)
WIFI 连接。ssid:账号；passwork：密码。
wlan.ifconfig([ip,subnet,gateway,dns])
设备信息配置。ip：IP 地址；subnet:子网掩码；gateway:网关地址；dns:DNS 信息。**（如果参数为空，则返回当前连接信息。）**
wlan.disconnected()
断开连接。

表 10-1　WLAN 对象

从上表可以看到 MicroPython 通过模块封装，让 WIFI 联网变得非常简单。模块包含热点 AP 模块和客户端 STA 模式，热点 AP 是指电脑端直接连接 ESP8266 发出的热点实现连接，但这样你的电脑就不能上网了，因此我们一般情况下都是使用 STA 模式。也就是电脑和设备同时连接到相同网段的路由器上。

模块上电后可以先判断是否已经连接到网络，如果是则无需再次连接，否的

话则进入 WIFI 连接状态，指示灯闪烁，连接成功后指示灯常亮，IP 等相关信息通过 OLED 显示和串口打印。另外需要配置超时 15 秒还没连接成功时执行取消连接，避免因无法连接而陷入死循环。代码编写流程如下：

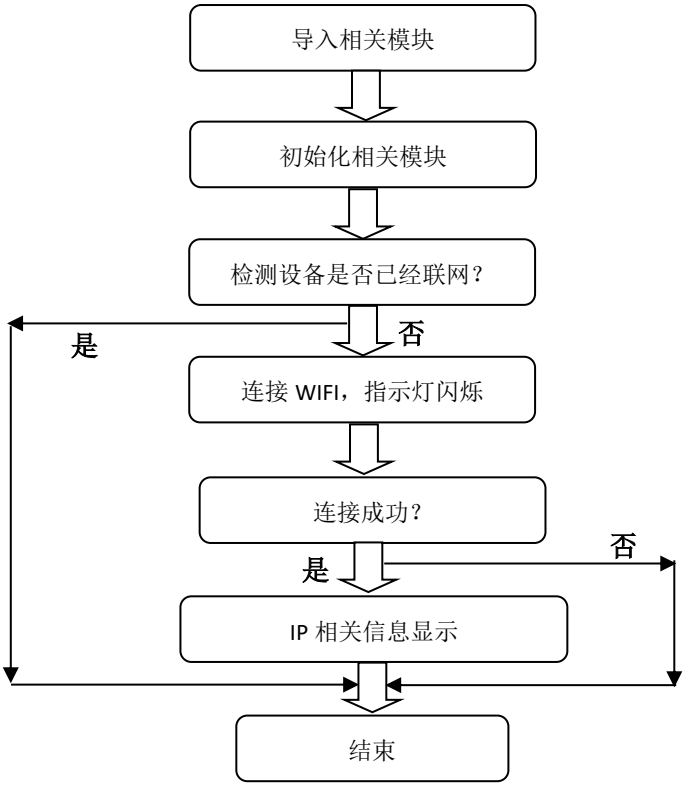


图 10-2 代码编写流程图

实验参考代码如下：

```
'''
实验名称：连接无线路由器
版本：v1.0
日期：2019.8
作者：01Studio
说明：编程实现连接路由器，将 IP 地址等相关信息通过 OLED 显示（只支持 2.4G 网络）。
'''

import network,time
from machine import I2C,Pin
from ssd1306 import SSD1306_I2C
```

```
#初始化相关模块
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#WIFI 连接函数
def WIFI_Connect():

    WIFI_LED=Pin(2, Pin.OUT, value=1) #初始化 WIFI 指示灯

    wlan = network.WLAN(network.STA_IF) #STA 模式
    wlan.active(True)                   #激活接口
    start_time=time.time()              #记录时间做超时判断

    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码

        while not wlan.isconnected():

            #LED 闪烁提示
            WIFI_LED.value(0)
            time.sleep_ms(300)
            WIFI_LED.value(1)
            time.sleep_ms(300)

            #超时判断,15 秒没连接成功判定为超时
            if time.time()-start_time > 15 :
                print('WIFI Connected Timeout!')
                break
```

```
    if wlan.isconnected():
        #LED 点亮
        WIFI_LED.value(0)

        #串口打印信息
        print('network information:', wlan.ifconfig())

        #OLED 数据显示
        oled.fill(0)   #清屏背景黑色
        oled.text('IP/Subnet/GW:',0,0)
        oled.text(wlan.ifconfig()[0], 0, 20)
        oled.text(wlan.ifconfig()[1],0,38)
        oled.text(wlan.ifconfig()[2],0,56)
        oled.show()

#执行 WIFI 连接函数
WIFI_Connect()
```

上面代码在 main.py 文件中，将 WIFI 连接封装成了子函数形式，我们也可以新建一个 WIFI.py 文件，然后通过模块引用方式来供 main.py 调用，以提高程序的灵活性。

- **实验结果：**

下载程序，可以观察到上电后 pyWiFi-ESP8266 上的 LED 快速闪烁，连接成功后 LED 常亮，OLED 显示当前 IP、子网掩发、网关的地址信息。

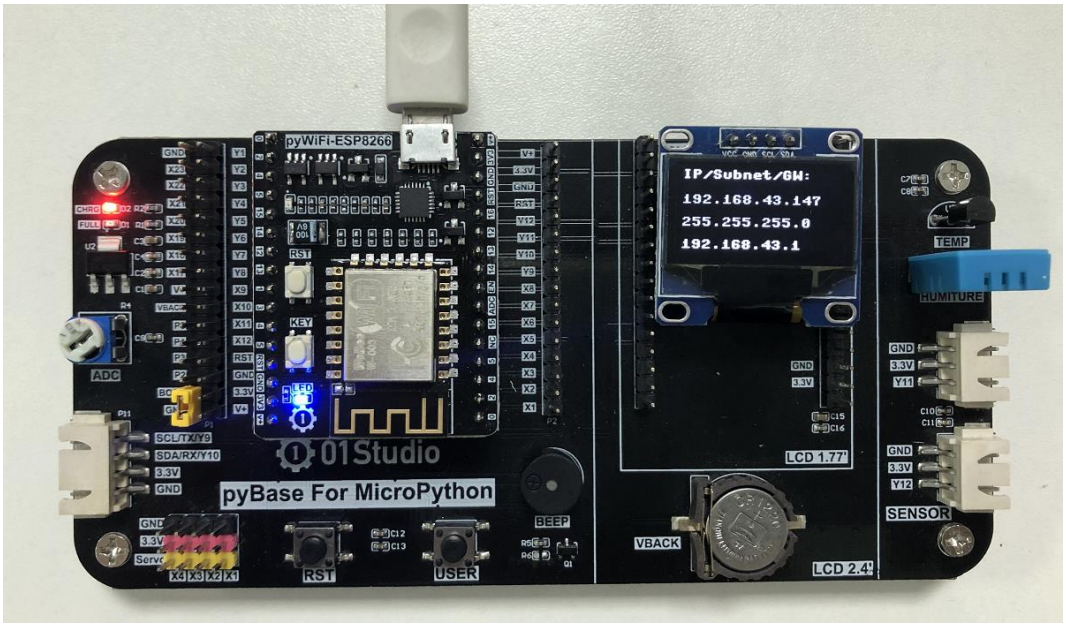


图 10-3 实验现象

- **总结：**

本节是 WIFI 应用的基础，成功连接到无线路由器的实验后，后面就可以做 socket 等相关网络通信的应用了。

10.2 Socket 通信

- **前言：**

上一节我们学习了如何通过 MicroPython 编程实现 pyWiFi-ESP8266 模块连接到无线路由器。这一节我们则来学习一下 Socket 通信实验。Socket 几乎是整个互联网通信的基础。

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。

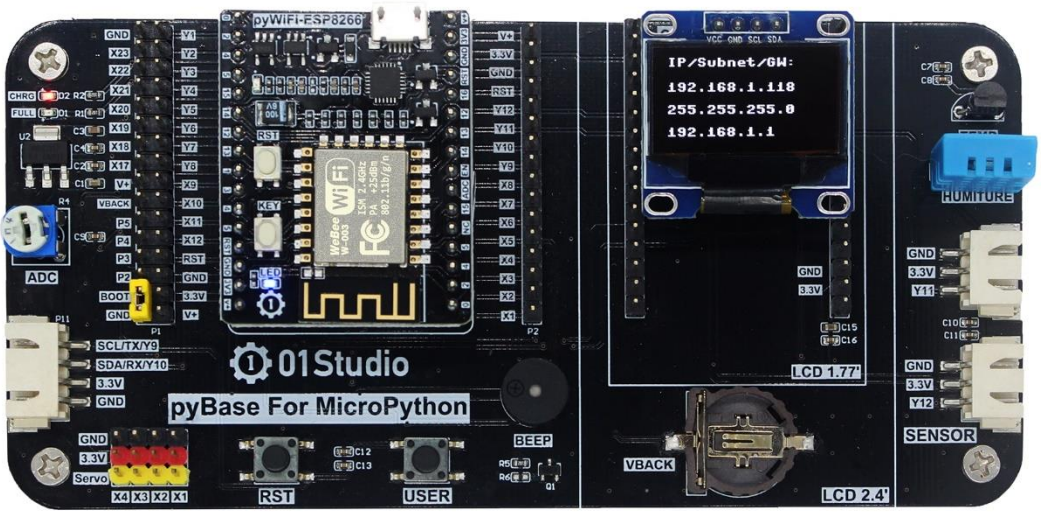


图 10-4 pyWiFi-ESP8266 开发套件

- **实验目的：**

通过 Socket 编程实现 pyWiFi-ESP8266 与电脑服务器助手建立连接，相互收发数据。

- **实验讲解：**

Socket 我们听得非常多了，但由于网络工程是一门系统工程，涉及的知识非常广，概念也很多，任何一个知识点都能找出一堆厚厚的的书，因此我们经常会混淆。在这里，我们尝试以最容易理解的方式来讲述 Socket，如果需要全面了解，可以自行查阅相关资料学习。

我们先来看看网络层级模型图，这是构成网络通信的基础：

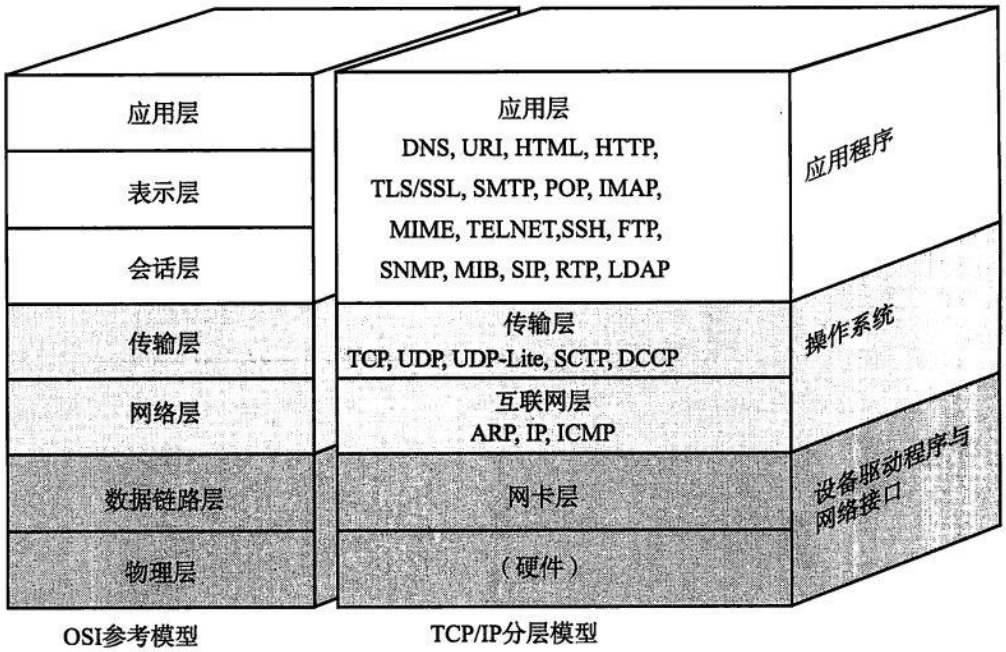


图 10-5 网络层级模型

我们看看 TCP/IP 模型的传输层和应用层，传输层比较熟悉的概念是 TCP 和 UDP，UPD 协议基本就没有对 IP 层的数据进行任何的处理了。而 TCP 协议还加入了更加复杂的传输控制，比如滑动的数据发送窗口（Slice Window），以及接收确认和重发机制，以达到数据的可靠传送。应用层中网页常用的则是 HTTP。那么我们先来解析一下这 TCP 和 HTTP 两者的关系。

我们知道网络通信是最基础是依赖于 IP 和端口的，HTTP 一般情况下默认使用端口 80。举个简单的例子：我们逛淘宝，浏览器会向淘宝网的网址（本质是 IP）和端口发起请求，而淘宝网收到请求后响应，向我们手机返回相关网页数据信息，实现了网页交互的过程。而这里就会引出一个多人连接的问题，很多人访问淘宝网，实际上接收到网页信息后就断开连接，否则淘宝网的服务器是无法支撑这么多人长时间的连接的，哪怕能支持，也非常占资源。

也就是应用层的 HTTP 通过传输层进行数据通信时，TCP 会遇到同时为多个应用程序进程提供并发服务的问题。多个 TCP 连接或多个应用程序进程可能需要通过同一个 TCP 协议端口传输数据。为了区别不同的应用程序进程和连接，许多计算机操作系统为应用程序与 TCP / IP 协议交互提供了套接字(Socket)接口。应用层可以和传输层通过 Socket 接口，区分来自不同应用程序进程或网络连接的通信，实

现数据传输的并发服务。

简单来说，Socket 抽象层介于传输层和应用层之间，跟 TCP/IP 并没有必然的联系。Socket 编程接口在设计的时候，就希望也能适应其他的网络协议。

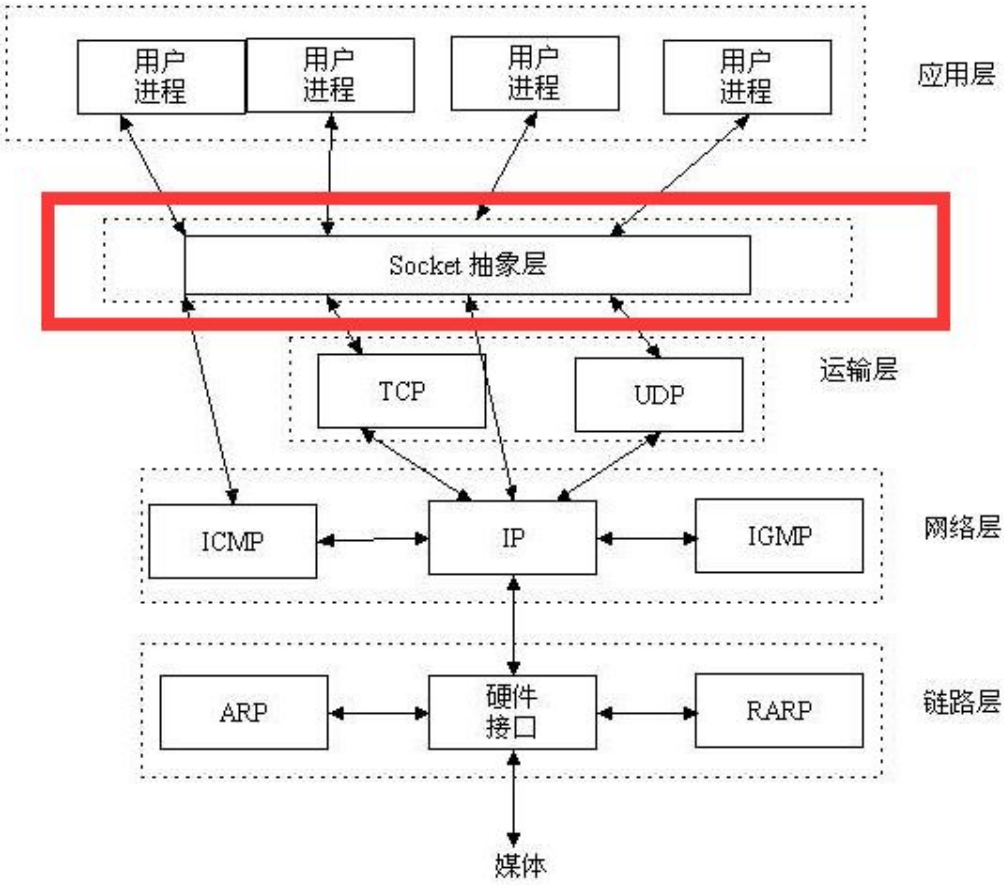


图 10-6　Socket 抽象层

套接字（socket）是通信的基石，是支持 TCP/IP 协议的网络通信的基本操作单元。它是网络通信过程中端点的抽象表示，包含进行网络通信必须的五种信息：**连接使用的协议（通常是 TCP 或 UDP），本地主机的 IP 地址，本地进程的协议端口，远地主机的 IP 地址，远地进程的协议端口。**

所以，socket 的出现只是可以更方便的使用 TCP/IP 协议栈而已，简单理解就是其对 TCP/IP 进行了抽象，形成了几个最基本的函数接口。比如 create，listen，accept，connect，read 和 write 等等。以下是通讯流程：

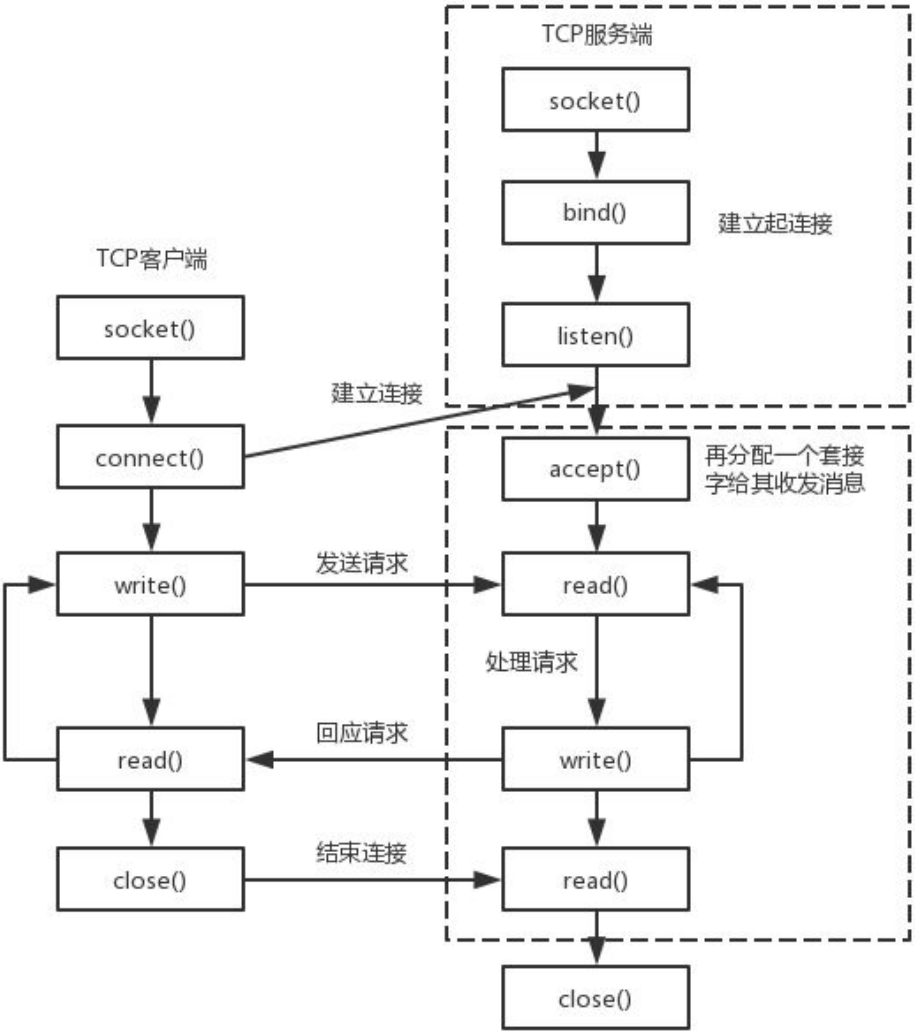


图 10-7　Socket 通信过程

从上图可以看到，建了 Socket 通信需要一个服务器端和一个客户端，以本实验为例，pyWiFi-ESP8266 作为客户端，电脑使用网络调试助手作为服务器端，双方使用 TCP 协议传输。对于客户端，则需要知道电脑端的 IP 和端口即可建立连接。（端口可以自定义，范围在 0~65535，注意不占用常用的 80 等端口即可。）

以上的内容，简单来说就是如果用户面向应用来说，那么 ESP8266 只需要知道**通讯协议是 TCP 或 UDP、服务器的 IP 和端口号**这 3 个信息，即可向服务器发起连接和发送信息。就这么简单。

MicroPython 已经封装好相关模块 usocket,跟传统的 socket 大部分兼容，两者均可使用，本实验使用 usocket，对象如下介绍：

构造函数
s=usocket.socekt(af=AF_INET, type=SOCK_STREAM,proto=IPPROTO_TCP)
构建 usocket 对象。 af: AF_INET→IPV4，AF_INET6 → IPV6； type: SCOK_STREAM→TCP，SOCK_DGRAM→UDP； proto: IPPROTO_TCP→TCP 协议，IPPROTO_UDP→UDP 协议。 （如果要构建 TCP 连接，可以使用默认参数配置，即不输入任何参数。）
使用方法
addr=usocket.getaddrinfo('www.01studio.org', 80)[0][-1]
获取 Socket 通信格式地址。返回：('47.91.208.161',80)
s.connect(address)
创建连接。address:地址格式为 IP+端口。例：('192.168.1.115',10000)
s.send(bytes)
发送。bytes：发送内容格式为字节
s.recv(bufsize)
接收数据。bufsize：单次最大接收字节个数。
s.bind(address)
绑定，用于服务器角色
s.listen([backlog])
监听，用于服务器角色。backlog:允许连接个数，必须大于 0。
s.accept()
接受连接，用于服务器角色。
*其它更多用法请阅读 MicroPython 文档：(搜索:usocket) 中文文档链接：http://docs.micropython.01studio.org/

表 10-2　Socket 对象

本实验中 pyWiFi-ESP8266 属于客户端，因此只用到客户端的函数即可。实验代码编写流程如下：

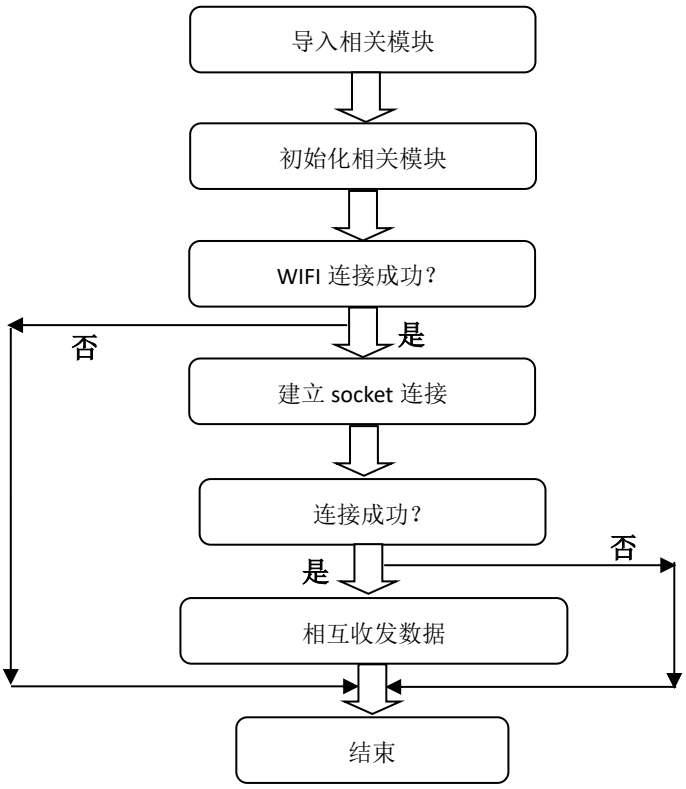


图 10-8 代码编写流程图

实验参考代码：

```
'''
实验名称：连接无线路由器
版本：v1.0
日期：2019.8
作者：01Studio
说明：通过 Socket 编程实现 pyWiFi-ESP8266 与电脑服务器助手建立 TCP 连接，相互收发数据。
'''
#导入相关模块
import network,usocket,time
from machine import I2C,Pin,Timer
from ssd1306 import SSD1306_I2C
```

```
#初始化相关模块
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

# WIFI 连接函数
def WIFI_Connect():

    WIFI_LED=Pin(2, Pin.OUT, value=1) #初始化 WIFI 指示灯

    wlan = network.WLAN(network.STA_IF) #STA 模式
    wlan.active(True)                   #激活接口
    start_time=time.time()              #记录时间做超时判断

    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码
        while not wlan.isconnected():
            #LED 闪烁提示
            WIFI_LED.value(0)
            time.sleep_ms(300)
            WIFI_LED.value(1)
            time.sleep_ms(300)

            #超时判断,15 秒没连接成功判定为超时
            if time.time()-start_time > 15 :
                print('WIFI Connected Timeout!')
                break

    if wlan.isconnected():
        #LED 点亮
```

```
        WIFI_LED.value(0)

        #串口打印信息
        print('network information:', wlan.ifconfig())

        #OLED 数据显示
        oled.fill(0)   #清屏背景黑色
        oled.text('IP/Subnet/GW:',0,0)
        oled.text(wlan.ifconfig()[0], 0, 20)
        oled.text(wlan.ifconfig()[1],0,38)
        oled.text(wlan.ifconfig()[2],0,56)
        oled.show()
        return True

    else:
        return False

def Socket_fun(tim):

    text=s.recv(128) #单次最多接收 128 字节
    if text == '':
        pass

    else: #打印接收到的信息为字节，可以通过 decode('utf-8')转成字符串
        print(text)
        s.send('I got:'+text.decode('utf-8'))

#判断 WIFI 是否连接成功
if WIFI_Connect():
```

```
#创建 socket 连接 TCP 类似，连接成功后发送“Hello 01Studio!”给服务器。
s=usocket.socket()
addr=('192.168.1.115',10000) #服务器 IP 和端口
s.connect(addr)
s.send('Hello 01Studio!')

#开启 RTOS 定时器，编号为-1,周期 300ms，执行 socket 通信接收任务
tim = Timer(-1)
tim.init(period=300, mode=Timer.PERIODIC,callback=Socket_fun)
```

WIFI 连接代码在上一节已经讲解，这里不再重复，WIFI 连接成功后返回 True，否则返回 False。程序在返回连接成功后建了 Socket 连接，连接成功发送‘Hello 01Studio!’信息到服务器。另外 RTOS 定时器设定了了每 300ms 处理从服务器接收到的数据。将接收到数据通过串口打印和发送给服务器。

- **实验结果：**

先在电脑端打开网络调试助手并建立服务器，软件在 零一科技（01Studio）MicroPython 开发套件配套资料 latest\01-开发工具\01-Windows\网络调试助手目录下的 NetAssist.exe ，直接双击打开即可！

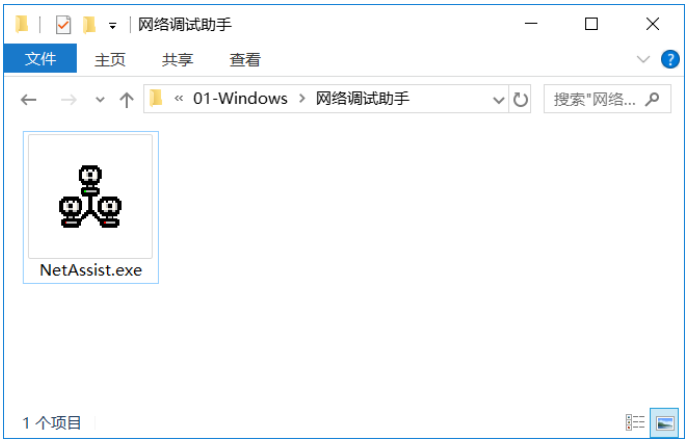


图 10-9 网络调试助手

以下是新建服务器的方法，打开网络调试助手后在左上角协议类型选择 TCP

Server；中间的本地 IP 地址是自动识别的，不要修改，这个就是服务器的 IP 地址。然后端口写 10000（0-65535 都可以。），点击连接，成功后红点亮。如下图：



图 10-10 TCP 服务器配置

在时候服务器已经在监听状态！用户需要根据自己的实际情况自己输入 WIFI 信息和服务器 IP 地址+端口。即修改上面的代码以下部分内容。（服务器 IP 和端口可以在网络调试助手找到。）

```
wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码
addr=('192.168.1.115',10000) #服务器 IP 和端口
```

下载程序，开发板成功连接 WIFI 后，发起了 socket 连接，连接成功可以可以看到网络调试助手收到了开发板发来的信息。在下方列表多了一个连接对象，点击选中：

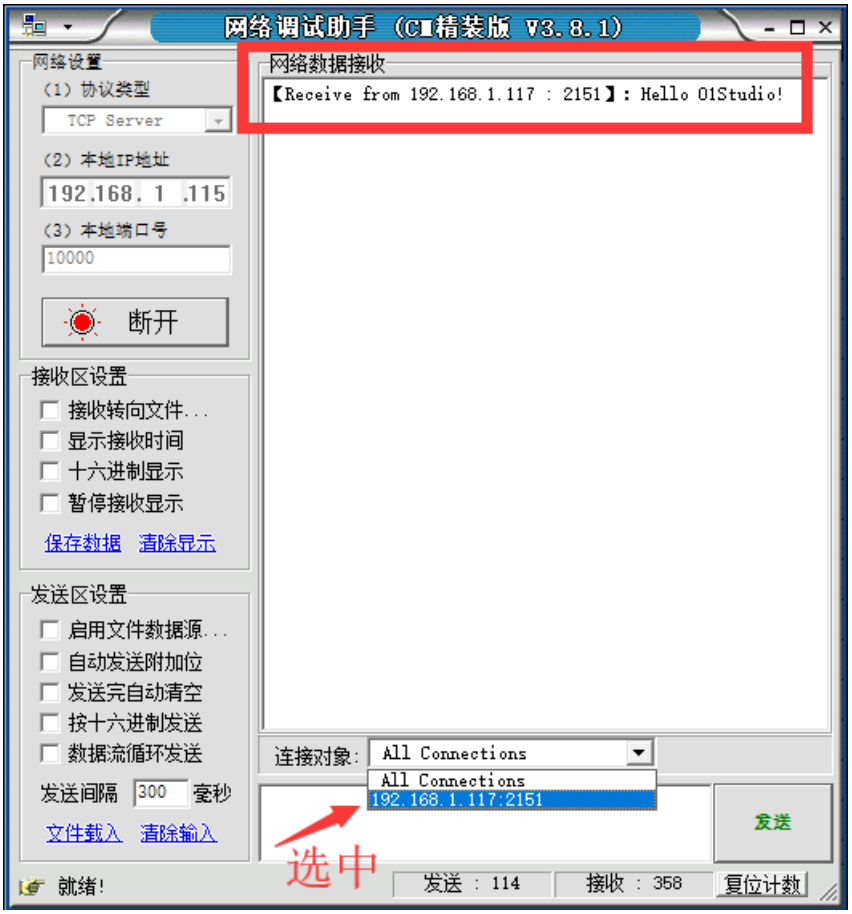


图 10-11 选中连接对象

选中后我们在发送框输入信息“Hi”，点击发送，可以看到开发板的 REPL 打印出来信息 Hi。为字节数据。另外由于程序将收到的信息发回给服务器，所以在网络调试助手中也接收到开发板返回的信息：I got:Hi。

```
MicroPython v1.11-8-g48dcbbe60 on 2019-05-29; ESP module with ESP8266
Type "help()" for more information.
>>> Warning: Comparison between bytes and str
b'Hi'
```

图 10-12

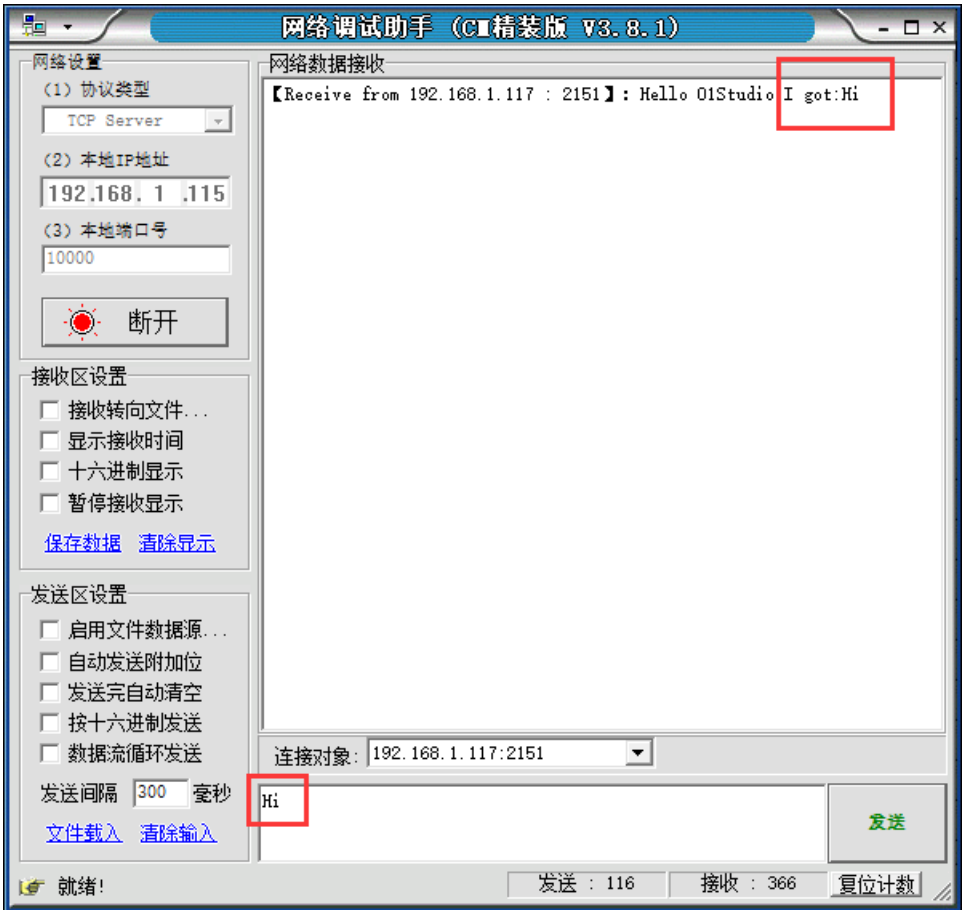


图 10-13 服务器发送数据

- **总结：**

通过本节学习，我们了解了 socket 通信原理以及使用 MicroPython 进行 socket 编程并且通信的实验。得益于优秀的封装，让我们可以直接面向 socket 对象编程就可以快速实现 socket 通信，从而开发更多的网络应用，例如将前面采集到的传感器数据发送到服务器。

10.3 MQTT 通信

- **前言：**

上一节，我们学习了 Socket 通信，当服务器和客户端建立起连接时，就可以相互通信了。在互联网应用大多使用 WebSocket 接口来传输数据。而在物联网应用中，常常出现这样的情况：海量的传感器，需要时刻保持在线，传输数据量非常低，有着大量用户使用。如果仍然使用 socket 作为通信，那么服务器的压力和通讯框架的设计随着数量的上升将变得异常复杂！

那么有无一个框架协议来解决这个问题呢，答案是有的。那就是 MQTT(消息队列遥测传输)。

- **实验平台：**

pyWiFi-ESP8266 和 pyBase 开发底板。

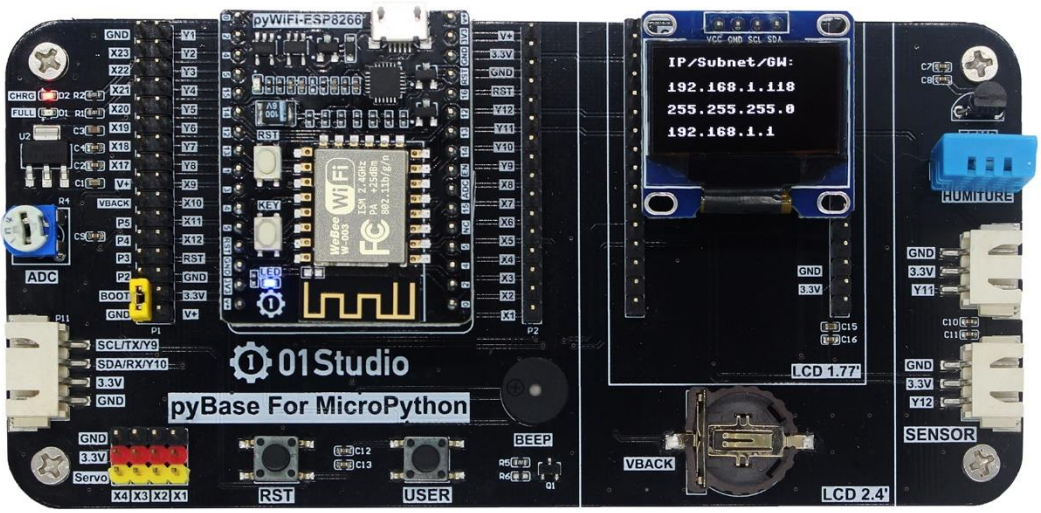


图 10-14 pyWiFi-ESP8266 开发套件

- **实验目的：**

通过编程实现 pyWiFi-ESP8266 实现 MQTT 协议信息的发布和订阅（接收）。

- **实验讲解：**

MQTT 是 IBM 于 1999 年提出的，和 HTTP 一样属于应用层，它工作在 TCP/IP 协议族上，通常还会调用 socket 接口。是一个基于客户端-服务器的消息发布/订

阅传输协议。其特点是协议是轻量、简单、开放和易于实现的，这些特点使它适用范围非常广泛。在很多情况下，包括受限的环境中，如：机器与机器（M2M）通信和物联网（IoT）。其在，通过卫星链路通信传感器、偶尔拨号的医疗设备、智能家居、及一些小型化设备中已广泛使用。

总结下来 MQTT 有如下特性/优势：

- **异步消息协议**
- **面向长连接**
- **双向数据传输**
- **协议轻量级**
- **被动数据获取**

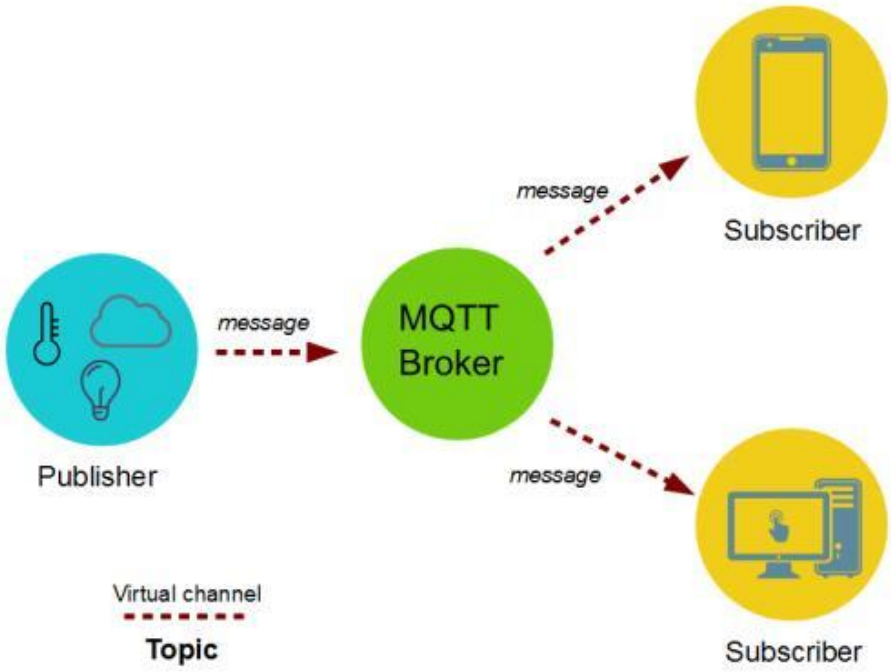


图 10-15 MQTT 通信流程

从上图可以看到，MQTT 通信的角色有两个，分别是服务器和客户端。服务器只负责中转数据，不做存储；客户端可以是信息发送者或订阅者，也可以同时是两者。具体如下图：

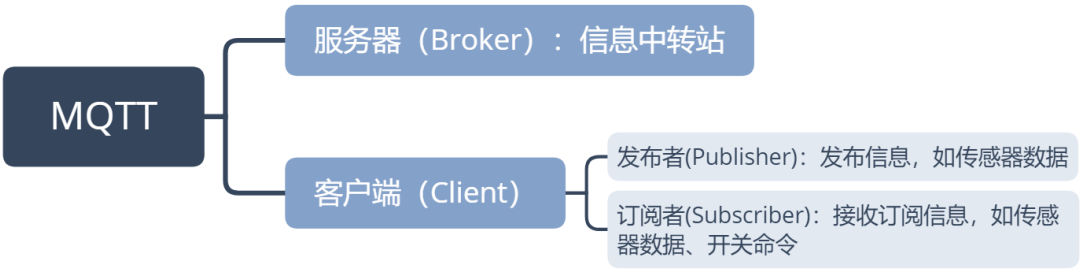


图 10-16　MQTT 角色说明

确定了角色后是如何传输数据呢？下表示 MQTT 最基本的数据帧格式，例如温度传感器发布主题“Temperature”编号,消息是“25”（表示温度）。那么所有订阅了这个主题编号的客户端（手机应用）就会收到相关信息，从而实现通信。如下表所示：

MQTT 数据帧格式	
Topic ID (主题编号)	Message (消息)
Temperature	25

图 10-17 MQTT 数据格式

由于特殊的发布/订阅机制，服务器不需要存储数据（当然也可以在服务器的设备上建立一个客户端来订阅保存信息），因此非常适合海量设备的传输。

人生苦短，而 MicroPython 已经封装好了 MQTT 客户端的库文件。让我们的应用变得简单美妙。MQTT 模块文件为例程文件夹里面的 simple.py 文件。使用方法如下：

构造函数
client=simple. MQTTClient (client_id, server, port)
构建 MQTT 客户端对象。 client_id: 客户端 ID，具有唯一性； server: 服务器地址，可以是 IP 或者网址； port:服务器端口。（默认是 1883，服务器通常采用的端口，也可以自定义。）
使用方法

client.connect()
连接到服务器。
client.publish(TOPIC,message)
发布。TOPIC：主题编号；message：信息内容，例：'Hello 01Studio!'
client.subscribe(TOPIC)
订阅。TOPIC：主题编号。
client.set_callback(callback)
设置回调函数。callback：订阅后如果接收到信息，就执行相名称的回调函数。
client.check_msg()
检查订阅信息。如收到信息就执行设置过的回调函数 callback。

表 10-3　MQTT 客户端对象

由于客户端分为发布者和订阅者角色，因此为了方便大家更好理解，本实验分开两个案例来编程，分别为发布者和订阅者。再结合 MQTT 网络调试助手来测试。代表编写流程图如下：

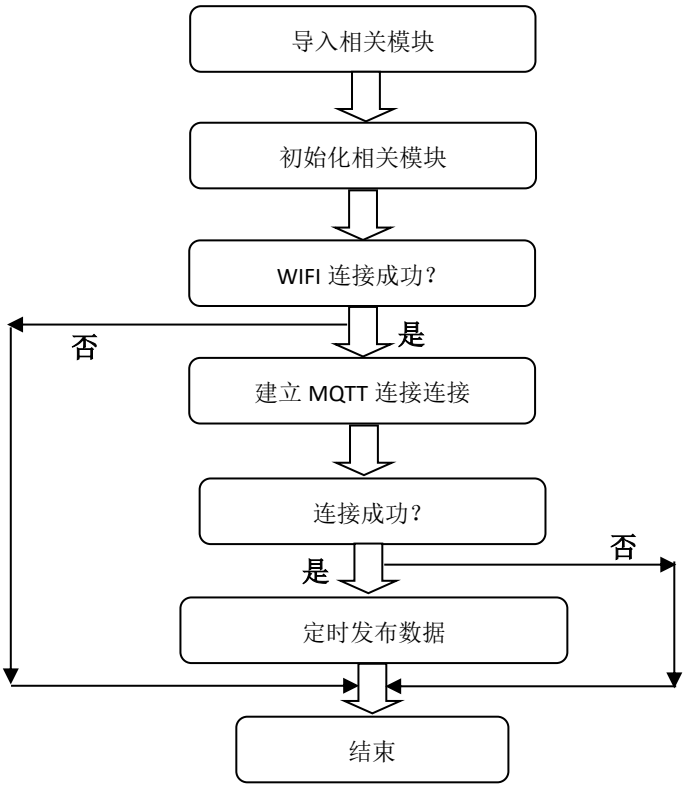


图 10-18 发布者代码编写流程

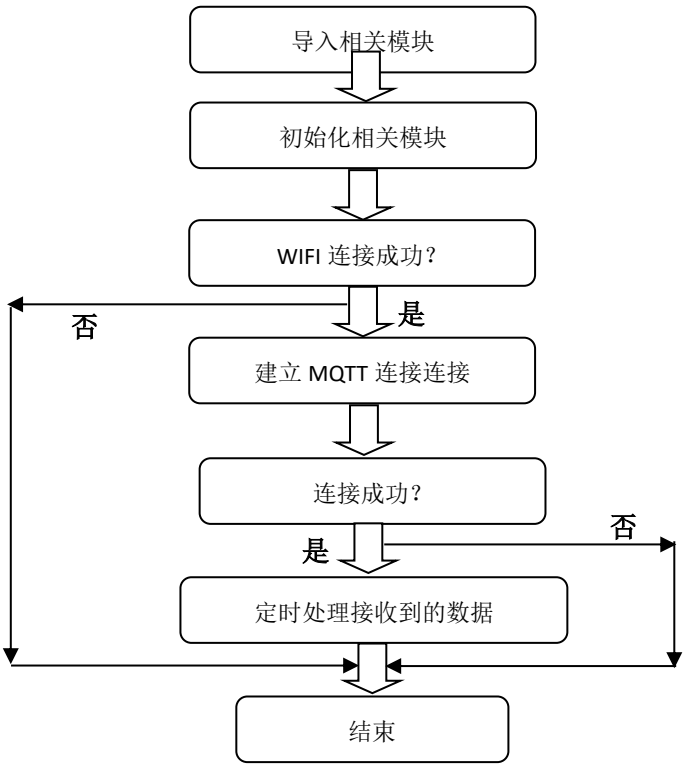


图 10-19 订阅者代码编写流程

发布者（publish）参考代码：

```
'''
实验名称：MQTT 通信
版本：v1.0
日期：2019.8
作者：01Studio
说明：编程实现 MQTT 通信，实现发布数据。
'''
import network,time
from simple import MQTTClient #导入 MQTT 板块
from machine import I2C,Pin,Timer
from ssd1306 import SSD1306_I2C


#初始化相关模块
```

```
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#WIFI 连接函数
def WIFI_Connect():

    WIFI_LED=Pin(2, Pin.OUT, value=1) #初始化 WIFI 指示灯

    wlan = network.WLAN(network.STA_IF) #STA 模式
    wlan.active(True)                   #激活接口
    start_time=time.time()              #记录时间做超时判断

    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码

        while not wlan.isconnected():

            #LED 闪烁提示
            WIFI_LED.value(0)
            time.sleep_ms(300)
            WIFI_LED.value(1)
            time.sleep_ms(300)

            #超时判断,15 秒没连接成功判定为超时
            if time.time()-start_time > 15 :
                print('WIFI Connected Timeout!')
                break

    if wlan.isconnected():
```

```
        #LED 点亮
        WIFI_LED.value(0)

        #串口打印信息
        print('network information:', wlan.ifconfig())

        #OLED 数据显示（如果没接 OLED，请将下面代码屏蔽）
        oled.fill(0)   #清屏背景黑色
        oled.text('IP/Subnet/GW:',0,0)
        oled.text(wlan.ifconfig()[0], 0, 20)
        oled.text(wlan.ifconfig()[1],0,38)
        oled.text(wlan.ifconfig()[2],0,56)
        oled.show()
        return True

    else:
        return False

#发布数据任务
def MQTT_Send(tim):
    client.publish(TOPIC, 'Hello 01Studio!')

#执行 WIFI 连接函数并判断是否已经连接成功
if WIFI_Connect():

    SERVER = 'mqtt.p2hp.com'
    PORT = 1883
    CLIENT_ID = '01Studio-ESP8266' # 客户端 ID
    TOPIC = '/public/01Studio/1' # TOPIC 名称
    client = MQTTClient(CLIENT_ID, SERVER, PORT)
```

```
    client.connect()


    #开启 RTOS 定时器，编号为-1,周期 1000ms，执行 socket 通信接收任务
    tim = Timer(-1)
    tim.init(period=1000, mode=Timer.PERIODIC,callback=MQTT_Send)
```

订阅者（subscribe）参考代码：

```
'''
实验名称：MQTT 通信
版本：v1.0
日期：2019.8
作者：01Studio
说明：编程实现 MQTT 通信，实现订阅（接收）数据。
'''
import network,time
from simple import MQTTClient #导入 MQTT 板块
from machine import I2C,Pin,Timer
from ssd1306 import SSD1306_I2C


#初始化相关模块
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)


#WIFI 连接函数
def WIFI_Connect():


    WIFI_LED=Pin(2, Pin.OUT, value=1) #初始化 WIFI 指示灯


    wlan = network.WLAN(network.STA_IF) #STA 模式
```

```
wlan.active(True)                    #激活接口
start_time=time.time()               #记录时间做超时判断

if not wlan.isconnected():
    print('connecting to network...')
    wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码

    while not wlan.isconnected():

        #LED 闪烁提示
        WIFI_LED.value(0)
        time.sleep_ms(300)
        WIFI_LED.value(1)
        time.sleep_ms(300)

        #超时判断,15 秒没连接成功判定为超时
        if time.time()-start_time > 15 :
            print('WIFI Connected Timeout!')
            break

if wlan.isconnected():
    #LED 点亮
    WIFI_LED.value(0)

    #串口打印信息
    print('network information:', wlan.ifconfig())

    #OLED 数据显示（如果没接 OLED，请将下面代码屏蔽）
    oled.fill(0)   #清屏背景黑色
    oled.text('IP/Subnet/GW:',0,0)
```

```
        oled.text(wlan.ifconfig()[0], 0, 20)
        oled.text(wlan.ifconfig()[1],0,38)
        oled.text(wlan.ifconfig()[2],0,56)
        oled.show()
        return True

    else:
        return False

#设置 MQTT 回调函数,有信息时候执行
def MQTT_callback(topic, msg):
    print('topic: {}'.format(topic))
    print('msg: {}'.format(msg))

#接收数据任务
def MQTT_Rev(tim):
    client.check_msg()

#执行 WIFI 连接函数并判断是否已经连接成功
if WIFI_Connect():

    SERVER = 'mqtt.p2hp.com'
    PORT = 1883
    CLIENT_ID = '01Studio-ESP8266' # 客户端 ID
    TOPIC = '/public/01Studio/1' # TOPIC 名称

    client = MQTTClient(CLIENT_ID, SERVER, PORT) #建立客户端对象
    client.set_callback(MQTT_callback)  #配置回调函数
    client.connect()
    client.subscribe(TOPIC) #订阅主题
```

```
#开启 RTOS 定时器，编号为-1,周期 300ms，执行 socket 通信接收任务
tim = Timer(-1)
tim.init(period=300, mode=Timer.PERIODIC,callback=MQTT_Rev)
```

从以上代码可以看到发布者和订阅者的编程方式相近，另外本实验需要一个 MQTT 服务器（Broker），这里使用的是跟我们将用到的 MQTT 网络助手同一个服务器和端口。

```
SERVER = 'mqtt.p2hp.com'
PORT = 1883
```

- **实验结果：**

为了方便测试，我们可以使用 MQTT 网络助手进行调试。这里推荐一个在线 MQTT 网络调试助手：http://mqtt.p2hp.com/websocket/

打开上面网址，即可看到 MQTT 在线调试助手。可以配置基本信息，这里默认即可，点击连接。

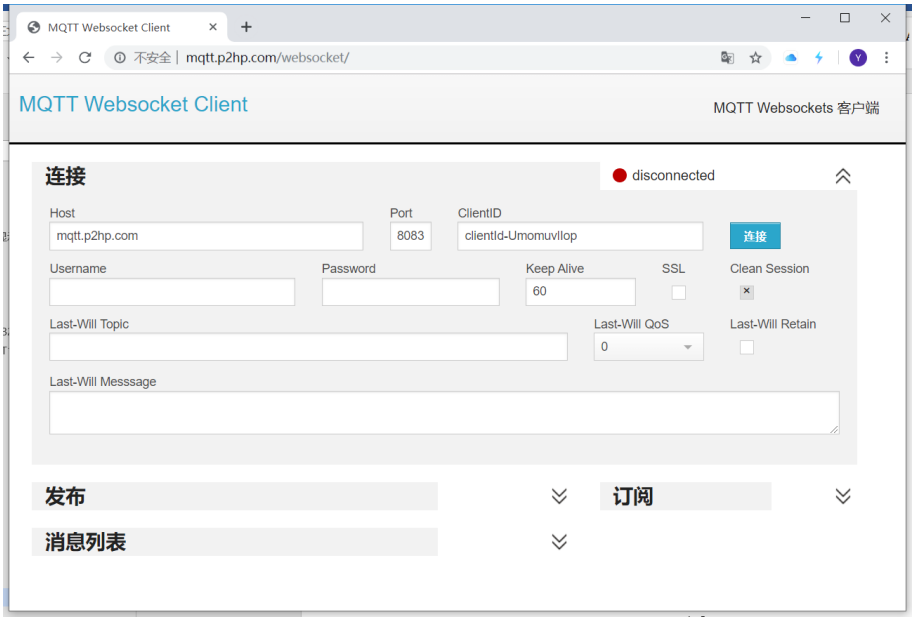


图 10-20 MQTT 助手

连接成功可以看到显示 Connected。

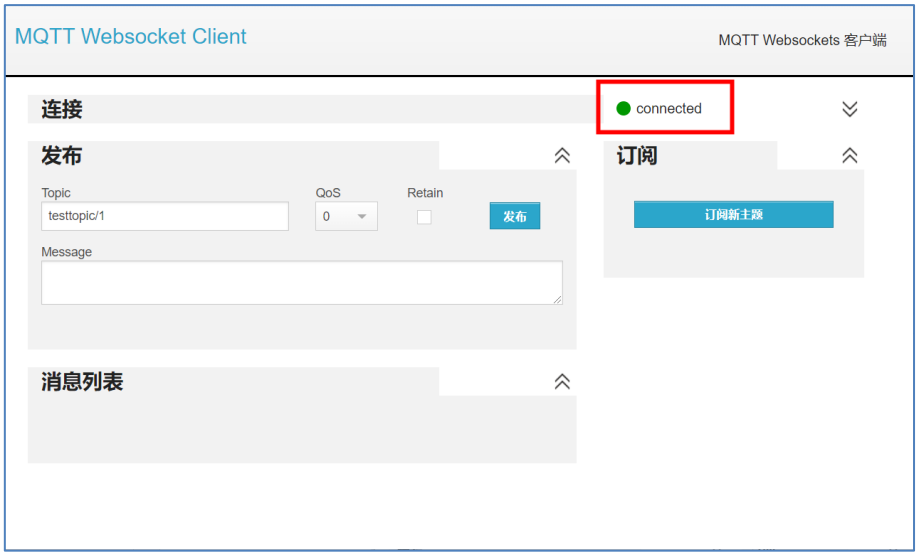


图 10-21 MQTT 助手连接成功

测试“发布者”代码：

我们先测试“发布者”代码。将“发布者”代码下载到开发板，然后在 MQTT 助手中订阅主题修改为：'/public/01Studio/1'（跟代码发布的主题一致），QOS 选择 0 即可。然后点击订阅主题。

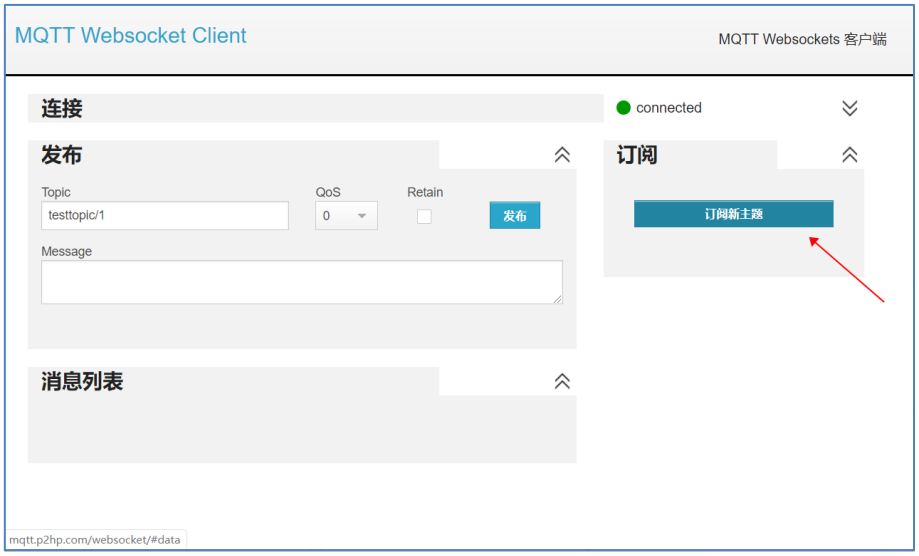


图 10-22 订阅主题

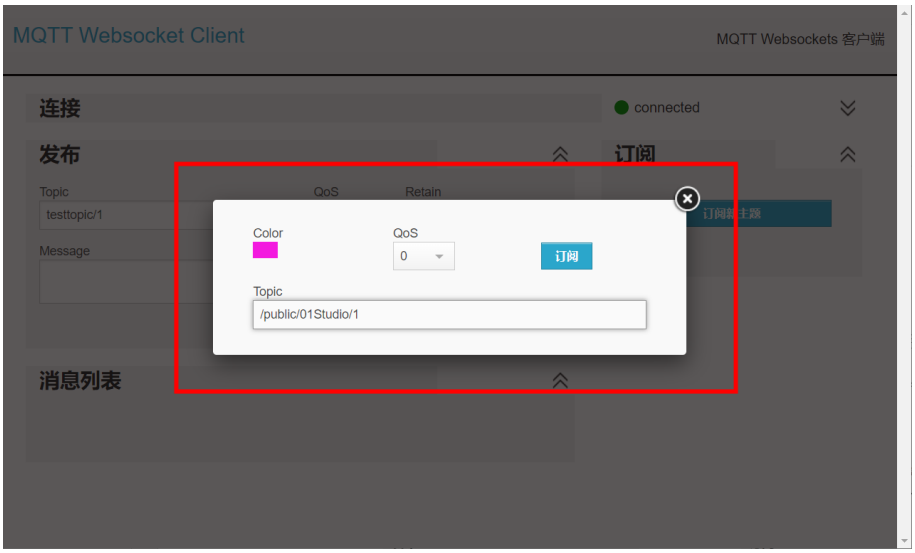


图 10-23

订阅后可以看到接收框收到来自开发板发布的信息。

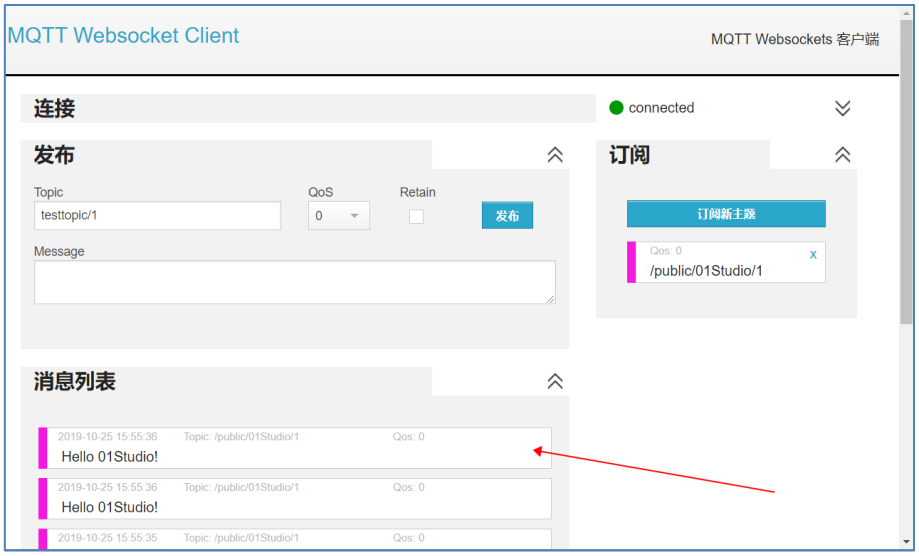


图 10-24 MQTT 助手收到开发板发布的信息

测试“订阅者”代码：

“订阅者”代码测试方法跟“发布者”相反。将“订阅者”代码下载到开发板，然后在电脑 MQTT 助手中发布主题修改为：'/public/01Studio/1'（跟代码发布的主题一致。）在下方空白框输入“Hello 01Studio!”。

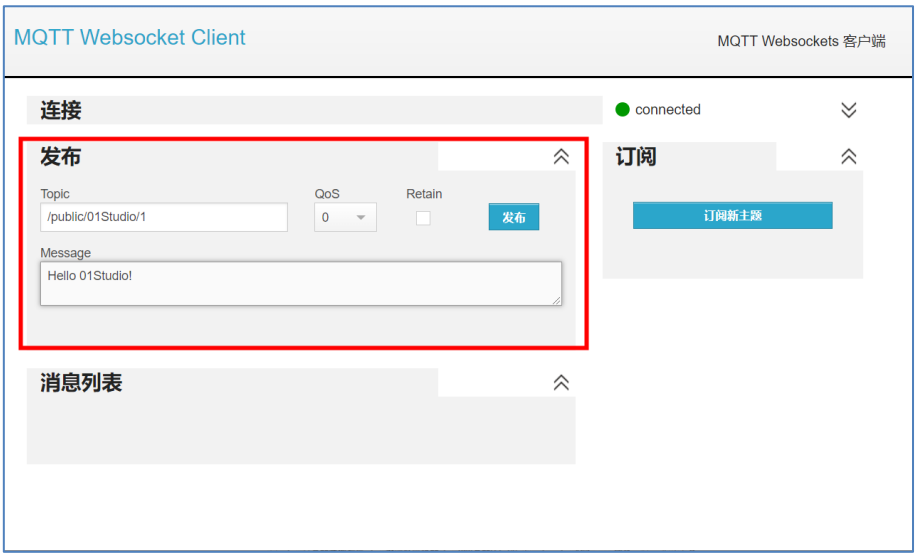


图 10-25 输入即将发布的主题和信息

将“订阅者”代码下载到开发板，成功连接后回到 MQTT 助手点击【发布】发布信息，可以在开发板的 REPL 看到接收到的订阅信息“Hello 01Studio!”被打印出来了（数据格式为字节数据）。

```
ESP MicroPython REPL
network information: ('192.168.1.117', '255.255.255.0', '192.168.1.1', '192.168.1.1')
◆◆>
MicroPython v1.11-8-g48dcbbe60 on 2019-05-29; ESP module with ESP8266
Type "help()" for more information.
>>> topic: b'/public/01Studio/1
msg: b'Hello 01Studio!'
```

图 10-26 开发板接收到相关信息并打印

当然你也可以在同一个 MQTT 在线助手下测试发布和订阅功能，只需要将主题设置一致即可，如下图所示：

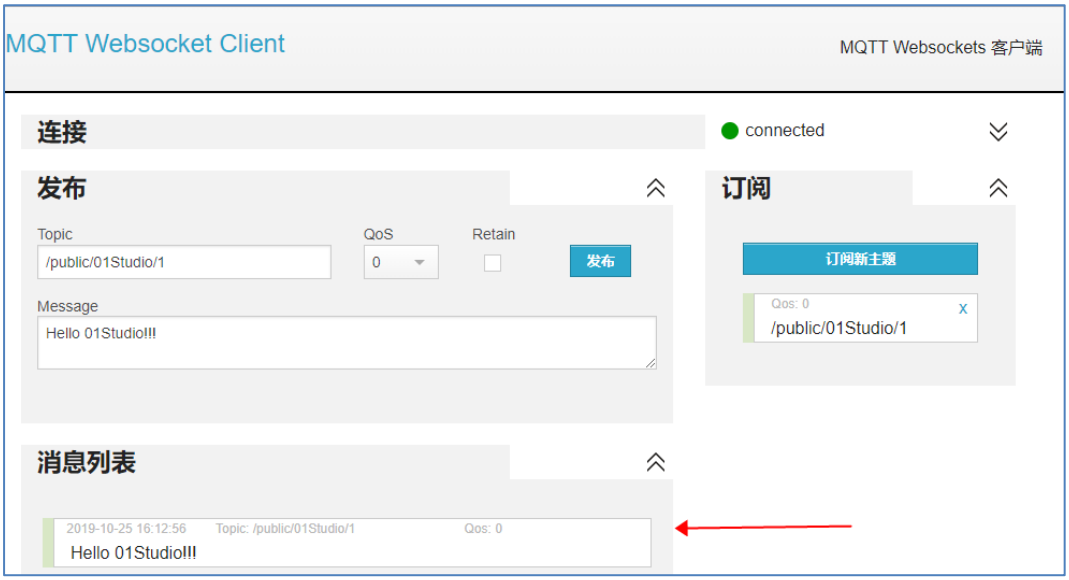


图 10-27 客户端同时发布和订阅信息

- **总结：**

通过本节我们了解了 MQTT 通信原理以及成功实现通信。目前市面上大部分物联网云平台支持 MQTT，原理大同小异。大家可以基于不同平台协议来开发，实现自己的物联网设备远程连接。

10.4 WebREPL

串口的 REPL【读取(Read)-运算(Eval)-输出(Print)-循环(Loop) 】我们已经非常熟悉了使用了。而针对 WIFI 类设备，在有些时候设备已经部署好后不方便通过串口线连接，那么我们就可以通过局域网的方式来调试或者传输文件。以更加便捷的的方式调试产品。例如我们完成了一个小车产品，那么就可以通过 WebREPL 方式来调试。以下是具体方法介绍。

第1步:

确保你的电脑和设备已经连接到同一个网络上（通常指同一个网段，如 192.168.1.X）。具体可以参考本章**连接无线路由器**章节实验。

第2步: 配置开发板

在串口终端输入以下命令：

```
import webrepl_setup
```

在弹出的串口提示框中输入'E'，按回车，使能 WebREPL 相关功能；

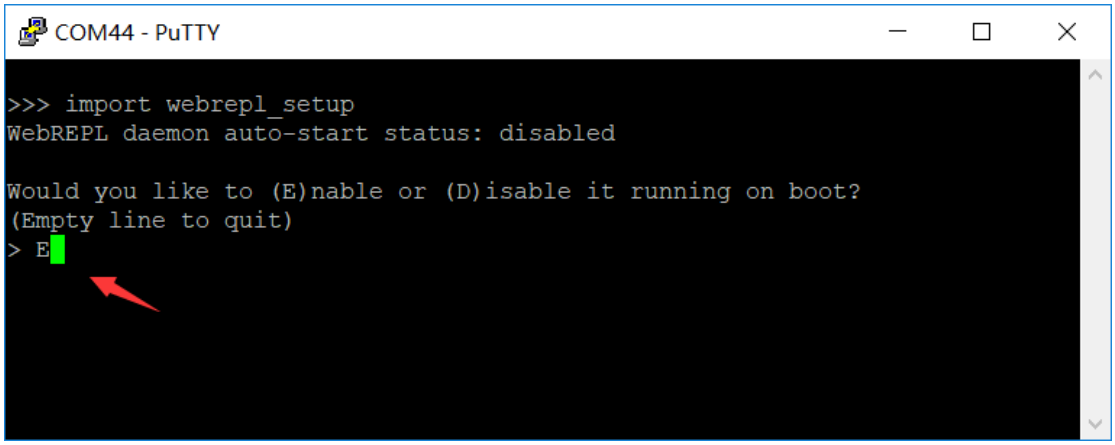


图 10-28

接下来设置密码，设置完成后输入‘y’,选择重启。

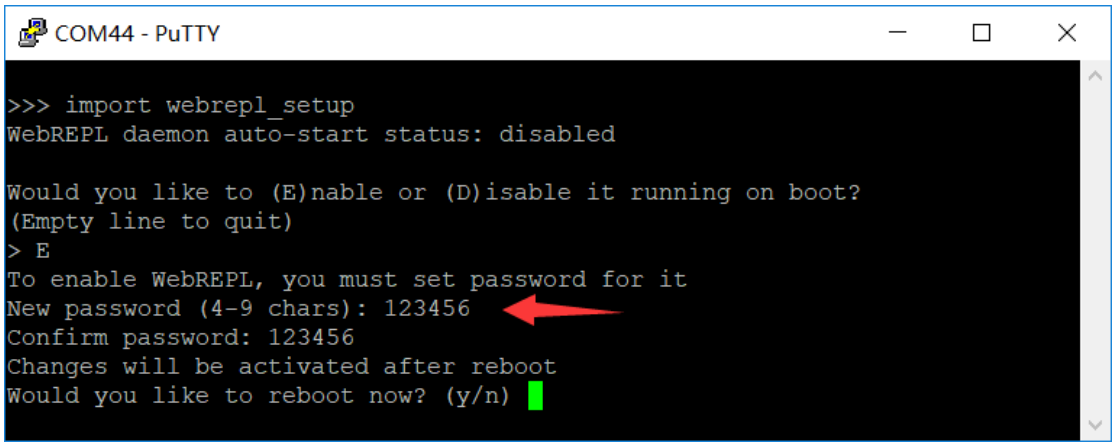


图 10-29

第3步:

打开浏览器，输入 http://webrepl.01studio.org/ 进入，可以见到 WebREPL 界面。

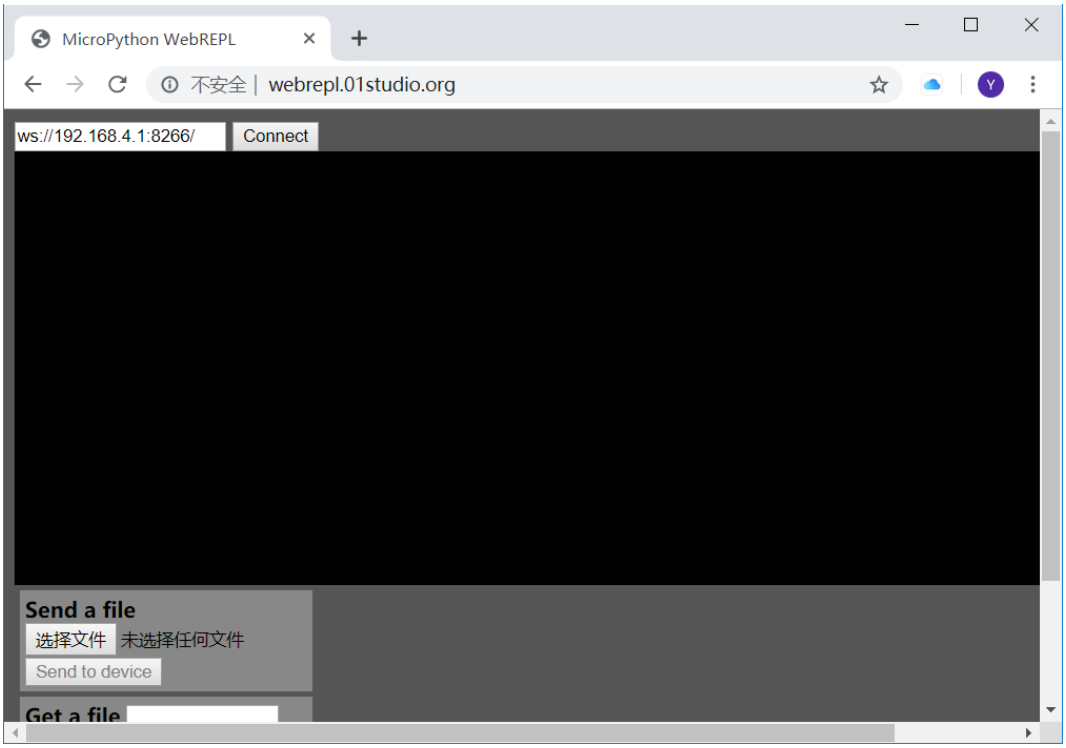


图 10-30 WebREPL

第4步:

在左上角输入 pyWiFi-ESP8266 当前的 IP 地址，端口为 8266。点击 Connect 连接。看到出现密码输入提示：

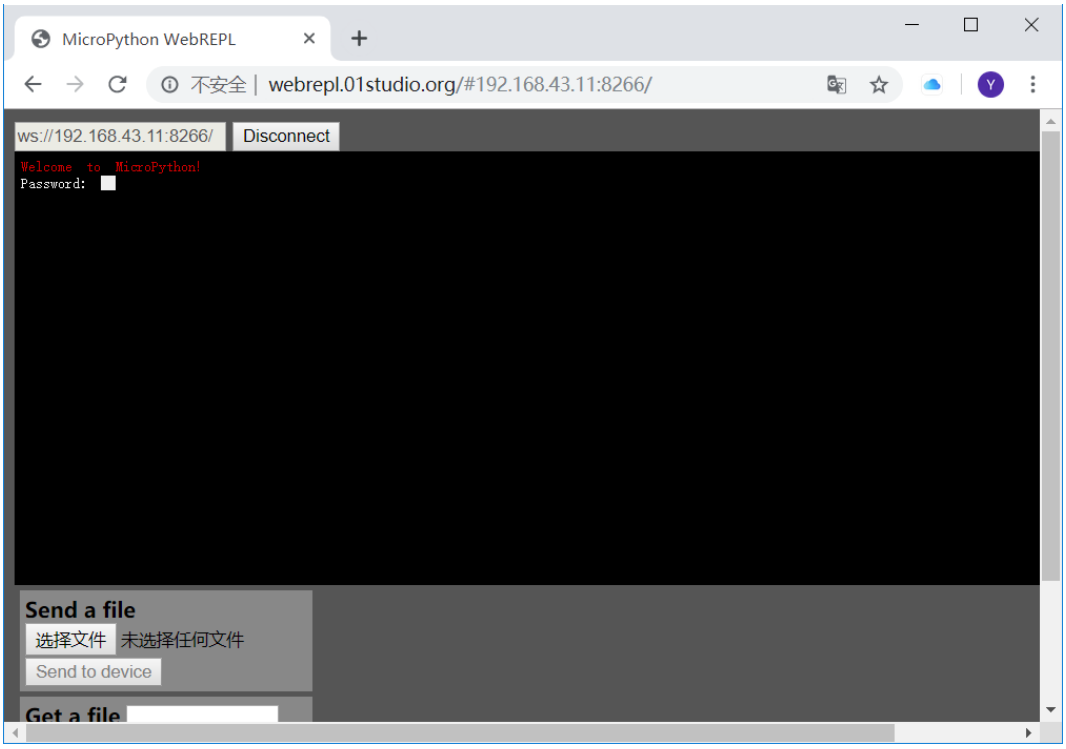


图 10-31

输入之前设置的密码（密码不显示），按回车。连接成功后出现 REPL 交互提示的功能。这是即可以使用跟 REPL 一样的功能。

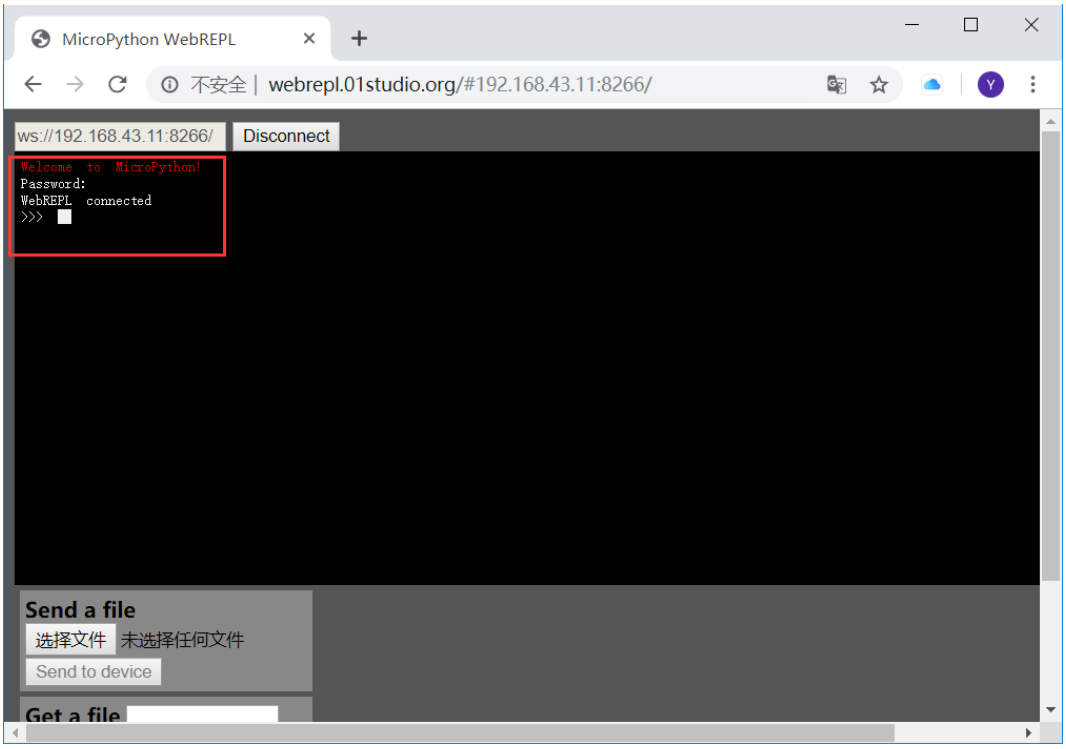


图 10-32　WebREPL 连接成功

第5步： 文件传输

使用 WebREPL 网页下方提供文件传输功能，需要给设备发送文件时候，选择文件并发送。

除此之外也可以使用 Get a file 功能从设备中读取文件，注意要求输入文件名称完整。如“boot.py”。

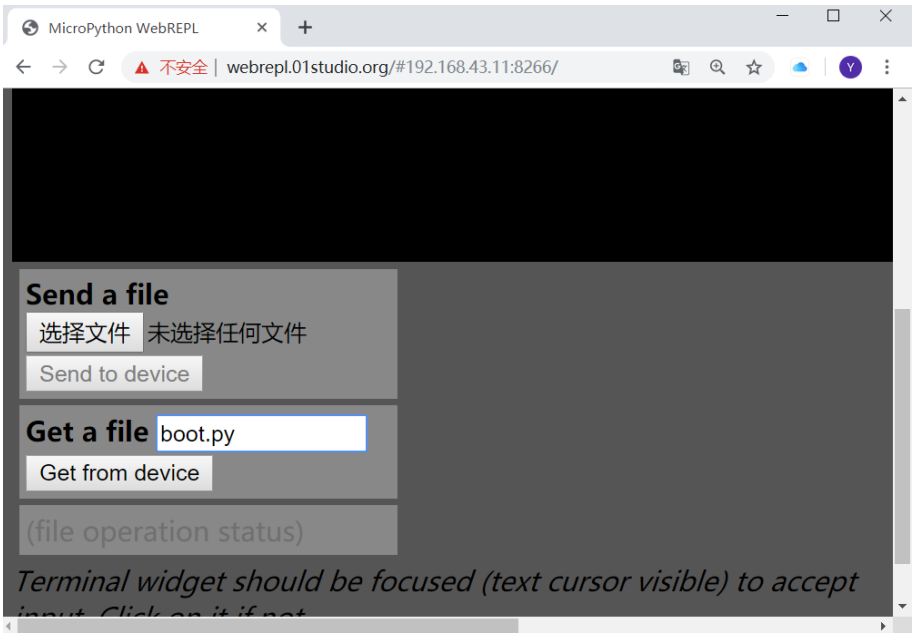
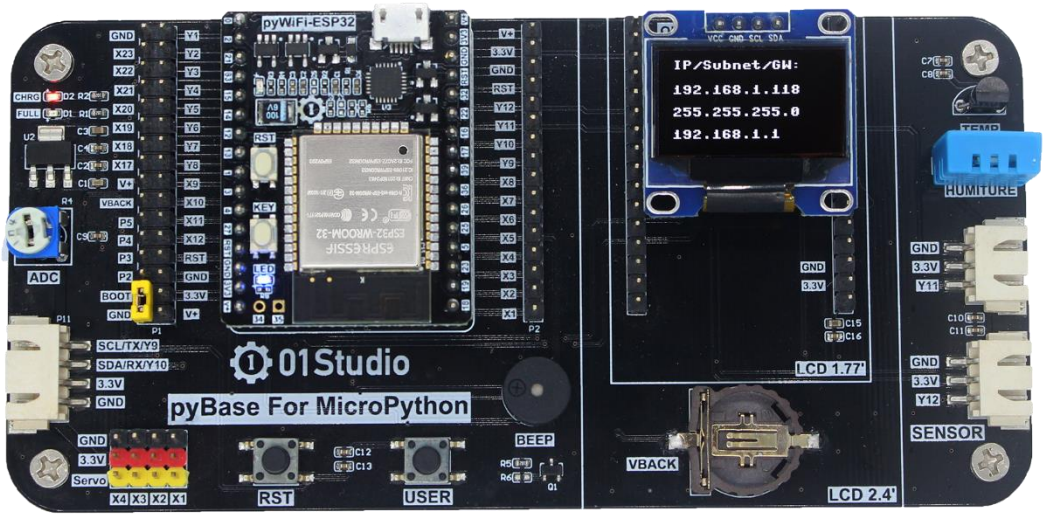


图 10-33　文件收发

第四部分　基于 ESP32 平台

ESP32 是继 ESP8266 后上海乐鑫公司主推的一款芯片，可以理解支持 ESP8266 的超级升级版。相对于 ESP8266 而言，ESP32 芯片的资源非常丰富，除了拥有更多的引脚外设外，还同时支持 WIFI、蓝牙功能。

如果你已经学习过 MicroPython 基于 ESP8266 平台的内容，那么恭喜你，本部分内容的学习将会变得轻松，因为它们的开发环境和编程方式都非常相似。当然拥有更多资源和 ESP32 更适合开发一些高级的实验。让我们开始学习吧！



pyWiFi-ESP32 开发套件

第11章　开发环境快速建立

由于 ESP32 可以看做是 ESP8266 的超级加强版，所以他们的开发环境是相当类似的，为了避免无效的重复，本章部分内容会直接指向基于 ESP8266 平台章节内容，读者结合阅读即可。

11.1　基于 Windows

11.1.1　安装 python3

说明：我们提供安装 python3 教程只是为了在 PC 上体验一下 python3 安装和使用过程，由于 micropython 固件集成了 python 库，所以电脑并非一定要安装，本节可以跳过。

请阅读 **第二部分　3.1.1　安装 python3** 章节内容，这里不再重复。

11.1.2　安装开发软件 Mu

请阅读 **第三部分　7.1.2　安装开发软件 Mu** 章节内容，这里不再重复。

11.1.3　开发套件使用

11.1.3.1　驱动安装

主要是安装 USB 转串口驱动。我们将 pyWiFi-ESP32 开发板通过 MicroUSB 数据线连接到电脑：

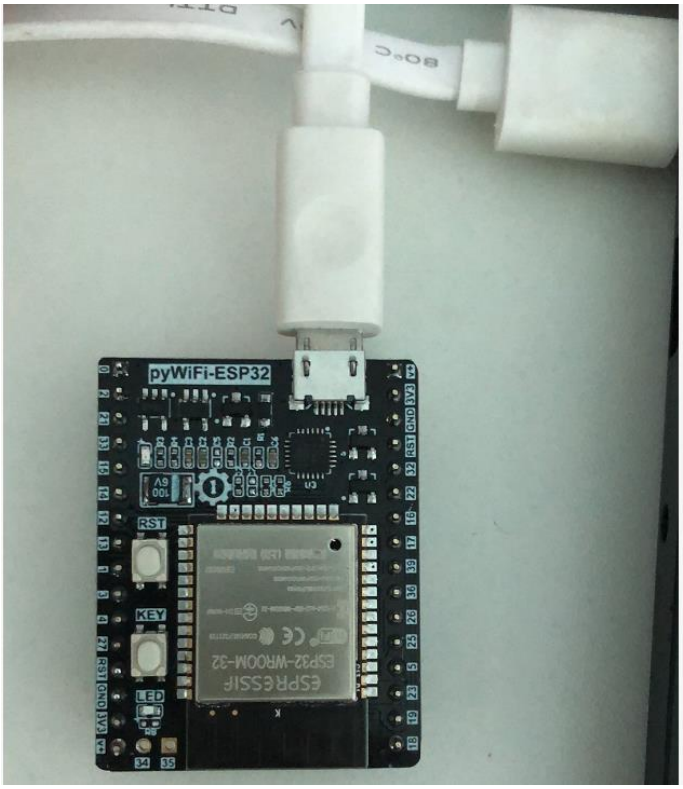


图 11-1 通过 MicroUSB 线连接到电脑

如果你的操作系统是 Win10,一般情况下能自动安装。鼠标右键点击“我的电脑”—属性—设备管理器： 出现串口号说明安装成功，如下图所示。

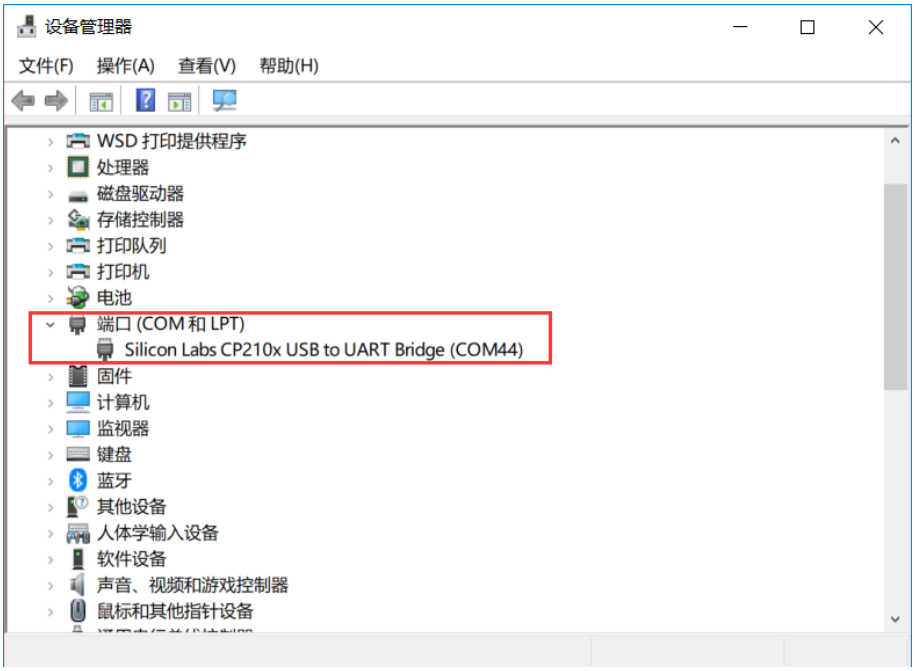


图 11-2 串口驱动成功安装

如果无法安装，请手动安装驱动，方法如下：

不能自动安装时候，设备会出现黄色叹号，这时候点击设备右键，选择“更新驱动程序”，选择“浏览计算机查找驱动”：

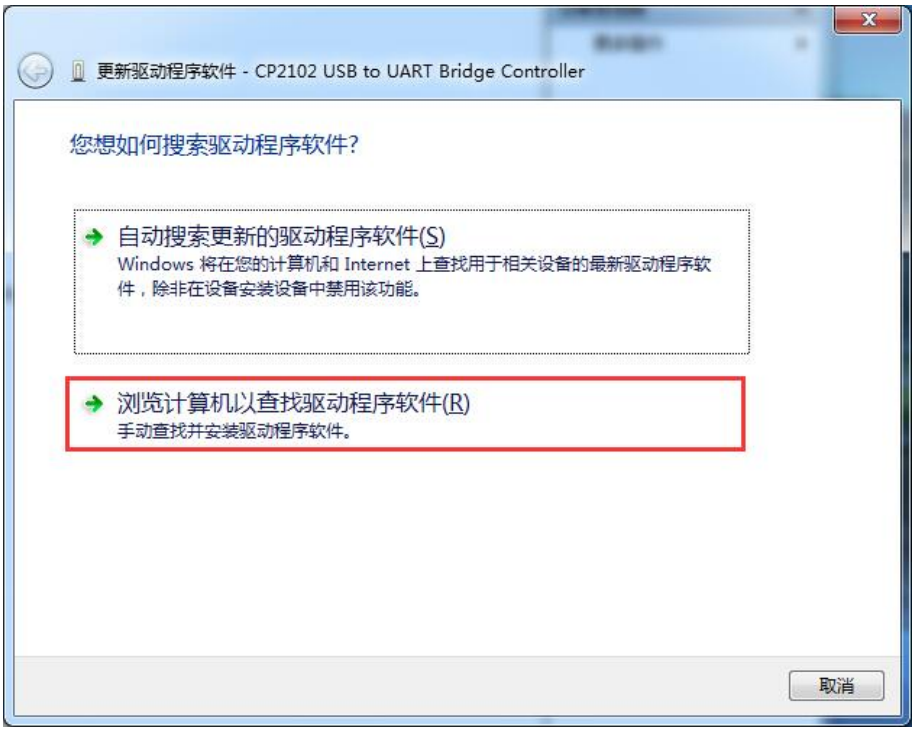


图 11-3

驱动路径选择：零一科技（01Studio）MicroPython 开发套件配套资料\01-开发工具\01-Windows\串口终端工具\CP210x 驱动，点击确认后即可自动安装：

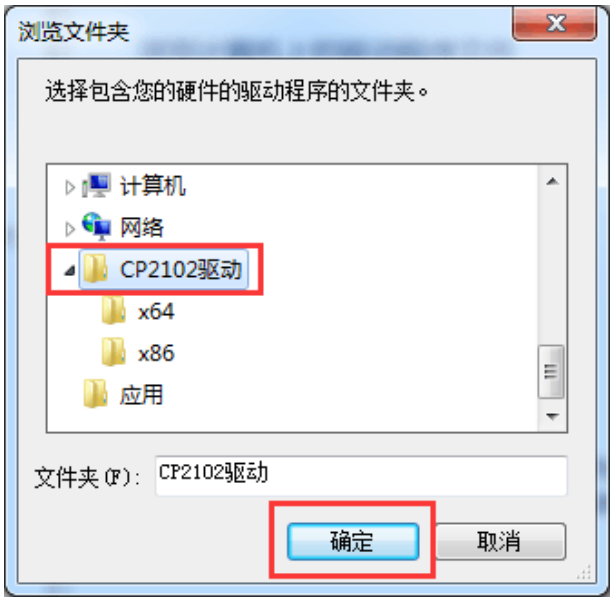


图 11-4

安装成功后如下图：

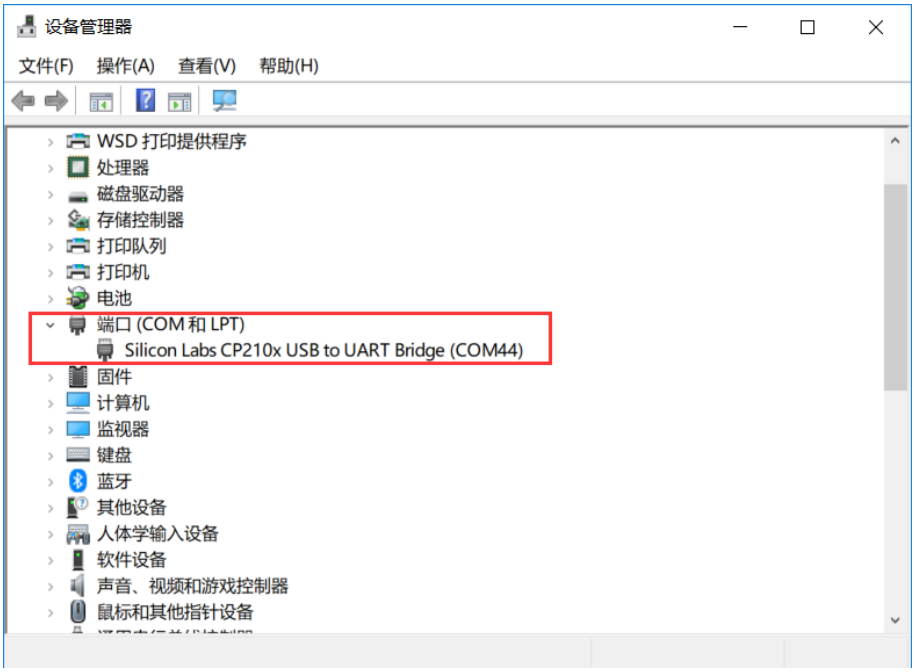


图 11-5 串口驱动安装成功

11.1.3.2 REPL 串口交互调试

请阅读 第二部分 3.1.3.3 REPL 串口交互调试 章节内容，这里不再重复。

需要补充的是 ESP8266 在 Mu 软件下可以直接进入 REPL 串口调试，方法如下：

打开 Mu 软件，连接 pyWiFi-ESP32 开发板，点击上方 REPL 按钮，软件下方即出现 REPL 调试界面：

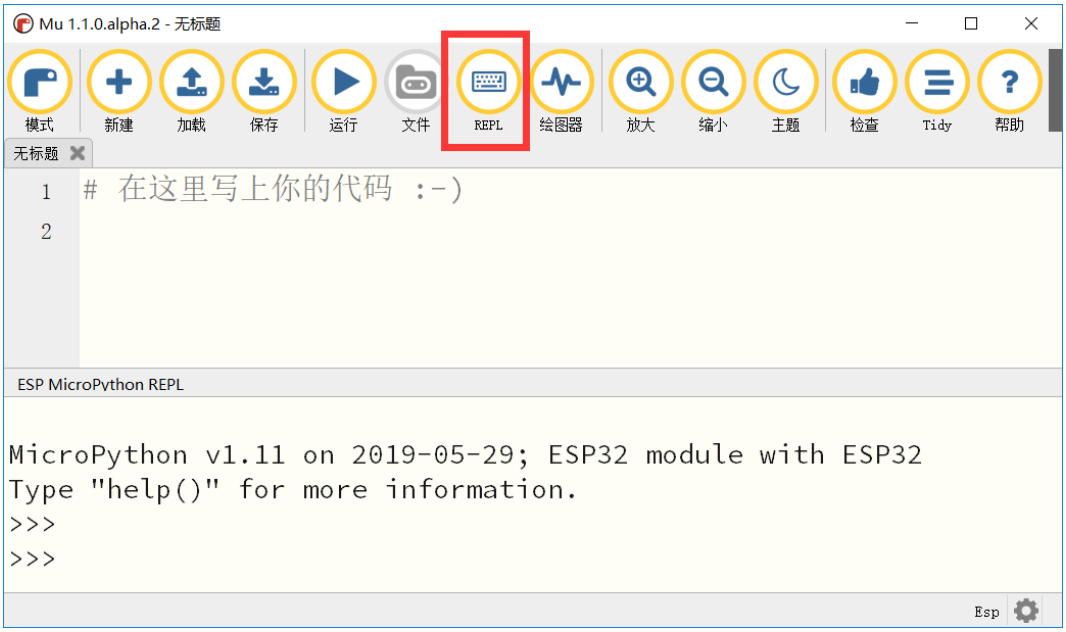


图 11-6 Mu 里面的 REPL

除此之外，还可以使用其它串口软件来进行 REPL 调试：

（1）Putty：位于 MicroPython 开发套件配套资料\开发工具\串口终端工具\Putty.exe；

（2）超级终端：位于 MicroPython 开发套件配套资料\开发工具\Windows\串口终端工具 文件夹下

11.1.3.3 文件系统

pyWiFi-ESP32 里面内置了文件系统，可以简单理解成上电后运行的 python 文件，这个可以通过 Mu 非常方便地读取。

连接开发板，打开 Mu，点击上方“文件”按钮，可以在下方弹出的方框看到开发板里的文件和本地 PC 的文件。

（由于串口是共用的，因此有些时候可能打不开，这时候要先关掉 Mu 里所有文件或者 repl 窗口，按下开发板复位键，再点击打开。）

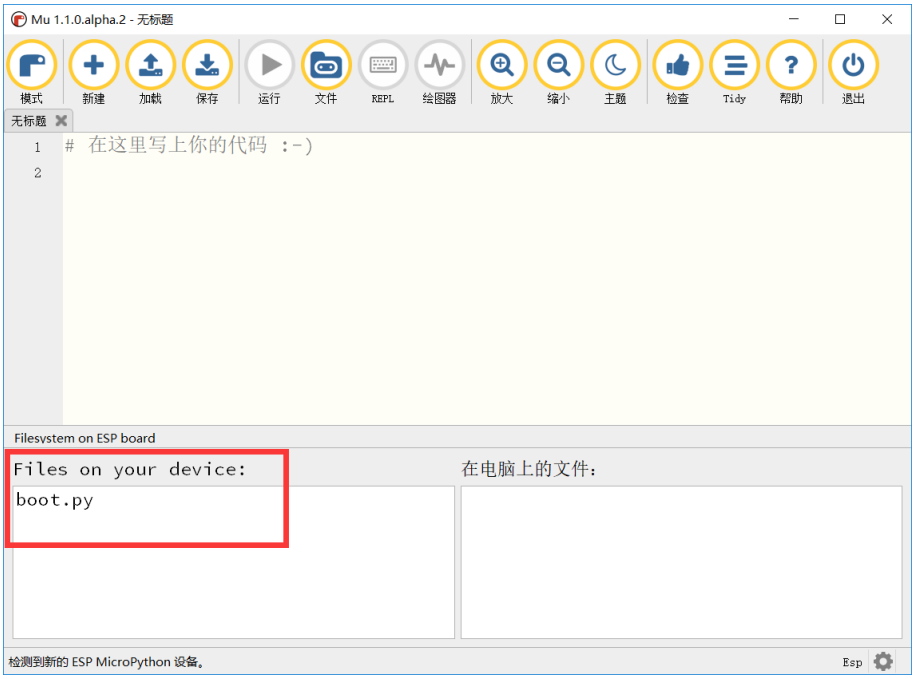


图 11-7

从上图可以看到开发板上的有个“boot.py”文件，这是系统启动后执行的第一个文件，通常做初始化工作。我们尝试将 boot.py 文件拖动到电脑的空白区，拖完后即完成从设备到电脑的文件复制，反过来就是从电脑复制文件到设备，使用非常方便直观。

复制完成后可以点击电脑端文件右键，选择在 Mu 中打开，即可查看和编辑该文件。**（新版固件可能没 boot.py 文件，不影响使用。）**

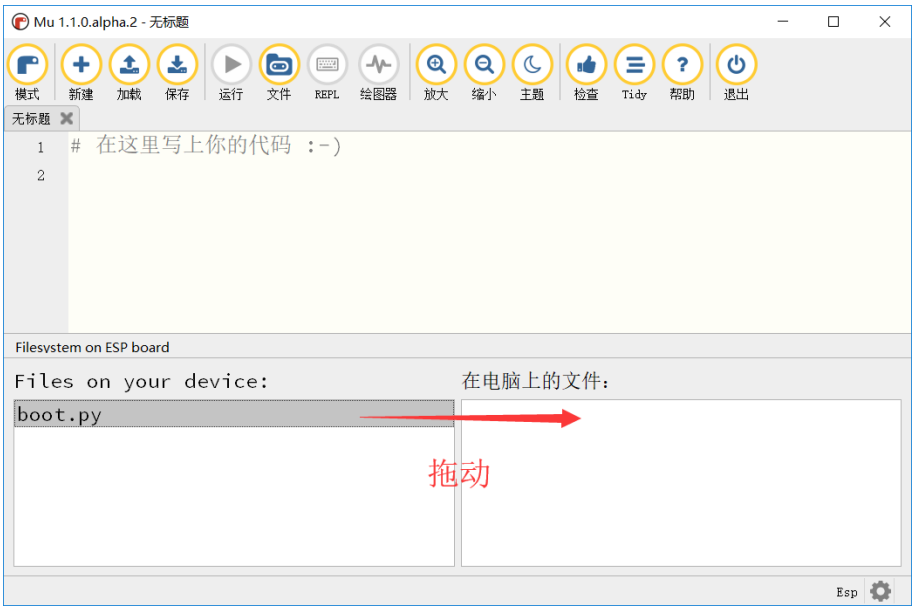


图 11-8

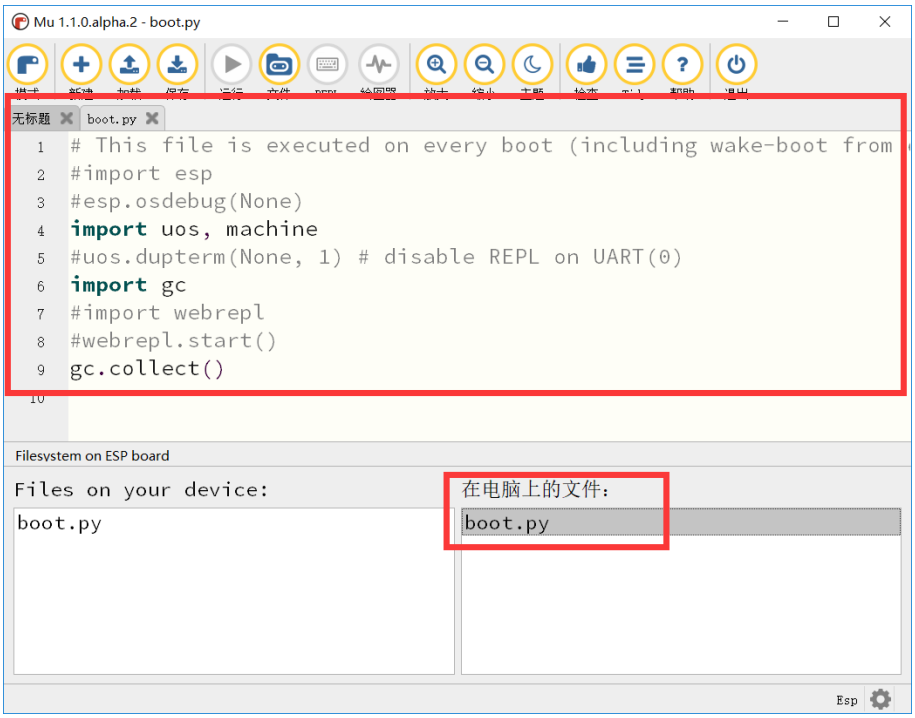


图 11-9 查看本地文件

那么本地文件都放在什么地方呢，可以在 Mu 中双击文件上方名称，即可在弹出的对话框中看到该文件的保存位置。

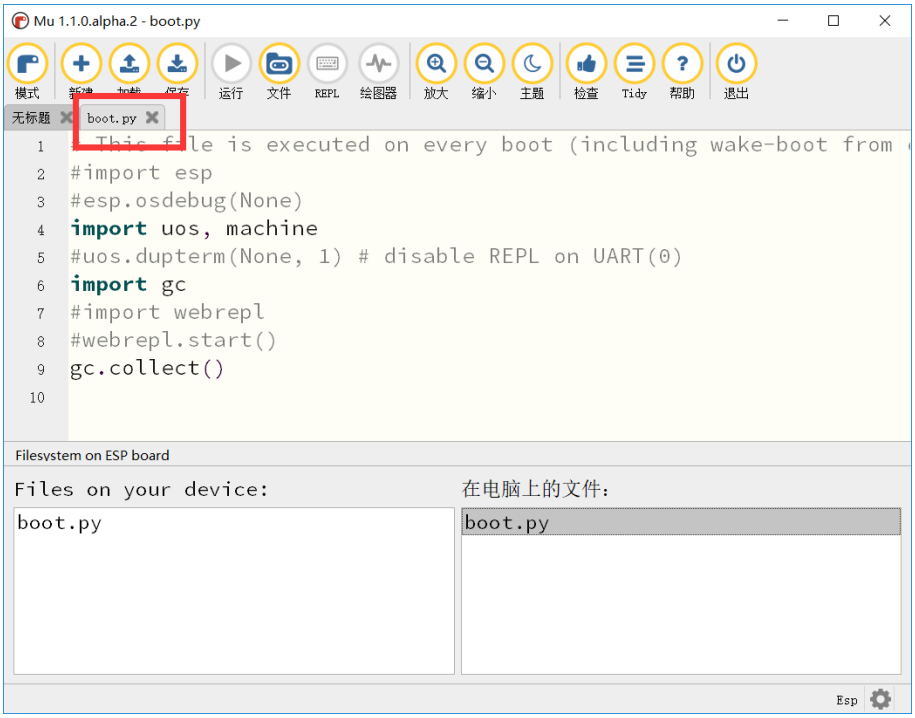

Mu 1.1.0.alpha.2 - boot.py
模式
新建
加载
保存
运行
文件
REPL
绘图器
放大
缩小
主题
检查
Tidy
帮助
退出
无标题
boot.py
This file is executed on every boot (including wake-boot from
#import esp
#esp.osdebug(None)
import uos, machine
#uos.dupterm(None, 1) # disable REPL on UART(0)
import gc
#import webrepl
#webrepl.start()
gc.collect()
Filesystem on ESP board
Files on your device:
boot.py
在电脑上的文件:
boot.py
Esp

图 11-10

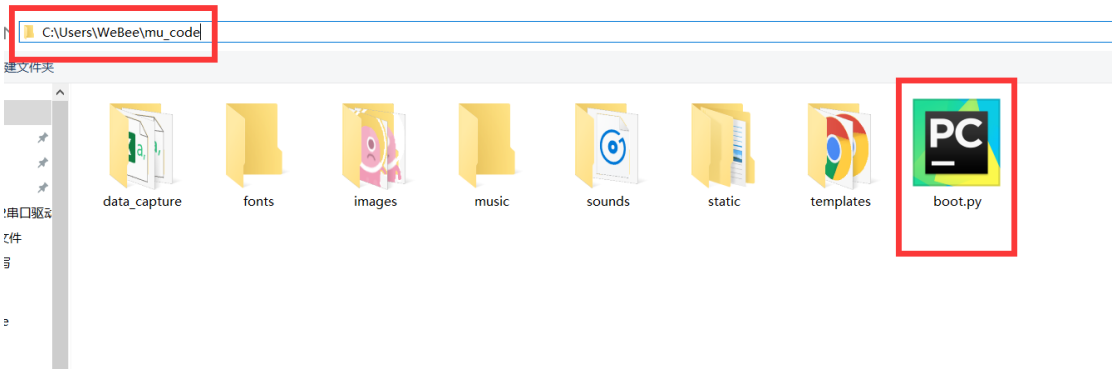


图 11-11 本地文件位置查看

我们在 Mu 中新建一个空白文件，保存名称为 main.py 到刚刚查看到的那个本地文件的路径：



图 11-12

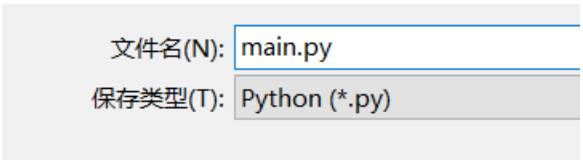


图 11-13

点击【文件】按钮关闭文件下方方框，再次点击【文件】按钮一下，可以发现出现了刚刚保存的 main.py 文件。将 main.py 文件拖动到左边设备栏，可以看到设备栏出现了 main.py 文件，说明复制成功。这就是使用 Mu 实现电脑跟设备文件相互传输的方法。



图 11-14

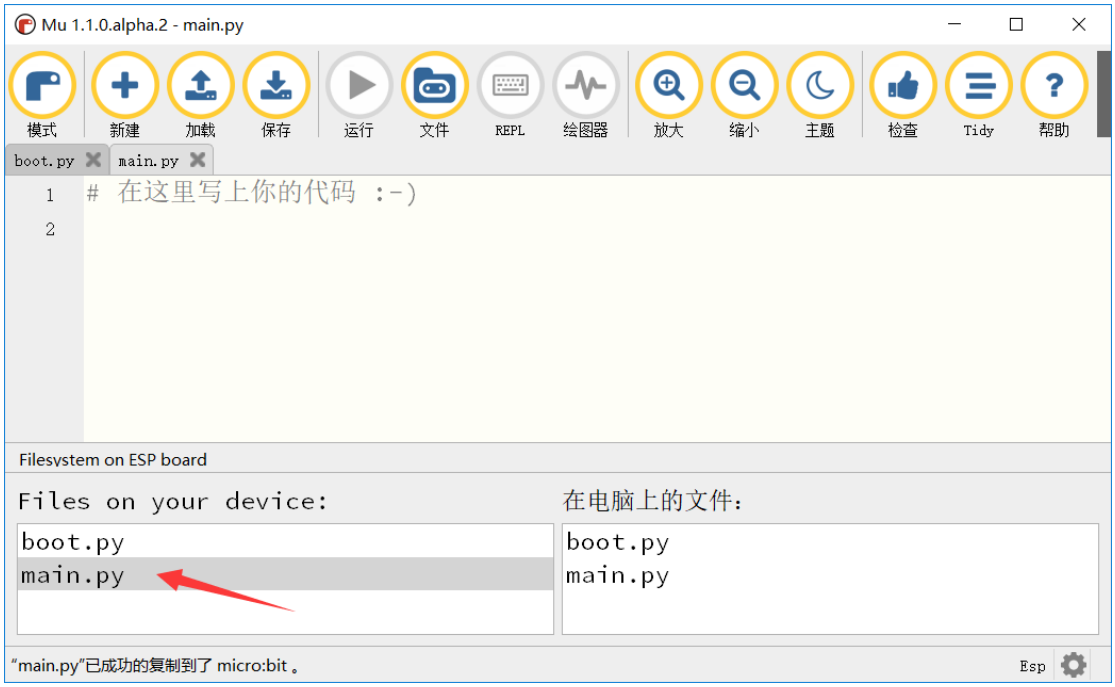


图 11-15 成功复制到设备

main.py 是主文件，是运行完 boot.py 后执行的第一个程序文件，我们主要也是在 main.py 文件里面编程，这在后面的实验会详细讲解。

11.1.3.4 固件更新

我们的开发板出厂已经烧录好了固件，固件更新是指重新烧写开发板的出厂文件或者是升级的固件，烧录使用上海乐鑫提供的官方软件烧录：

找到路径：**零一科技（01Studio）MicroPython 开发套件配套资料_latest\01-开发工具\01-Windows\固件更新工具\flash_download_tools_v3.6.5** 下的 flash_download_tools_v3.6.5.exe 软件，双击打开。

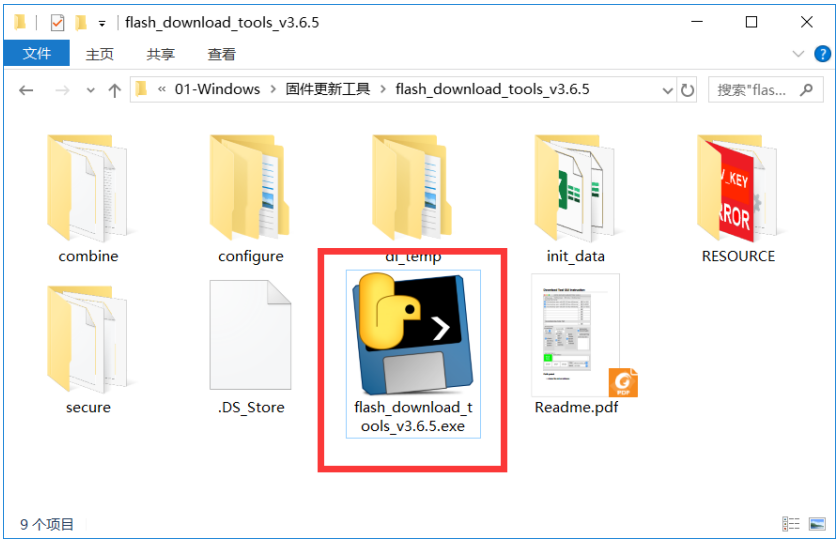


图 11-16

选择 ESP32 Download tool：

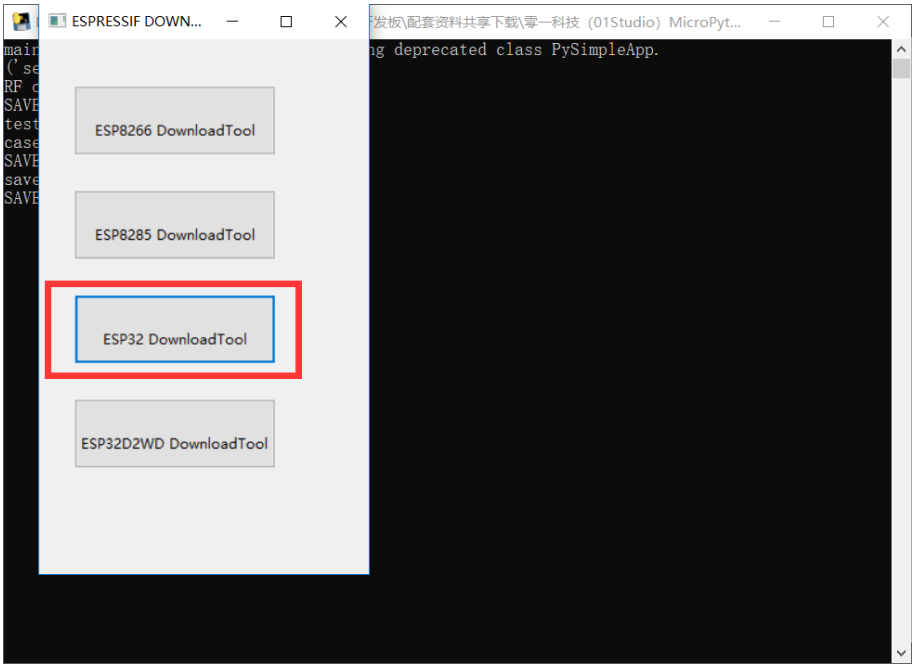


图 11-17 选择 ESP32

选择 SPIDownload，在下图箭头位置点击，选择要烧录固件。固件位置位于：**零一科技（01Studio）MicroPython 开发套件配套资料\03-相关固件\01-pyWiFi-ESP32** 目录下。

其它配置选项也请参考下图，注意 ESP32 的下载地址是 0x1000。（COM 串口是选择自己的串口，在设备管理器查询。）

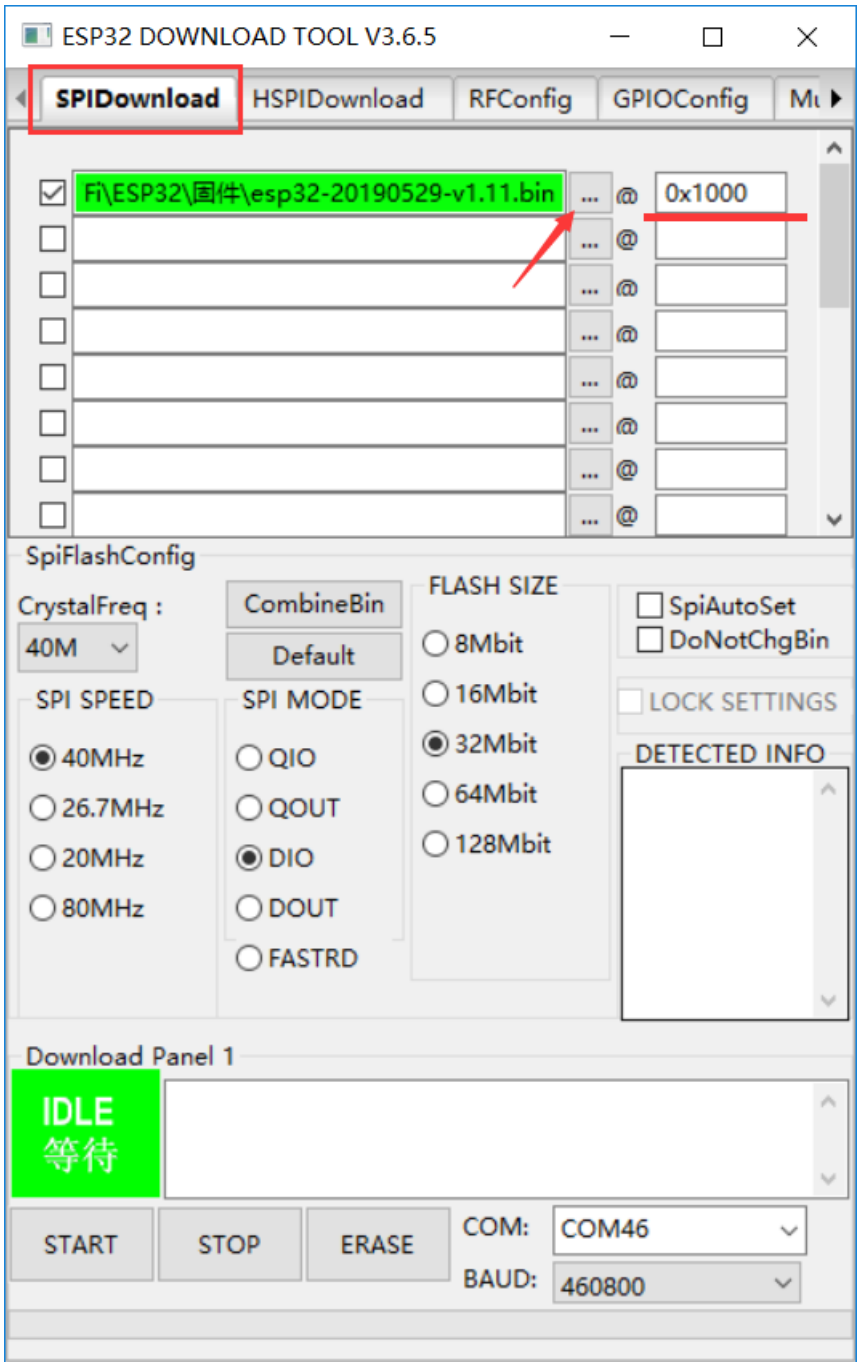


图 11-18 配置

配置好后，先点击“ERASE”按钮刷除模块里面内容。点击软件下方“ERASE”按钮，刷除成功后，左边绿色框出现完成字样。

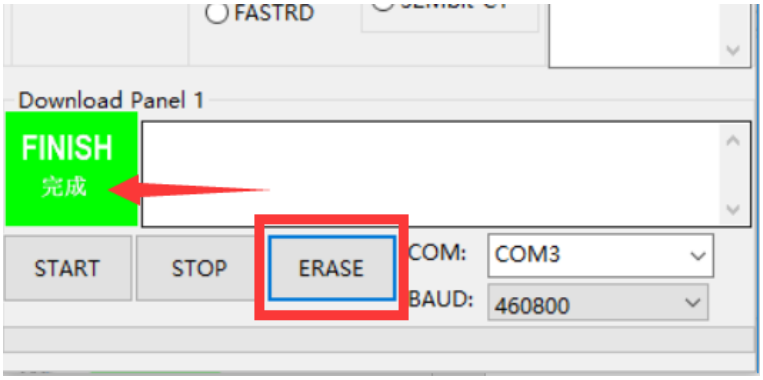


图 11-19 刷除 flash 成功

刷除成功后，点击“START”按钮开始烧录，烧录完成有左边绿色框出现“完成”字样。完成后记得点”stop”按钮或者关闭软件释放串口。

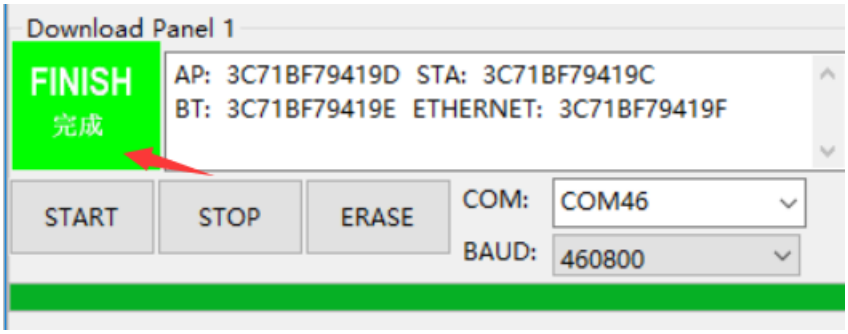


图 11-20 烧录成功

11.2 基于 Mac OS

由于 Mac OS 中 ESP32 平台和 ESP8266 平台的开发环境并无差异，所以用户可以直接参阅 **第三部分 7.2 基于 Mac OS** 章节内容，这里不再重复。

需要注意的是 ESP32 下载固件指令和 ESP8266 区别如下：（ESP32 固件从 0x1000 地址开始烧录。）

```
esptool.py --chip esp32 --port /dev/tty.SLAB_USBtoUART write_flash -z
0x1000 esp32-20180511-v1.9.4.bin
```

烧录成功如下图：

```
MacBook-Air:Desktop jackey$ esptool.py --chip esp32 --port /dev/tty.SLAB_USBtoUART write_fla
sh -z 0x1000 esp32-20190529-v1.11.bin
esptool.py v2.6
Serial port /dev/tty.SLAB_USBtoUART
Connecting........_
Chip is ESP32D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme None
MAC: a4:cf:12:5f:11:54
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 1146864 bytes to 717504...
Wrote 1146864 bytes (717504 compressed) at 0x00001000 in 63.3 seconds (effective 144.9 kbit/s)...
Hash of data verified.
```

图 11-21 固件烧录成功

另外就是 ESP32 固件没有固件检测功能，因此 **esp.check_fw()**指令无效。

11.1 基于 Linux

基于 Linux 平台开发环境搭建请阅读 **第二部分 3.3 基于 Linux** 章节内容，这里不再重复。

第12章　基础实验

MicroPython 更强调的是针对应用的学习，强大的底层库函数让我们可以直接关心功能的实现，也就是说我们只要理解和熟练相关的函数用法，就可以很好的玩转 MicroPython。它让我们可以做到不关心硬件和底层原理（当然有兴趣和能力的小伙伴可以深入研究）而直接跑起硬件。

基础实验是针对一些简单的实验学习讲解，可以让我们更快和更直接感受到 MicroPython 针对嵌入式开发的强大之处， 我后面的学习和开发夯实基础。

请先看实验讲解格式预览，每一节我们都会以以下形式讲解，图文并茂，务求达到快速理解和学习的作用：

（1） 前言：简单介绍这个实验；

（2） 实验平台：实验所用到的开发板、配件和接线说明；

（3） 实验目的：本实验要实现的功能；

（4） 实验讲解：对函数、代码、编程方法以及实验的详细讲解；

（5） 实验结果：记录程序下载到开发板上的图片示例；

（6） 总结:对实验进行总结。

12.1 点亮第一个 LED

- **前言：**

相信大部分人开始学习嵌入式单片机编程都会从点亮 LED 开始，我们 MicroPython 的学习也不例外，通过点亮第一个 LED 能让你对编译环境和程序架构有一定的认识，为以后的学习和更大型的程序打下基础，增加信心。

- **实验平台：**

pyWiFi-ESP32。

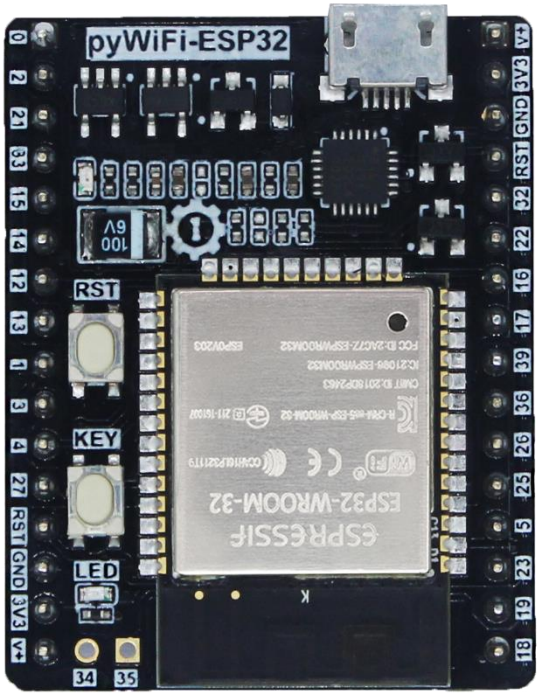


图 12-1 pyWiFi-ESP32

- **实验目的：**

点亮 LED（蓝灯）。

- **实验讲解：**

pyWiFi-ESP32 上有 1 个 LED（蓝色），控制 LED 使用 machine 中的 Pin 对象，其构造函数和使用方法如下：

构造函数
led=machine.Pin(id,mode,pull)
构建 led 对象。id:引脚编号；mode:输入输出方式；pull:上下拉电阻配置。
使用方法
led.value([x])
引脚电平值。输出状态：x=0 表示低电平，x=1 表示高电平；输入状态：无须参数，返回当前引脚值。
led.on()
使引脚输出高电平“1”。
led.off()
使引脚输出低电平“0”。

表 12-1　Pin 对象

上表对 MicroPython 的 machine 中 Pin 对象做了详细的说明，machine 是大模块，Pin 是 machine 下面的其中一个小模块，在 python 编程里有两种方式引用相关模块:

方式 1 是：import machine，然后通过 machine.Pin 来操作；

方式 2 是：from machine import Pin,意思是直接从 machine 中引入 Pin 模块，然后直接通过构建 led 对象来操作。显然方式 2 会显得更直观和方便，本实验也是使用方式 2 来编程。代码编写流程如下：

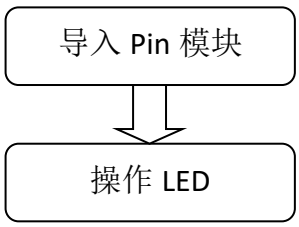


图 12-2 代码编写流程

LED 跟模块引脚 2 相连，通过输出高电平方式点亮。

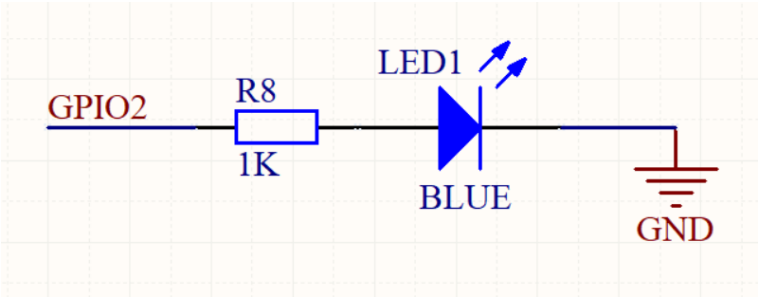


图 12-3　LED 接线图

参考代码如下 ：

```
'''
实验名称：点亮 LED 蓝灯
版本：v1.0
日期：2019.7
作者：01Studio
'''

from machine import Pin #导入 Pin 模块
led=Pin(2,Pin.OUT) #构建 led 对象，GPIO2,输出
led.value(1) #点亮 LED，也可以使用 led.on()
```

运行程序有两个方法：

方法一：

编写好代码后点击 Mu 上方的“运行”按钮，可以直接观察到代码运行情况。这个方法不会将程序代码保存到 ESP32 模块的 flash 里面。这注意是方便调试使用。

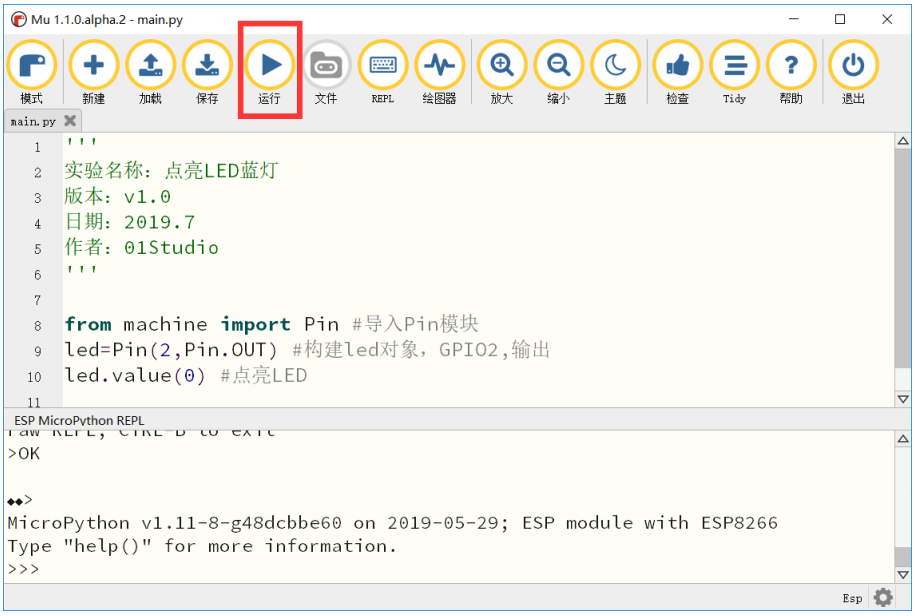


图 12-4 运行仿真

方法二：

将新建的文件保存名称为 “main.py” 的 py 文件，使用 Mu 的文件功能，从本地拖动到设备。然后按下复位按键，设备运行相关代码，这个方式相当于将程序烧录到设备 flash，可以脱机使用。 操作方法参考：**11.1.3.3 文件系统** 章节内容。

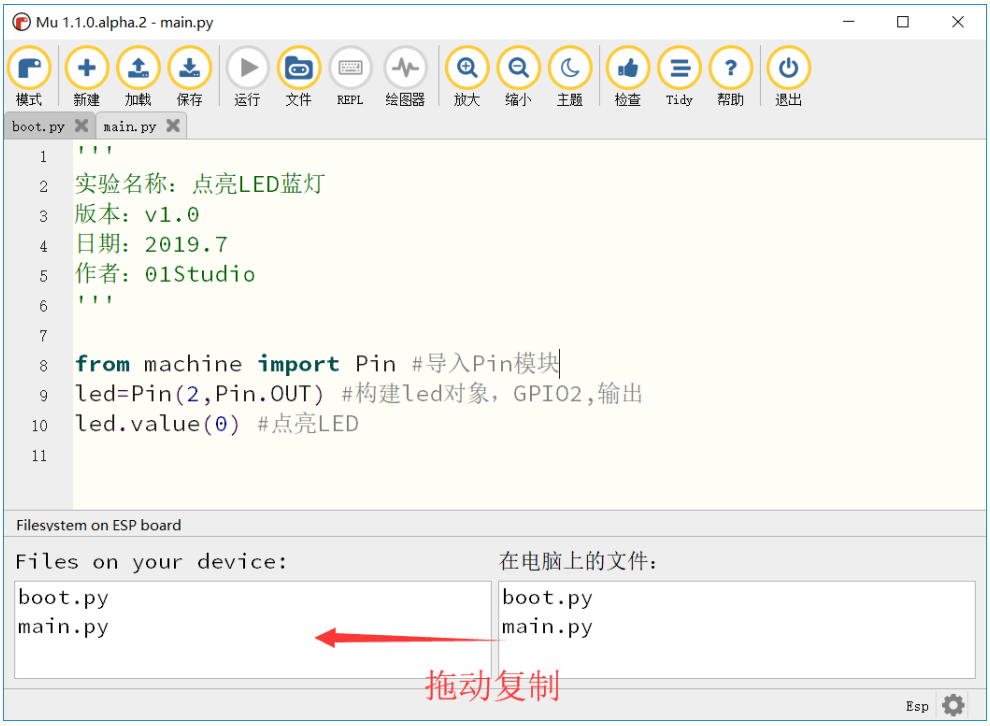


图 12-5 拷贝文件到设备

- **实验结果：**

下载程序，可以看到 LED（蓝灯）成功点亮。



图 12-6

- **总结：**

从第一个实验我们可以看到，使用 MicroPython 来开发关键是要学会构造函数和其使用方法，便可完成对相关对象的操作，在强大的模块函数支持下，实验只用了简单的两行代码便实现了点亮 LED 灯。

12.2 按键

- **前言：**

按键是最简单也最常见的输入设备，很多产品都离不开按键，包括早期的iphone，今天我们就来学习一下如何使用 MicroPython 来编写按键程序。有了按键输入功能，我们就可以做很多好玩的东西了。

- **实验平台：**

pyWiFi-ESP32。

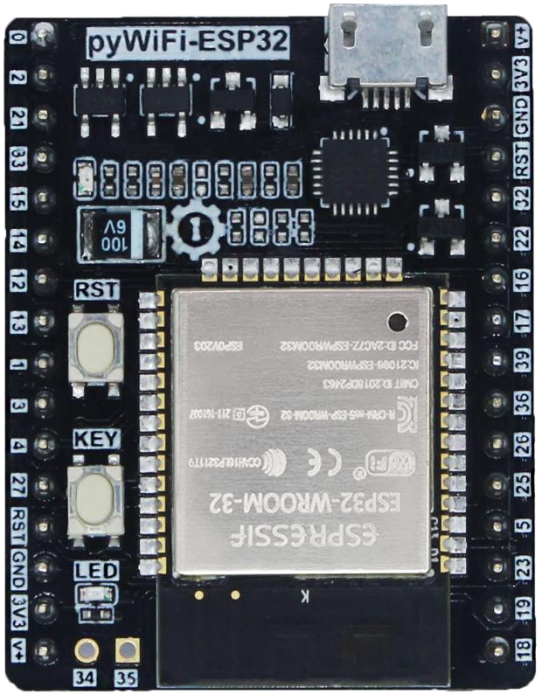


图 12-7 pyWiFi-ESP32

- **实验目的：**

使用按键功能，通过检测按键被按下后，改变 LED（蓝灯）的亮灭状态。

- **实验讲解：**

pyWiFi-ESP32 开发板上有 2 个按键，RST 和 KEY，RST 顾名思义是复位用的，所以真正自带可以用的就只有 1 个按键 KEY。

让我们先来搞清楚 MicroPython 里面 Pin 模块实现按键的构造函数和使用方法。

构造函数
KEY=machine.Pin(id,mode,pull)
构建按键对象。id:引脚编号；mode:输入输出方式；pull:上下拉电阻配置。
使用方法
KEY.value()
引脚电平值。输入状态：无须参数，返回当前引脚值 0 或者 1。

表 12-2　KEY 对象

可以看到跟上一节 LED 一样，只是输入/输出状态的一个改变。从下面原理图可以看到，我们只需要在开发板上电后判断 KEY 引脚的电平，当被按下时候引脚为低电平“0”。

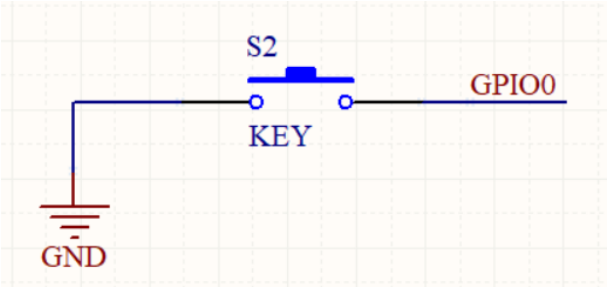


图 12-8 按键原理图

按键被按下时候可能会发生抖动，抖动如下图，有可能造成误判，因此我们需要使用延时函数来进行消抖：

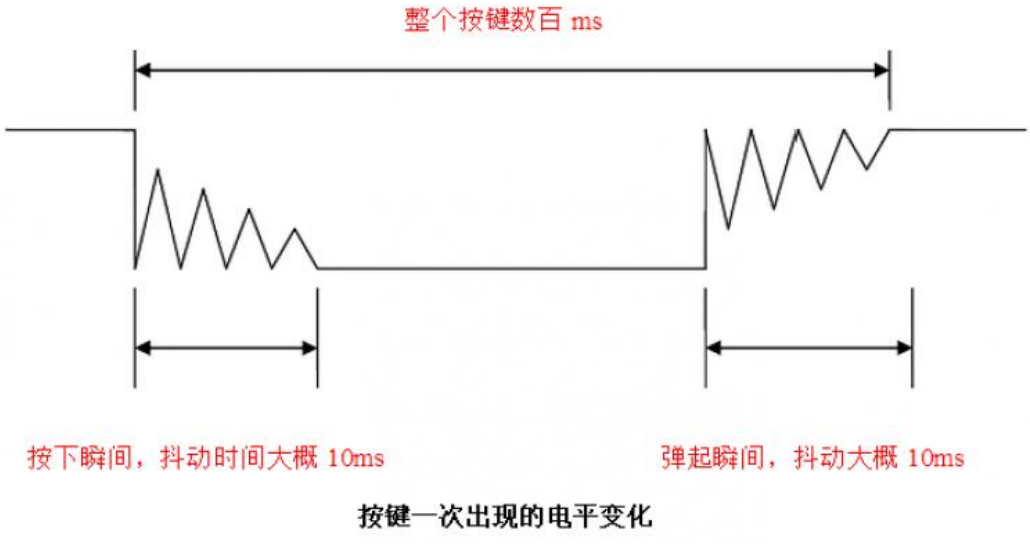


图 12-9 按键按下过程

常用的方法就是当检测按键值为 0 时，延时一段时间，大约 10ms，再判断按键引脚值仍然是 0，是的话说明按键被按下。延时使用 time 模块，使用方法如下：

```
import time

time.sleep(1)           # 睡眠 1 秒
time.sleep_ms(500)      # 睡眠 500 毫秒
time.sleep_us(10)       # 睡眠 10 微妙
start = time.ticks_ms() # 获取毫秒计时器开始值

delta = time.ticks_diff(time.ticks_ms(), start) # 计算从上电开始到当前时间的差值
```

编程流程图如下：

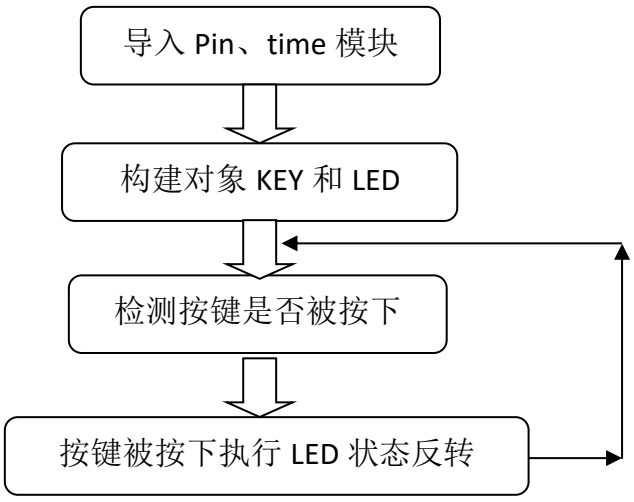


图 12-10 代码编写流程

实验参考代码如下：

```
'''
实验名称：按键
版本：v1.0
日期：2019.8
作者：01Studio
说明：通过按键改变 LED 的亮灭状态
'''
from machine import Pin
import time

LED=Pin(2,Pin.OUT) #构建 LED 对象,开始熄灭
KEY=Pin(0,Pin.IN,Pin.PULL_UP) #构建 KEY 对象
state=0 #LED 引脚状态

while True:
    if KEY.value()==0:   #按键被按下
        time.sleep_ms(10) #消除抖动
        if KEY.value()==0: #确认按键被按下
            state=not state  #使用 not 语句而非~语句
            LED.value(state) #LED 状态翻转
            while not KEY.value(): #检测按键是否松开
                pass
```

从上面代码可以看到，初始化各个对象后，进入循环，当检测到 KEY 的值为 0（按键被按下）时候，先做了 10ms 的延时，再次判断；

state 为 LED 状态的值，每次按键按下后通过使用 not 来改变。这里注意的是在 python 里使用‘not’而不是‘~’的方式。not 返回的是 True 和 False，即 0,1。而~ 是取反操作，会导致出错。

- **实验结果：**

将以上代码编写到 main.py 文件，拷贝到设备。复位后即可运行程序。可以看到每当按键 KEY 被按下后，LED 的亮灭状态发生改变。如下图所示：

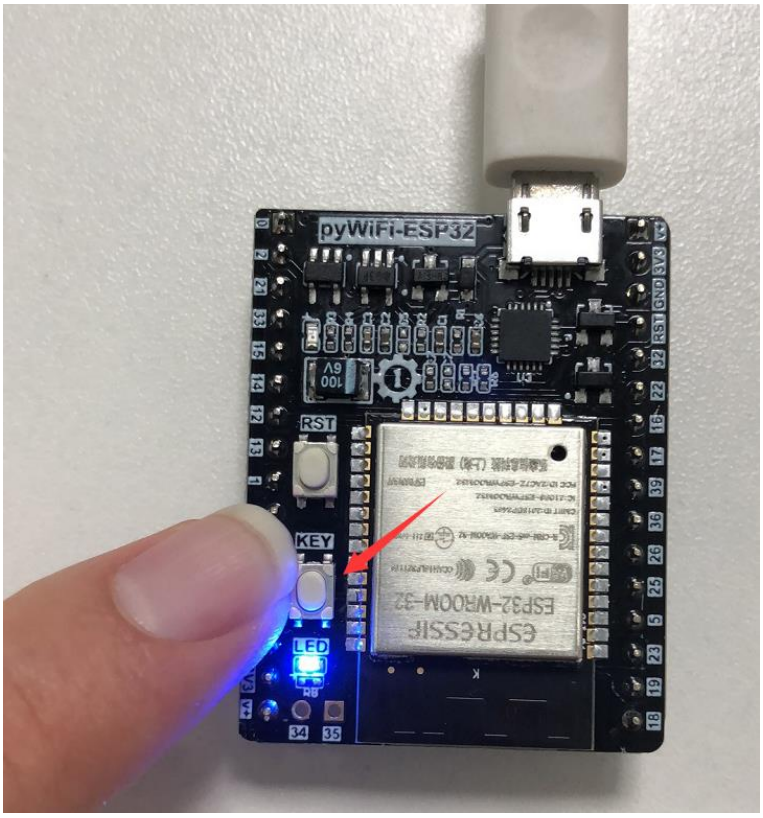


图 12-11

- **总结：**

按键作为我们学习的第一个输入设备，有了输入设备我们就可以跟硬件做人机交互了，这对后面的学习非常有意义。可以看到按键在 MicroPython 下开发也显得很简单。

12.3 外部中断

- **前言：**

前面我们在做普通的 GPIO 时候，虽然能实现 IO 口输入输出功能，但代码是一直在检测 IO 输入口的变化，因此效率不高，特别是在一些特定的场合，比如某个按键，可能 1 天才按下一次去执行相关功能，这样我们就浪费大量时间来实时检测按键的情况。

为了解决这样的问题，我们引入外部中断概念，顾名思义，就是当按键被按下(产生中断)时，我们才去执行相关功能。这大大节省了 CPU 的资源，因此中断的在实际项目的应用非常普遍。

- **实验平台：**

pyWiFi-ESP32 开发板。

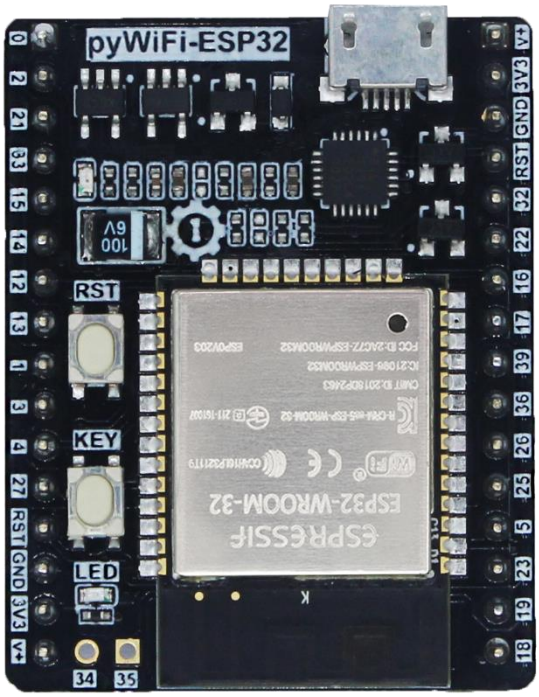


图 12-12　pyWiFi-ESP32

- **实验目的：**

利用中断方式来检查按键 KEY 状态，被按键被按下（产生外部中断）后使 LED 的亮灭状态翻转。

- **实验讲解：**

外部中断也是通过 machine 模块的 Pin 子模块来配置，我们先来看看其配构造函数和使用方法：

构造函数
KEY=machine.Pin(id,mode,pull)
构建按键对象。id:引脚编号；mode:输入输出方式；pull:上下拉电阻配置。
使用方法
KEY.irq(handler,trigger)
配置中断模式。 handler:中断执行的回调函数； trigger: 触发中断的方式，共 4 种，分别是 Pin.IRQ_FALLING（下降沿触发）、Pin.IRQ_RISING（上升沿触发）、Pin.IRQ_LOW_LEVEL（低电平触发）、Pin.IRQ_HIGH_LEVEL（高电平触发）。

表 12-3 按键对象

上升沿和下降沿触发统称边沿触发。从上一节按键可以看到，按键被按下时一个引脚值从 1 到 0 变化的过程，边沿触发就是指这个过程。

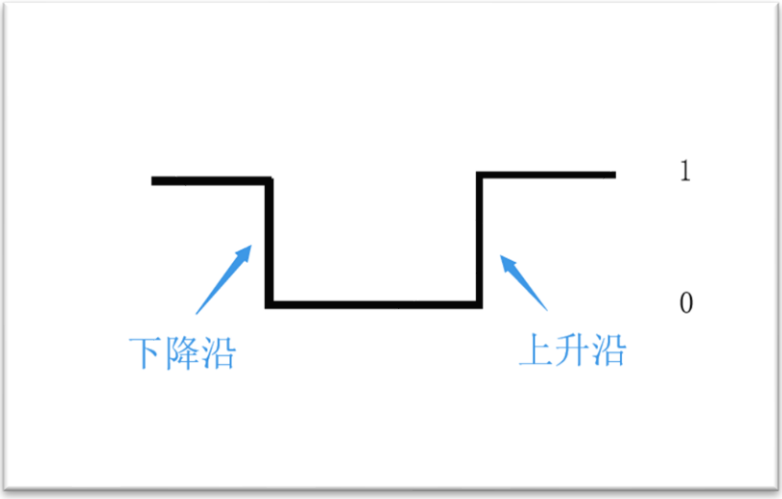


图 12-13 边沿触发

由此可见，我们可以选择下降沿方式触发外部中断，也就是当按键被按下的时候立即产生中断。

编程思路中断跟按键章节类似，在初始化中断后，当系统检测到外部终端时候，执行 LED 亮灭状态反转的代码即可。

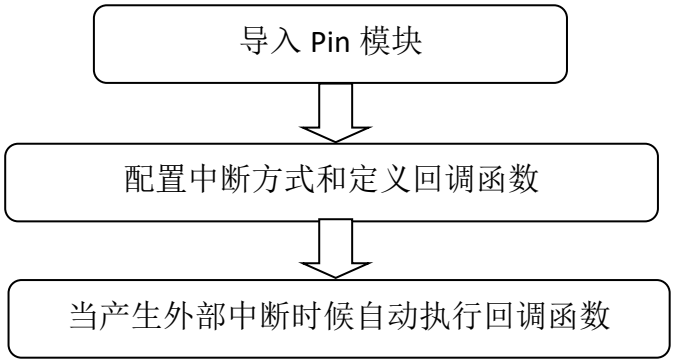


图 12-14 代码编写流程

参考程序代码如下：

```
'''
实验名称：外部中断
版本：v1.0
日期：2019.8
作者：01Studio
说明：通过按键改变 LED 的亮灭状态（外部中断方式）
'''

from machine import Pin
import time

LED=Pin(2,Pin.OUT) #构建 LED 对象,开始熄灭
KEY=Pin(0,Pin.IN,Pin.PULL_UP) #构建 KEY 对象
state=0  #LED 引脚状态

#LED 状态翻转函数
def fun(KEY):
```

```
    global state
    time.sleep_ms(10) #消除抖动
    if KEY.value()==0: #确认按键被按下
        state = not state
        LED.value(state)

KEY.irq(fun,Pin.IRQ_FALLING) #定义中断，下降沿触发
```

以上代码中需要注意的地方：

1、state 是全局变量，因此在 fun 函数里面用该变量必须添加 global state 代码，否则会在函数里面新建一个样的变量造成冲突。

2、在定义回调函数 fun 的时候，需要将 Pin 对象 KEY 传递进去。

- **实验结果：**

将 main.py 文件拷贝到设备，复位。

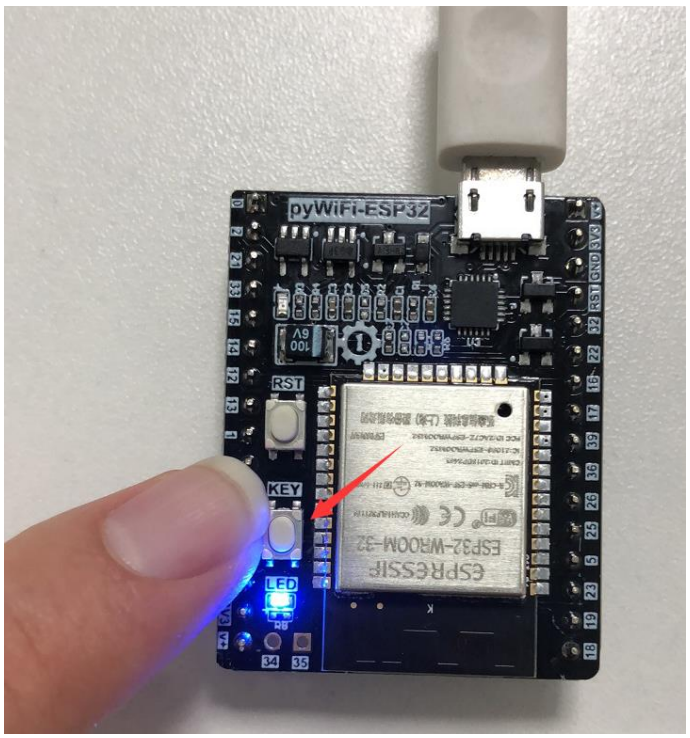


图 12-15 实验现象

- **总结：**

从参考代码来看，只是用了几行代码就实现了实验功能，而且相对于使用 while True 实时检测函数来看，代码的效率大大增强。外部中断的应用非常广，出来普通的按键输入和电平检测外，很大一部分输入设备，比如传感器也是通过外部中断方式来实时检测，这个在后面的章节会讲述。

12.4 定时器

- **前言：**

定时器，顾名思义就是用来计时的，我们常常会设定计时或闹钟，然后时间到了就告诉我们要做什么了。单片机也是这样，通过定时器可以完成各种预设好的任务。

- **实验平台：**

pyWiFi-ESP32。

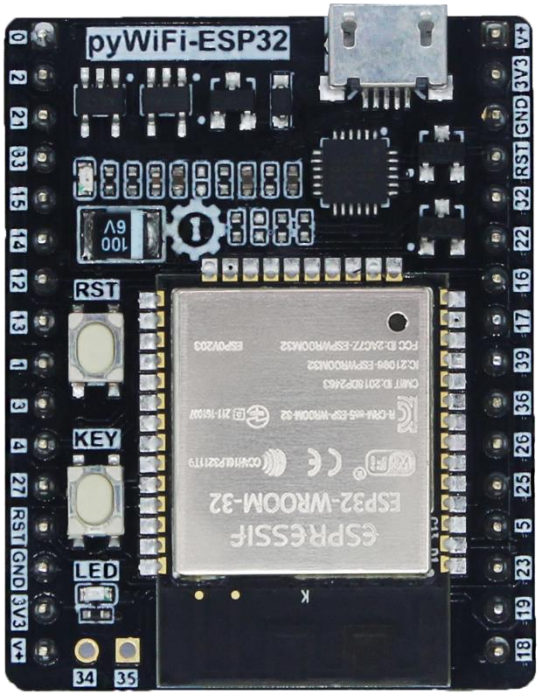


图 12-16 pyWiFi-ESP32

- **实验目的：**

通过定时器让 LED 周期性每秒闪烁 1 次。

- **实验讲解：**

ESP32 内置 RTOS（实时操作系统）定时器，在 machine 的 Timer 模块中。通过 MicroPython 可以轻松编程使用。我们也是只需要了解其构造对象函数和使用方法即可！

构造函数
tim=machine.Timer(-1)
构建定时器对象。RTOS 定时器编号为-1；
使用方法
tim.init(period,mode,callback)
定时器初始化。 period:单位为 ms； mode：2 种工作模式，Timer.ONE_SHOT（执行一次）、Timer.PERIODIC（周期性）；callback:定时器中断后的回调函数。

表 12-4 定时器对象

定时器到了预设指定时间后，也会产生中断，因此跟外部中断的编程方式类似，代码编程流程图如下：

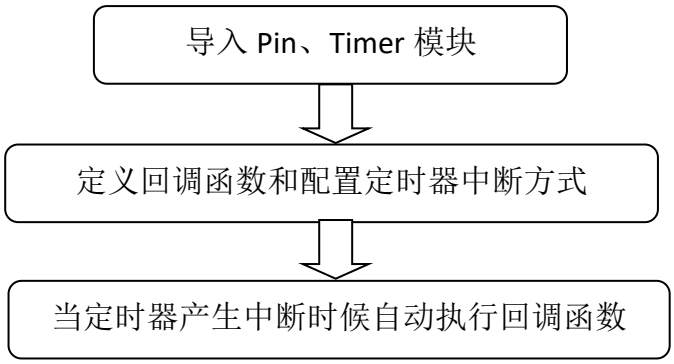


图 12-17 代码编写流程

参考代码如下：

```
'''
实验名称：定时器
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过定时器让 LED 周期性每秒闪烁 1 次
'''
```

```
from machine import Pin,Timer


led=Pin(2,Pin.OUT)
Counter = 0
Fun_Num = 0


def fun(tim):


    global Counter
    Counter = Counter + 1
    print(Counter)
    led.value(Counter%2)


#开启 RTOS 定时器，编号为-1
tim = Timer(-1)
tim.init(period=1000, mode=Timer.PERIODIC,callback=fun) #周期为 1000ms
```

- **实验结果：**



图 12-18　LED 以周期 1 秒的间隔闪烁

- **总结：**

本节实验介绍了 RTOS 定时器的使用方式，有用户可能会认为使用延时延时也可以实现这个功能，但相比于延时函数，定时器的好处就是不占用 CPU 资源。

12.5 I2C 总线（OLED 显示屏）

- **前言：**

前面学习了按键输入设备后，这一节我们来学习输出设备 OLED 显示屏，其实之前的 LED 灯也算是输出设备，因为它们确切地告诉了我们硬件的状态。只是相对于只有亮灭的 LED 而言，显示屏可以显示更多的信息，体验更好。

本章节的 OLED 显示屏学习，实际上是在使用 I2C 的总线接口，pyWiFi-ESP32 是通过 I2C 总线与 OLED 显示屏通讯的。这章稍微复杂一点，但我们把它放在了前面来学习，是因为学会了显示屏的使用，那么在后面的实验中可玩性就更强了。

- **实验平台：**

pyWiFi-ESP32 和 pyBase 开发底板。

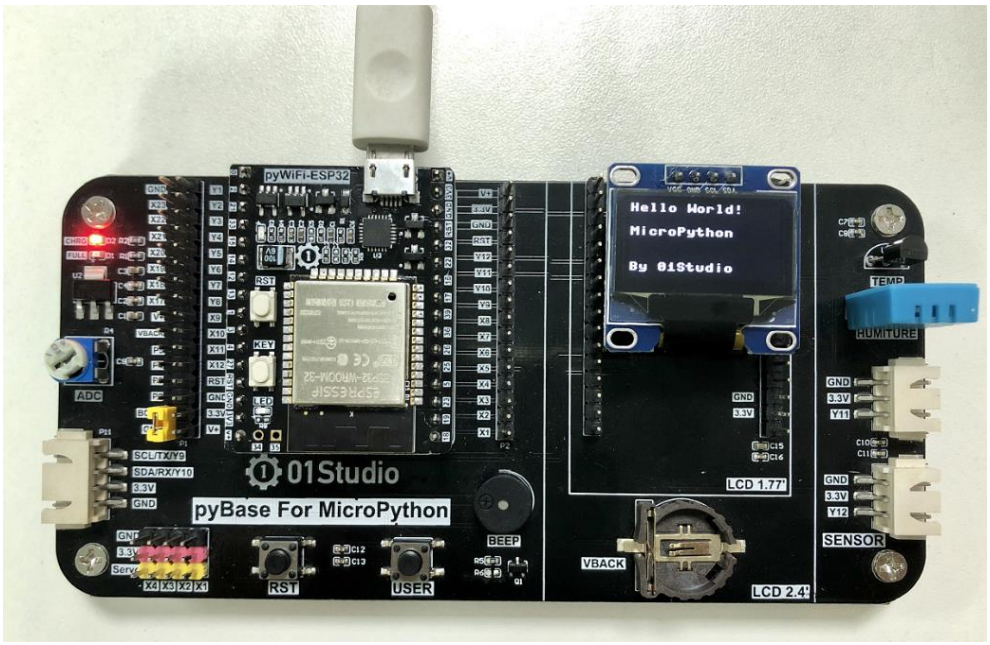


图 12-19 pyWiFi-ESP32 开发套件

- **实验目的：**

学习使用 MicroPython 的 I2C 总线通讯编程和 OLED 显示屏的使用。

- **实验讲解：**

什么是 I2C?

I2C 是用于设备之间通信的双线协议，在物理层面，它由 2 条线组成：SCL 和 SDA，分别是时钟线和数据线。也就是说不通设备间通过这两根线就可以进行通

信。

什么是 OLED 显示屏？

OLED 的特性是自己发光，不像 TFT LCD 需要背光，因此可视度和亮度均高，其次是电压需求低且省电效率高，加上反应快、重量轻、厚度薄，构造简单，成本低等特点。简单来说跟传统液晶的区别就是里面像素的材料是由一个个发光二极管组成，因为密度不高导致像素分辨率低，所以早期一般用作户外 LED 广告牌。随着技术的成熟，使得集成度越来越高。小屏也可以制作出较高的分辨率。



图 12-20　I2C 接口的 OLED

在了解完 I2C 和 OLED 显示屏后，我们先来看看 pyBase 开发板的原理图，也就是上面的 OLED 接口是如何连线的。

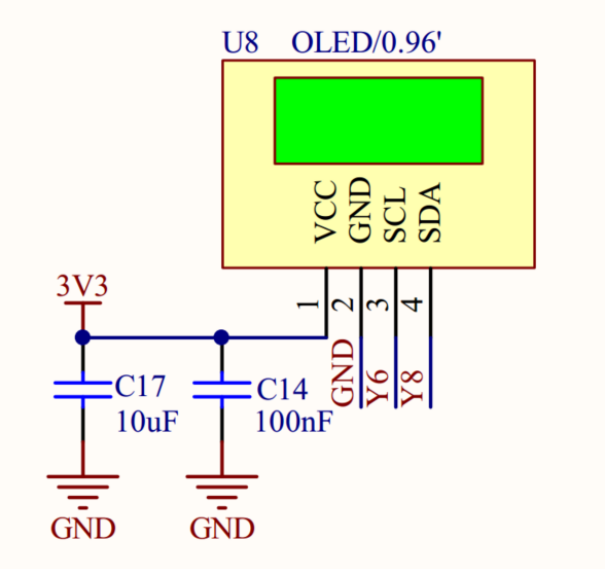


图 12-21　OLED 接口原理图

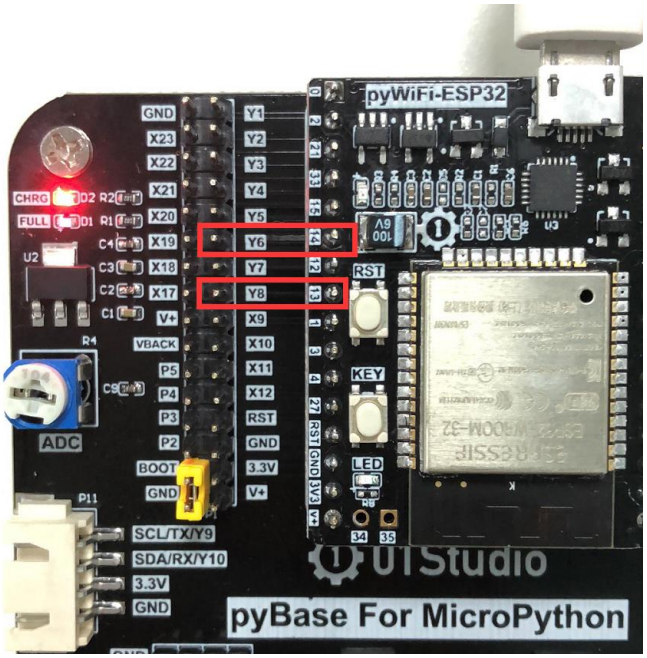


图 12-22　pyWiFi-ESP32 和 pyBase 的 OLED 连线图

从上图可见连接到 OLED 的别是 Y6→SCL 和 Y8→SDA。跟 pyWiFi-ESP32 的 14、13 引脚相连。本实验将使用 MicroPython 的 Machine 模块来定义 Pin 口和 I2C 初始化。具体如下：

构造函数
i2c = machine.I2C(scl,sda)
构建 I2C 对象。scl:时钟引脚；sda:数据引脚。
使用方法
i2c.scan()
扫描 I2C 总线的设备。返回地址，如：0x3c；
i2c.readfrom(addr,nbytes)
从指定地址读数据。addr:指定设备地址；nbytes:读取字节数；
i2c.write(buf)
写数据。buf:数据内容；
*其它更多用法请阅读 MicroPython 文档： 中文文档链接：http://docs.micropython.01studio.org/

表 12-5　I2C 对象

定义好 I2C 后，还需要驱动一下 OLED。这里我们已经写好了 OLED 的库函数，在 ssd1306.py 文件里面。开发者只需要拷贝到 pyBoard 文件系统里面，然后在 main.py 里面调用函数即可。人生苦短，我们学会调用函数即可，也就是注重顶层的应用，想深入的小伙伴也可以自行研究 ssd1306.py 文件代码。OLED 显示屏对象介绍如下：

构造函数
oled = SSD1306_I2C(width, height, i2c, addr)
构 OLED 显示屏对象。width:屏幕宽像素；height: 屏幕高像素；i2c:定义好的 I2C 对象; addr:显示屏设备地址。
使用方法
`oled.text(string,x,y)`
将 string 字符写在指定为位置。string：字符；x:横坐标；y:纵坐标。
`oled.show()`
执行显示。
`oled.fill(RGB)`
清屏。RGB：`0` 表示黑色，`1` 表示白色。

表 12-6　OLED 对象

学习了 I2C、OLED 对象用法后我们通过编程流程图来理顺一下思路：

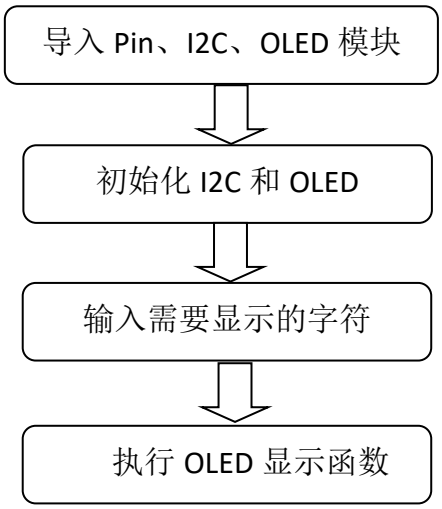


图 12-23 代码编写流程

main.py 文件参考代码如下：

```
'''
实验名称：I2C 总线(OLED 显示屏)
版本：v1.0
日期：2019.7
作者：01Studio
'''

from machine import I2C,Pin       #从 machine 模块导入 I2C、Pin 子模块
from ssd1306 import SSD1306_I2C   #从 ssd1306 模块中导入 SSD1306_I2C 子模块

i2c = I2C(sda=Pin(13), scl=Pin(14))  #I2C 初始化：sda-->13, scl -->14

#OLED 显示屏初始化：128*64 分辨率,OLED 的 I2C 地址是 0x3c
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

oled.text("Hello World!", 0,  0)      #写入第 1 行内容
oled.text("MicroPython",  0, 20)      #写入第 2 行内容
oled.text("By 01Studio",  0, 50)      #写入第 3 行内容

oled.show()   #OLED 执行显示
```

上述代码中 OLED 的 I2C 地址是 0x3C,不同厂家的产品地址可能预设不一样，具体参考厂家的说明书。或者也可以通过 I2C.scan()来获取设备地址。

另外记得将我们提供的示例代码中的 ssd1306.py 驱动文件拷贝到 pyWiFi-ESP32 的文件系统下，跟 main.py 保持同一个路径。

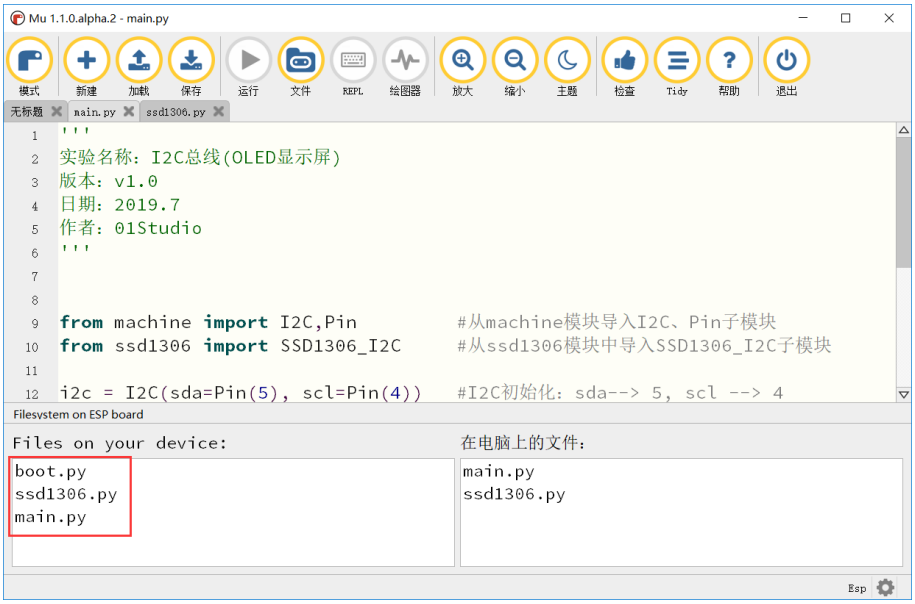


图 12-24 将 main.py、ssd1306.py 拷贝到设备

- **实验结果：**

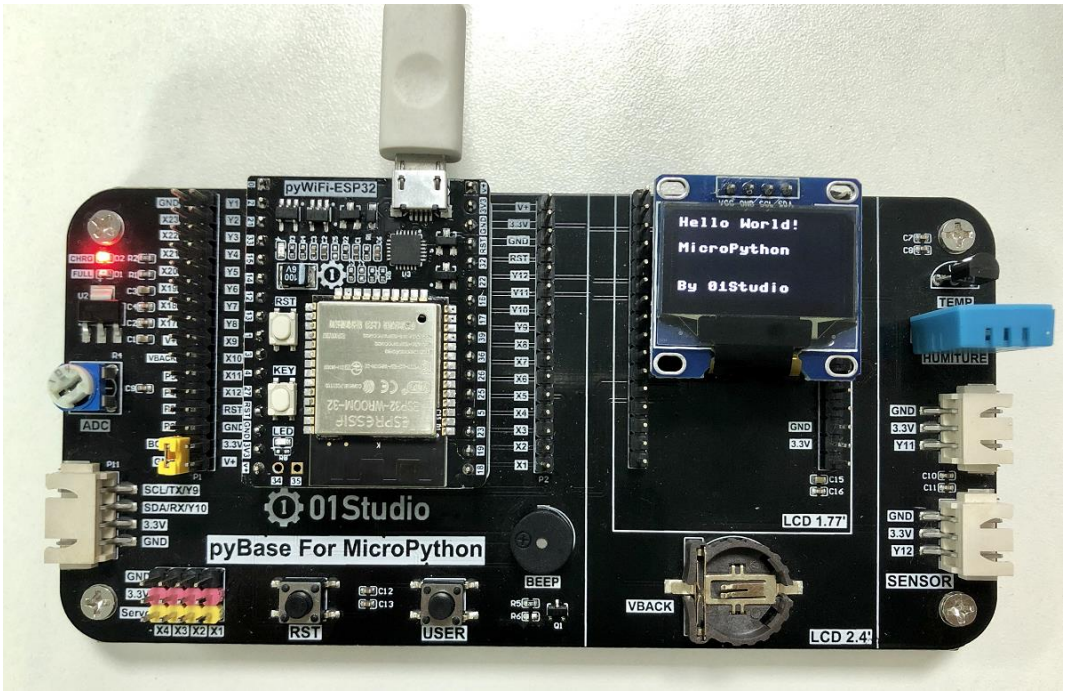


图 12-25 OLED 实验结果

- **总结：**

这一节我们学会了驱动 OLED 显示屏，换着以往如果从使用单片机从 0 开发的话你需要了解 I2C 总线原理，了解 OLED 显示屏的使用手册，编程 I2C 代码，

有经验的嵌入式工程师搞不好也要弄个几天。现在基本半个小时解决问题。当然前提是别人已经给你搭好桥了，有了强大的底层驱动代码支持，我们只做好应用就好。

这一节学习的意义不仅在完成实验。在学习完 OLED 显示屏实验后，接下来我们的实验都可以使用这个 OLED 来跟用户交互了，这大大提高了实验的可观性。

12.6 RTC 实时时钟

- **前言：**

时钟可以说我们日常最常用的东西了，手表、电脑、手机等等无时无刻不显示当前的时间。可以说每一个电子爱好者心中都希望拥有属于自己制作的一个电子时钟，接下来我们就用 MicroPython 开发板来制作一个属于自己的电子时钟。

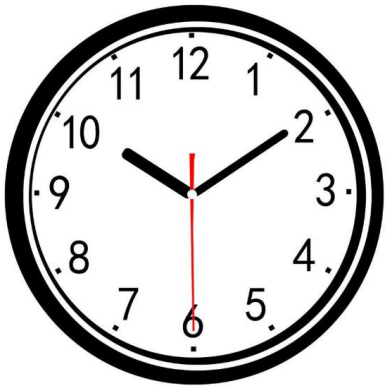


图 12-26 时钟

- **实验平台：**

pyWiFi-ESP32 和 pyBase 开发底板。

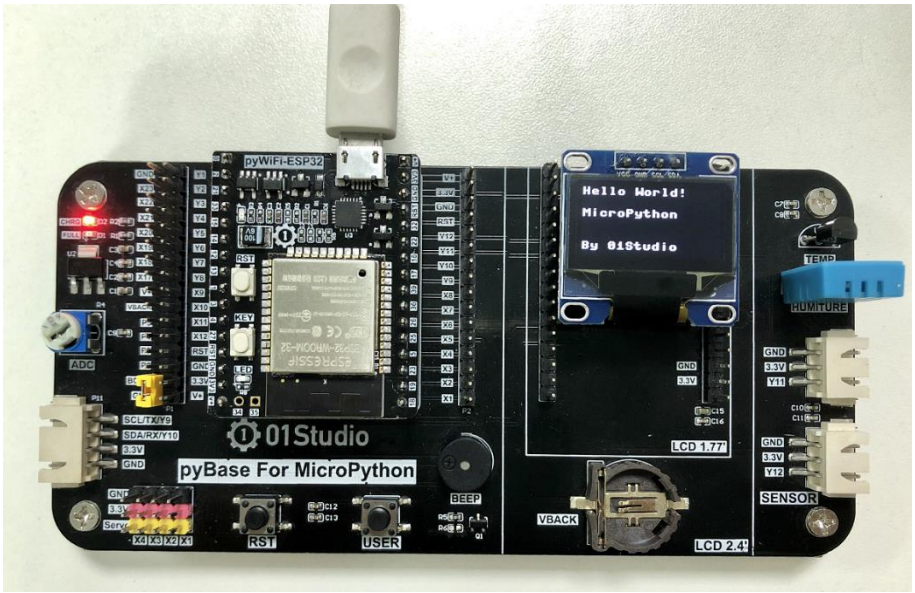


图 12-27 pyWiFi-ESP32 开发套件

- **实验目的：**

  学习 RTC 编程和制作电子时钟，使用 OLED 显示。

- **实验讲解：**

实验的原理是读取 RTC 数据，然后通过 OLED 显示。毫无疑问，强大的 MicroPython 已经集成了内置时钟函数模块。位于 machine 的 RTC 模块中，具体介绍如下：

构造函数
rtc=machine.RTC()
构建 RTC 对象。
使用方法
`rtc.datetime((2019, 4, 1, 0, 0, 0, 0, 0))`
设置日期和时间。按顺序分别是：（年，月，日，星期，时，分，秒，微秒），其中星期使用 0-6 表示周一至周日。
`rtc.datetime()`
获取当前日期和时间。

表 12-7

从上表可以看到 RTC（）的使用方法，我们需要做的就是先设定时间，然后再获取当前芯片里的时间，通过 OLED 显示屏显示，如此循环。在循环里，如果一直获取日期时间数据会造成资源浪费，所以可以每隔第一段时间获取一次数据，又由于肉眼需要看到至少每秒刷新一次即可，这里每隔 300ms 获取一次数据，使用前面学习过的 RTOS 定时器来计时，具体编程流程如下：

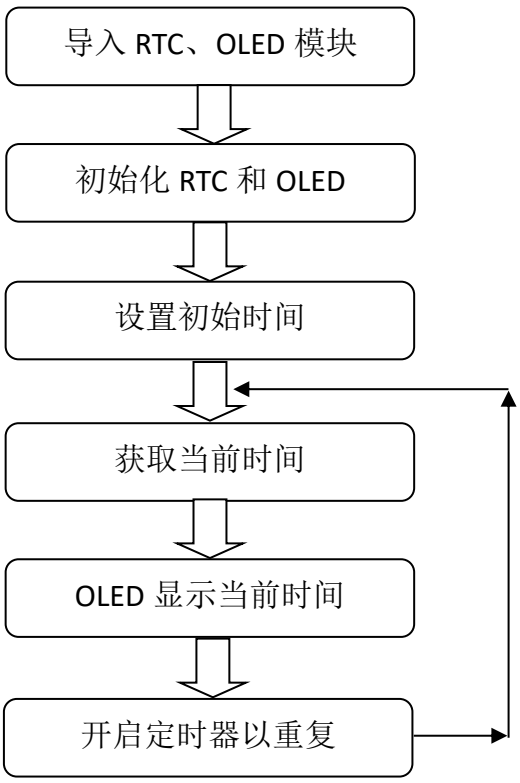


图 12-28

实验参考代码如下：

```
'''
实验名称：RTC 实时时钟
版本：v1.0
日期：2019.7
作者：01Studio
'''

# 导入相关模块
from machine import Pin, I2C, RTC,Timer
from ssd1306 import SSD1306_I2C

# 定义星期和时间（时分秒）显示字符列表
week = ['Mon', 'Tues', 'Wed', 'Thur', 'Fri', 'Sat', 'Sun']
```

```
time_list = ['', '', '']

# 初始化所有相关对象
i2c = I2C(sda=Pin(13), scl=Pin(14)) #I2C 初始化：sda-->13, scl -->14
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
rtc = RTC()

# 首次上电配置时间，按顺序分别是：年，月，日，星期，时，分，秒，次秒级；这里做
#了一个简单的判断，检查到当前年份不对就修改当前时间，开发者可以根据自己实际情况
#来修改。
if rtc.datetime()[0] != 2019:
    rtc.datetime((2019, 4, 1, 0, 0, 0, 0, 0))

def RTC_Run(tim):
    datetime = rtc.datetime()  # 获取当前时间

    oled.fill(0)  # 清屏显示黑色背景
    oled.text('01Studio', 0, 0)    # 首行显示 01Studio
    oled.text('RTC Clock', 0, 15)  # 次行显示实验名称

    # 显示日期，字符串可以直接用“+”来连接
    oled.text(str(datetime[0]) + '-' + str(datetime[1]) + '-' +
str(datetime[2]) + ' ' + week[datetime[3]], 0, 40)

# 显示时间需要判断时、分、秒的值否小于 10，如果小于 10，则在显示前面补“0”以
# 达到较佳的显示效果
    for i in range(4, 7):
        if datetime[i] < 10:
            time_list[i - 4] = "0"
        else:
```

```
            time_list[i - 4] = ""


    # 显示时间
    oled.text(time_list[0] + str(datetime[4]) + ':' + time_list[1] +
str(datetime[5]) + ':' + time_list[2] + str(datetime[6]), 0, 55)


    oled.show()


#开启 RTOS 定时器
tim = Timer(-1)
tim.init(period=300, mode=Timer.PERIODIC, callback=RTC_Run) #周期 300ms
```

由于实验要用到 OLED 显示屏，所以同样别忘了将示例代码该实验文件夹下的 ssd1306.py 文件复制到 pyWiFi-ESP32 的文件系统里面。

- **实验结果：**

烧录程序复位后，可以看到时钟开始跑起来。

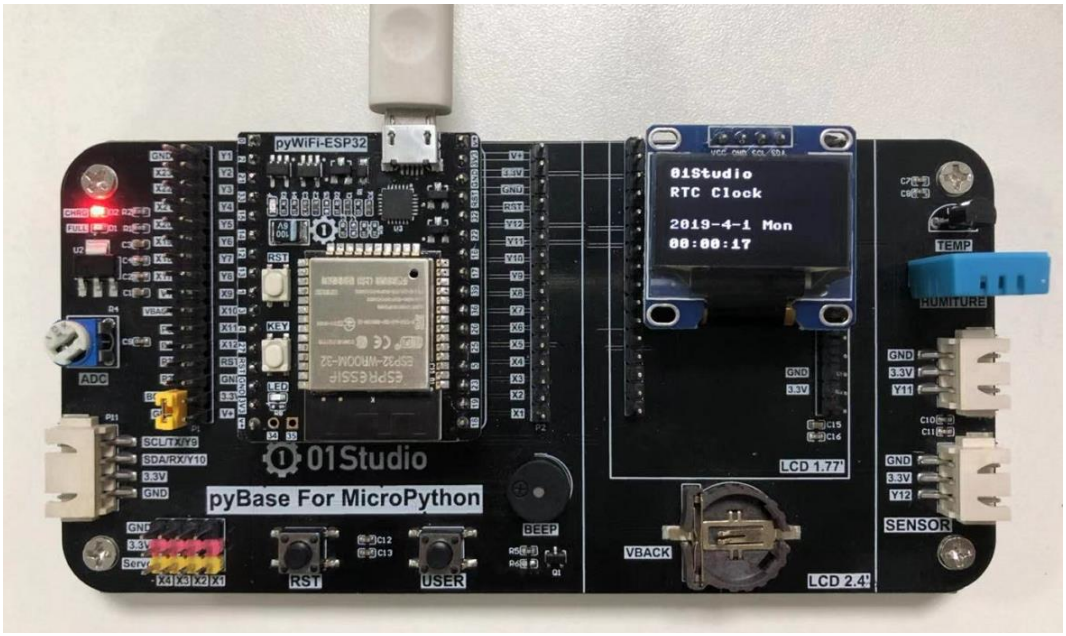


图 12-29 RTC 实时时钟实验

由于 ESP32 没有后备电池引脚，所以不支持掉电保存。因此 pybase 上面的纽扣电池是不起作用的。

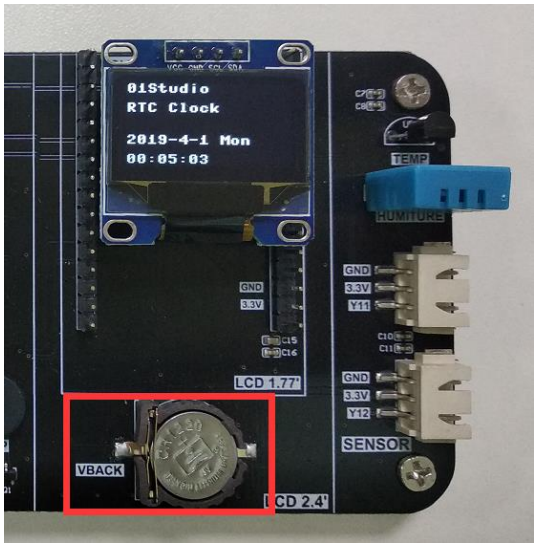


图 12-30 后备电池（VBACK）

- **总结：**

RTC 实时时钟的可玩性很强，我们还可以根据自己的风格来设定数字显示位置，以及加上一些属于自己的字符标识。打造自己的电子时钟。

12.7 ADC

- **前言：**

ADC(analog to digital conversion) 模拟数字转换。意思就是将模拟信号转化成数字信号，由于单片机只能识别二级制数字，所以外界模拟信号常常会通过 ADC 转换成其可以识别的数字信息。常见的应用就是将变化的电压转成数字信号。

- **实验平台：**

pyWiFi-ESP32 和 pyBase 开发底板。

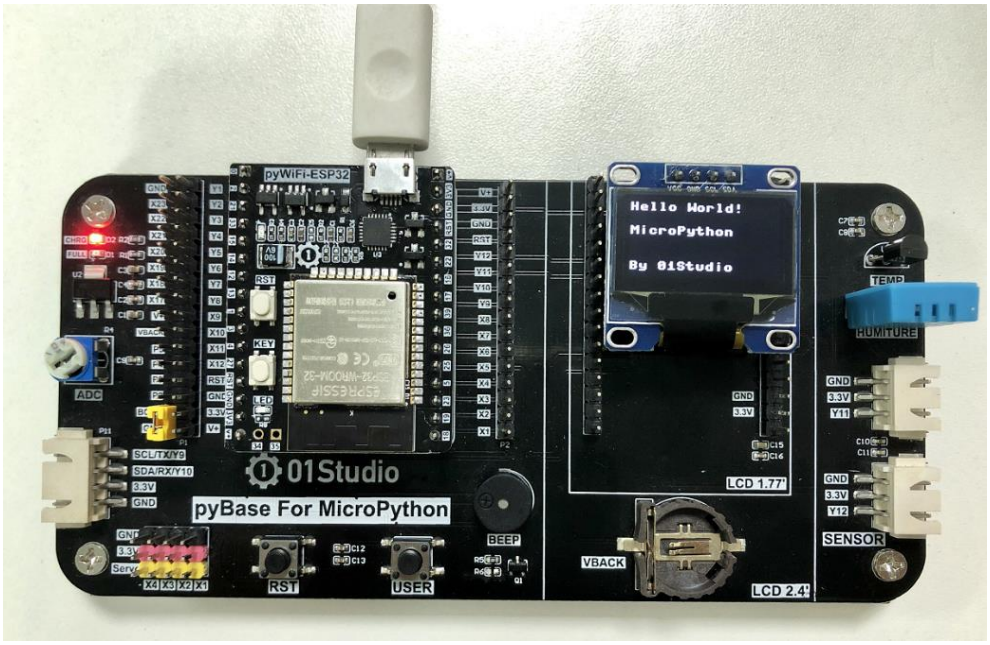


图 12-31 pyWiFi-ESP32 开发套件

- **实验目的：**

通过编程调用 MicroPython 的内置 ADC 函数，实现测量输入电压，并显示到屏幕上。

- **实验讲解：**

pyBase 开发底板的 X7 引脚连接到了电位器，通过电位器的调节可以使得 X7 引脚上的电压变化范围实现从 0-3.3V。

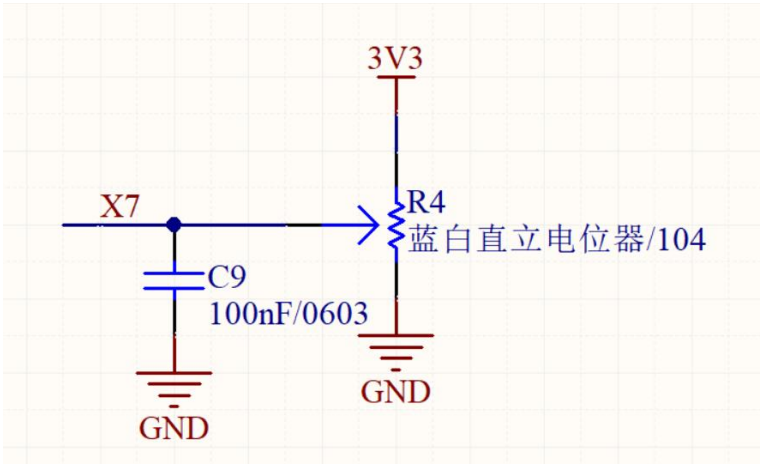


图 12-32 电位器原理图

电位器引脚对应 pyBase 的 X7,实际是跟 pyWiFi-ESP32 的 36 引脚 ADC 输入引脚相连。ESP32 的 ADC 默认只能测量 0-1V 的量程，但 ESP32 内部集成了衰减器，最大支持 11dB 衰减，通过配置衰减器最多能测量 3V 左右的电压。我们来看看 ADC 模块的构造函数和使用方法。

构造函数
adc=machine.ADC(Pin(id))
构建 ADC 对象。id: ESP32 的 ADC 在 32-39 引脚上。
使用方法
adc.read()
获取 ADC 值。测量精度是 12 位，返回 0-4095（表示 0-1V）。
adc.atten(attenuation)
配置衰减器。配置衰减器能增加电压测量范围，但是以精度为代价的。 attenuation:衰减设置 ADC.ATTN_0DB： 0dB 衰减，最大输入电压为 1.00v - 这是默认配置； ADC.ATTN_2_5DB： 2.5dB 衰减，最大输入电压约为 1.34v； ADC.ATTN_6DB：6dB 衰减，最大输入电压约为 2.00v； ADC.ATTN_11DB：11dB 衰减，最大输入电压约为 3.3v。

表 12-8 ADC 对象

你没看错，就这么简单，两句函数就可以获得 ADC 数值。我们将在本实验中以默认的量程 0-1V 来测试。让我们来理顺一下编程逻辑。先导入相关模块，然后初始化模块。在循环中不断读取 ADC 的值，转化成电压值后在 OLED 上面显示，每隔 300 毫秒读取一次，具体如下：

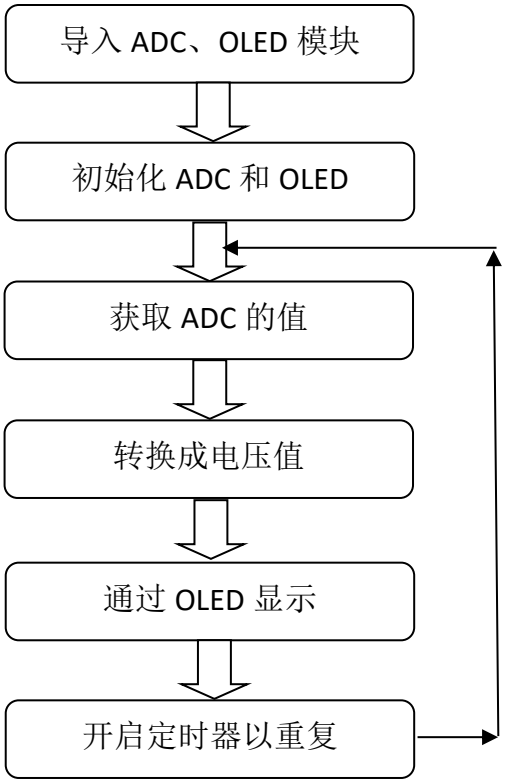


图 12-33 代码编写流程

实验参考代码：

```
'''
实验名称：ADC-电压测量
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过对 ADC 数据采集，转化成电压在显示屏上显示。ADC 精度 12 位，电压 0-1V。
'''
```

```
#导入相关模块
from machine import Pin,I2C,ADC,Timer
from ssd1306 import SSD1306_I2C

#初始化相关模块
i2c = I2C(sda=Pin(13), scl=Pin(14))  #I2C 初始化：sda-->13, scl -->14
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
adc = ADC(Pin(36)) #36 引脚跟 pyBase 的电位器相连接

def ADC_Test(tim):

    oled.fill(0)  # 清屏显示黑色背景
    oled.text('01Studio', 0, 0)  # 首行显示 01Studio
    oled.text('ADC', 0, 15)      # 次行显示实验名称

    #获取 ADC 数值
    oled.text(str(adc.read()),0,40)
    oled.text('(4095)',40,40)

    #计算电压值，获得的数据 0-4095 相当于 0-1V，（'%.2f'%）表示保留 2 位小数
    oled.text(str('%.2f'%(adc.read()/4095)),0,55)
    oled.text('V',40,55)

    oled.show()

#开启 RTOS 定时器
tim = Timer(-1)
tim.init(period=300, mode=Timer.PERIODIC, callback=ADC_Test) #周期 300ms
```

- **实验结果：**

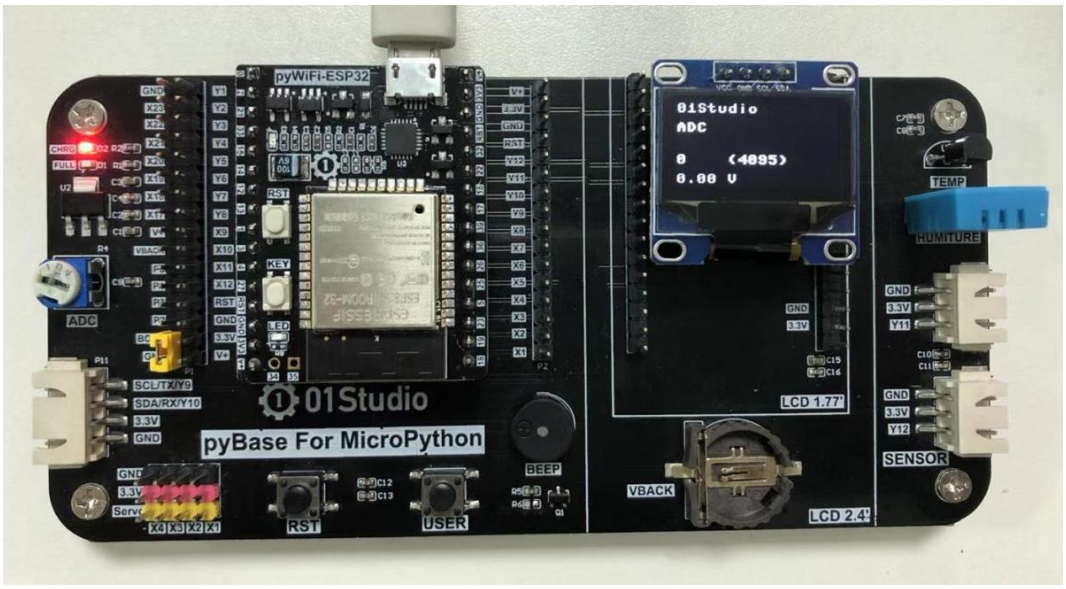


图 12-34 电位器顺时钟拧到尽头是 0V

通过调节电位器，可以发现电压在不断变化。由于本实验的电压测量量程限制，当超出 1V 量程后，始终显示电压在 1V，测量值是 4095。

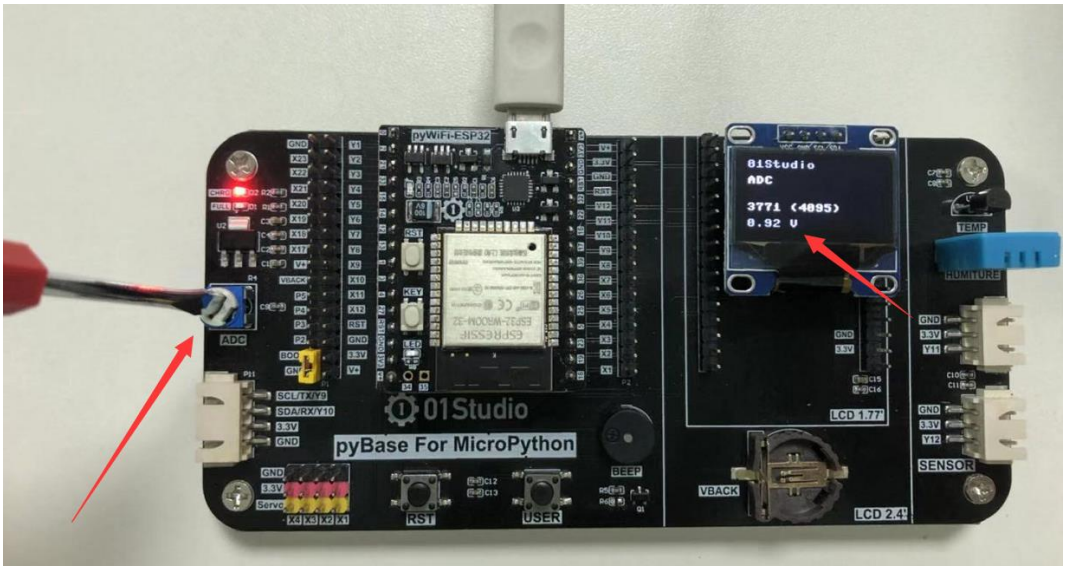


图 12-35 调节电位器来改变电压输入

- **总结：**

这一节我们学习了 ADC 的应用，主要用于电压的检测。有兴趣的用户可以尝试使用其衰减器测试，可以扩充电压量程，但精度会有所下降。

12.8 PWM

- **前言：**

上一节的 ADC 是信号输入，这节的 PWM 就是一个信号输出。PWM（脉冲宽度调制），主要用于输出不同频率、占空比（一个周期内高电平出现时间占总时间比例）的方波。以实现固定频率或平均电压输出。

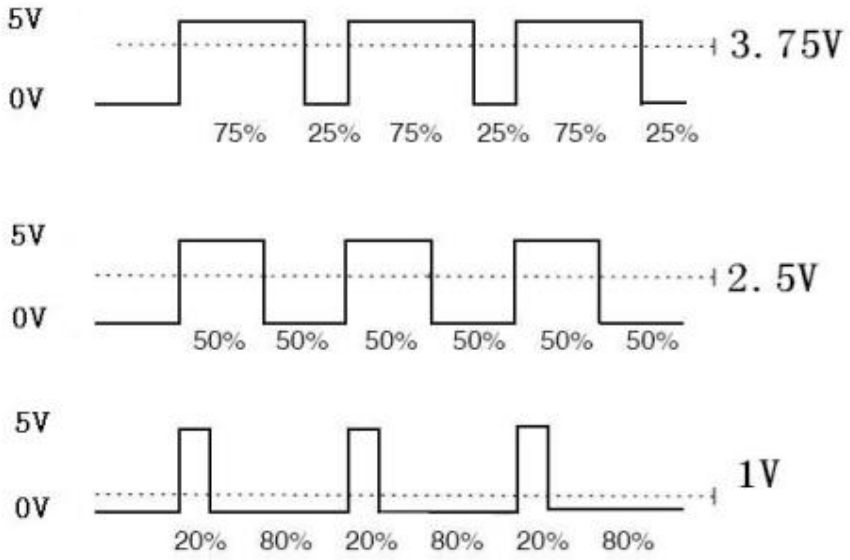


图 12-36 PWM 波形示例

- **实验平台：**

pyWiFi-ESP32 和 pyBase 开发底板。

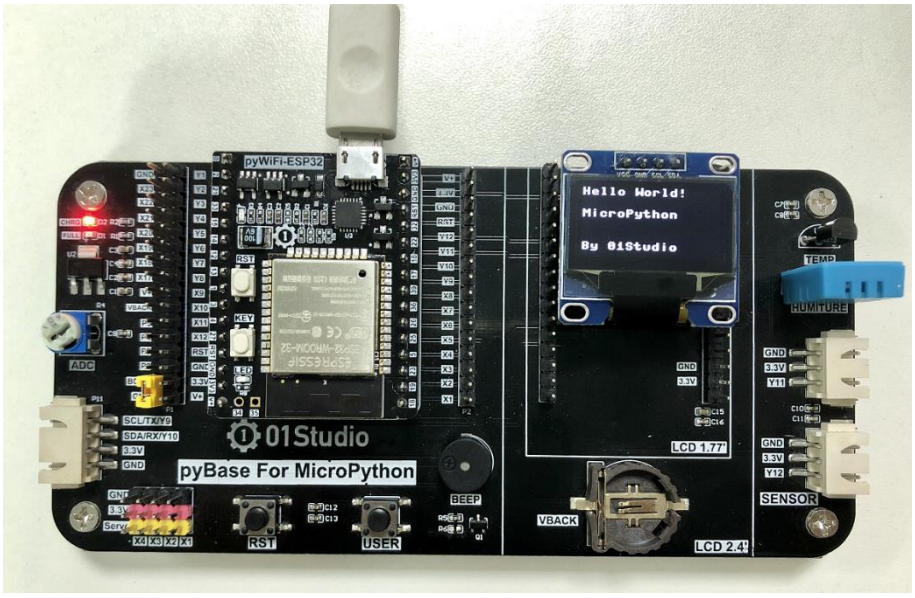


图 12-37 pyWiFi-ESP32 开发套件

- **实验目的：**

通过不同频率的 PWM 信号输出，驱动无源蜂鸣器发出不同频率的声音。

- **实验讲解：**

蜂鸣器分有源蜂鸣器和无源蜂鸣器，有源蜂鸣器的使用方式非常简单，只需要接上电源，蜂鸣器就发声，断开电源就停止发声。而本实验用到的无源蜂鸣器，是需要给定指定的频率，才能发声的，而且可以通过改变频率来改变蜂鸣器的发声音色，以此来判定 pyWiFi-ESP32 的 PWM 输出频率是在变化的。

pyBase 开发底板上的无源蜂鸣器连接到引脚 X5。如下图所示：

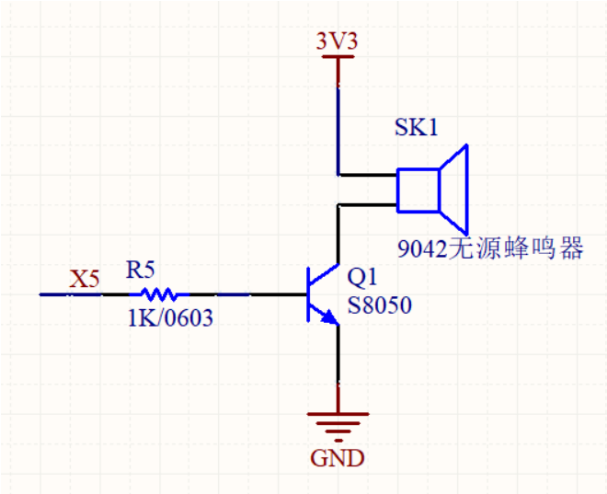


图 12-38 蜂鸣器原理图

PWM 可以通过 ESP32 所有 GPIO 引脚输出. 所有通道都有 1 个特定的频率，从 1 到 40M 之间（单位是 Hz）。占空比的值为 0 至 1023 之间。在本实验中我们用到引脚 25。

先看看 PWM 模块对象：

构造函数
pwm25=machine.PWM(machine.Pin(id),freq,duty)
构建 PWM 对象。id:引脚编号；freq:频率值；duty:占空比；配置完后 PWM 自动生效。
使用方法
pwm25.freq(freq)
设置频率。freq:频率值在 1-1000 之间，freq 为空时表示获取当前频率值。

pwm25.duty(duty)
设置占空比。duty:占空比在 0-1023 之间，duty 为空时表示获取当前占空比值。
pwm25.deinit()
关闭 PWM。

表 12-9　PWM 对象

无源蜂鸣器我们可以用特定频率的方波来驱动，方波的原理很简单，就是一定频率的高低电平转换，可以简单理解成占空比为 50%的 PWM 输出。

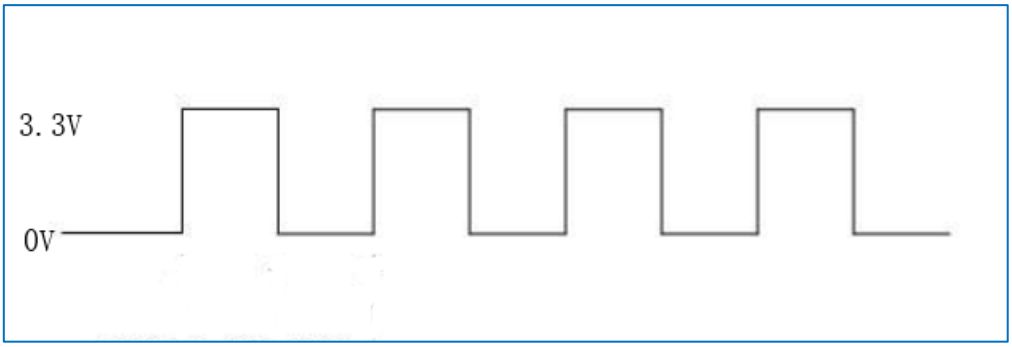


图 12-39　方波信号

结合上述讲解，总结出代码编写流程图如下：

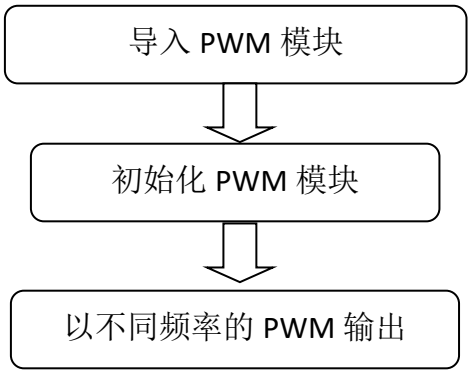


图 12-40 代码编写流程图

实验参考代码：

```
'''
实验名称：PWM
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过不同频率的 PWM 信号输出，驱动无源蜂鸣器发出不同频率的声音。
'''

from machine import Pin, PWM
import time

Beep = PWM(Pin(25), freq=0, duty=512) # 在同一语句下创建和配置 PWM

#蜂鸣器发出频率 200Hz 响声
Beep.freq(200)
time.sleep_ms(1000)

#蜂鸣器发出频率 400Hz 响声
Beep.freq(400)
time.sleep_ms(1000)

#蜂鸣器发出频率 600Hz 响声
Beep.freq(600)
time.sleep_ms(1000)



#蜂鸣器发出频率 800Hz 响声
Beep.freq(800)
time.sleep_ms(1000)
```

```
#蜂鸣器发出频率 1000Hz 响声
Beep.freq(1000)
time.sleep_ms(1000)

#停止
Beep.deinit()
```

- **实验结果：**

下载程序，复位后可以听到蜂鸣器依次发出不同频率的响声。

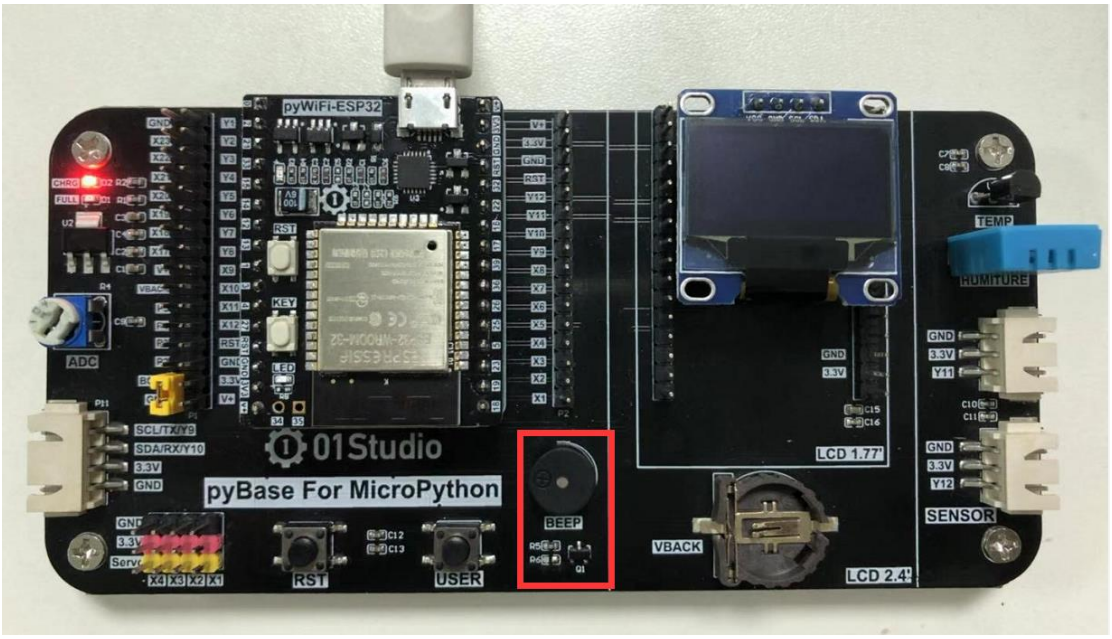


图 12-41 实验现象

有条件的朋友可以使用示波器测量 pyWiFi-ESP32 的 25 引脚或 pyBase 的 X5 接口，观察信号波形的变化：

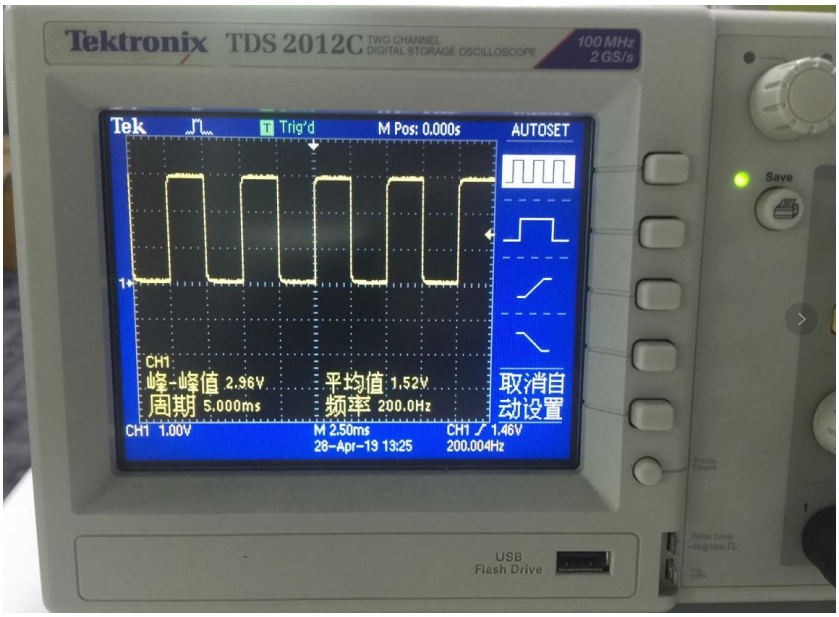


图 12-42　示波器显示波形

- **总结：**

到了这一节，我们发现实验中对象函数使用方法非常简单，这是好事，让我们可以将更多精力放在应用上，做出更多好玩的创意。而不需要过多的关注复杂的底层代码开发。而随着要实现功能的复杂化让编程的代码数量变多，逻辑也将略显复杂。

第13章　传感器实验

基础实验让我们对 MicroPython 的操作和编程有了比较全面的了解，但其魅力绝不限于此。日常生活中我们会用到各式各样的外设或者传感器，还是那句，一个有经验的嵌入式开发工程师驱动一款未接触过的传感器的一般流程是：了解传感器原理、设计电路图、信号时序分析和编程。没个几天折腾不出来。

生活中有很多传感器已经是非常通用了，前人已经做好封装函数模块，我们直接调用函数即可。我们不需要将时间花在“怎么用”上，而更多的是考虑“用到什么地方”!

本章实验以最常见的传感器出发，讲述是如何通过 MicroPyhon 编程实现传感器应用的。实验还是基于我们的 MicroPython 开发板。而且实验例程将持续更新。

13.1 温度传感器 DS18B20

- **前言：**

相信没有电子爱好者不知道 DS18B20 的，DS18B20 是常用的数字温度传感器，其输出的是数字信号，具有体积小，硬件开销低，抗干扰能力强，精度高的特点。DS18B20 数字温度传感器接线方便，封装成后可应用于多种场合，如管道式，螺纹式，磁铁吸附式，不锈钢封装式，型号多种多样。

主要根据应用场合的不同而改变其外观。封装后的 DS18B20 可用于电缆沟测温，高炉水循环测温，锅炉测温，机房测温，农业大棚测温，洁净室测温，弹药库测温等各种非极限温度场合。耐磨耐碰，体积小，使用方便，封装形式多样，适用于各种狭小空间设备数字测温和控制领域。

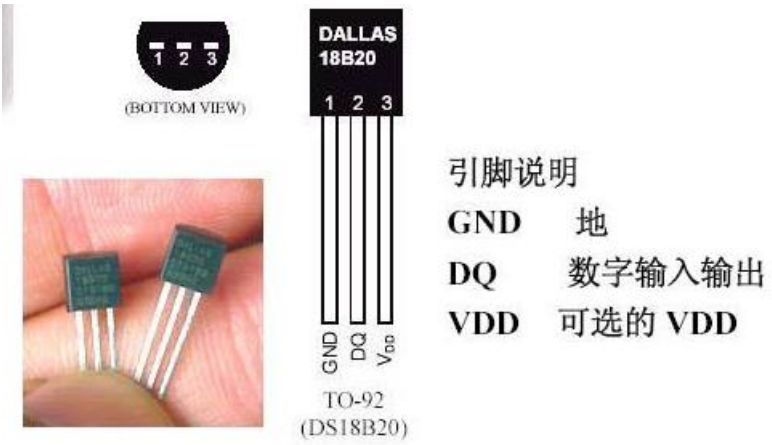


图 13-1 DS18B20 传感器



图 13-2 DS18B20 金属探头封装

- **实验平台：**

pyWiFi-ESP32 和 pyBase 开发底板。DS18B20 温度传感器位于右上方。

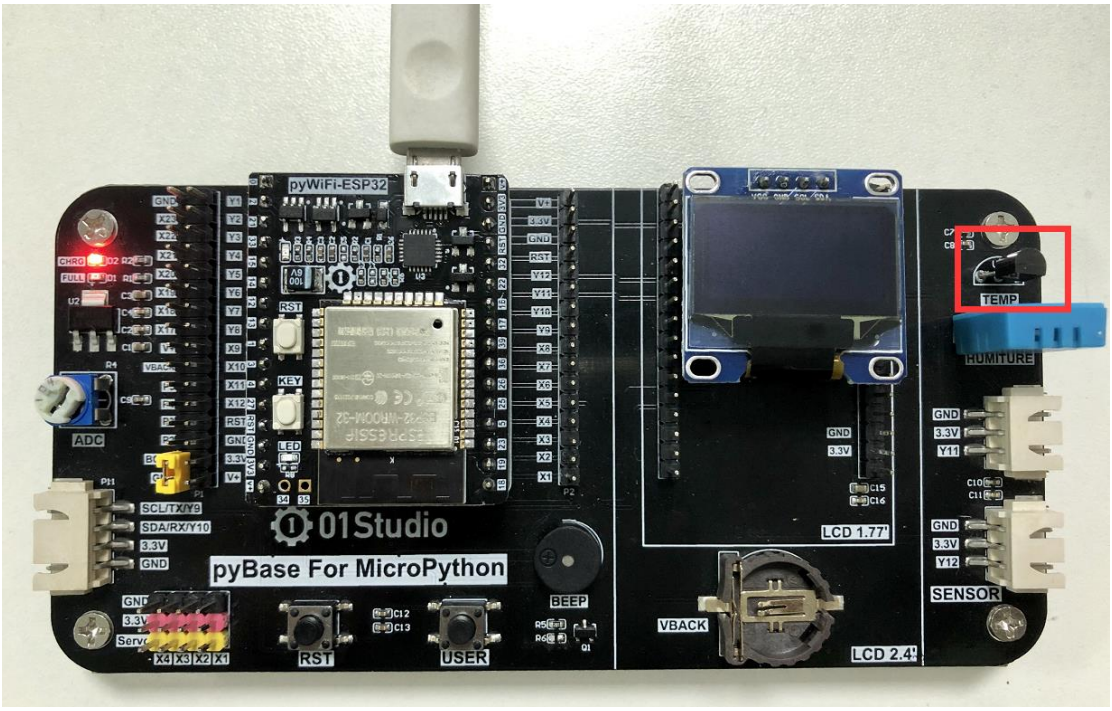


图 13-3 pyWiFi-ESP32 开发套件

- **实验目的：**

通过编程采集温度数据，并在 OLED 上显示。

- **实验讲解：**

DS18B20 是单总线驱动（onewire）传感器，也就是说只占用 1 个 IO 口。我们来看看原理图：

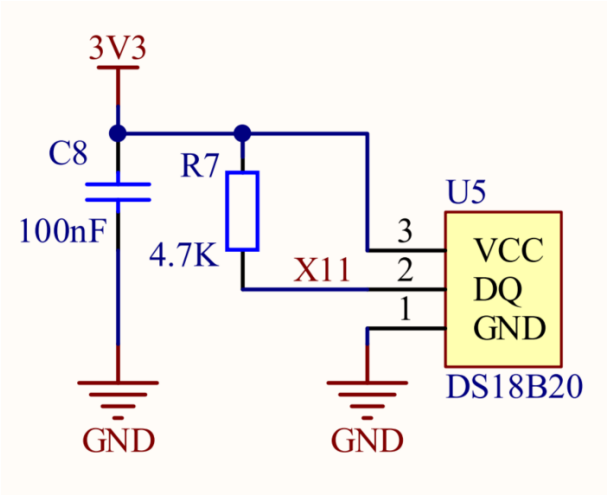


图 13-4 DS18B20 接线原理图

可以看到 DS18B20 传感器连接到了 pyBase 的 X11 引脚上。也就是连接到 pyWiFi-ESP32 的引脚 4，如下图所示：



图 13-5 用跳线帽将 pyBase 的 X19、Y6 短接

也就是说我们需要针对引脚 4 编写程序来驱动 DS18B20。那么我们需要自己来编写驱动么？如果你有兴趣的可以自己尝试一下。这部分我们 01Studio 已经收集整理和编写好了，单总线模块文件是：onewire.py，DS18B20 模块的文件是 ds18x20.py。如果你学习过前面基于 STM32 平台应该不陌生。而对于 ESP32,这两个模块已经集成到了初始化固件中，也就是说我们可以直接在 main.py 导入模块并调用即可！

单总线模块（onewire）和 ds18x20 模块说明如下：

构造函数
ow=onewire.OneWire(machine.Pin(id))
构建单总线对象。id:引脚编号；
使用方法
ow.scan()
扫描总线上的设备。返回设备地址，支持多设备同时挂载。
ow.reset()
总线设备复位。
ow.readbyte()
读 1 个字节。
ow.writebyte(0x12)
写入 1 个字节。
ow.write('123')
写入多个字节。
ow.select_rom(b'12345678')
根据 ROM 编号选择总线上指定设备。

图 13-6 单总线对象

构造函数
ds=ds18x20.DS18X20(ow)
构建 DS18B20 传感器对象。ow:定义好的单总线对象；
使用方法
ds.scan()
扫描总线上的设备。返回设备地址，支持多设备同时挂载。
ds.convert_temp()
温度转换。
ds.read_temp(rom)
获取温度值。rom：表示对应的设备号。

图 13-7 DS18B20 对象

大部分场景下温度的变化不会太频繁，我们可以每隔 1 秒采集一次，显示精度为小数点后 2 位，基本满足大部分应用需求。编程逻辑如下：

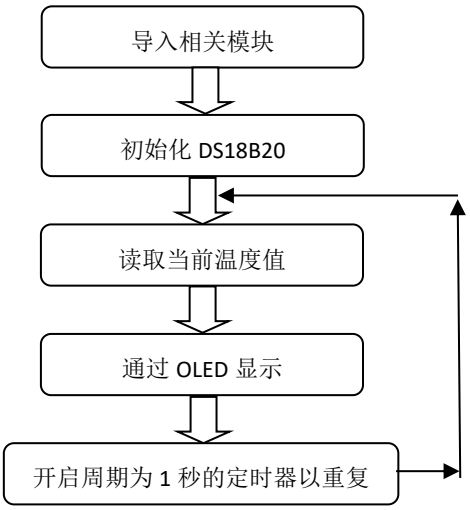


图 13-8　代码编写流程图

实验参考代码：

```
'''
实验名称：温度传感器 DS18B20
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过编程采集温度数据，并在 OLED 上显示。。
'''

#引用相关模块
from machine import Pin,I2C,Timer
from ssd1306 import SSD1306_I2C
import onewire,ds18x20

#初始化相关模块
```

```
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化 DS18B20
ow= onewire.OneWire(Pin(4))  #使能单总线
ds = ds18x20.DS18X20(ow)      #传感器是 DS18B20
rom = ds.scan()         #扫描单总线上的传感器地址，支持多个传感器同时连接

def temp_get(tim):
    ds.convert_temp()
    temp = ds.read_temp(rom[0]) #温度显示,rom[0]为第 1 个 DS18B20

    #OLED 数据显示
    oled.fill(0)   #清屏背景黑色
    oled.text('MicroPython', 0, 0)
    oled.text('Temp test:',0,20)
    oled.text(str('%.2f'%temp)+' C',0,40) #显示 temp,保留 2 位小数
    oled.show()

#开启 RTOS 定时器，编号为-1
tim = Timer(-1)
#定时器周期为 1000ms
tim.init(period=1000, mode=Timer.PERIODIC,callback=temp_get)
```

实验使用到 OLED 显示屏，记得同时将 ssd1306.py 文件拷贝到设备中。

- **实验结果：**

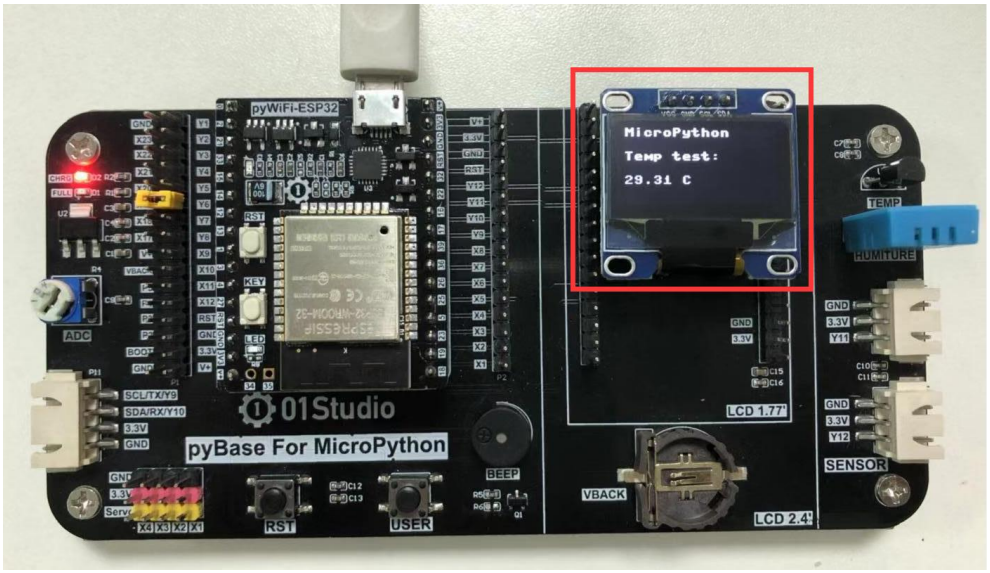


图 13-9 温度检测实验

- **实验拓展：**

pyBase 开发底板预留了外界传感器接口，只要接线正确就可以进行更多的传感器实验。我们将带金属探头的 DS18B20 传感器接到 pyBase 右侧上面的传感器母座，其连接到 pyBase 的“Y11”引脚,也就是对应 pyWiFi-ESP32 的引脚 22。所以只要将原程序代码： ow= onewire.OneWire(Pin(14)) 改成 ow= onewire.OneWire(Pin(22))，即可驱动外接的 DS18B20。

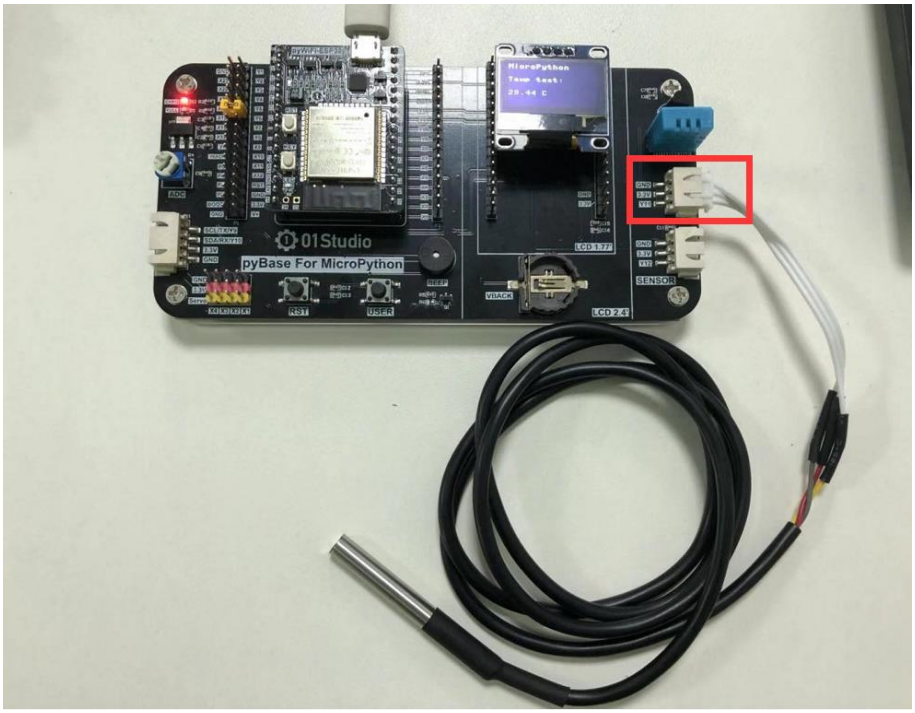


图 13-10 外接温度传感器

我们将金属探头放在水里面，可以见到测试到的温度。

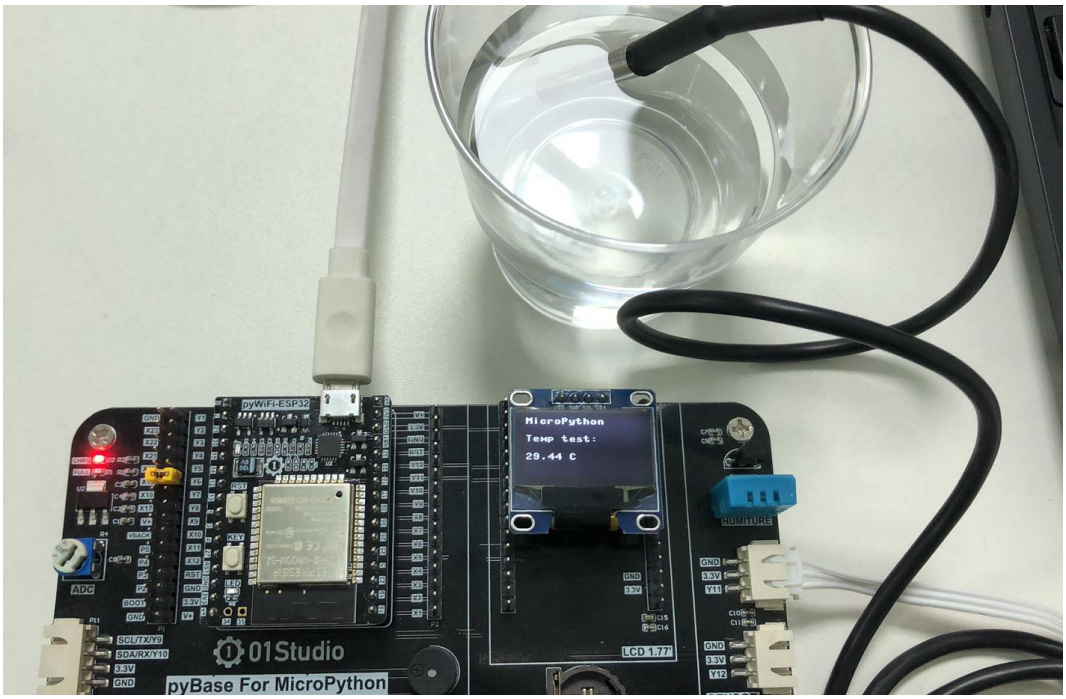


图 13-11 实现金属探头 DS18B20 测量

- **总结**

DS18B20 作为我们第一个实验传感器，使用 MicroPython 编程非常容易就用起来了，而且精度和稳定性丝毫没有影响。温度传感器只是一个敲门砖，接下来我们将会学习更多的传感器应用。

13.2 温湿度传感器 DHT11

- **前言：**

温湿度也是我们日常非常常见的指标，我们使用的是 DHT11 数字温湿度传感器。这是一款含有已校准数字信号输出的温湿度复合传感器，它应用专用的数字模块采集技术和温湿度传感技术，确保产品具有极高的可靠性和卓越的长期稳定性。

DHT11 具有小体积、极低的功耗，信号传输距离可达 20 米以上，使其成为给类应用甚至最为苛刻的应用场合的最佳选择。产品为 4 针单排引脚封装，连接方便。

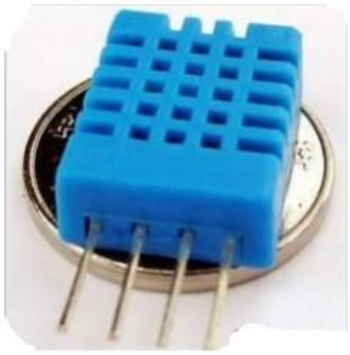


图 13-12 DHT11 温湿度传感器

- **实验平台：**

pyWiFi-ESP32 和 pyBase 开发底板。DHT11 温湿度传感器位于右上方。

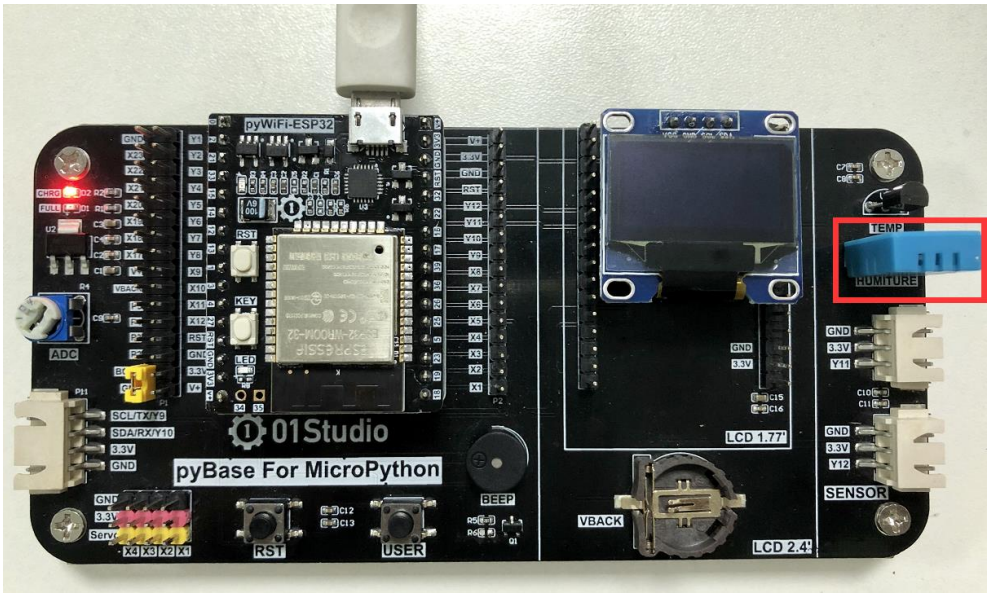


图 13-13 温湿度传感器 DHT11 位于右上方

- **实验目的：**

通过编程采集温湿度数据，并在 OLED 上显示。

- **实验讲解：**

DHT11 虽然有 4 个引脚，但其中第 3 个引脚是悬空的，也就是说 DHT11 也是单总线的传感器，只占用 1 个 IO 口。

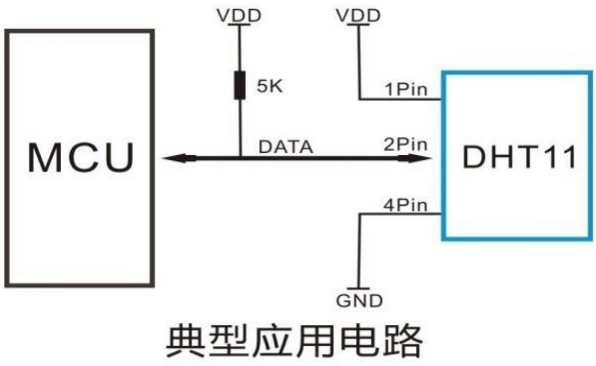


图 13-14　DHT11 应用电路

我们来看看 DHT11 在开发板上的接线图：

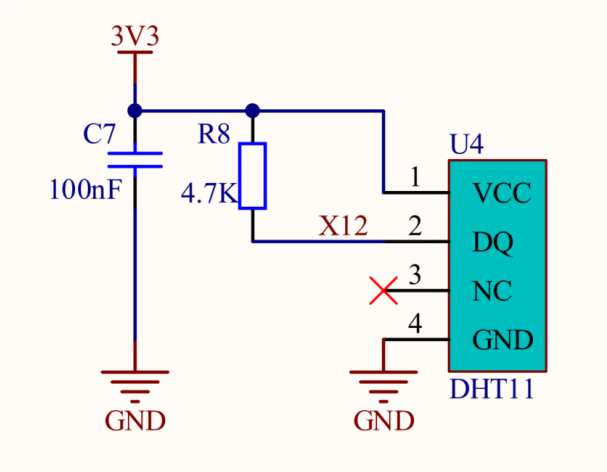


图 13-15　DHT11 接线图

可以看到 DHT11 连接到 pyBase 的‘X12’引脚，也就是连接到 pyWiFi-ESP32 的引脚 27，如下图所示：

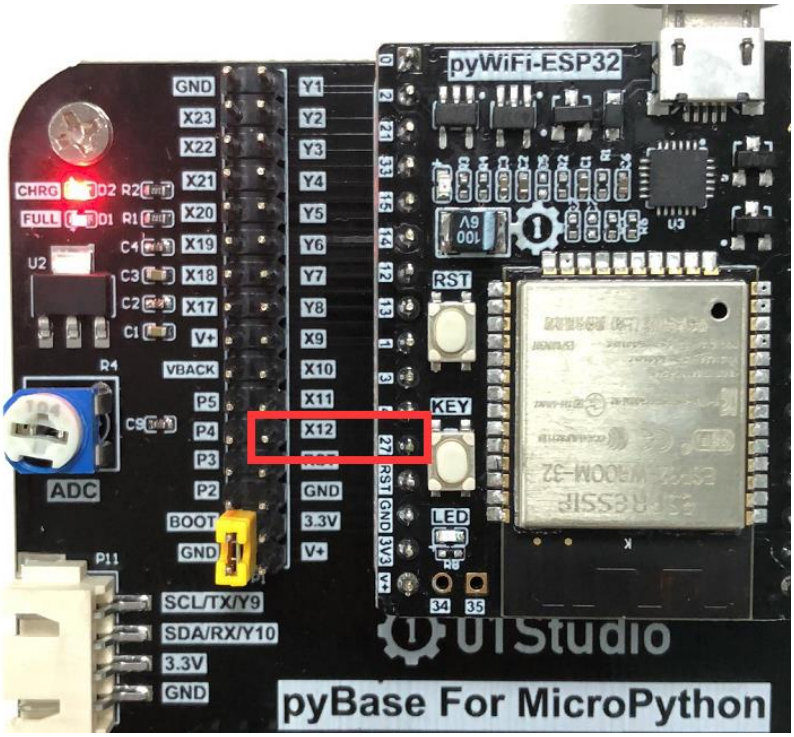


图 13-16 用跳线帽将 pyBase 的 X20、Y5 短接

因此可以针对引脚 27 编程来驱动 DHT11 传感器，模块文件是 dht.py，如果你学习过前面基于 STM32 平台应该不陌生。而对于 ESP32,这个模块已经集成到了初始化固件中，也就是说我们可以直接在 main.py 导入模块并调用即可。函数模块说明如下：

构造函数
d = dht.DHT11(machine.Pin(id))
构建 DHT11 传感器对象。id:传感器所连接的引脚；
使用方法
d.measure()
测量温湿度。
d.temperature()
获取温度值。
d.humidity()
获取湿度值。

表 13-1 DHT11 对象

建议上电先延时 1 秒，让 DHT11 稳定后再开始读取。代码编写流程如下：

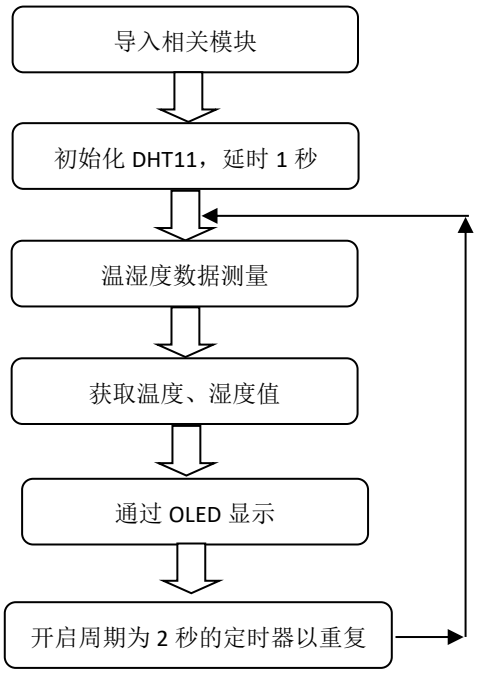


图 13-17 代码编写流程图

实验参考代码：

```
'''
实验名称：温湿度传感器 DHT11
版本：v1.0
日期：2019.7
作者：01Studio
说明：通过编程采集温湿度数据，并在 OLED 上显示。。
'''

#引入相关模块
from machine import Pin,I2C,Timer
from ssd1306 import SSD1306_I2C
import dht,time
```

```
#初始化相关模块
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#创建 DTH11 对象
d = dht.DHT11(Pin(27)) #传感器连接到引脚 15
time.sleep(1)   #首次启动停顿 1 秒让传感器稳定

def dht_get(tim):

    d.measure()         #温湿度采集

   #OLED 显示温湿度
    oled.fill(0) #清屏背景黑色
    oled.text('01Studio', 0, 0)
    oled.text('DHT11 test:',0,15)
    oled.text(str(d.temperature() )+' C',0,40)   #温度显示
    oled.text(str(d.humidity())+' %',48,40)  #湿度显示
    oled.show()

#开启 RTOS 定时器，编号为-1，周期 2 秒
tim = Timer(-1)
tim.init(period=2000, mode=Timer.PERIODIC,callback=dht_get)
```

- **实验结果：**

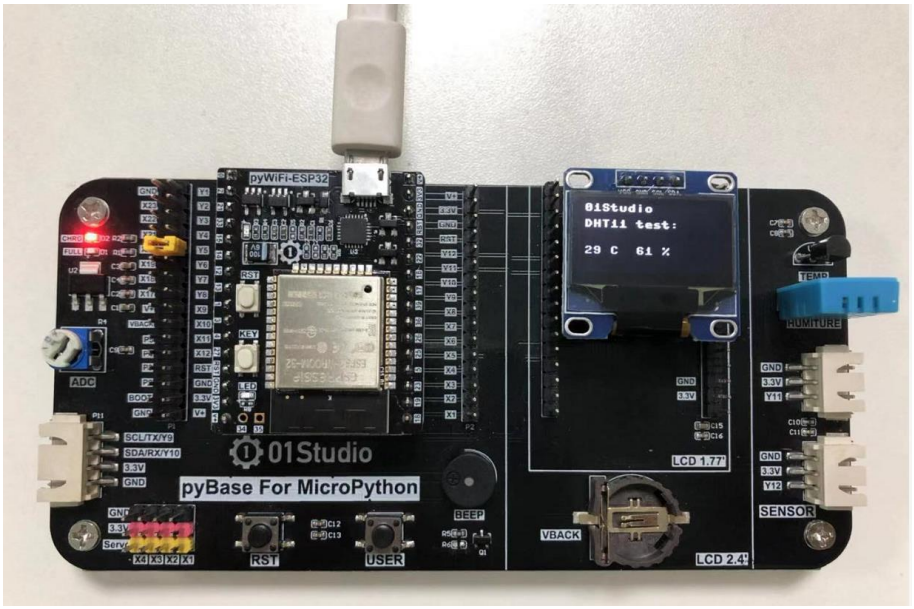


图 13-18　DHT11 实验结果

- **总结：**

通过本节学习我们学会了使用 MicroPython 来驱动 DTH11 温湿度传感器，DHT11 性价比比较高，是很适合学习使用的，但精度和响应速度有点低，需要更高要求应用的用户可以使用 DHT22 或者其他更高级的传感器。

13.3 人体感应传感器

- **前言：**

人体感应传感器，在室内安防应用非常普遍，其原理是由探测元件将探测到人体的红外辐射转变成微弱的电压信号，经过放大后输出。为了提高探测器的探测灵敏度以增大探测距离，一般在探测器的前方装设一个塑料的菲涅尔透镜，它和放大电路相配合，可将信号放大 70dB 以上，这样就可以测出 5~10 米范围内人的行动。

- **实验平台：**

pyWiFi-ESP32 开发套件和人体感应传感器模块。

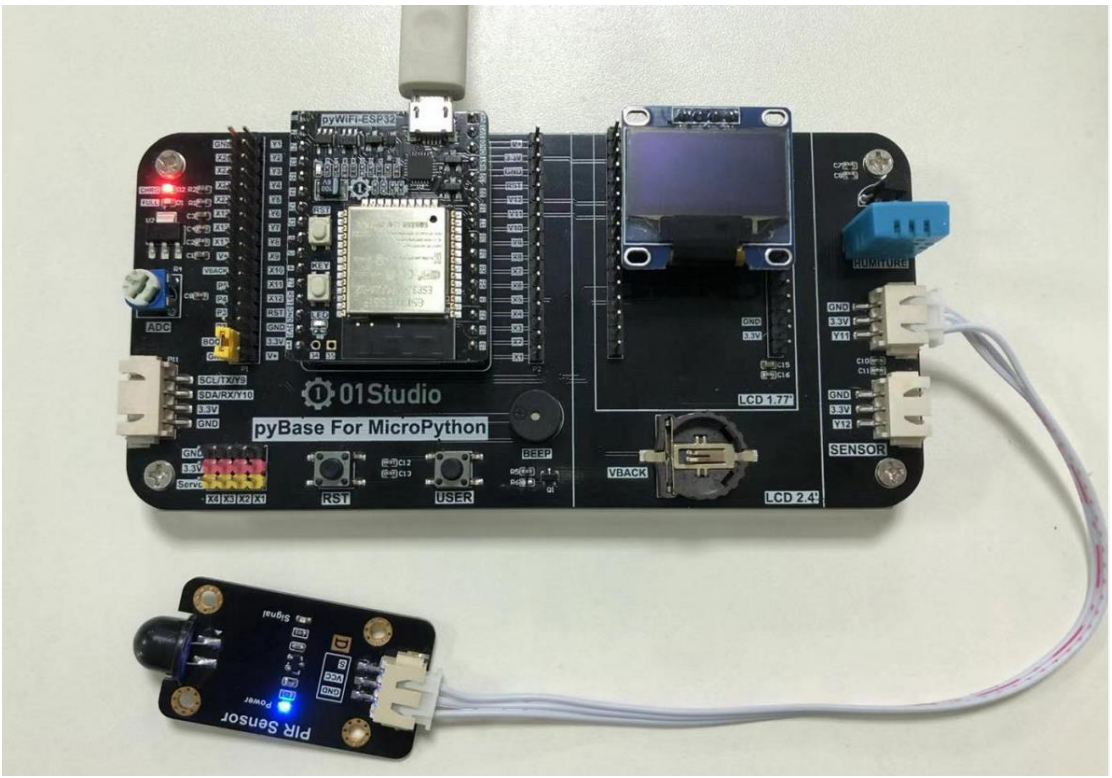


图 13-19 开发套件和传感器接线

- **实验目的：**

通过外部中断编程来检测人体感应模块，当有人出现时候 OLED 通过“Get People!!!”闪烁提示。

- **实验讲解：**

我们先来看看人体感应传感器的介绍：

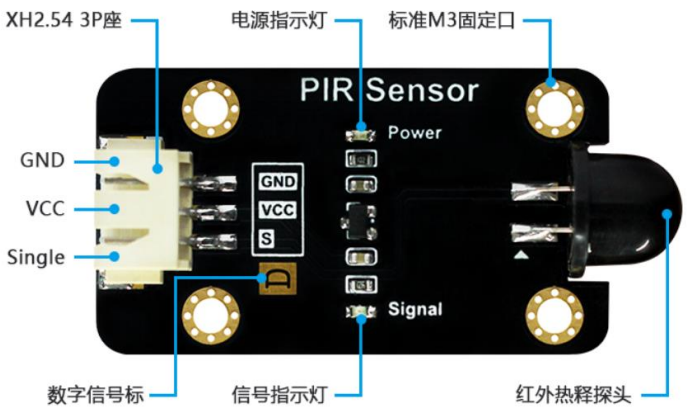


图 13-20 人体感应传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 65℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	数字信号： 检测到人体：高电平 3.3V；持续 3-8 秒 未检测到人体：低电平 0V。
感应角度	100°
感应距离	8 米
模块尺寸	4.5*2.5cm

图 13-21 人体感应传感器模块参数说明

从上表可以知，当检测到人体红外时候，传感器的输出信号为高电平并持续 3-5 秒。如下图所示：

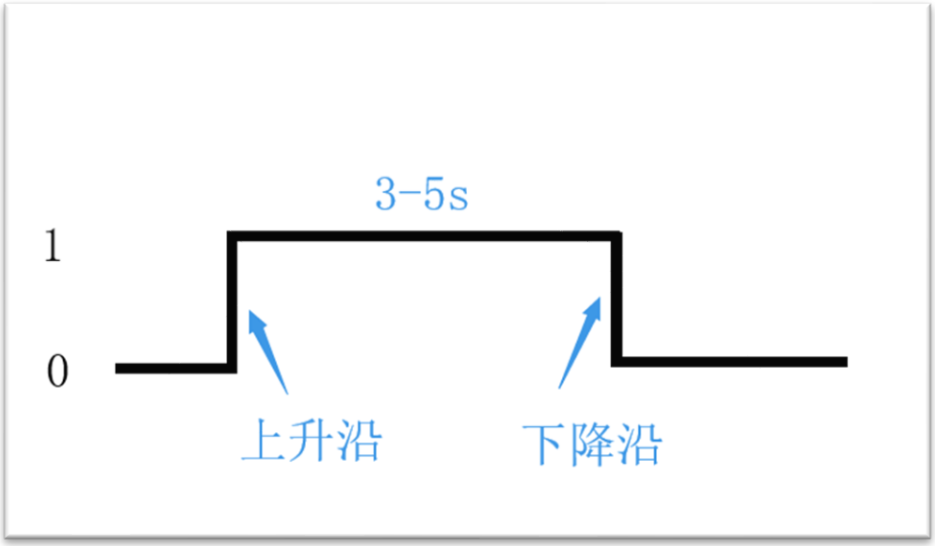


图 13-22 人体感应传感器电平变化

由此可见，可以使用外部中断结合上升沿的出发方式来编程实现相关功能。传感器的输出引脚连接 Sensor1 接口，即连接到 pyBase 的“Y11”引脚。也就是 pyWiFi-ESP32 的‘22’引脚。编程方法可以是当“22”引脚产生中断时候，说明传感器检测到人体红外线，此时可以在 OLED 上显示相关信息。（有关中断的介绍可以看上一章“外部中断”内容。）

关于外部中断的编程方法可以参考基础实验的外部中断章节，这里不再重复，具体的编程流程图如下：

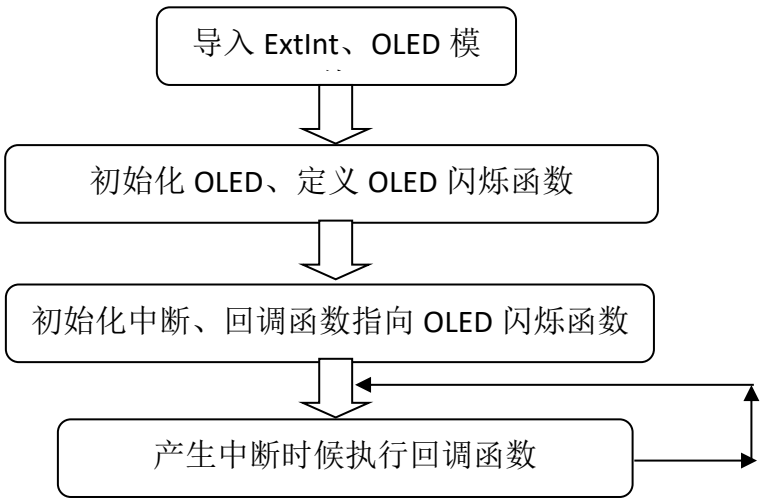


图 13-23 代码编写流程

参考例程代码如下：

```
'''
实验名称：人体感应传感器
版本：v1.0
日期：2019.8
作者：01Studio（www.01studio.org）
说明：人体红外感应传感器应用
'''

import time
from machine import I2C,Pin        #从 machine 模块导入 I2C、Pin 子模块
from ssd1306 import SSD1306_I2C    #从 ssd1306 模块中导入 SSD1306_I2C 子模块

#pyBoard I2C 初始化
i2c = I2C(sda=Pin(13), scl=Pin(14))
#OLED 显示屏初始化：128*64 分辨率,OLED 的 I2C 地址是 0x3c
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

Human=Pin(22,Pin.IN,Pin.PULL_UP) #构建人体红外对象

#OLED 初始信息显示
oled.fill(0)  # 清屏背景黑色
oled.text("01Studio", 0, 0)  # 写入第 1 行内容
oled.text("Human body test:", 0, 15)  # 写入第 2 行内容
oled.show()  # OLED 执行显示

def fun(Human): #Get People 闪烁 5 次效果!

    for i in range(5):
```

```
        oled.fill(0)  # 清屏背景黑色
        oled.text("01Studio", 0, 0)  # 写入第 1 行内容
        oled.text("Human body test:", 0, 15)  # 写入第 2 行内容
        oled.text("Get People!!!", 0, 40)  # 写入第 3 行内容
        oled.show()  # OLED 执行显示
        time.sleep_ms(500)

        oled.fill(0)  # 清屏背景黑色
        oled.text("01Studio", 0, 0)  # 写入第 1 行内容
        oled.text("Human body test:", 0, 15)  # 写入第 2 行内容
        oled.text("                ", 0, 40)  # 写入第 3 行内容
        oled.show()  # OLED 执行显示
        time.sleep_ms(500)

Human.irq(fun,Pin.IRQ_RISING) #定义中断，上升沿触发
```

- 实验结果：

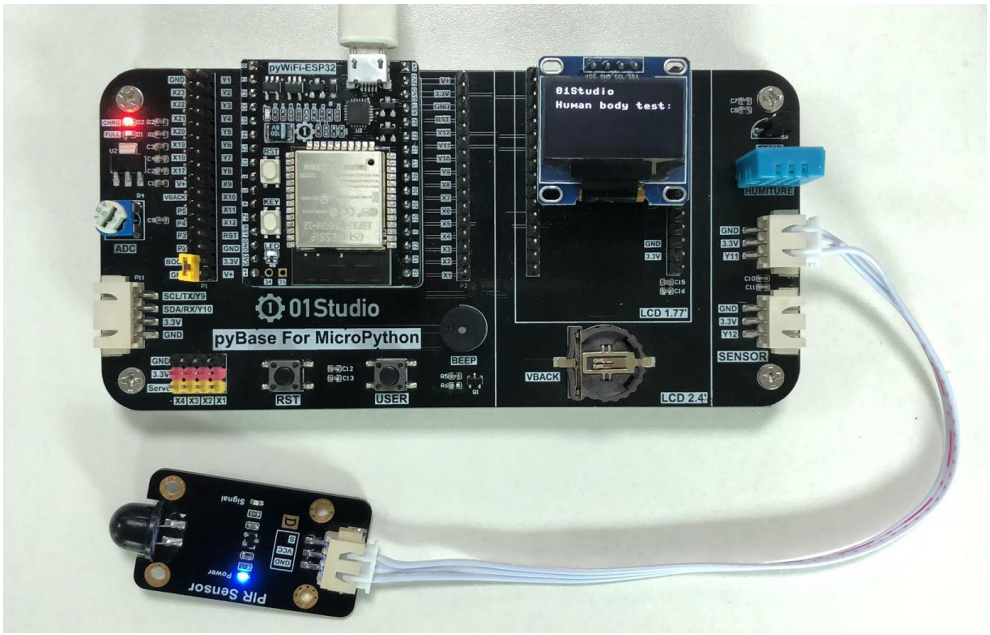


图 13-24 没有检测到到人体

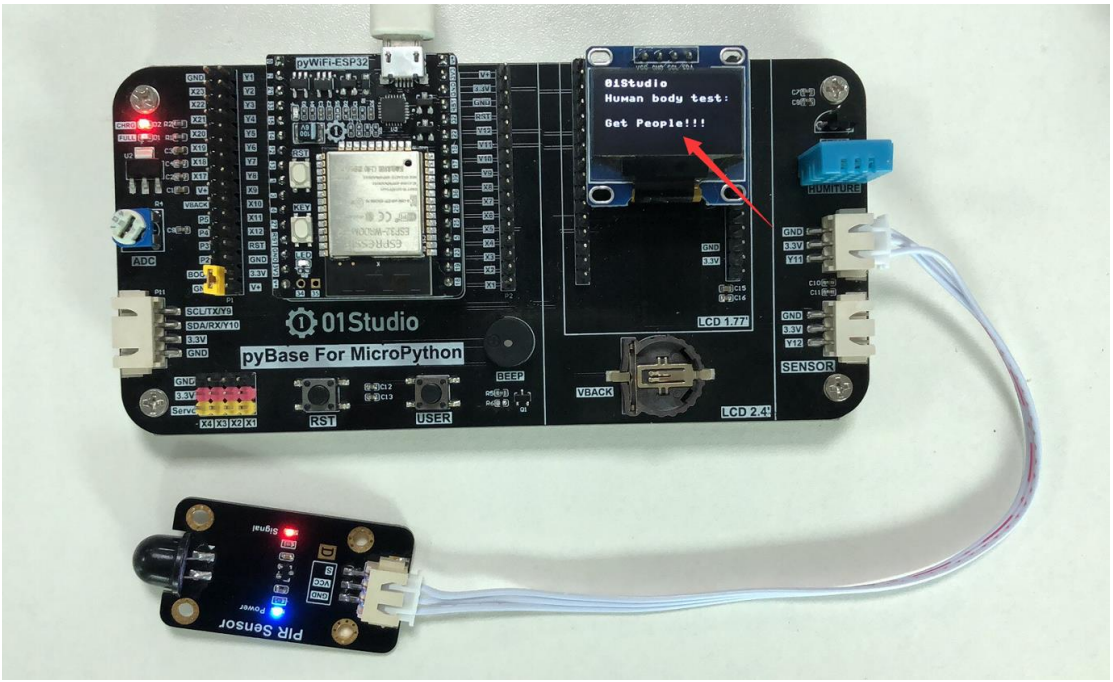


图 13-25 检测到人体，OLED 闪烁

- **总结：**

本节通过简单的中断方式便实现了对人体感应传感器的检测，人体感应传感器的应用非常广泛，特别是在安防领域，结合其它硬件模块可以实现当发现有人入侵时，执行发出警报、实现远程提醒等功能。

13.4 光敏传感器

- **前言：**

光敏传感器实际是一个款可以检测光照强度的传感器，可以应用于我们日常生活中植物光照强度、室内光线检测、以及某些场合的亮度检测。传感器的原理就是将外界模拟变化的信号转变成数字信号（电压值）让单片机出来，处理方式就是常用的 ADC（模拟数字转换）。

- **实验平台：**

pyWiFi-ESP32 开发套件开发套件和光敏传感器模块。

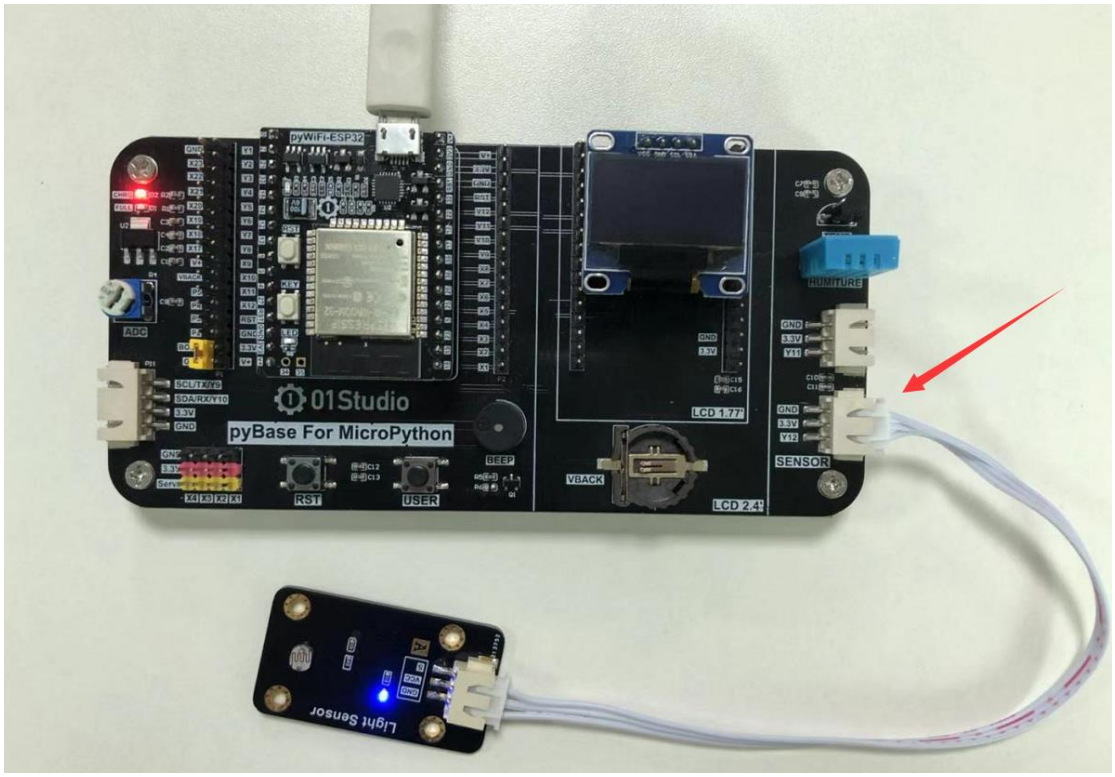


图 13-26 开发套件和光敏传感器接线

- **实验目的：**

采集当前环境的光照强度并在 OLED 显示，显示方式为：Bright-强，Normal-中等，Weak-弱。

- **实验讲解：**

我们先来看看光敏传感器模块的介绍：

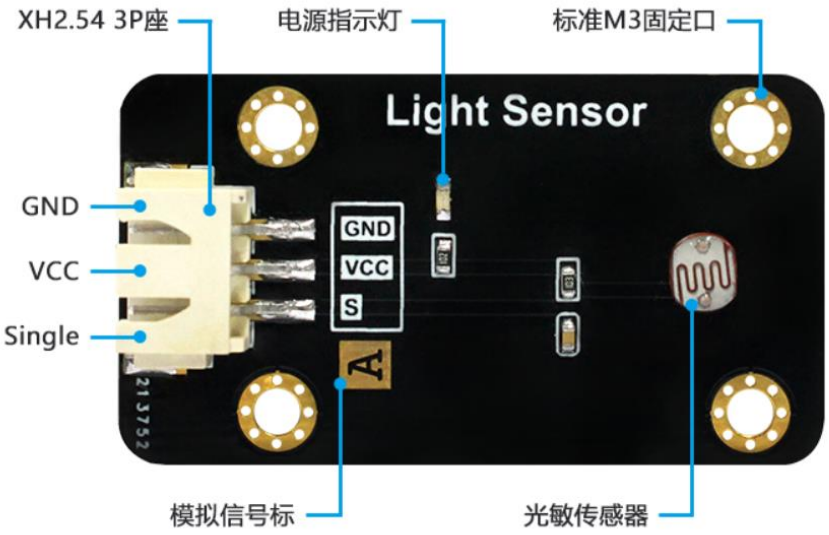


图 13-27 光敏传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	模拟信号：0-3.3V （VCC=3.3V 时）
模块尺寸	4.5*2.5 cm

表 13-2

从上表可以看到，光敏传感器输出的是模拟信号：0-3.3V，这代表了外界的光照强度。接近 0V 时光线最强，接近 3.3V 时，光线最弱。因此我们可以使用基础实验-ADC 学习过的内容来编程。

光敏传感器接在传感器接口 2，对应 pyBase 的引脚是“Y12”，即 pyWiFi-ESP32 的‘32’引脚。我们使用内部衰减器将测量电压量程从 0-1V 增加至 0-3.3V。（具体参考上一章“ADC”部分章节内容。）

ADC 使用 12bit 精度，即最大值为 2^12-1=4095。然后我们将检测到的数值 0-4095 分成三段，分别代表光线强：【0-1365】，中等：【1365-2730】，弱：【2730-

4095】。当然开发者可以根据自己实际情况来调整数值。编程流程图如下：

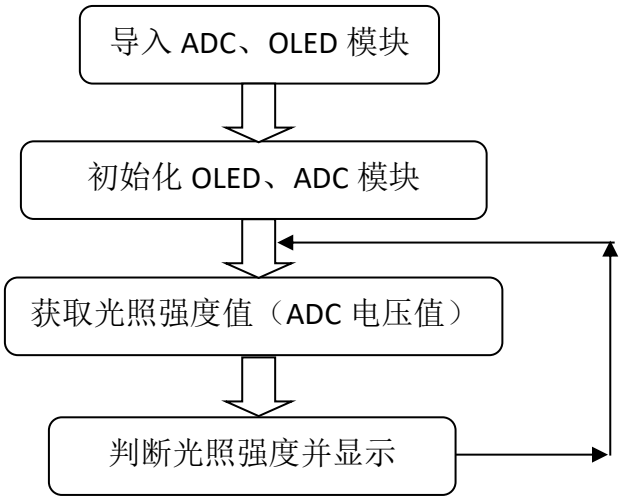


图 13-28 代码编写流程

参考程序代码如下：

```
'''
实验名称：光敏传感器
版本：v1.0
日期：2019.8
作者：01Studio 【www.01Studio.org】
说明：通过光敏传感器对外界环境光照强度测量并显示。
'''

#导入相关模块

from machine import Pin,I2C,ADC,Timer
from ssd1306 import SSD1306_I2C

#初始化相关模块
```

```
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化 ADC,Pin=36，11DB 衰减，测量电压 0-3.3V
Light = ADC(Pin(32))
Light.atten(ADC.ATTN_11DB)

#中断回调函数
def fun(tim):

    oled.fill(0)  # 清屏显示黑色背景
    oled.text('01Studio', 0, 0)  # 首行显示 01Studio
    oled.text('Light test:', 0, 15)      # 次行显示实验名称

    value=Light.read() #获取 ADC 数值

    #显示数值
    oled.text(str(value)+' (4095)',0,40)
    #计算电压值，获得的数据 0-4095 相当于 0-3.3V，（'%.2f'%）表示保留 2 位小数
    oled.text(str('%.2f'%(value/4095*3.3))+' V',0,55)

    #判断光照强度，分 3 档显示。
    if 0 <= value <=1365:
        oled.text('Bright', 60, 55)

    if 1365 < value <= 2730:
        oled.text('Normal', 60, 55)

    if 2730 < value <= 4095:
        oled.text('Weak  ', 60, 55)
```

```
    oled.show()


#开启 RTOS 定时器
tim = Timer(-1)
tim.init(period=1000, mode=Timer.PERIODIC, callback=fun) #周期 1s
```

- **实验结果：**

用手遮挡住光敏传感器，可以见到 OLED 显示光线强势是弱 Weak：

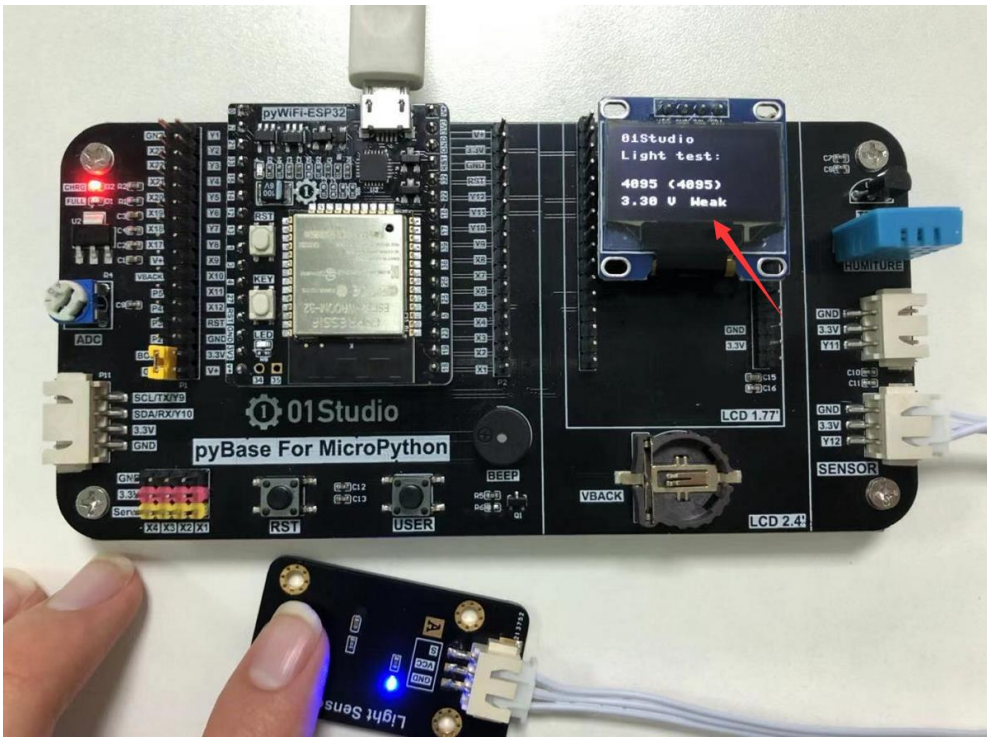


图 13-29 光线强度弱

打开手机手电筒，照在光敏传感器上，可以见到光线强度是强 Bright：

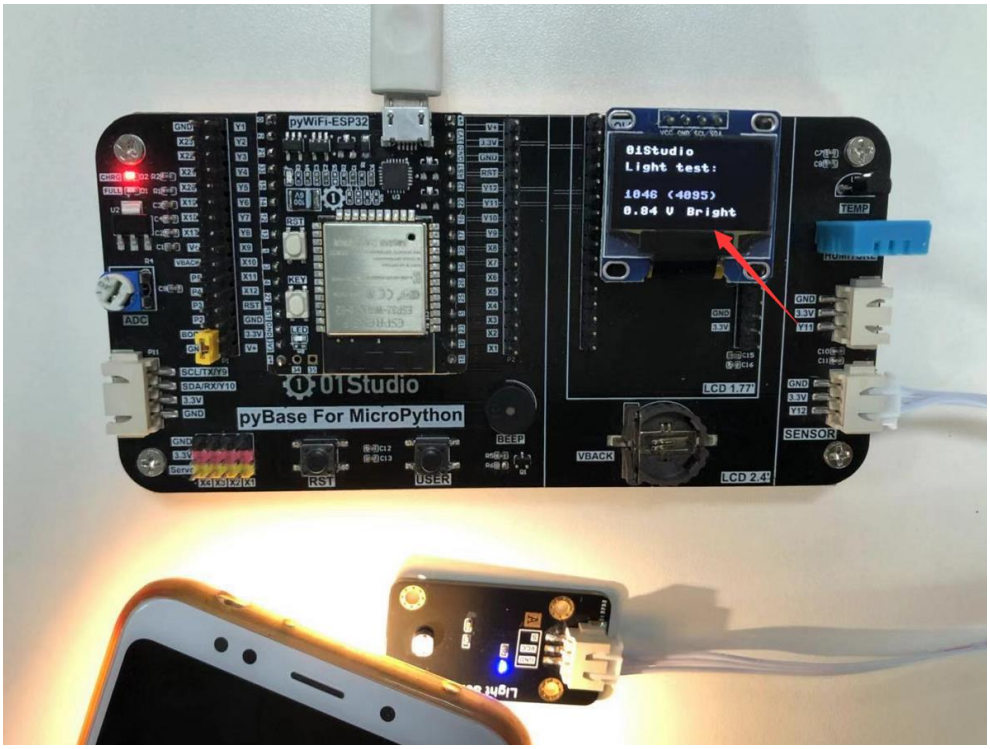


图 13-30 光线强度强

- **总结：**

从实验可以看到，光敏传感器背后的原理是对 ADC 的应用，实现了改功能后。我们可以自行扩展深入，制作自己喜欢的电子产品。

13.5 土壤湿度传感器

- **前言：**

土壤湿度传感器用于检测盆栽泥土的湿度，当泥土干枯时候，我们就需要给植物浇水了。这个用途非常广泛，如自动灌溉。

- **实验平台：**

pyWiFi-ESP32 开发套件和土壤湿度传感器模块。

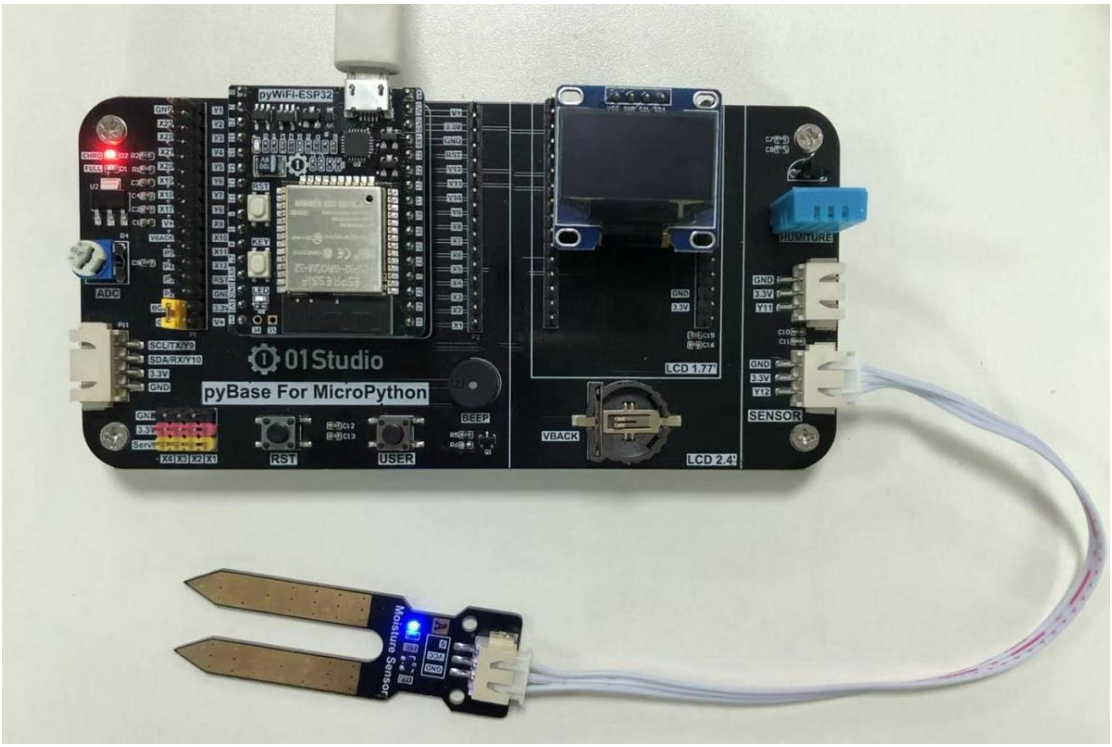


图 13-31 开发套件和传感器连接图

- **实验目的：**

采集盆栽土壤的的光照强度并在 OLED 显示，显示方式为：Dry-干，Normal-中等，Wet-湿。

- **实验讲解：**

我们先来看看土壤湿度传感器模块的介绍：

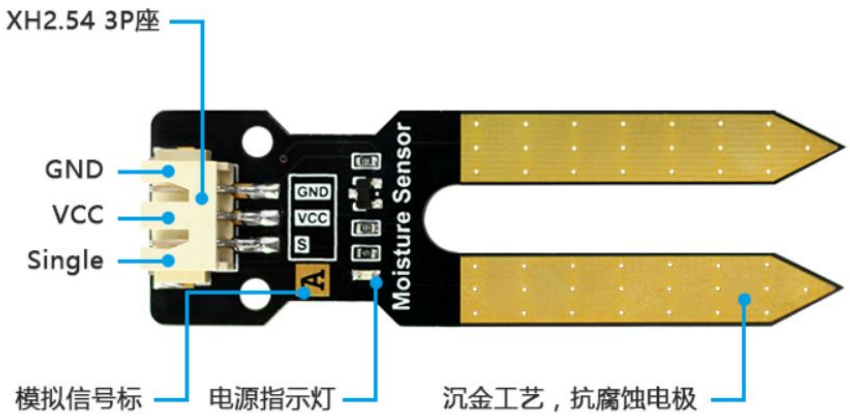


图 13-32 土壤湿度传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-40 至 85℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	模拟信号：0-3.3V （VCC=3.3V 时）
模块尺寸	6.6*2.0cm

表 13-3

从上表可以看到，土壤湿度传感器输出的是模拟信号：0-3.3V，这代表土壤的湿度情况。接近 0V 时湿度为干燥，接近 3.3V 时，湿度情况为湿润。因此我们可以使用基础实验-ADC 学习过的内容来编程。

土壤湿度传感器接在传感器接口 2，对应 pyBase 的引脚是“Y12”，也就是 pyWiFi-ESP32 的‘32’引脚，ADC 使用内部 11DB 衰减器，将量程增至 0-3.3V，使用 12bit 精度，即最大值为 2^12-1=4095。然后我们根据实际测试数据将检测到的数值 0-4095 分成三段，分别代表土壤干燥:【0-1247】，中等:【1247-2238】，湿润:【2238-4095】。当然开发者可以根据自己实际情况来调整数值。编程流程图如下：

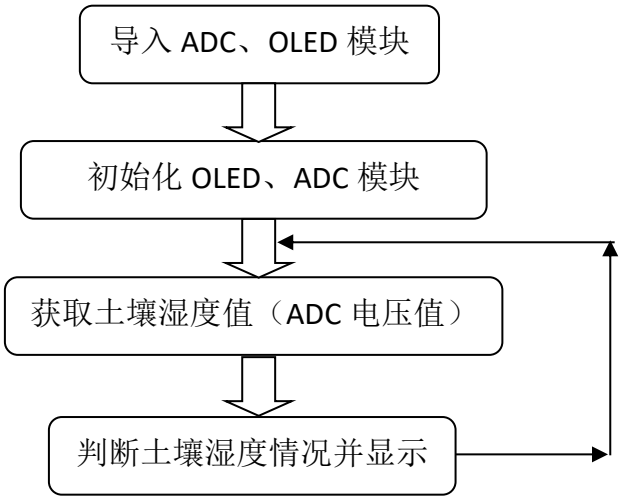


图 13-33 代码编写流程

参考程序代码如下：

```
'''
实验名称：土壤湿度传感器
版本：v1.0
日期：2019.8
作者：01Studio 【www.01Studio.org】
说明：通过土壤湿度传感器对土壤湿度测量并显示。
'''

#导入相关模块

from machine import Pin,I2C,ADC,Timer
from ssd1306 import SSD1306_I2C

#初始化相关模块
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
```

```
#初始化 ADC,Pin 是 32，开启 11DB 衰减，量程增至 0-3.3V
Soil = ADC(Pin(32))
Soil.atten(ADC.ATTN_11DB)



#中断回调函数
def fun(tim):



    oled.fill(0)  # 清屏显示黑色背景
    oled.text('01Studio', 0, 0)  # 首行显示 01Studio
    oled.text('Soil test:', 0, 15)      # 次行显示实验名称

    value=Soil.read() #获取 ADC 数值

    #显示数值
    oled.text(str(value)+' (4095)',0,40)
    #计算电压值，获得的数据 0-4095 相当于 0-3.3V，（'%.2f'%）表示保留 2 位小数
    oled.text(str('%.2f'%(value/4095*3.3))+' V',0,55)

    #判断土壤湿度，分 3 档显示。
    if 0 <= value <=1247:
        oled.text('Dry', 60, 55)

    if 1247 < value <= 2238:
        oled.text('Normal', 60, 55)

    if 2238 < value <= 4095:
        oled.text('Wet  ', 60, 55)
```

```
    oled.show()


#开启 RTOS 定时器
tim = Timer(-1)
tim.init(period=1000, mode=Timer.PERIODIC, callback=fun) #周期 1s
```

- **实验结果：**

将传感器插到干燥的土壤中，可以见到 OLED 显示干燥-Dry：

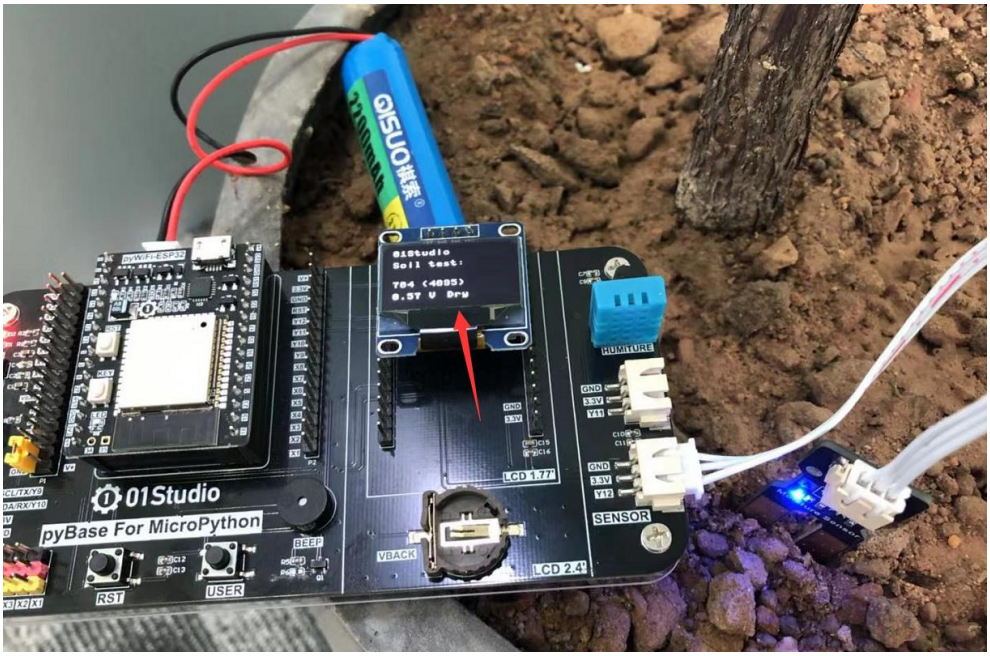


图 13-34 干燥的土壤

从传感器侧面往土壤浇水，让土壤变得湿润。**注意不要浇到传感器电路上！**

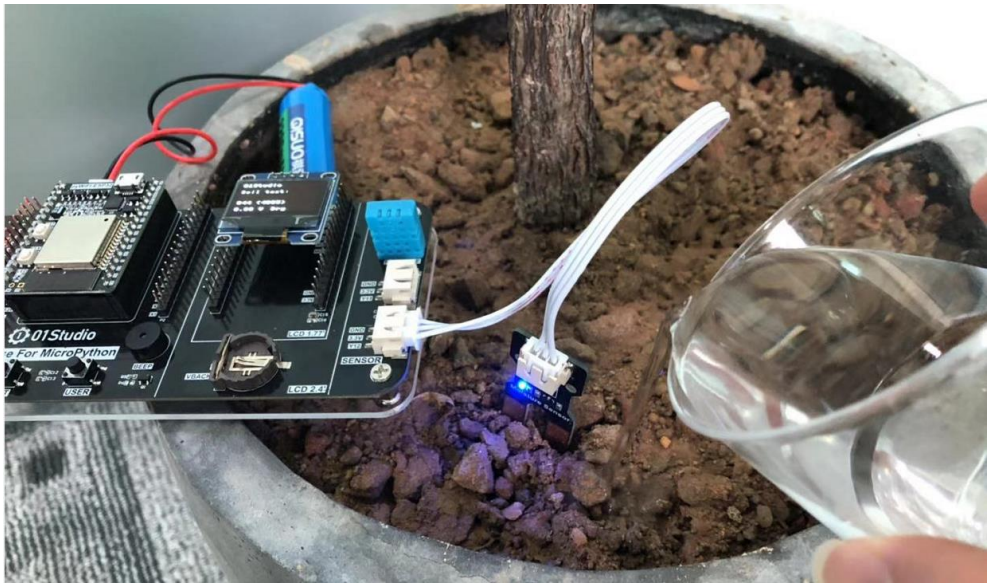


图 13-35 浇水

可以看到 OLED 显示湿润-Wet:

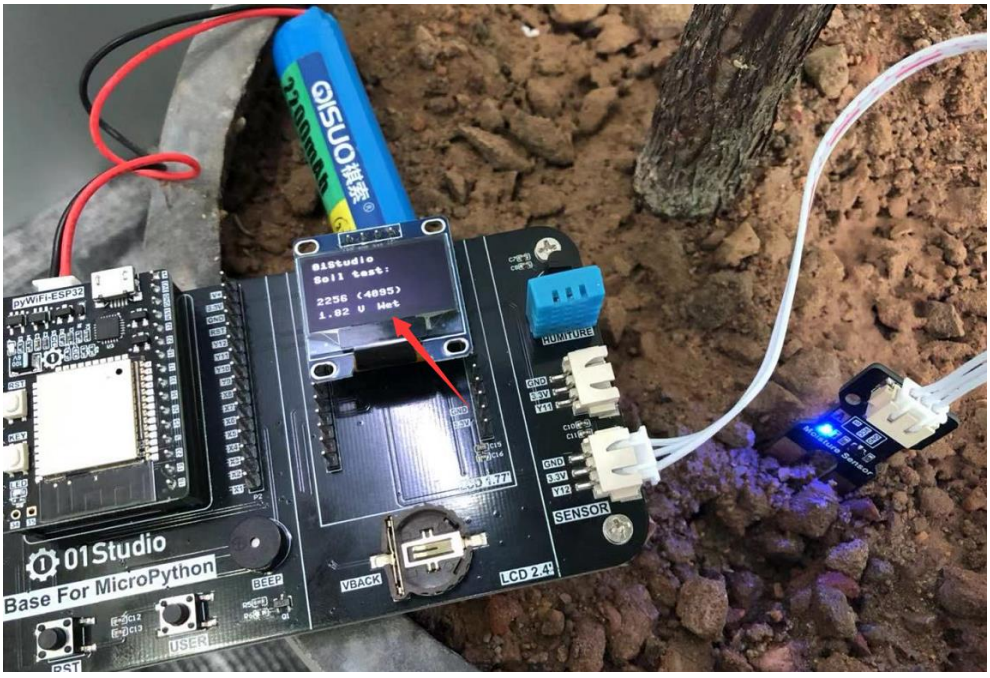


图 13-36 湿润的土壤

- **总结：**

从实验可以看到，土壤湿度传感器背后的原理是对 ADC 的应用，实现了该功能后。我们可以自行扩展深入，制作自己喜欢的电子产品。

13.6 水位传感器

- **前言：**

水位（液位）传感器是一款简单易用、小巧轻便、水位/水滴识别检测传感器，传感器通过一系列的暴露的平行导线线迹测量其水滴/水量大小使得传感器输出的模拟信号产生相应的变化，轻松完成水量到模拟信号的转换，从而判断水位。

- **实验平台：**

pyWiFi-ESP32 开发套件和水位传感器模块。

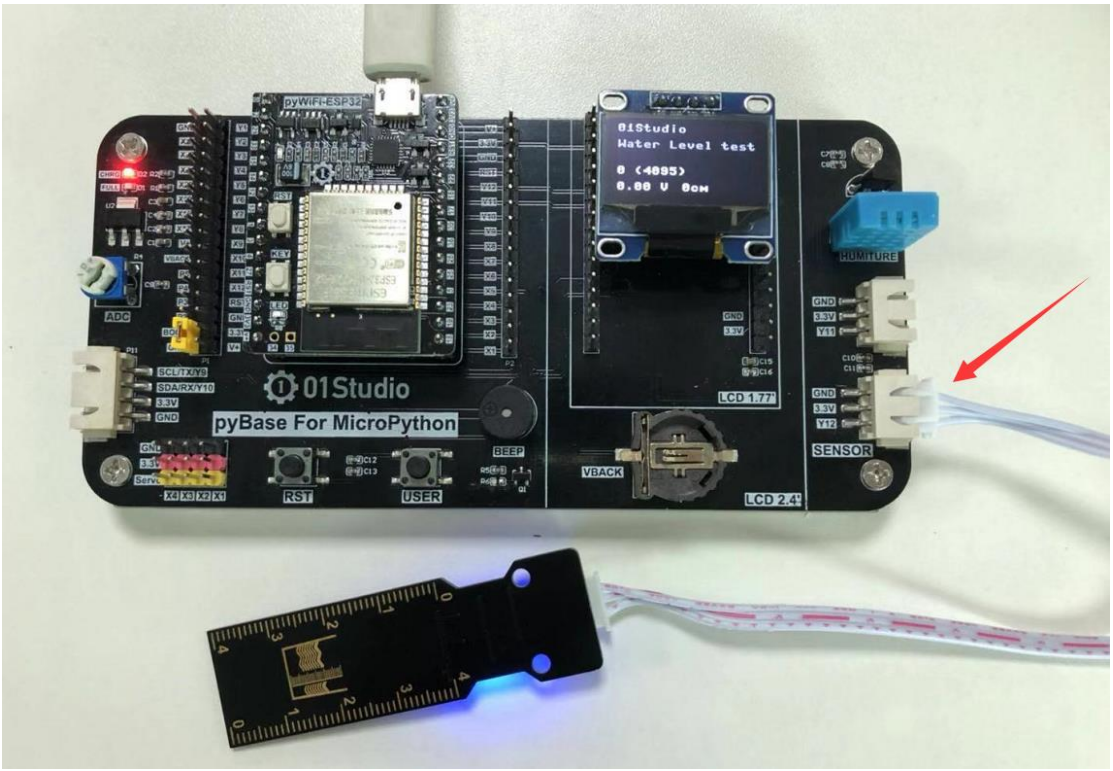


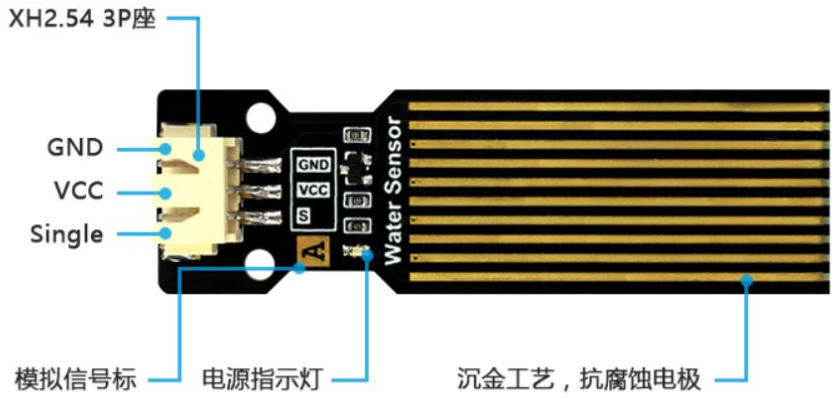
图 13-37 开发套件和水位传感器接线图

- **实验目的：**

测量水位高度并在 OLED 显示，显示方式为：0-4cm。

- **实验讲解：**

我们先来看看水位传感器的介绍：



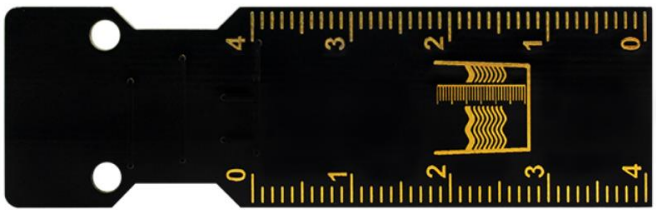


图 13-38 水位传感器模块正反面

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	0 至 60℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	模拟信号：0-3.3V （VCC=3.3V 时）
模块尺寸	6.6*2.0cm
注意事项	水位传感器接口旁元件不能接触水，否则容易短路。所以测量时候要注意液体水位的高度。

图 13-39

从上表可以看到，水位传感器的量程为 4cm，输出的是模拟信号：0-3.3V（蒸馏水实测 0-1V）。这代表水位高度情况。接近 0V 时湿度为 0cm 或没有放进液体，电压升高，意味着水位升高。因此我们可以使用基础实验-ADC 学习过的内容来编程。

水位传感器接在传感器接口 2，对应 pyBase 的引脚是“Y12”。也就是 pyWiFi-ESP32 的引脚‘32’。ADC 使用默认量程 0-1V，12bit 精度，即最大值为 2^12-1=4095。我们使用的是普通桶装水，根据实际测试数据将检测到的数值 0-4095 分成 5 档。分别代表水位高度 0cm：【0-1300】，1cm：【1300-2300】，2cm：【2300-3300】，3cm：【3300-3800】，4cm：【>3800】。

当然开发者可以根据自己测量液体不同或其它情况来调整数值。编程流程图如下：

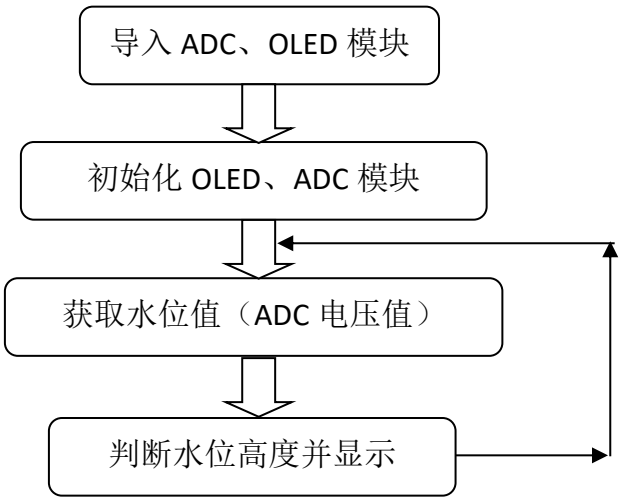


图 13-40 代码编写流程

参考程序代码如下：

```
'''
实验名称：水位传感器
版本：v1.0
日期：2019.8
作者：01Studio 【www.01Studio.org】
说明：通过水位传感器对水位测量并显示。
'''

#导入相关模块
from machine import Pin,I2C,ADC,Timer
from ssd1306 import SSD1306_I2C
```

```
#初始化相关模块
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化 ADC,Pin 是引脚 32，默认量程 0-1V
Water_level = ADC(Pin(32))

#中断回调函数
def fun(tim):

    oled.fill(0)  # 清屏显示黑色背景
    oled.text('01Studio', 0, 0)  # 首行显示 01Studio
    oled.text('Water Level test', 0, 15)      # 次行显示实验名称

    value=Water_level.read() #获取 ADC 数值

    #显示数值
    oled.text(str(value)+' (4095)',0,40)
    #计算电压值，获得的数据 0-4095 相当于 0-1V，（'%.2f'%）表示保留 2 位小数
    oled.text(str('%.2f'%(value/4095))+' V',0,55)

    #判断水位，分 5 档显示，0-4cm
    if 0 <= value <=1300:
        oled.text('0cm', 60, 55)

    if 1300 < value <= 2300:
        oled.text('1cm', 60, 55)

    if 2300 < value <= 3300:
```

```
        oled.text('2cm', 60, 55)


    if 3300 < value <= 3800:
        oled.text('3cm', 60, 55)


    if 3800 <= value:
        oled.text('4cm', 60, 55)


    oled.show()


#开启 RTOS 定时器
tim = Timer(-1)
tim.init(period=1000, mode=Timer.PERIODIC, callback=fun) #周期 1s
```

- **实验结果：**

  当没插入水中时，OLED 显示当前水位为 0cm:



图 13-41

将水位传感器模块慢慢放入水中，可以看到 OLED 显示从 1cm-4cm 变化。

（注意量程是 4cm，请勿将超出的电路部分放置水中以免发生短路。）

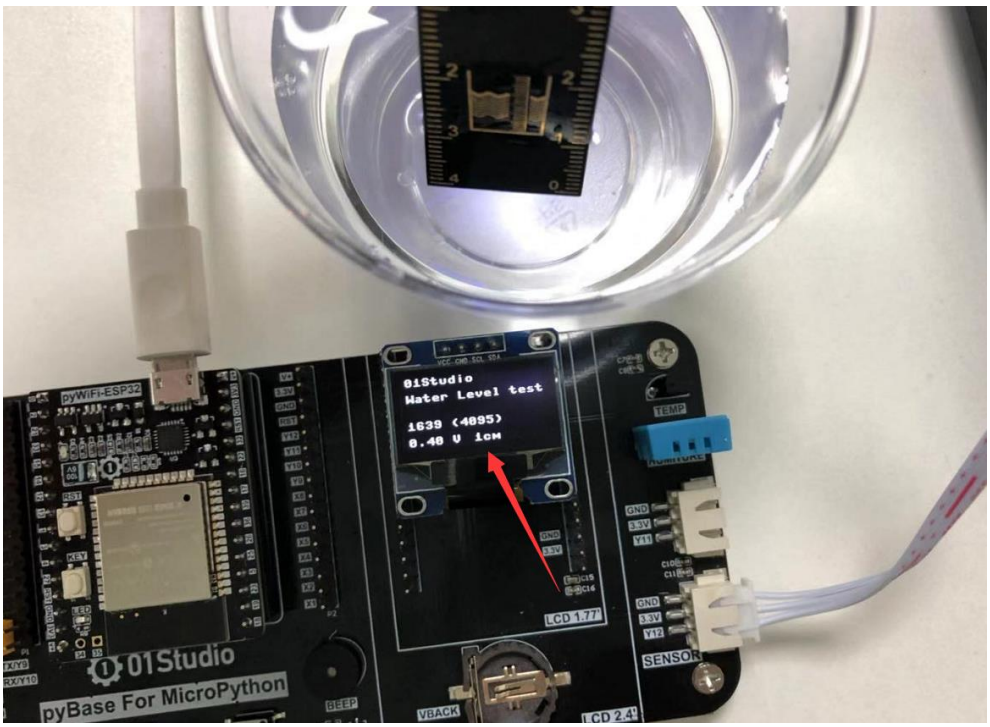


图 13-42 水位接近 1cm

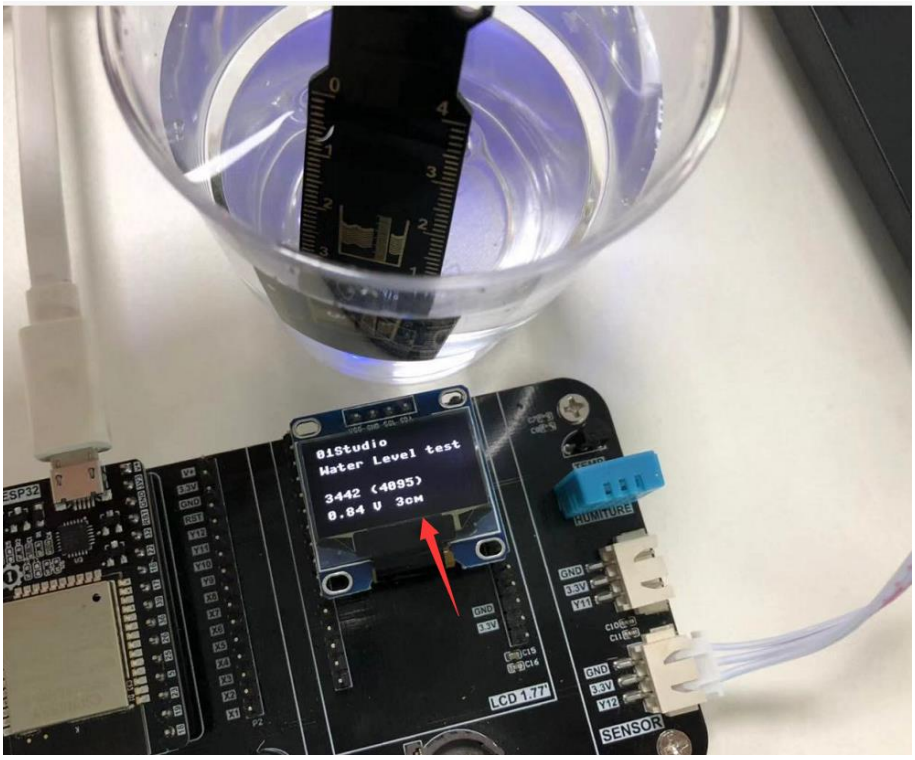


图 13-43 水位接近 3cm

- **总结：**

从实验可以见到，这款水位传感器出了可以用来测量水位（液位）高度外，还可以用来测量室外下雨情况和家里的淹水情况。有兴趣的小伙伴可以动手制作一下！

13.7 大气压强传感器

- **前言：**

本实验中大气压强传感器模块使用的是 BMP280，这是一款专为移动应用而设计的高精度大气压传感器， 传感器模块采用极其紧凑的封装，其小尺寸和低功耗可以应用在手机、 GPS 模块、 手表的电池供电设备中。 BMP280 传感器除了可以测量大气压强，还可以测量温度（温度精度不高）以及通过计算公式来换算出海拔高度。

- **实验平台：**

pyWiFi-ESP32 开发套件和大气压强传感器模块，传感器连接到 I2C/UART 扩展接口。

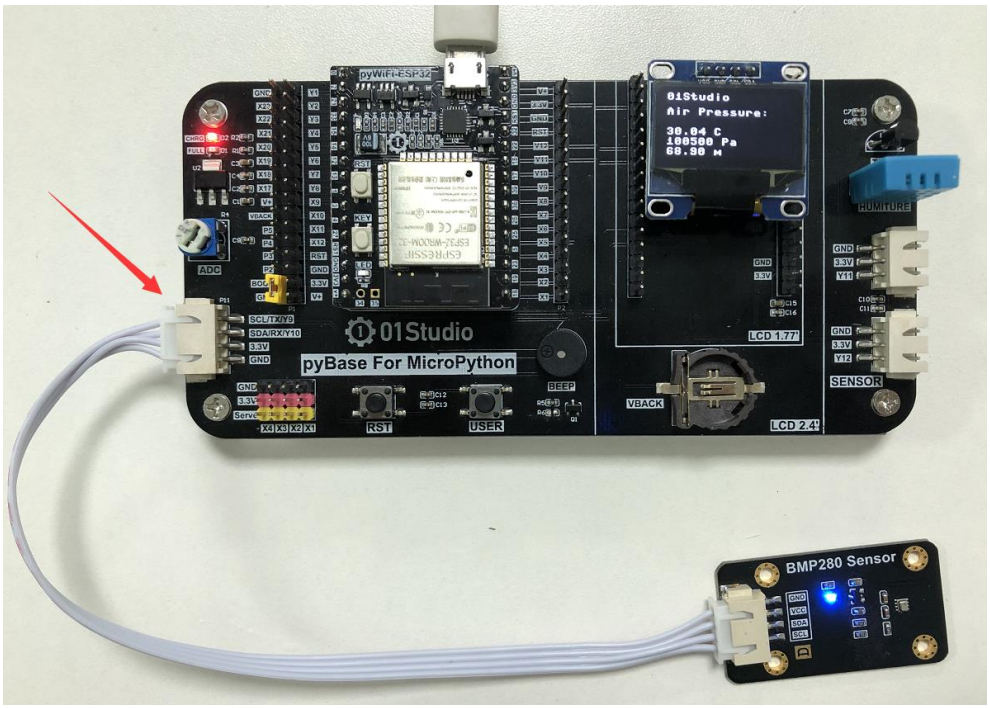


图 13-44 开发套件和大气压强传感器接线方法

- **实验目的：**

编程测量当前环境的大气压强、温度信息，将大气压强通过公式计算出当前海拔高度，并在 OLED 显示。打造自己的气压计！

- **实验讲解：**

我们先来看看大气压强传感器模块的介绍：

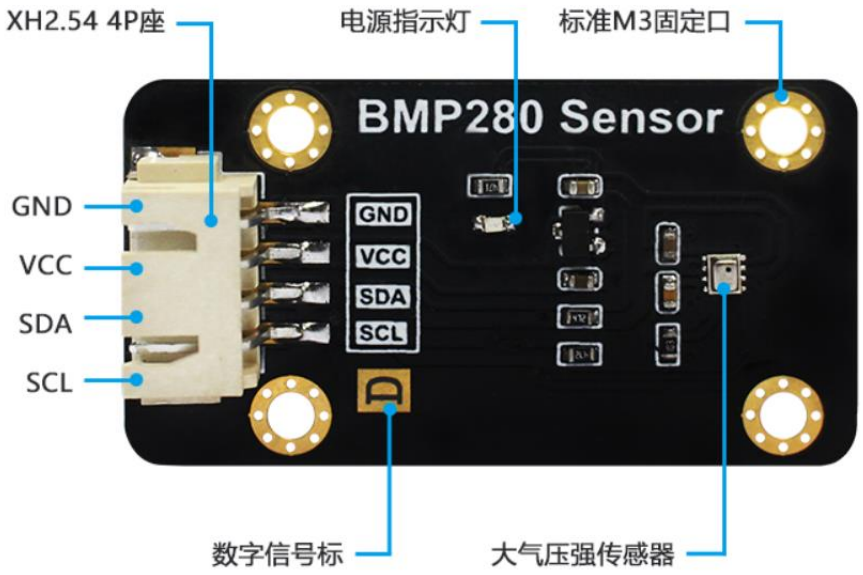


图 13-45 大气压强传感器模块

功能参数	
传感器型号	BMP280
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85℃
接口定义	XH2.54 防呆接口（4Pin）【GND、VCC、SDA、SCL】
通讯信号	I2C 总线
I2C 地址	0x76（BMP280 的 SDO 默认下拉）； 当 BMP280 的 SDO 引脚上拉时，I2C 地址为：0x77
模块尺寸	4.5*2.5cm

图 13-46

从上面介绍可以看到，BMP280 是一款通过 I2C 接口驱动的传感器。连接到 pyBase 左下角的 I2C 外扩接口上。我们通过前面学习的 I2C 接口使用的方式，即可以对该模块实现数据通讯。

标准大气压是指把温度为 0℃、纬度 45 度海平面（海拔为 0 米）上的气压

称为 1 个大气压，其数值为 101325 帕斯卡（Pa）。

大气压同海拔高度的关系：P=P0×（1-H/44300)^5.256

因此计算高度公式为：H=44300*(1- (P/P0)^(1/5.256))

式中：H 为海拔高度，P0=大气压（0℃，101325Pa）

从上面公式可以看到，高度是通过大气压强换算出来的，从物理学的角度我们可以知道，高度越高的地方，空气越稀薄，大气压强越低。通过气压的变化我们就可以计算出海拔高度；但是这存在特定条件，那就是温度为 0℃的时候，而温度越高的地方，空气越稀薄，大气压强就越低。因此高度数据理论上需要做温度补偿，因此本实验的高度值换算存在轻微误差。有兴趣的小伙伴可以自行深入研究。

MicroPython 的强大之处是其有丰富的模块和函数库，一旦模块建立了，那么后来者使用起来就非常简单，无须再去做底层驱动的开发。从而实现面向对象的编程，而当有需要的时候又可以去改底层代码，可以说是进可攻退可守，非常灵活。在这里我们直接调用已经编写好的驱动文件 bmp280.py，该文件实现了对大气压、温度、高度的测量和计算。用户直接使用即可，具体如下：

构造函数	说明
bmp280.BMP280(I2C)	定义传感器的 I2C 接口。
使用方法	**说明**
getTemp()	获取温度数据
getPress()	获取气压数据
getAlititude()	获取高度数据

表 13-4 大气压强传感器 BMP280 对象

打开示例程序的 bmp280.py 文件，找到 getAltitude() 的定义函数，可以看到计算公式跟前面分析的一样，这里保留了 2 位小数：

```
    # Calculating absolute altitude
    def getAltitude(self):
        return '%.2f'%(44330*(1-(self.getPress()/101325)**(1/5.256)))
```

理解了传感器原理和模块使用方法后，我们可以整理出编程思路，流程图如下：

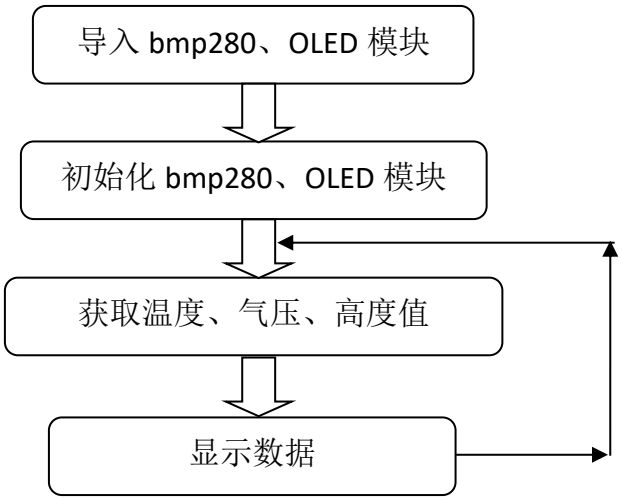


图 13-47 代码编写流程

参考程序代码如下：

```
'''
实验名称：大气压强传感器
版本：v1.0
日期：2019.8
作者：01Studio 【www.01Studio.org】
说明：测量 BMP280 温度、气压和计算海拔值，并在 OLED 上显示。。
'''

import time
import bmp280
```

```
from machine import Pin,I2C,Timer
from ssd1306 import SSD1306_I2C

#初始化 OLED
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#初始化 BMP280，定义第二个 I2C 接口 i2c2 用于连接 BMP280
i2c2 = I2C(sda=Pin(16), scl=Pin(17))
BMP = bmp280.BMP280(i2c2)

#中断回调函数
def fun(tim):

    oled.fill(0)  # 清屏,背景黑色
    oled.text('01Studio', 0, 0)
    oled.text('Air Pressure:', 0, 15)

    # 温度显示
    oled.text(str(BMP.getTemp()) + ' C', 0, 35)
    # 湿度显示
    oled.text(str(BMP.getPress()) + ' Pa', 0, 45)
    # 海拔显示
    oled.text(str(BMP.getAltitude()) + ' m', 0, 55)

    oled.show()

#开启 RTOS 定时器
tim = Timer(-1)
tim.init(period=1000, mode=Timer.PERIODIC, callback=fun) #周期 1s
```

- **实验结果：**

程序烧写完复位后可以见到实验结果。

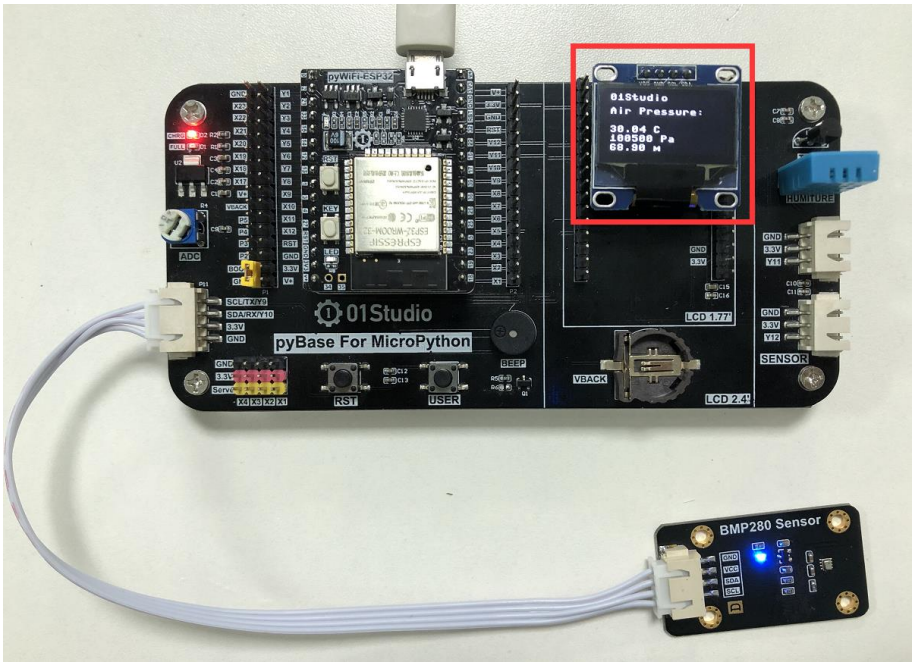


图 13-48 气压测量

开发板接上锂电池供电，尝试上下楼，会发现气压和海拔数值相应的变化。

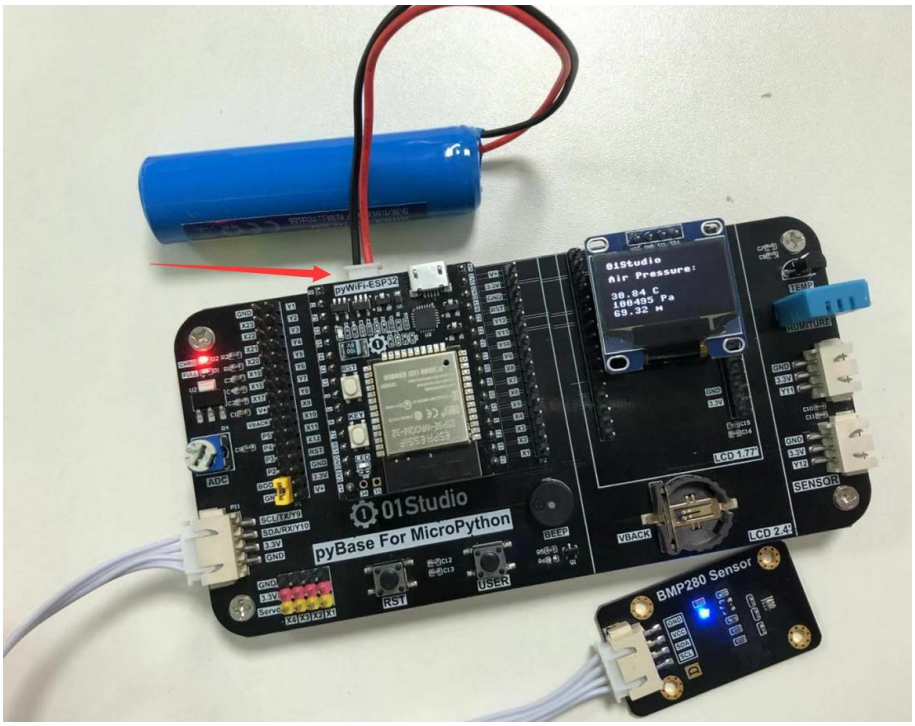


图 13-49 锂电池供电

- **总结：**

本节实现了大气压强传感器模块 BMP280 的应用，而且实验结果的精度很高，有兴趣的小伙伴可以结合自己情况，打造一个属于自己的气压计，然后到户外暴走！

13.8 超声波传感器

- **前言：**

超声波传感器是一款测量距离的传感器。其原理是利用声波在遇到障碍物反射接收结合声波在空气中传播的速度计算的得出。在测量、避障小车，无人驾驶等领域都有相关应用。

- **实验平台：**

pyWiFi-ESP32 开发套件和超声波传感器模块，传感器模块连接到 I2C/UART 扩展接口。

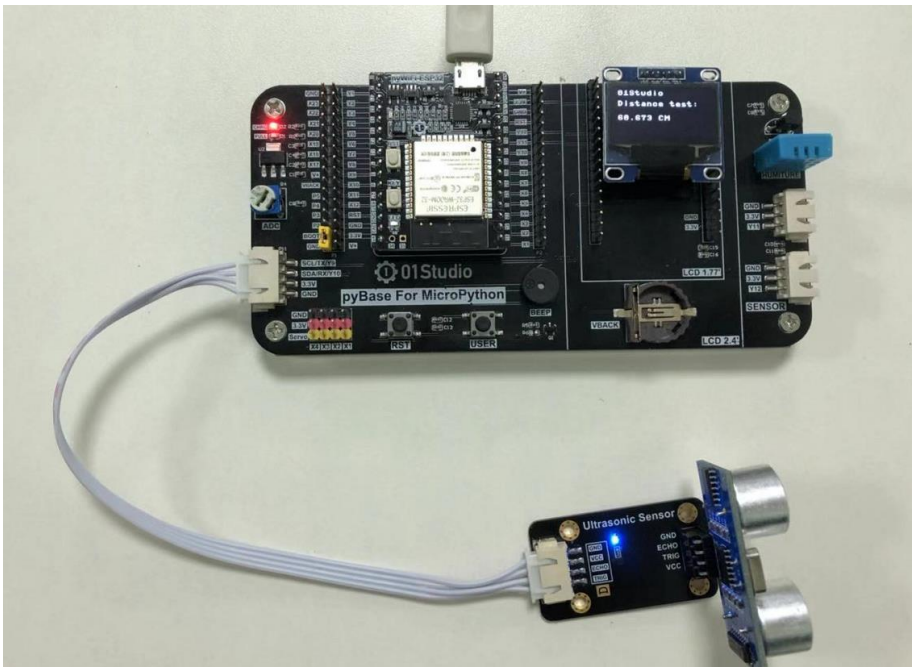


图 13-50 开发套件与传感器连接

- **实验目的：**

测量距离并在 OLED 上显示相关数据。

- **实验讲解：**

我们先来看看超声波传感器模块的介绍：

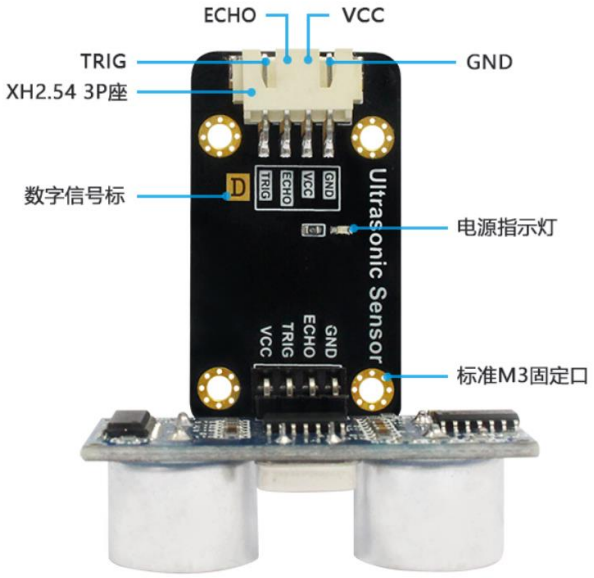


图 13-51 超声波模块

功能参数	
传感器型号	HCSR04
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85℃
接口定义	XH2.54 防呆接口（4Pin） 【GND、VCC、ECHO、TRIG】
通讯信号	IO 数字接口
测量距离	2-450 cm
测量精度	0.5 cm
整体尺寸	5.5*4.5*3.0 cm

表 13-5

超声波传感器模块使用两个 IO 口分别控制超声波发送和接收，工作原理如下：

6. 给超声波模块接入电源和地；
7. 给脉冲触发引脚（trig）输入一个长为 20us 的高电平方波；

8. 输入方波后，模块会自动发射 8 个 40KHz 的声波，与此同时回波引脚（echo）端的电平会由 0 变为 1；（此时应该启动定时器计时）
9. 当超声波返回被模块接收到时，回波引 脚端的电平会由 1 变为 0；（此时应该停止定时器计数），定时器记下的这个时间即为超声波由发射到返回的总时长；
10. 根据声音在空气中的速度为 340 米/秒，即可计算出所测的距离。

要学习和应用传感器，学会看懂传感器的时序图是很关键的，所以我们来看一下超声波传感器 HCSR04 的时序触发图。

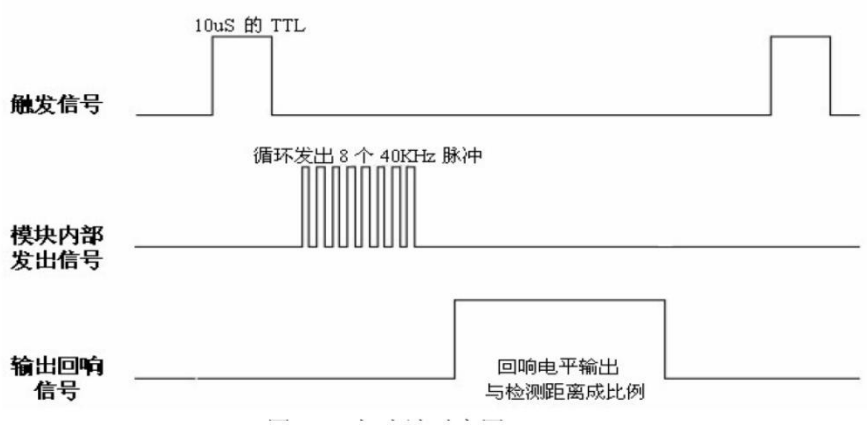


图 13-52 超声波传感器 HCSR04 时序图

以上普及了超声波传感器的原理，我们已经将其集成在 HCSR04.py 文件，如想了解代码原理可以打开 HCSR04.py 文件查看代码实现原理。使用 MicroPython 开发的用户只需要直接使用即可。使用方法如下：

构造函数	说明
HCSR04.HCSR04(trig,echo)	定义连接传感器的 trig 和 echo 引脚
使用方法	说明
getDistance()	获取距离数据

表 13-6 超声波传感器 HCSR04 对象

从上表看到，超声波传感器的使用方法非常简单，使用 2 个代码就完成对距离信息的采集，代码编写流程如下：

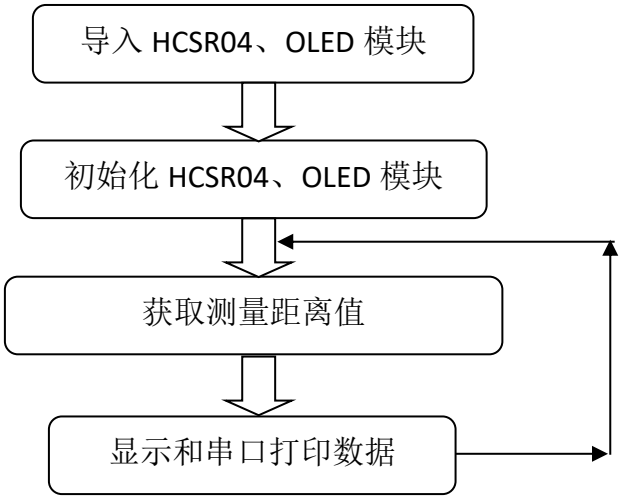


图 13-53 代码编写流程

主程序 main.py 参考代码如下：

```
'''
实验名称：超声波传感器
版本：v1.0
日期：2019.8
作者：01Studio 【www.01Studio.org】
说明：通过超声波传感器测距，并在 OLED 上显示。
'''

from HCSR04 import HCSR04     #子文件夹下的调用方式
from machine import Pin,I2C,Timer
from ssd1306 import SSD1306_I2C

#初始化 OLED
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
```

```
#初始化接口 trig=17,echo=16
trig = Pin(17,Pin.OUT)
echo = Pin(16,Pin.IN)
HC=HCSR04(trig,echo)

#中断回调函数
def fun(tim):

    oled.fill(0)  # 清屏,背景黑色
    oled.text('01Studio', 0, 0)
    oled.text('Distance test:', 0, 15)

    Distance = HC.getDistance() #测量距离

    # OLED 显示距离
    oled.text(str(Distance) + ' CM', 0, 35)

    oled.show()

    #串口打印
    print(str(Distance)+' CM')

#开启 RTOS 定时器
tim = Timer(-1)
tim.init(period=1000, mode=Timer.PERIODIC, callback=fun) #周期 1s
```

- **实验结果：**

在超声波传感器 30cm 外放置障碍物，可以看到 OLED 显示屏实时显示距离数据约为 30cm。

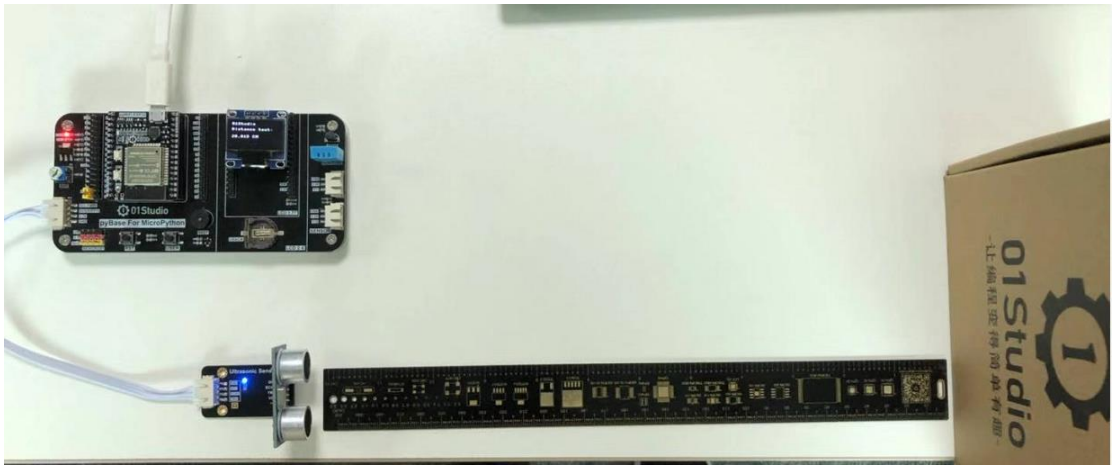


图 13-54 障碍物距离为 30cm

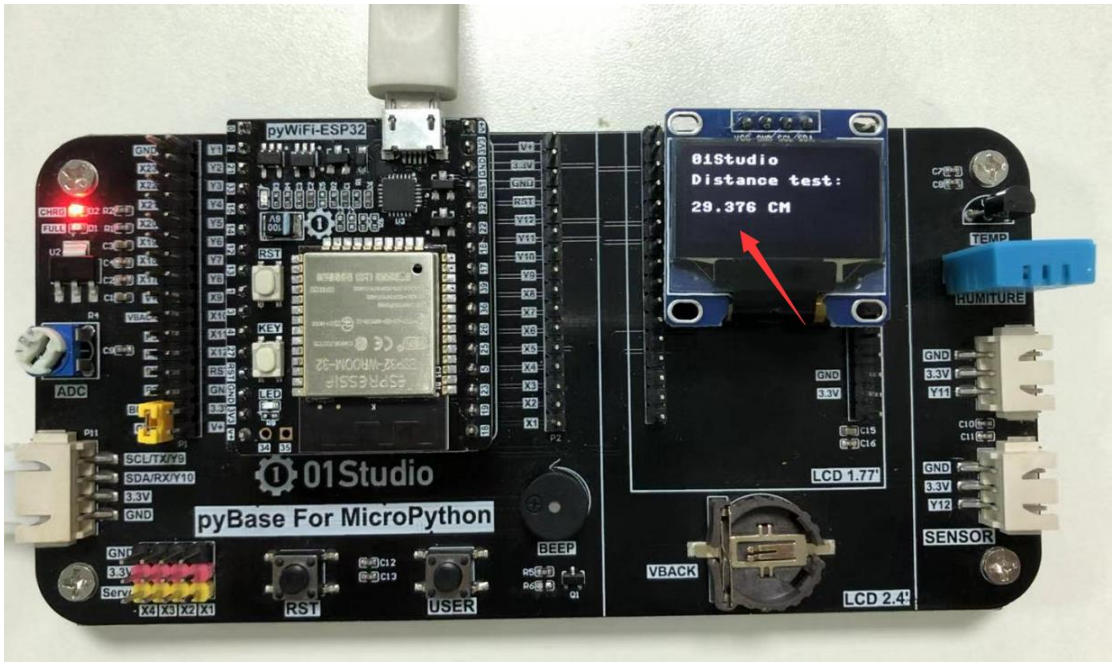


图 13-55 实验结果约为 30cm

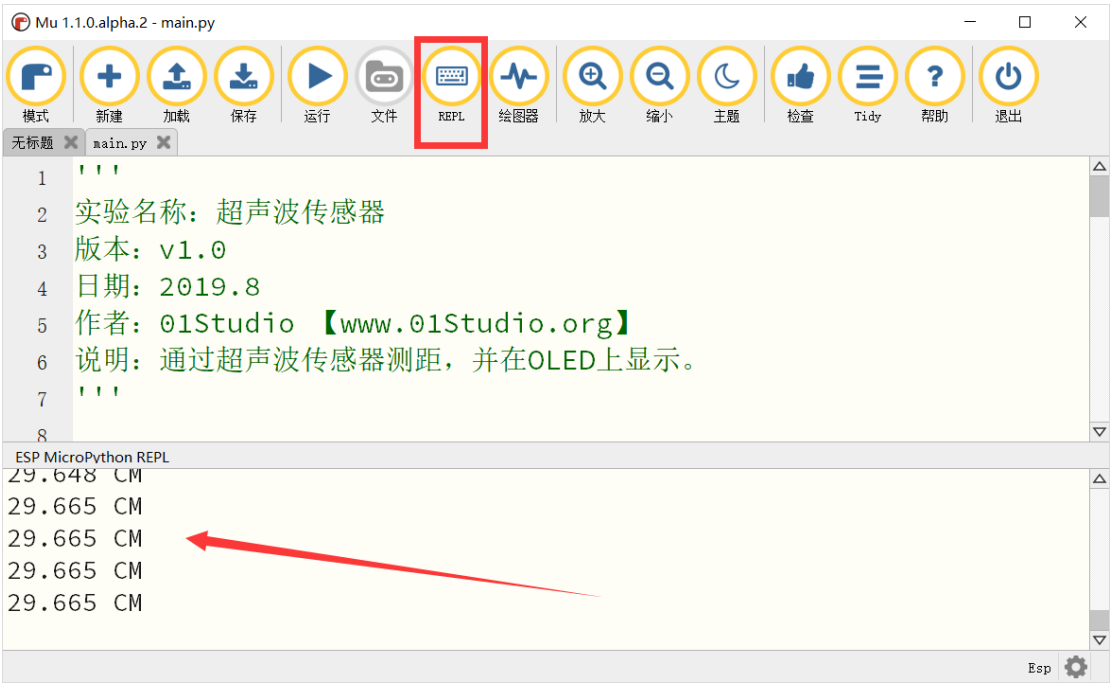


图 13-56　REPL 同步打印实验结果

- **总结：**

出去 OLED 显示代码，我们实际上只用了 2 行代码：初始化和调用测量函数就实现了对超神波传感器测距的应用。这让我们再一次感受到了 MicroPython 的魅力。赶快动作制作自己的避障小车和其他好玩的创作吧。

第14章　拓展实验

在学习了 MicroPython 前面内容后，相信我们已经对其编程方法有了一定的认识，那么除了较为简单的基础实验外，我们还可以使用 MicroPython 开发板连接非常多的外接设备，如常见的舵机、LCD 等。

这些外设的编程难度分两块：如果是仅仅面向应用的话，那么我们可以使用非常简单的方式就用起来；由于代码是完全开源，所以如果想深入学习，则可以看一下驱动代码的实验原理，甚至自行编写驱动程序。从而一起丰富 MicroPython 的周边的生态产品。

14.1 舵机

- **前言：**

舵机又叫伺服电机，是一个可以旋转特定角度的电机，可转动角度通常是 90°、180° 和 360°（360° 可以连续旋转）。我们看到的机器人身上就有非常多的舵机，它们抬手或者摇头的动作往往是通过舵机完成，因此机器人身上的舵机越多，意味着动作越灵活。

- **实验平台：**

pyWiFi-ESP32 开发套件和舵机。

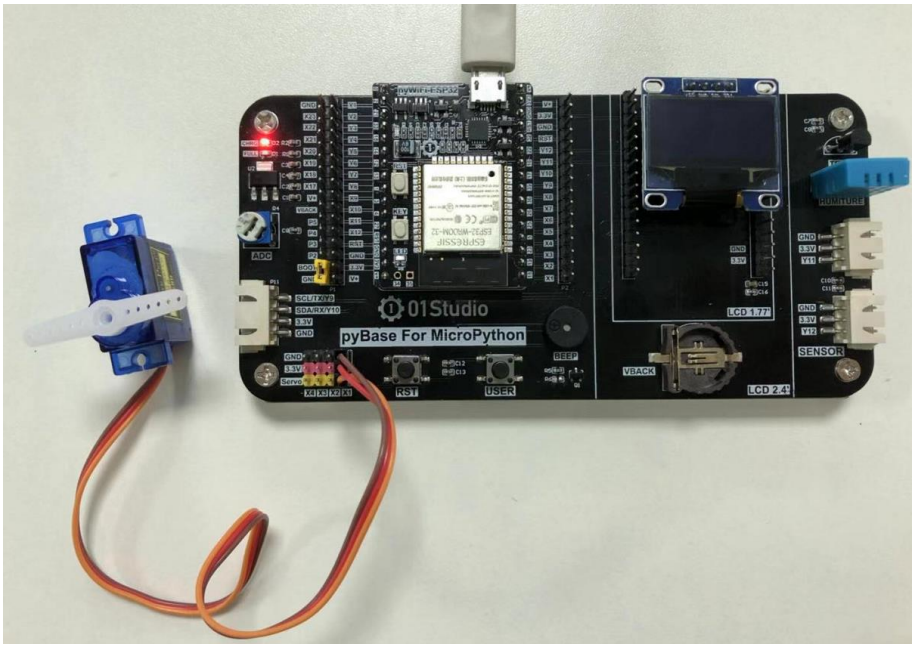


图 14-1 舵机接线

通常情况下：**黑色表示 GND，红色表示 VCC，橙色表示信号线。**

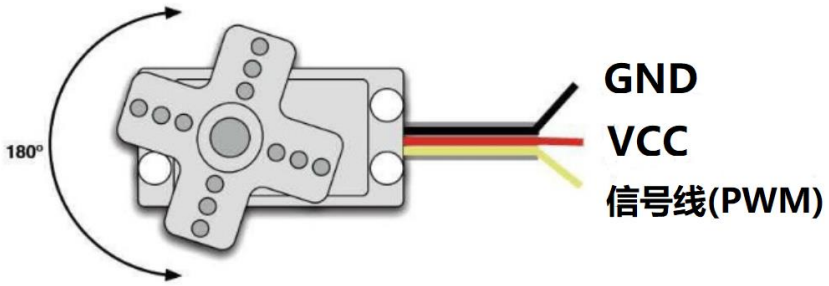


图 14-2 舵机示意图

- **实验目的：**

通过编程实现对舵机的控制。

- **实验讲解：**

伺服电机对象通过 3 线（信号，电源，地）控制，pyBase 上有 4 个位置可以插这些电机，分别是 X1-X4 引脚。对应 pyWiFi-ESP32 的 18、19、23、5 脚。

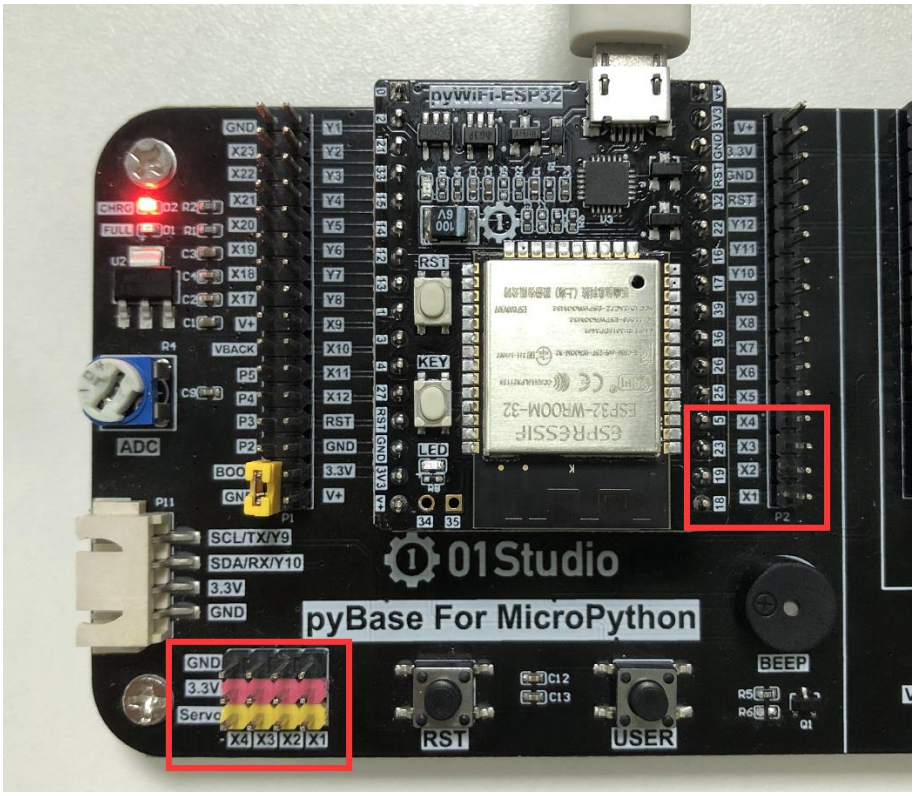


图 14-3　pyBase 的 4 路舵机接口

180°舵机的控制一般需要一个 20ms 左右的时基脉冲，该脉冲的高电平部分一般为 0.5ms-2.5ms 范围内的角度控制脉冲部分，总间隔为 2ms。以 180 度角度伺服为例，在 MicroPython 编程对应的控制关系是从-90°至 90°，示例图如下：

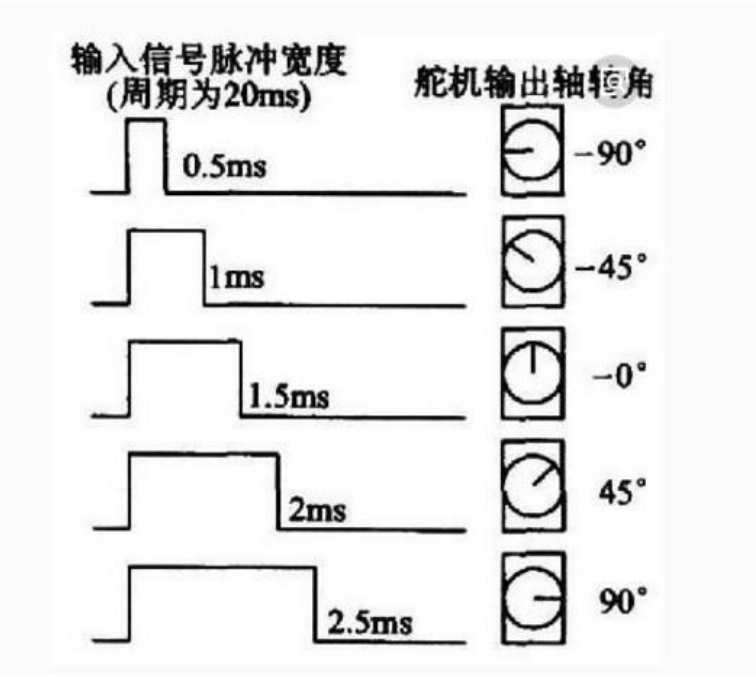


图 14-4　PWM 与角度关系

而对于 360°连续旋转舵机，上面的脉冲表则对应从正向最大速度旋转到反向最大速度旋转的过程。

如果过你学习过前面基于 STM32 平台的舵机实验，那就知道在 STM32 平台集成了舵机模块，使用起来非常方便。当前的 ESP32 平台并没有集成 Servo 模块，但从上面可以看到上面是通过 PWM 来控制的，我们可以直接写 PWM 函数驱动即可。代码编程流程图如下：

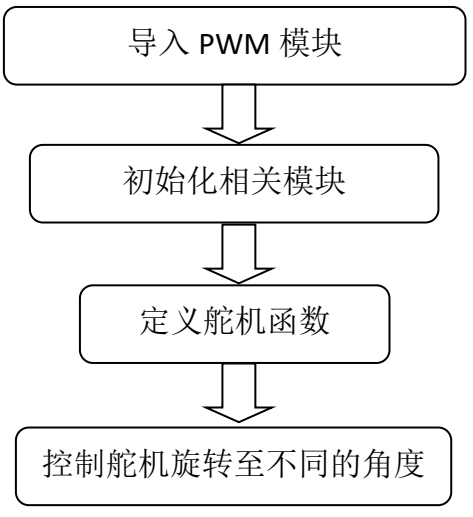


图 14-5 编程流程图

参考代码如下：

```
'''
实验名称：舵机控制
版本：v1.0
日期：2019.8
作者：01Studio 【www.01Studio.org】
说明：通过编程控制舵机旋转到不同角度
'''

from machine import Pin, PWM
import time

S1 = PWM(Pin(18), freq=50, duty=0) # Servo1 的引脚是 18

'''
说明：舵机控制函数
功能：180 度舵机：angle:-90 至 90 表示相应的角度
     360 连续旋转度舵机：angle:-90 至 90 旋转方向和速度值。
'''
def Servo(servo,angle):
    S1.duty(int(((angle+90)*2/180+0.5)/20*1023))

#-90 度
Servo(S1,-90)
time.sleep(1)

#-90 度
Servo(S1,-45)
```

```
time.sleep(1)


#-90 度
Servo(S1,0)
time.sleep(1)


#-90 度
Servo(S1,45)
time.sleep(1)


#-90 度
Servo(S1,90)
time.sleep(1)
```

- **实验结果：**

将 180° 舵机插到 pyBase 的 X1 的三线接口，下载程序并按下复位。可以看到舵机依次旋转至不同角度。

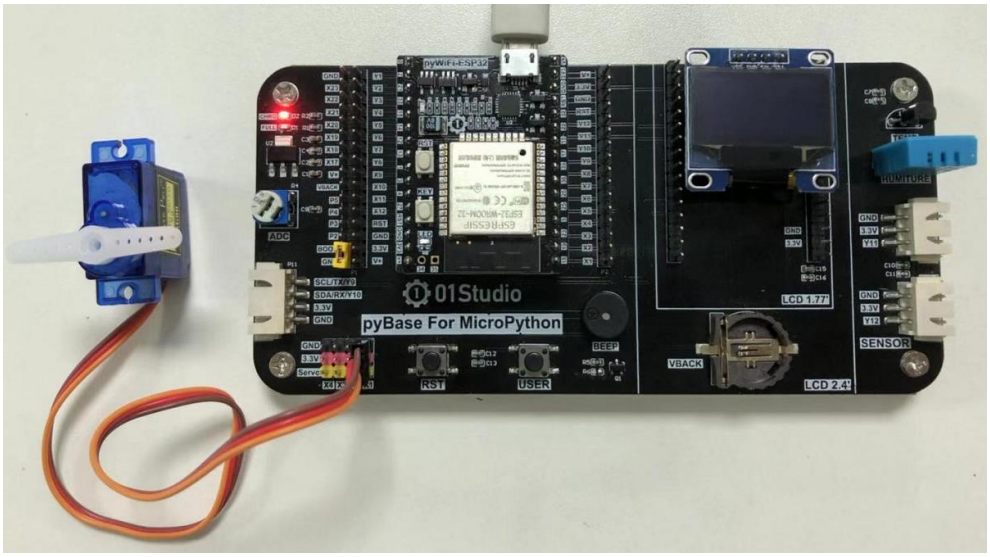


图 14-6 舵机角度旋转至不同角度

- **实验拓展：**

我们刚刚实现了 180° 舵机的角度控制，现在来做一下 360° 连续旋转舵机的实验，360° 连续旋转舵机可以实现直流减速电机功能，用在小车或者航模上。

实验的代码不变，参数【-90 至 90】代表旋转方向和速度值大小。插上 360° 连续旋转舵机。可以看到舵机的旋转速度和方向逐渐变变化。

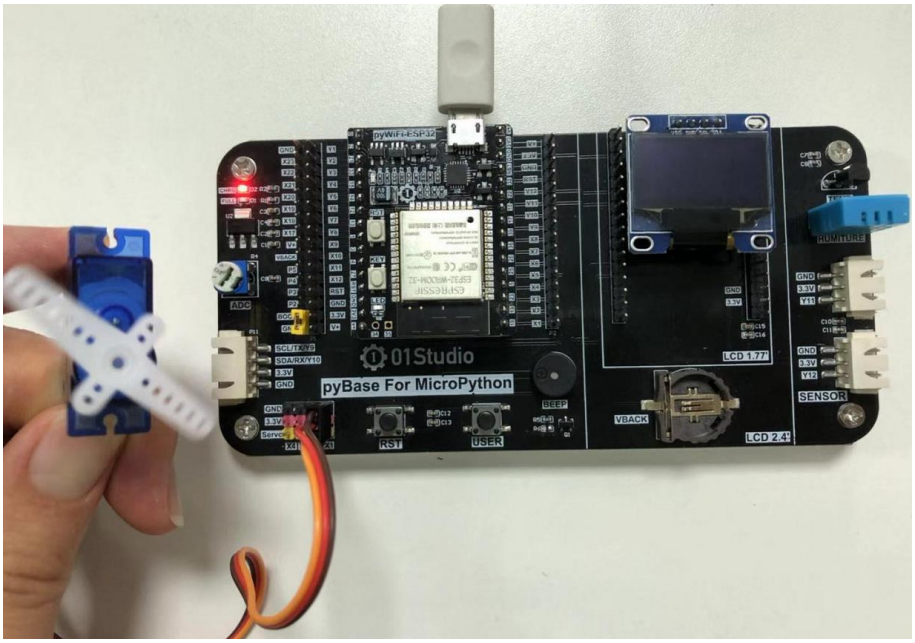


图 14-7　360°连续旋转舵机

- **总结：**

通过本节我们学会了使用不同类型的舵机，通过多路电机的组合使用，可以实现模型、航模、小车、机器人的实验。

14.2 RGB 灯带

- **前言：**

RGB 灯带是彩灯的一种，我们看到长长的灯带以及 LED 彩色显示屏，都是采用一个个灯珠组合而成。在本章节我们将通过编程实现 RGB 灯带的控制，可以应用到家居布置、节日气氛的场景中去。

- **实验平台：**

pyWiFi-ESP32 开发套件和 RGB 灯带。连接 pyBase 到“Y11”接口。

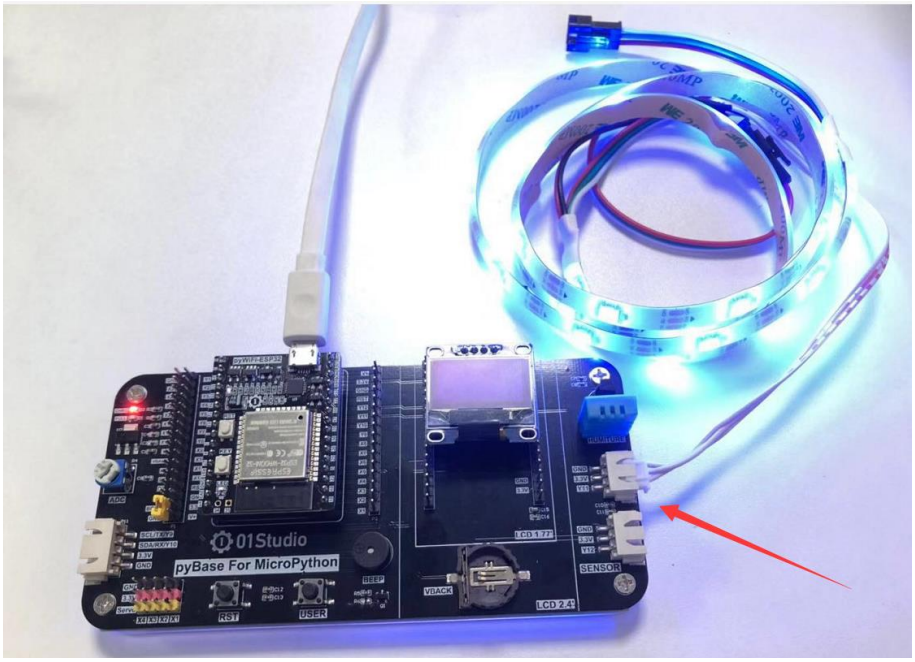


图 14-8 RGB 灯带接线方式

- **实验目的：**

通过编程实现灯带循环显示红色（RED）、绿色（GREEN）和蓝色（BLUE）。

- **实验讲解：**

先来介绍一下本实验用到的 RGB 灯带。

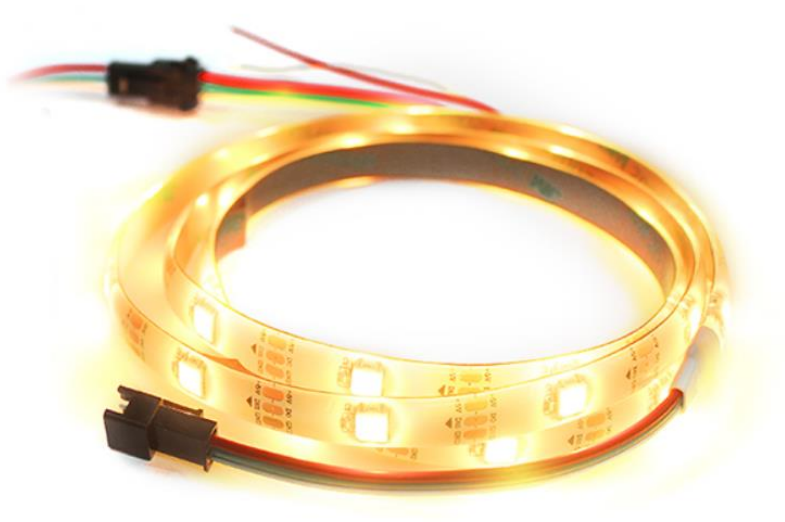


图 14-9 RGB 灯带

该灯带为 3 线接口，长 1 米，有 30 颗灯珠，每个灯珠里面都有一个驱动 IC，型号是 WS2812B，单总线驱动。每个灯珠首尾相连。灯带末端预留了接口可以串联更多的灯带。也就是说只需要通过 1 个 GPIO 口就可以控制了数十上百的灯珠了。

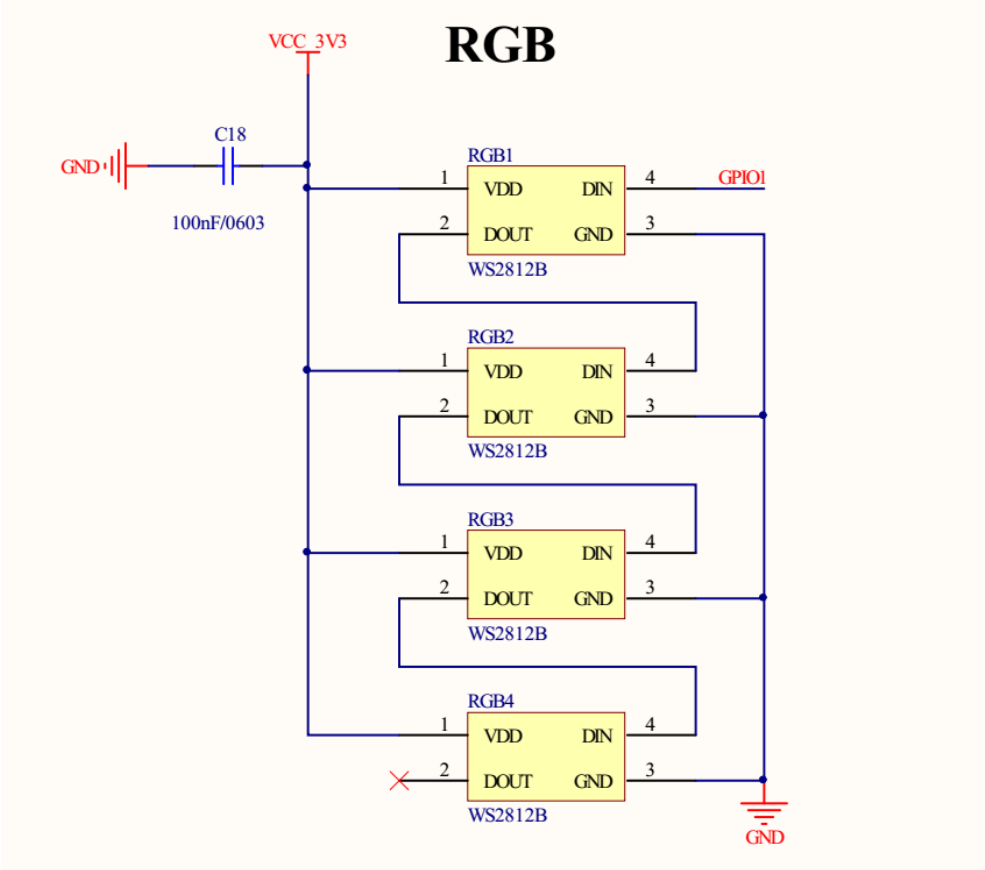


图 14-10 灯珠串联接线方式

了解了接线方式后，我们知道颜色是由最基本的三种颜色的不同亮度混合出新颜色。这 3 个最基本的颜色顺序分别是红，绿，蓝（RGB）。这里每个颜色的亮度级别从 0-255,0 表示没有，255 表示最亮。如（255,0,0）则表示红色最亮。GPIO 口就是将这些数据逐一发送给灯带。

ESP32 的 MicroPython 固件集成了彩灯驱动模块 NeoPixel，适用于 WS2812B 驱动的灯珠。因此我们可以直接使用。说明如下：

构造函数
np=neopixel.NeoPixel(pin, led_count)
构建灯带对象。pin:灯带控制引脚，led_count:灯珠数量；
使用方法
np[0]=(R,G,B)
设置第 1 个灯珠参数。
np.write()
往灯带写入设置的数据。

表 14-1 灯带对象

编程流程图如下：

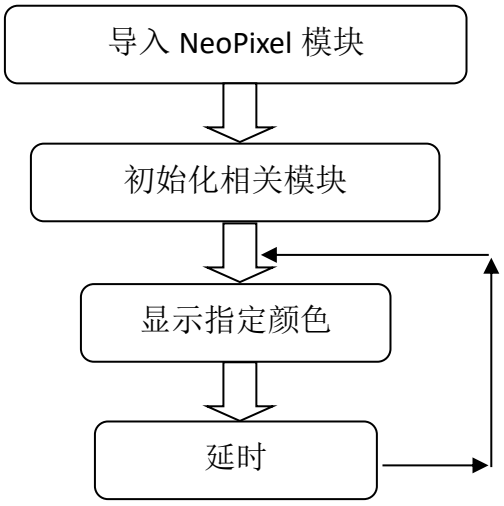


图 14-11 编程流程图

main.py 参考程序代码如下：

```
'''
实验名称：RGB 彩灯控制
版本：v1.0
日期：2019.8
作者：01Studio 【www.01Studio.org】
说明：通过编程实现灯带不同颜色的变化。
'''
import time
from machine import Pin,Timer
from neopixel import NeoPixel

#定义红、绿、蓝三种颜色
RED=(255,0,0)
GREEN=(0,255,0)
BLUE=(0,0,255)

#22 引脚连接灯带，灯珠数量 30
pin = Pin(22, Pin.OUT)
np = NeoPixel(pin, 30)

#设置灯珠颜色，本实验供 30 个灯珠
def Color_buf(color):
    for i in range(30):
        np[i]=color

while True:

    Color_buf(RED) #红色
```

```
    np.write()     # 写入数据
    time.sleep(1)

    Color_buf(GREEN) #红色
    np.write()     # 写入数据
    time.sleep(1)

    Color_buf(BLUE) #红色
    np.write()     # 写入数据
    time.sleep(1)
```

- **实验结果：**

将程序下载到开发套件，可以看到灯带的颜色在红、绿、蓝三色中循环变化。

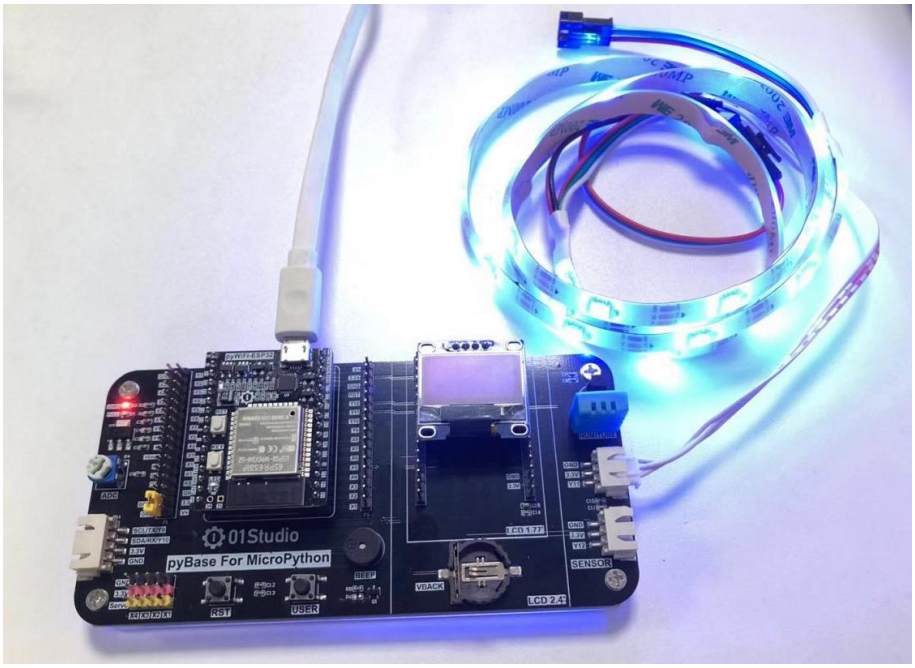


图 14-12 实验结果

- **总结：**

RGB 灯带的单总线特点使得我们可以轻易的增加和减少灯带的数量而无需过多的修改程序。但需要注意的是如果灯带数量过多，那么需要外接电源供电，否则会因为电流不足而影响使用。

第15章　WiFi 应用

通过前面的实验，我们已经对 ESP32 有了一定的了解。从本章开始，将迎来非常重要实用的内容，那就是 WIFI 应用。ESP32 就是为 WIFI 无线连接而生的。通过本章内容，我们可以看到基于 MicroPython 的 WIFI 开发是多么的简单而美妙。物联网的学习变得非常简单有趣！事不宜迟，马上开始学习。

15.1 连接无线路由器

- **前言：**

WIFI 是物联网中非常重要的角色，现在基本上家家户户都有 WIFI 网络了，通过 WIFI 接入到互联网，成了智能家居产品普遍的选择。而要想上网，首先需要连接上无线路由器。这一节我们就来学习如何通过 MicroPython 编程连上路由器。

- **实验平台：**

pyWiFi-ESP32 和 pyBase 开发底板。

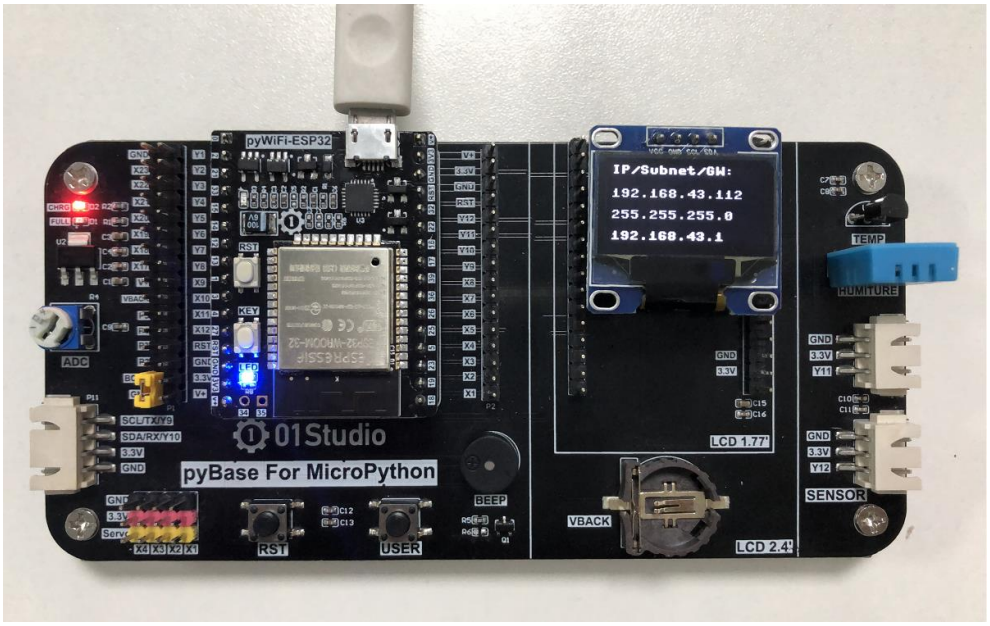


图 15-1 pyWiFi-ESP32 开发套件

- **实验目的：**

编程实现连接路由器，将 IP 地址等相关信息通过 OLED 显示（只支持 2.4G 网络）。

- **实验讲解：**

连接路由器上网是我们每天都做的事情，日常生活中我们只需要知道路由器的账号和密码，就能使用电脑或者手机连接到无线路由器，然后上网冲浪。

MicroPython 已经集成了 network 模块，开发者使用内置的 network 模块函数可以非常方便地连接上路由器。但往往也有各种连接失败的情况，如密码不正

确等。这时候我们只需要再加上一些简单的判断机制，避免陷入连接失败的死循环即可！

我们先来看看 network 基于 WiFi（WLAN 模块）的构造函数和使用方法。

构造函数
wlan = network.WLAN(interface_id)
构建 WIFI 连接对象。interface_id:分为热点 network.AP_IF 和客户端 network.STA_IF 模式。
使用方法
wlan.active([is_active])
激活 wlan 接口。Ture：激活；False:关闭。
wlan.scan ()
扫描允许访问的 SSID。
wlan.isconnected()
检查设备是否已经连接上。返回 Ture:已连接；False：未连接。
wlan.connected(ssid,passwork)
WIFI 连接。ssid:账号；passwork：密码。
wlan.ifconfig([ip,subnet,gateway,dns])
设备信息配置。ip：IP 地址；subnet:子网掩码；gateway:网关地址；dns:DNS 信息。**（如果参数为空，则返回当前连接信息。）**
wlan.disconnected()
断开连接。

表 15-1　WLAN 对象

从上表可以看到 MicroPython 通过模块封装，让 WIFI 联网变得非常简单。模块包含热点 AP 模块和客户端 STA 模式，热点 AP 是指电脑端直接连接 ESP32 发出的热点实现连接，但这样你的电脑就不能上网了，因此我们一般情况下都是使用 STA 模式。也就是电脑和设备同时连接到相同网段的路由器上。

模块上电后可以先判断是否已经连接到网络，如果是则无需再次连接，否的

话则进入 WIFI 连接状态，指示灯闪烁，连接成功后指示灯常亮，IP 等相关信息通过 OLED 显示和串口打印。另外需要配置超时 15 秒还没连接成功时执行取消连接，避免因无法连接而陷入死循环。代码编写流程如下：

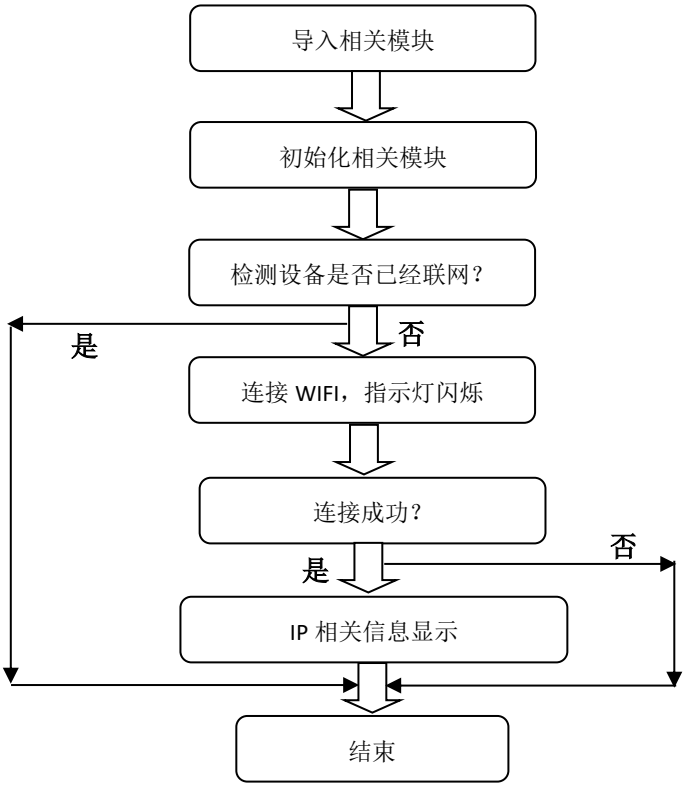


图 15-2 代码编写流程图

实验参考代码如下：

```
'''
实验名称：连接无线路由器
版本：v1.0
日期：2019.8
作者：01Studio
说明：编程实现连接路由器，将 IP 地址等相关信息通过 OLED 显示（只支持 2.4G 网络）。
'''
import network,time
from machine import I2C,Pin
```

```
from ssd1306 import SSD1306_I2C

#初始化相关模块
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#WIFI 连接函数
def WIFI_Connect():

    WIFI_LED=Pin(2, Pin.OUT) #初始化 WIFI 指示灯

    wlan = network.WLAN(network.STA_IF) #STA 模式
    wlan.active(True)                   #激活接口
    start_time=time.time()              #记录时间做超时判断

    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码

        while not wlan.isconnected():

            #LED 闪烁提示
            WIFI_LED.value(1)
            time.sleep_ms(300)
            WIFI_LED.value(0)
            time.sleep_ms(300)

            #超时判断,15 秒没连接成功判定为超时
            if time.time()-start_time > 15 :
                print('WIFI Connected Timeout!')
```

```
            break

    if wlan.isconnected():
        #LED 点亮
        WIFI_LED.value(1)

        #串口打印信息
        print('network information:', wlan.ifconfig())

        #OLED 数据显示
        oled.fill(0)   #清屏背景黑色
        oled.text('IP/Subnet/GW:',0,0)
        oled.text(wlan.ifconfig()[0], 0, 20)
        oled.text(wlan.ifconfig()[1],0,38)
        oled.text(wlan.ifconfig()[2],0,56)
        oled.show()

#执行 WIFI 连接函数
WIFI_Connect()
```

上面代码在 main.py 文件中，将 WIFI 连接封装成了子函数形式，我们也可以新建一个 WIFI.py 文件，然后通过模块引用方式来供 main.py 调用，以提高程序的灵活性。

- **实验结果：**

下载程序，可以观察到上电后 pyWiFi-ESP32 上的 LED 快速闪烁，连接成功后 LED 常亮，OLED 显示当前 IP、子网掩发、网关的地址信息。

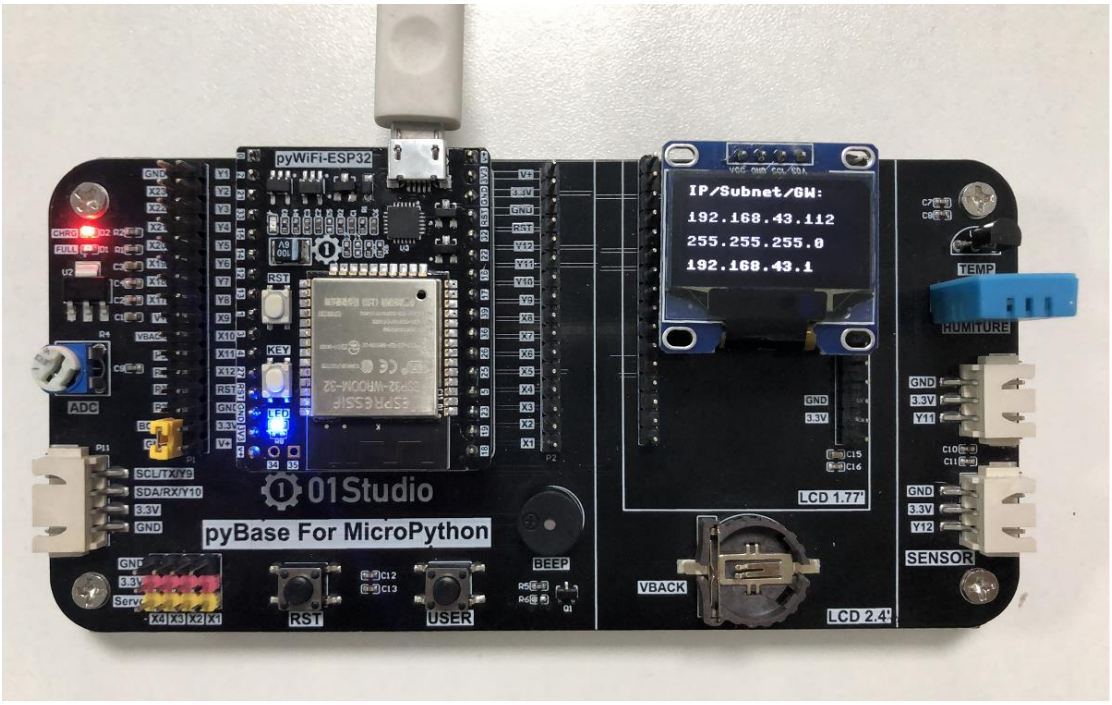


图 15-3

- **总结：**

本节是 WIFI 应用的基础，成功连接到无线路由器的实验后，后面就可以做 socket 等相关网络通信的应用了。

15.2 Socket 通信

- **前言：**

上一节我们学习了如何通过 MicroPython 编程实现 pyWiFi-ESP32 模块连接到无线路由器。这一节我们则来学习一下 Socket 通信实验。Socket 几乎是整个互联网通信的基础。

- **实验平台：**

pyWiFi-ESP32 和 pyBase 开发底板。

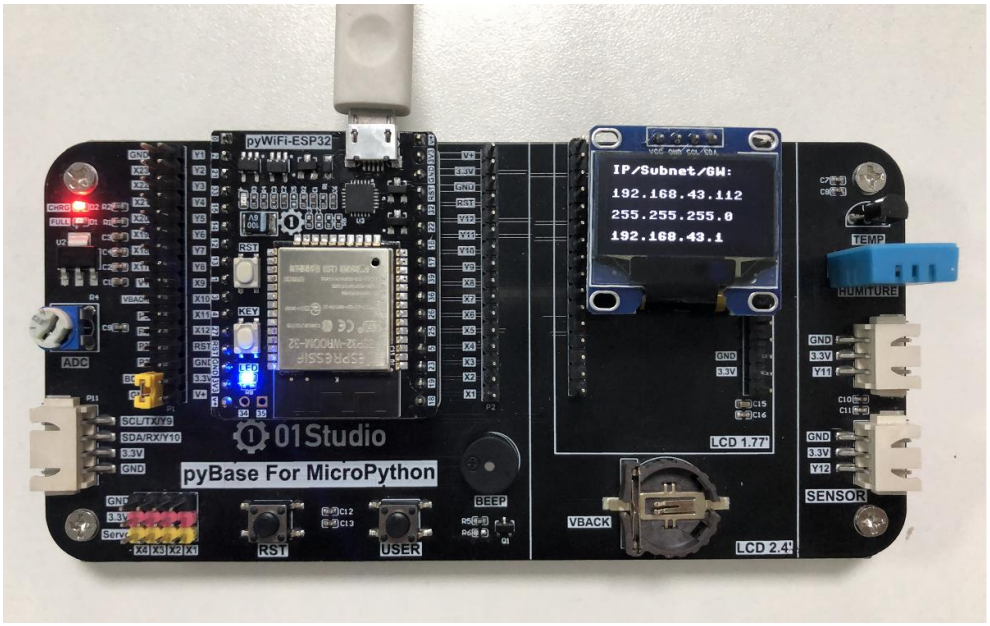


图 15-4 pyWiFi-ESP32 开发套件

- **实验目的：**

通过 Socket 编程实现 pyWiFi-ESP32 与电脑服务器助手建立连接，相互收发数据。

- **实验讲解：**

Socket 我们听得非常多了，但由于网络工程是一门系统工程，涉及的知识非常广，概念也很多，任何一个知识点都能找出一堆厚厚的的书，因此我们经常会混淆。在这里，我们尝试以最容易理解的方式来讲述 Socket，如果需要全面了解，可以自行查阅相关资料学习。

我们先来看看网络层级模型图，这是构成网络通信的基础：

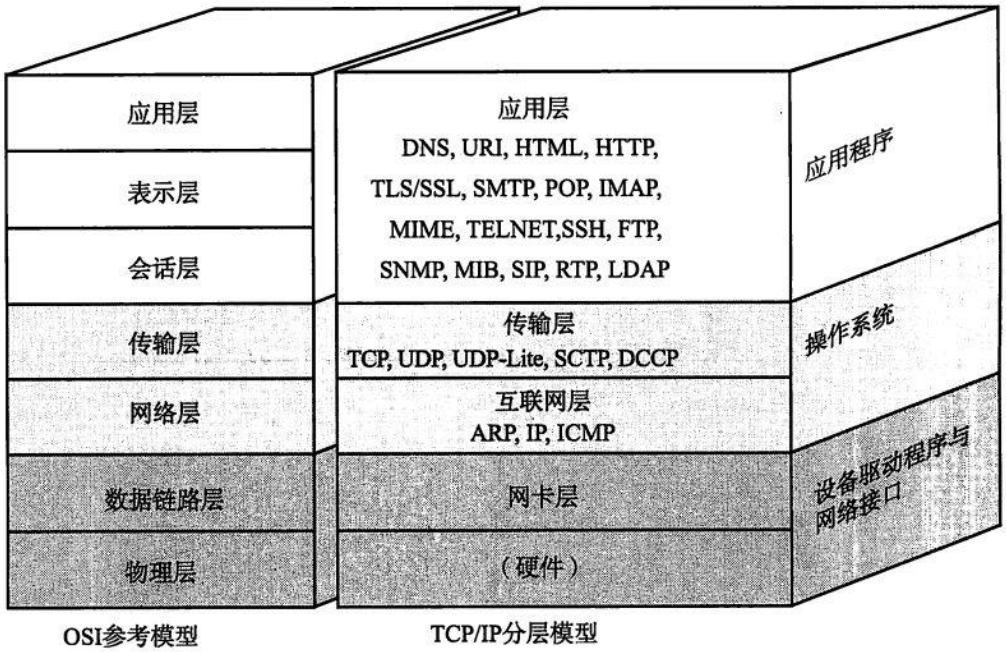


图 15-5 网络层级模型

我们看看 TCP/IP 模型的传输层和应用层，传输层比较熟悉的概念是 TCP 和 UDP，UPD 协议基本就没有对 IP 层的数据进行任何的处理了。而 TCP 协议还加入了更加复杂的传输控制，比如滑动的数据发送窗口（Slice Window），以及接收确认和重发机制，以达到数据的可靠传送。应用层中网页常用的则是 HTTP。那么我们先来解析一下这 TCP 和 HTTP 两者的关系。

我们知道网络通信是最基础是依赖于 IP 和端口的，HTTP 一般情况下默认使用端口 80。举个简单的例子：我们逛淘宝，浏览器会向淘宝网的网址（本质是 IP）和端口发起请求，而淘宝网收到请求后响应，向我们手机返回相关网页数据信息，实现了网页交互的过程。而这里就会引出一个多人连接的问题，很多人访问淘宝网，实际上接收到网页信息后就断开连接，否则淘宝网的服务器是无法支撑这么多人长时间的连接的，哪怕能支持，也非常占资源。

也就是应用层的 HTTP 通过传输层进行数据通信时，TCP 会遇到同时为多个应用程序进程提供并发服务的问题。多个 TCP 连接或多个应用程序进程可能需要通过同一个 TCP 协议端口传输数据。为了区别不同的应用程序进程和连接，许多计算机操作系统为应用程序与 TCP / IP 协议交互提供了套接字(Socket)接口。应用层可以和传输层通过 Socket 接口，区分来自不同应用程序进程或网络连接的通信，实

现数据传输的并发服务。

简单来说，Socket 抽象层介于传输层和应用层之间，跟 TCP/IP 并没有必然的联系。Socket 编程接口在设计的时候，就希望也能适应其他的网络协议。

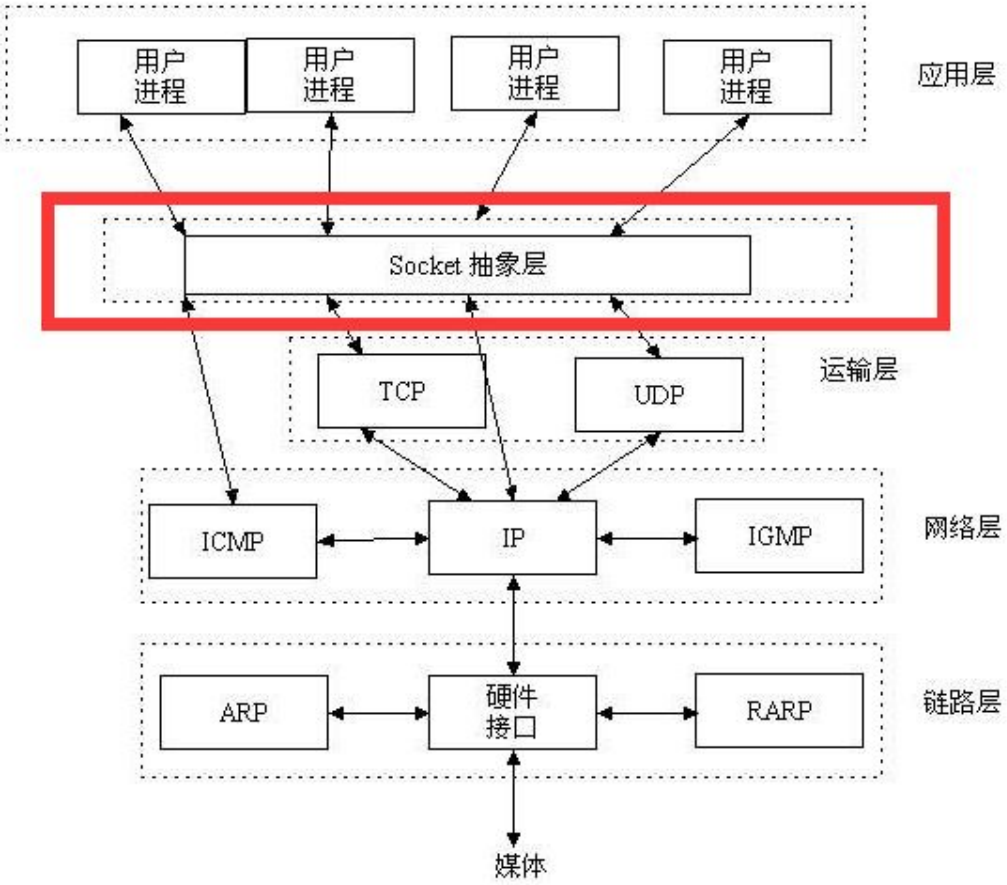


图 15-6　Socket 抽象层

套接字（socket）是通信的基石，是支持 TCP/IP 协议的网络通信的基本操作单元。它是网络通信过程中端点的抽象表示，包含进行网络通信必须的五种信息：**连接使用的协议（通常是 TCP 或 UDP），本地主机的 IP 地址，本地进程的协议端口，远地主机的 IP 地址，远地进程的协议端口。**

所以，socket 的出现只是可以更方便的使用 TCP/IP 协议栈而已，简单理解就是其对 TCP/IP 进行了抽象，形成了几个最基本的函数接口。比如 create，listen，accept，connect，read 和 write 等等。以下是通讯流程：

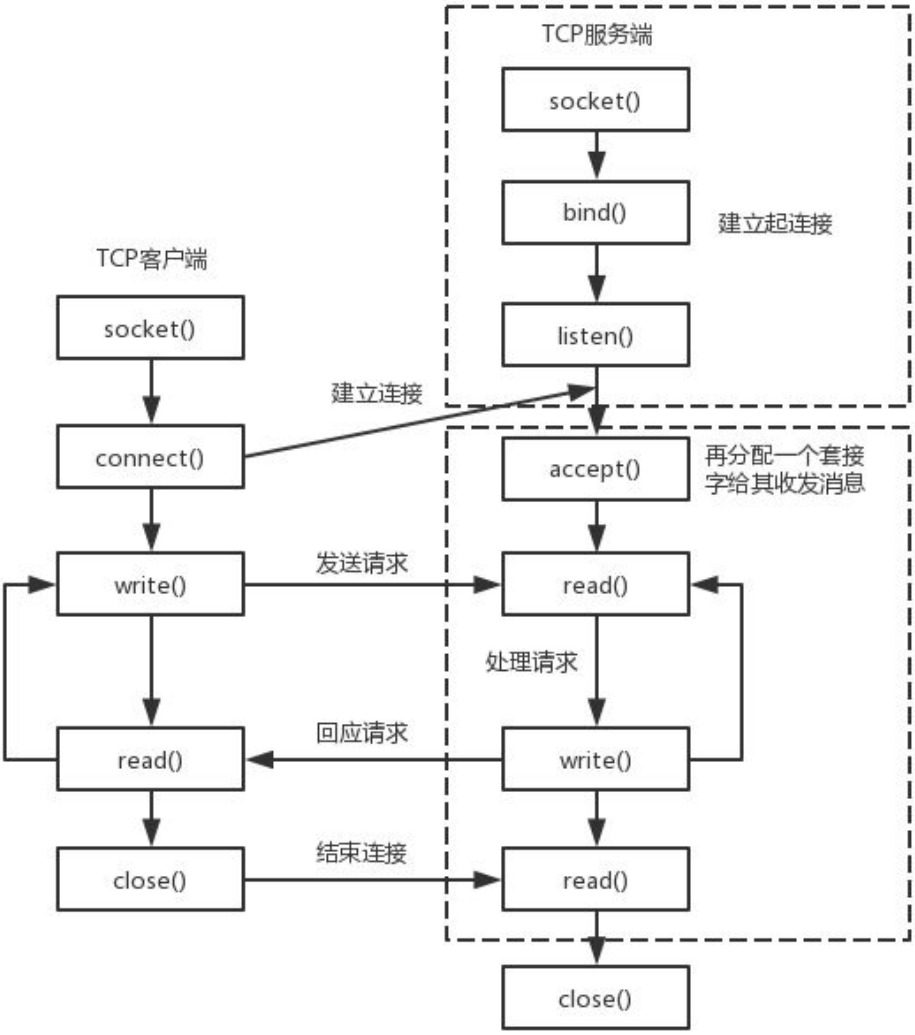


图 15-7　Socket 通信过程

从上图可以看到，建了 Socket 通信需要一个服务器端和一个客户端，以本实验为例，pyWiFi-ESP32 作为客户端，电脑使用网络调试助手作为服务器端，双方使用 TCP 协议传输。对于客户端，则需要知道电脑端的 IP 和端口即可建立连接。（端口可以自定义，范围在 0~65535，注意不占用常用的 80 等端口即可。）

以上的内容，简单来说就是如果用户面向应用来说，那么 ESP32 只需要知道**通讯协议是 TCP 或 UDP、服务器的 IP 和端口号**这 3 个信息，即可向服务器发起连接和发送信息。就这么简单。

MicroPython 已经封装好相关模块 usocket,跟传统的 socket 大部分兼容，两者均可使用，本实验使用 usocket，对象如下介绍：

构造函数
s=usocket.socekt(af=AF_INET, type=SOCK_STREAM,proto=IPPROTO_TCP)
构建 usocket 对象。 af: AF_INET→IPV4，AF_INET6 → IPV6； type: SCOK_STREAM→TCP，SOCK_DGRAM→UDP； proto: IPPROTO_TCP→TCP 协议，IPPROTO_UDP→UDP 协议。 （如果要构建 TCP 连接，可以使用默认参数配置，即不输入任何参数。）
使用方法
addr=usocket.getaddrinfo('www.01studio.org', 80)[0][-1]
获取 Socket 通信格式地址。返回：('47.91.208.161',80)
s.connect(address)
创建连接。address:地址格式为 IP+端口。例：('192.168.1.115',10000)
s.send(bytes)
发送。bytes：发送内容格式为字节
s.recv(bufsize)
接收数据。bufsize：单次最大接收字节个数。
s.bind(address)
绑定，用于服务器角色
s.listen([backlog])
监听，用于服务器角色。backlog:允许连接个数，必须大于 0。
s.accept()
接受连接，用于服务器角色。
*其它更多用法请阅读 MicroPython 文档：(搜索:usocket) 中文文档链接：http://docs.micropython.01studio.org/

表 15-2　Socket 对象

本实验中 pyWiFi-ESP32 属于客户端，因此只用到客户端的函数即可。实验代码编写流程如下：

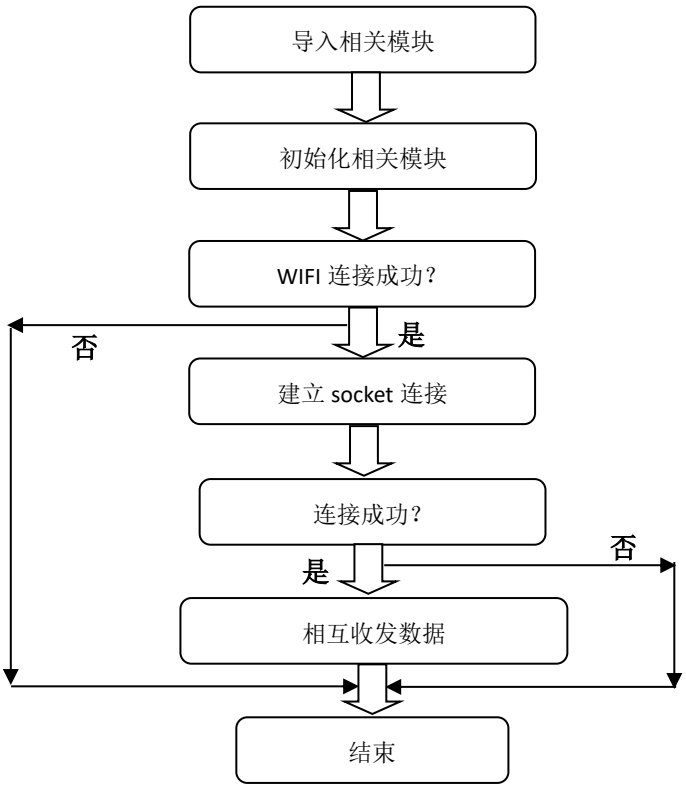


图 15-8 代码编写流程图

实验参考代码：

```
'''
实验名称：连接无线路由器
版本：v1.0
日期：2019.8
作者：01Studio
说明：通过 Socket 编程实现 pyWiFi-ESP32 与电脑服务器助手建立 TCP 连接，相互收发数据。
'''
#导入相关模块
import network,usocket,time
from machine import I2C,Pin,Timer
from ssd1306 import SSD1306_I2C
```

```
#初始化相关模块
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

# WIFI 连接函数
def WIFI_Connect():

    WIFI_LED=Pin(2, Pin.OUT) #初始化 WIFI 指示灯

    wlan = network.WLAN(network.STA_IF) #STA 模式
    wlan.active(True)                   #激活接口
    start_time=time.time()              #记录时间做超时判断

    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码
        while not wlan.isconnected():
            #LED 闪烁提示
            WIFI_LED.value(1)
            time.sleep_ms(300)
            WIFI_LED.value(0)
            time.sleep_ms(300)

            #超时判断,15 秒没连接成功判定为超时
            if time.time()-start_time > 15 :
                print('WIFI Connected Timeout!')
                break

    if wlan.isconnected():
        #LED 点亮
```

```
        WIFI_LED.value(1)

        #串口打印信息
        print('network information:', wlan.ifconfig())

        #OLED 数据显示
        oled.fill(0)   #清屏背景黑色
        oled.text('IP/Subnet/GW:',0,0)
        oled.text(wlan.ifconfig()[0], 0, 20)
        oled.text(wlan.ifconfig()[1],0,38)
        oled.text(wlan.ifconfig()[2],0,56)
        oled.show()
        return True

    else:
        return False

def Socket_fun(tim):

    text=s.recv(128) #单次最多接收 128 字节
    if text == '':
        pass

    else: #打印接收到的信息为字节，可以通过 decode('utf-8')转成字符串
        print(text)
        s.send('I got:'+text.decode('utf-8'))

#判断 WIFI 是否连接成功
if WIFI_Connect():
```

```
#创建 socket 连接 TCP 类似，连接成功后发送“Hello 01Studio!”给服务器。
s=usocket.socket()
addr=('192.168.1.115',10000) #服务器 IP 和端口
s.connect(addr)
s.send('Hello 01Studio!')

#开启 RTOS 定时器，编号为-1,周期 300ms，执行 socket 通信接收任务
tim = Timer(-1)
tim.init(period=300, mode=Timer.PERIODIC,callback=Socket_fun)
```

WIFI 连接代码在上一节已经讲解，这里不再重复，WIFI 连接成功后返回 True，否则返回 False。程序在返回连接成功后建了 Socket 连接，连接成功发送‘Hello 01Studio!’信息到服务器。另外 RTOS 定时器设定了了每 300ms 处理从服务器接收到的数据。将接收到数据通过串口打印和发送给服务器。

- **实验结果：**

先在电脑端打开网络调试助手并建立服务器，软件在 零一科技（01Studio）MicroPython 开发套件配套资料_latest\01-开发工具\01-Windows\网络调试助手下的 NetAssist.exe ， 直接双击打开即可！

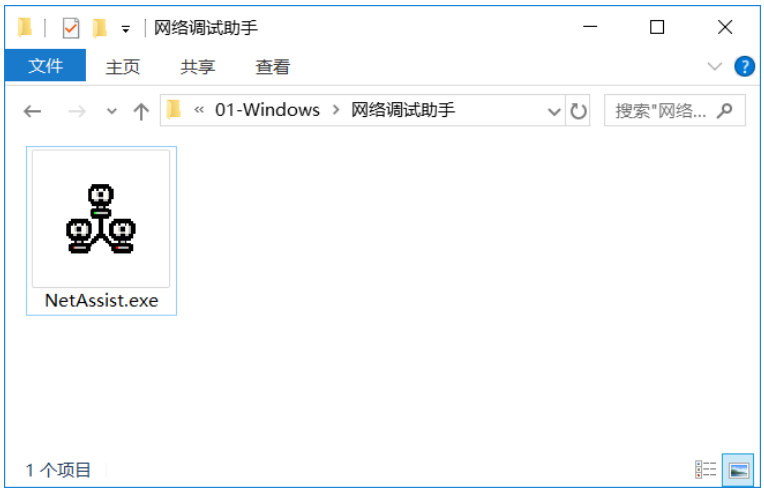


图 15-9 网络调试助手

以下是新建服务器的方法，打开网络调试助手后在左上角协议类型选择 TCP Server；中间的本地 IP 地址是自动识别的，不要修改，这个就是服务器的 IP 地址。然后端口写 10000（0-65535 都可以。），点击连接，成功后红点亮。如下图：



图 15-10 TCP 服务器配置

在时候服务器已经在监听状态！用户需要根据自己的实际情况自己输入 WIFI 信息和服务器 IP 地址+端口。即修改上面的代码以下部分内容。（服务器 IP 和端口可以在网络调试助手找到。）

```
wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码

addr=('192.168.1.115',10000) #服务器 IP 和端口
```

下载程序，开发板成功连接 WIFI 后，发起了 socket 连接，连接成功可以可以看到网络调试助手收到了开发板发来的信息。在下方列表多了一个连接对象，点击选中：

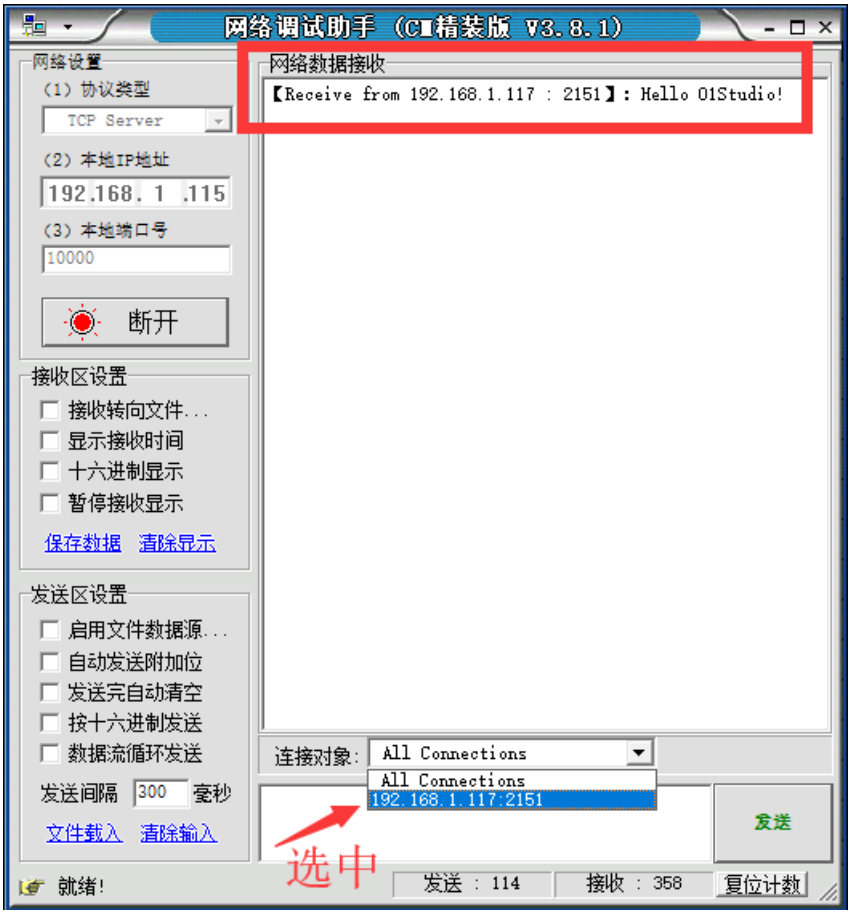


图 15-11 选中连接对象

选中后我们在发送框输入信息“Hi”，点击发送，可以看到开发板的 REPL 打印出来信息 Hi。为字节数据。另外由于程序将收到的信息发回给服务器，所以在网络调试助手中也接收到开发板返回的信息：I got:Hi。

```
MicroPython v1.11-8-g48dcbbe60 on 2019-05-29; ESP module with ESP8266
Type "help()" for more information.
>>> Warning: Comparison between bytes and str
b'Hi'
```

图 15-12

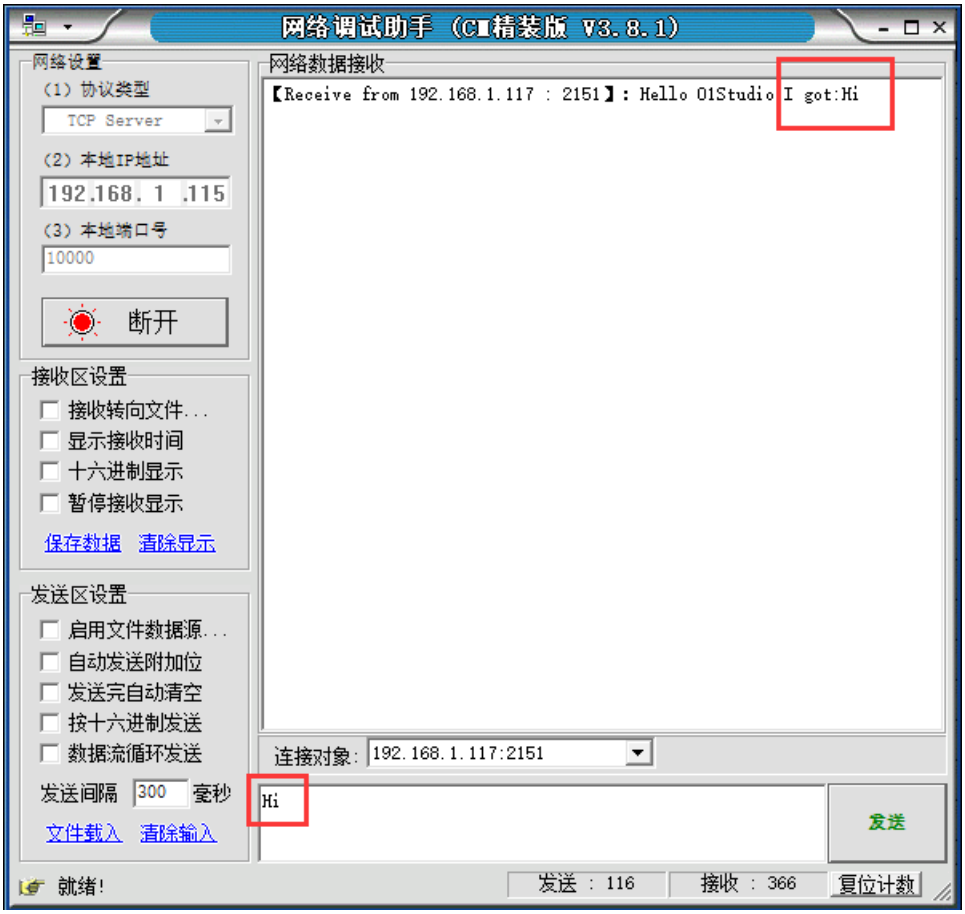


图 15-13 服务器发送数据

- **总结：**

通过本节学习，我们了解了 socket 通信原理以及使用 MicroPython 进行 socket 编程并且通信的实验。得益于优秀的封装，让我们可以直接面向 socket 对象编程就可以快速实现 socket 通信，从而开发更多的网络应用，例如将前面采集到的传感器数据发送到服务器。

15.3 MQTT 通信

- **前言：**

上一节，我们学习了 Socket 通信，当服务器和客户端建立起连接时，就可以相互通信了。在互联网应用大多使用 WebSocket 接口来传输数据。而在物联网应用中，常常出现这样的情况：海量的传感器，需要时刻保持在线，传输数据量非常低，有着大量用户使用。如果仍然使用 socket 作为通信，那么服务器的压力和通讯框架的设计随着数量的上升将变得异常复杂！

那么有无一个框架协议来解决这个问题呢，答案是有的。那就是 MQTT(消息队列遥测传输)。

- **实验平台：**

pyWiFi-ESP32 和 pyBase 开发底板。

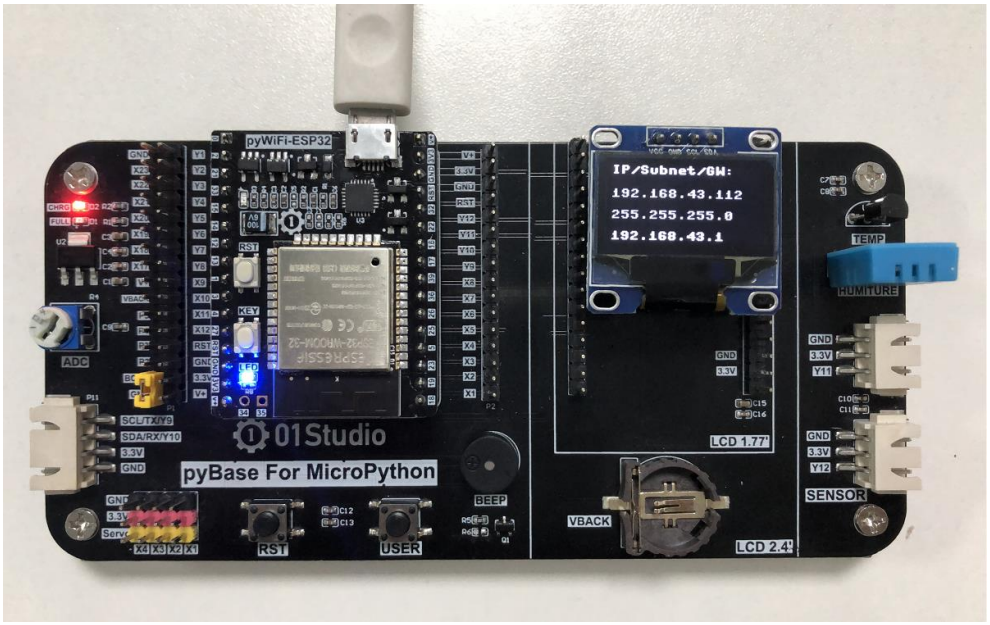


图 15-14 pyWiFi-ESP32 开发套件

- **实验目的：**

通过编程实现 pyWiFi-ESP32 实现 MQTT 协议信息的发布和订阅（接收）。

- **实验讲解：**

MQTT 是 IBM 于 1999 年提出的，和 HTTP 一样属于应用层，它工作在 TCP/IP 协议族上，通常还会调用 socket 接口。是一个基于客户端-服务器的消息发布/订

阅传输协议。其特点是协议是轻量、简单、开放和易于实现的，这些特点使它适用范围非常广泛。在很多情况下，包括受限的环境中，如：机器与机器（M2M）通信和物联网（IoT）。其在，通过卫星链路通信传感器、偶尔拨号的医疗设备、智能家居、及一些小型化设备中已广泛使用。

总结下来 MQTT 有如下特性/优势：

- **异步消息协议**
- **面向长连接**
- **双向数据传输**
- **协议轻量级**
- **被动数据获取**

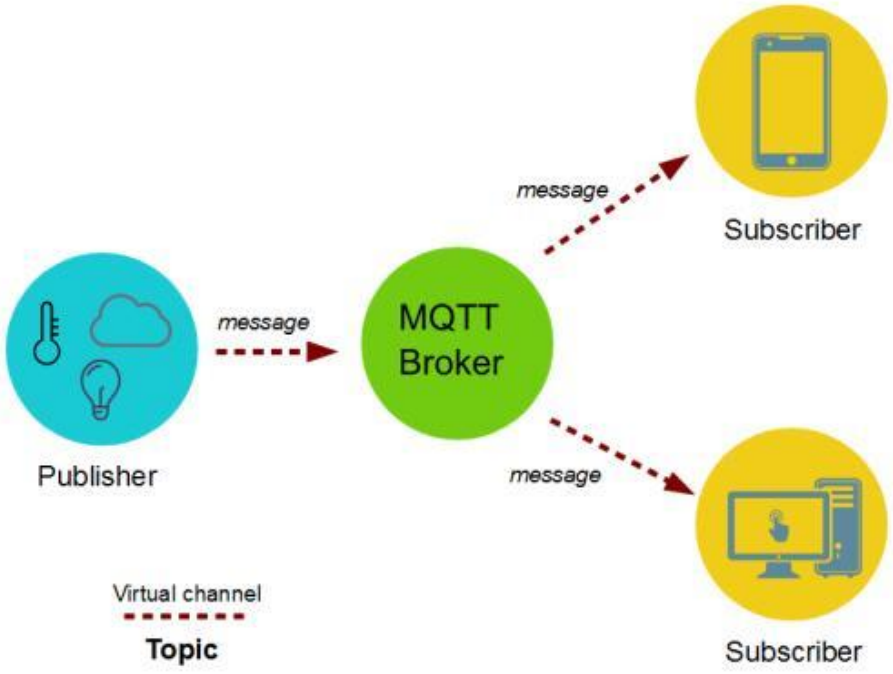


图 15-15 MQTT 通信流程

从上图可以看到，MQTT 通信的角色有两个，分别是服务器和客户端。服务器只负责中转数据，不做存储；客户端可以是信息发送者或订阅者，也可以同时是两者。具体如下图：

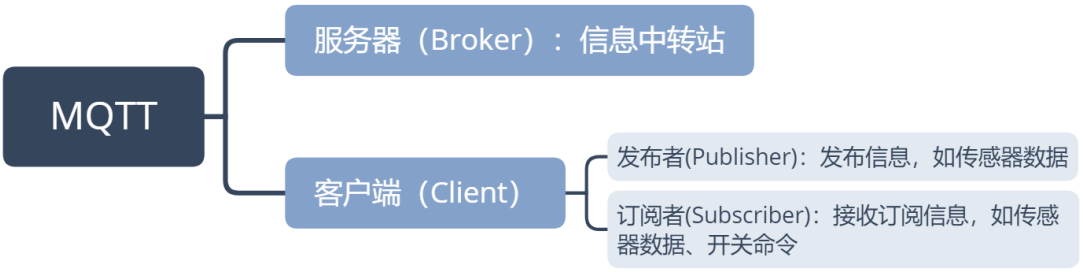


图 15-16　MQTT 角色说明

确定了角色后是如何传输数据呢？下表示 MQTT 最基本的数据帧格式，例如温度传感器发布主题“Temperature”编号,消息是“25”（表示温度）。那么所有订阅了这个主题编号的客户端（手机应用）就会收到相关信息，从而实现通信。如下表所示：

MQTT 数据帧格式	
Topic ID (主题编号)	Message (消息)
Temperature	25

表 15-3　MQTT 数据格式

由于特殊的发布/订阅机制，服务器不需要存储数据（当然也可以在服务器的设备上建立一个客户端来订阅保存信息），因此非常适合海量设备的传输。

人生苦短，而 MicroPython 已经封装好了 MQTT 客户端的库文件。让我们的应用变得简单美妙。MQTT 模块文件为例程文件夹里面的 simple.py 文件。使用方法如下：

构造函数
client=simple. MQTTClient (client_id, server, port)
构建 MQTT 客户端对象。 client_id: 客户端 ID，具有唯一性； server: 服务器地址，可以是 IP 或者网址； port:服务器端口。（默认是 1883，服务器通常采用的端口，也可以自定义。）
使用方法

client.connect()
连接到服务器。
client.publish(TOPIC,message)
发布。TOPIC：主题编号；message：信息内容，例：'Hello 01Studio!'
client.subscribe(TOPIC)
订阅。TOPIC：主题编号。
client.set_callback(callback)
设置回调函数。callback：订阅后如果接收到信息，就执行相名称的回调函数。
client.check_msg()
检查订阅信息。如收到信息就执行设置过的回调函数 callback。

表 15-4　MQTT 客户端对象

由于客户端分为发布者和订阅者角色，因此为了方便大家更好理解，本实验分开两个案例来编程，分别为发布者和订阅者。再结合 MQTT 网络调试助手来测试。代表编写流程图如下：

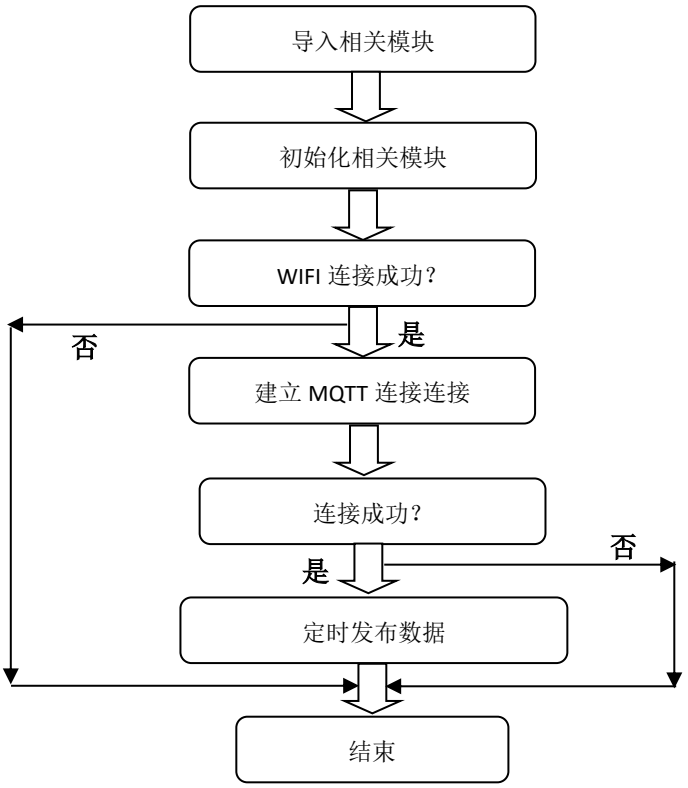


图 15-17 发布者代码编写流程

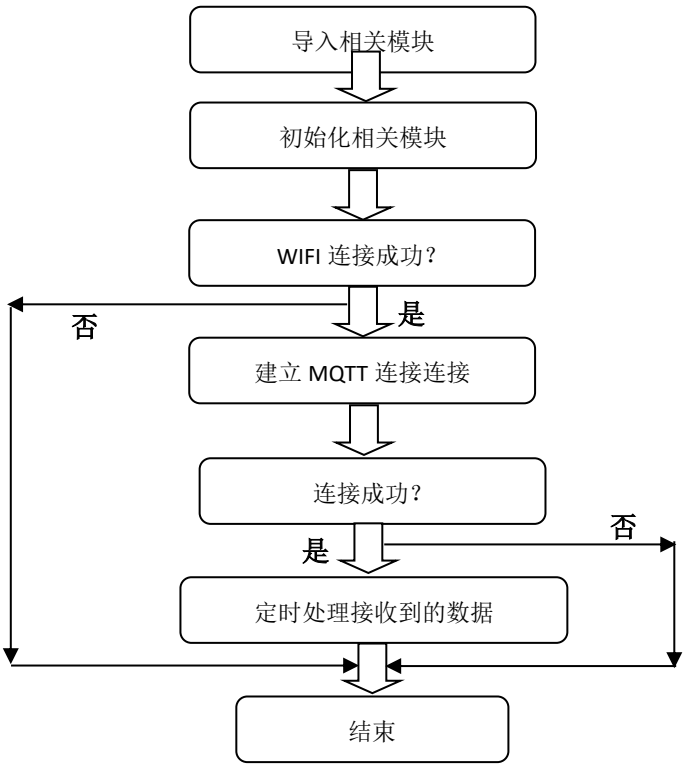


图 15-18 订阅者代码编写流程

发布者（publish）参考代码：

```
'''
实验名称：MQTT 通信
版本：v1.0
日期：2019.8
作者：01Studio
说明：编程实现 MQTT 通信，实现发布数据。
'''
import network,time
from simple import MQTTClient #导入 MQTT 板块
from machine import I2C,Pin,Timer
from ssd1306 import SSD1306_I2C


#初始化相关模块
```

```
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

#WIFI 连接函数
def WIFI_Connect():

    WIFI_LED=Pin(2, Pin.OUT) #初始化 WIFI 指示灯

    wlan = network.WLAN(network.STA_IF) #STA 模式
    wlan.active(True)                   #激活接口
    start_time=time.time()              #记录时间做超时判断

    if not wlan.isconnected():
        print('connecting to network...')
        wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码

        while not wlan.isconnected():

            #LED 闪烁提示
            WIFI_LED.value(1)
            time.sleep_ms(300)
            WIFI_LED.value(0)
            time.sleep_ms(300)

            #超时判断,15 秒没连接成功判定为超时
            if time.time()-start_time > 15 :
                print('WIFI Connected Timeout!')
                break

    if wlan.isconnected():
```

```
        #LED 点亮
        WIFI_LED.value(1)

        #串口打印信息
        print('network information:', wlan.ifconfig())

        #OLED 数据显示（如果没接 OLED，请将下面代码屏蔽）
        oled.fill(0)   #清屏背景黑色
        oled.text('IP/Subnet/GW:',0,0)
        oled.text(wlan.ifconfig()[0], 0, 20)
        oled.text(wlan.ifconfig()[1],0,38)
        oled.text(wlan.ifconfig()[2],0,56)
        oled.show()
        return True

    else:
        return False

#发布数据任务
def MQTT_Send(tim):
    client.publish(TOPIC, 'Hello 01Studio!')

#执行 WIFI 连接函数并判断是否已经连接成功
if WIFI_Connect():

    SERVER = 'mqtt.p2hp.com'
    PORT = 1883
    CLIENT_ID = 'ESP-32' # 客户端 ID
    TOPIC = '/public/01Studio/1' # TOPIC 名称
    client = MQTTClient(CLIENT_ID, SERVER, PORT)
```

```
    client.connect()


    #开启 RTOS 定时器，编号为-1,周期 1000ms，执行 socket 通信接收任务
    tim = Timer(-1)
    tim.init(period=1000, mode=Timer.PERIODIC,callback=MQTT_Send)
```

订阅者（subscribe）参考代码：

```
'''
实验名称：MQTT 通信
版本：v1.0
日期：2019.8
作者：01Studio
说明：编程实现 MQTT 通信，实现订阅（接收）数据。
'''
import network,time
from simple import MQTTClient #导入 MQTT 板块
from machine import I2C,Pin,Timer
from ssd1306 import SSD1306_I2C


#初始化相关模块
i2c = I2C(sda=Pin(13), scl=Pin(14))
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)


#WIFI 连接函数
def WIFI_Connect():


    WIFI_LED=Pin(2, Pin.OUT) #初始化 WIFI 指示灯


    wlan = network.WLAN(network.STA_IF) #STA 模式
```

```
wlan.active(True)                    #激活接口
start_time=time.time()               #记录时间做超时判断

if not wlan.isconnected():
    print('connecting to network...')
    wlan.connect('01Studio', '88888888') #输入 WIFI 账号密码

    while not wlan.isconnected():

        #LED 闪烁提示
        WIFI_LED.value(1)
        time.sleep_ms(300)
        WIFI_LED.value(0)
        time.sleep_ms(300)

        #超时判断,15 秒没连接成功判定为超时
        if time.time()-start_time > 15 :
            print('WIFI Connected Timeout!')
            break

if wlan.isconnected():
    #LED 点亮
    WIFI_LED.value(1)

    #串口打印信息
    print('network information:', wlan.ifconfig())

    #OLED 数据显示（如果没接 OLED，请将下面代码屏蔽）
    oled.fill(0)   #清屏背景黑色
    oled.text('IP/Subnet/GW:',0,0)
```

```
        oled.text(wlan.ifconfig()[0], 0, 20)
        oled.text(wlan.ifconfig()[1],0,38)
        oled.text(wlan.ifconfig()[2],0,56)
        oled.show()
        return True

    else:
        return False

#设置 MQTT 回调函数,有信息时候执行
def MQTT_callback(topic, msg):
    print('topic: {}'.format(topic))
    print('msg: {}'.format(msg))

#接收数据任务
def MQTT_Rev(tim):
    client.check_msg()

#执行 WIFI 连接函数并判断是否已经连接成功
if WIFI_Connect():

    SERVER = 'mqtt.p2hp.com'
    PORT = 1883
    CLIENT_ID = '01Studio-ESP32' # 客户端 ID
    TOPIC = '/public/01Studio/1' # TOPIC 名称

    client = MQTTClient(CLIENT_ID, SERVER, PORT) #建立客户端对象
    client.set_callback(MQTT_callback)  #配置回调函数
    client.connect()
    client.subscribe(TOPIC) #订阅主题
```

```
#开启 RTOS 定时器，编号为-1,周期 300ms，执行 socket 通信接收任务
tim = Timer(-1)
tim.init(period=300, mode=Timer.PERIODIC,callback=MQTT_Rev)
```

从以上代码可以看到发布者和订阅者的编程方式相近，另外本实验需要一个 MQTT 服务器（Broker），这里使用的是跟我们将用到的 MQTT 在线网络助手同一个服务器和端口。

```
SERVER = 'mqtt.p2hp.com'
PORT = 1883
```

- **实验结果：**

为了方便测试，我们可以使用 MQTT 网络助手进行调试。这里推荐一个在线 MQTT 网络调试助手：http://mqtt.p2hp.com/websocket/

打开上面网址，即可看到 MQTT 在线调试助手。可以配置基本信息，这里默认即可，点击连接。

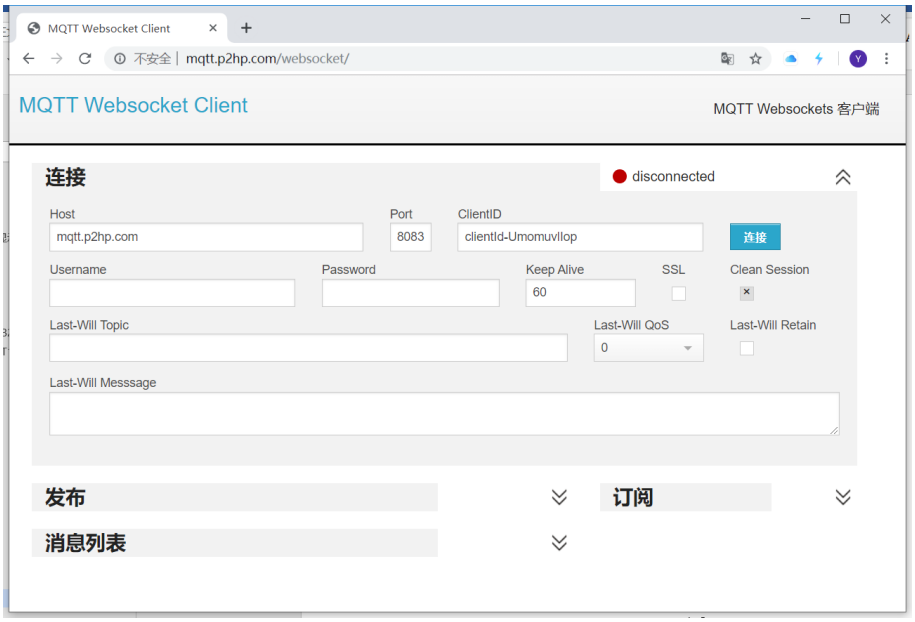


图 15-19 MQTT 助手

连接成功可以看到显示 Connected。

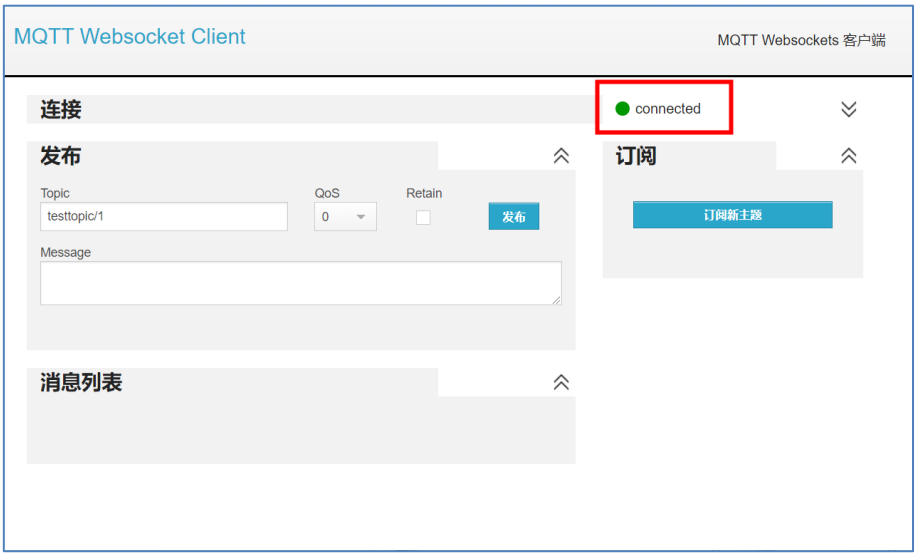


图 15-20 MQTT 助手连接成功

测试“发布者”代码：

我们先测试“发布者”代码。将“发布者”代码下载到开发板，然后在 MQTT 助手中订阅主题修改为：'/public/01Studio/1'（跟代码发布的主题一致），QOS 选择 0 即可。然后点击订阅主题。

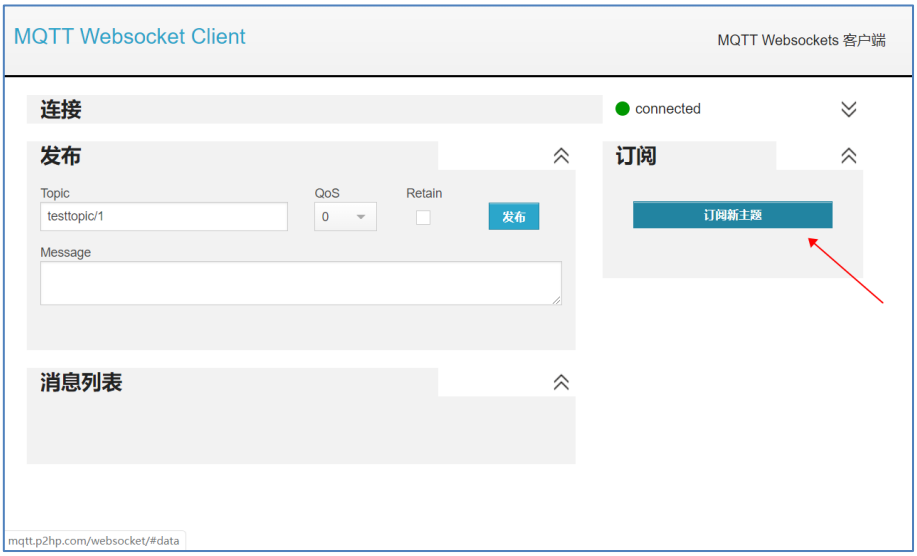


图 15-21 订阅主题

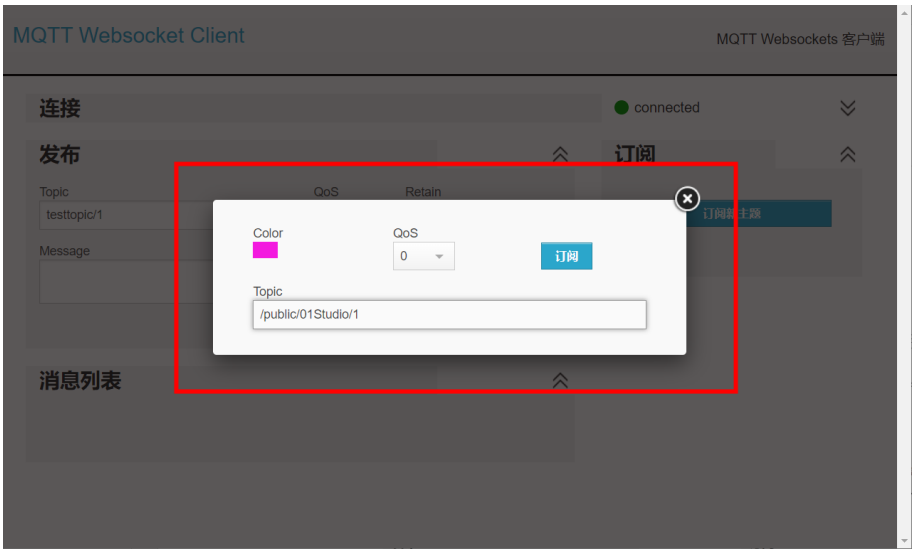


图 15-22

订阅后可以看到接收框收到来自开发板发布的信息。

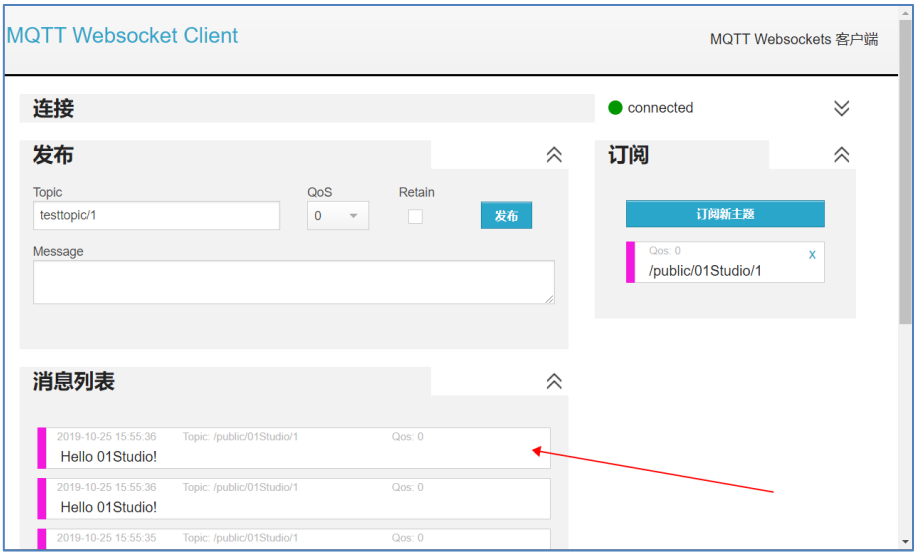


图 15-23 MQTT 助手收到开发板发布的信息

测试“订阅者”代码：

“订阅者”代码测试方法跟“发布者”相反。将“订阅者”代码下载到开发板，然后在电脑 MQTT 助手中发布主题修改为：'/public/01Studio/1'（跟代码发布的主题一致。）在下方空白框输入“Hello 01Studio!”。

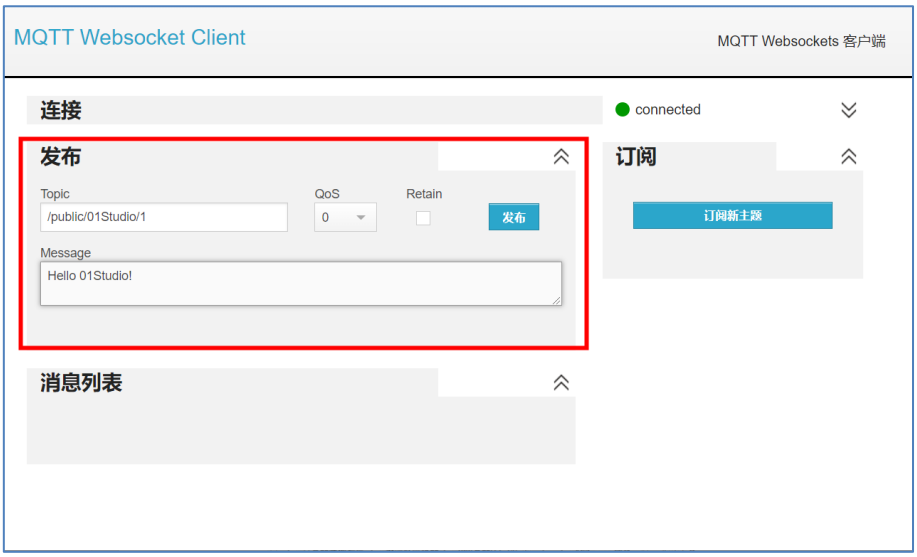


图 15-24 输入即将发布的主题和信息

将“订阅者”代码下载到开发板，成功连接后回到 MQTT 助手点击【发布】发布信息，可以在开发板的 REPL 看到接收到的订阅信息“Hello 01Studio!”被打印出来了（数据格式为字节数据）。

```
ESP MicroPython REPL
network information: ('192.168.1.117', '255.255.255.0', '192.168.1.1', '192.168.1.1')
◆◆>
MicroPython v1.11-8-g48dcbbe60 on 2019-05-29; ESP module with ESP8266
Type "help()" for more information.
>>> topic: b'/public/01Studio/1
msg: b'Hello 01Studio!'
```

图 15-25 开发板接收到相关信息并打印

当然你也可以在同一个 MQTT 在线助手下测试发布和订阅功能，只需要将主题设置一致即可，如下图所示：

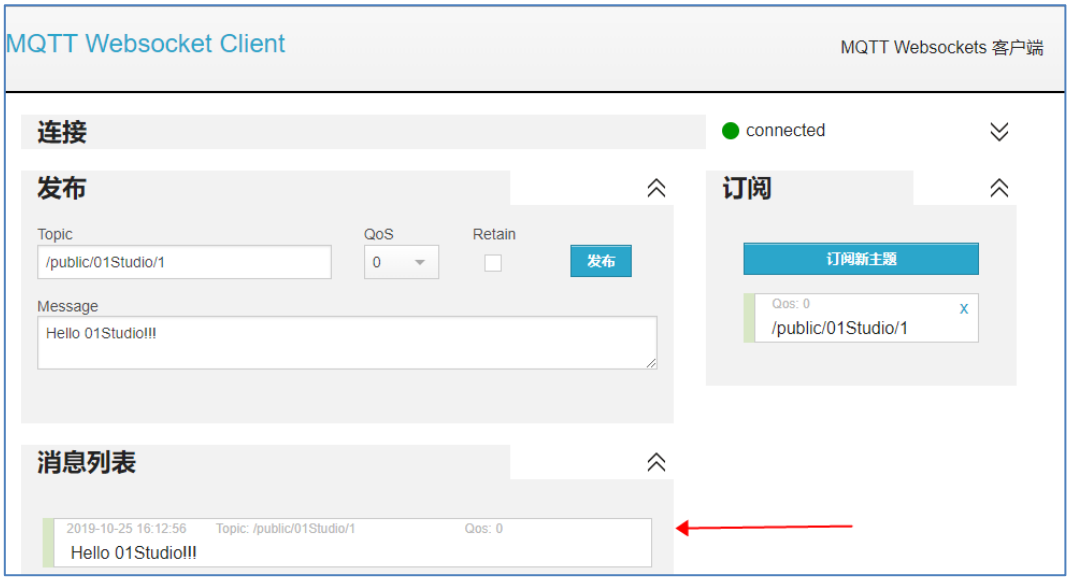


图 15-26 客户端同时发布和订阅信息

- **总结：**

通过本节我们了解了 MQTT 通信原理以及成功实现通信。目前市面上大部分物联网云平台支持 MQTT，原理大同小异。大家可以基于不同平台协议来开发，实现自己的物联网设备远程连接。

15.4 WebREPL

WebREPL 是指通过网页来和 pyWiFi-ESP32 做交互调试和传输文件，具体请阅读 **第三部分 10.4 WebREPL** 章节内容，这里不再重复。

第16章　LVGL（LittleVGL）

嵌入式设备开发离不开 GUI（图形用户界面），漂亮的界面能让你的产品设备倍增光彩，然而在 MicroPython 下有一款漂亮、控件齐全且简单易用的 GUI 嵌入式开源库正在日益成熟，那就是 LVGL。

本章以应用为主，着重介绍 LVGL 常规控件的使用方法，让用户可以轻松学习并将控件部署到自己的产品中去。

本实验平台使用 pyWiFi-ESP32 或 pyWiFi-ESP32P，我们更推荐使用 pyWiFi-ESP32P, 因为它搭载的主控模块是 ESP32-WROVER-B，外置 8M 的 PSRAM，有足够的内存来应对更复杂的 UI。

LCD 可以使用 01Studio 的 2.4 寸或者 3.2 寸 LCD，这两款 LCD 均使用 ILI9341(液晶驱动)和 XPT2046(触摸驱动)，接口兼容，只是尺寸大小的差异。

为了达到更好的演示效果，本章使用 pyWiFi-ESP32P + 3.2 寸 LCD（电阻触摸）来进行实验讲解。



图 16-1 LVGL 官方图片

16.1 LVGL（LittleVGL）简介

LittlevGL 是一个免费的开源图形库，提供了创建嵌入式 GUI 所需的一切，具有易于使用的图形元素、漂亮的视觉效果和低内存占用。

- **占用资源少**

  64Kb Flash 和 8kB RAM 足以满足简单的用户界面需求。

- **多种窗口小部件**

  可从 30 多种小部件中进行选择，并轻松定义和使用。

- **多平台**

  可以在众多平台使用 LVGL，例如：STM32, NXP, PIC, Arduino, ESP32， Raspberry 等。

- **支持多种显示器**

  可驱动单色、OLED、TFT 显示器和其它显示器。

- **支持 MicroPython**

  可在 MicroPython 中使用 LVGL 轻松创建你的 UI。

- **开源、免费**

  LVGL 由 MIT 许可的方式托管在 GitHub 上，真正的开源免费使用。

  英文官网：https://lvgl.io/

  中文网站：https://littlevgl.cn/

  官方文档（Docs）：https://docs.lvgl.io/latest/en/html/index.html

16.2　LCD 和触摸屏校准

- **前言：**

LVGL 对每个平台的通用性是建立在已经定义好了 LCD 和触摸屏的参数，不同的 LCD 和触摸有不同的定义，我们通过本节实现了 LCD 显示和触摸校准后，就可以轻松进行后面的 UI 实验了。

- **实验平台：**

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 16-2　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

编程实现 LCD 的参数初始化以及触摸屏校准。

- **实验讲解：**

LVGL 需要烧录专门的固件，固件路径位于**零一科技（01Studio）MicroPython 开发套件配套资料_latest\03-相关固件\03-pyWiFi-ESP32\02-LVGL 版（带 LittleVGL）**目录下，烧录固件方法请参考：**11.1.3.4　　固件更新** 章节内容，这里不再重复。

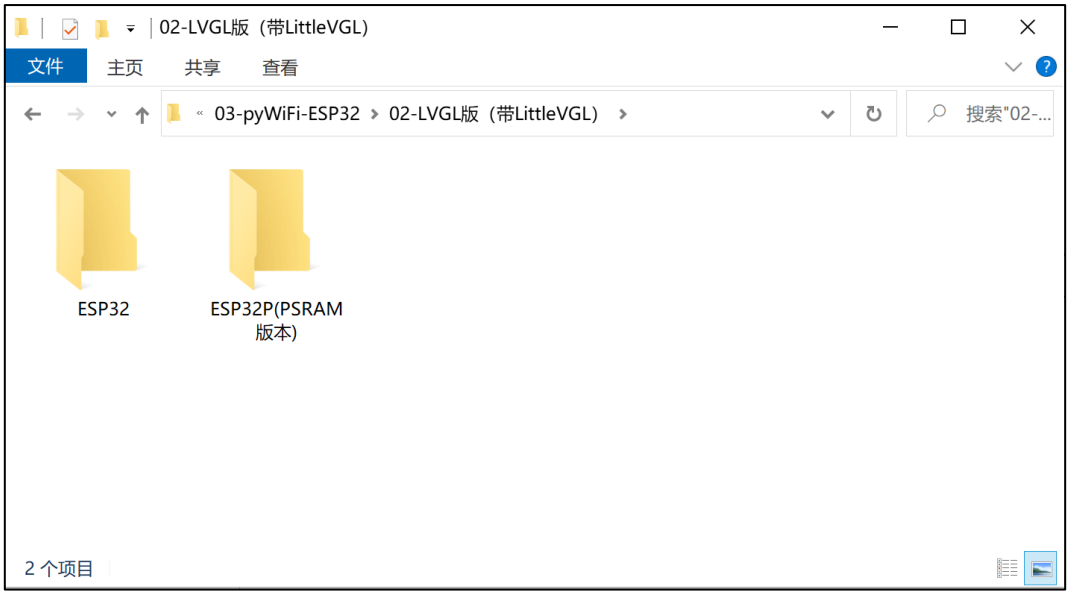


图 16-3 带 LVGL 的固件

01Studio 的 3.2 寸 LCD 电阻屏使用 ili9341+XPT2406(触摸)方案，我们先来看一下 LCD 和触摸屏的初始化函数用法。

LCD 初始化函数
disp = ili9341(miso, mosi, clk, cs, dc, rst, width, height, rot)
LCD 初始化； 【miso】SPI 引脚； 【mosi】SPI 引脚； 【clk】SPI 引脚； 【cs】SPI 引脚，片选； 【rst】复位 【width】LCD 宽度像素 【height】LCD 高度像素 【rot】LCD 方向；ili9341.PORTRAIT：垂直，ili9341.LANDSCAPE：水平

表 16-1 LCD 初始化函数

触摸屏初始化函数
touch = xpt2046(cs, transpose, cal_x0, cal_x1, cal_y0, cal_y1)

触摸屏初始化。 【cs】SPI 引脚，片选； 【transpose】方向。1：竖屏，0：横屏。 【cal_x0, cal_x1, cal_y0, cal_y1】触摸屏校准参数。

表 16-2 触摸屏初始化函数

在实际使用过程成我们并非每次都需要对触摸屏进行校准，正确的方式是将首次校准的参数保存为文件（json）存放在开发板中。以后上电时调用即可，同时应该支持横屏和竖屏的的校准。

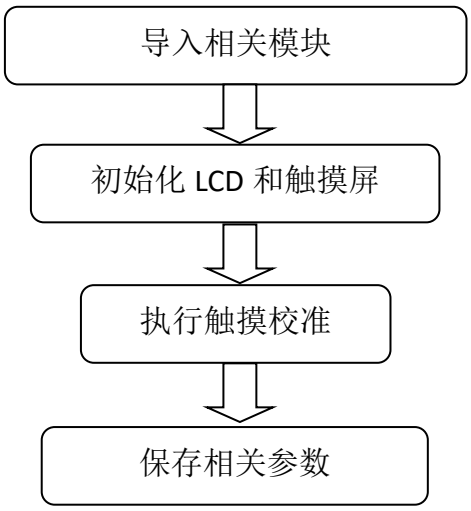


图 16-4 编程流程图

参考代码：

```
'''
实验名称：LCD 初始化和触摸屏校准
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
```

```
import lvgl as lv
import time,ujson

from ili9341 import ili9341
from xpt2046 import xpt2046

TFT_IS_PORTRAIT =1 #竖屏：1 ，横屏：0 ；
TOUCH_READY = 0 #用于检测触摸屏是否已经校准过；

#LCD ili9341 初始化
disp = ili9341(
    miso=12,
    mosi=13,
    clk=14,
    cs=15,
    dc=21,
    rst=33,
    power=50,  #硬件不支持，随便配一个参数
    backlight=51, #硬件不支持，随便配一个参数
    backlight_on= 1,
    power_on= 1,
    width=240 if TFT_IS_PORTRAIT else 320,
    height=320 if TFT_IS_PORTRAIT else 240,
    rot=ili9341.PORTRAIT if TFT_IS_PORTRAIT else ili9341.LANDSCAPE
    #垂直方向 PORTRAIT ；水平方向：LANDSCAPE
)

#触摸屏设置校准
TOUCH_CS = 2  #触摸屏 CS 片选引脚
TOUCH_INTERRUPT=0 #横屏
```

```
if TFT_IS_PORTRAIT:
    TOUCH_CALI_FILE = "touch_cali_PORTRAIT.json" #保存为竖屏触摸参数
else:
    TOUCH_CALI_FILE = "touch_cali_LANDSCAPE.json" #保存为横屏触摸参数

#从没做过触摸校准
if TOUCH_CALI_FILE not in uos.listdir():
    touch = xpt2046(
        cs=TOUCH_CS,
        transpose=TFT_IS_PORTRAIT,
    )

    from touch_cali import TouchCali

    touch_cali = TouchCali(touch, TOUCH_CALI_FILE)
    touch_cali.start()

#已经做过触摸校准，直接调用触摸参数文件
else:
    with open(TOUCH_CALI_FILE, 'r') as f:
        param = ujson.load(f)
        touch_x0 = param['cal_x0']
        touch_x1 = param['cal_x1']
        touch_y0 = param['cal_y0']
        touch_y1 = param['cal_y1']

    touch = xpt2046(
        cs=TOUCH_CS,
        transpose=TFT_IS_PORTRAIT,
```

```
        cal_x0=touch_x0,
        cal_x1=touch_x1,
        cal_y0=touch_y0,
        cal_y1=touch_y1,
    )

    TOUCH_READY = 1 #表示已经配置好触摸参数

################################################
################  简单的触摸测试 ###############
################################################
if TOUCH_READY:

    print("LCD and TOUCH is ready!")

    #简单的矩形块触摸拖动测试
    obj = lv.obj(lv.scr_act())
    obj.set_size(100,50)
    obj.align(None, lv.ALIGN.CENTER, 0, 0)
    obj.set_drag(True)
```

- **实验结果：**

记得将例程中的 touch_cali.py 文件拷贝到开发板文件系统中，然后运行主函数代码，可以看到 LCD 提示触摸校准，我们按提示一次点击“+”标志来校准。

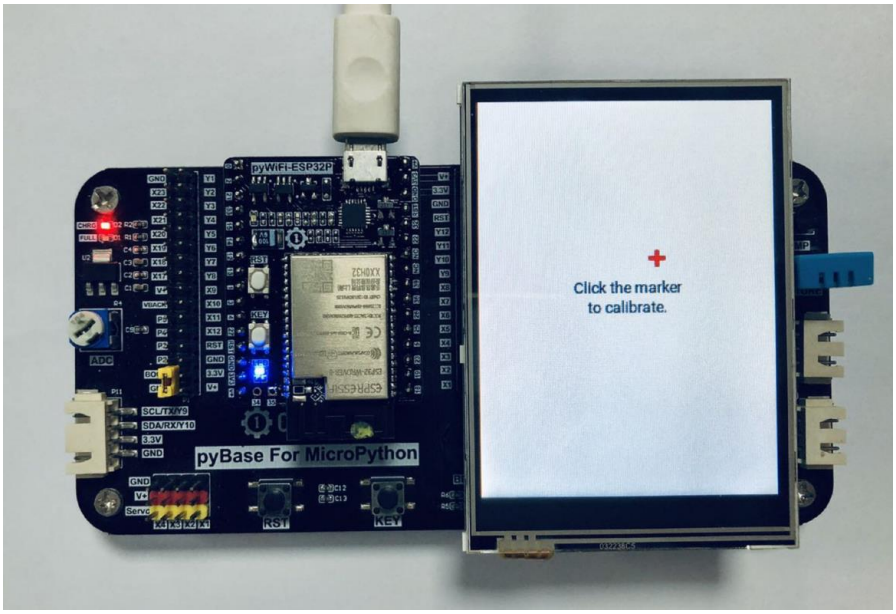


图 16-5 校准界面

校准完成后点击屏幕，开发板自动复位。（也可以直接按开发板 RST），这是我们在 Mu 中打开文件系统，可以看到多了一个“touch_cali_PORTRAIT.json”文件，这个就是刚刚记录的触摸参数文件，以后每次上电都会自动读取参数。

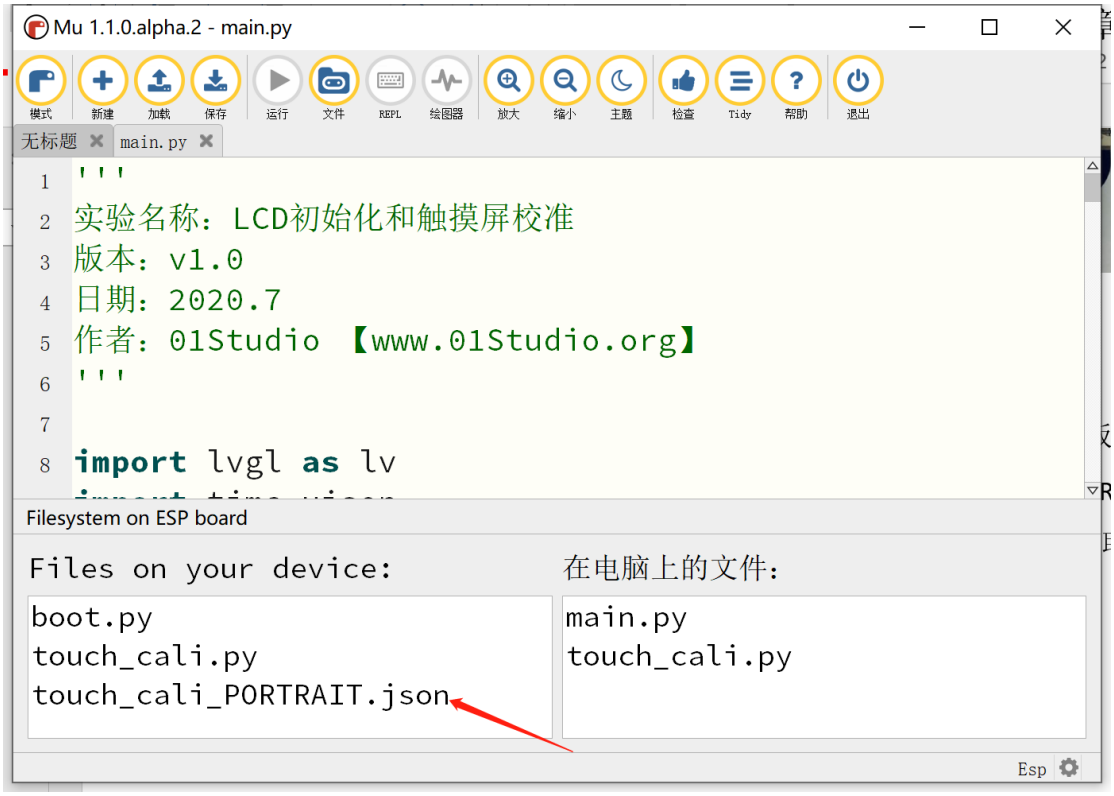


图 16-6 触摸参数文件

同时重新运行主程序后我们可以看到 LCD 出现一个方块，可以拖动用户测试触摸屏。

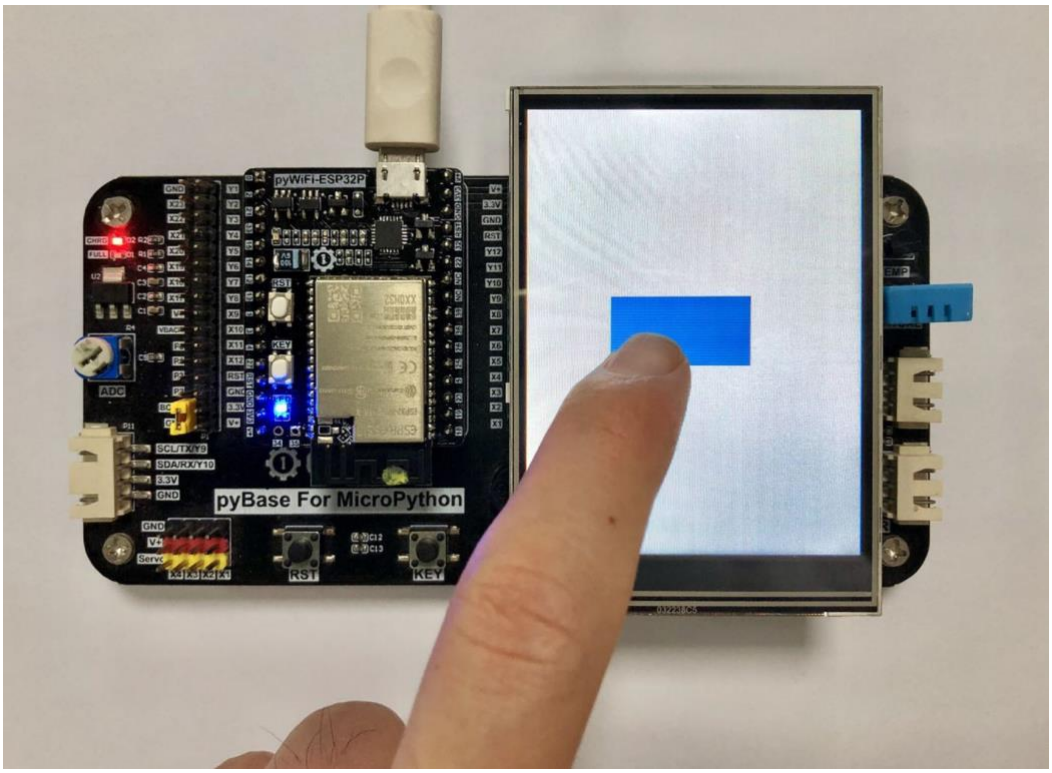


图 16-7 可以触摸拖动的方块

- **总结：**

本节我们学习了最基础但最重要的 LCD 初始化和触摸校准，这取决于我们采用的硬件平台和液晶屏参数。不同初始化成功后，后面 LVGL 的使用方式和代码都是完全一样的，可移植性非常强。

16.3 小部件实验

一个完整的 UI 可以拆分成多个小部件（Widgets），如按钮、开关、菜单、列表等等，LVGL 目前拥有数十个小部件，本节将对每个小部件的功能和编程方法进行讲解。

16.3.1 Basc object（基础对象）

- **前言：**

基础对象是最基础的部件，但它体现了小部件的一些基本属性，如坐标、父对象继承、风格、可点击和拖动等。

- **实验平台：**

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 16-8 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

学习创建不同的基础对象。

- **实验讲解：**

基础对象默认的就是矩形方块形状，可以通过编程赋予不同的显示风格和相关功能。

构造函数
obj = lv.obj(lv.scr_act())
构建 obj 对象； lv.scr_act() :表示在当前屏幕创建；

使用方法
obj.set_size()
设置尺寸；
obj.set_style()
设置风格；
obj.align()
设置对齐方式；
obj.set_drag()
设置可拖动；
更多用法请参阅小部件说明文档： （或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/obj.html

表 16-3 object 对象

编程思路如下：

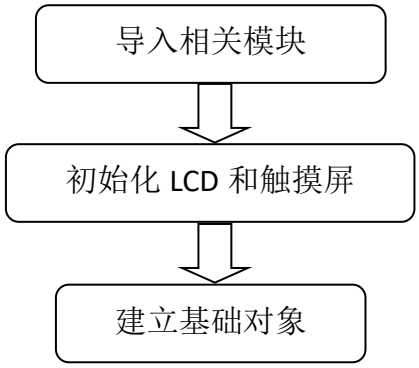


图 16-9 编程流程图

参考代码：**（LCD 和触摸校准代码上一节已讲述，这里省略，完整代码参考配套资料包文件夹里面的示例程序）。**

```
'''
实验名称：Base object(基础对象)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
```

```
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件

################################################
################  Basc object  ##############
################################################
if TOUCH_READY:

    #基础对象 1
    obj1 = lv.obj(lv.scr_act())
    obj1.set_size(100,50)
    obj1.set_style(lv.style_plain_color)
    obj1.align(None, lv.ALIGN.CENTER, -60, -30)

    #基础对象 2，继承于基础对象 1，设置为可拖动
    obj2 = lv.obj(lv.scr_act(),obj1)
    obj2.set_style(lv.style_pretty_color)
    obj2.align(None, lv.ALIGN.CENTER, 0, 0)
    obj2.set_drag(True)

    #基础对象 3，继承于基础对象 2，新增风格
    style_shadow = lv.style_t()
    lv.style_copy(style_shadow, lv.style_pretty)
    style_shadow.body.shadow.width = 6
    style_shadow.body.radius = 800 # large enough to make it round

    obj3 = lv.obj(lv.scr_act(),obj2)
    obj3.set_style(style_shadow)
    obj3.align(None, lv.ALIGN.CENTER, 60, 30)
```

本例程创建了 3 个基础对象，obj2 继承于 obj1，obj3 继承于 obj2，其中 obj2 和 obj3 均可拖动。

- **实验结果：**

将示例代码的 py 库文件拷贝到开发板文件系统（例程提供了触摸参数文件，可以直接使用，也可以自行触摸校准来生成）。

然后通过运行或者拷贝 main.py 到开发板方式运行程序。

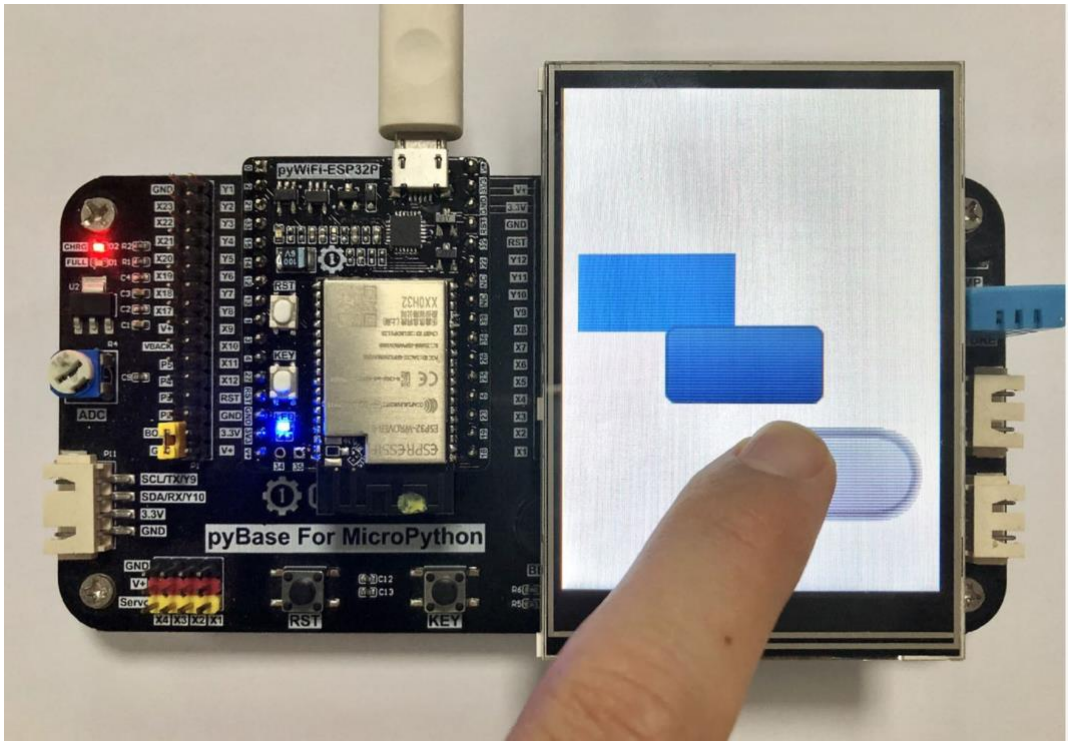


图 16-10 实验结果

- **总结**

基础对象是最基础的小部件，通过本节的学习，让我们对 LVGL 的编程方法有了一定的了解。

16.3.2 Bar（进度条）

- **前言：**

  进度条常用于显示加载进度。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

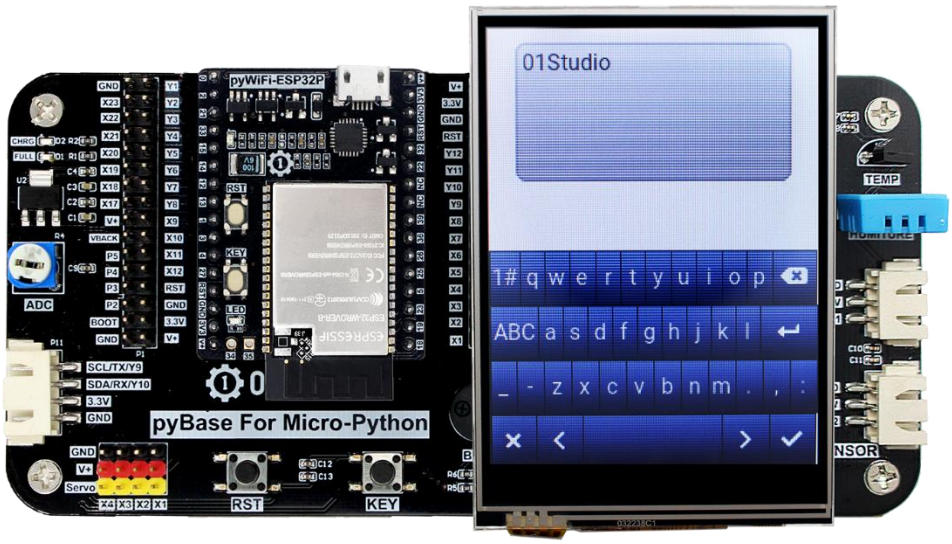


图 16-11　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建进度条。

- **实验讲解：**

  我们来看看进度条对象。

构造函数
`bar = lv.bar(lv.scr_act())`
构建 bar 对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
bar.set_size()

设置尺寸；
bar.align()
设置对齐方式；
bar.set_anim_time()
设置进度条加载时间；
bar.set_value(value,anim)
【value】范围：0-100，表示进度条最大值； 【anim】 lv.ANIM.ON：用动画设置值; lv.ANIM.OFF：立即更改值。
更多用法请参阅基小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/bar.html

表 16-4 进度条对象

编程思路如下：

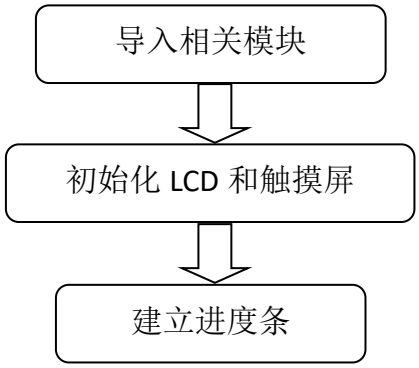


图 16-12 编程流程图

参考代码如下：

```
'''
实验名称：Bar(进度条)
版本：v1.0
日期：2020.7
```

```
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件


################################################
####################  Bar  #################
################################################
if TOUCH_READY:


    bar = lv.bar(lv.scr_act()) #构建进度条对象
    bar.set_size(200, 30) #设置尺寸
    bar.align(None, lv.ALIGN.CENTER, 0, 0) #居中
    bar.set_anim_time(1000) #进度条加载时间
    bar.set_value(100, lv.ANIM.ON)
    #进度条最大值 100（满），lv.ANIM.ON 表示动画效果。
```

本例程创建了 1 个进度条，进度条值为满 100%，动画效果。

- **实验结果：**

运行代码，可以看到 LCD 显示一个进度条。

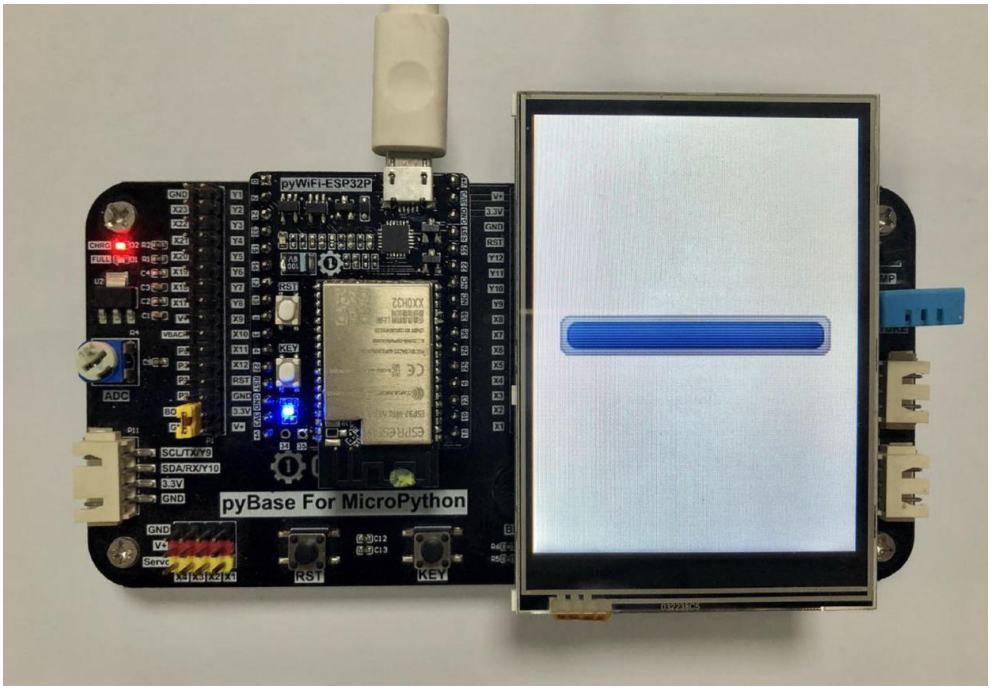


图 16-13 进度条实验结果

- **总结**

本节主要讲解了 LVGL 中进度条的编程使用方法。

16.3.3 Button（按钮）

- **前言：**

  按钮是非常常用的交互工具部件。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

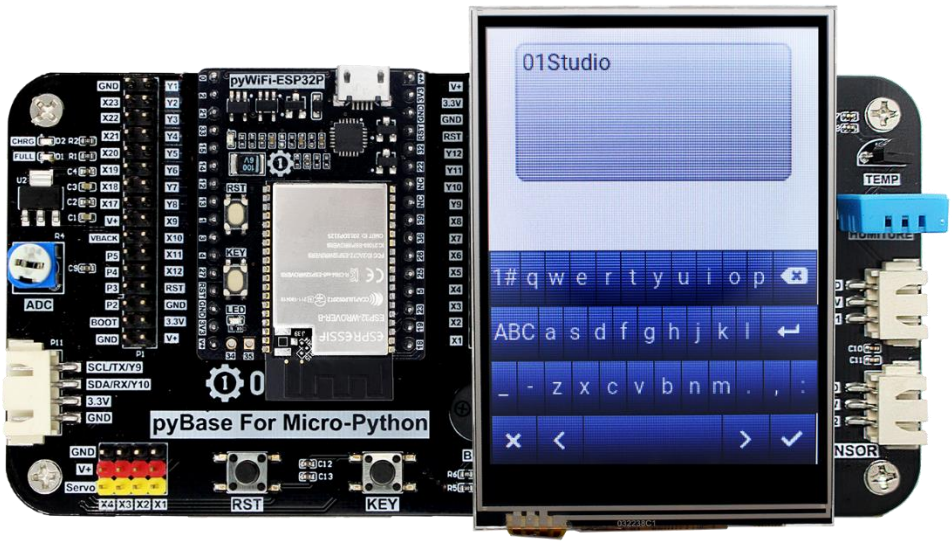


图 16-14　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建按钮。

- **实验讲解：**

  我们来看看按钮条对象。

构造函数
`btn = lv.btn(lv.scr_act())`
构建按钮对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
btn.set_event_cb(callback)

定义按钮事件对应的回调函数；
btn.align()
设置对齐方式；
btn.set_toggle(True)
设置为状态型按钮；
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/btn.html

表 16-5 按钮对象

编程思路如下：

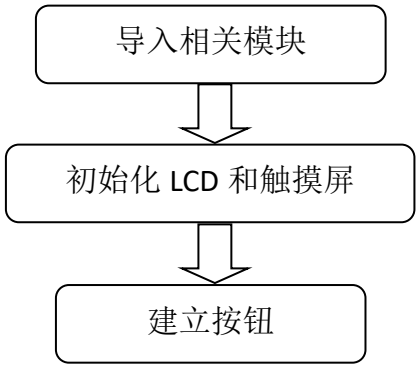


图 16-15 编程流程图

参考代码如下：

```
'''
实验名称：Button(按钮)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件


################################################
```

```
####################  Button  ###############
################################################
#按钮回调函数
def event_handler(obj, event):
    if event == lv.EVENT.CLICKED:
        print("Clicked")


if TOUCH_READY:


    #### 普通按钮 ####
    btn1 = lv.btn(lv.scr_act())
    btn1.set_event_cb(event_handler) #定义按钮触发的回调函数
    btn1.align(None, lv.ALIGN.CENTER, 0, -40)


    label = lv.label(btn1) #为 btn1 增加 label
    label.set_text("Button")


    #### 开关状态型按钮 ####
    btn2 = lv.btn(lv.scr_act())
    # lambda 实现回调函数:
    btn2.set_event_cb(lambda obj, event: print("Toggled") if event ==
                          lv.EVENT.VALUE_CHANGED else None)
    btn2.align(None, lv.ALIGN.CENTER, 0, 40)
    btn2.set_toggle(True) #设置为状态按钮


    label = lv.label(btn2) #为 btn2 增加 label
    label.set_text("Toggled")
```

本例程创建了 2 个按钮，点击后可以在 REPL 打印出按钮操作相关信息。

- **实验结果：**

运行代码，可以看到屏幕显示 2 个按钮。

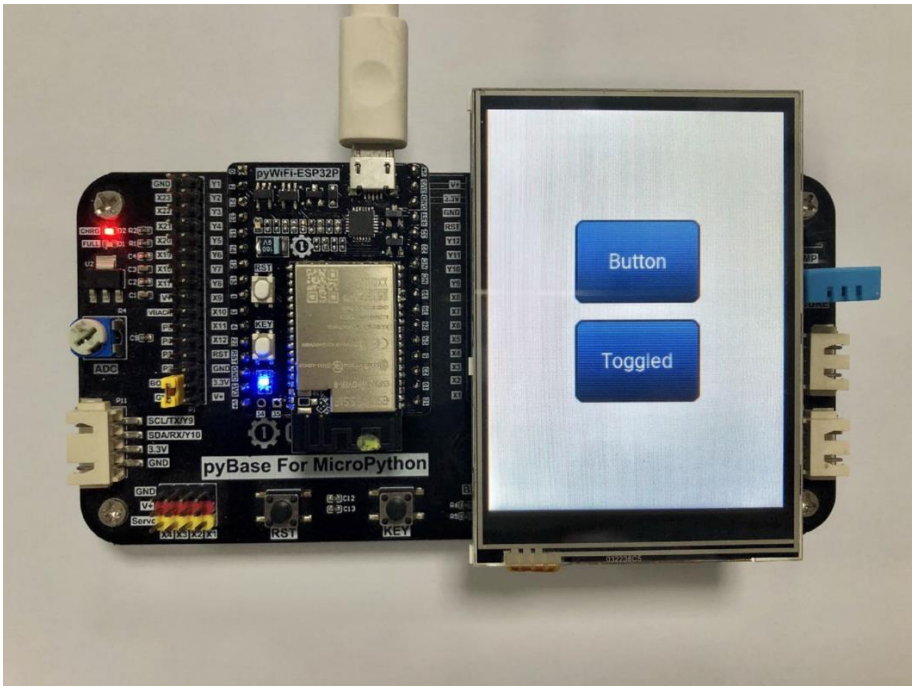
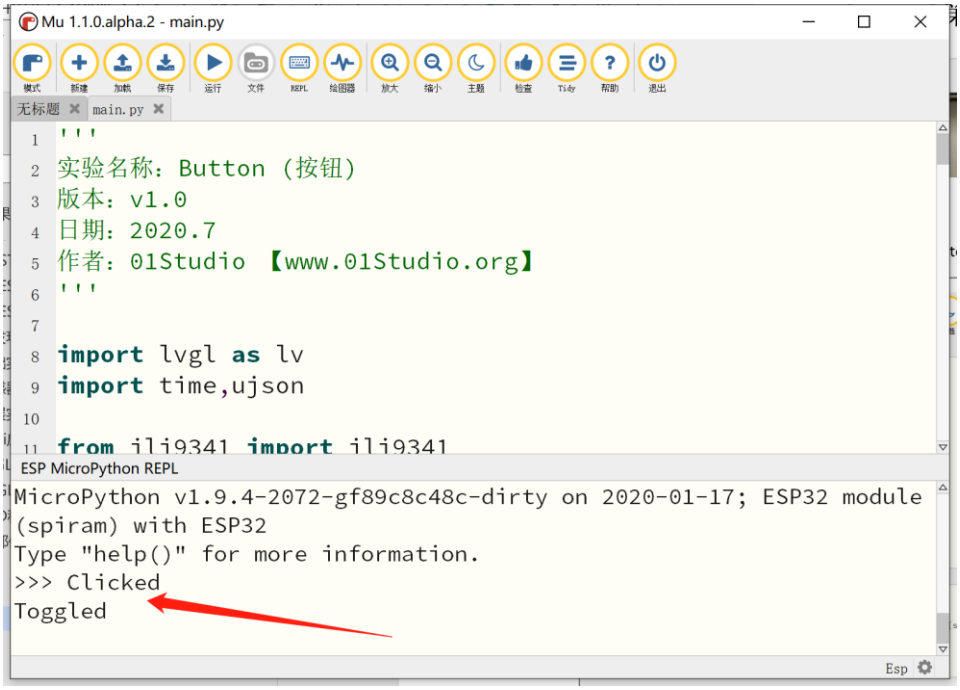


图 16-16 按钮显示

分别按 2 个按钮，可以看到串口 REPL 显示 click 和 toggle 提示信息：



- **总结**

本节主要讲解了 LVGL 中按钮的编程使用方法。

16.3.4 Button Matrix（矩阵键盘）

- **前言：**

矩阵键盘是一系列按钮的集合，如计算器、电脑键盘等。在 LVGL 下定义和使用非常方便。

- **实验平台：**

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 16-17　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

创建矩阵键盘。

- **实验讲解：**

我们来看看矩阵键盘对象。

构造函数
`btnm = lv.btnm(lv.scr_act())`
构建矩阵键盘对象； lv.scr_act() :表示在当前屏幕创建；
使用方法

btnm.set_event_cb(callback)
定义按钮事件对应的回调函数；
btnm.align()
设置对齐方式；
btnm.set_map(btnm_map)
定义矩阵键盘所有按钮；
btnm.set_width()
设置宽度
btnm.set_btn_width(num,size)
设置指定按钮的大小。 【num】第 num 个按钮，例如 10 表示第 10 个按钮； 【size】默认 1，例如 2 表示是默认的 2 倍尺寸
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/btnmatrix.html

表 16-6 矩阵键盘对象

编程思路如下：

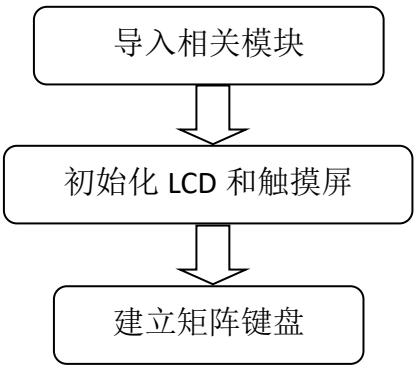


图 16-18 编程流程图

参考代码如下：

```
'''
实验名称：Bar(进度条)
版本：v1.0
```

```
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
################################################
############### Button Matrix ###############
################################################
#矩阵键盘回调函数
def event_handler(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        txt = obj.get_active_btn_text()
        print("%s was pressed" % txt)

if TOUCH_READY:
    #定义矩阵键盘，"\n"表示换行
    btnm_map = ["1", "2", "3", "4", "5", "\n",
                "6", "7", "8", "9", "0", "\n",
                "Action1", "Action2", ""]

    btnm1 = lv.btnm(lv.scr_act())
    btnm1.set_map(btnm_map)
    btnm1.set_btn_width(10, 2)
    # Make "Action1" twice as wide as "Action2"

    btnm1.set_width(240)
    btnm1.align(None, lv.ALIGN.CENTER, 0, 0)
    btnm1.set_event_cb(event_handler) #定义按钮事件回调函数
```

本例程创建了 1 个矩阵键盘，点击后可以在 REPL 打印出按钮操作相关信息。

- **实验结果：**

运行代码，可以看到屏幕显示 2 个按钮。

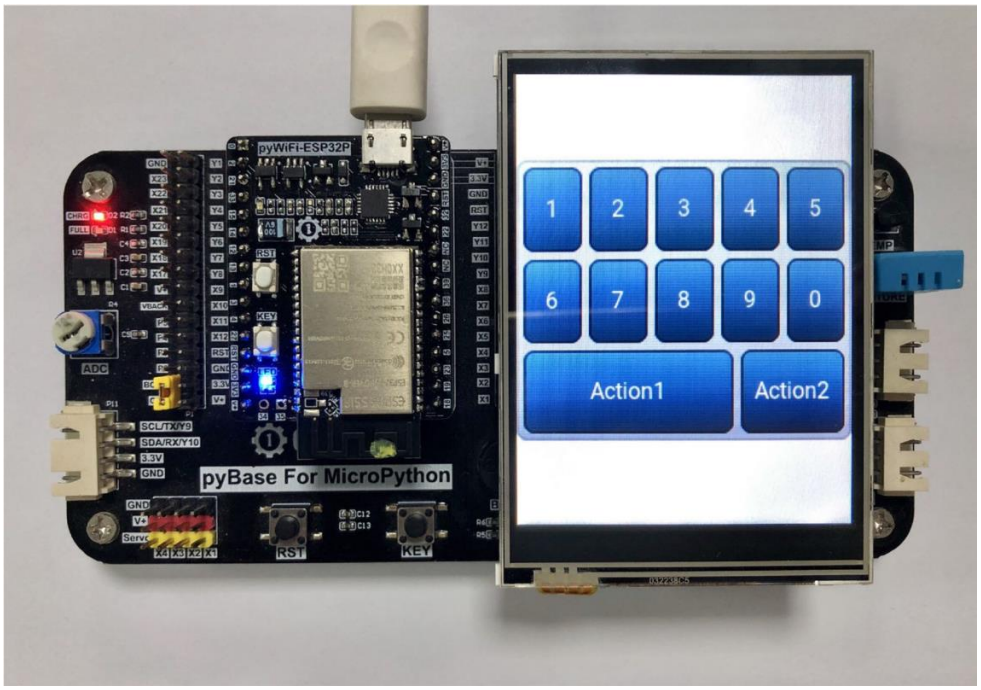


图 16-19 矩阵键盘显示

分别按 2 个按钮，可以看到串口 REPL 显示 click 和 toggle 提示信息：

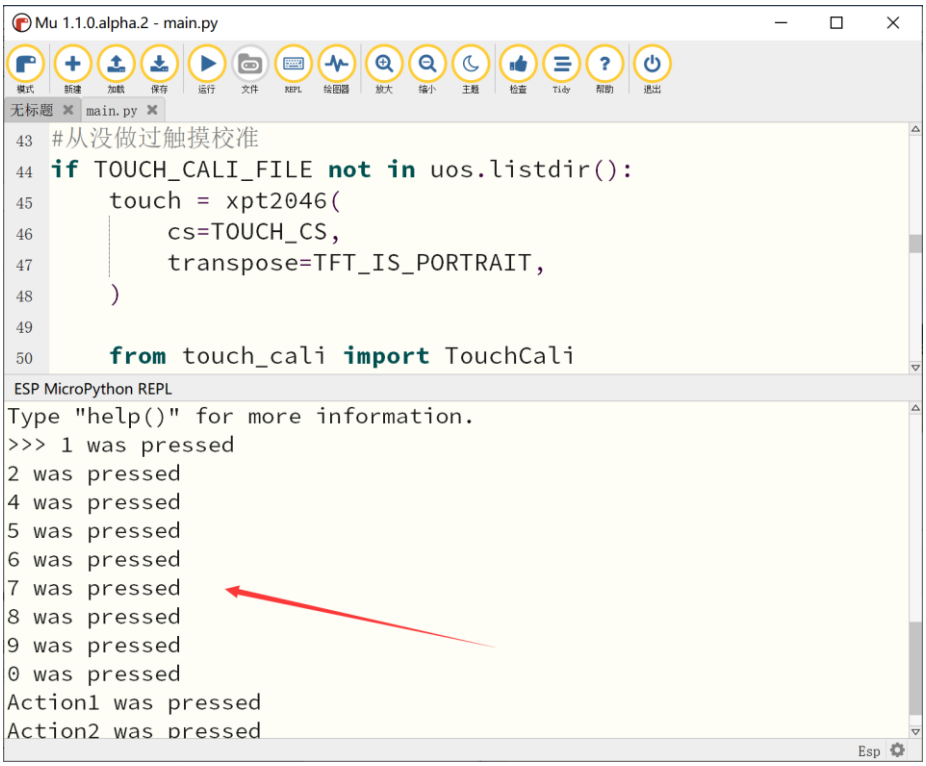


图 16-20 REPL 显示按钮出发信息

- **总结**

本节主要讲解了 LVGL 中矩阵键盘的编程使用方法。

16.3.5 Calendar（日历）

- **前言：**

  日历就是常说的万年历，显示年、月、日、星期信息。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

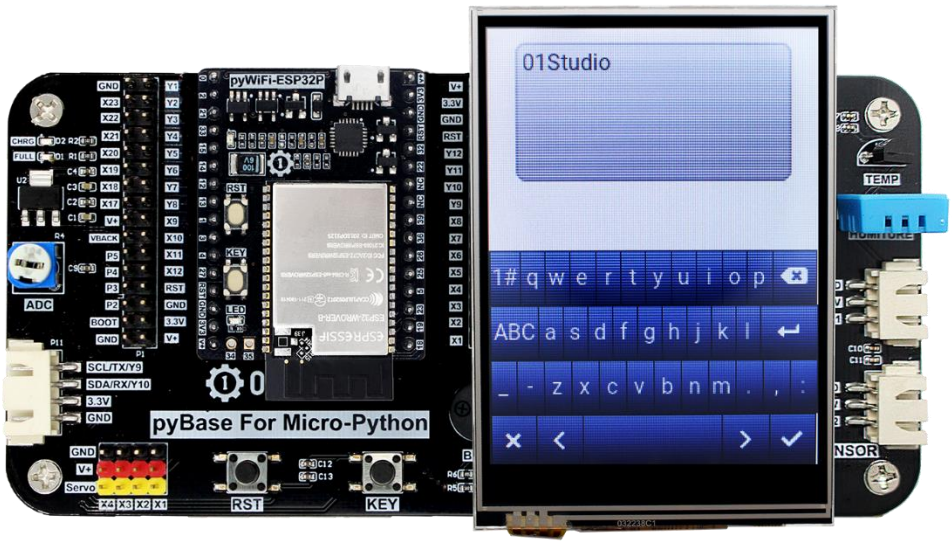


图 16-21　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建日历。

- **实验讲解：**

  我们来看看日历对象。

构造函数
calendar = lv.calendar(lv.scr_act())
构建日历对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
calendar.set_event_cb(callback)

定义触摸事件对应的回调函数；
calendar.align()
设置对齐方式；
calendar.set_size(width,height)
定义显示尺寸。
calendar.set_today_date(today)
设置日期；需要构建 today 对象； 例：定义 2020 年 7 月 1 日 today = lv.calendar_date_t() today.year = 2020 today.month = 7 today.day =1
calendar.set_showed_date(today)
显示指定日期。
calendar.set_highlighted_dates(highlihted_days, len(highlihted_days))
特定日期高亮状态显示。
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/calendar.html

表 16-7 日期对象

编程思路如下：

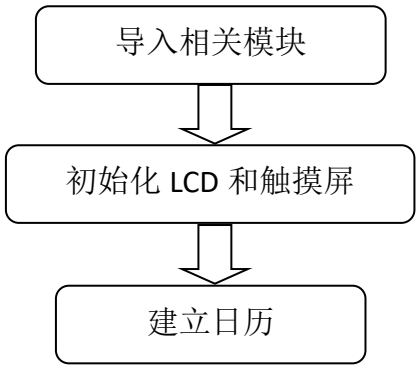


图 16-22 编程流程图

参考代码如下：

```
'''
实验名称：Calendar(日历)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
################################################
############### Calendar(日历) ##############
################################################
#日历触摸回调函数
def event_handler(obj, event):
    if event == lv.EVENT.CLICKED:
        date = obj.get_pressed_date()
        if date is not None:
            obj.set_today_date(date) #显示按下选中的日期

if TOUCH_READY:

    calendar = lv.calendar(lv.scr_act())
    calendar.set_size(230, 230)
    calendar.align(None, lv.ALIGN.CENTER, 0, 0)
    calendar.set_event_cb(event_handler)

    #设置当前日期
    today = lv.calendar_date_t() #构建日期对象
    today.year = 2020
    today.month = 7
    today.day = 1
```

```
calendar.set_today_date(today)
calendar.set_showed_date(today) #设置当前显示日期

#设置显示 3 个高亮日期
highlihted_days = [
    lv.calendar_date_t({'year':2020, 'month':7, 'day':10}),
    lv.calendar_date_t({'year':2020, 'month':7, 'day':15}),
    lv.calendar_date_t({'year':2020, 'month':7, 'day':20})
]

calendar.set_highlighted_dates(highlihted_days,len(highlihted_days))
```

- **实验结果：**

运行代码，可以看到屏幕显示 2 个按钮。

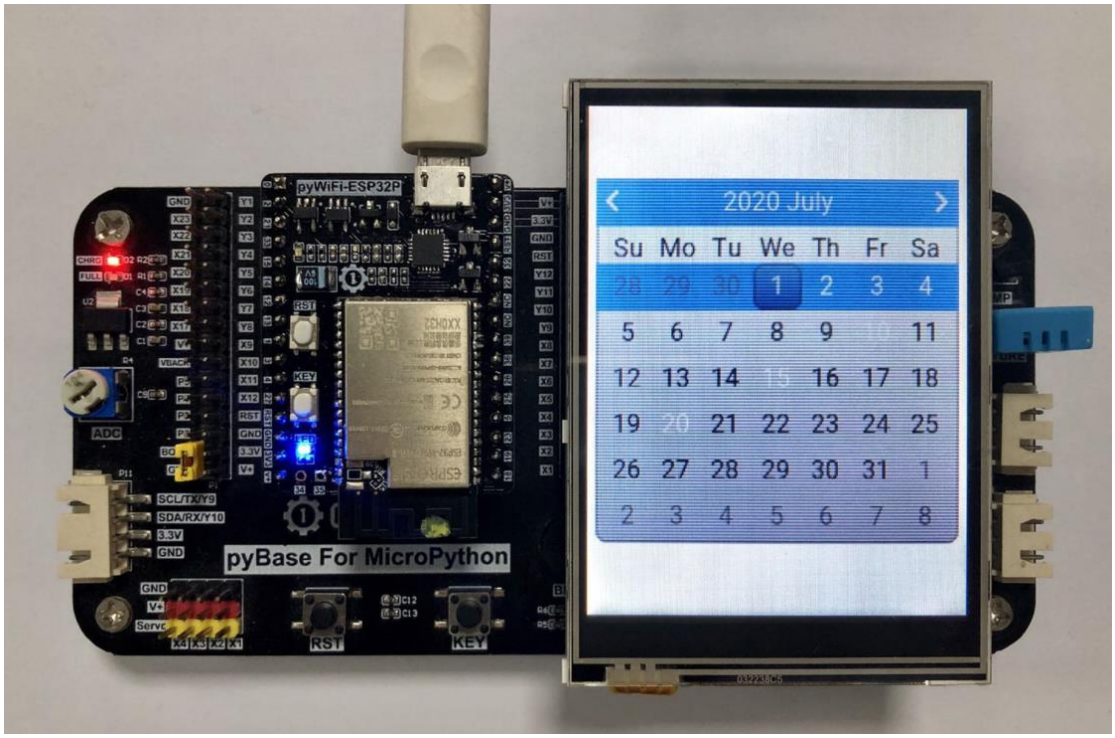


图 16-23 日期显示

- **总结**

本节主要讲解了 LVGL 中日历小部件的编程使用方法。

16.3.6 CheckBox（复选框）

- **前言：**

  在做调查问卷中我们遇到多项选择题每一个选项就是一个复选框。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 16-24　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建复选框。

- **实验讲解：**

  我们来看看复选框对象。

构造函数
cb= lv.cb(lv.scr_act())
构建复选框对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
cb.set_event_cb(callback)

定义触摸事件对应的回调函数；
cb.align()
设置对齐方式；
cb.set_text(string)
设置复选框文本；
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/checkbox.html

表 16-8 复选框对象

编程思路如下：

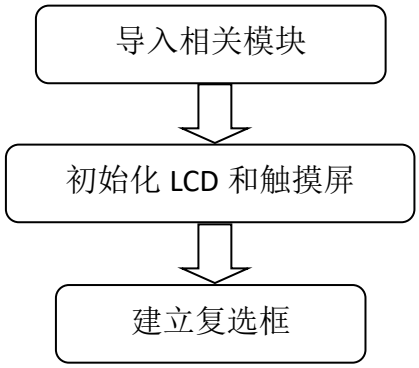


图 16-25 编程流程图

参考代码如下：

```
'''
实验名称：CheckBox(复选框)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
################################################
###############    Checkbox    ################
################################################
```

```
def event_handler(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        print("State: %s" % ("Checked" if obj.is_checked() else
"Unchecked"))

if TOUCH_READY:

    #构建复选框对象
    cb = lv.cb(lv.scr_act())
    cb.set_text("I agree !")
    cb.align(None, lv.ALIGN.CENTER, 0, 0)
    cb.set_event_cb(event_handler) #定义选中回调函数
```

- **实验结果：**

运行代码，可以看到屏幕显示 2 个按钮。

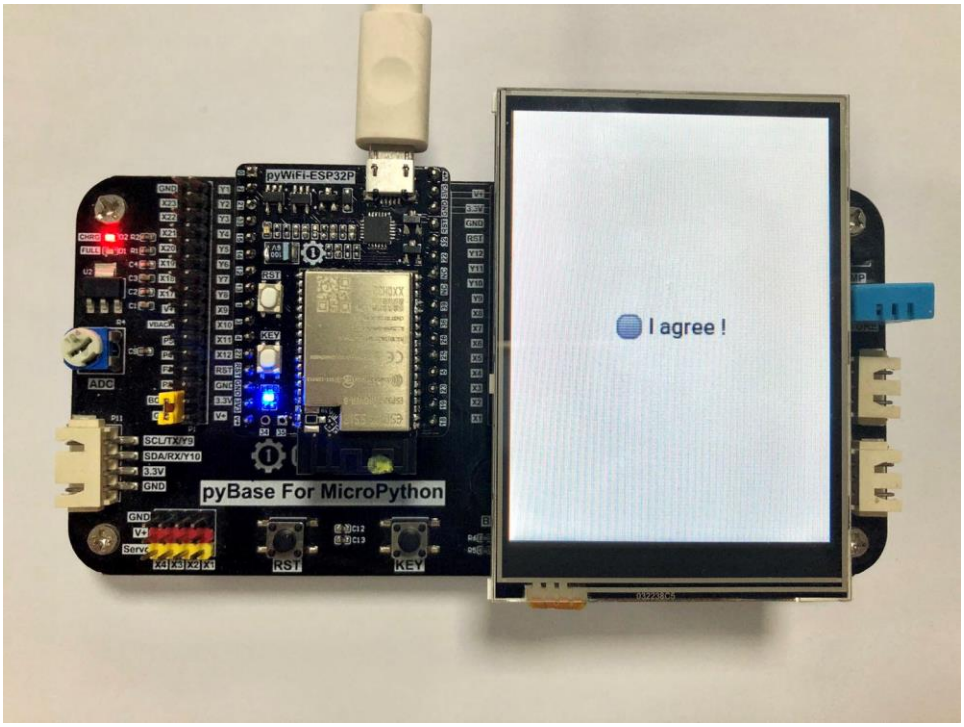


图 16-26 实验结果

- **总结**

本节主要讲解了 LVGL 中复选框小部件的编程使用方法。

16.3.7 Chart（图表）

- **前言：**

  图表是常用的功能，如柱状图、折线图等绘制都用到图表。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

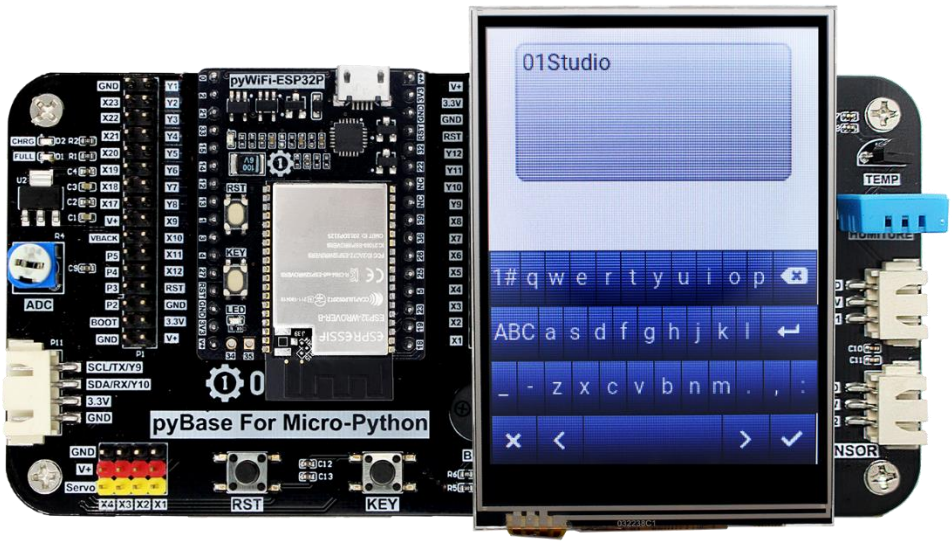


图 16-27　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建图表。

- **实验讲解：**

  我们来看看图表对象。

构造函数
chart= lv.chart(lv.scr_act())
构建图表对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
chart.set_size (width,height)

定义尺寸。
chart.align()
设置对齐方式；
chart.set_type(lv.chart.TYPE.POINT | lv.chart.TYPE.LINE)
设置图表显示类型； 例：lv.chart.TYPE.POINT | lv.chart.TYPE.LINE 表示同时显示线和点。
chart.set_series_width(value)
设置线宽大小；
chart.set_range(0, 100)
设置纵坐标范围：0-100
ser1 = chart.add_series(lv.color_make(0xFF,0,0))
新建一个项目并设置颜色：lv.color_make(0xFF,0,0) 表示红色。
chart.set_points(ser1, [10, 10, 10, 10, 10, 10, 10, 30, 70, 90])
设置项目的曲线值。
更多用法请参阅小部件说明文档： （或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/chart.html

表 16-9 图表对象

编程思路如下：

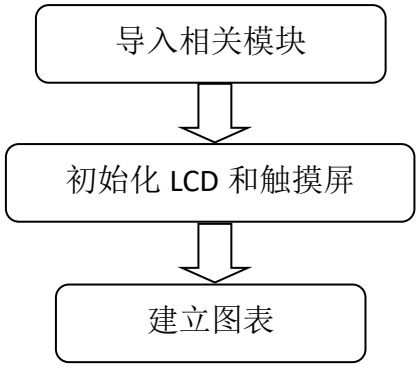


图 16-28 编程流程图

参考代码如下：

```
'''
实验名称：Chart(图表)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
################################################
###############   Chart   ###################
################################################

if TOUCH_READY:

    #创建图表
    chart = lv.chart(lv.scr_act())
    chart.set_size(200, 150)
    chart.align(None, lv.ALIGN.CENTER, 0, 0)
   # 同时显示线和点
    chart.set_type(lv.chart.TYPE.POINT | lv.chart.TYPE.LINE)
    chart.set_series_width(4)   # 线宽

    chart.set_range(0, 100) #纵坐标范围

    # Add two data series
    ser1 = chart.add_series(lv.color_make(0xFF,0,0))
    ser2 = chart.add_series(lv.color_make(0,0x80,0))

    # Set points on 'dl1'
    chart.set_points(ser1, [10, 10, 10, 10, 10, 10, 10, 30, 70, 90])
```

```
# Set points on 'dl2'
chart.set_points(ser2, [90, 70, 65, 65, 65, 65, 65, 65, 65, 65])
```

- **实验结果：**

运行代码，可以看到屏幕显示图表出现 2 个曲线。

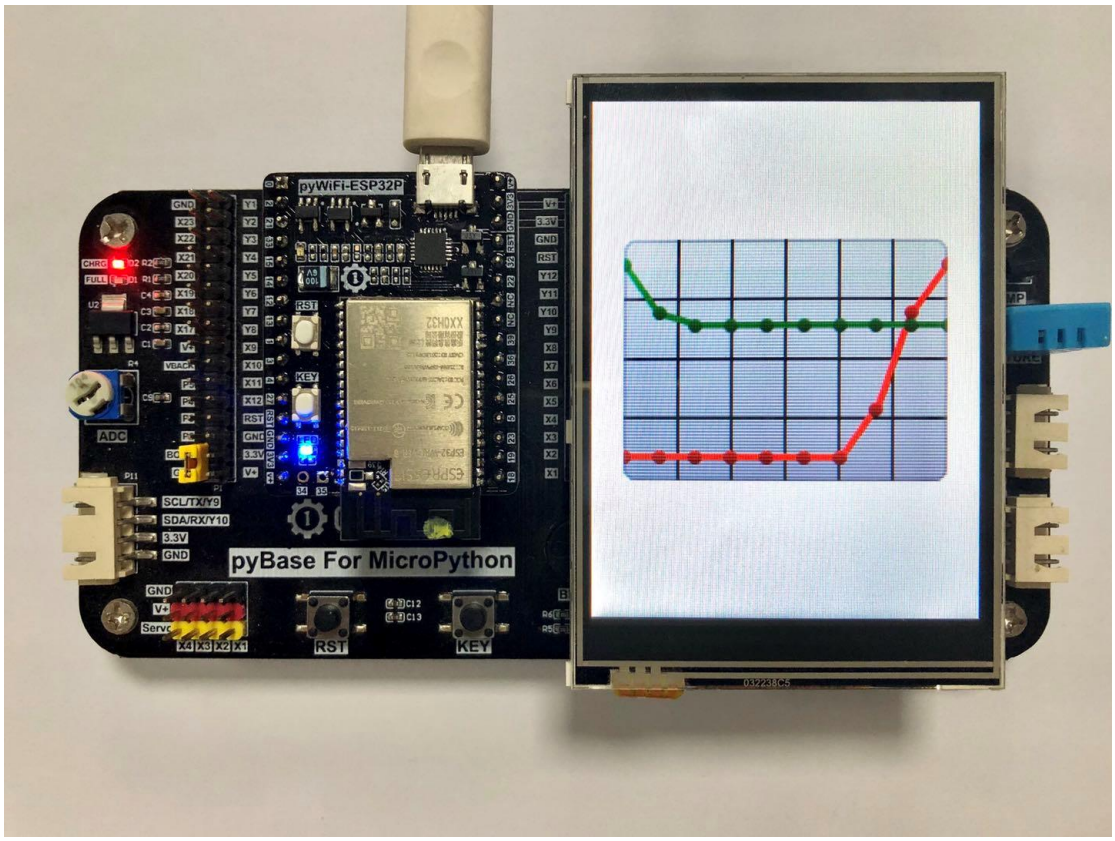


图 16-29 实验结果

- **总结**

本节主要讲解了 LVGL 中图表的编程使用方法。

16.3.8 Container（容器）

- **前言：**

  容器可以用于存放多个文本或者小部件。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 16-30　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建容器。

- **实验讲解：**

  我们来看看容器对象。

构造函数
`cont = lv.cont(lv.scr_act())`
构建容器对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`cont.set_auto_realign(True)`

自动调整尺寸
cont.align_origo()
设置对齐方式；
cont.set_fit(lv.FIT.TIGHT)
自动调整窗口大小；
cont.set_layout (lv.LAYOUT.COL_M)
自动调整容器内容布局；
更多用法请参阅小部件说明文档： （或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/cont.html

表 16-10 容器对象

编程思路如下：

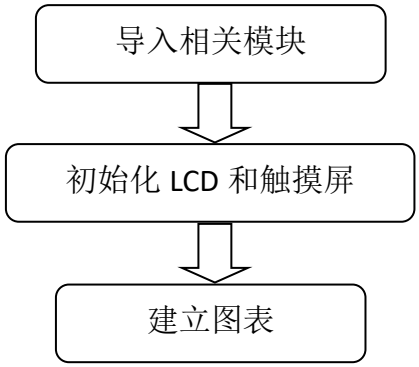


图 16-31 编程流程图

参考代码如下：

```
'''
实验名称：Container(容器)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
```

```
#############################################
###############   Cont   ###################
#############################################

if TOUCH_READY:

    cont = lv.cont(lv.scr_act())
    cont.set_auto_realign(True)   # Auto realign when the size changes
    cont.align_origo(None, lv.ALIGN.CENTER, 0, 0)
    # This parametrs will be sued when realigned

    cont.set_fit(lv.FIT.TIGHT) #自动调整窗口大小
    cont.set_layout(lv.LAYOUT.COL_M) #自动调整容器内容布局

    label = lv.label(cont)
    label.set_text("Short text")

    label = lv.label(cont)
    label.set_text("It is a long text")

    label = lv.label(cont)
    label.set_text("Here is an even longer text")
```

- **实验结果：**

  运行代码，可以看到 3 个 label 信息都位于容器框内。



图 16-32 实验结果

- **总结**

本节主要讲解了 LVGL 中容器的编程使用方法。

16.3.9 Drawdown List（下拉列表）

- **前言：**

  下拉列表可以提非常方便的多项选择功能。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 16-33　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建下拉列表。

- **实验讲解：**

  我们来看看下拉列表对象。

构造函数
`ddlist = lv.ddlist(lv.scr_act())`
构建下拉列表对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`ddlist.set_options()`

设置下拉列表的选项；
ddlist.set_draw_arrow(True)
设置下拉列表箭头显示。
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/dropdown.html

表 16-11 下拉列表对象

编程思路如下：

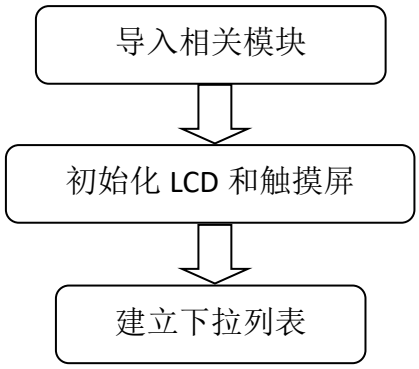


图 16-34 编程流程图

参考代码如下：

```
'''
实验名称：Drawdown List(下拉列表)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
################################################
###############   Drawdown List  ############
################################################
```

```
def event_handler(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        option = " "*10 # should be large enough to store the option
        obj.get_selected_str(option, len(option))
        # .strip() removes trailing spaces
        print("Option: \"%s\"" % option.strip())

if TOUCH_READY:

    # Create a drop down list
    ddlist = lv.ddlist(lv.scr_act())
    ddlist.set_options("\n".join([
                        "Apple",
                        "Banana",
                        "Orange",
                        "Melon",
                        "Grape",
                        "Raspberry"]))

    ddlist.set_fix_width(150)
    ddlist.set_draw_arrow(True)
    ddlist.align(None, lv.ALIGN.IN_TOP_MID, 0, 20)
    ddlist.set_event_cb(event_handler)
```

- **实验结果：**

  运行代码，可以看到 LCD 出现下拉列表。

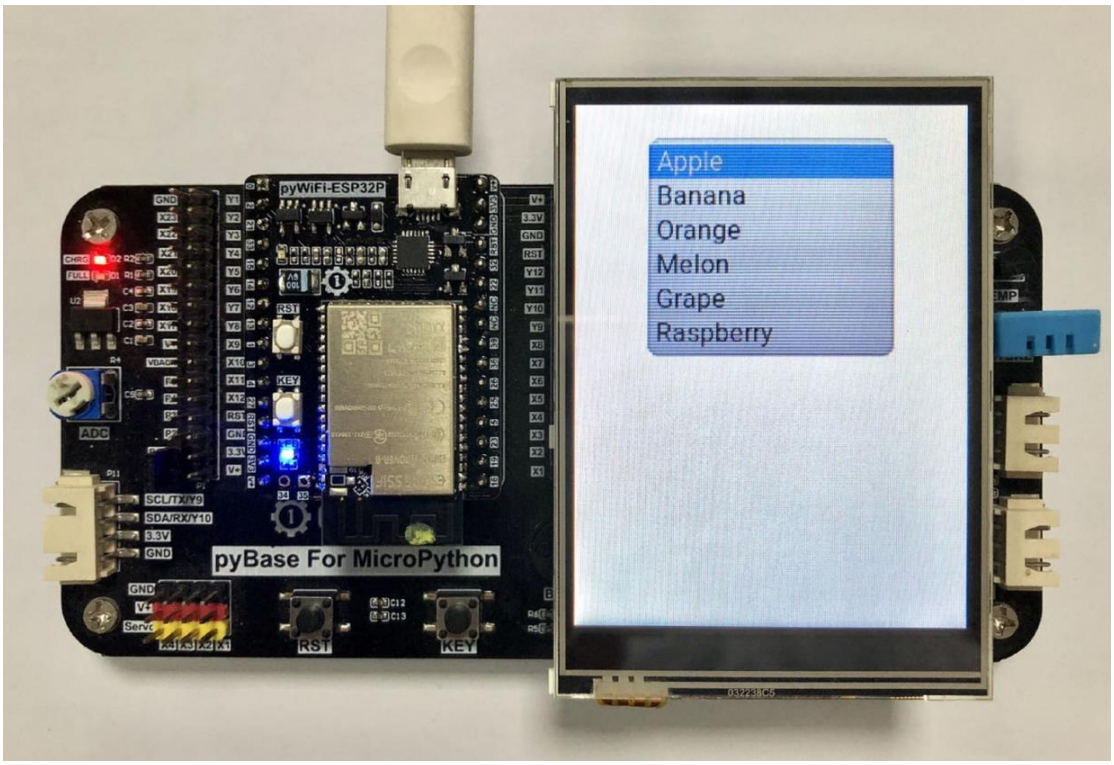


图 16-35 实验结果

- **总结**

本节主要讲解了 LVGL 中下拉列表的编程使用方法。

16.3.10 Gauge（计量器）

- **前言：**

  计量器可以用于仪器仪表盘数据展示。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

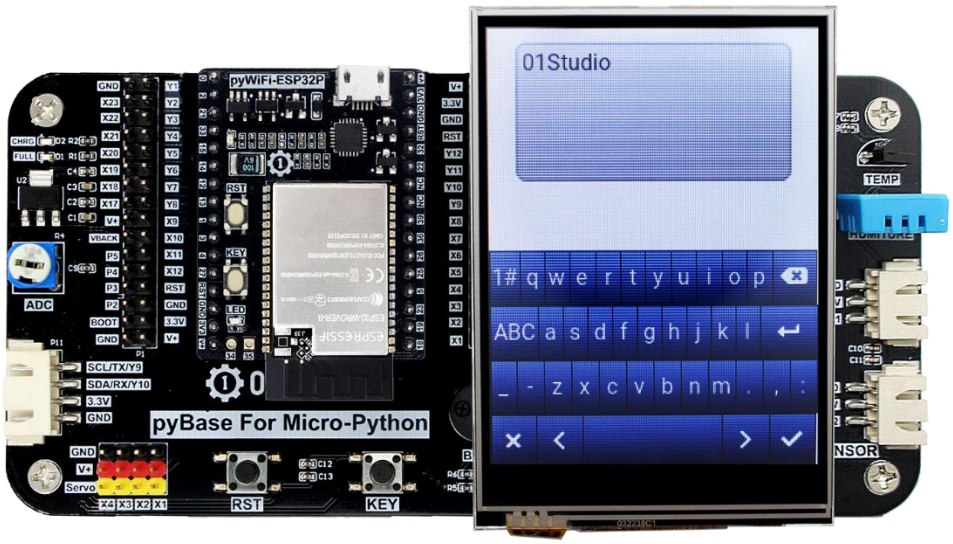


图 16-36　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建计量器。

- **实验讲解：**

  我们来看看计量器对象。

构造函数
gauge1 = lv.gauge(lv.scr_act())
构建计量器对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
gauge1.set_ style()

设置风格；
gauge1.set_needle_count()
设置指针数量和颜色；
gauge1.set_value(num,value)
设置指针值。 【num】指针编号； 【value】值。
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/gauge.html

表 16-12 计量器对象

编程思路如下：

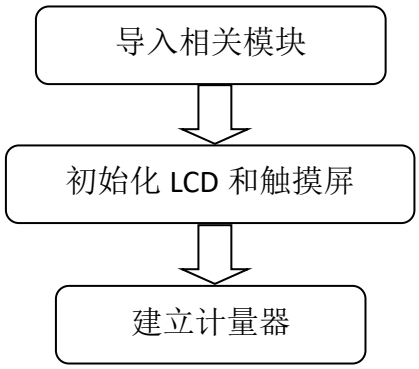


图 16-37 编程流程图

参考代码如下：

```
'''
实验名称：Gauge(计量器)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
```

```
#############################################
###############   Gauge   ###################
#############################################
if TOUCH_READY:
    # Create a style
    style = lv.style_t()
    lv.style_copy(style, lv.style_pretty_color)
   # Line color at the beginning
    style.body.main_color = lv.color_hex3(0x666)
   # Line color at the end
    style.body.grad_color =  lv.color_hex3(0x666)
    style.body.padding.left = 10         # Scale line length
    style.body.padding.inner = 8         # Scale label padding
    # Needle middle circle color
    style.body.border.color = lv.color_hex3(0x333)
    style.line.width = 3
    style.text.color = lv.color_hex3(0x333)
    # Line color after the critical value
    style.line.color = lv.color_hex3(0xF00)

    # Describe the color for the needles
    needle_colors = [
        lv.color_make(0x00, 0x00, 0xFF),
        lv.color_make(0xFF, 0xA5, 0x00),
        lv.color_make(0x80, 0x00, 0x80)
    ]

    # Create a gauge
    gauge1 = lv.gauge(lv.scr_act())
    gauge1.set_style(lv.gauge.STYLE.MAIN, style)
```

```
gauge1.set_needle_count(len(needle_colors), needle_colors)
gauge1.set_size(150, 150)
gauge1.align(None, lv.ALIGN.CENTER, 0, 20)

# Set the values
gauge1.set_value(0, 10)
gauge1.set_value(1, 20)
gauge1.set_value(2, 30)
```

- **实验结果：**

运行代码，可以看到计量器相关部件信息。

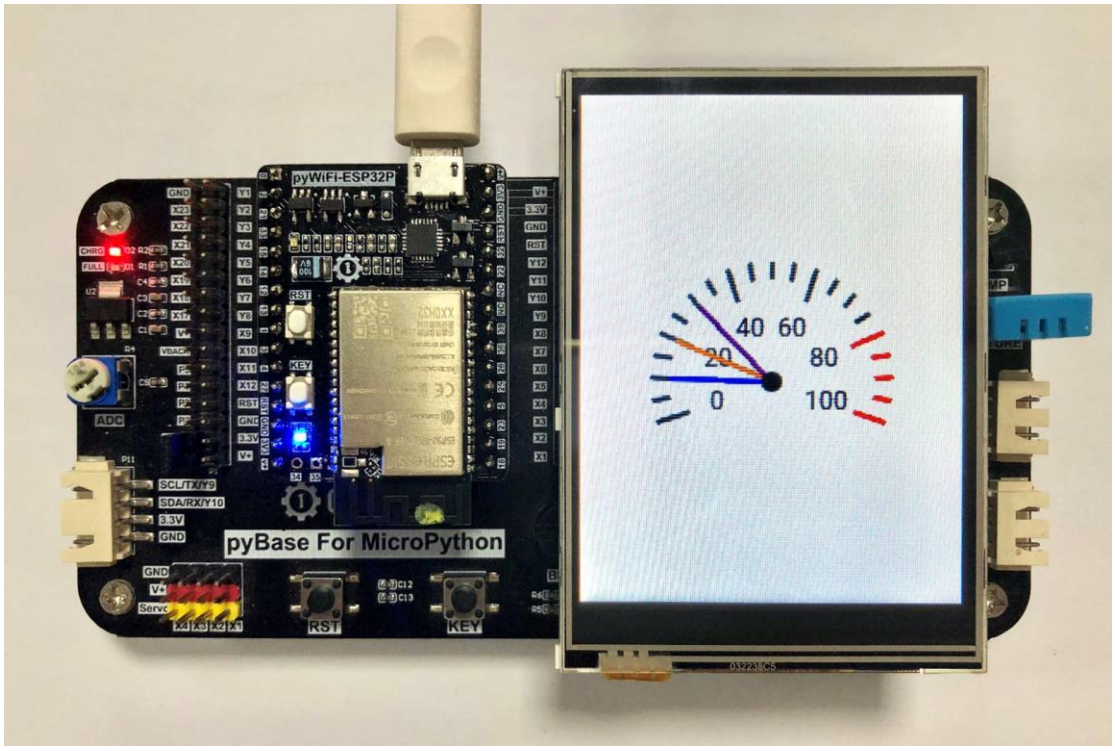


图 16-38 实验结果

- **总结**

本节主要讲解了 LVGL 中计量器使用方法。

16.3.11 Keyboard（键盘）

- **前言：**

  键盘是最常见的输入交互工具。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

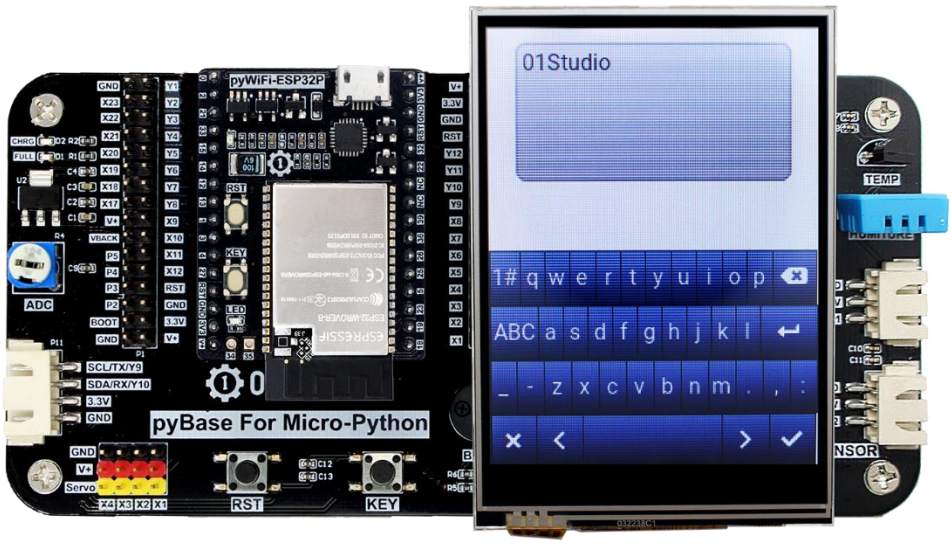


图 16-39　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建键盘。

- **实验讲解：**

  我们来看看键盘对象。

构造函数
`kb = lv.kb(lv.scr_act())`
构建键盘对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`kb.set_cursor_manage(True)`

设置光标闪烁；
kb.set_style()
设置键盘风格；
kb.set_ta(ta)
设置输入字符区域。 【ta】text area 字符区域对象；
更多用法请参阅小部件说明文档： （或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/keyboard.html

表 16-13 键盘对象

编程思路如下：

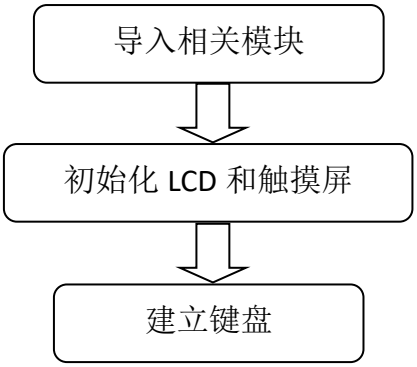


图 16-40 编程流程图

参考代码如下：

```
'''
实验名称：Keyboard(键盘)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
################################################
```

```
###############  Keyboard ###################
################################################
if TOUCH_READY:

    # Create styles for the keyboard
    rel_style = lv.style_t()
    pr_style  = lv.style_t()

    lv.style_copy(rel_style, lv.style_btn_rel)
    rel_style.body.radius = 0
    rel_style.body.border.width = 1

    lv.style_copy(pr_style, lv.style_btn_pr)
    pr_style.body.radius = 0
    pr_style.body.border.width = 1

    # Create a keyboard and apply the styles
    kb = lv.kb(lv.scr_act())
    kb.set_cursor_manage(True)
    kb.set_style(lv.kb.STYLE.BG, lv.style_transp_tight)
    kb.set_style(lv.kb.STYLE.BTN_REL, rel_style)
    kb.set_style(lv.kb.STYLE.BTN_PR, pr_style)

    # Create a text area. The keyboard will write here
    ta = lv.ta(lv.scr_act())
    ta.align(None, lv.ALIGN.IN_TOP_MID, 0, 10)
    ta.set_text("")

    # Assign the text area to the keyboard
    kb.set_ta(ta)
```

- **实验结果：**

运行代码，可以看到出现键盘部件，输入信息在文本框显示。

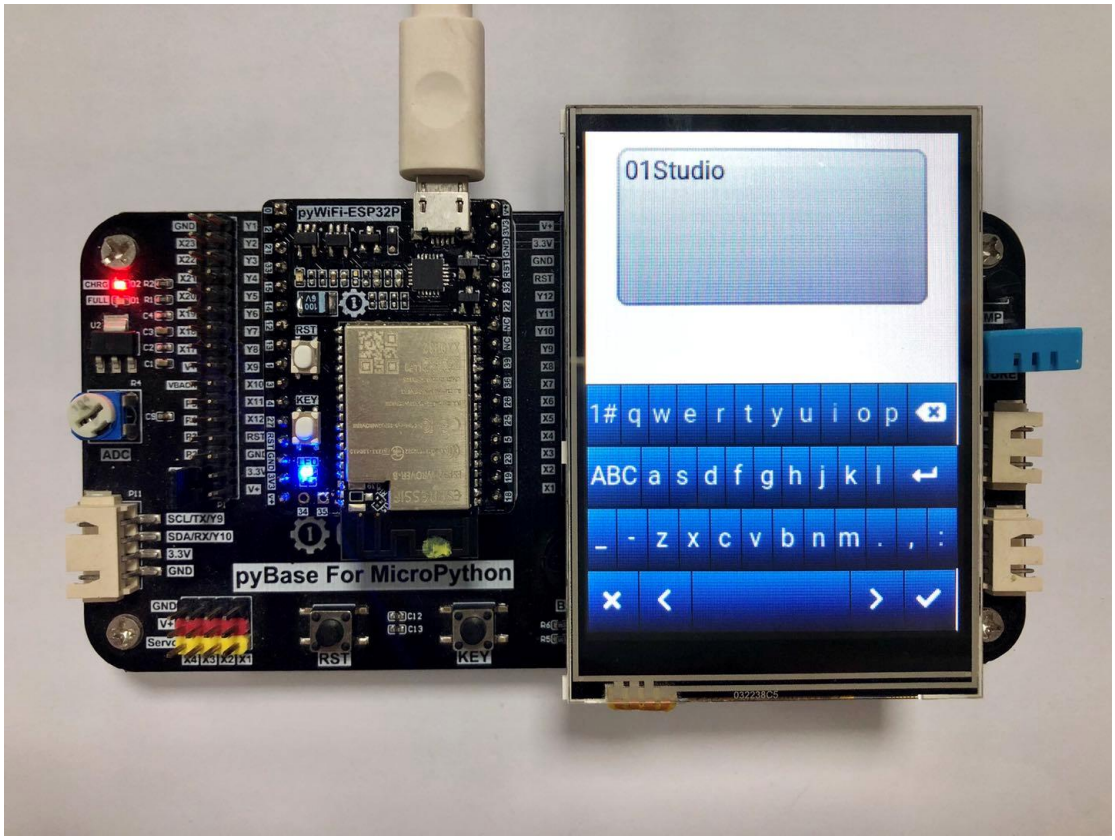


图 16-41 实验结果

- **总结**

本节主要讲解了 LVGL 中键盘使用方法。

16.3.12 Label（标签）

- **前言：**

  标签可以理解为简单的标题或文字。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

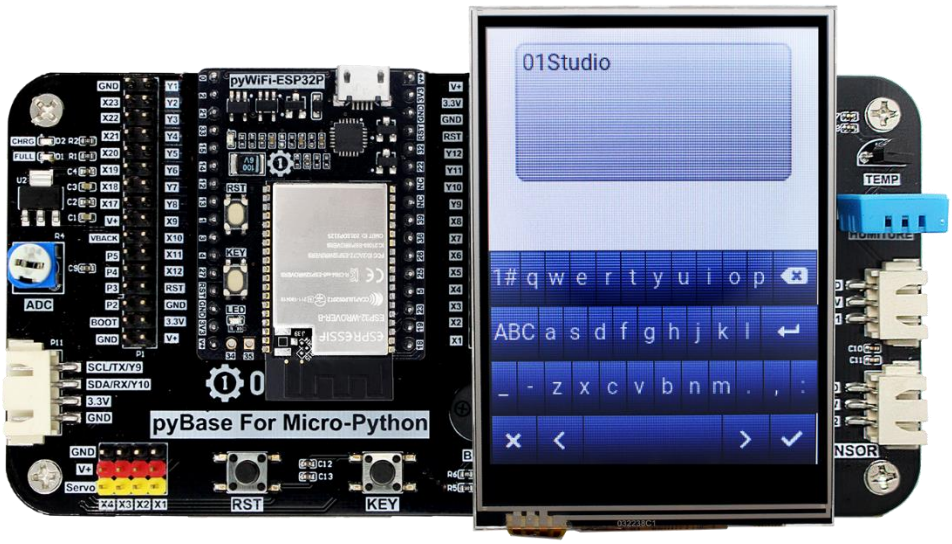


图 16-42 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建标签。

- **实验讲解：**

  我们来看看标签对象。

构造函数
`label = lv.label(lv.scr_act())`
构建标签对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`label.set_long_mode(mode)`

长文本模式； 【mode】 lv.label.LONG.BREAK：保持对象宽度，文本自动折断换行； lv.label.LONG.SROLL_CIRC：循环滚动标签；
label.set_recolor(True)
允许通过文本指令修改颜色；
label.set_text(str)
设置标签显示的文本。当允许文本指令修改字符颜色时，可通过下面方法来设置文本颜色，如：#0000ff Re-color# 表示将 Re-color 设置成蓝色并显示。
label.align()
设置对齐方式。
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/label.html

表 16-14 标签对象

编程思路如下：

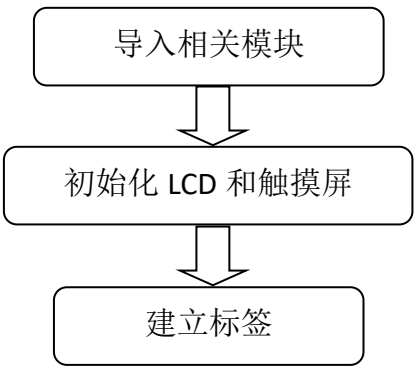


图 16-43 编程流程图

参考代码如下：

```
'''
实验名称：Label(标签)
版本：v1.0
```

```
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
################################################
###############  Label    ####################
################################################
if TOUCH_READY:

    label1 = lv.label(lv.scr_act())
    label1.set_long_mode(lv.label.LONG.BREAK) # Break the long lines
    # Enable re-coloring by commands in the text
    label1.set_recolor(True)
    label1.set_align(lv.label.ALIGN.CENTER)  # Center aligned lines
    label1.set_text("#0000ff Re-color# #00ff00 words# #ff0000 of a#
                      label " + "and  wrap long text automatically.")
    label1.set_width(150)
    label1.align(None, lv.ALIGN.CENTER, 0, -30)

    label2 = lv.label(lv.scr_act())
    label2.set_long_mode(lv.label.LONG.SROLL_CIRC)   # Circular scroll
    label2.set_width(150)
    label2.set_text("It is a circularly scrolling text. ")
    label2.align(None, lv.ALIGN.CENTER, 0, 30)
```

- **实验结果：**

  运行代码，可以看到 LCD 出现编程实现的标签小部件。

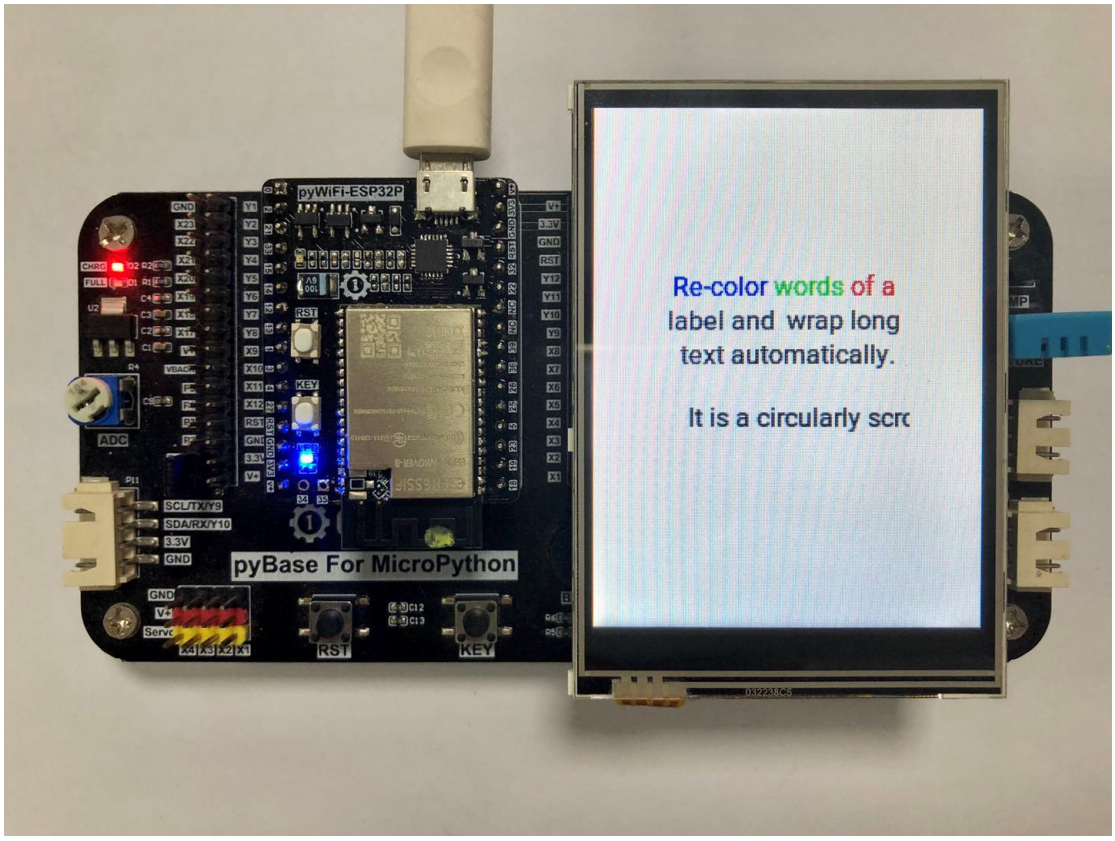


图 16-44 实验结果

- **总结**

本节主要讲解了 LVGL 中标签使用方法。

16.3.13 Line（线）

- **前言：**

  线部件可以通过定义各个点来绘制折线图。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

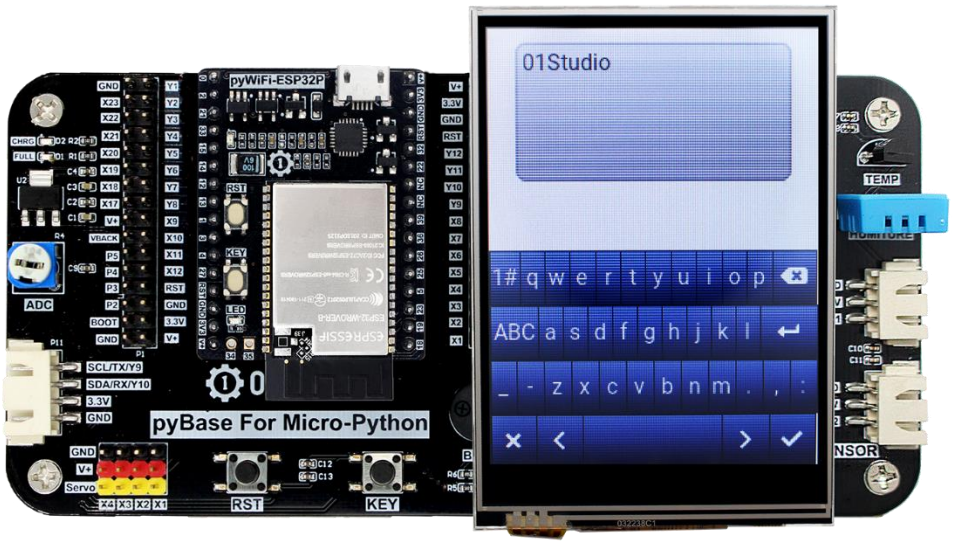


图 16-45　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建线部件。

- **实验讲解：**

  我们来看看线对象。

构造函数
`line = lv.line(lv.scr_act())`
构建线对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`line.set_points(line_points,len(line_points))`

设置线段 【line_points】设置线经过的点，例如经过 2 个点： line_points=[{"x":5, "y":5}, {"x":10, "y":10}] 【len(line_points)】对应点的数量。
line.set_style()
设置风格；
line.align()
设置对齐方式；
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/line.html

表 16-15 线对象

编程思路如下：

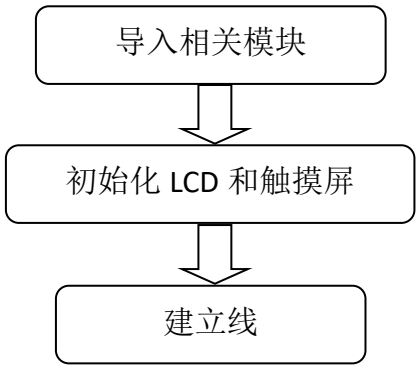


图 16-46 编程流程图

参考代码如下：

```
'''
实验名称：Line(线)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
```

```
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
#############################################
###############  Line    ###################
#############################################
if TOUCH_READY:

    # Create an array for the points of the line
    line_points = [ {"x":5, "y":5},
                    {"x":70, "y":70},
                    {"x":120, "y":10},
                    {"x":180, "y":60},
                    {"x":240, "y":10}]

    # Create new style (thick dark blue)
    style_line = lv.style_t()
    lv.style_copy(style_line, lv.style_plain)
    style_line.line.color = lv.color_make(0x00, 0x3b, 0x75)
    style_line.line.width = 3
    style_line.line.rounded = 1

    # Copy the previous line and apply the new style
    line1 = lv.line(lv.scr_act())
    line1.set_points(line_points, len(line_points))  # Set the points
    line1.set_style(lv.line.STYLE.MAIN, style_line)
    line1.align(None, lv.ALIGN.CENTER, 0, 0)
```

- **实验结果：**

  运行代码，可以看到 LCD 出现线小部件。

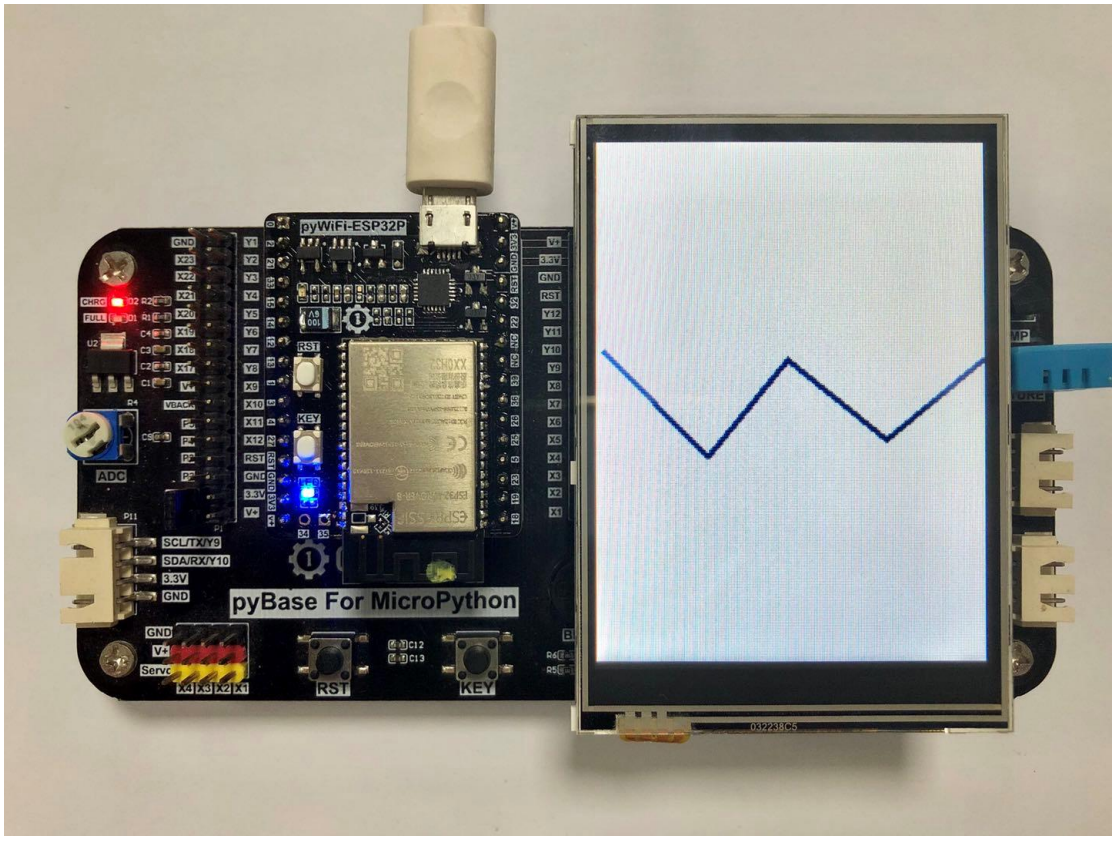


图 16-47 实验结果

- **总结**

本节主要讲解了 LVGL 中线的使用方法。

16.3.14 List（菜单）

- **前言：**

  菜单可以完成方便快捷的功能选择。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 16-48　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建菜单。

- **实验讲解：**

  我们来看看菜单对象。

构造函数
`list = lv.list(lv.scr_act())`
构建菜单对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`list.set_size(width, height)`

设置菜单尺寸： 【width】宽； 【height】高。
list.aline()
设置对齐方式；
list_btn = list.add_btn()
为菜单增加按钮
list_btn.set_event_cb(callback)
新增按钮按下的回调函数
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/list.html

表 16-16 菜单对象

编程思路如下：

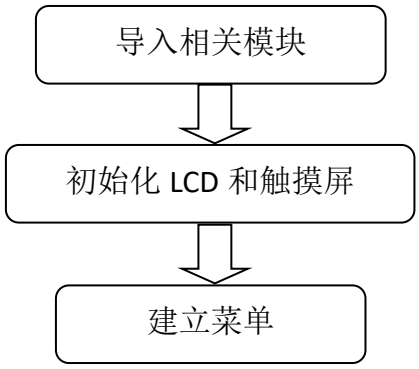


图 16-49 编程流程图

参考代码如下：

```
'''
实验名称：List(菜单)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
```

```
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
################################################
###############  List    ####################
################################################
def event_handler(obj, event):
    if event == lv.EVENT.CLICKED:
        print("Clicked: %s" % lv.list.get_btn_text(obj))

if TOUCH_READY:
    # Create a list
    list1 = lv.list(lv.scr_act())
    list1.set_size(160, 200)
    list1.align(None, lv.ALIGN.CENTER, 0, 0)

    # Add buttons to the list
    list_btn = list1.add_btn(lv.SYMBOL.FILE, "New")
    list_btn.set_event_cb(event_handler)

    list_btn = list1.add_btn(lv.SYMBOL.DIRECTORY, "Open")
    list_btn.set_event_cb(event_handler)

    list_btn = list1.add_btn(lv.SYMBOL.CLOSE, "Delete")
    list_btn.set_event_cb(event_handler)

    list_btn = list1.add_btn(lv.SYMBOL.EDIT, "Edit")
    list_btn.set_event_cb(event_handler)

    list_btn = list1.add_btn(lv.SYMBOL.SAVE, "Save")
list_btn.set_event_cb(event_handler)
```

- **实验结果：**

运行代码，可以看到 LCD 出现菜单。

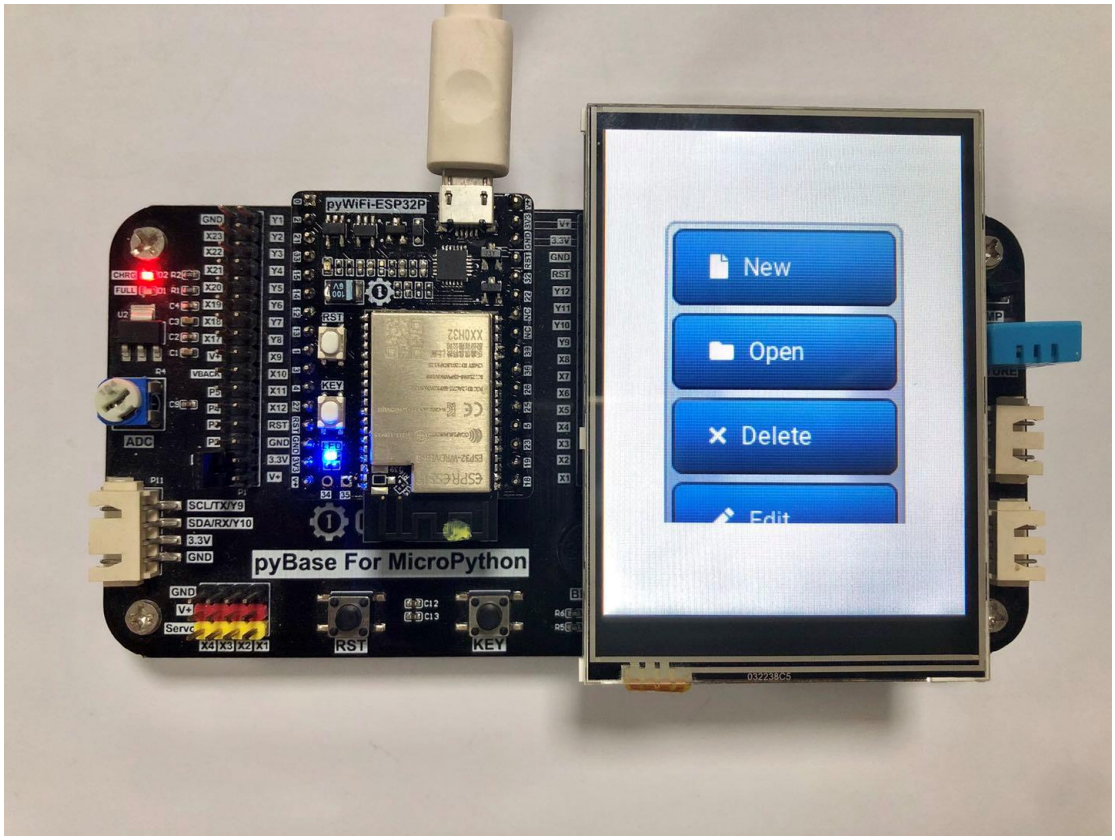


图 16-50 实验结果

- **总结**

本节主要讲解了 LVGL 中菜单的编程和使用方法。

16.3.15 Message box（消息框）

- **前言：**

  消息框可以用于弹出信息或者选择框。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

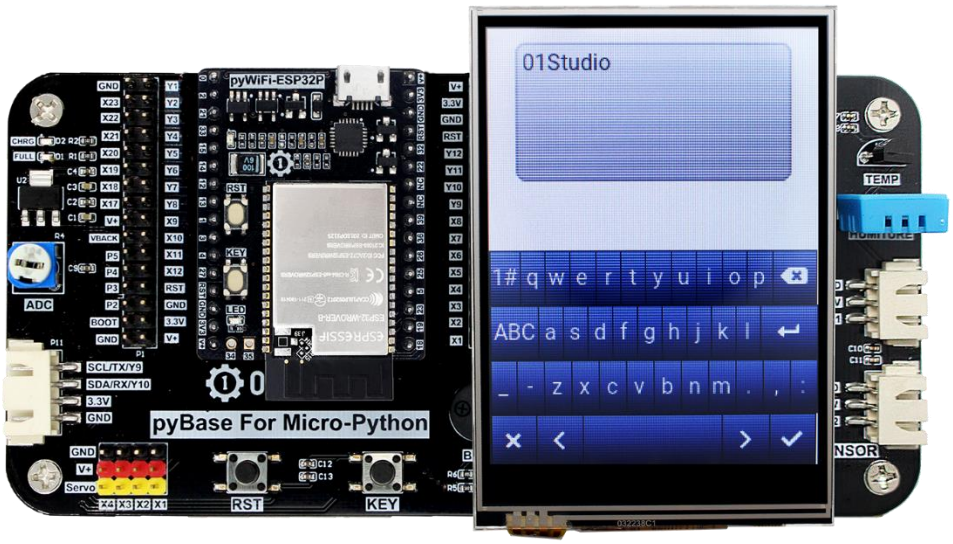


图 16-51 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建消息框。

- **实验讲解：**

  我们来看看消息框对象。

构造函数
`mbox = lv.mbox(lv.scr_act())`
构建菜单对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`mbox.set_text(str)`

设置消息框显示文本；
mbox.add_btns()
为消息框增加按钮；
mbox = list.add_btn()
为菜单增加按钮
mbox.set_event_cb(callback)
新增按钮按下的回调函数
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/msgbox.html

表 16-17 消息框对象

编程思路如下：

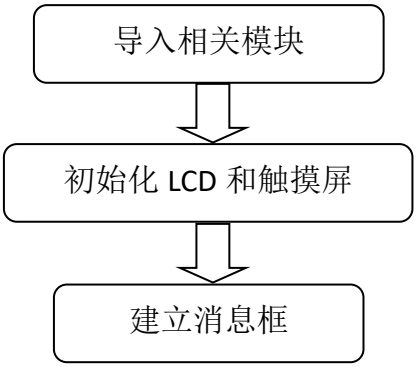


图 16-52 编程流程图

参考代码如下：

```
'''
实验名称：Message box(消息框)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
```

```
################################################
###############  Message box ###############
################################################
def event_handler(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        print("Button: %s" % lv.mbox.get_active_btn_text(obj))


if TOUCH_READY:


    btns = ["Apply", "Close", ""]


    mbox1 = lv.mbox(lv.scr_act())
    mbox1.set_text("A message box with two buttons.");
    mbox1.add_btns(btns)
    mbox1.set_width(200)
    mbox1.set_event_cb(event_handler)
    mbox1.align(None, lv.ALIGN.CENTER, 0, 0)  # Align to the corner
```

- **实验结果：**

  运行代码，可以看到出现消息框和里面的 2 个按钮。

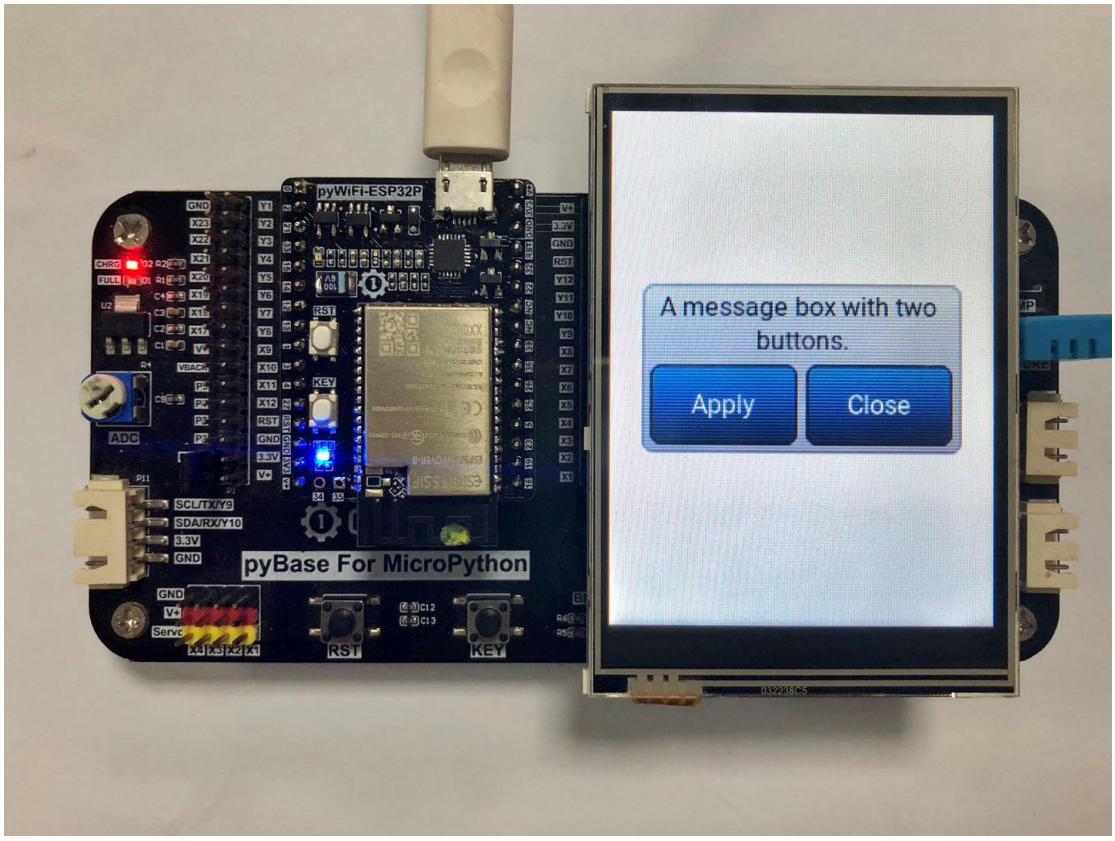


图 16-53 实验结果

- **总结**

本节主要讲解了 LVGL 中消息框编程和使用方法。

16.3.16 Page（页）

- **前言：**

  页相当于文本阅读器，可以存放大量文字信息(label)。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 16-54　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建页。

- **实验讲解：**

  我们来看看页对象。

构造函数
`page = lv.page(lv.scr_act())`
构建菜单对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`page.set_size(width, height)`

设置页的尺寸； 【width】宽度； 【height】高度。
page.align()
对齐方式；
label = lv.label(page)
在页中加入 label
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/page.html

表 16-18 页对象

编程思路如下：

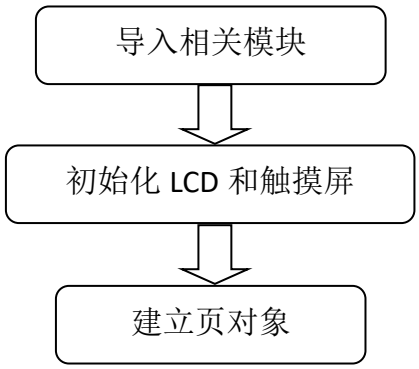


图 16-55 编程流程图

参考代码如下：

```
'''
实验名称：Page(页)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
```

```
#############################################
##################  Page ####################
#############################################
if TOUCH_READY:
    #Create a scroll bar style
    style_sb = lv.style_t()
    lv.style_copy(style_sb, lv.style_plain)
    style_sb.body.main_color = lv.color_make(0,0,0)
    style_sb.body.grad_color = lv.color_make(0,0,0)
    style_sb.body.border.color = lv.color_make(0xff,0xff,0xff)
    style_sb.body.border.width = 1
    style_sb.body.border.opa = lv.OPA._70
    style_sb.body.radius = 800 # large enough to make a circle
    style_sb.body.opa = lv.OPA._60
    style_sb.body.padding.right = 3
    style_sb.body.padding.bottom = 3
    style_sb.body.padding.inner = 8       # Scrollbar width

    # Create a page
    page = lv.page(lv.scr_act())
    page.set_size(240, 300)
    page.align(None, lv.ALIGN.CENTER, 0, 0)
    page.set_style(lv.page.STYLE.SB, style_sb) #Set the scrollbar style

    # Create a label on the page
    label = lv.label(page)
   #Automatically break long lines
    label.set_long_mode(lv.label.LONG.BREAK)
   # Set the label width to max value to not show hor. scroll bars
    label.set_width(page.get_fit_width())
```

```
label.set_text("""Lorem ipsum dolor sit amet, consectetur adipiscing
elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure
dolor in reprehenderit in voluptate velit esse cillum dolore
eu fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident, sunt in culpa
qui officia deserunt mollit anim id est laborum.""")
```

- **实验结果：**

运行代码，可以看到出现页面以及文字。

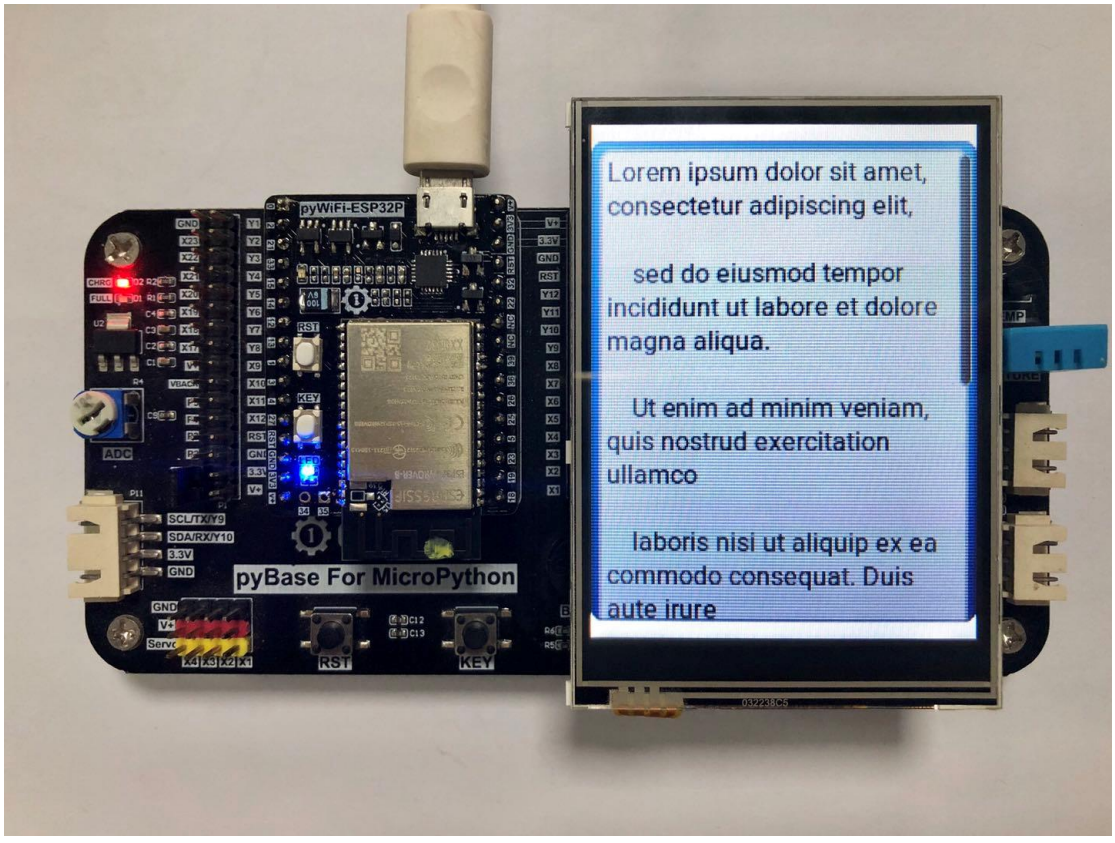


图 16-56 实验结果

- **总结**

本节主要讲解了 LVGL 中页面的使用方法。

16.3.17 Roller（滚筒）

- **前言：**

  滚筒可以用于多个选项的快速选择，例如时间和日期。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 16-57 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建滚筒。

- **实验讲解：**

  我们来看看滚筒对象。

构造函数
`roller = lv.roller(lv.scr_act())`
构建滚筒对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`roller.set_options()`

设置选项；设置方法参考例程代码。
roller.set_visible_row_count(num)
滚筒窗口可视数量个数； 【num】可视选项数量。
roller.align()
对齐方式；
roller.set_event_cb(callback)
设置回调函数；
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/roller.html

表 16-19 滚筒对象

编程思路如下：

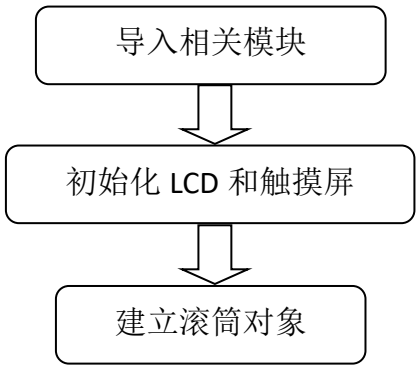


图 16-58 编程流程图

参考代码如下：

```
'''
实验名称：Roller(滚筒)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
```

```
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
################################################
##################  Roller ##################
################################################
def event_handler(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        option = " "*10
        obj.get_selected_str(option, len(option))
        print("Selected month: %s" % option.strip())



if TOUCH_READY:

    roller1 = lv.roller(lv.scr_act())
    roller1.set_options("\n".join([
                        "January",
                        "February",
                        "March",
                        "April",
                        "May",
                        "June",
                        "July",
                        "August",
                        "September",
                        "October",
                        "November",
                        "December"]), lv.roller.MODE.INIFINITE)

    roller1.set_visible_row_count(4)
    roller1.align(None, lv.ALIGN.CENTER, 0, 0)
```

```
roller1.set_event_cb(event_handler)
```

- **实验结果：**

运行代码，可以看到出现滚筒和对应的选项。

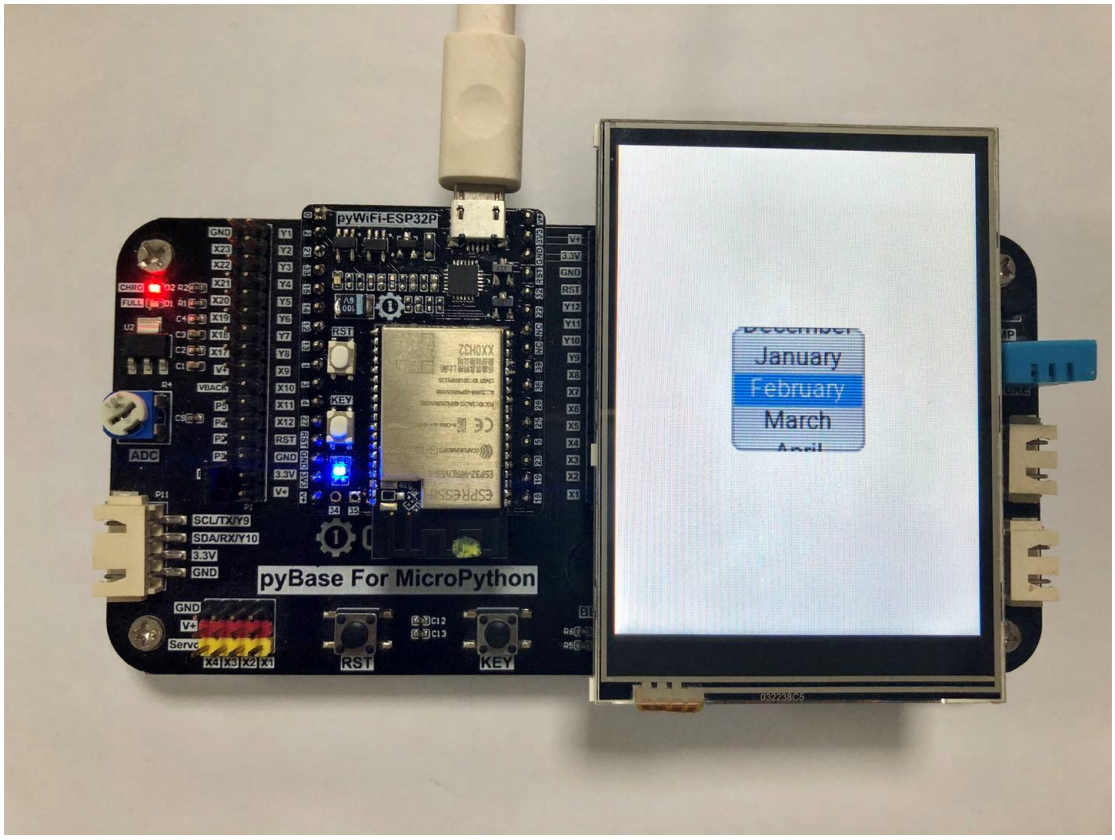


图 16-59 实验结果

- **总结**

本节主要讲解了 LVGL 中滚筒的使用方法。

16.3.18 Slider（滑动条）

- **前言：**

  滑动条可以用于进度的选择。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

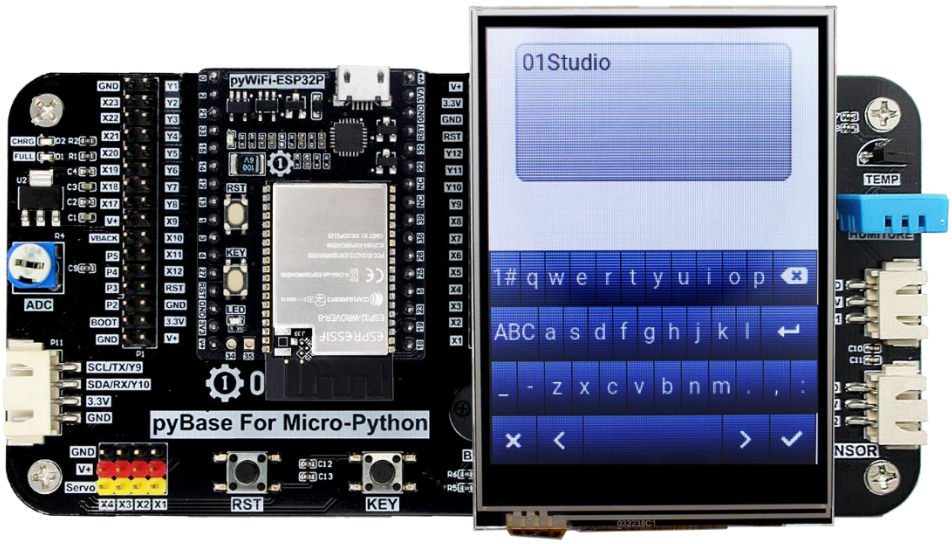


图 16-60　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建滑动条。

- **实验讲解：**

  我们来看看滑动条对象。

构造函数
`slider = lv.slider(lv.scr_act())`
构建滑动条对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`slider.set_width(200)`

设置宽度。
slider.set_range(0,100)
设置滑动条最大值和最小值；(0,100)表示从 0 到 100。
slider.align()
对齐方式；
slider.set_event_cb(callback)
设置回调函数；
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/slider.html

表 16-20 滑动条对象

编程思路如下：

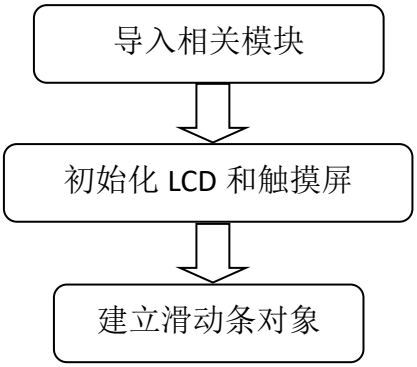


图 16-61 编程流程图

参考代码如下：

```
'''
实验名称：Slider(滑动条)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
```

```
#############################################
##################  Slider ##################
#############################################
def slider_event_cb(slider, event):
    if event == lv.EVENT.VALUE_CHANGED:
        slider_label.set_text("%u" % slider.get_value())

if TOUCH_READY:

    # Create a slider in the center of the display
    slider = lv.slider(lv.scr_act())
    slider.set_width(200)
    slider.align(None, lv.ALIGN.CENTER, 0, 0)
    slider.set_event_cb(slider_event_cb)
    slider.set_range(0, 100)

    # Create a label below the slider
    slider_label = lv.label(lv.scr_act())
    slider_label.set_text("0")
    slider_label.set_auto_realign(True)
    slider_label.align(slider, lv.ALIGN.OUT_BOTTOM_MID, 0, 10)
```

- **实验结果：**

  运行代码，可以看 LCD 出现滑动条，拖动滑动条，数字发生变化。

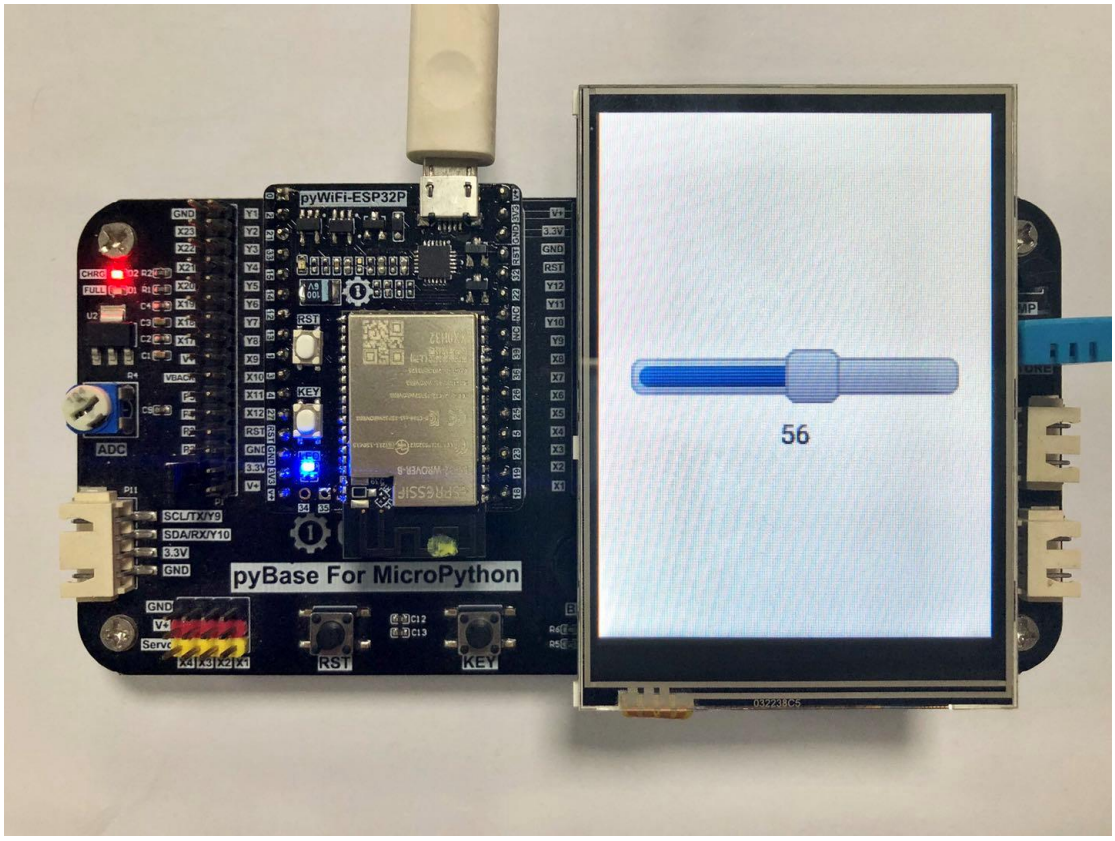


图 16-62 实验结果

- **总结**

本节主要讲解了 LVGL 中滚滑动条的使用方法。

16.3.19 Spinner（旋转加载条）

- **前言：**

旋转加载条用于等待加载指示。

- **实验平台：**

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

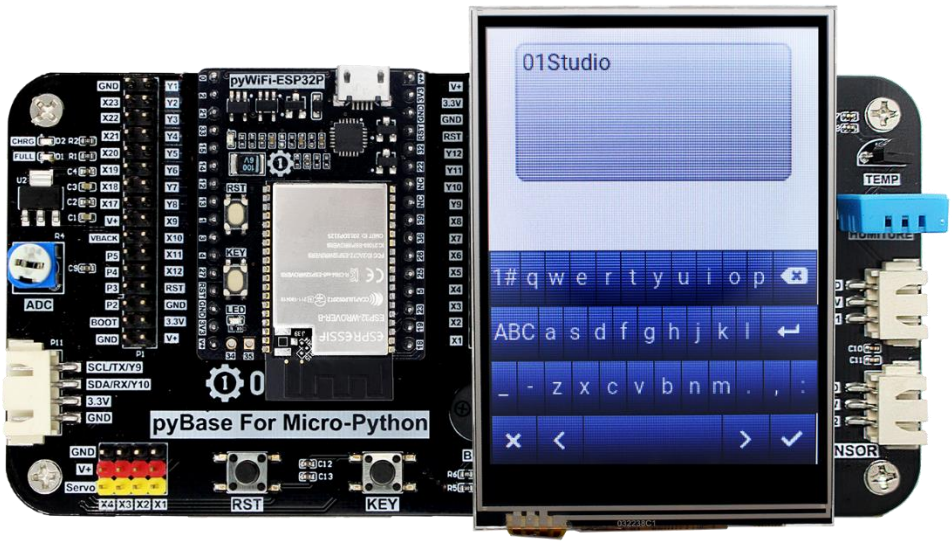


图 16-63 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

创建旋转加载条。

- **实验讲解：**

我们来看看旋转加载条对象。

构造函数
`preload = lv.preload(lv.scr_act())`
构建旋转加载条对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`preload.set_size(width,height)`

设置尺寸。 【width】宽度； 【height】高度。
preload.align()
对齐方式；
preload.set_style()
设置风格；
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/spinner.html

表 16-21 旋转加载条对象

编程思路如下：

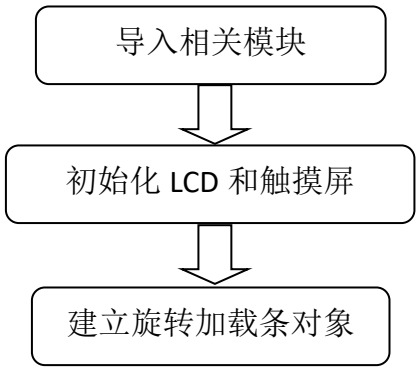


图 16-64 编程流程图

参考代码如下：

```
'''
实验名称：Spinner(旋转加载条)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
```

```
#############################################
##################  Spinner #################
#############################################
if TOUCH_READY:

    # Create a style for the Preloader
    style = lv.style_t()
    lv.style_copy(style, lv.style_plain)
    style.line.width = 10                       # 10 px thick arc
    style.line.color = lv.color_hex3(0x258)    # Blueish arc color

   # Gray background color
    style.body.border.color = lv.color_hex3(0xBBB)
    style.body.border.width = 10
    style.body.padding.left = 0

    # Create a Preloader object
    preload = lv.preload(lv.scr_act())
    preload.set_size(100, 100)
    preload.align(None, lv.ALIGN.CENTER, 0, 0)
    preload.set_style(lv.preload.STYLE.MAIN, style)
```

- **实验结果：**

  运行代码，可以看 LCD 出现旋转加载条。

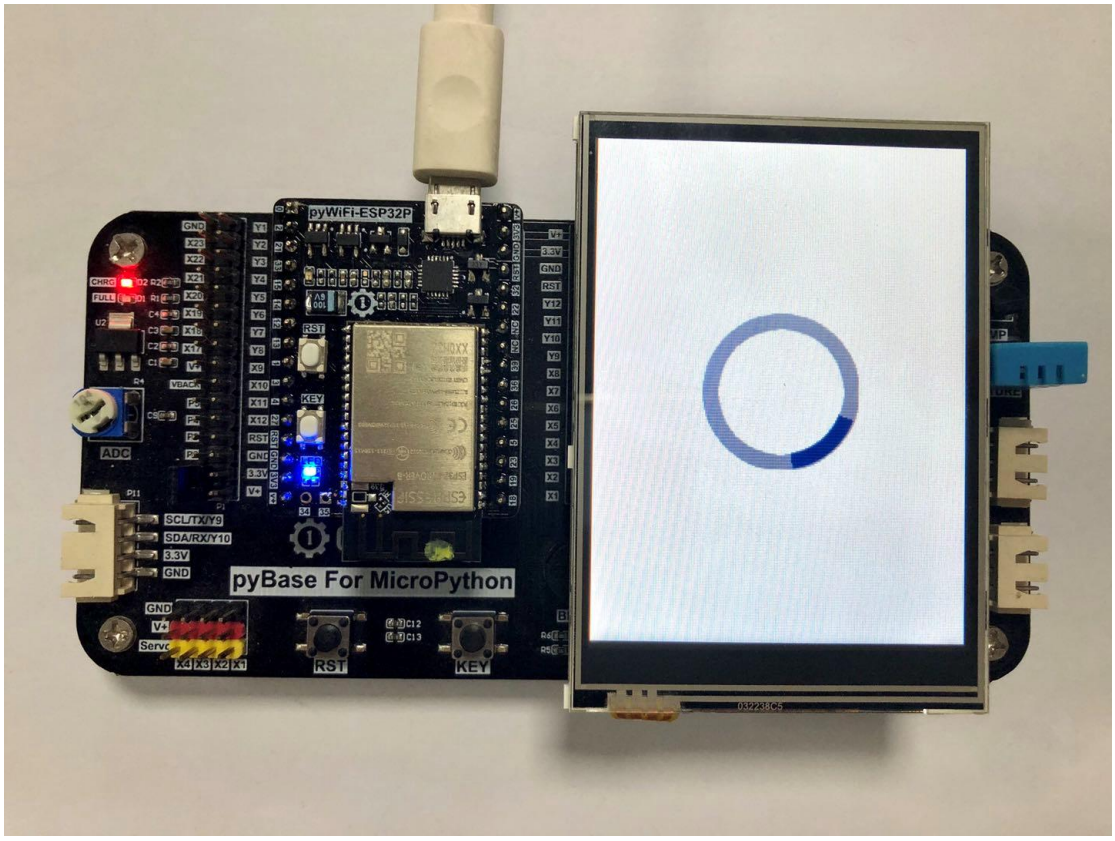


图 16-65 实验结果

- **总结**

本节主要讲解了 LVGL 中旋转加载条的使用方法。

16.3.20 Swtich（开关）

- **前言：**

  开关是非常常见和常用的小部件。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

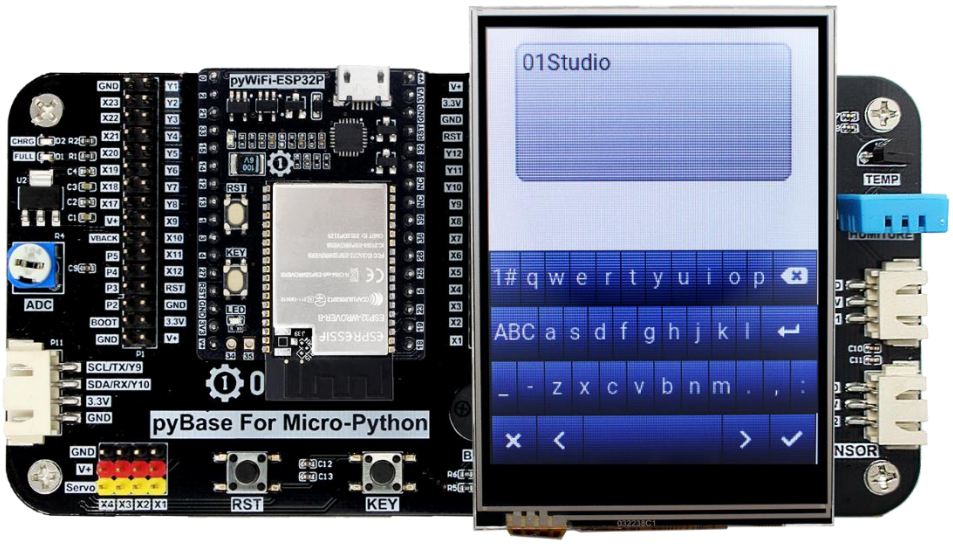


图 16-66 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建开关。

- **实验讲解：**

  我们来看看开关对象。

构造函数
`sw = lv.sw(lv.scr_act())`
构建开关对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`sw.set_style()`

设置风格。
sw.align()
对齐方式；
sw. set_event_cb(callback)
设置开关回调函数；
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/switch.html

表 16-22 开关对象

编程思路如下：

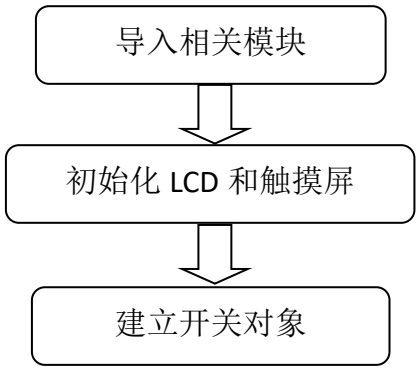


图 16-67 编程流程图

参考代码如下：

```
'''
实验名称：Switch(开关)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
################################################
##################  Switch  #################
################################################
```

```
def event_handler(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        print("State: %s" % ("On" if obj.get_state() else "Off"))

if TOUCH_READY:

    # Create styles for the switch
    bg_style = lv.style_t()
    indic_style = lv.style_t()
    knob_on_style = lv.style_t()
    knob_off_style = lv.style_t()

    lv.style_copy(bg_style, lv.style_pretty)
    bg_style.body.radius = 800
    bg_style.body.padding.top = 6
    bg_style.body.padding.bottom = 6

    lv.style_copy(indic_style, lv.style_pretty_color)
    indic_style.body.radius = 800
    indic_style.body.main_color = lv.color_hex(0x9fc8ef)
    indic_style.body.grad_color = lv.color_hex(0x9fc8ef)
    indic_style.body.padding.left = 0
    indic_style.body.padding.right = 0
    indic_style.body.padding.top = 0
    indic_style.body.padding.bottom = 0

    lv.style_copy(knob_off_style, lv.style_pretty)
    knob_off_style.body.radius = 800
    knob_off_style.body.shadow.width = 4
    knob_off_style.body.shadow.type = lv.SHADOW.BOTTOM
```

```
lv.style_copy(knob_on_style, lv.style_pretty_color)
knob_on_style.body.radius = 800
knob_on_style.body.shadow.width = 4
knob_on_style.body.shadow.type = lv.SHADOW.BOTTOM

# Create a switch and apply the styles
sw1 = lv.sw(lv.scr_act())
sw1.set_style(lv.sw.STYLE.BG, bg_style)
sw1.set_style(lv.sw.STYLE.INDIC, indic_style)
sw1.set_style(lv.sw.STYLE.KNOB_ON, knob_on_style)
sw1.set_style(lv.sw.STYLE.KNOB_OFF, knob_off_style)
sw1.align(None, lv.ALIGN.CENTER, 0, -50)
sw1.set_event_cb(event_handler)

# Copy the first switch and turn it ON
sw2 = lv.sw(lv.scr_act(), sw1)
sw2.on(lv.ANIM.ON)
sw2.align(None, lv.ALIGN.CENTER, 0, 50)
sw2.set_event_cb(lambda o,e: None)
```

- **实验结果：**

运行代码，可以看出现 2 个开关，打开和关闭上面的开关，可以在 REPL 看到状态变化提示信息。

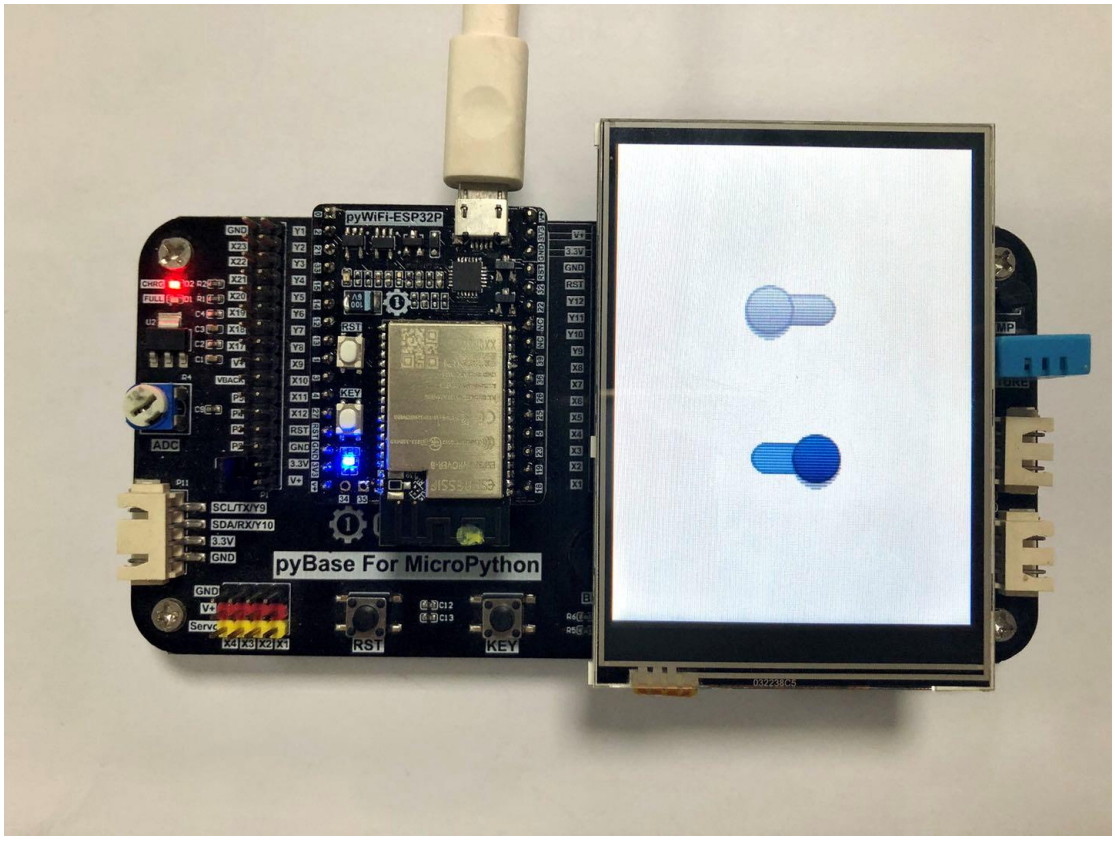


图 16-68 实验结果

```
ESP MicroPython REPL
Enable backlight
Double buffer
>>>
MicroPython v1.9.4-2072-gf89c8c48c-dirty on 2020-01-17; ESP32 module (spiram) with ESP32
Type "help()" for more information.
>>> State: On
State: Off
```

图 16-69 REPL 提示

- **总结**

本节主要讲解了 LVGL 中开关的使用方法。

16.3.21 Table（表格）

- **前言：**

表格是非常常用的小部件。

- **实验平台：**

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

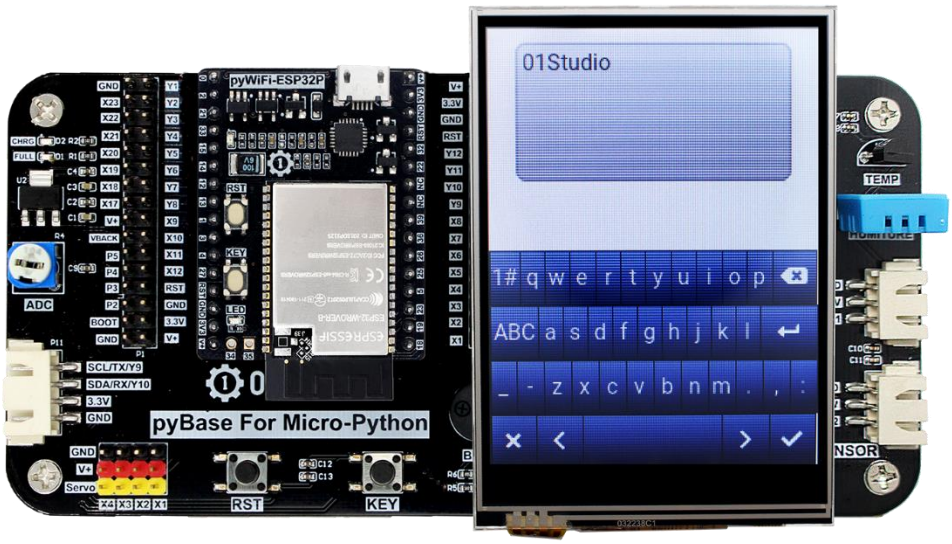


图 16-70 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

创建表格。

- **实验讲解：**

我们来看看表格对象。

构造函数
`table = lv.table(lv.scr_act())`
构建表格对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`table.set_style()`

设置风格。
table.set_col_cnt(num)
设置列数；
table.set_row_cnt(num)
设置行数；
table.align()
表格对齐方式；
table.set_cell_align(row, col, lv.label.ALIGN.CENTER)
设置某各单元格对齐方式； 【row】单元格行； 【col】单元格列； 【lv.label.ALIGN.CENTER】居中对齐。
table.set_cell_value(row, col, str)
设置某个单元格值； 【row】单元格行； 【col】单元格列； 【str】单元格值；
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/table.html

表 16-23 表格对象

编程思路如下：

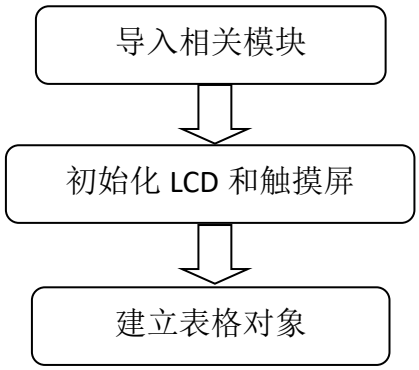


图 16-71 编程流程图

参考代码如下：

```
'''
实验名称：Table(表格)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
################################################
##################  Table  ##################
################################################
if TOUCH_READY:

    # Create a normal cell style
    style_cell1 = lv.style_t()
    lv.style_copy(style_cell1, lv.style_plain)
    style_cell1.body.border.width = 1
    style_cell1.body.border.color = lv.color_make(0,0,0)

    # Crealte a header cell style
    style_cell2 = lv.style_t()
    lv.style_copy(style_cell2, lv.style_plain)
    style_cell2.body.border.width = 1
    style_cell2.body.border.color = lv.color_make(0,0,0)
    style_cell2.body.main_color = lv.color_make(0xC0, 0xC0, 0xC0)
    style_cell2.body.grad_color = lv.color_make(0xC0, 0xC0, 0xC0)

    table = lv.table(lv.scr_act())
    table.set_style(lv.table.STYLE.CELL1, style_cell1)
    table.set_style(lv.table.STYLE.CELL2, style_cell2)
```

```
table.set_style(lv.table.STYLE.BG, lv.style_transp_tight)
table.set_col_cnt(2)
table.set_row_cnt(4)
table.align(None, lv.ALIGN.CENTER, 0, 0)

# Make the cells of the first row center aligned
table.set_cell_align(0, 0, lv.label.ALIGN.CENTER)
table.set_cell_align(0, 1, lv.label.ALIGN.CENTER)

# Make the cells of the first row TYPE = 2 (use `style_cell2`)
table.set_cell_type(0, 0, 2)
table.set_cell_type(0, 1, 2)

# Fill the first column
table.set_cell_value(0, 0, "Name")
table.set_cell_value(1, 0, "Apple")
table.set_cell_value(2, 0, "Banana")
table.set_cell_value(3, 0, "Citron")

# Fill the second column
table.set_cell_value(0, 1, "Price")
table.set_cell_value(1, 1, "$7")
table.set_cell_value(2, 1, "$4")
table.set_cell_value(3, 1, "$6")
```

- **实验结果：**

  运行代码，可以看见编程实现的表格内容。

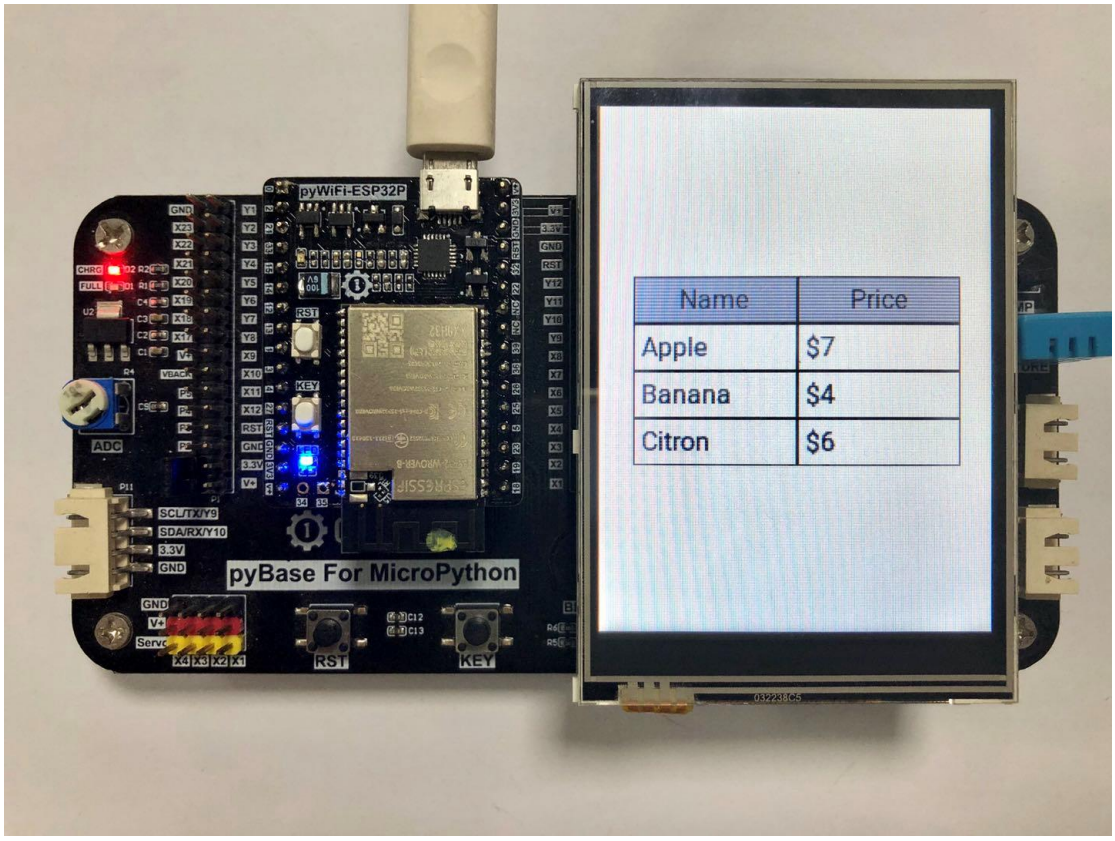


图 16-72 实验结果

- **总结**

本节主要讲解了 LVGL 中表格的使用方法。

16.3.22 Tabview（选项卡视图）

- **前言：**

选项卡视图可以用于切换不同的界面。

- **实验平台：**

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

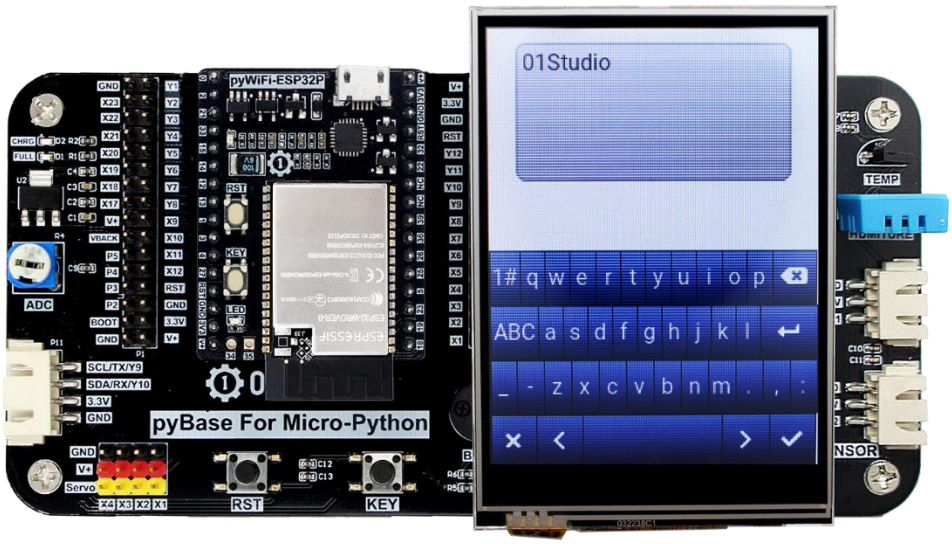


图 16-73 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

创建选项卡视图。

- **实验讲解：**

我们来看看选项卡视图对象。

构造函数
`tabview = lv.tabview(lv.scr_act())`
构建选项卡视图对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`tab1 = tabview.add_tab("Tab 1")`

增加一个选项卡
label = lv.label(tab1)
在选项卡加入标签（label）
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/tabview.html

表 16-24 选项卡视图对象

编程思路如下：

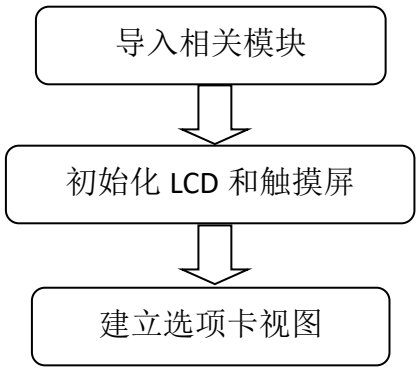


图 16-74 编程流程图

参考代码如下：

```
'''
实验名称：Tabview(选项卡视图)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
################################################
################## Tabview ##################
################################################
if TOUCH_READY:

    # Create a Tab view object
```

```
tabview = lv.tabview(lv.scr_act())


# Add 3 tabs (the tabs are page (lv_page) and can be scrolled
tab1 = tabview.add_tab("Tab 1")
tab2 = tabview.add_tab("Tab 2")
tab3 = tabview.add_tab("Tab 3")


# Add content to the tabs
label = lv.label(tab1)
label.set_text("""  This the first tab
If the content
of a tab
become too long
the it
automatically
become
scrollable.""")


label = lv.label(tab2)
label.set_text("Second tab")


label = lv.label(tab3)
label.set_text("Third tab")
```

- **实验结果：**

  运行代码，可以看见编程实现的 3 个选项卡视图内容。

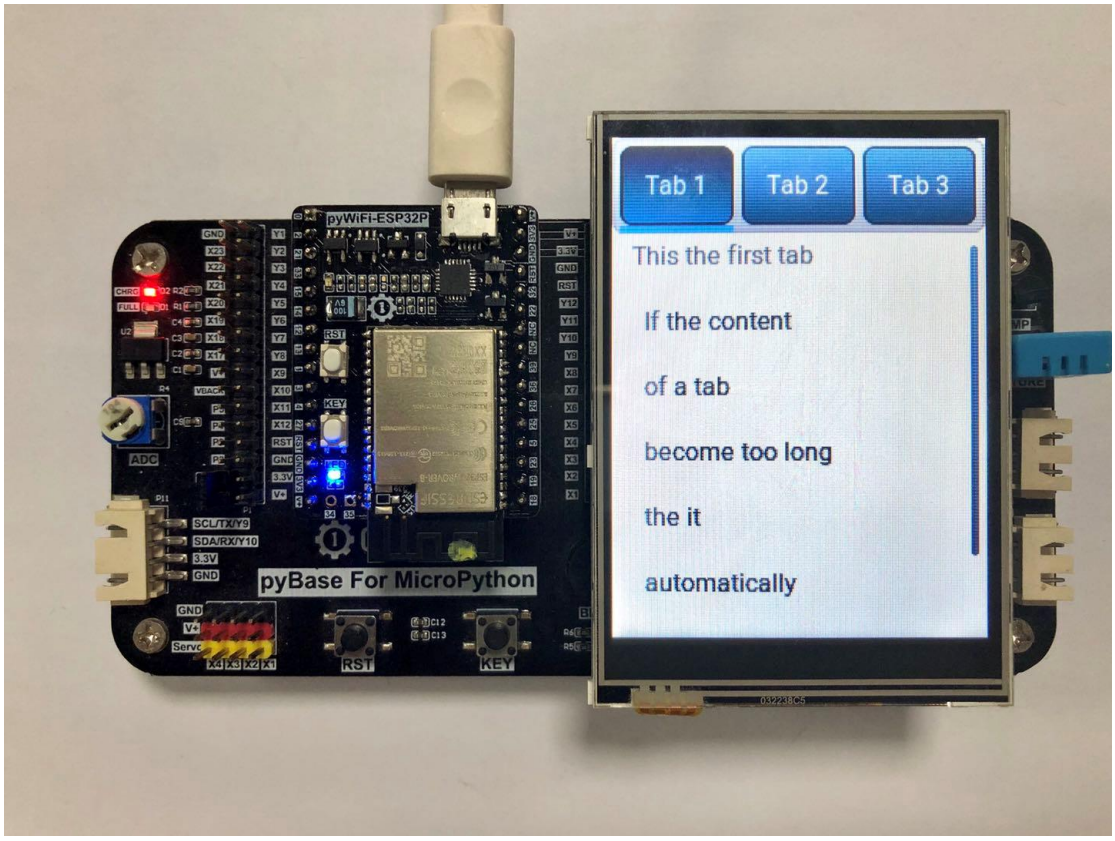


图 16-75 实验结果

- **总结**

本节主要讲解了 LVGL 中选项卡视图的使用方法。

16.3.23 Textarea（文本区域）

- **前言：**

  文本区域用于展示文本。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

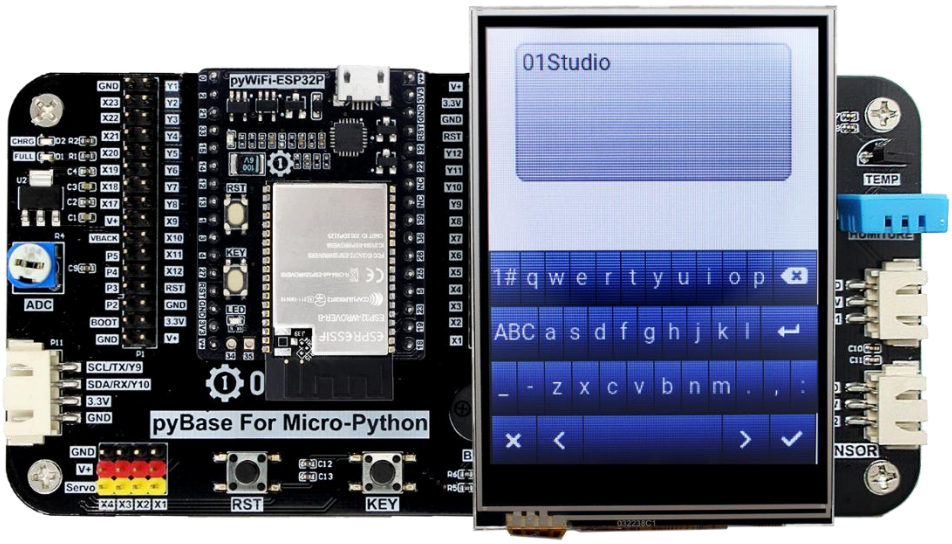


图 16-76　pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建文本区域。

- **实验讲解：**

  我们来看看文本区域对象。

构造函数
`ta = lv.ta(lv.scr_act())`
构建文本区域对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`ta.set_size(width, height)`

设置尺寸； 【width】宽度； 【height】高度。
ta.align()
对齐方式；
ta.set_text(str)
为文本区域添加文本。
更多用法请参阅小部件说明文档：（或在 REPL 中构建对象然后通过 tab 补全功能来查看。） https://docs.lvgl.io/latest/en/html/widgets/textarea.html

表 16-25 文本区域对象

编程思路如下：

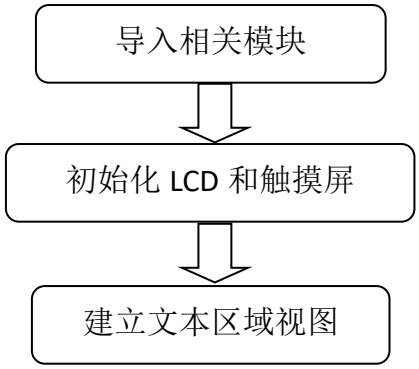


图 16-77 编程流程图

参考代码如下：

```
'''
实验名称：Textarea(文本区域)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码，详情请参考例程代码文件
################################################
################## Textarea ##################
```

```
################################################
if TOUCH_READY:


    ta1 = lv.ta(lv.scr_act())
    ta1.set_size(200, 100)
    ta1.align(None, lv.ALIGN.CENTER, 0, 0)
    ta1.set_cursor_type(lv.CURSOR.BLOCK)
    ta1.set_text("A text in a Text Area")     # Set an initial text
```

- **实验结果：**

  运行代码，可以看见编程实现的文本区域小部件。

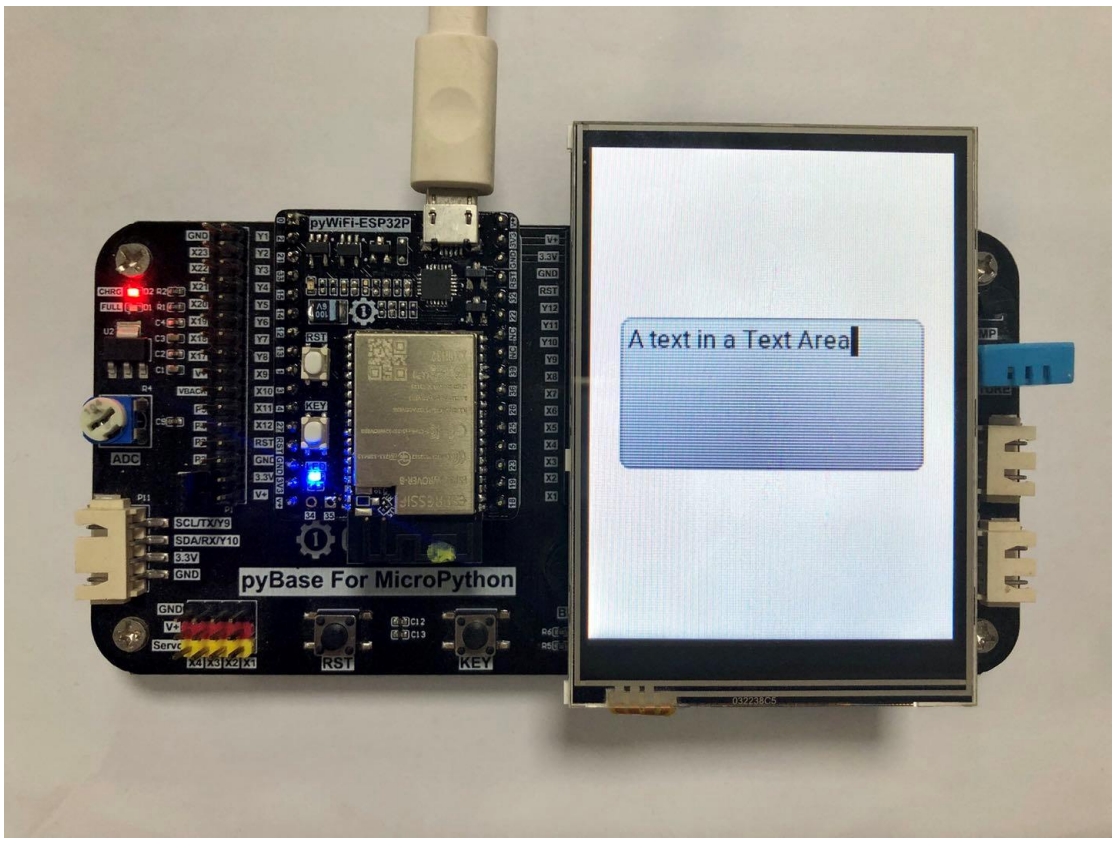


图 16-78 实验结果

- **总结**

  本节主要讲解了 LVGL 中文本区域的使用方法。

16.3.24 Window（窗口）

- **前言：**

  窗口除了展示内容还可以添加关闭、设置等等按钮。

- **实验平台：**

  pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。

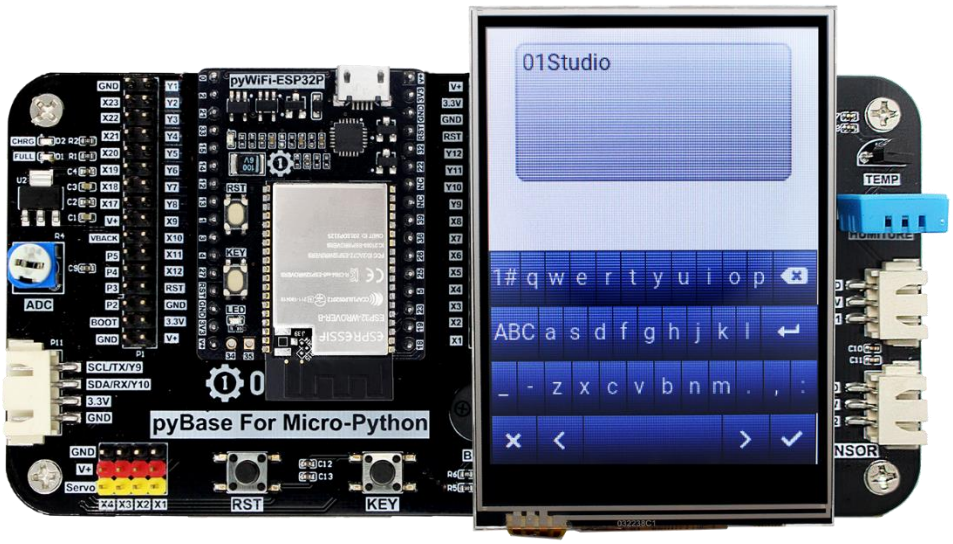


图 16-79 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

  创建窗口。

- **实验讲解：**

  我们来看看窗口对象。

构造函数
`win = lv.win(lv.scr_act())`
构建文本区域对象； lv.scr_act() :表示在当前屏幕创建；
使用方法
`win.set_title(str)`

设置窗口标题; 【str】标题内容;
close_btn = win.add_btn(lv.SYMBOL.CLOSE)
为窗口添加关闭按钮
更多用法请参阅小部件说明文档:(或在 REPL 中构建对象然后通过 tab 补全功能来查看。) https://docs.lvgl.io/latest/en/html/widgets/win.html

表 16-26 窗口对象

编程思路如下:

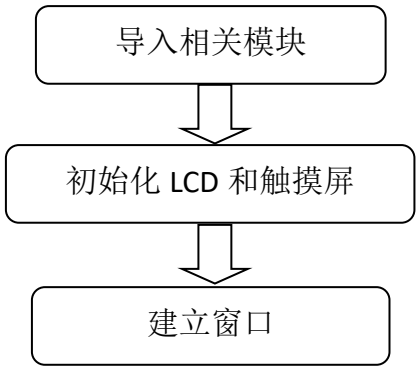


图 16-80 编程流程图

参考代码如下:

```
'''
实验名称:Window(窗口)
版本:v1.0
日期:2020.7
作者:01Studio 【www.01Studio.org】
'''
# 省略 LCD 初始化和触摸屏校准代码,详情请参考例程代码文件
################################################
################## Window ##################
################################################
if TOUCH_READY:
```

```
    # Create a window
    win = lv.win(lv.scr_act())
    win.set_title("Window title")                # Set the title



    # Add control button to the header
    close_btn = win.add_btn(lv.SYMBOL.CLOSE)      # Add close button and
use built-in close action
    close_btn.set_event_cb(lv.win.close_event_cb)
    win.add_btn(lv.SYMBOL.SETTINGS)              # Add a setup button

    # Add some dummy content
    txt = lv.label(win)
    txt.set_text(
"""This is the content of the window
You can add control buttons to
the window header
The content area becomes automatically
scrollable is it's large enough.
You can scroll the content
See the scroll bar on the right!"""
    )
```

- **实验结果：**

  运行代码，可以看见编程实现的窗口。



图 16-81 实验结果

- **总结**

本节主要讲解了 LVGL 中窗口的使用方法。

16.4 项目应用

16.4.1 网络时钟

- **前言：**

日常生活中我们的时钟走着走着可能会出现偏差，这时候最方便的方法就是通过互联网获取当前时间信息，从而更新本地时间。这用到了 NTP 服务器【Network Time Protocol（NTP）】是用来使计算机时间同步化的一种协议，它可以使计算机对其服务器或时钟源（如石英钟，GPS 等等)做同步化，它可以提供高精准度的时间校正。

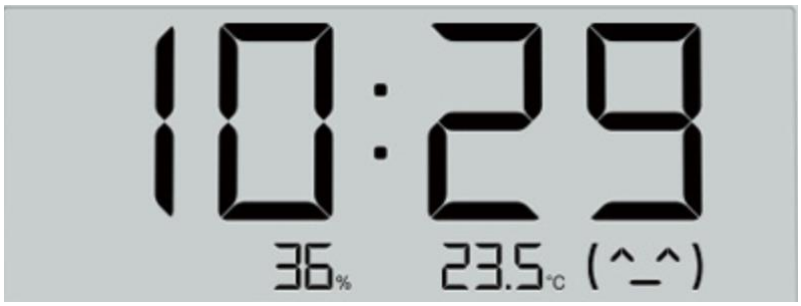


图 16-82 时钟

- **实验平台：**

pyWiFi-ESP32P + 3.2 寸 LCD 开发套件。



图 16-83 pyWiFi-ESP32P + 3.2 寸 LCD 开发套件

- **实验目的：**

编程实现 NTP 时钟，并显示温湿度数据。

- **实验讲解：**

Micropython 基于 ESP32 平台的库自带了 RTC 实时时钟以及 NTP 功能，也就是说我们可以结合之前的内容，可以轻松编程实现在线时钟同步。Wifi 连接的相关对象用法这里不再重复。

先来看看 NTP 的用法：

构造函数
`import ntptime`
直接导入模块
使用方法
`ntptime.settime()`
将当前 RTC 时间设置成 NTP 获取的时间。

表 16-27 NTP 对象

编程思路如下：

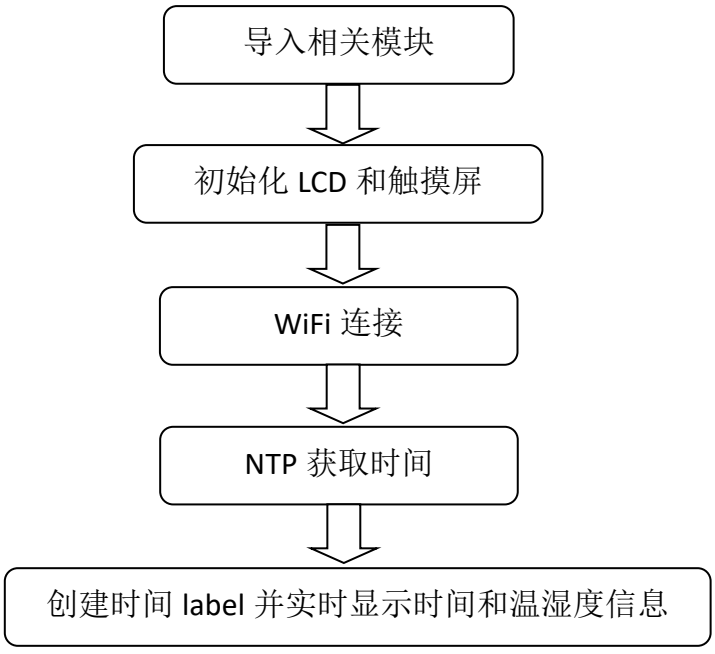


图 16-84 编程流程图

参考代码如下：

```
'''
实验名称：NTP(网络时钟)
版本：v1.0
日期：2020.7
作者：01Studio 【www.01Studio.org】
'''
############################################################
###  省略 LCD 初始化、触摸屏校准代码，详情请参考例程代码文件 ###
############################################################

#RTC 初始化
# 定义星期和时间（时分秒）显示字符列表
week = ['Mon', 'Tues', 'Wed', 'Thur', 'Fri', 'Sat', 'Sun']
time_list = ['', '', '']
rtc = RTC()

#WIFI 连接函数,连接成功后更新时间
def WIFI_Connect():

    wlan = network.WLAN(network.STA_IF) #STA 模式
    wlan.active(True)                   #激活接口
    start_time=time.time()              #记录时间做超时判断

    if not wlan.isconnected():
        print('connecting to network...')
        #输入 WIFI 账号密码
        wlan.connect('01Studio', '88888888')
        while not wlan.isconnected():

            #超时判断,15 秒没连接成功判定为超时
```

```
            if time.time()-start_time > 15 :
                print('WIFI Connected Timeout!')
                break

    if wlan.isconnected():

        #串口打印信息
        print('network information:', wlan.ifconfig())

        for i in range(5): #最多尝试获取 5 次时间
            try:
                ntptime.settime()
                print(rtc.datetime())
                time.sleep_ms(500)
                return None

            except:
                print("Can not get time!")

##############################################
################ NTP Clock #################
##############################################
if TOUCH_READY:

    WIFI_Connect() #连接 WiFi

    #Create a style based on style_plain
    mystyle = lv.style_t(lv.style_plain)
    #mystyle.text.font = roboto_80 # font roboto 80 pixel
    mystyle.text.font = lv.font_roboto_28
```

```
# background top color (main), 0xRRGGBB
mystyle.body.main_color = lv.color_hex(0x000000)
# background bottom color (gradient), 0xRRGGBB
mystyle.body.grad_color = lv.color_hex(0x000000)
mystyle.text.color = lv.color_hex(0xffffff) # text-colour, 0xRRGGBB

#Create screen and labels
scr = lv.obj()
scr.set_style(mystyle)

LOGO = lv.label(scr)
LOGO.set_pos(0,0)
LOGO.set_drag(True)
LOGO.set_text("01Studio")

WIFI_ICON = lv.label(scr)
WIFI_ICON.set_pos(200,0)
WIFI_ICON.set_drag(True)
WIFI_ICON.set_text(lv.SYMBOL.WIFI)

DATE = lv.label(scr)
DATE.set_pos(0,64)
DATE.set_drag(True)

TIME = lv.label(scr)
TIME.set_pos(0,128)
TIME.set_drag(True)

TEMP = lv.label(scr)
TEMP.set_pos(0,192)
```

```
    TEMP.set_drag(True)

    HUMI = lv.label(scr)
    HUMI.set_pos(0,256)
    HUMI.set_drag(True)

    a=0

#Create content
    while(1):

        datetime = list(rtc.datetime())  # 获取当前时间

        #北京时间，月、日、星期需要适当调整
        datetime[4]=datetime[4]+8 #北京时间，东八区
        if datetime[4] >= 24:
            datetime[4]=datetime[4]%24
            if datetime[1] in [1,3,5,7,8,10,12]: #大月
                datetime[2] = (datetime[2]+1)%32
            else: datetime[2] = (datetime[2]+1)%31
            datetime[3] = (datetime[3]+1)%8

        #显示日期
        DATE.set_text(str(datetime[0]) + '-' + str(datetime[1]) + '-' +
                      str(datetime[2]) + ' ' + week[datetime[3]])

        # 显示时间需要判断时、分、秒的值否小于10，如果小于10，则在显示前面
        #补“0”以达到较佳的显示效果
        for i in range(4, 7):
            if datetime[i] < 10:
```

```
            time_list[i - 4] = "0"
        else:
            time_list[i - 4] = ""
    TIME.set_text(time_list[0] + str(datetime[4]) + ':' +
              time_list[1] + str(datetime[5]) + ':' + time_list[2] +
              str(datetime[6]))

    a=a+1 #控制 DHT11 采集时间，大于 2 秒间隔
    if a ==7:
        d.measure()         #温湿度采集
        a=0
        TEMP.set_text(str(d.temperature() )+' C')
        HUMI.set_text(str(d.humidity())+' %')

    # 300ms 刷屏一次
    lv.scr_load(scr)
    time.sleep_ms(300)
```

- **实验结果：**

运行代码，可以看见当成功获取 NTP 时间后，LCD 显示时钟以及采集到的温湿度数据。Label 标签被定义成了可拖动，所以用户可以根据自己的喜好自行对显示进行布局。

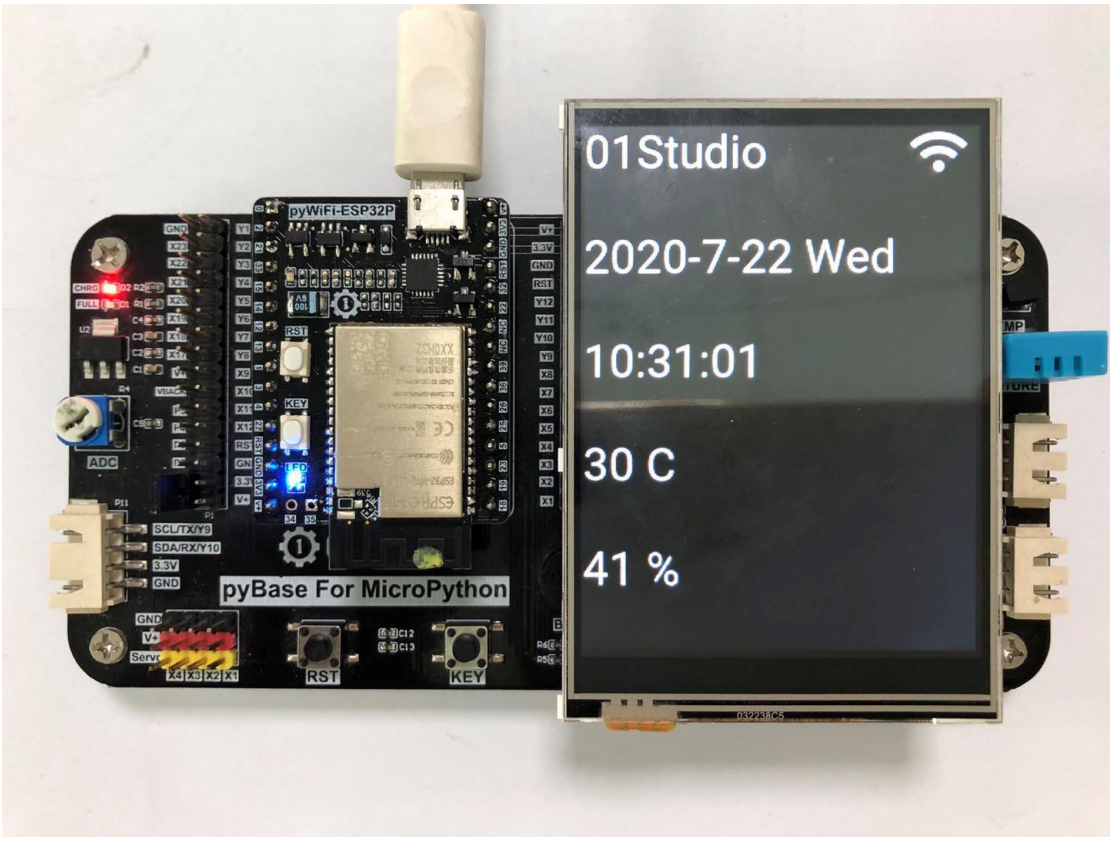


图 16-85 实验结果

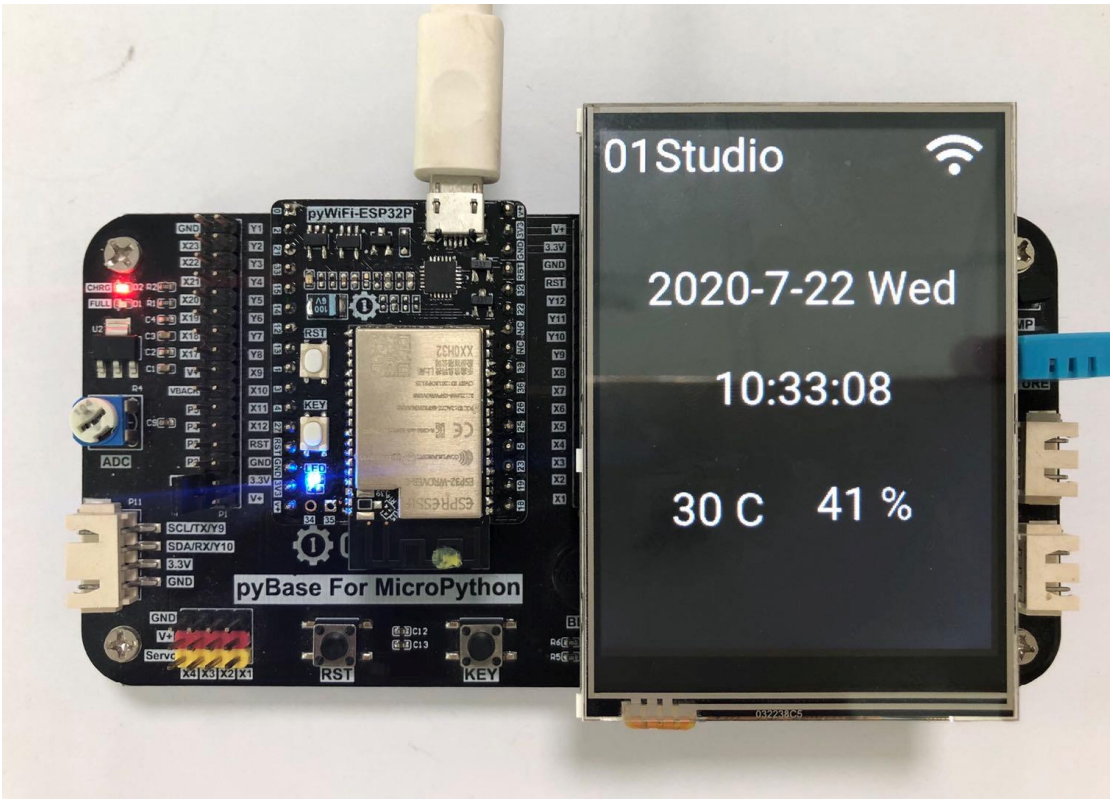


图 16-86 可拖动 label 实现自己喜欢的布局

- **总结**

本节结合了 NTP、WiFi、RTC 以及 LVGL 内容，实现了一个简易的实时时钟和温湿度计，但 UI 明显是比较简洁的，有兴趣的用户可以结合前面 LVGL 小部件内容，尝试实现自己精美的 UI 时钟界面。

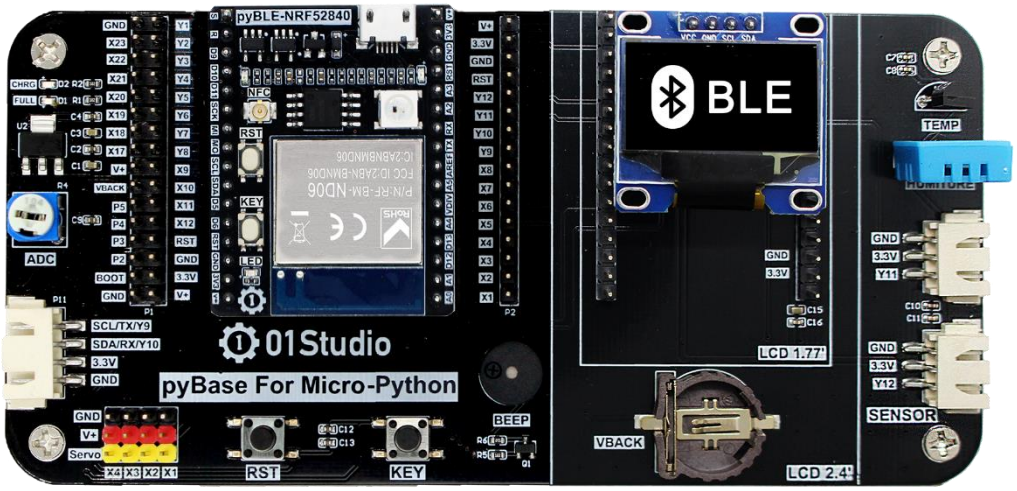
第五部分　基于 NRF52840 平台

NRF52840 是 Nordic 公司推出的一款强大的 BLE SOC 芯片，该芯片的特点是集成度非常高，拥有 256KB RAM 和 1MB Flash，支持蓝牙 5.0，同时还集成了 USB 2.0。

传统的 BLE（低功耗蓝牙）开发方式是基于嵌入式 C 语言体系，这为一部分想快速应用物联网的用户增加了门槛，NRF52840 的协议栈已经被成功移植到 CircuitPython（MicroPython 的一个分支），而且拥有非常多的拓展库模块，我们使用 Python 进行开发，可以快速实现基础功能和联网，非常高效！

如果你已经学习过 01Studio 其它 MicroPython 开发平台，那么恭喜你，你需要的只是搭建一下开发环境就可以快速开发了。如果没有学习过相关内容也没关系，本部分也是会从 0 到 1，带你一步步学会 MicroPython 基于 NRF52840 的编程应用。

本部分内容包含开发环境快速建立、基础实验、拓展模块和 BLE 应用，由于 NRF52840 主要是实现物联网 BLE 无线连接，因此我们会重点讲解 BLE 联网实验内容。



pyBLE-NRF52840 开发套件

第17章　开发环境快速建立

17.1　基于 Windows

Windows 是大部分开发人员的主流平台，毕竟微软凭借其 windows 操作系统在 PC 时代统治了几十年，大部分软件厂商还是愿意针对 windows 来开发桌面软件，但这也跟乔布斯英年早逝有关。扯远了，所以为了后面省事，我们还是推荐使用 Windows 作为首选的开发环境，推荐使用 Windows 10, 微软官方已经宣布停止对 win 7 更新和维护了，win10 功能强大，有些驱动还能自动联网安装，免去人工安装的麻烦。

17.1.1　安装 python3

说明：我们提供安装 python3 教程只是为了在 PC 上体验一下 python3 安装和使用过程，由于 micropython 固件集成了 python 库，所以电脑并非一定要安装，本节可以跳过。

python3 并不是 python2 的升级版，而是 python2 的一个分支，它们都发展了自己独特的功能和函数，市面上有些产品开发时候甚至不相互兼容，总体来说初学者使用 python3 较多。具体安装方法如下：

在 MicroPython 开发套件配套资料\开发工具\Windows\python3 安装包下找到 python-3.7.3-amd64.exe 文件打开（这个文件是针对 64bit 操作系统的用户，32bit 系统的用户自行到 python 官网下载相关版本安装：www.python.org ）。

打开后一直按下一步安装即可。安装路径不能有中文，推荐默认路径安装。安装过程中务必勾上 Add to PATH 选项。

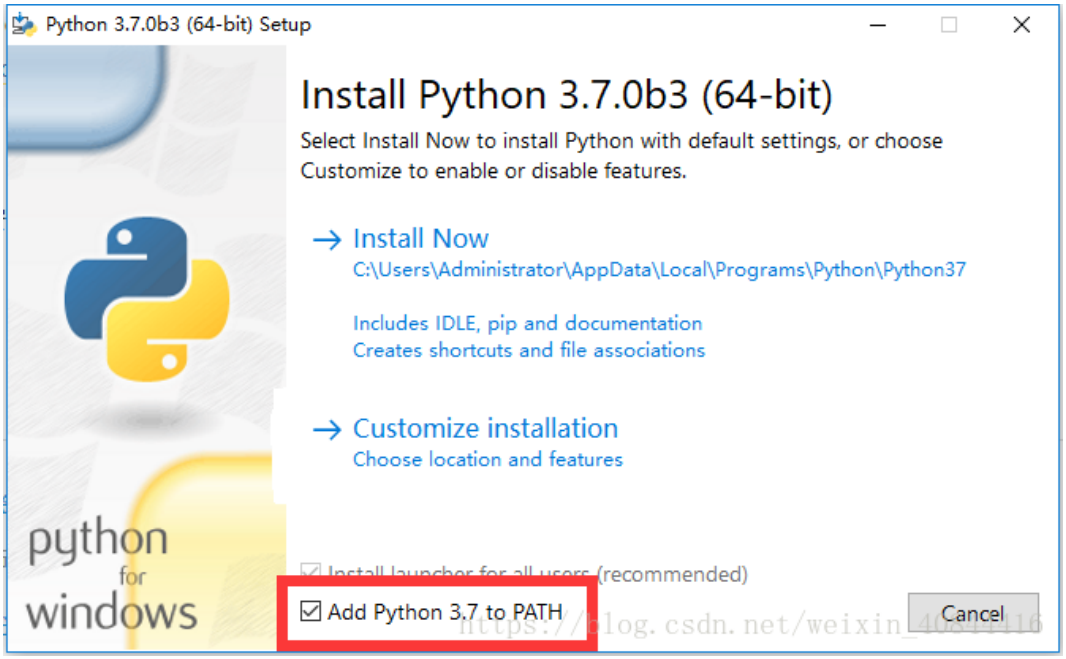


图 17-1 务必勾上 Add to PATH

安装完成后点击计算机的运行（快捷键是 windows+R），输入 cmd，弹出命令框；

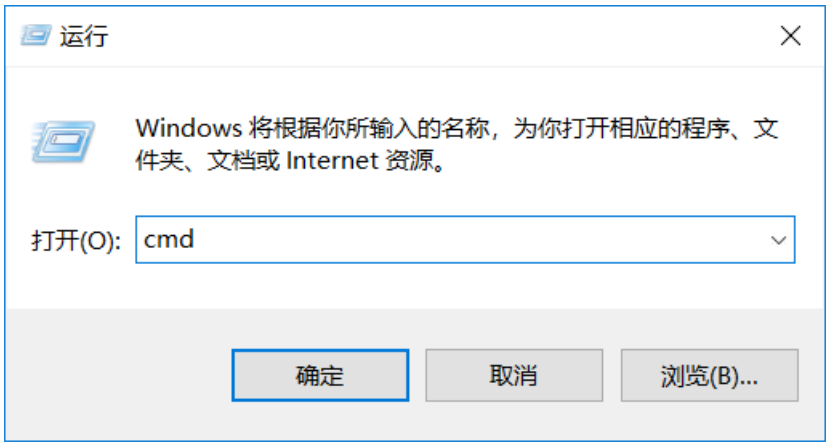


图 17-2 运行输入 cmd 进入命令

输入 python 按回车，出现 Python 3 版本信息即安装成功。

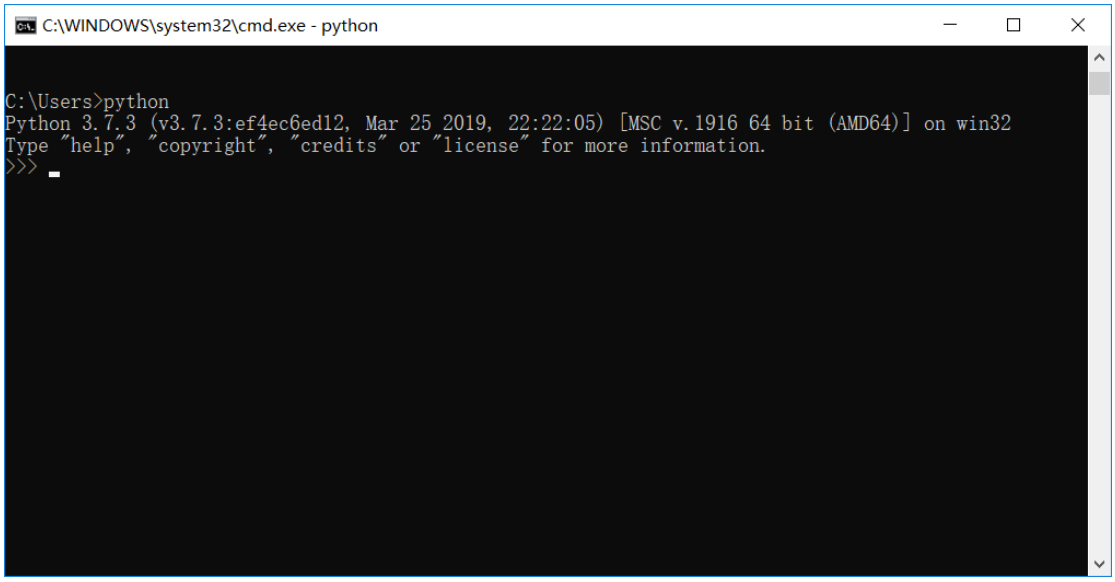


图 17-3 python 成功安装信息

此时已经进入 python 自带的编译器，我们在>>>后面输入 print(“Hello World!”)按回车，可见系统打印出来 Hello World! 需要退出的话直接关闭窗口即可。

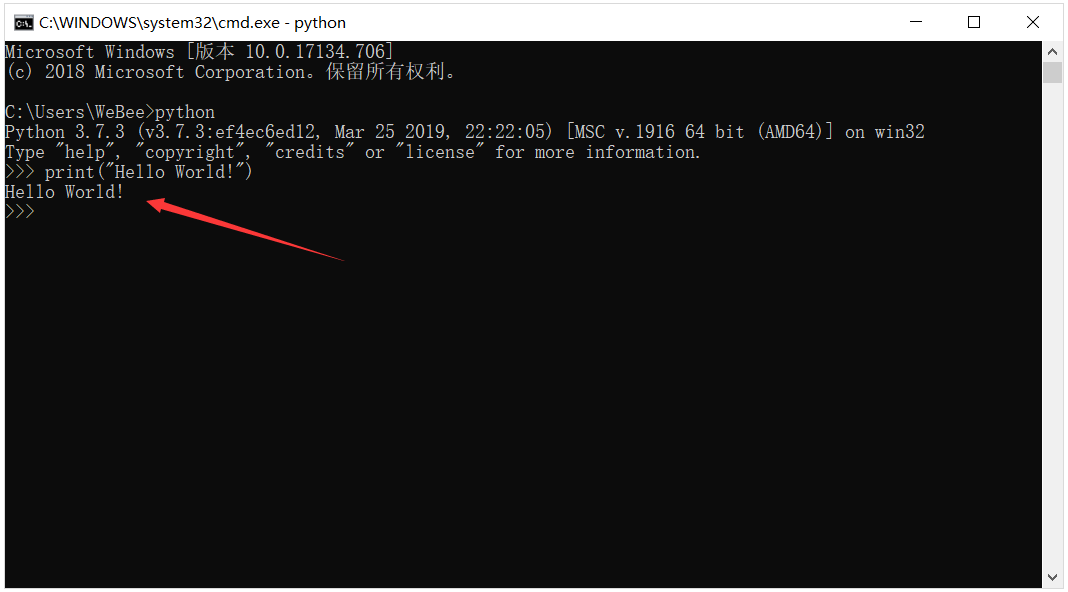


图 17-4 打印 Hello World 操作

至此，电脑已经成功安装 python3。

17.1.2 安装开发软件 Mu

开发软件是指我们用来写代码的工具，上一节那个命令框里面其实就是调用了自带的编程器，但是我想没有人希望一直在那黑框里面撸代码。 Python 拥有众多的编程器，如果你之前已经熟练掌握 python 或已经使用 python 开发，那么可以直接使用你原来习惯的开发软件来编程。如果你是初学者或者喜欢简单而快速应用，我们推荐使用 Mu。

Mu 是一款开源软件，以极简方式设计，对 MicroPython 的兼容性非常友善。而且支持 Windows、Mac OS、Linux、树莓派。由于开源，所以软件迭代速度非常快，功能日趋成熟。具体安装方法如下：

在 https://codewith.mu/en/download 下载地址，选择自己的开发平台进行下载安装即可：

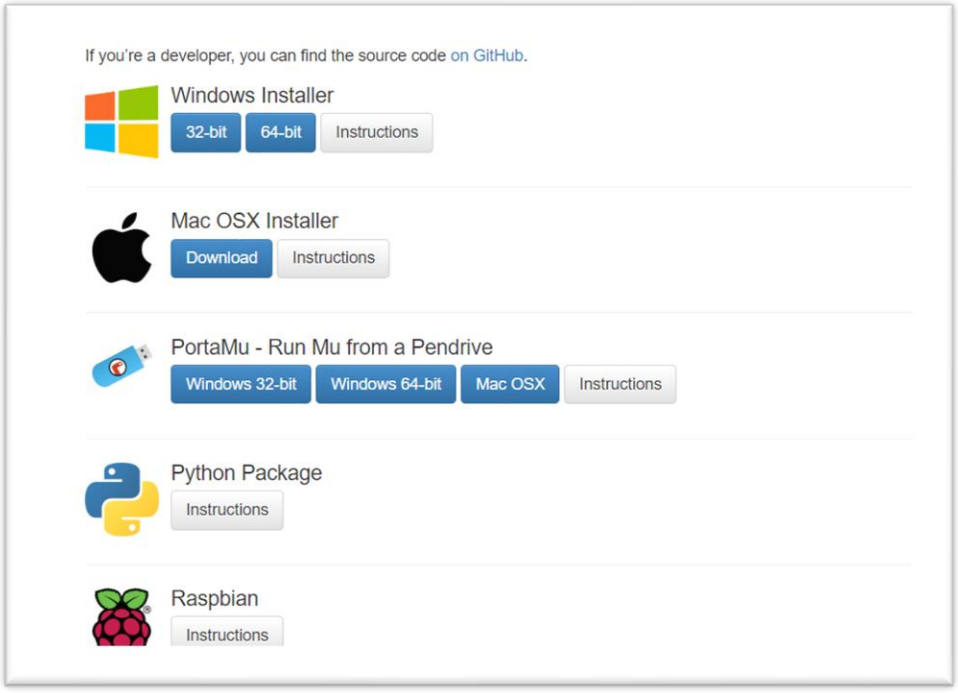


图 17-5 稳定版下载

由于 Mu 更新非常快，因此社区也同时提供最新版下载，给愿意尝鲜和给予软件反馈的用户，下载位置如下：

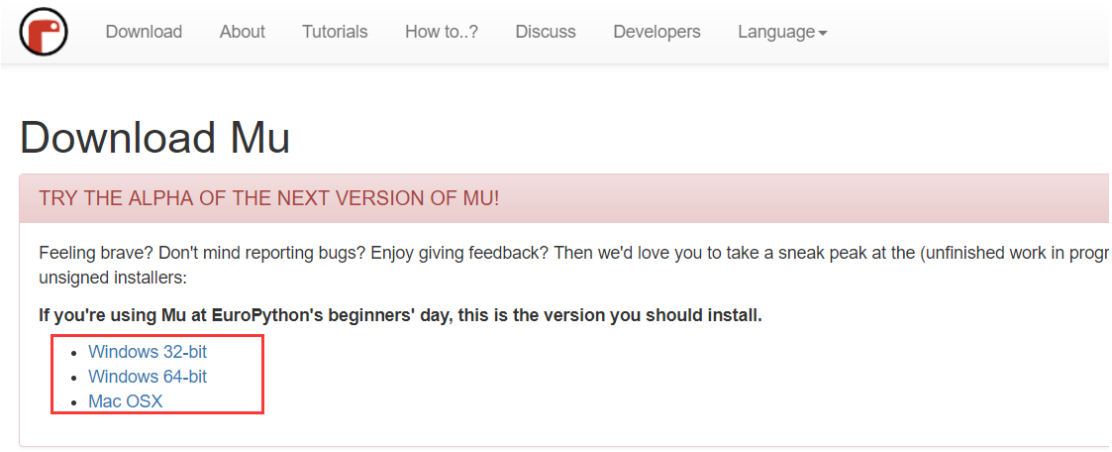


图 17-6 尝鲜版下载

我们推荐使用尝鲜版，能使用最新的功能。下载安装后，打开 Mu，可以看你的界面非常简洁：

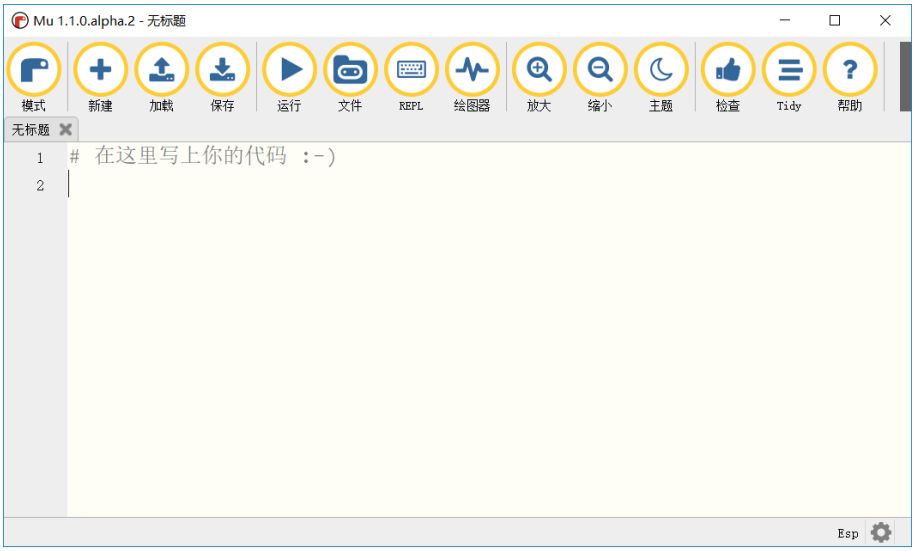


图 17-7 Mu 主界面

点击左上角“模式”，弹出对话框，可以看到支持 MicroBit、ESP 等一系列 MicroPython 产品。基于 pyBLE-NRF52840 系列产品选择【Circuit Python】模式进行开发。

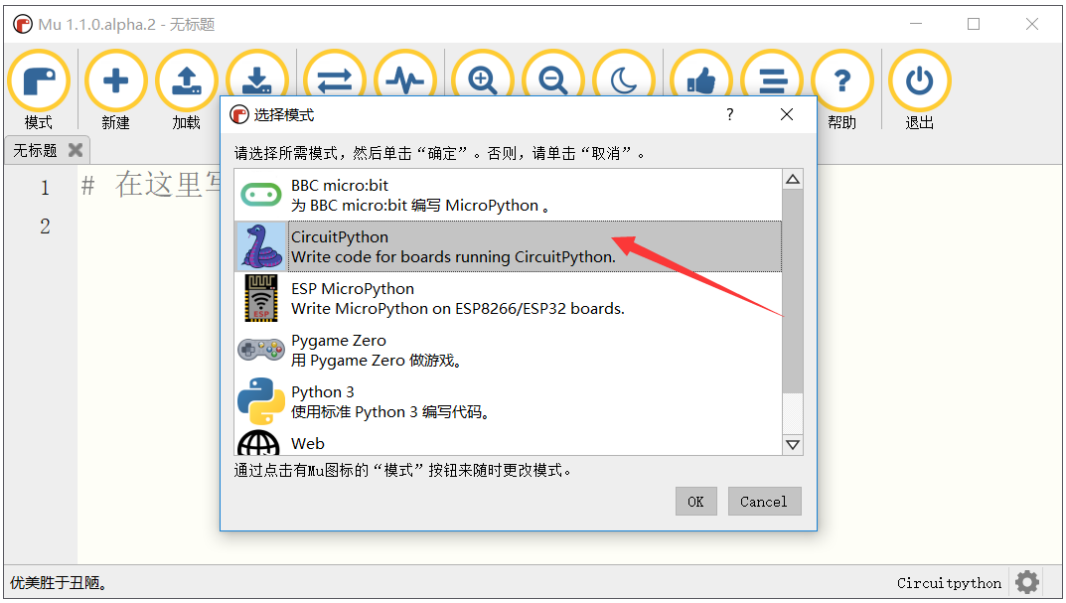


图 17-8　选择 CircuitPython

如果出现以下界面，直接点 OK 忽略即可。

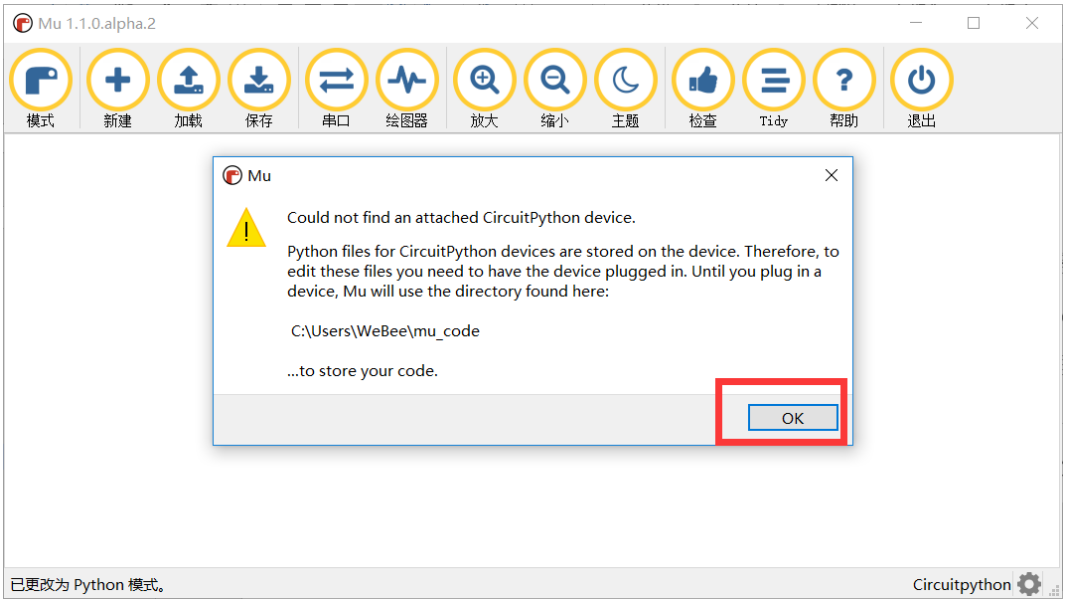


图 17-9 点击 OK 忽略提示

至此，Mu 安装完成。

17.1.3 开发套件使用

17.1.3.1 驱动安装

我们将 pyBLE-NRF52840 开发板通过 MicroUSB 线连接到电脑：

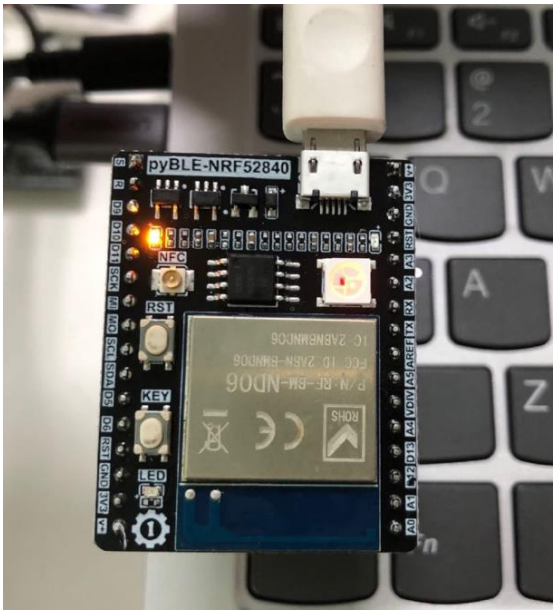


图 17-10

连接电脑后，会发现出现一个容量约 2MB 的 U 盘，名字是 CIRCUITPY。

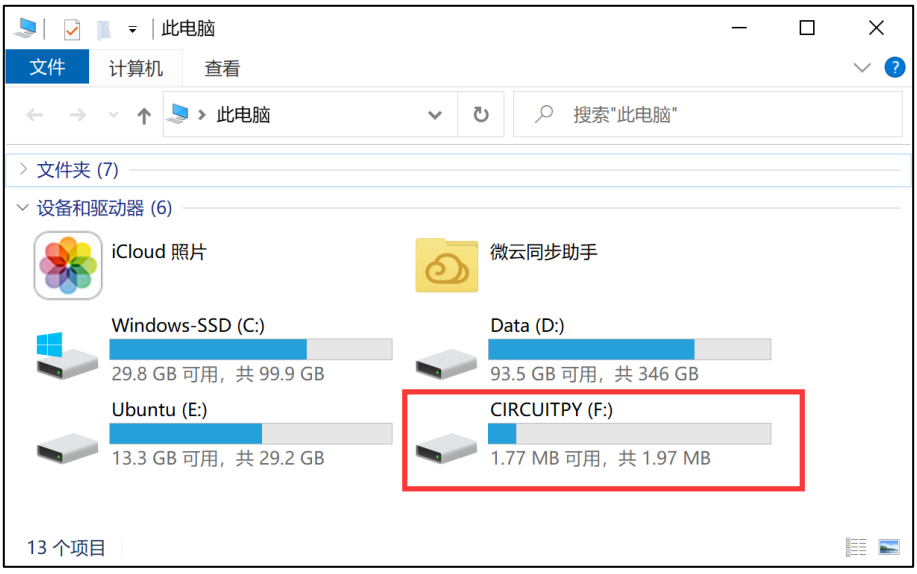


图 17-11

U 盘里面有一些文件如下图(有可能出现更多文件)，我们后面再做说明。

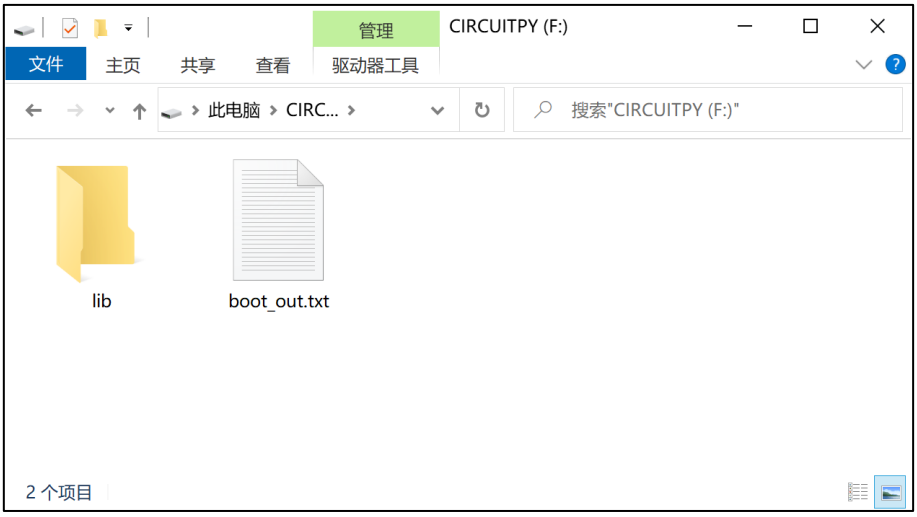


图 17-12

除了 U 盘设备外，开发板还集成了一个虚拟串口设备，可以鼠标右键点击“我的电脑”—“属性”—“设备管理器”查看。设备名称一般为 COM。

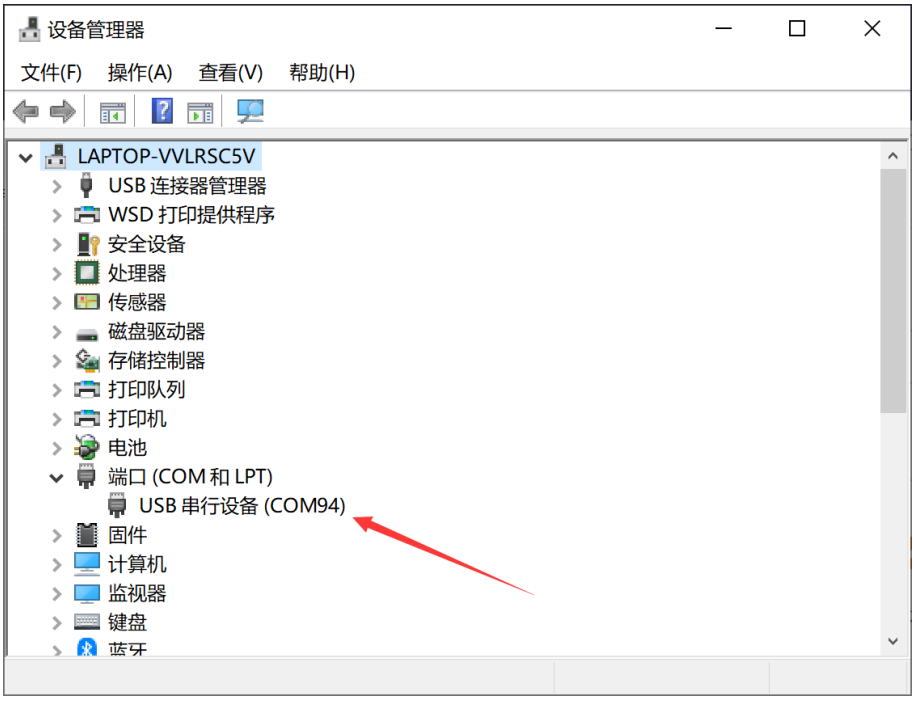


图 17-13 虚拟串口驱动

17.1.3.2 REPL 串口交互调试

Python 开发板集成了交互解释器 REPL 【读取(Read)-运算(Eval)-输出(Print)-循环(Loop) 】，开发者可以直接通过串口终端来调试开发板。而开发软件 Mu 则直接集成了这个功能。

将开发板连接到电脑，打开 Mu 软件，在模式下选择 CircuitPython。

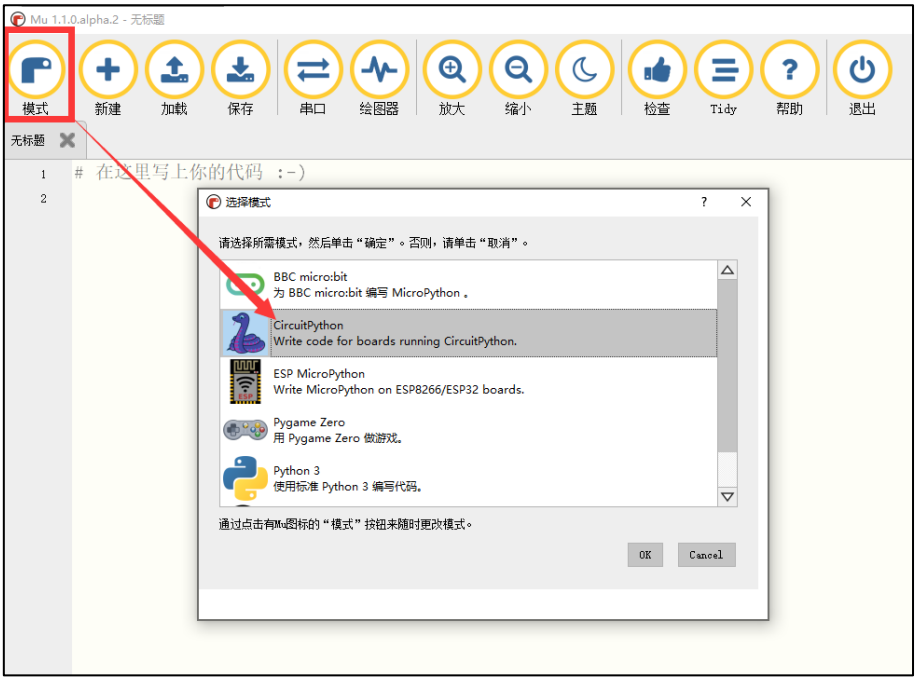


图 17-14

然后点击“串口”图标，可以看到弹出一个 REPL 交互窗口。

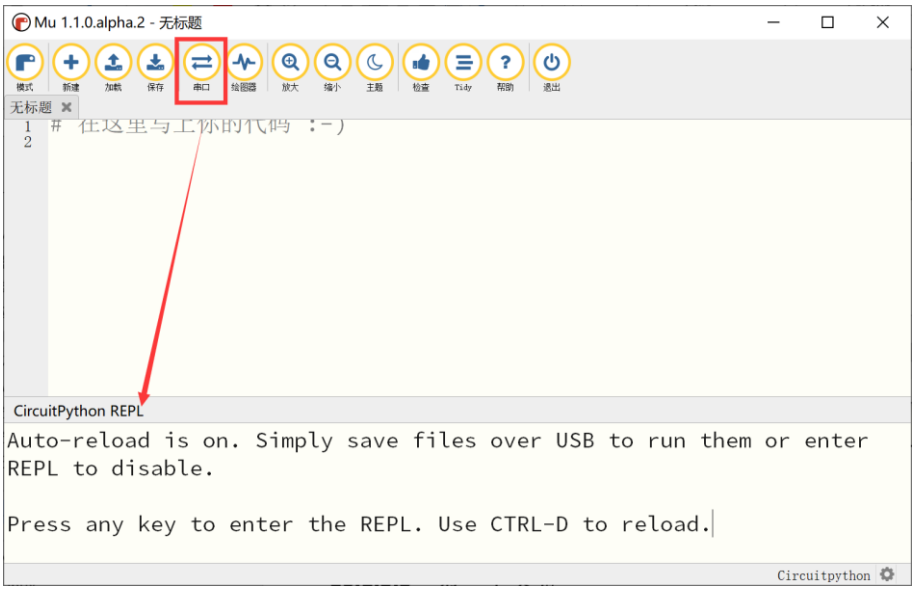


图 17-15

在窗口中按下回车键，即出现开发板相关信息（pyBLE-NRF52840 与 adafruit nrf52840 硬件基本兼容）。



图 17-16

现在相当于对话框连接上了开发板，由于开发板集成了 python 解析器，我们在这里可以进行调试和简单编程，接下来我们测试一下，在对话框输入下面代码，按回车，可以看到代码运行情况。

```
print(“Hello 01Studio!”)
```

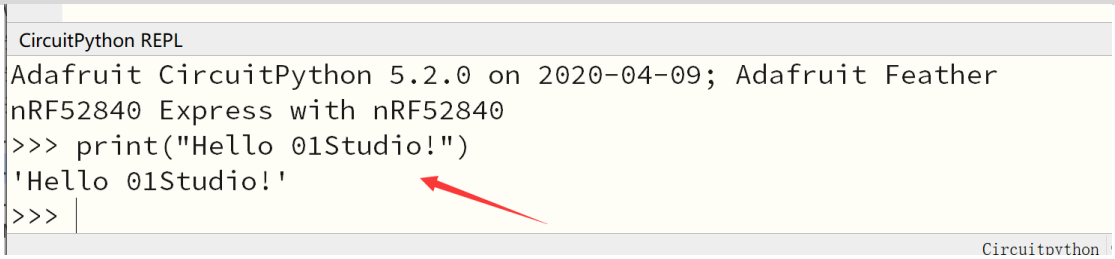


图 17-17

我们再输入 1+1，按回车，得到计算结果为 2。

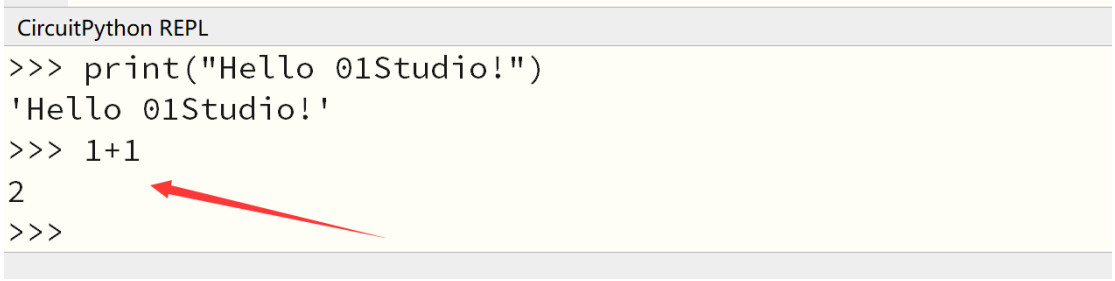


图 17-18

实际上你可以将我们后面讲到的编程实验代码逐行输入到 REPL，开发板也是可以正常运行相应例程的。

提示：如果程序在运行无法进入 REPL，可以按 Ctrl + C 打断程序，再按回车进入。另外 Ctrl + D 是让开发板重启。

另外你也可以使用 putty 作为 REPL 工具，详见：

17.1.3.3 文件系统

pyBLE-NRF52840 内置了文件系统（2M 的 U 盘），可以简单理解成是开发板上电后需要的代码文件和库模块存放处。可以通过 USB 非常方便地查看。

CircuitPython 上电默认运行文件系统里面的 main.py 或者 code.py 文件，为了跟 Micropython 保持一致性，我们本教程统一使用 main.py 作为主程序文件。接下来可以简单测试一下：

将 **开发板配套资料—示例程序—pyBLE-NRF52840—基础实验—点亮第一个 LED** 文件夹下的 main.py 复制到 CircuitPy U 盘。**（如果开发板里面有 code.py，请把 code.py 删除，因为开发板上电默认先运行 code.py，再运行 main.py。当然你也可以把文件改成 code.py 拷贝进去运行。）**

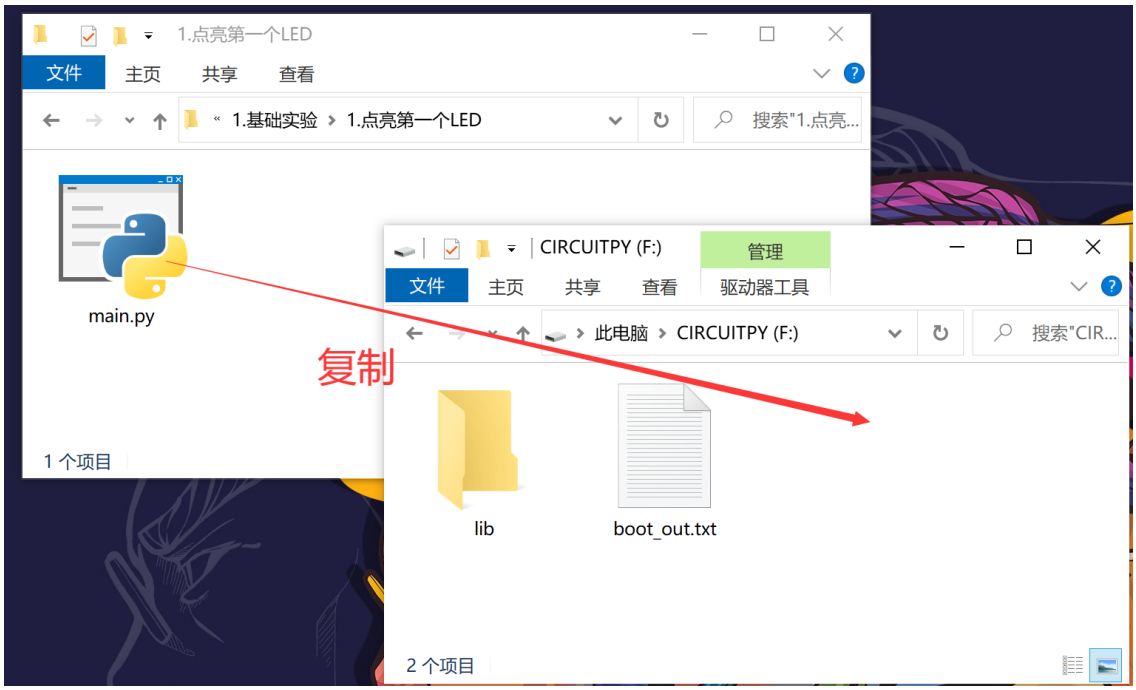


图 17-19

复制完成后按下开发板复位（RST）按键，可以看到开发板重新运行，LED 蓝灯灯被点亮。

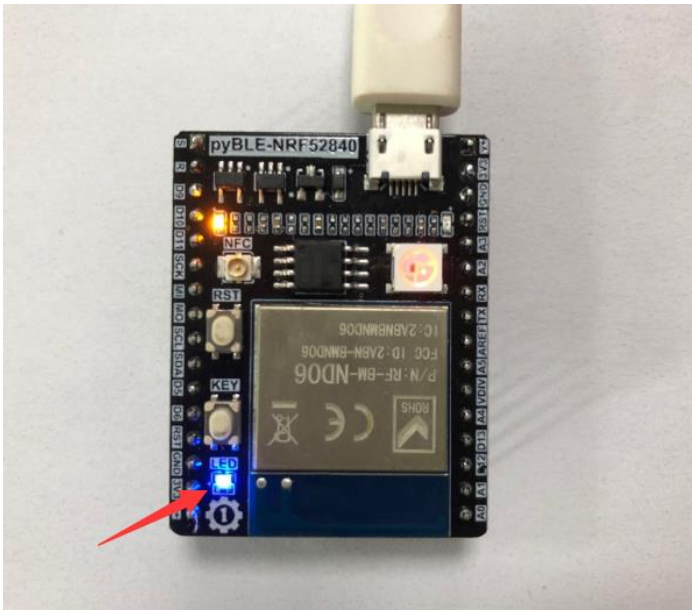


图 17-20

我们可以使用 Mu 来查看和编辑软件代码，在 Mu 中使用‘加载’图标打开 circuitpy U 盘里面的 main.py 文件。然后可以在上面修改，**在 Mu 中修改后按保存，开发板会自动重新运行新的程序代码。**

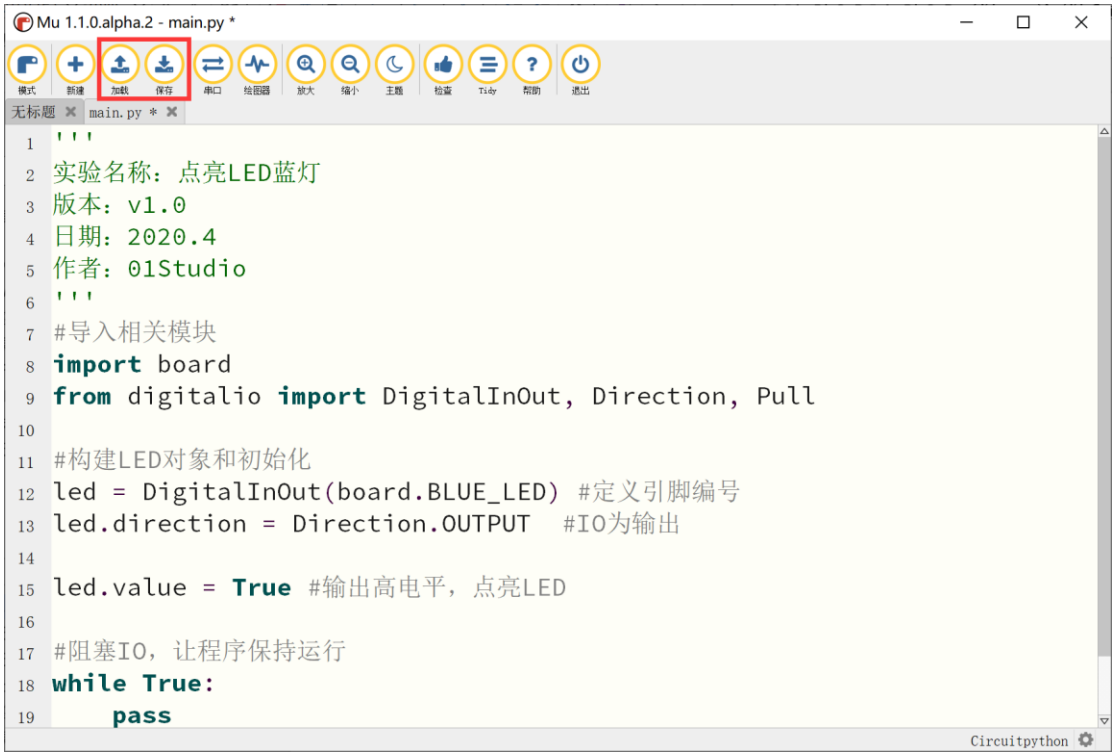


图 17-21

17.1.3.4 恢复出厂设置

pyBLE-NRF52840 目前仅支持 REPL 指令恢复出厂设置，也就是刷除整个文件系统，恢复初始文件。在 Mu 中点击串口，输入下面指令可以实现，改指令表示擦除整个文件系统并重新初始化文件系统。

```
import storage

storage.erase_filesystem()
```

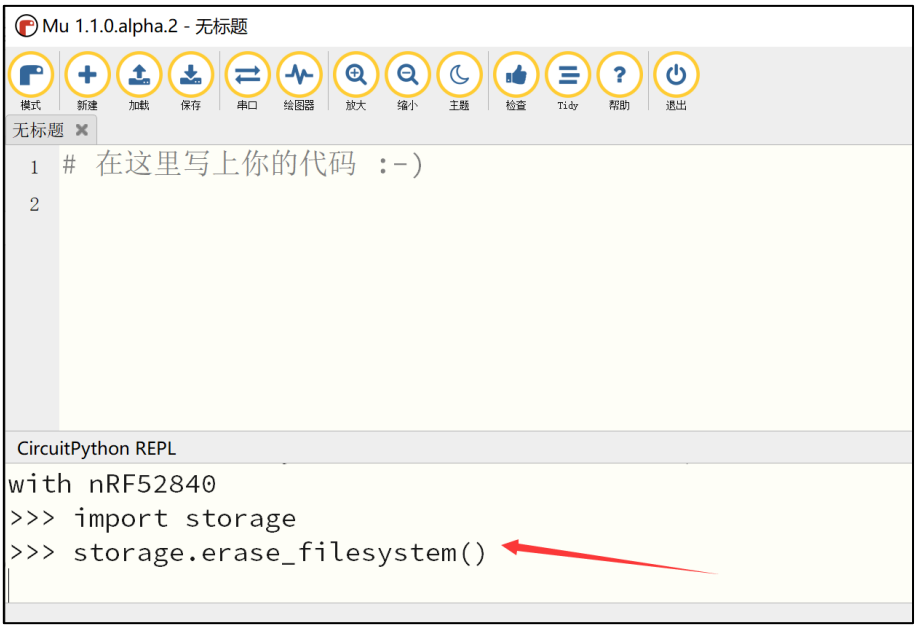


图 17-22 通过 REPL 指令恢复出厂设置

恢复后 CIRCUITPY U 盘文件如下图所示：

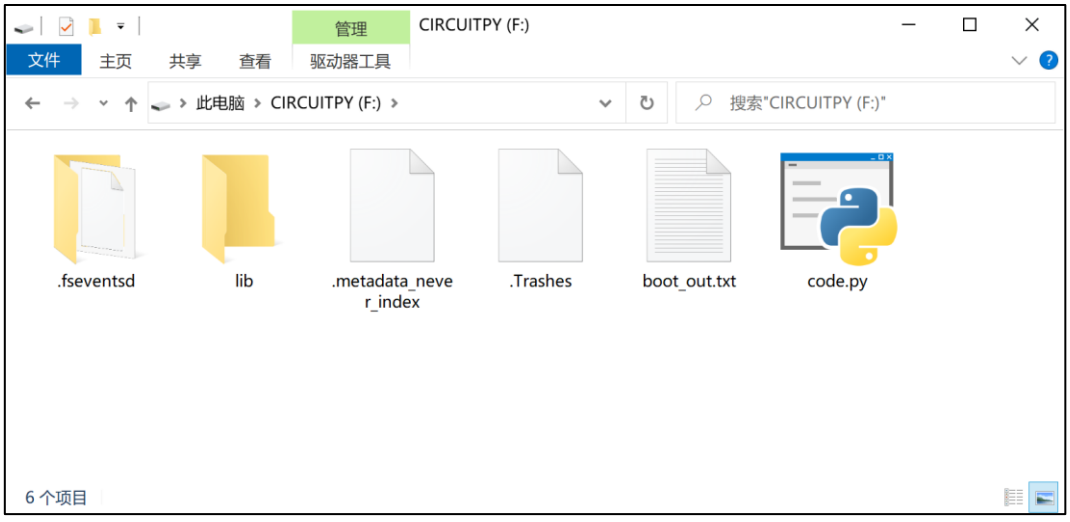


图 17-23 文件系统初始化

17.1.3.5 CircuitPython 固件升级

CircuitPython 大部分固件由 adafruit 官方在维护，而且隔断时间会升级，我们开发板升级的方法非常简单，具体如下：

快速按 2 下开发板的 RST 复位按键便可以进入 CircuitPython 固件更新模式，可以看到 U 盘名称和里面文件改变了：

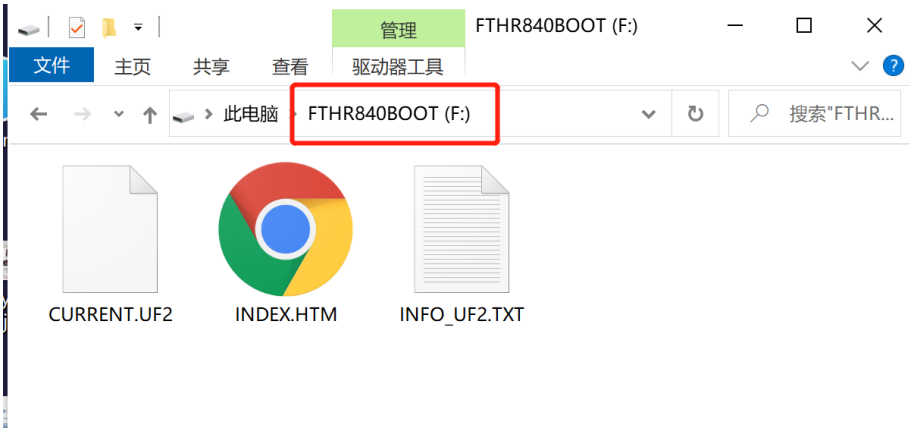


图 17-24

然后将 **零一科技（01Studio）MicroPython 开发套件配套资料_lastest\03-相关固件\04-pyBLE-NRF52840\01-CircuitPython 固件** 里面的 uf2 固件文件直接拖动到 U 盘，即可自动更新。

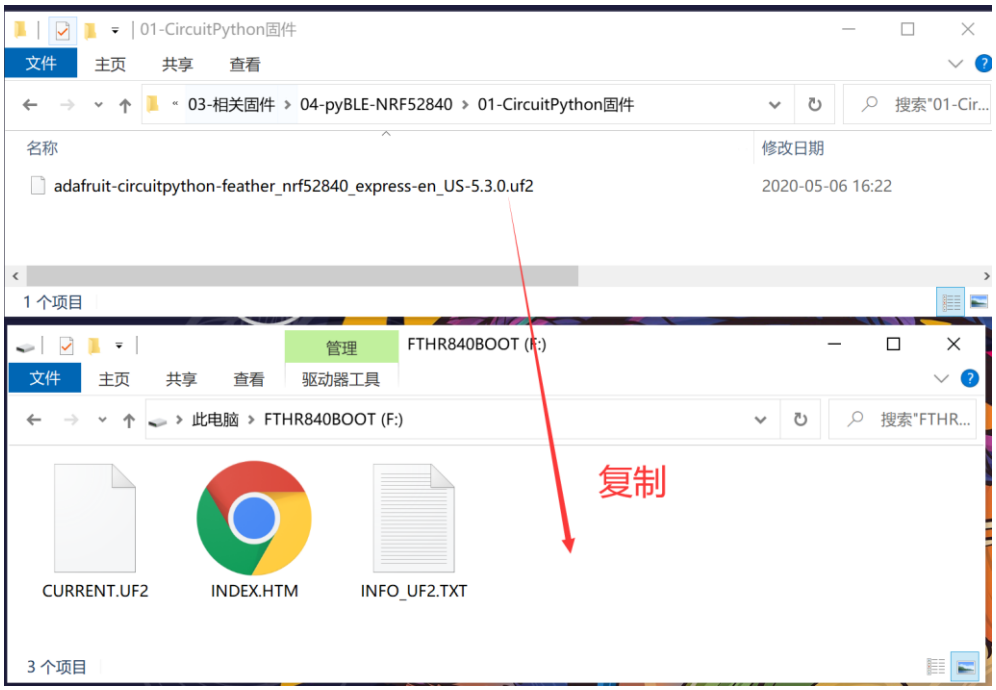


图 17-25 CircuitPython 版本升级

升级完成后自动重启（或者按下复位键），重新出现 CIRCUITPY U 盘，里面的代码是不会丢失的。

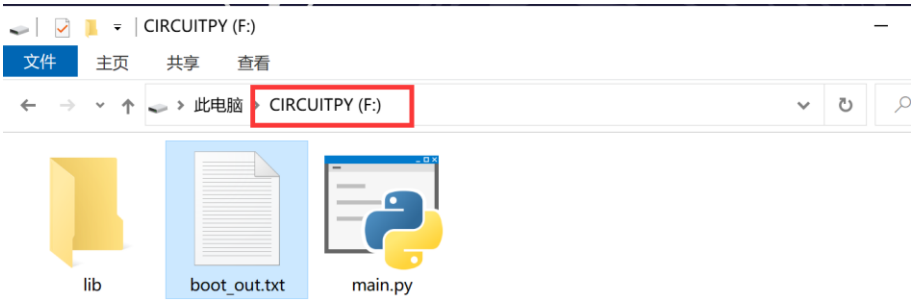


图 17-26 升级完成

17.1.3.6 Bootloader 固件烧写

对于一个完全空的 NRF52840 新芯片或者模块。我们首次需要烧录一个 bootloader 固件，这样才会出现上一节提到的 Boot U 盘，才能升级 CircuitPython 固件。比较简单的一个方法是使用 J-Link 的 SWD 功能来烧写。

可以使用官方的 Jlink 或者廉价版的 Jlink OB 来下载：



图 17-27 官方 Jlink



图 17-28 Jlink-OB

pyBLE-NRF52840 背部一个 Jtag 接口，而实际烧录只需要连接 4 根线 SWD 接口，分别是 VCC（3.3V），GND，SWDIO，SWCLK 4 个接口。

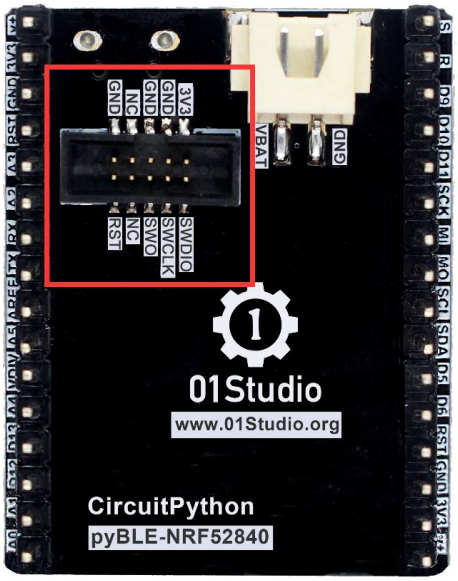


图 17-29 J-TAG 接口

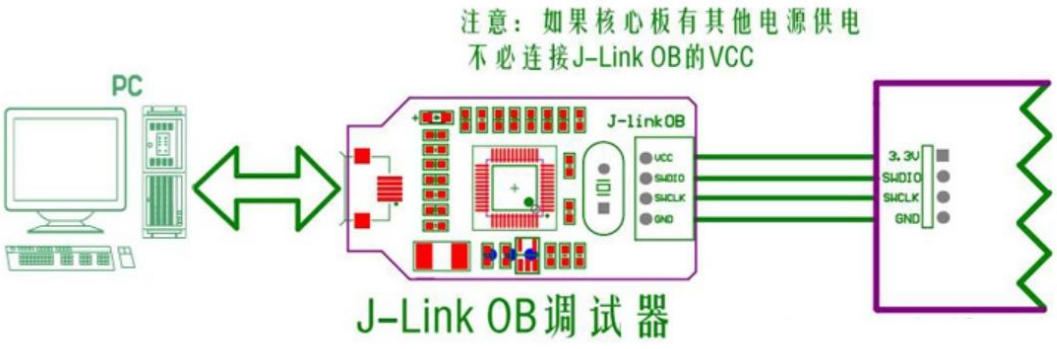


图 17-30 SWD 下载

可以使用 01Studio JlinkOB 配套的转接板直接下载，接线如下：

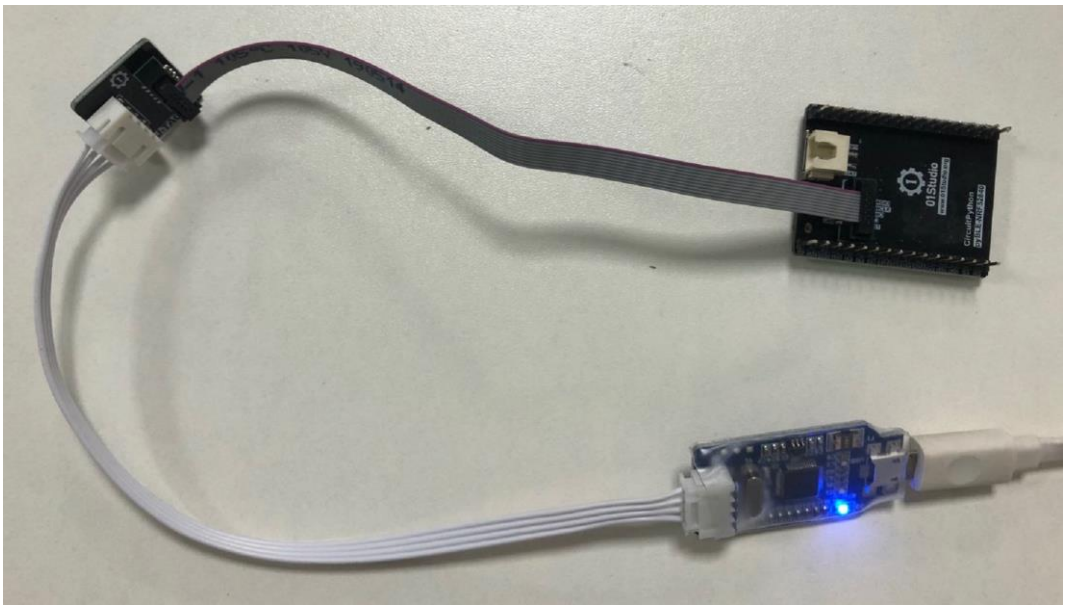


图 17-31 配合转接板连接开发板

接下来我们在电脑安装 J-Link 下载软件 J-Flash，打开链接：https://www.segger.com/products/debug-probes/j-link/tools/j-flash/about-j-flash/

在打开网页找到下图内容，选择自己操作系统版本，然后点击 Download。

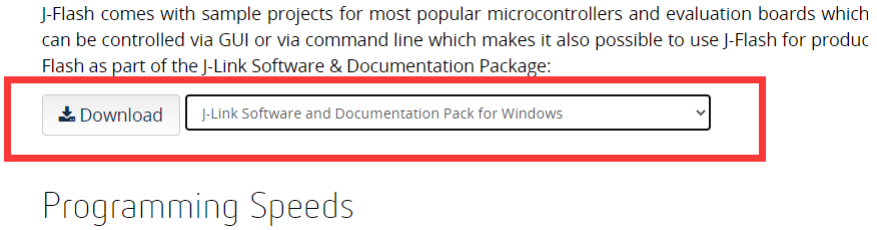


图 17-32 Windows 版本下载

下载完成后直接安装，有选择的地方全部默认选择即可。

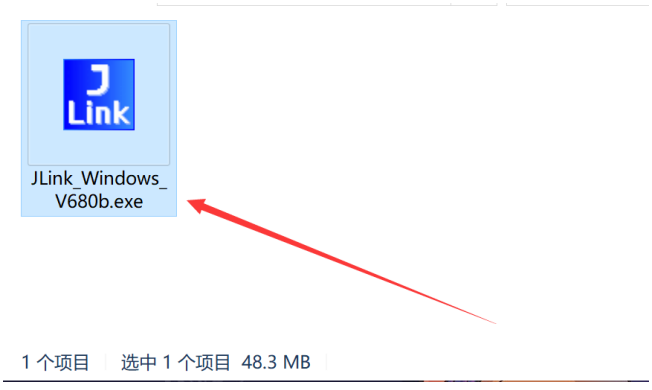


图 17-33

安装完成后桌面可能没有应用图标，这时候可以在搜索栏搜索‘J-flash’,然后打开：

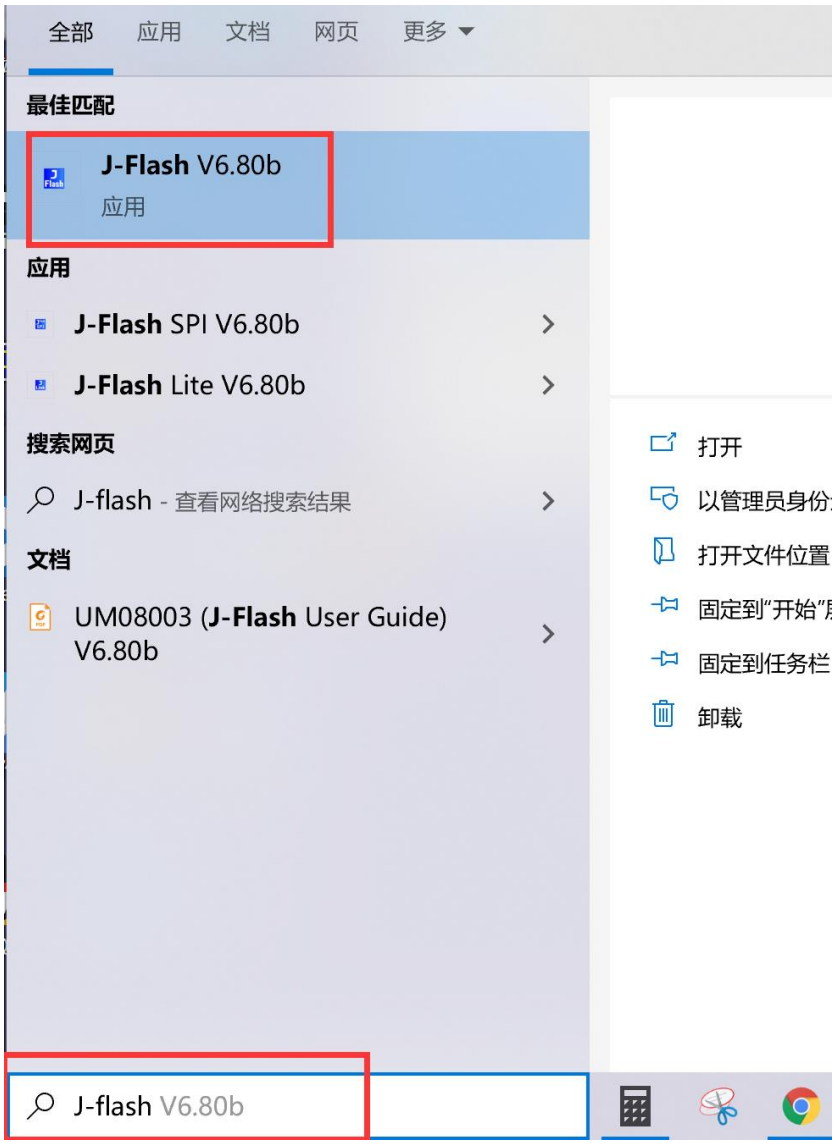


图 17-34 Windows 搜索栏

选择新建一个工程，然后 start 开始：

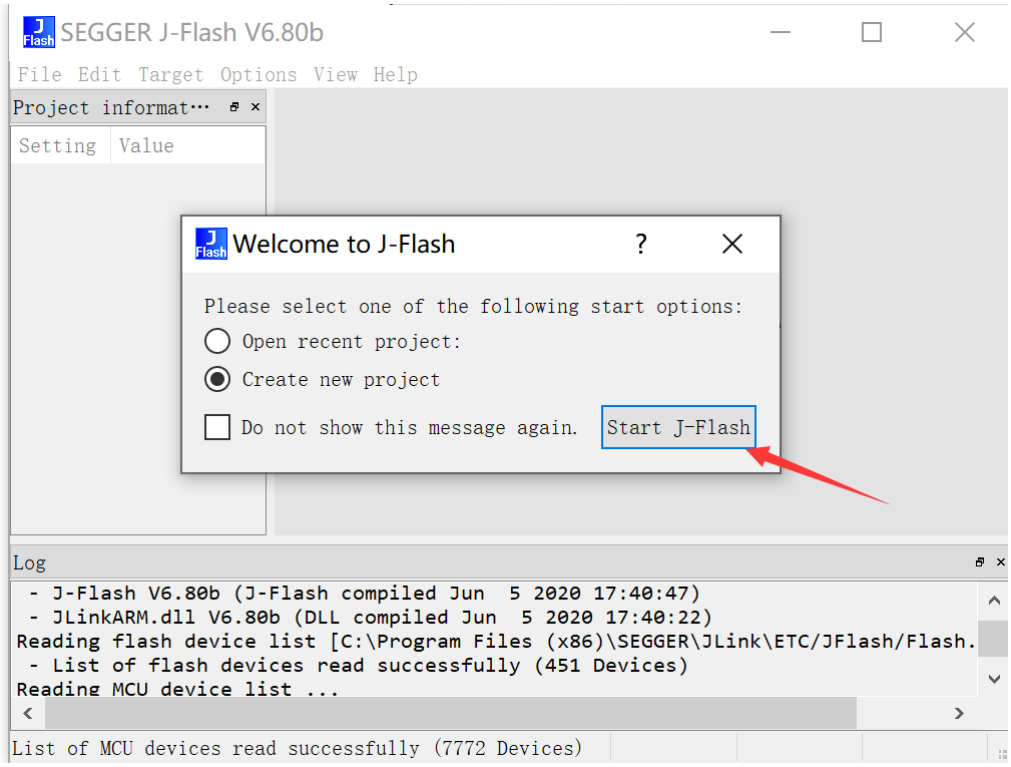


图 17-35

在‘Target device’点击按钮，弹出列表后在 Device 栏输入‘52840’，选择搜索到的芯片 nrf52840。然后点击 OK。

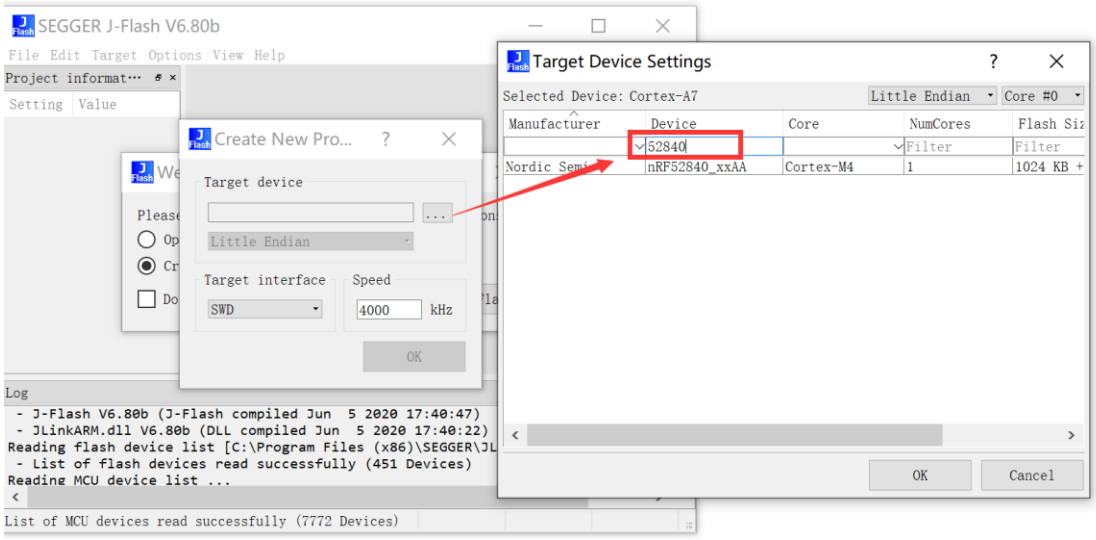


图 17-36 选择 NRF52840 芯片

接口选择“SWD”，速度默认。点击 OK。

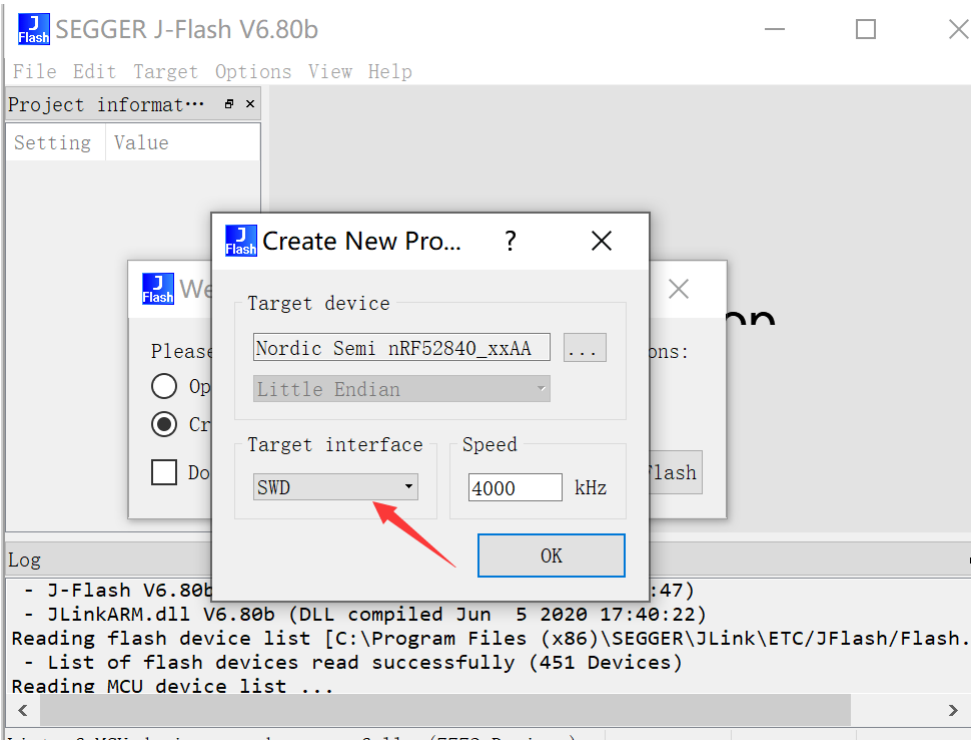


图 17-37 接口选择 SWD

这时候保证开发板已经通过 J-Link 连接到电脑，在软件中点击 Target – Connecct:

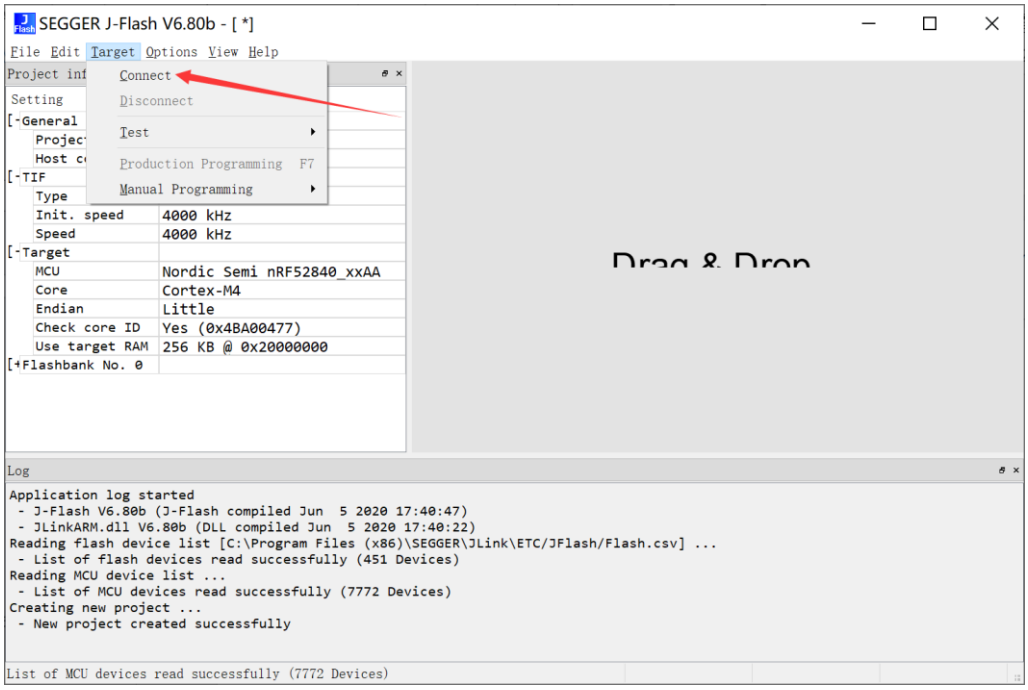


图 17-38 点击 Connct 连接

连接成功后在软件下方 log 出现“connected succcessfully”：（如果连接不成功，可能是供电问题，这时可以将 pyBLE 开发板插入 usb 线供电，然后再连接。）

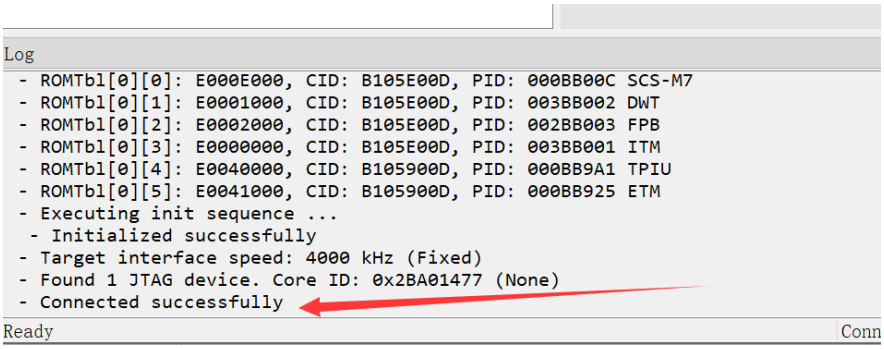


图 17-39 连接成功

接下来先刷除芯片，点击 Target – Manual Programming – Erase Chip :

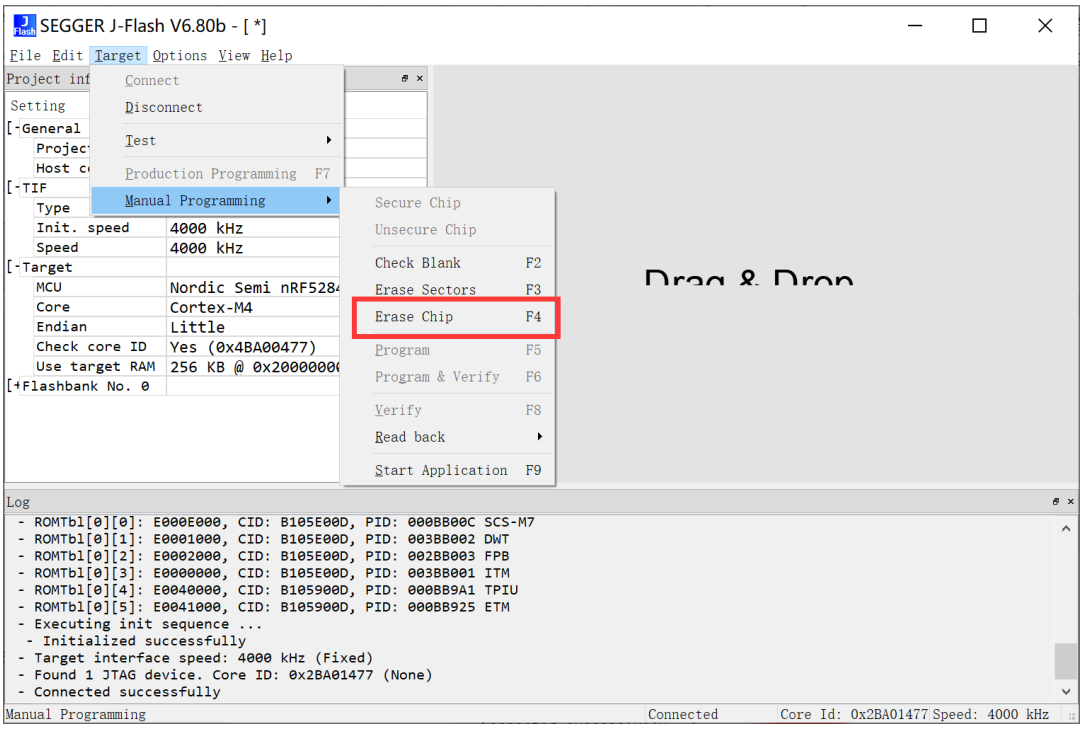


图 17-40 芯片擦除

擦除成功后会弹出提示 successfully ：

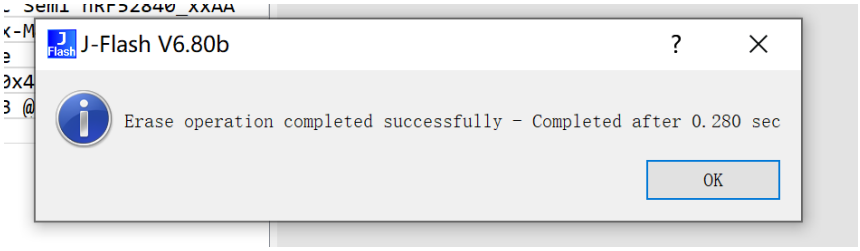


图 17-41 擦除成功

接下来打开要烧写的固件，固件位于：**零一科技（01Studio）MicroPython 开发套件配套资料_lastest\03-相关固件\04-pyBLE-NRF52840\02-Bootloader** 文件夹线下，固件可以通过在 File – Open data file.. 打开或者将文件直接拖动到右边灰色界面：**（由于 J-Flash 不支持中文路径，所以先建固件拷贝到桌面或者盘根目录再操作！）**

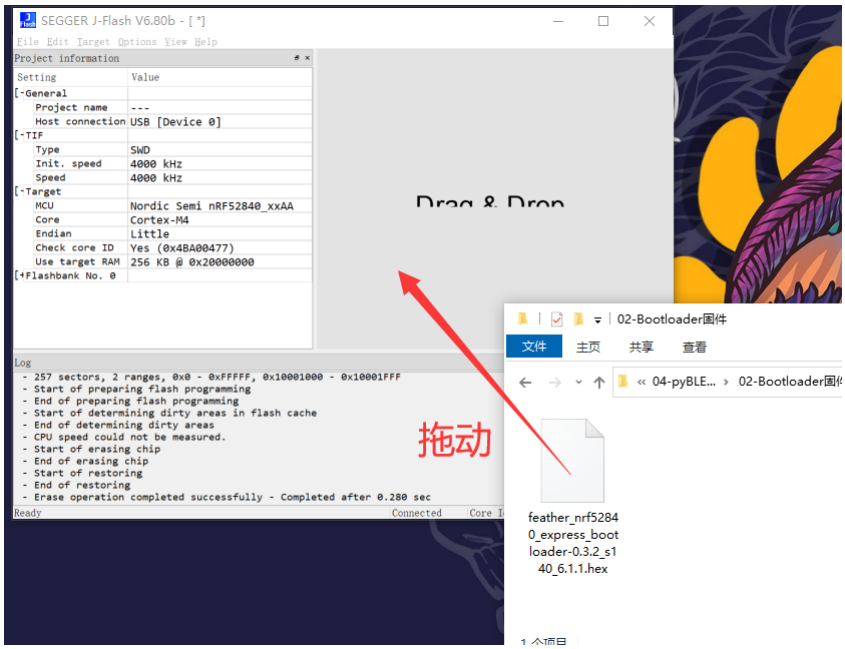


图 17-42 固件需要英文路径

成功打开后出现数据：

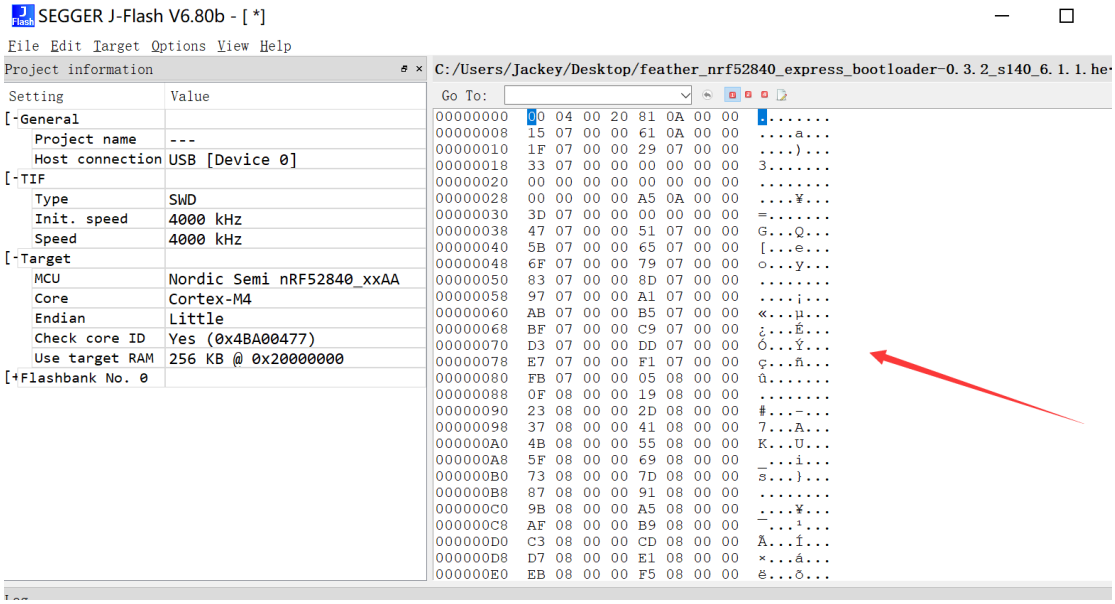


图 17-43 固件成功打开

然后点击 Target – Programming (F7) 烧录：

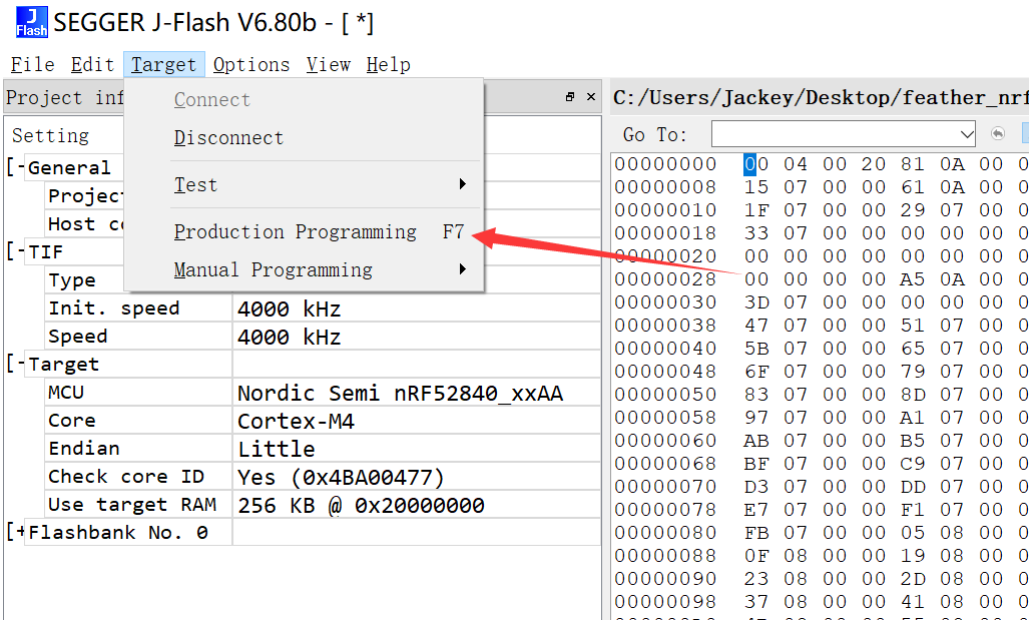


图 17-44 固件烧录

烧录成功后如下图所示：

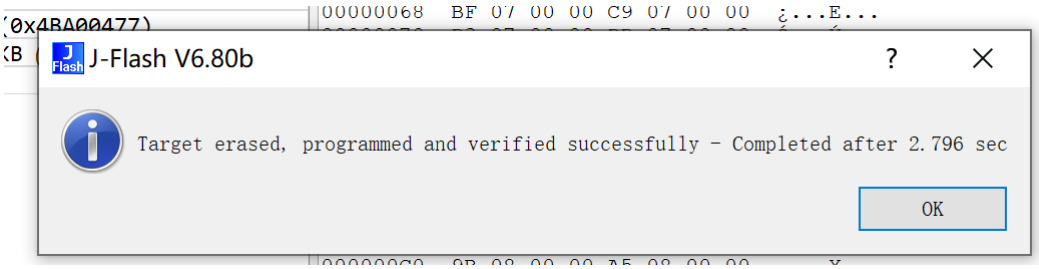


图 17-45 烧录成功

然后断开连接线，将 pyBLE-NRF52840 通过 usb 线连接到电脑，可以看到出现了 bootloader 盘符。

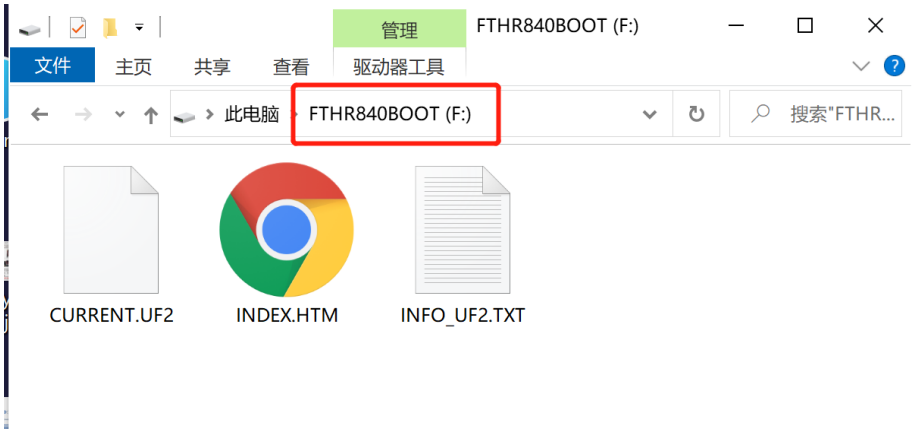


图 17-46 bootloader 盘符

然后将 **零一科技（01Studio）MicroPython 开发套件配套资料_lastest\03-相关固件\04-pyBLE-NRF52840\01-CircuitPython 固件** 里面的 uf2 固件文件直接拖动到 U 盘，即可自动更新。

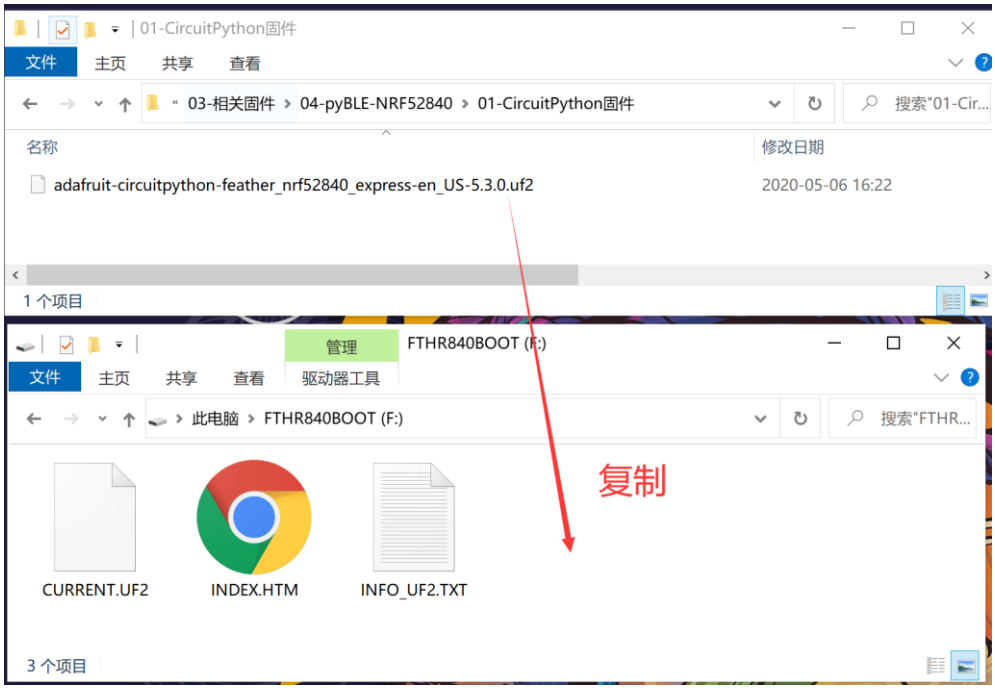


图 17-47 CircuitPython 版本升级

升级完成后自动重启（或者按下复位键），重新出现 CIRCUITPY U 盘，到这里重烧固件完成了。

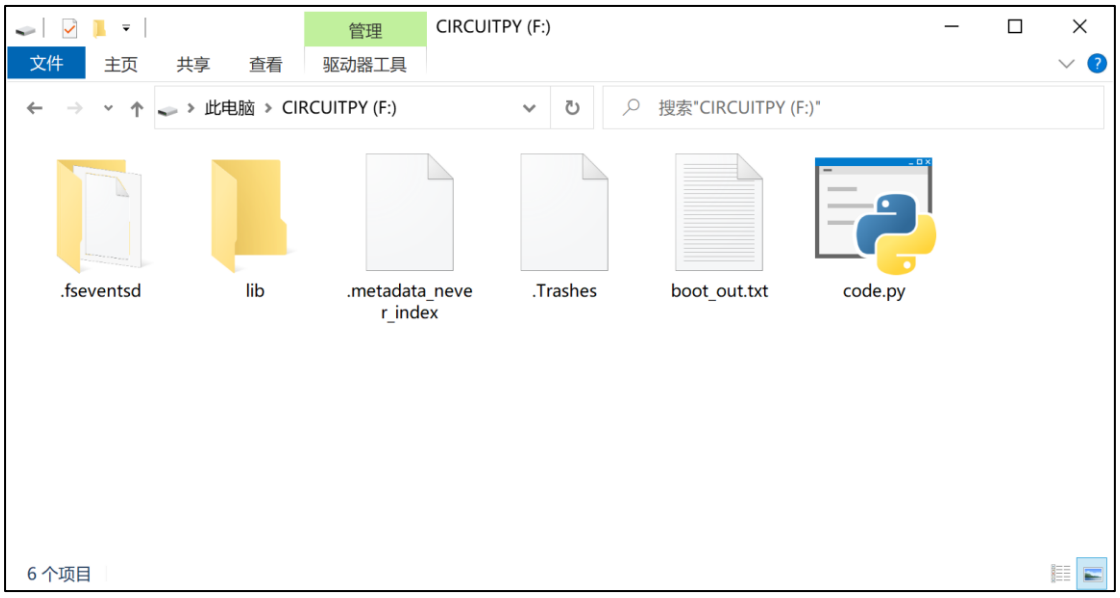


图 17-48 升级完成

17.2 基于 Mac OS

还记得当年乔布斯从大信封里面拿出 macbook 么，苹果和微软在 PC 时代走了两个极端，乔布斯主导下的苹果要求软硬一体化，封闭，务求最佳的用户体验；而微软则主打开发，让操作系统兼容不同的 PC 厂家，使得其 windows 一度占据了 90%的市场份额。从商业角度，盖茨毫无疑问是赢家，就个人而言，我更认可乔布斯的方式，现在的 win10 和微软自家的笔记本，某程度上都有一体化的思想。我用了半天的 Macbook 和 MAC 系统，如果不是要做硬件发，基本上可以放弃十多年的 windows 了。

17.2.1 安装 python3

在 MAC 系统下，安装 python3 方法也很简单，在官网 www.python.org 下载 Mac 版 python3，根据自己系统情况选择 32bit 或者 64bit 下载安装。

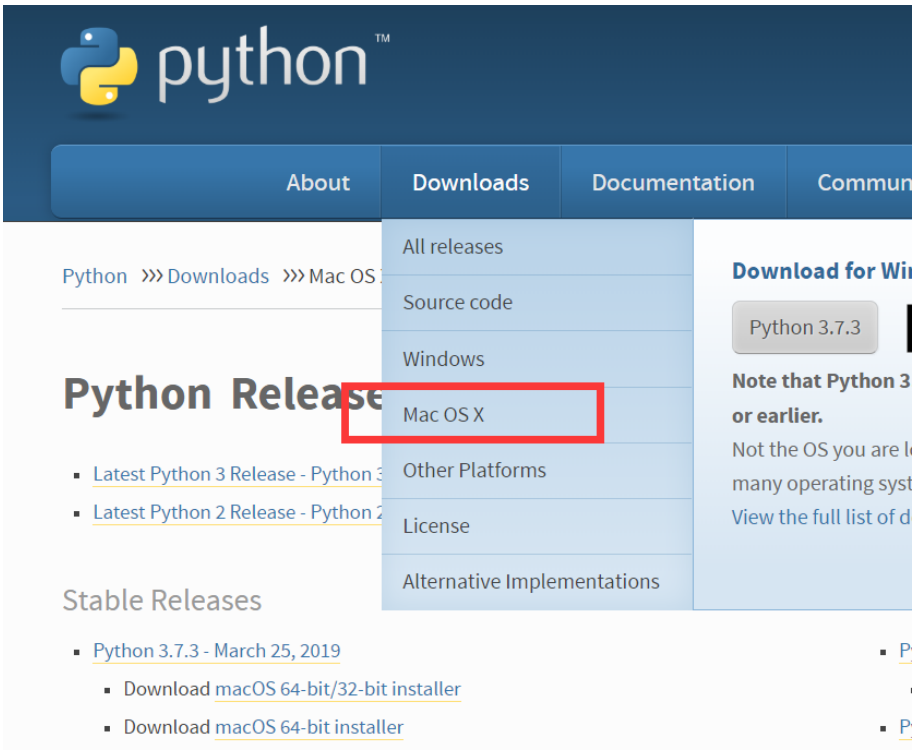


图 17-49

安装完成后在终端输入 python3，出现版本好的话说明安装成功。

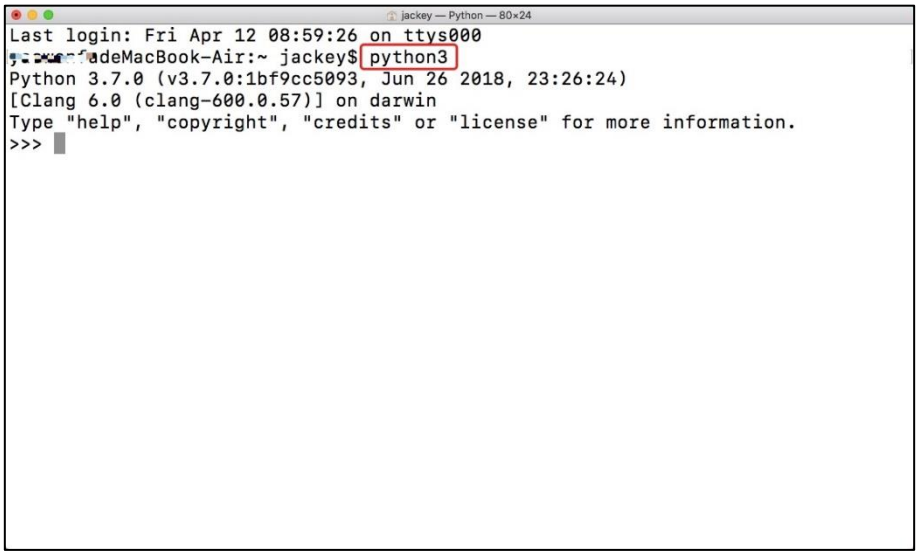


图 17-50 成功安装

17.2.2 安装开发软件 Mu

Mu 在 MAC OS 环境下的安装和使用方法和 Windows 一样，具体请参考：**17.1.2 安装开发软件 Mu** 章节内容，这里不再重复。

17.2.3 开发套件使用

请参考 **17.1.3 开发套件使用** 章节内容，这里不再重复。

17.3 基于 Linux

Linux 系统为大多工程师热衷的开源操作系统，一般内部都集成了 python3 和 IDE，用户直接使用即可。本节教程同样适用于树莓派的 Linux 系统。

17.3.1 开发软件 Mu 安装

安装方法请参考：https://codewith.mu/en/howto/1.0/install_raspberry_pi

其它内容请参考 **基于 Windows 平台。**

第18章　基础实验

MicroPython 更强调的是针对应用的学习，强大的底层库函数让我们可以直接关心功能的实现，也就是说我们只要理解和熟练相关的函数用法，就可以很好的玩转 MicroPython。它让我们可以做到不关心硬件和底层原理（当然有兴趣和能力的小伙伴可以深入研究）而直接跑起硬件。

基础实验是针对一些简单的实验学习讲解，可以让我们更快和更直接感受到 MicroPython 针对嵌入式开发的强大之处， 我后面的学习和开发夯实基础。

请先看实验讲解格式预览，每一节我们都会以以下形式讲解，图文并茂，务求达到快速理解和学习的作用：

（1） 前言：简单介绍这个实验；

（2） 实验平台：实验所用到的开发板、配件和接线说明；

（3） 实验目的：本实验要实现的功能；

（4） 实验讲解：对函数、代码、编程方法以及实验的详细讲解；

（5） 实验结果：记录程序下载到开发板上的图片示例；

（6） 总结:对实验进行总结。

18.1 点亮第一个 LED

- **前言：**

相信大部分人开始学习嵌入式单片机编程都会从点亮 LED 开始，我们 MicroPython 的学习也不例外，通过点亮第一个 LED 能让你对编译环境和程序架构有一定的认识，为以后的学习和更大型的程序打下基础，增加信心。

- **实验平台：**

pyBLE-NRF52840 或 pyBLE-NRF52840 开发套件。



图 18-1 pyBoard v1.1-CN

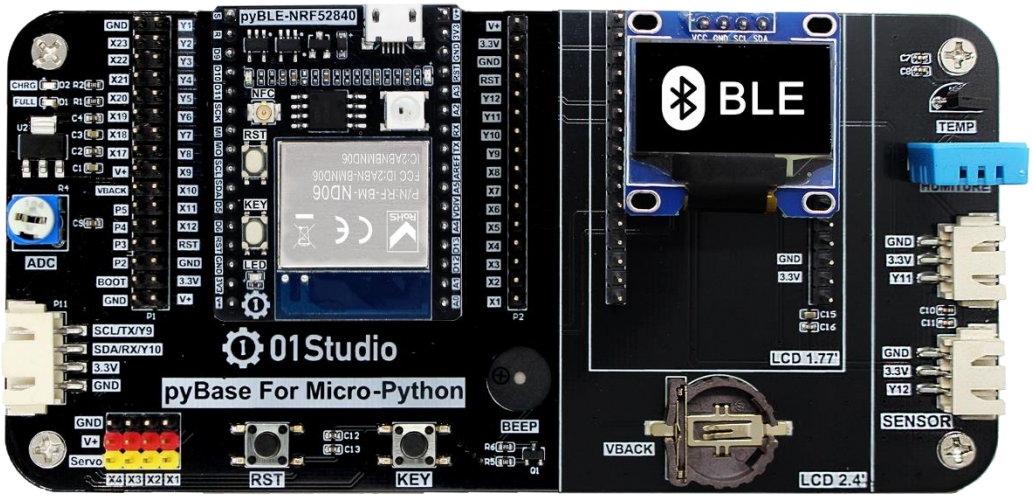


图 18-2 MicroPython 开发套件

- **实验目的：**

  点亮 LED（蓝灯）。

- **实验讲解：**

  pyBLE-NRF52840 上通过 IO 控制的 LED 总共有 2 个 LED，分别是 LED1(蓝灯)和 LED2（红灯）。

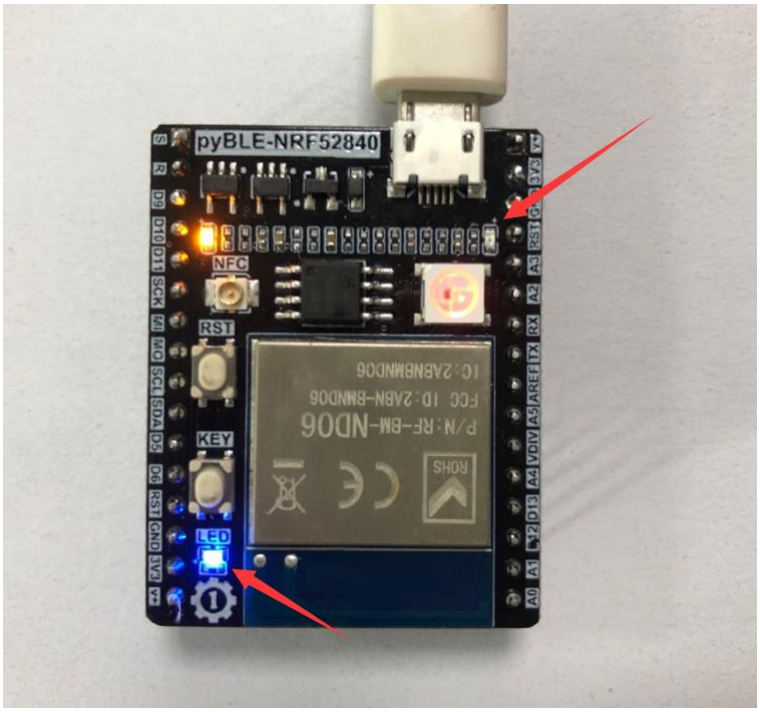


图 18-3 LED 灯

从下面原理图可以看到，通过 IO 口输出高电平点亮 LED。

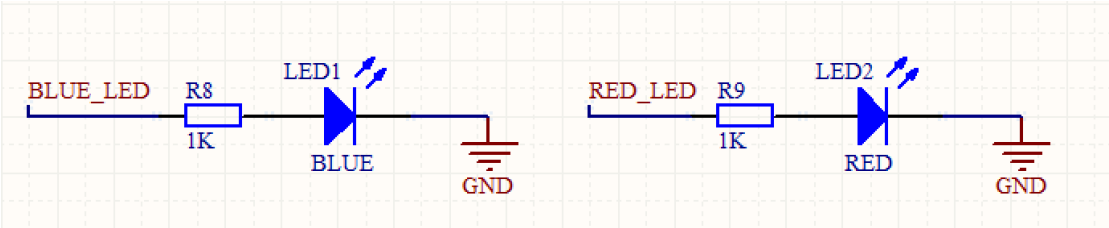


图 18-4

LED 的控制主要还是 GPIO 的控制，pyBLE-NRF52840 开发板 GPIO 引脚是被定义好的，存放在 board 模块下，我们可以使用 REPL 来查看开发板的引脚，在 REPL 中输入下面指令：

```
import board

board.
```

上面输入 board.后按下键盘的 tab 键补全，就可以看到所有的 IO 口定义，这个方法也可以用于看某个库模块的函数用法。程序可以通过诸如 board.A1 、board.BLUE_LED 方式来指定引脚。

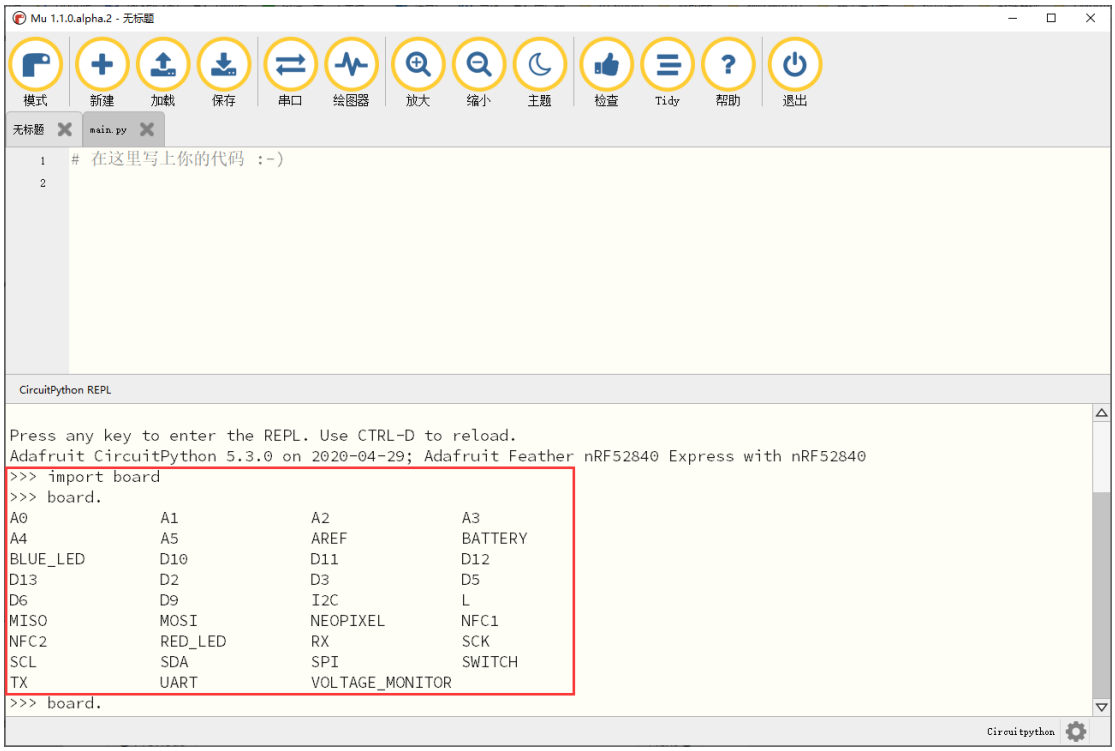


图 18-5 引脚定义

在 CircuitPython 中可以直接使用 digitalio（数字 IO）模块编程实现 IO 输出从而点亮 LED 功能。具体介绍如下：

构造函数
led=digitalio.DigitalInOut (pin)
led(GPIO)构造函数； 【pin】开发板的引脚，例：board.D5

使用方法
led.direction = value
引脚方向定义。 **value 匹配值如下：** 【digitalio.Direction.INPUT】输入； 【digitalio.Direction.OUTPUT】输出；
led.pull = value
上下拉电阻。 **value** 匹配值如下： 【digitalio.Pull.UP】上拉； 【digitalio.Pull.DOWN】下拉。
led.drive_mode = value
输出方式。 **value** 匹配值如下： 【digitalio.DriveMode.PUSH_PULL】推挽输出； 【digitalio.DriveMode.OPEN_DRAIN】漏极开路。
led.value = value
IO 电平 value 匹配值如下： 【True】高电平 【False】低电平
更多有关引脚和功能注册信息请看官方文档： https://circuitpython.readthedocs.io/en/5.0.x/shared-bindings/digitalio/DigitalInOut.html

表 18-1 GPIO 对象

上表对 CircuitPython 的 DigitalInOut 对象做了详细的说明，digitalio 是大模块，DigitalInOut、Direction、Pull、DriveMode 是 digitalio 下面的其中一个小模块，在 python 编程里有两种方式引用相关模块:

方式 1 是：import digitalio，然后通过 digitalio. DigitalInOut 来操作；

方式 2 是：from digitalio import DigitalInOut,意思是直接从 digitalio 中引入 DigitalInOut 模块，然后直接通过 DigitalInOut 来操作。显然方式 2 会显得更直观和方便，本实验也是使用方式 2 来编程。代码编写流程如下：

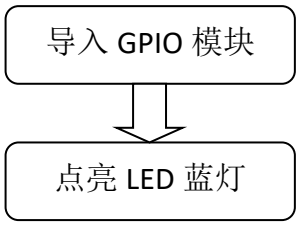


图 18-6 代码编写流程

以下是本实验示例代码：**（本教程所有例程代码都可以在配套资料包--示例程序文件夹下找到。）**

```
'''
实验名称：点亮 LED 蓝灯
版本：v1.0
日期：2020.4
作者：01Studio
'''
#导入相关模块
import board
from digitalio import DigitalInOut, Direction, Pull

#构建 LED 对象和初始化
led = DigitalInOut(board.BLUE_LED) #定义引脚编号
led.direction = Direction.OUTPUT  #IO 为输出

led.value = True #输出高电平，点亮 LED
```

```
#阻塞 IO，让程序保持运行
while True:
    pass
```

运行程序有 2 个方法：

1、直接在 Mu 软件中编写 CIRCUITPY U 盘里面的 main.py 文件，点击“保存”后开发板自动运行新程序。

2、将编写好的 main.py 文件拷贝到 CIRCUITPY U 盘里面，按下 RST 复位键开发板重新运行新程序。

- **实验结果：**

LED 蓝灯被点亮。

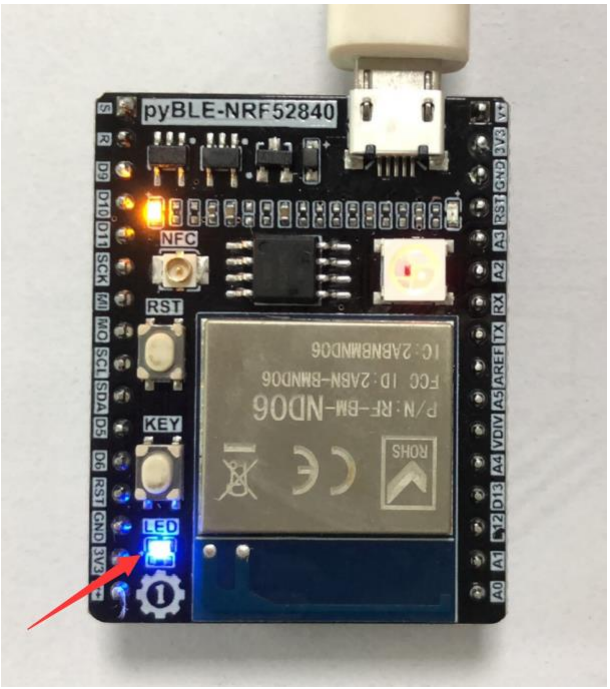


图 18-7 点亮 LED

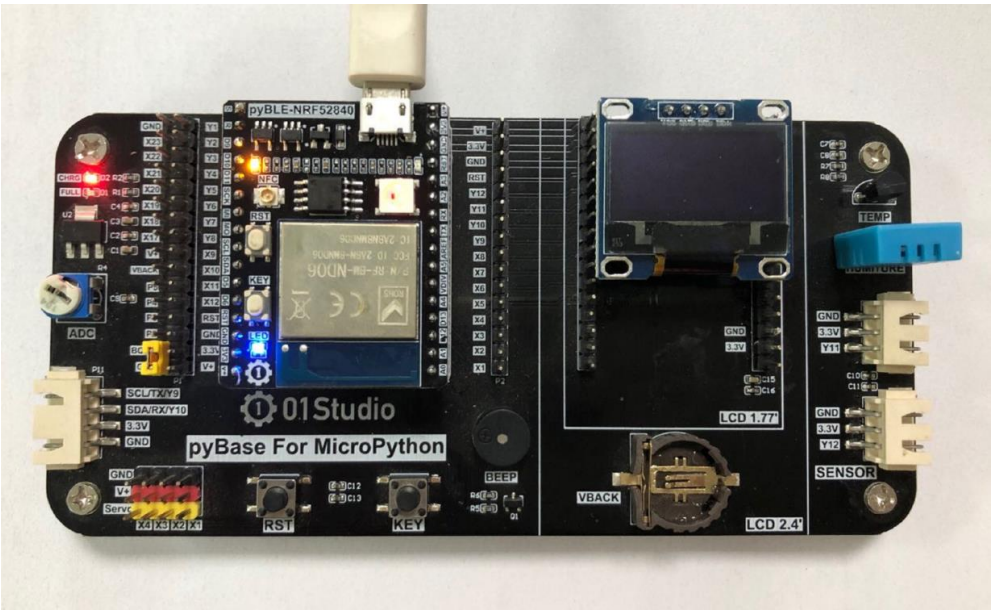


图 18-8 基于 MicroPython 开发套件

- **总结：**

从第一个实验我们可以看到，使用 CircuitPython (MicroPython)开发关键是要学会构造函数和其使用方法，便可完成对相关对象的操作，在强大的模块函数支持下，实验只用了简单的几行行代码便实现了点亮 LED 灯。无需关注单片机底层寄存器等相关配置。这让编程变得更简单，让开发变得更高效！

18.2 按键

- **前言：**

按键是最简单也最常见的输入设备，很多产品都离不开按键，包括早期的 iphone，今天我们就来学习一下如何使用 MicroPython 来编写按键程序。有了按键输入功能，我们就可以做很多好玩的东西了。

- **实验平台：**

pyBLE-NRF52840。

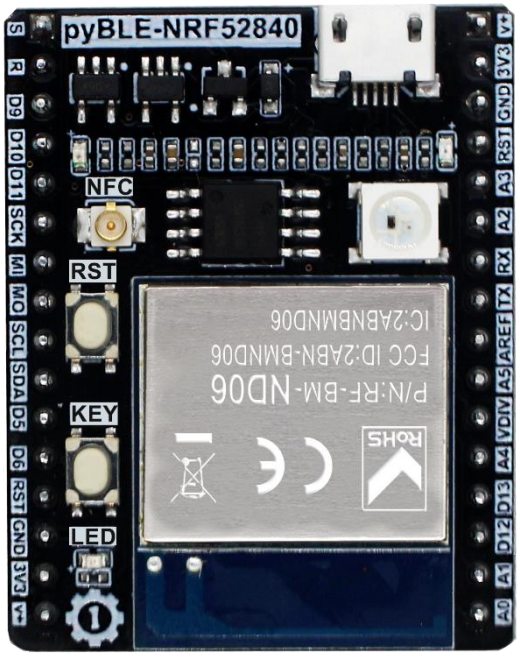


图 18-9

- **实验目的：**

学习 CircuitPython（MicroPython）编程实现按键输入检测。

- **实验讲解：**

我们先来看看 pyBLE-NRF52840 的原理图，看看按键对应的外部 IO 引脚。开发板一共有 2 个按键，其中一个是复位 RST，而功能按键名称为 SWITCH。

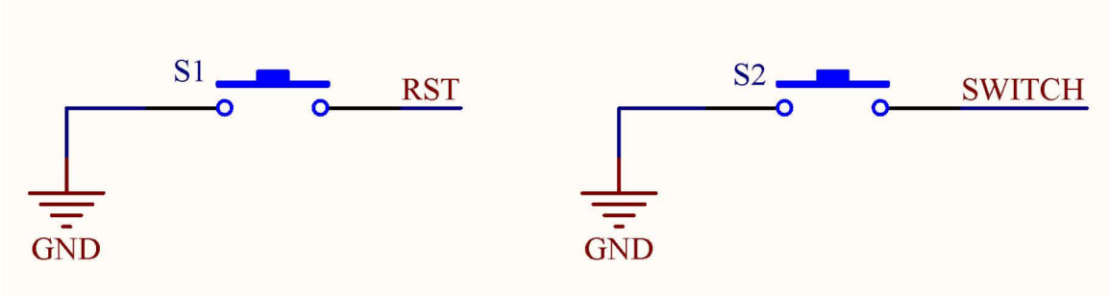


图 18-10 按键引脚图

从原理图可以看到，按键 SWITCH 的一端连接到 NRF52840 的 IO 口，另一端连接到 GND。所以按键在没按下时候输入高电平（1），按下时候输入低电平（0）。

和 LED 一样，按键的输入检测也是用到 GPIO 对象模块，但方式为输入，让我们回顾一下上一节 GPIO 对象：

构造函数
led=digitalio.DigitalInOut (pin)
led(GPIO)构造函数； 【pin】开发板的引脚，例：board.D5
使用方法
led.direction = value
引脚方向定义。 **value 匹配值如下：** 【digitalio.Direction.INPUT】输入； 【digitalio.Direction.OUTPUT】输出；
led.pull = value
上下拉电阻。 **value** 匹配值如下： 【digitalio.Pull.UP】上拉； 【digitalio.Pull.DOWN】下拉。
led.drive_mode = value
输出方式。 **value** 匹配值如下：

【digitalio.DriveMode.PUSH_PULL】推挽输出； 【digitalio.DriveMode.OPEN_DRAIN】漏极开路。
led.value = value
IO 电平 value 匹配值如下： 【True】高电平 【False】低电平
更多有关引脚和功能注册信息请看官方文档： https://circuitpython.readthedocs.io/en/5.0.x/shared-bindings/digitalio/DigitalInOut.html

表 18-2　GPIO 对象

GPIO 对象使用非常简单，我们将按键即 SWITCH 引脚配置成输入，实现当检测到按键被按下时候点亮 LED 蓝灯，松开时关闭 LED 蓝灯来做指示。代码编写流程如下：

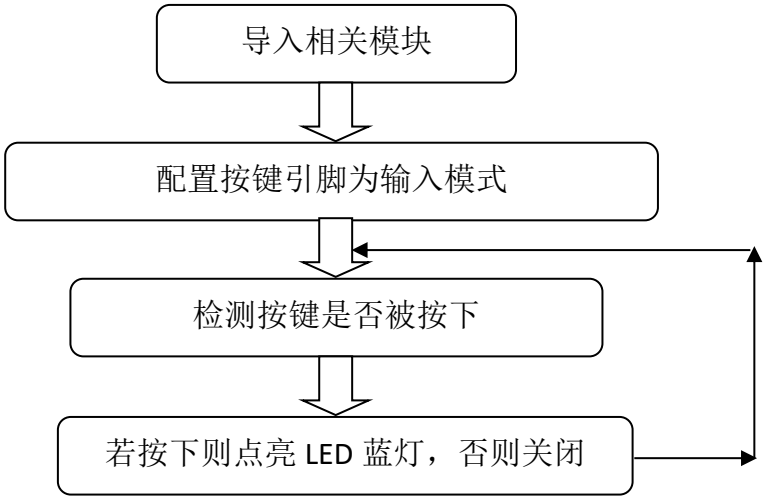


图 18-11 代码编写流程

参考代码如下：

```
'''
实验名称：按键（GPIO）
版本：v1.0
```

```
日期：2020.4
作者：01Studio
'''

#导入相关模块
import board
from digitalio import DigitalInOut, Direction, Pull

#构建 LED 对象和初始化
led = DigitalInOut(board.BLUE_LED) #定义引脚编号
led.direction = Direction.OUTPUT  #IO 为输出

#构建按键对象和初始化
switch = DigitalInOut(board.SWITCH) #定义引脚编号
switch.direction = Direction.INPUT #IO 为输入
switch.pull = Pull.UP #打开上拉电阻

while True:

    if switch.value == 0: #按键被按下
        led.value = True #点亮 LED

    else:
        led.value = False #关闭 LED
```

- **实验结果：**

运行代码，可以看到当按键被按下时候，LED 蓝灯点亮，松开时熄灭。按键可以按下 pyBLE-NRF52840 的按键或者 pyBase 开发底板上的 KEY 按键。

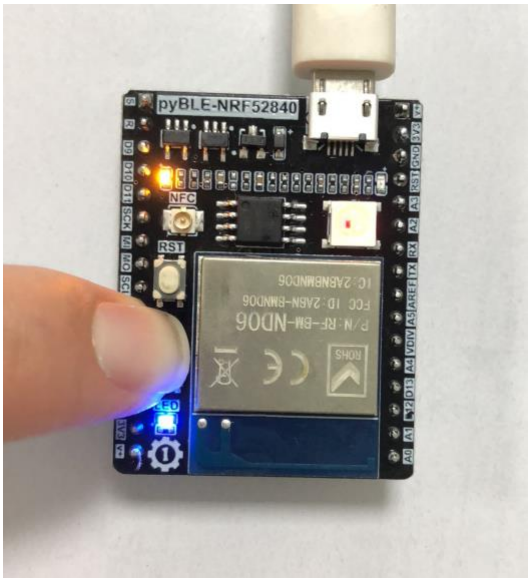


图 18-12 按键实验

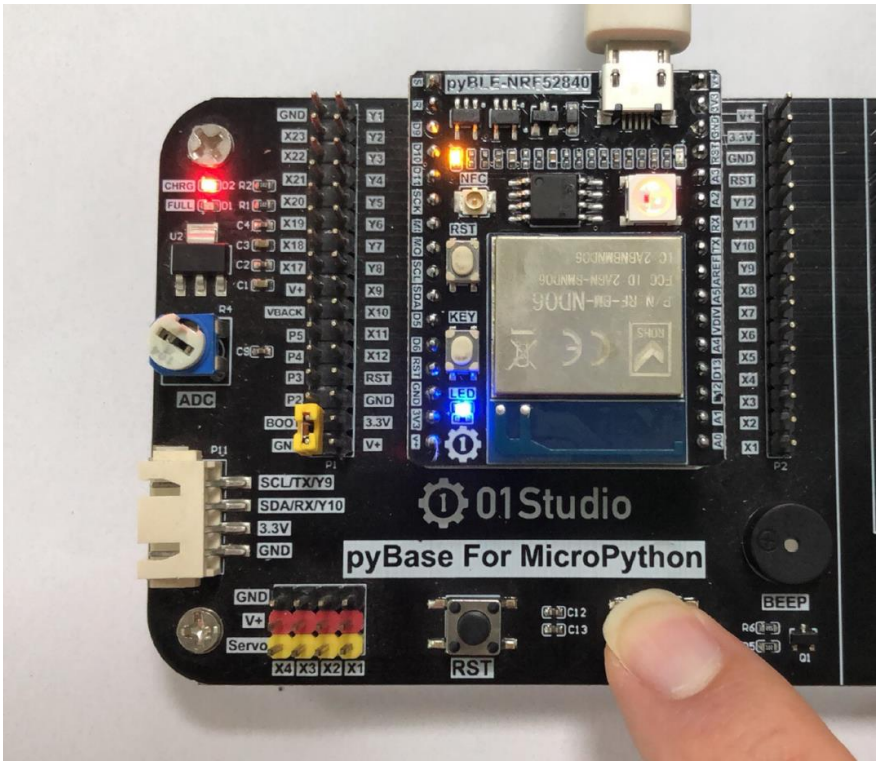


图 18-13 pyBase 上的按键

- **总结：**

GPIO 是非常通用的功能，学会了 GPIO，就可以把开发板所有的引脚为自己所用，灵活性很强。

18.3 ADC

- **前言：**

ADC(analog to digital conversion) 模拟数字转换。意思就是将模拟信号转化成数字信号，由于单片机只能识别二级制数字，所以外界模拟信号常常会通过 ADC 转换成其可以识别的数字信息。常见的应用就是将变化的电压转成数字信号。

- **实验平台：**

pyBLE-NRF52840 开发套件。

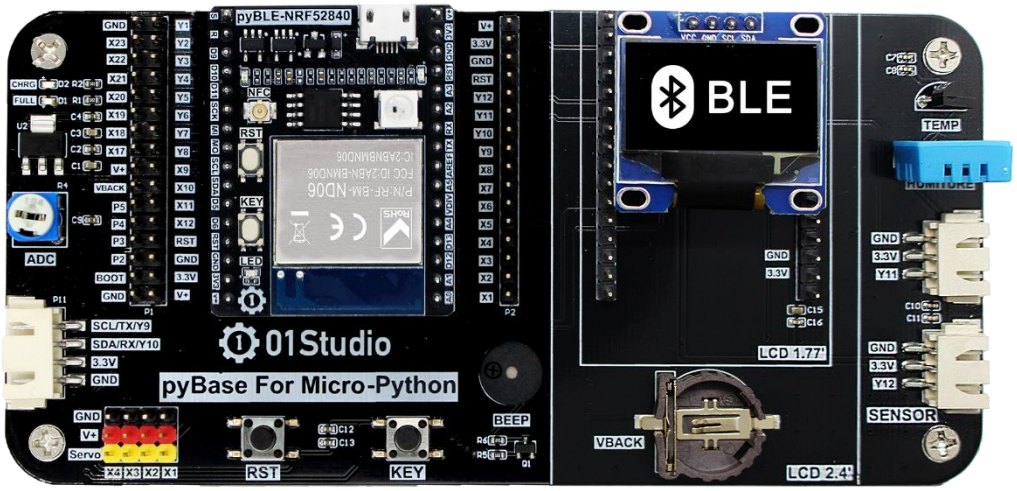


图 18-14 pyBLE-NRF52840 开发套件

- **实验目的：**

通过编程调用 MicroPython 编程，实现测量 0-3.3V 电压，并显示到屏幕上。

- **实验讲解：**

pyBLE-NRF52840 的 A5 引脚连接到了 pyBase 的电位器(X7)，通过电位器的调节可以使得 A5 引脚上的电压变化范围实现从 0-3.3V。

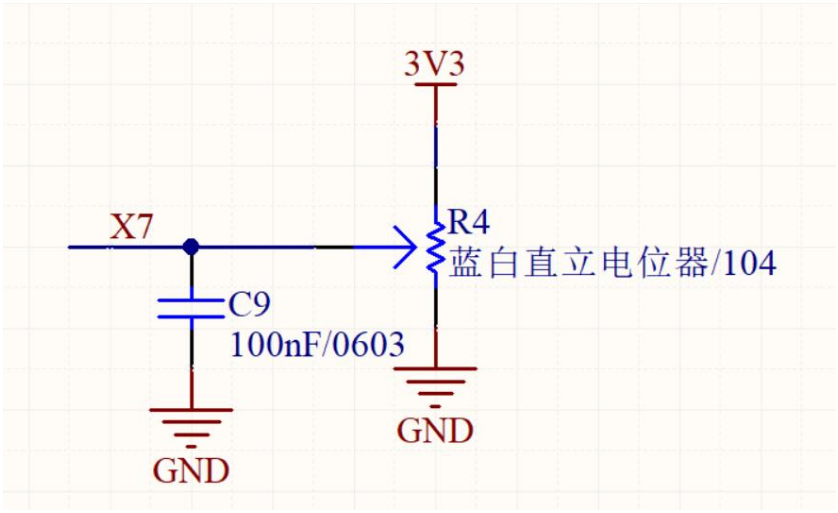


图 18-15 电位器原理图

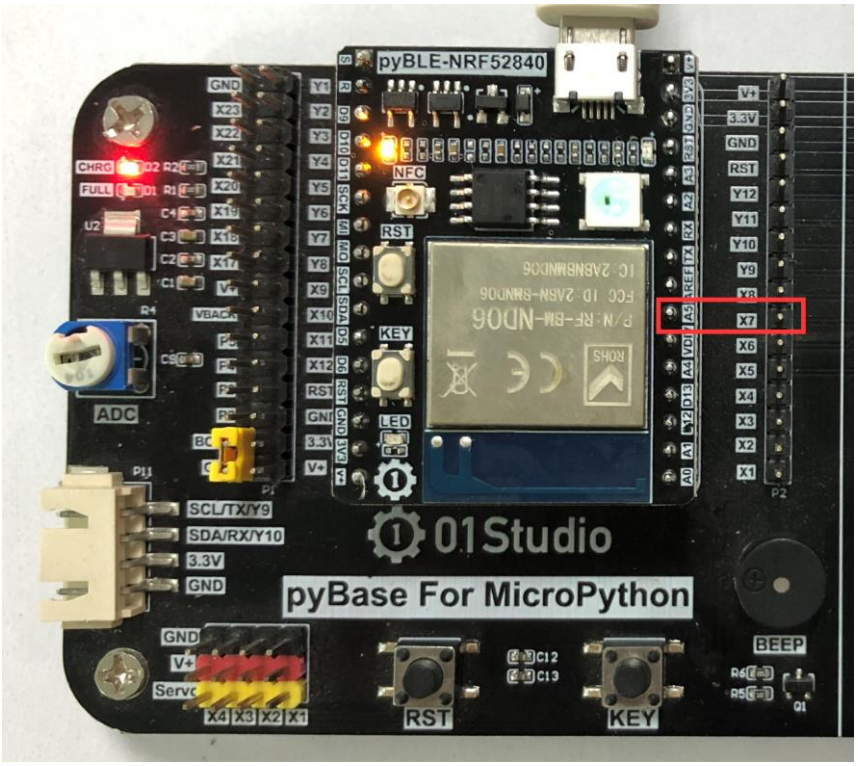


图 18-16 A5 与 pyBase 底板 X7 相连接

在了解了原理图后，我们来看看内置 ADC 模块的函数使用方法。

构造函数
adc=analogio.AnalogIn (pin)
ADC 构造函数； 【pin】开发板的引脚，例：board.A5
使用方法
adc.value
电压值。 返回 0-65535，16 位精度的 ADC（2^16-1=65535）。

表 18-3　ADC 对象

你没看错，就这么简单。两句函数就可以获得 AD 数值，让我们来理顺一下编程逻辑。先导入相关模块，然后初始化模块。在循环中不断读取 ADC 的值，通过串口打印，每隔 100 毫秒读取一次，具体如下：

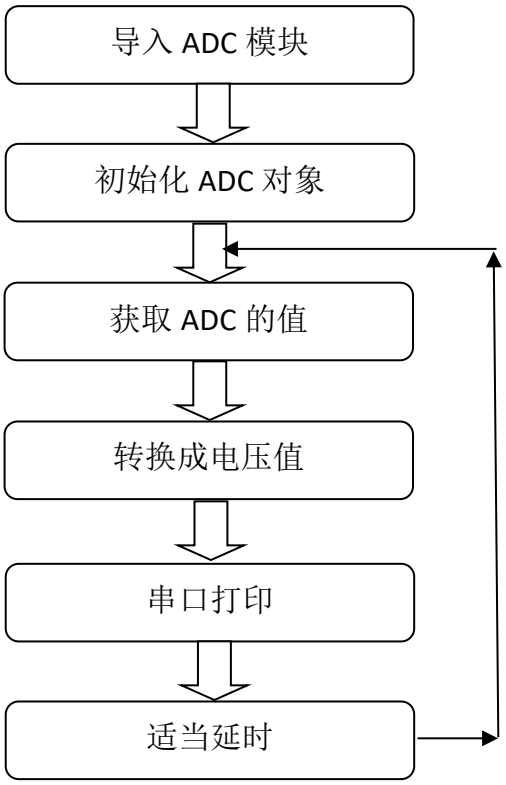


图 18-17 代码编写流程

实验参考代码：

```
'''
实验名称：ADC
版本：v1.0
日期：2020.4
说明：0-3.3V 电压测量
www.01studio.org
'''

#导入相关库
import time
import board
from analogio import AnalogIn

adc = AnalogIn(board.A5) #输入 IO

while True:
    #输出电压转换成 0-3.3V，保留 2 位小数；精度 16bit。
    print(round(adc.value*3.3/65535,2))
    time.sleep(0.1) #测量周期 100ms
```

- **实验结果：**

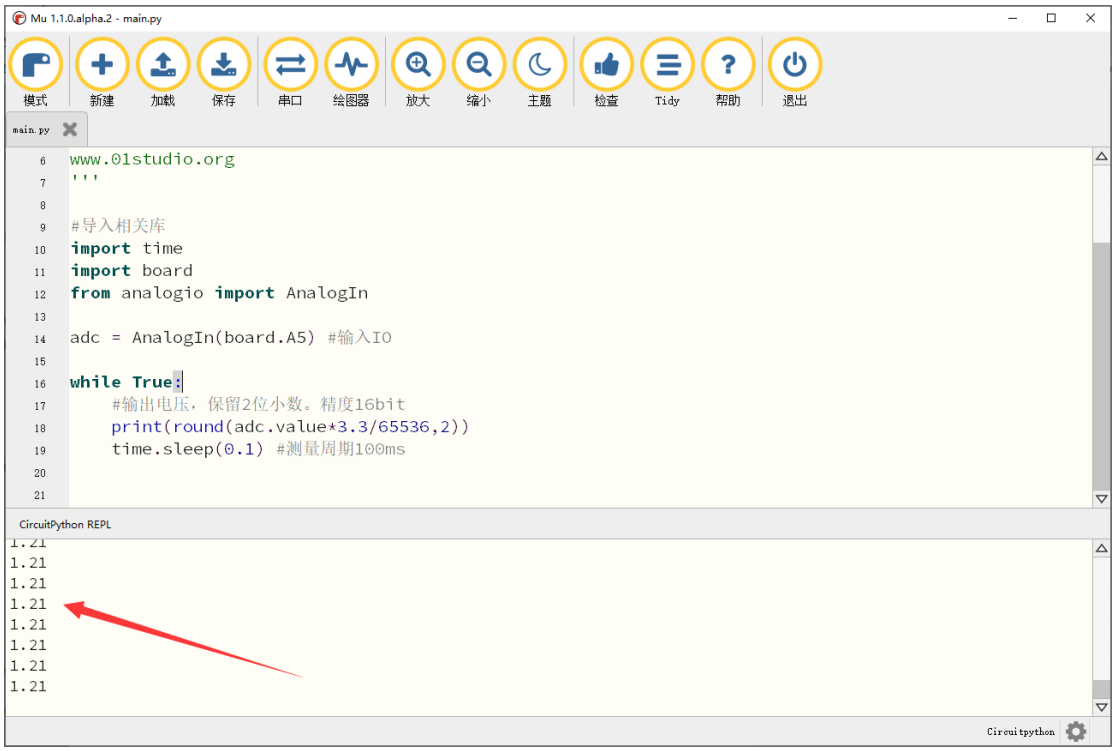


图 18-18 串口打印电压值

通过调节电位器，可以发现电压在不断变化。

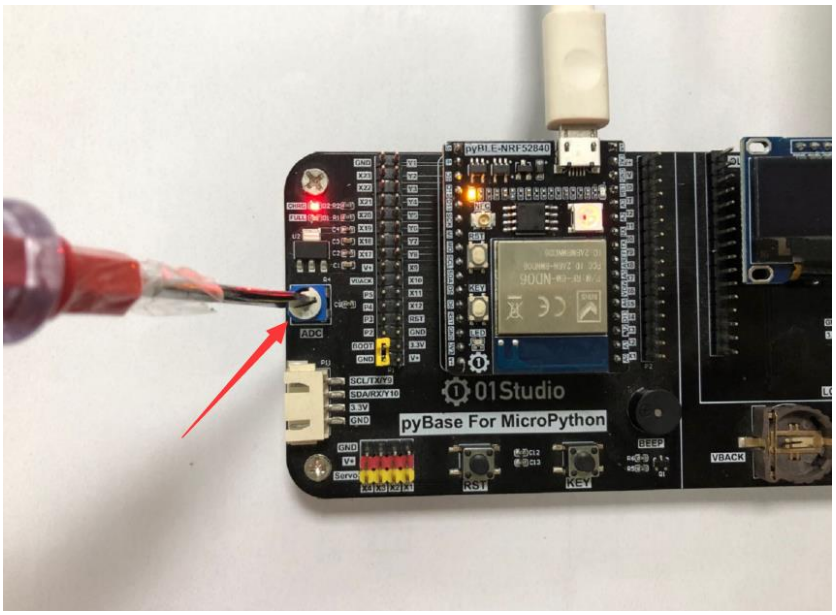


图 18-19 调节电位器来改变电压输入

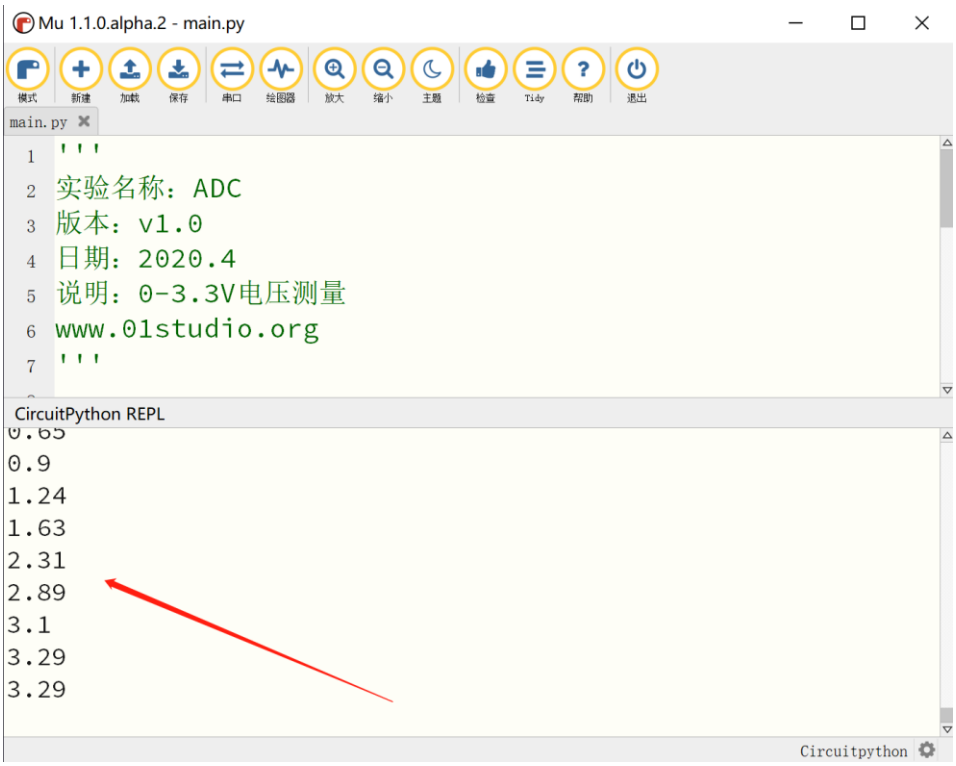


图 18-20 电压测量值变化

- **总结：**

这一节我们学习了 ADC 的应用。主要用于电压的检测，开发板上有很多 ADC 引脚，这意味着可以实现多路同时检测。有兴趣的小伙伴可以自行研究。

18.4 PWM

- **前言：**

PWM（脉冲宽度调制）就是一个特定信号输出，主要用于输出不同频率、占空比（一个周期内高电平出现时间占总时间比例）的方波。以实现固定频率或平均电压输出。

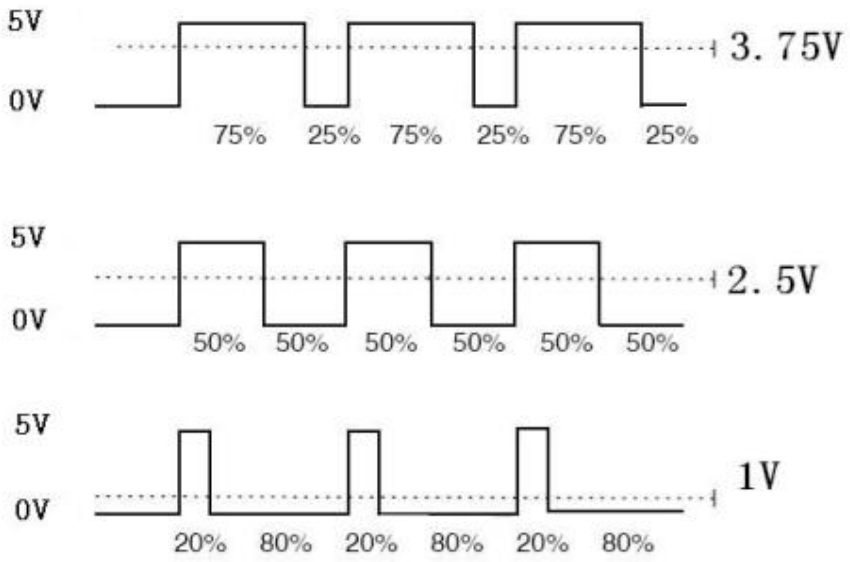


图 18-21 PWM 波形示例（右边为等效电压）

- **实验平台：**

pyBLE-NRF52840 开发套件。

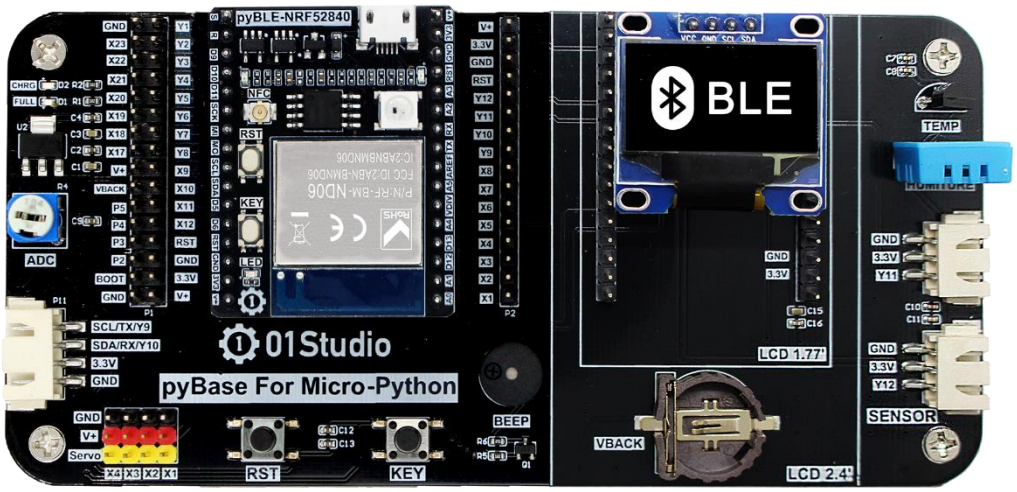


图 18-22

- **实验目的：**

通过不同频率的 PWM 信号输出，驱动无源蜂鸣器发出不同频率的声音。

- **实验讲解：**

蜂鸣器分有源蜂鸣器和无源蜂鸣器，有源蜂鸣器的使用方式非常简单，只需要接上电源，蜂鸣器就发声，断开电源就停止发声。而本实验用到的无源蜂鸣器，是需要给定指定的频率，才能发声的，而且可以通过改变频率来改变蜂鸣器的发声音色，以此来判定 pyBLE-NRF52840 的 PWM 输出频率是在变化的。

pyBase 开发底板上的无源蜂鸣器连接到引脚 X5。如下图所示：

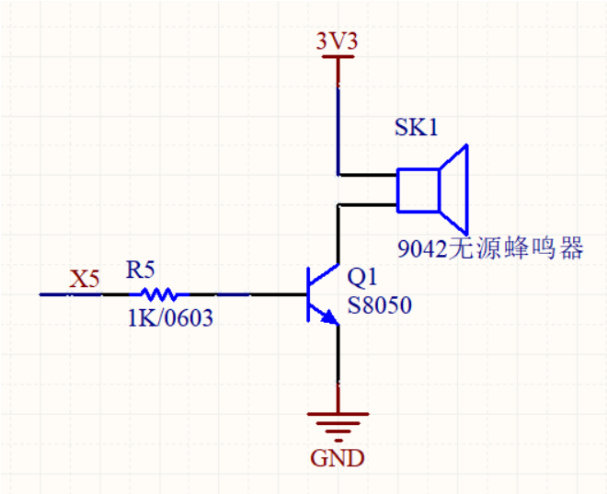


图 18-23 蜂鸣器原理图

结合资料包的 pyBLE-NRF52840 和 pybase 底板原理图可以看到，pyBLE-NRF52840 连接到底板蜂鸣器的是 A4 引脚。

我们来看看 PWM 模块对象：

构造函数
PWM = pulseio.PWMOut(pin, duty_cycle=0, frequency=500, variable_frequency = False)
构建 PWM，PWMOut 对象在 pulseio 模块下。 【pin】引脚。如 board.A4 【duty_cycle】占空比。0-65536，如 32768 则为占空比 50%。 【frequency】输出频率 【variable_frequency】允许更改频率。默认 False

使用方法
PWM.duty_cycle = value
设置占空比。范围：0-65536，如 value=32768 则为占空比 50%。
PWM.frequency = value
设置占空比。不传参数返回当前占空比值。[0-100]表示占空比百分比
PWM.deinit()
注销 PWM。释放 IO

表 18-4　PWM 对象

无源蜂鸣器我们可以用特定频率的方波来驱动，方波的原理很简单，就是一定频率的高低电平转换，可以简单理解成占空比为 50%的 PWM 输出。

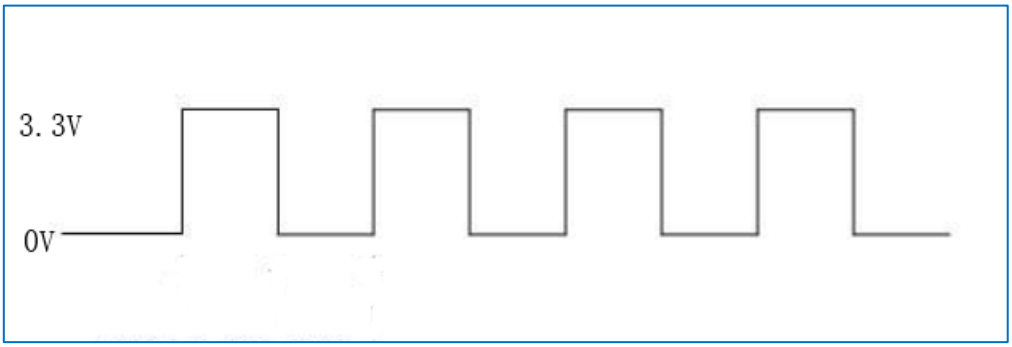


图 18-24　方波信号

结合上述讲解，总结出代码编写流程图如下：

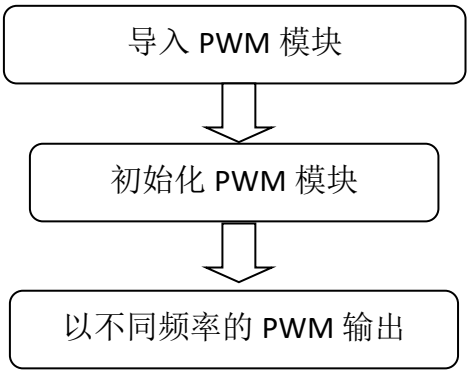


图 18-25 代码编写流程图

实验参考代码：

```
'''
实验名称：PWM
版本：v1.0
日期：2020.4
作者：01Studio
说明：输出 PWM 控制蜂鸣器发出不同频率声音
'''

#导入相关模块
import board
from pulseio import PWMOut
import time

#PWM 构建，蜂鸣器引脚 A4
PWM = PWMOut(board.A4,
duty_cycle=32768,frequency=200,variable_frequency=True)

#循环发出不同频率响声。
while True:

    PWM.frequency = 200
    time.sleep(1)

    PWM.frequency = 400
    time.sleep(1)

    PWM.frequency = 600
    time.sleep(1)
```

```
PWM.frequency = 800
time.sleep(1)

PWM.frequency = 1000
time.sleep(1)
```

- **实验结果：**

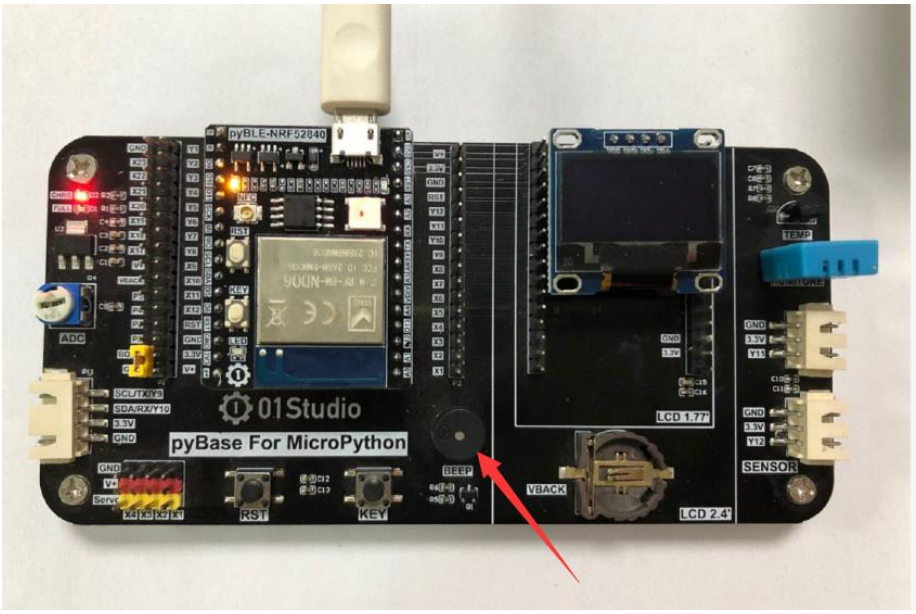


图 18-26

有条件的朋友可以使用示波器测量核心板的 A4 引脚或 pyBase 的 X5 引脚，观察信号波形的变化：

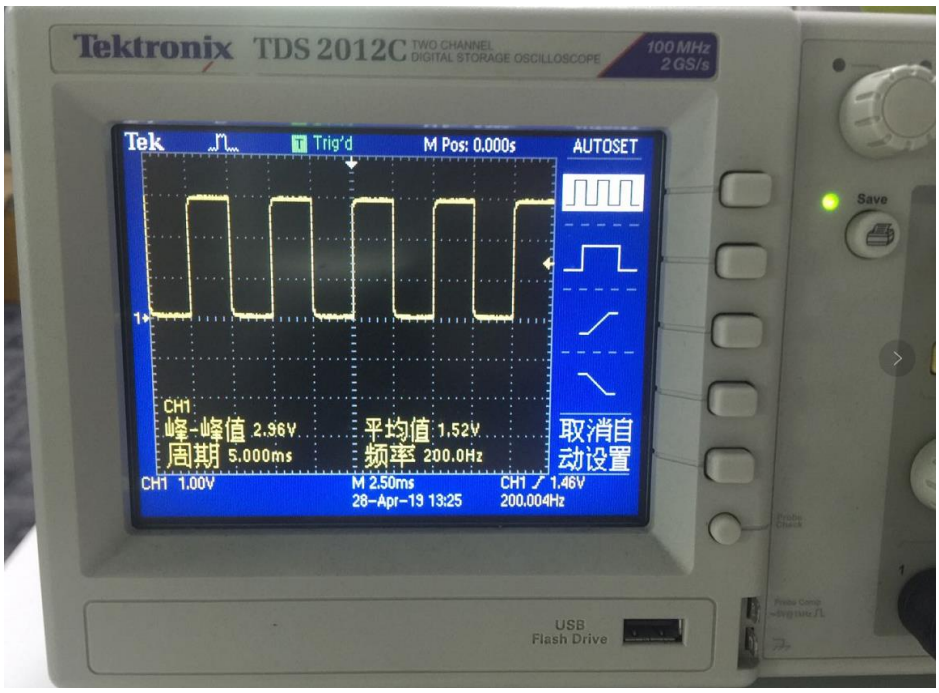


图 18-27　示波器显示波形

- **总结：**

到了这一节，我们发现实验中对象函数使用方法非常简单，这是好事，让我们可以将更多精力放在应用上，做出更多好玩的创意。而不需要过多的关注复杂的底层代码开发。而随着要实现功能的复杂化让编程的代码数量变多，逻辑也将略显复杂。

18.5 I2C 总线（OLED 显示屏）

- **前言：**

前面学习了按键输入设备后，这一节我们来学习输出设备 OLED 显示屏，其实之前的 LED 灯也算是输出设备，因为它们确切地告诉了我们硬件的状态。只是相对于只有亮灭的 LED 而言，显示屏可以显示更多的信息，体验更好。

本章节的 OLED 显示屏学习，实际上是在使用 I2C 的总线接口，开发板是通过 I2C 总线通讯方式与 OLED 显示屏通讯的。这章稍微复杂一点，但我们把它放在了前面来学习，是因为学会了显示屏的使用，那么在后面的实验中可玩性就更强了。

- **实验平台：**

PyBLE-NRF52840 开发套件。

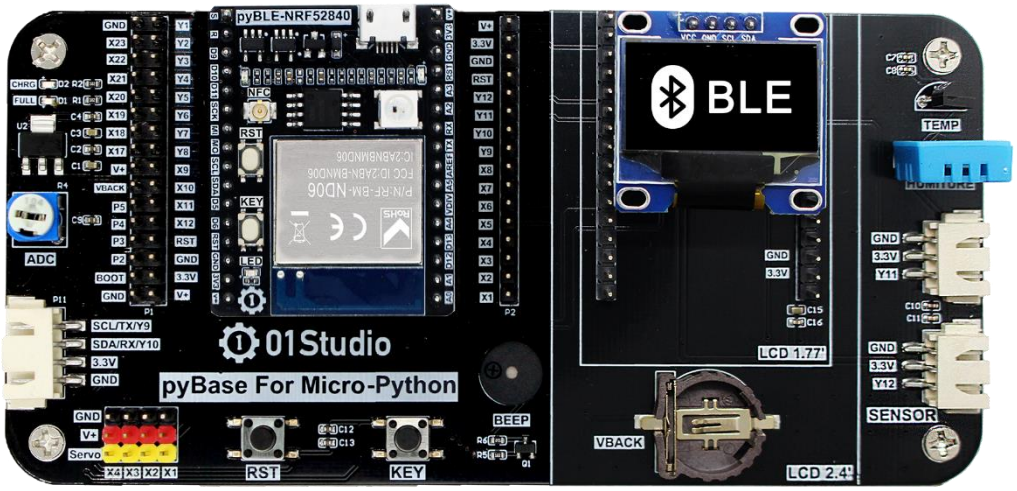


图 18-28

- **实验目的：**

学习使用 MicroPython 的 I2C 总线通讯编程和 OLED 显示屏的使用。

- **实验讲解：**

什么是 I2C?

I2C 是用于设备之间通信的双线协议，在物理层面，它由 2 条线组成：SCL 和 SDA，分别是时钟线和数据线。也就是说不通设备间通过这两根线就可以进行通信。

什么是 OLED 显示屏？

OLED 的特性是自己发光，不像 TFT LCD 需要背光，因此可视度和亮度均高，其次是电压需求低且省电效率高，加上反应快、重量轻、厚度薄，构造简单，成本低等特点。简单来说跟传统液晶的区别就是里面像素的材料是由一个个发光二极管组成，因为密度不高导致像素分辨率低，所以早期一般用作户外 LED 广告牌。随着技术的成熟，使得集成度越来越高。小屏也可以制作出较高的分辨率。



图 18-29 I2C 接口的 OLED

在了解完 I2C 和 OLED 显示屏后，我们先来看看开发板的原理图，也就是 MicroPython 上的 OLED 接口是如何连线的。下图是 pyBase 开发底板的原理图。

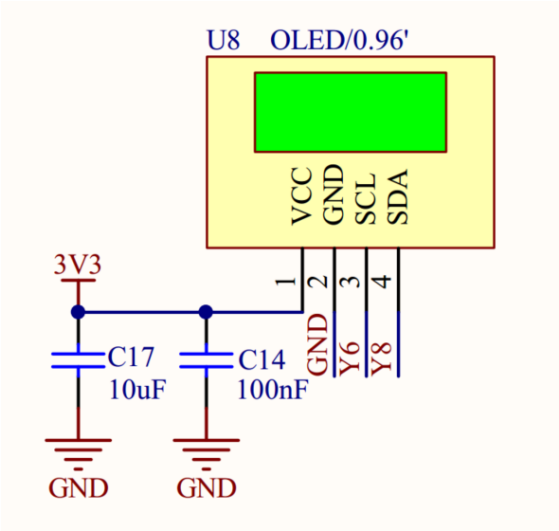


图 18-30 OLED 接口原理图

我们再来看看 pyBLE-NRF52840 的原理图接口部分。

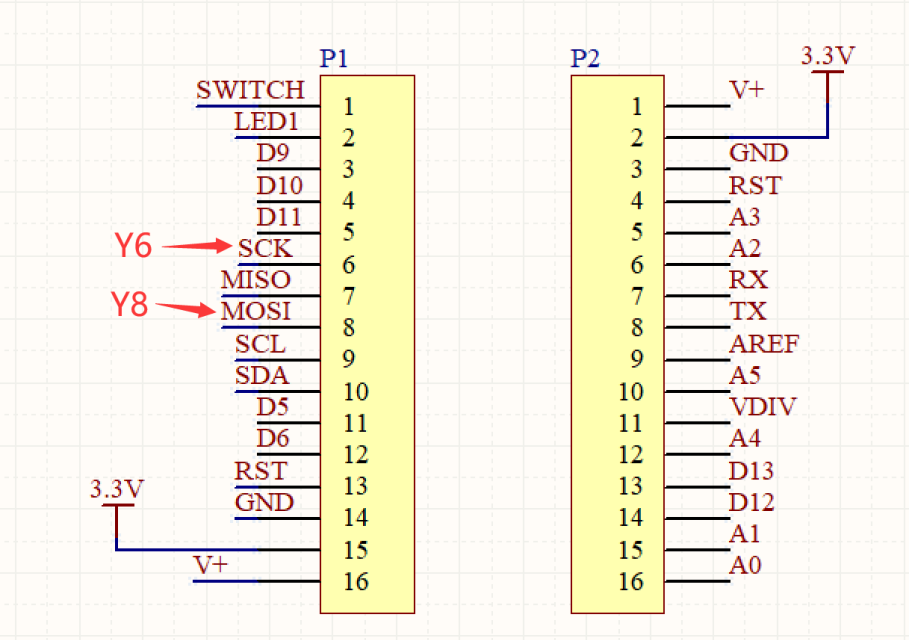


表 18-5　pyBLE-NRF52840 PYB 接口图

结合以上可以得知 pyBase 底板连接到 OLED 的对应关系是 Y6→SCL 和 Y8→SDA。对应 pyBLE-NRF52840 的关系是：SCK→Y6→SCL，MOSI→Y8→SDA。可以看到 OLED 没有直接接到开发板的默认 OLED 本例程将使用 MicroPython 的软件模拟 I2C 接口：

构造函数
`i2c=busio.I2C(scl,sda,frequency=400000,timeout=255)`
构建软件模拟的 I2C 对象。 【scl】时钟引脚； 【sda】数据引脚； 【frequency】通信频率，即速度； 【timeout】超时时间。
使用方法
`i2c.scan()`
扫描 I2C 总线的设备。返回地址，如：0x3c；
`i2c.readfrom(address,buffer)`

从指定地址读数据。address:设备地址；buffer:数据内容；
i2c.writeto(address,buffer)
写数据。address:设备地址；buffer:数据内容；
i2c.deinit()
注销 I2C。
*其它更多用法请阅读官方文档： 文档链接： https://circuitpython.readthedocs.io/en/5.0.x/shared-bindings/busio/I2C.html

表 18-6 I2C 对象

定义好 I2C 后，还需要驱动一下 OLED。OLED 的 python 模块已经写好。开发者只需要将相关库文件拷贝到开发板的文件系统（U 盘）里面，然后在 main.py 里面调用函数即可。人生苦短，我们学会调用函数即可，也就是注重顶层的应用，想深入的小伙伴也可以自行研究 py 库文件代码。

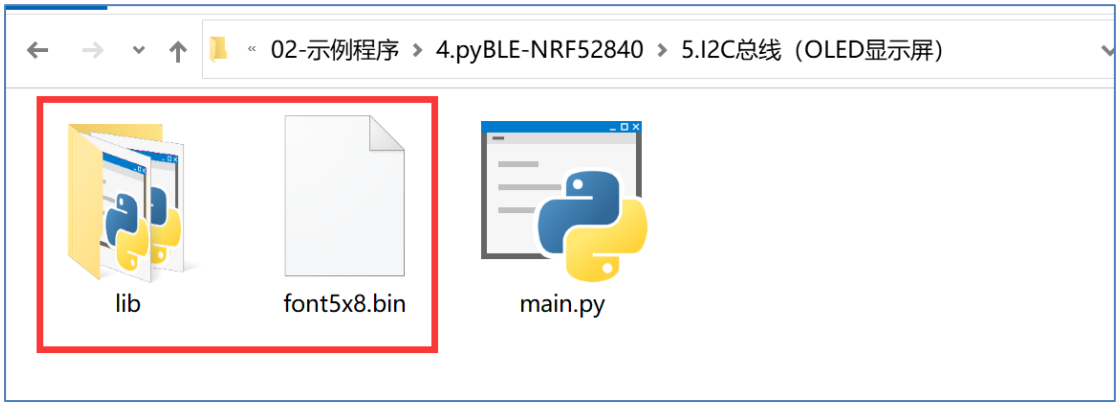


图 18-31 oled 库文件拷贝到 CIRCUITPY U 盘里面

OLED 显示屏对象介绍如下：

构造函数
display = adafruit_ssd1306.SSD1306_I2C(width, height, i2c, addr=0x3C)
构 OLED 显示屏对象。 【widht】oled 宽度; 【height】oled 高度；

【i2c】定义好的 I2C；

【addr】OLED 地址。

使用方法
display.fill([value])
清屏； value=0 (黑屏)；value=1(白屏)
display.show()
显示刷新。新增显示内容后需要执行此指令
display.pixel(x,y,color)
画点。 【x,y】坐标； 【color】颜色，1：白色；2：黑色。
display.hline(x,y,width,color)
画横线。 【x,y】起始坐标； 【width】长度； 【color】颜色，1：白色；2：黑色。
display.rect(x,y,width,height,color,fill=false)
画矩形。 【x,y】起始坐标； 【width】长度； 【height】高度； 【color】颜色，1：白色；2：黑色； 【fill】是否填充。
display.circle(center_x,center_y,radius,color)
画圆形。 【center_x,center_y】圆心； 【radius】半径； 【color】颜色，1：白色；2：黑色；

display.text(string,x,y,font_name='font5*8.bin',size=1)
将 string 字符写在指定为位置。 【string】：字符； 【x,y】起始坐标； 【font_name】字体 【size】尺寸
更多其它内容请看官方文档： https://circuitpython.readthedocs.io/projects/framebuf/en/latest/api.html

表 18-7　OLED 对象

学习了 I2C、OLED 对象用法后我们通过编程流程图来理顺一下思路：

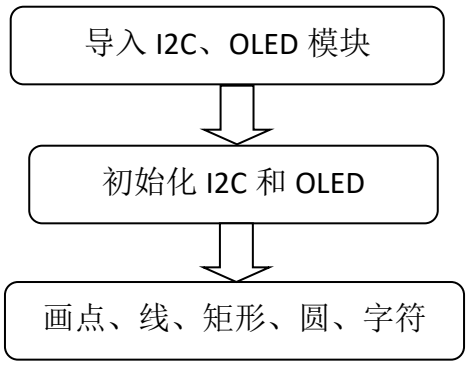


图 18-32 代码编写流程

参考代码如下：

```
"""
实验名称：I2C 总线（OLED 显示屏）
版本：v1.0
日期：2020.4
作者：01Studio
实验内容：学习使用 MicroPython 的 I2C 总线通讯编程和 OLED 显示屏的使用。
"""


# 导入相关模块
```

```
import time, board, busio
from digitalio import DigitalInOut
import adafruit_ssd1306



# 构建 I2C 对象
i2c = busio.I2C(board.SCK, board.MOSI)

# 构建 oled 对象,01Studio 配套的 OLED 地址为 0x3C
display = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c, addr=0x3C)

# 清屏
display.fill(0)
display.show()

# 画点(x,y,color)
display.pixel(5, 5, 1)

# 画线(x, y, width, color)
display.hline(5, 10, 20, 1)

# 画矩形(x, y, width, height, color, *, fill=False)
display.rect(5, 15, 20, 10, 1)

# 画圆
display.circle(50, 15, 10, 1)

# 字符
display.text("Hello 01Studio!", 5, 40, 1, font_name="font5x8.bin")
```

```
# 显示
display.show()
```

上述代码中 OLED 的 I2C 地址是 0x3C,不同厂家的产品地址可能预设不一样，具体参考厂家的说明书。或者也可以通过 I2C.scan()来获取设备地址。

另外记得将我们提供的**示例代码文件夹**中的驱动文件拷贝到 pyBLE-NRF52840 的文件系统下。

- **实验结果：**

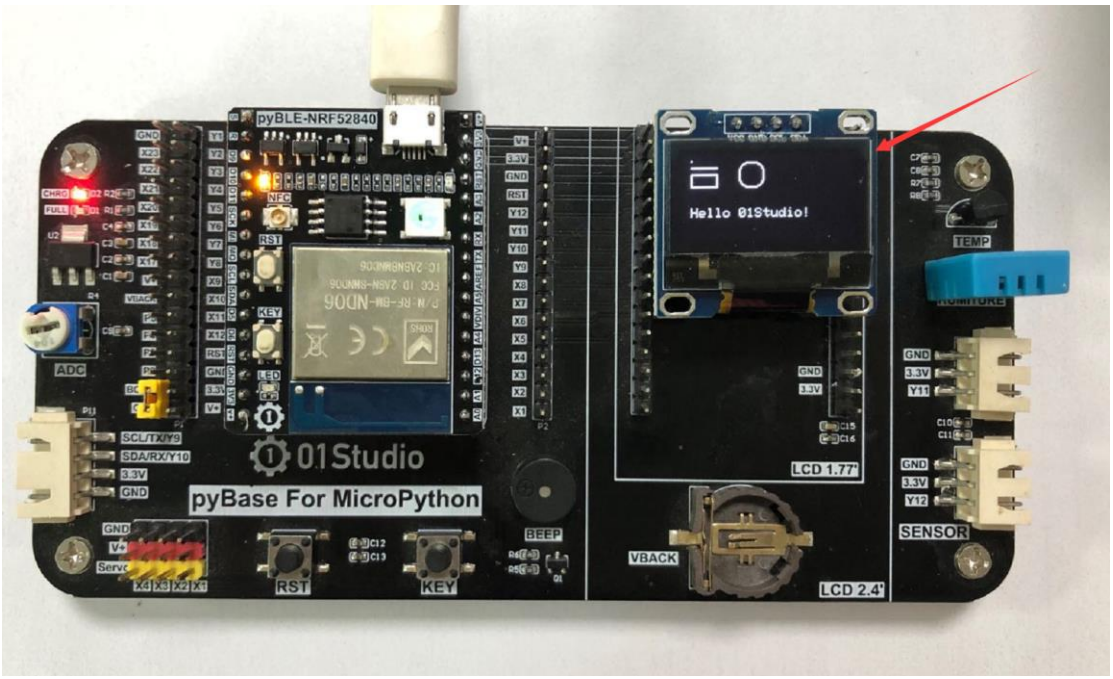


图 18-33　OLED 实验结果

- **总结：**

这一节我们学会了驱动 OLED 显示屏，换着以往如果从使用单片机从 0 开发的话你需要了解 I2C 总线原理，了解 OLED 显示屏的使用手册，编程 I2C 代码，有经验的嵌入式工程师搞不好也要弄个几天。现在基本半个小时解决问题。当然前提是别人已经给你搭好桥了，有了强大的底层驱动代码支持，我们只做好应用就好。

这一节学习的意义不仅在完成实验。在学习完 OLED 显示屏实验后，接下来我们的实验都可以使用这个 OLED 来跟用户交互了，这大大提高了实验的可观性。

18.6 UART（串口通讯）

- **前言：**

串口是非常常用的通信接口，有很多工控产品、无线透传模块都是使用串口来收发指令和传输数据，这样用户就可以在无须考虑底层实现原理的前提下将各类串口功能模块灵活应用起来。

- **实验平台：**

pyBLE-NRF52840 开发套件。

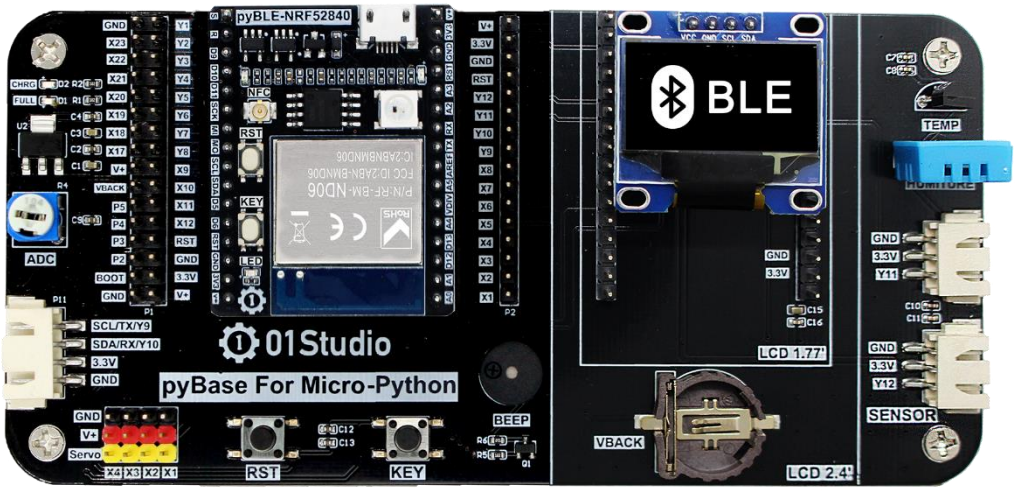


图 18-34 MicroPython 开发套件

- **实验目的：**

编程实现串口收发数据。

- **实验讲解：**

pyBLE-NRF52840 默认有 1 个串口（TX，RX），也可以通过软件来模拟。我们来了解一下 CircuitPython 串口对象的构造函数和使用方法：

构造函数
uart=busio.UART(tx,rx,baudrate=9600,bits=8,parity=None,stop=1,timeout=1, receiver_buffer_size=64)
创建 UART 对象。 【tx】串口发送引脚；

【rx】串口接收引脚； 【baudrate】波特率，默认 9600； 【bits】数据位,默认 8 【parity】校验；默认 None, 0(偶校验)，1(奇校验) 【stop】停止位，默认 1 【timeout】串口接收超时时间，默认 1 秒 【receiver_buffer_size】串口接收缓冲大小。
使用方法
uart.read(nbytes=None)
读取串口缓冲数据 【nbytes】读取字节数
uart.readline()
读取串口缓冲数据的行.
uart.write(buf)
串口发送数据 【buf】需要发送的数据
uart.deinit()
注销串口
*其它更多用法请阅读官方文档： 文档链接： https://circuitpython.readthedocs.io/en/5.0.x/shared-bindings/busio/UART.html#

表 18-8 UART 对象

我们可以用一个 USB 转 TTL 工具，配合电脑上位机串口助手来跟 pyBLE-NRF52840 开发板模拟通信。

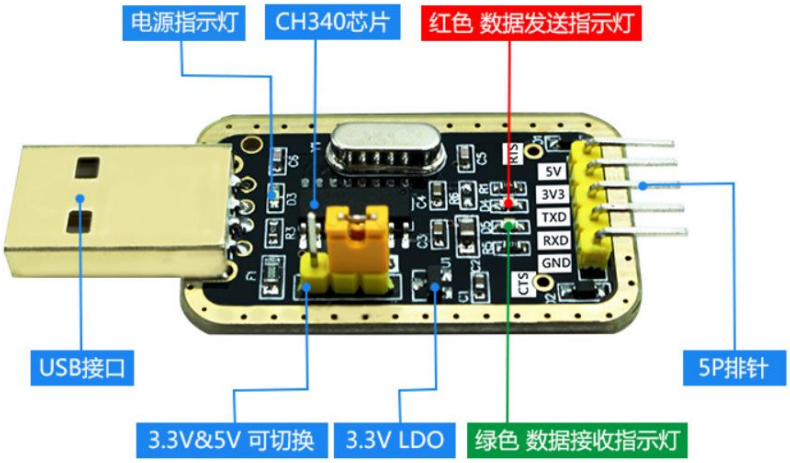


图 18-35 常用 USB 转串口工具

注意要使用 3.3V 电平的 USB 转串口 TTL 工具，本实验我们使用 pyBase 的外接串口引脚，也就是 Y9（TX）和 Y10（RX），接线示意图如下：



图 18-36 串口接线图

从 pyBLE-NRF52840 原理图可以看到外部 Y9→RX，Y10→TX。

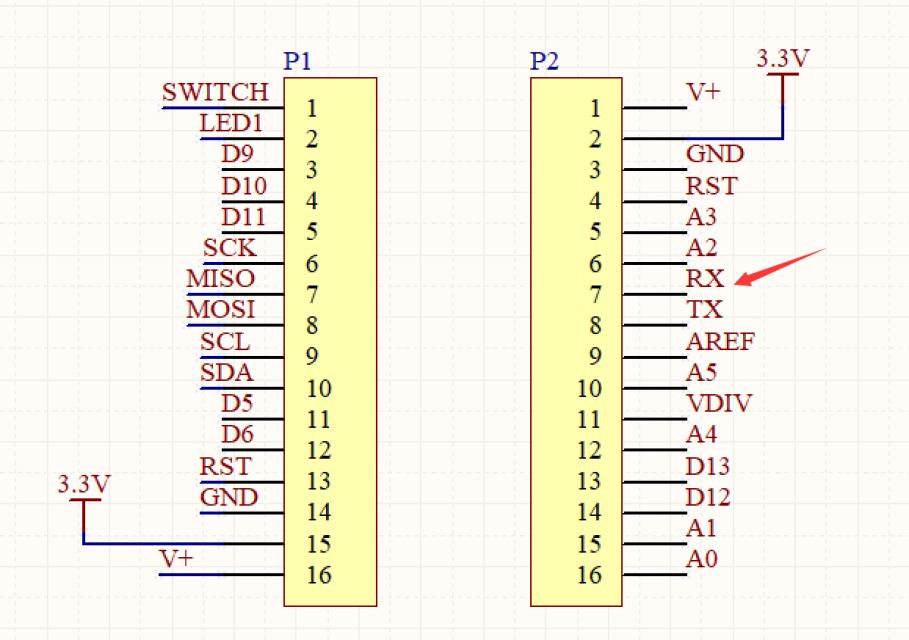


图 18-37 串口引脚

在本实验中我们可以先初始化串口，然后给串口发去一条信息，这样 PC 机的串口助手就会在接收区显示出来，然后进入循环，当检测到有数据可以接收时候就将数据接收并打印，并通过 REPL 打印显示。代码编写流程图如下：

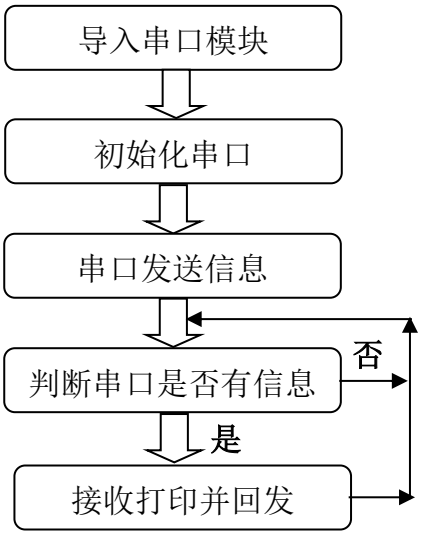


图 18-38 代码编程流程图

参考代码：

```
'''
实验名称：UART(串口通讯)
版本：v1.0
日期：2020.4
作者：01Studio
'''

#导入相关模块
import board
import busio

#串口初始化
uart = busio.UART(board.TX, board.RX, baudrate=9600)

#发送数据,要求字节数组
uart.write(b'Hello 01Studio!')

while True:

    data = uart.read(64)  #最多接收 64 个字节

    if len(data) != 0: #有数据
        print(data)    #REPL 打印
        uart.write(data) #数据回发
```

- **实验结果：**

我们按照上述方式将 USB 转 TTL 工具的 TX 接到 Y10(IO6)，RX 接到 Y9(IO7)。

GND 接一起，3.3V 可以选择接或不接。（可以直接接到 pyBase 左侧的 4P 防呆接口。）

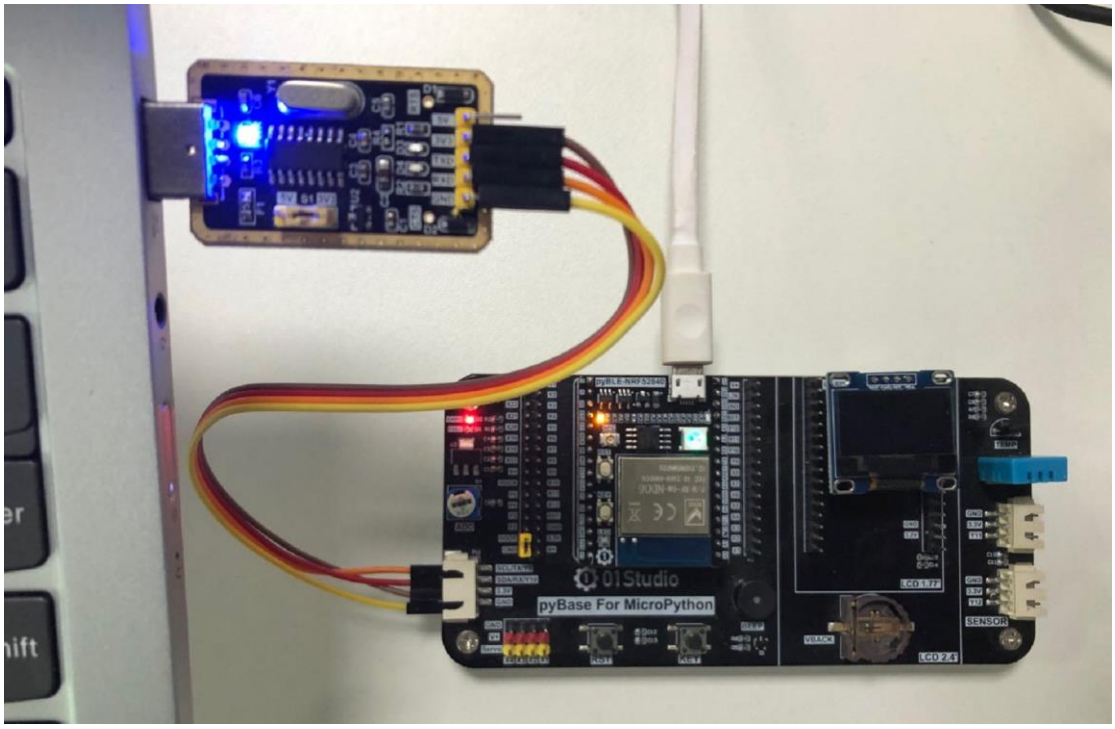


图 18-39 将 pyBLE-NRF52840 和 usb ttl 工具同时接入电脑

这时候打开电脑的设备管理器，能看到 2 个 COM。写着 CH340 的是串口工具，另外一个则是 pyAI-K210 的 REPL。如果 CH340 驱动没安装，则需要手动安装，驱动在：配套资料包→开发工具→windows→串口终端→CH340 文件夹下。



图 18-40

本实验要用到串口助手，打开配套资料包→开发工具→windows→串口终端工具下的【UartAssist.exe】软件。

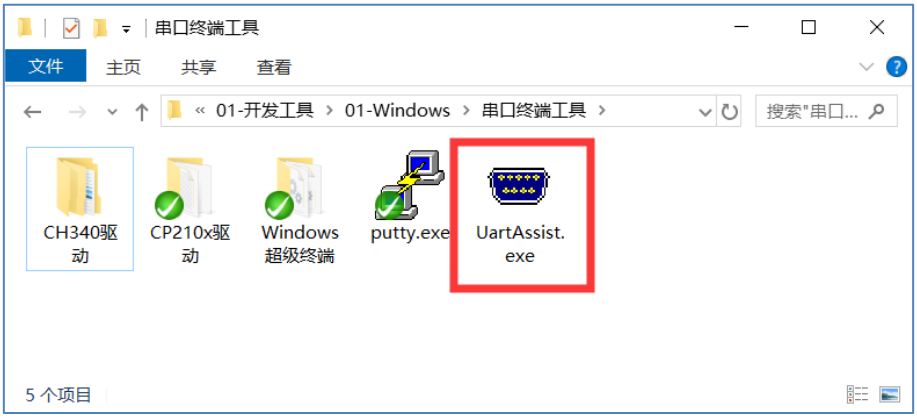


图 18-41

将串口工具配置成 COM14（根据自己的串口号调整）。波特率 9600。运行程序，可以看到一开始串口助手收到 pyBLE-NRF52840 上电发来的信息“Hello 01Studio!”。我们在串口助手的发送端输入“http://www.01studio.org”， 点击发送，可以看到 pyBLE-NRF52840 在接收到该信息后在 REPL 里面打印了出来，并发货给串口助手。如下图所示：

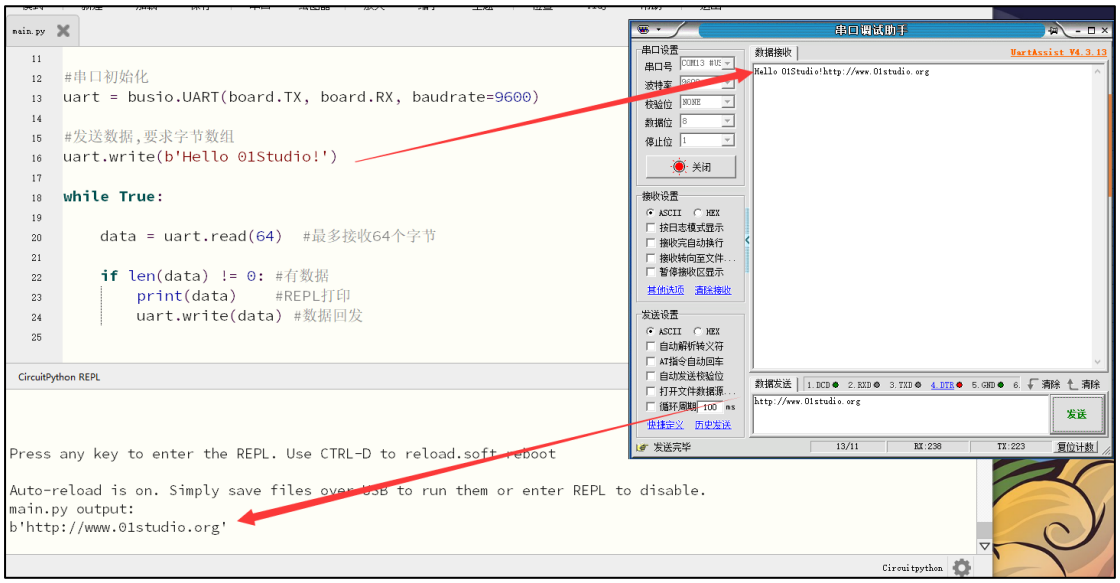


图 18-42 串口收发实验

- **总结：**

通过本节我们学会了串口收发应用，很多外设模块都是通过串口通讯。从而可以让我们开发板实现更多的拓展性。

第19章　传感器实验

基础实验让我们对 MicroPython 的操作和编程有了比较全面的了解，但其魅力绝不限于此。日常生活中我们会用到各式各样的外设或者传感器，还是那句，一个有经验的嵌入式开发工程师驱动一款未接触过的传感器的一般流程是：了解传感器原理、设计电路图、信号时序分析和编程。没个几天折腾不出来。

生活中有很多传感器已经是非常通用了，前人已经做好封装函数模块，我们直接调用函数即可。我们不需要将时间花在“怎么用”上，而更多的是考虑“用到什么地方”!

本章实验以最常见的传感器出发，讲述是如何通过 MicroPyhon 编程实现传感器应用的。实验还是基于我们的 MicroPython 开发板。而且实验例程将持续更新。

19.1 温度传感器 DS18B20

- **前言：**

相信没有电子爱好者不知道 DS18B20 的，DS18B20 是常用的数字温度传感器，其输出的是数字信号，具有体积小，硬件开销低，抗干扰能力强，精度高的特点。DS18B20 数字温度传感器接线方便，封装成后可应用于多种场合，如管道式，螺纹式，磁铁吸附式，不锈钢封装式，型号多种多样。

主要根据应用场合的不同而改变其外观。封装后的 DS18B20 可用于电缆沟测温，高炉水循环测温，锅炉测温，机房测温，农业大棚测温，洁净室测温，弹药库测温等各种非极限温度场合。耐磨耐碰，体积小，使用方便，封装形式多样，适用于各种狭小空间设备数字测温和控制领域。

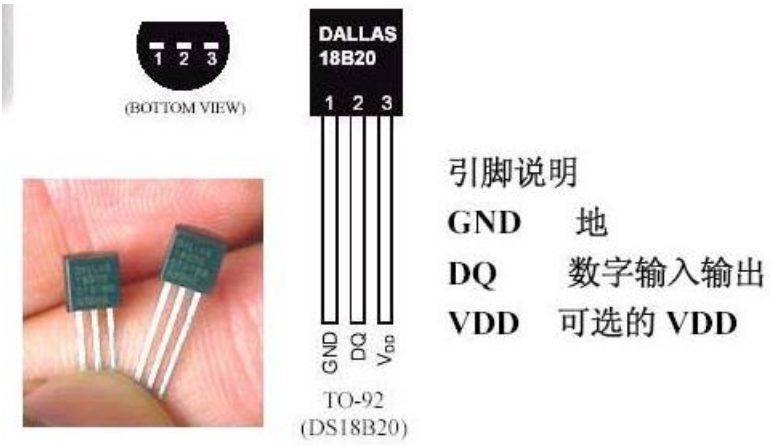


图 19-1 DS18B20 传感器



图 19-2 DS18B20 金属探头封装

- **实验平台：**

PyBLE-NRF52840 和 pyBase 开发底板。DS18B20 温度传感器位于右上方。

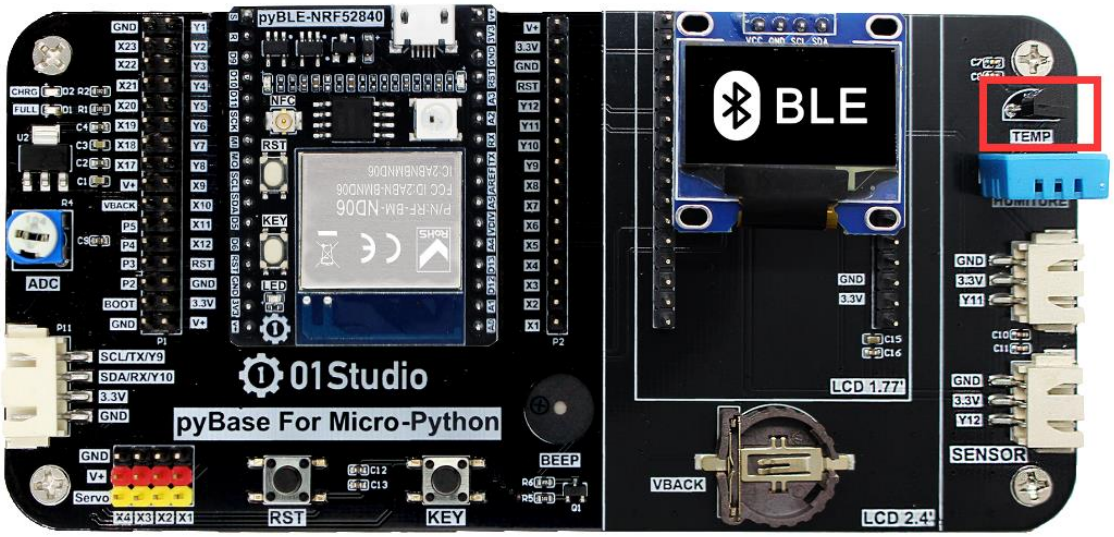


图 19-3 pyBLE-NRF52840 开发套件

- **实验目的：**

通过编程采集温度数据，并在 OLED 上显示。

- **实验讲解：**

DS18B20 是单总线驱动（onewire）传感器，也就是说只占用 1 个 IO 口。我们来看看原理图：

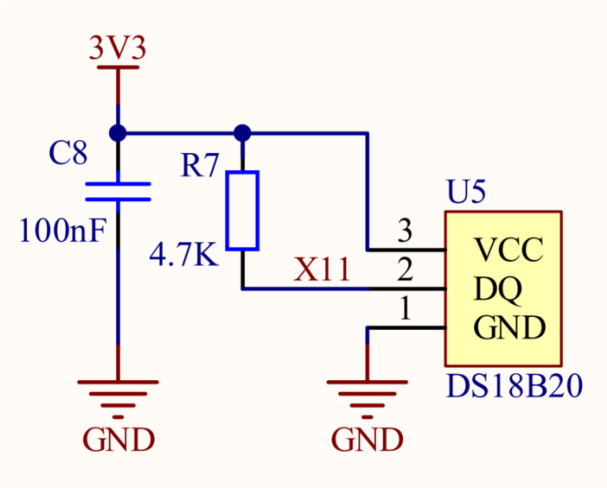


图 19-4 DS18B20 接线原理图

可以看到 DS18B20 传感器连接到了 pyBase 的 X11 引脚上。也就是连接到 pyBLE-NRF52840 的引脚 D5，如下图所示：

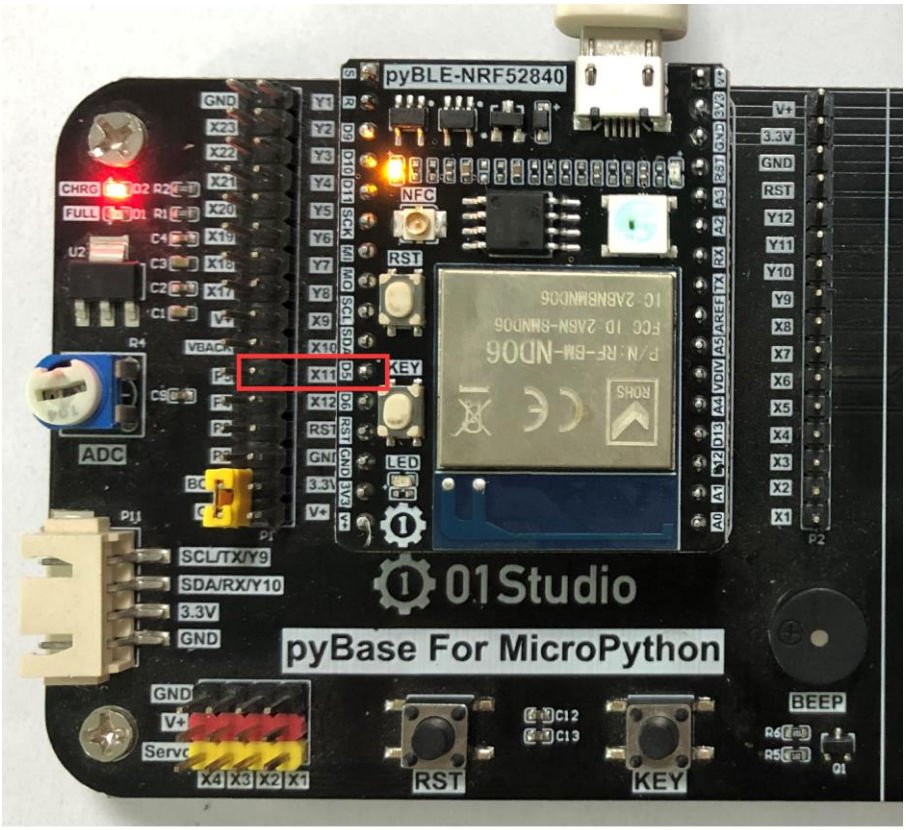


图 19-5 D5 与 pyBase 底板 X11 连接

也就是说我们需要针对引脚 D5 编写程序来驱动 DS18B20。那么我们需要自己来编写驱动么？如果你有兴趣的可以自己尝试一下。Python 的一个理念就是没必要重复做轮子，我们可以使用官方提供的库来完成。

单总线模块（onewire）和 ds18x20 模块说明如下：

构造函数
ow= adafruit_onewire.bus.OneWireBUS(pin)
构建单总线对象。 【pin】引脚，如 board.D5
使用方法
ow.reset()

复位总线设备。
ow.scan()
扫描设备地址。
ow.deinit()
注销总线设备，释放 IO。

图 19-6 单总线对象

构造函数
ds= adafruit_ds18x20.DS18X20(bus,address)
构建 DS18B20 传感器对象。 【bus】定义好的单总线对象； 【address】传感器地址。
使用方法
ds.temperature
返回温度数据。

图 19-7 DS18B20 对象

大部分场景下温度的变化不会太频繁，我们可以每隔 1 秒采集一次，显示精度为小数点后 2 位，基本满足大部分应用需求。编程逻辑如下：

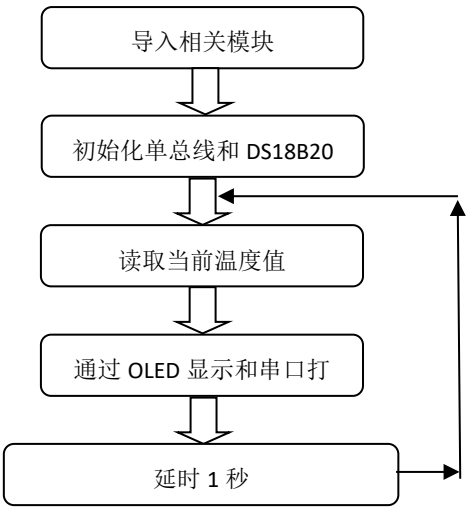


图 19-8 代码编写流程图

实验参考代码：

```
'''
实验名称：温度传感器 DS18B20
版本：v1.0
日期：2020.4
作者：01Studio
实验内容：采集温度数据并在 oled 上显示。
'''

#导入相关模块
import time,board,busio
from digitalio import DigitalInOut
import adafruit_ssd1306

#DS18X20 库模块
from adafruit_onewire.bus import OneWireBus
from adafruit_ds18x20 import DS18X20



#构建 I2C 对象
i2c = busio.I2C(board.SCK, board.MOSI)

#构建 oled 对象,01Studio 配套的 OLED 地址为 0x3C
display = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c, addr=0x3C)

#清屏
display.fill(0)
display.show()
```

```
# 初始化单总线对象，引脚为 D5.
ow = OneWireBus(board.D5)

# 搜索传感器，返回第 1 个
ds = DS18X20(ow, ow.scan()[0])

while True:

    temp=round(ds.temperature,2) #保留 2 位小数

    display.fill(0) #清屏
    display.text('01Studio', 0,0, 1,font_name='font5x8.bin')
    display.text('Temp Test', 0,20, 1,font_name='font5x8.bin')

    #显示温度数据
    display.text(str(temp)+' C', 0,40, 1,font_name='font5x8.bin')
    display.show()

    print(str(temp)+' C')  #REPL 串口打印数据

    time.sleep(1)
```

实验使用到 OLED 显示屏，记得同时将例程文件的所有库文件拷贝到开发板文件系统中。

- **实验结果：**

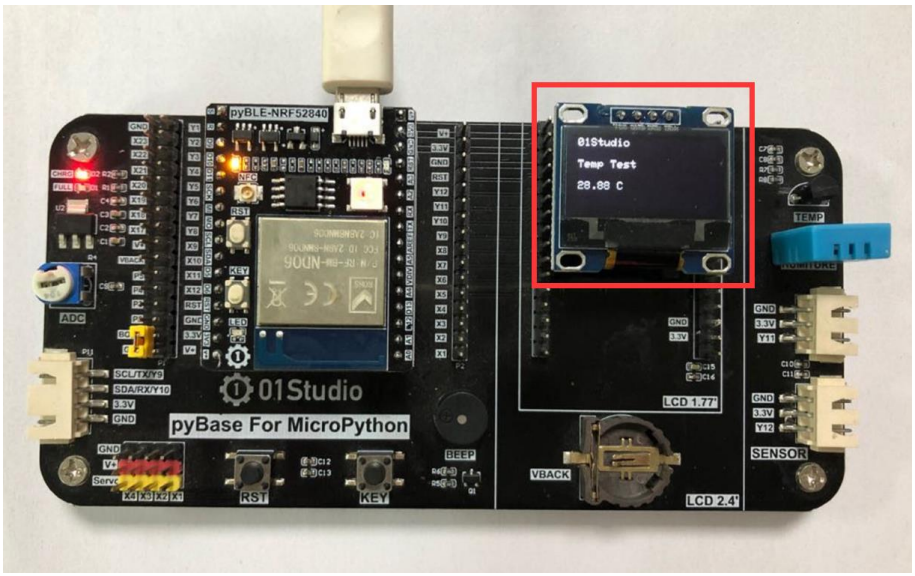


图 19-9 温度检测实验

- **实验拓展：**

pyBase 开发底板预留了外界传感器接口，只要接线正确就可以进行更多的传感器实验。我们将带金属探头的 DS18B20 传感器接到 pyBase 右侧上面的传感器母座，其连接到 pyBase 的“Y11”引脚,也就是对应 pyBLE-NRF52840 的引脚 A2。所以只要将原程序代码： ow = OneWireBus(board.D5) 改成 ow = OneWireBus(board.A2)，即可驱动外接的 DS18B20。

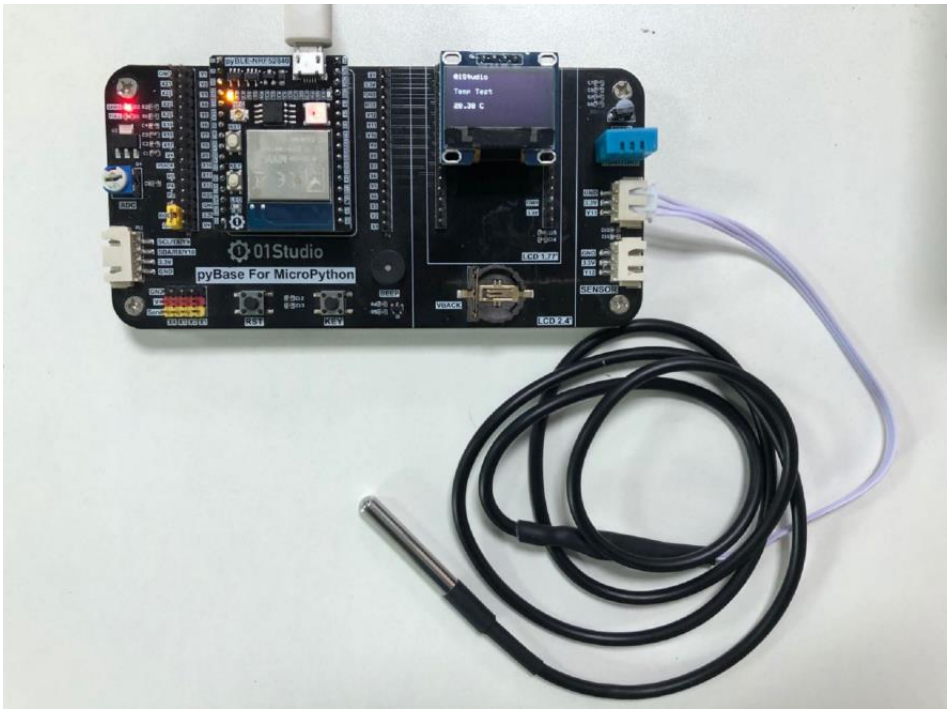


图 19-10 外接温度传感器

我们将金属探头放在水里面，可以见到测试到的温度。

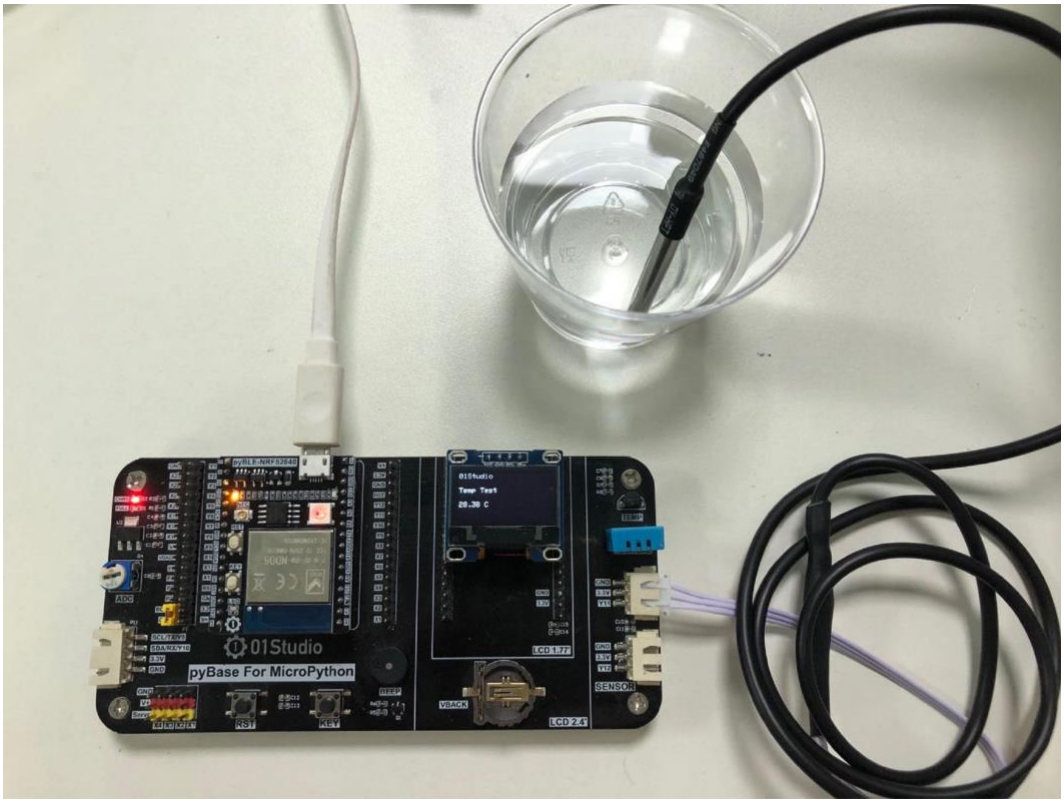


图 19-11 实现金属探头 DS18B20 测量

- **总结**

DS18B20 作为我们第一个实验传感器，使用 MicroPython 编程非常容易就用起来了，而且精度和稳定性丝毫没有影响。温度传感器只是一个敲门砖，接下来我们将会学习更多的传感器应用。

19.2 温湿度传感器 DHT11

- **前言：**

温湿度也是我们日常非常常见的指标，我们使用的是 DHT11 数字温湿度传感器。这是一款含有已校准数字信号输出的温湿度复合传感器，它应用专用的数字模块采集技术和温湿度传感技术，确保产品具有极高的可靠性和卓越的长期稳定性。

DHT11 具有小体积、极低的功耗，信号传输距离可达 20 米以上，使其成为给类应用甚至最为苛刻的应用场合的最佳选择。产品为 4 针单排引脚封装，连接方便。

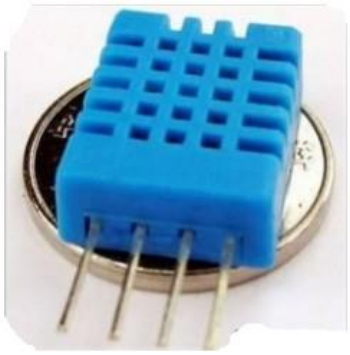


图 19-12 DHT11 温湿度传感器

- **实验平台：**

pyBLE-NRF52840 和 pyBase 开发底板。DHT11 温湿度传感器位于右上方。

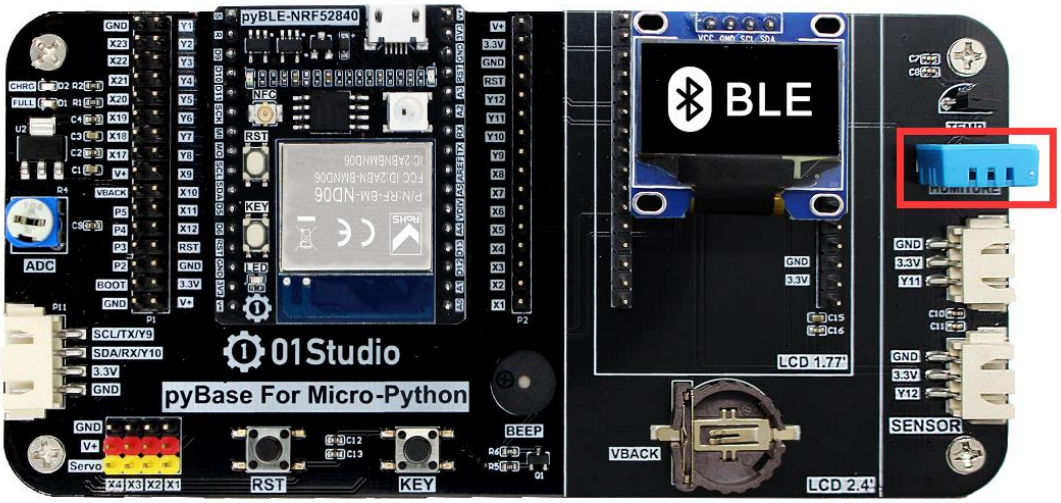


图 19-13 温湿度传感器 DHT11 位于右上方

- **实验目的：**

通过编程采集温湿度数据，并在 OLED 上显示。

- **实验讲解：**

DHT11 虽然有 4 个引脚，但其中第 3 个引脚是悬空的，也就是说 DHT11 也是单总线的传感器，只占用 1 个 IO 口。

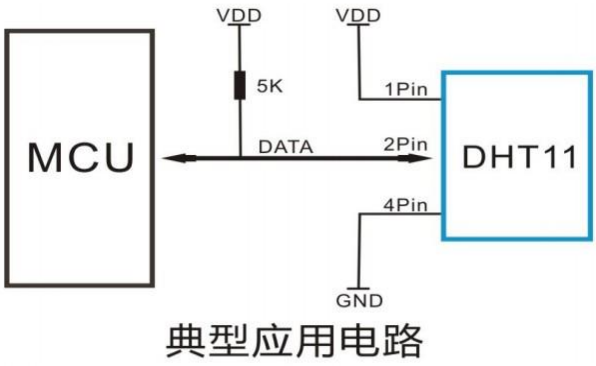


图 19-14　DHT11 应用电路

我们来看看 DHT11 在开发板上的接线图：

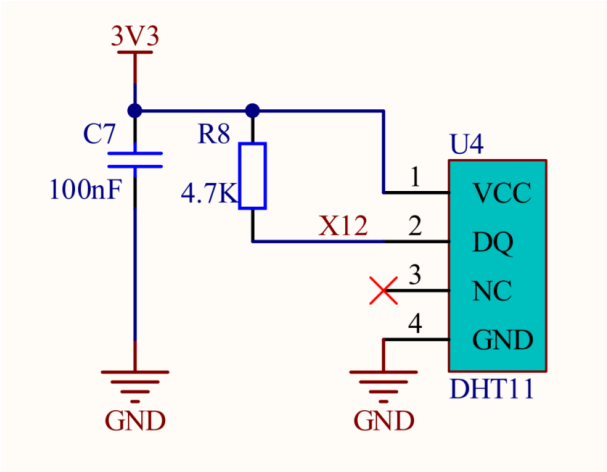


图 19-15　DHT11 接线图

可以看到 DHT11 连接到 pyBase 的‘X12’引脚，也就是连接到 pyBLE-NRF52840 的引脚 D6，如下图所示：

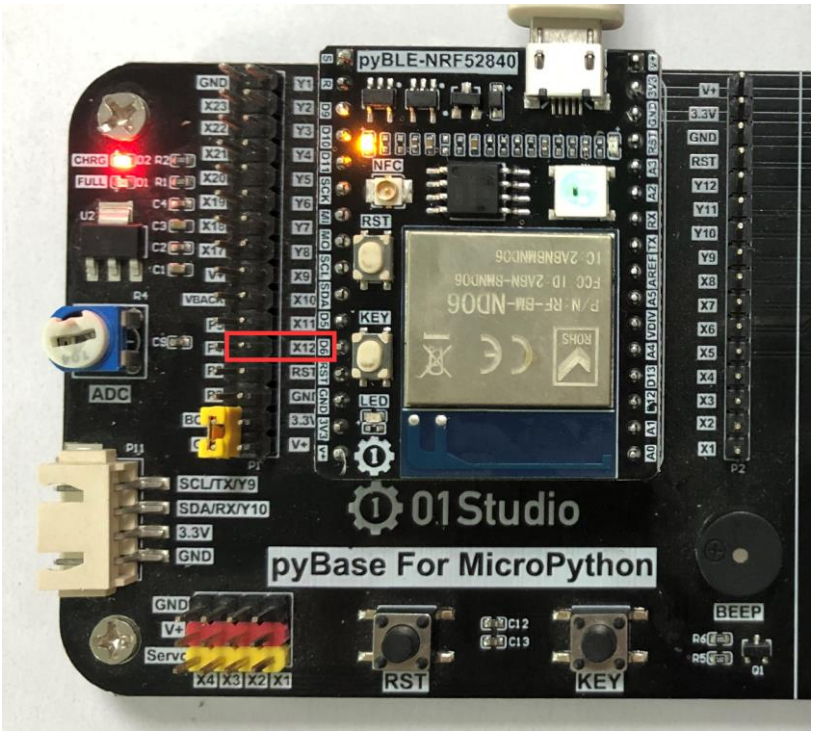


图 19-16 用跳线帽将 pyBase 的 X20、Y5 短接

因此可以针对引脚 D6 编程来驱动 DHT11 传感器，使用的是 adafruit_dht 库，也就是说跟以往一样我们可以直接在 main.py 导入模块并调用即可。函数模块说明如下：

构造函数
dht = adafruit_dht.DHT11(pin)
构建 DHT11 传感器对象。 【pin】传感器所连接的引脚,如：board.D6
使用方法
dht.temperature
获取温度；
dht. humidity
获取湿度；

表 19-1 DHT 对象

建议上电先延时 1 秒，让 DHT11 稳定后再开始读取。代码编写流程如下：

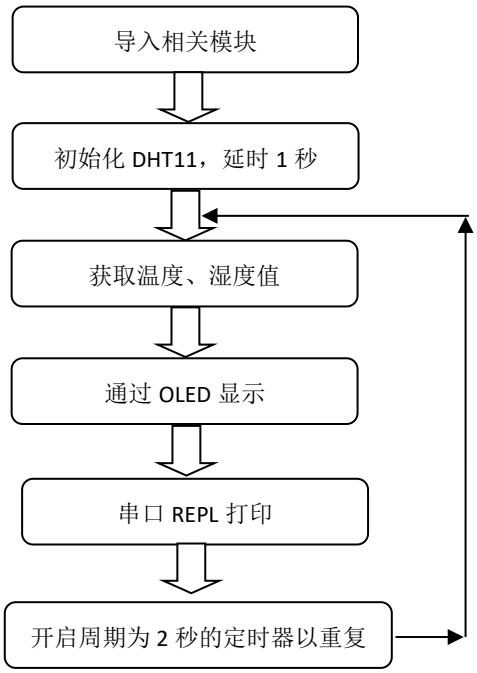


图 19-17 代码编写流程图

实验参考代码：

```
'''
实验名称：温湿度传感器 DHT11
版本：v1.0
日期：2020.4
作者：01Studio
实验内容：采集温湿度数据并在 oled 上显示。
'''

#导入相关模块
import time,board,busio
from digitalio import DigitalInOut
import adafruit_ssd1306
```

```
#DHT 库模块
import adafruit_dht



#构建 I2C 对象
i2c = busio.I2C(board.SCK, board.MOSI)

#构建 oled 对象,01Studio 配套的 OLED 地址为 0x3C
display = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c, addr=0x3C)

#清屏
display.fill(0)
display.show()

# 初始化 DHT11 对象，引脚为 D6.
dht = adafruit_dht.DHT11(board.D6)
time.sleep(1)#首次上电停顿 1 秒让传感器稳定

while True:

    temperature = dht.temperature #采集温度
    humidity = dht.humidity #采集湿度

    display.fill(0) #清屏
    display.text('01Studio', 0,0, 1,font_name='font5x8.bin')
    display.text('DHT11 Test', 0,20, 1,font_name='font5x8.bin')

    #显示温度数据
    display.text(str(temperature)+' C  '+str(humidity)+' %', 0,40,
                   1,font_name='font5x8.bin')
```

```
display.show()


print(str(temperature)+' C')  #REPL 串口打印数据
print(str(humidity)+' %')  #REPL 串口打印数据


time.sleep(2) #采集周期 2 秒
```

- **实验结果：**

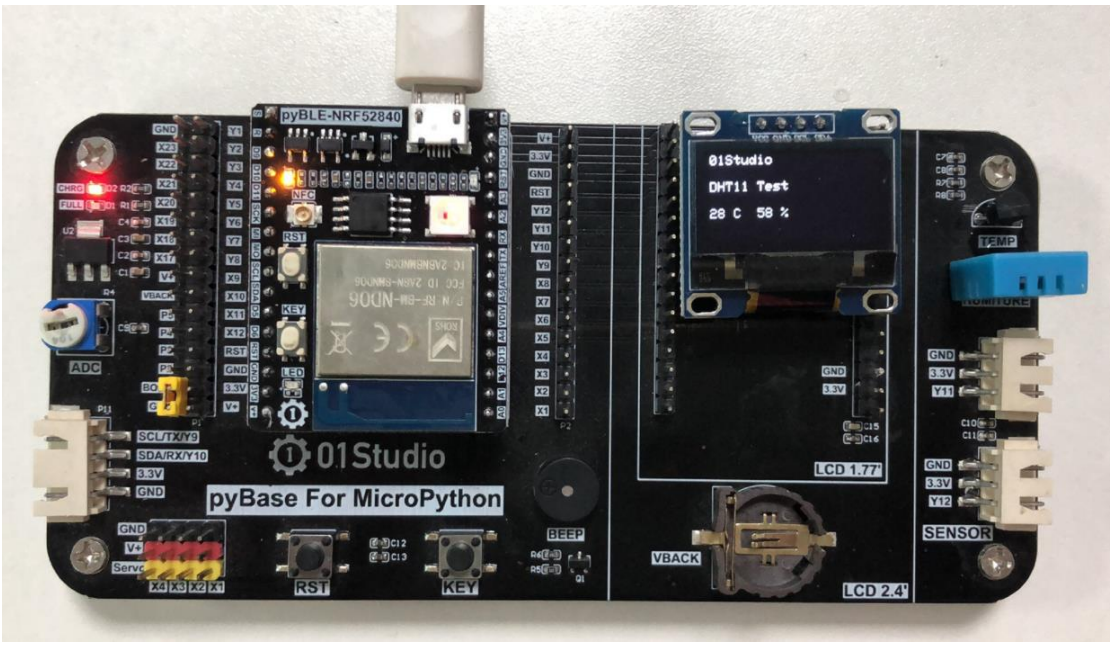


图 19-18　DHT11 实验结果

- **总结：**

通过本节学习我们学会了使用 MicroPython 来驱动 DTH11 温湿度传感器，DHT11 性价比比较高，是很适合学习使用的，但精度和响应速度有点低，需要更高要求应用的用户可以使用 DHT22 或者其他更高级的传感器。

19.3 人体感应传感器

- **前言：**

人体感应传感器，在室内安防应用非常普遍，其原理是由探测元件将探测到人体的红外辐射转变成微弱的电压信号，经过放大后输出。为了提高探测器的探测灵敏度以增大探测距离，一般在探测器的前方装设一个塑料的菲涅尔透镜，它和放大电路相配合，可将信号放大 70dB 以上，这样就可以测出 5~10 米范围内人的行动。

- **实验平台：**

pyBLE-NRF52840 开发套件和人体感应传感器模块。

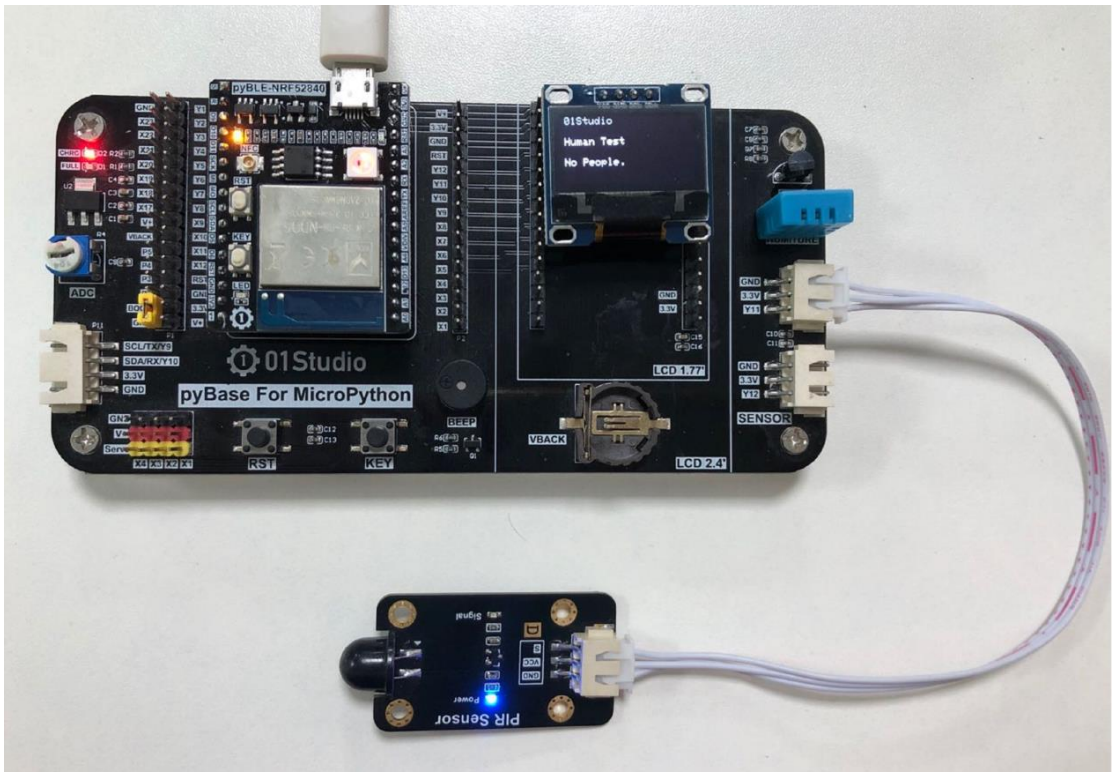


图 19-19 开发套件和传感器接线

- **实验目的：**

通过外部中断编程来检测人体感应模块，当有人出现时候 OLED 通过“Get People!!!”闪烁提示。

- **实验讲解：**

我们先来看看人体感应传感器的介绍：

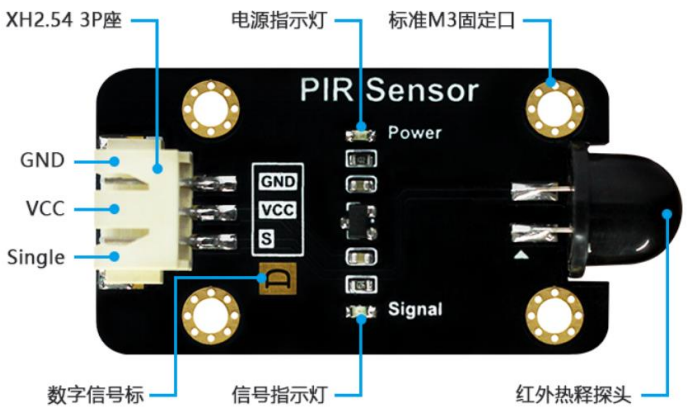


图 19-20 人体感应传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 65℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	数字信号： 检测到人体：高电平 3.3V；持续 3-8 秒 未检测到人体：低电平 0V。
感应角度	100°
感应距离	8 米
模块尺寸	4.5*2.5cm

图 19-21 人体感应传感器模块参数说明

从上表可以知，当检测到人体红外时候，传感器的输出信号为高电平并持续 3-5 秒。如下图所示：

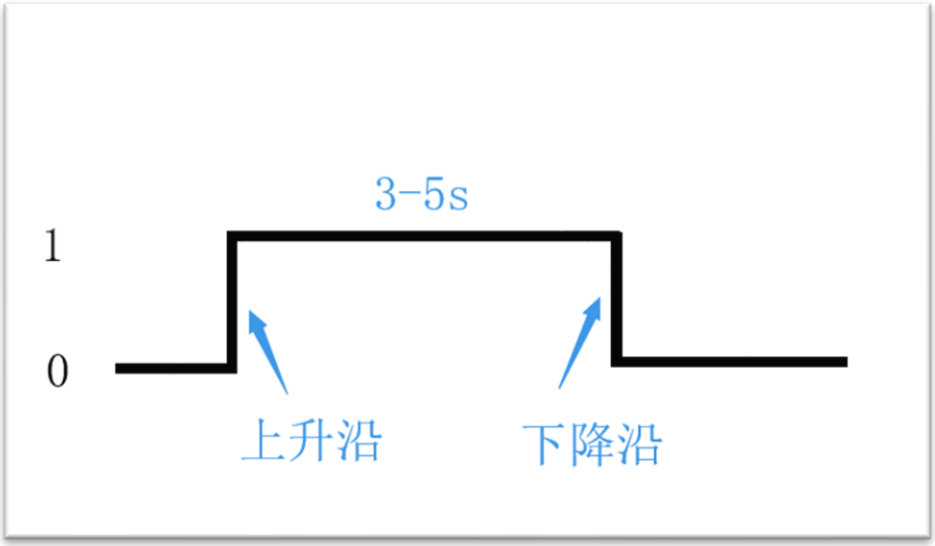


图 19-22 人体感应传感器电平变化

由此可见，这根我们前面基础实验做的按键实验非常类似，可以使用 GPIO 实时检测输入电平高低的方式来编程实现相关功能。传感器的输出引脚连接 Sensor1 接口，即连接到 pyBase 的“Y11”引脚。也就是 pyBLE-NRF52840 的‘A2’引脚。编程方法可以是当“A2”为高电平时，说明传感器检测到人体红外线，此时可以在 OLED 上显示相关信息，有关 GPIO 介绍可以看上一章“按键”实验章节内容，这里不再重复。具体的编程流程图如下：

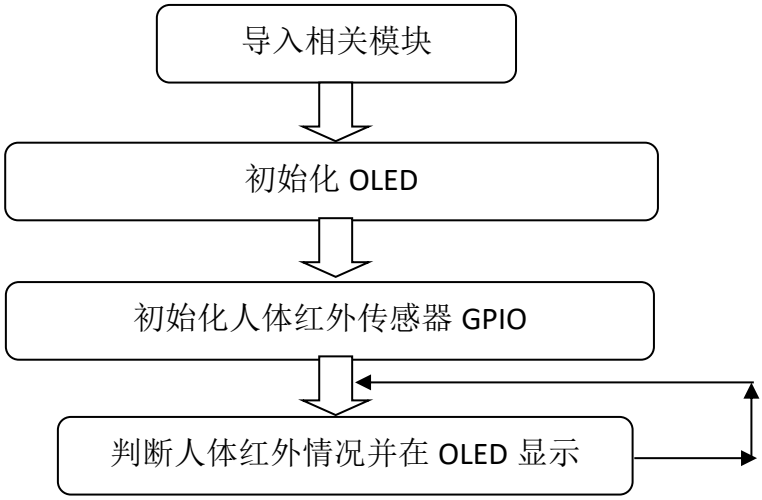


图 19-23 代码编写流程

参考例程代码如下：

```
'''
实验名称：人体感应传感器
版本：v1.0
日期：2020.5
作者：01Studio（www.01studio.org）
说明：人体红外感应传感器应用
'''

#导入相关模块
import time,board,busio
from digitalio import DigitalInOut, Direction, Pull
import adafruit_ssd1306



#构建 I2C 对象
i2c = busio.I2C(board.SCK, board.MOSI)

#构建 oled 对象,01Studio 配套的 OLED 地址为 0x3C
display = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c, addr=0x3C)

#清屏
display.fill(0)
display.show()

# 初始化人体红外传感器对象，引脚为 D6.
human = DigitalInOut(board.A2) #定义引脚编号
human.direction = Direction.INPUT #IO 为输入
human.pull = Pull.UP #打开上拉电阻
```

```
while True:

    if human.value == 1: #有人
        #OLED 显示有人 Got People
        display.fill(0) #清屏
        display.text('01Studio', 0,0, 1,font_name='font5x8.bin')
        display.text('Human Test', 0,20, 1,font_name='font5x8.bin')
        display.text('Got People!', 0,40, 1,font_name='font5x8.bin')
        display.show()

    else:
        #OLED 显示没人 No People
        display.fill(0) #清屏
        display.text('01Studio', 0,0, 1,font_name='font5x8.bin')
        display.text('Human Test', 0,20, 1,font_name='font5x8.bin')
        display.text('No People.', 0,40, 1,font_name='font5x8.bin')
        display.show()

    time.sleep(0.5) #检测周期 0.5 秒
```

- 实验结果：

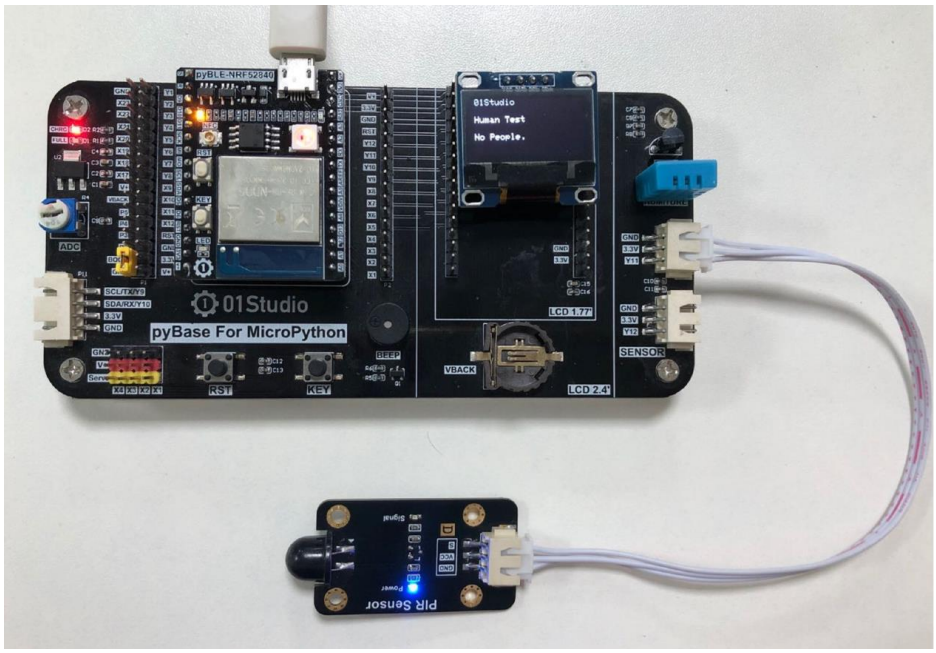


图 19-24 没有检测到到人体

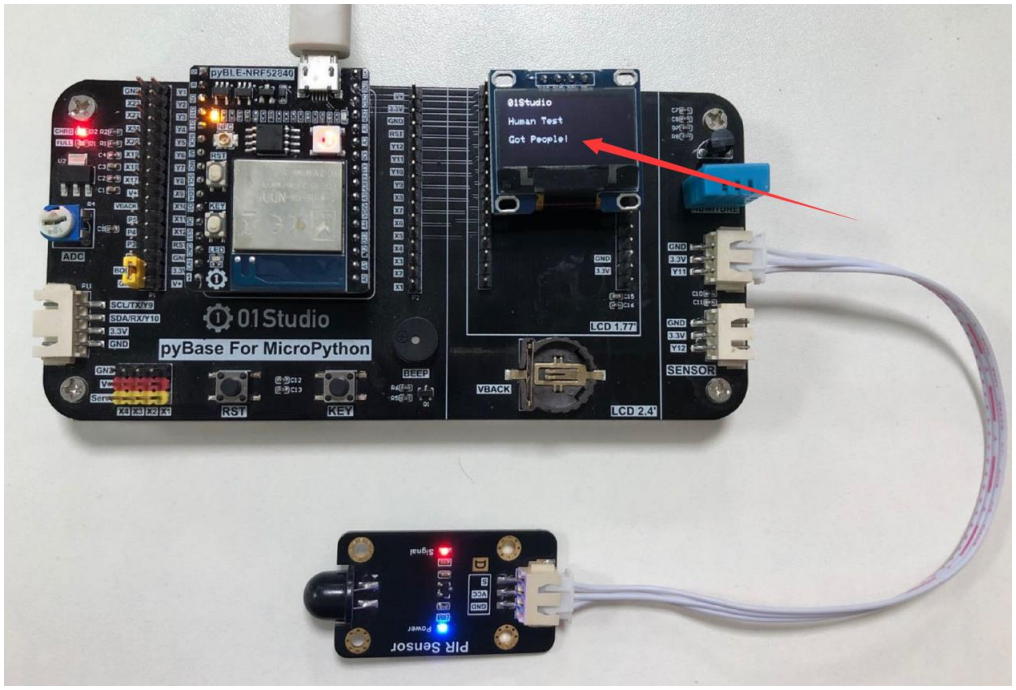


图 19-25 检测到人体，OLED 闪烁

- **总结：**

本节通过简单的中断方式便实现了对人体感应传感器的检测，人体感应传感器的应用非常广泛，特别是在安防领域，结合其它硬件模块可以实现当发现有人入侵时，执行发出警报、实现远程提醒等功能。

19.4 光敏传感器

- **前言：**

光敏传感器实际是一个款可以检测光照强度的传感器，可以应用于我们日常生活中植物光照强度、室内光线检测、以及某些场合的亮度检测。传感器的原理就是将外界模拟变化的信号转变成数字信号（电压值）让单片机出来，处理方式就是常用的 ADC（模拟数字转换）。

- **实验平台：**

pyBLE-NRF52840 开发套件开发套件和光敏传感器模块。

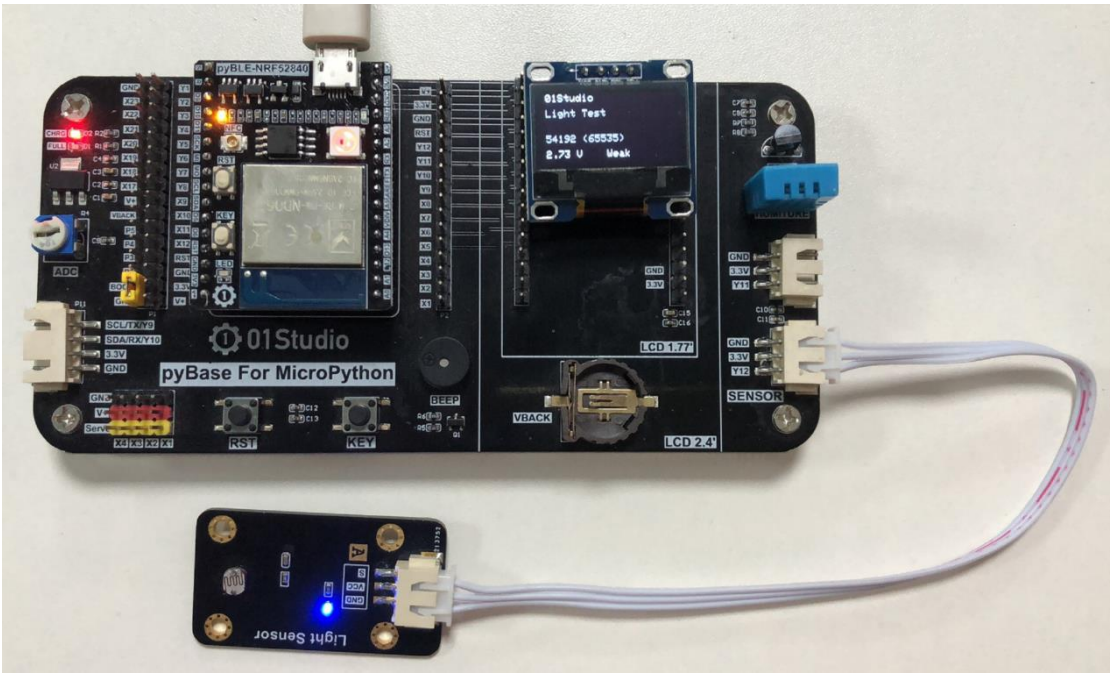


图 19-26 开发套件和光敏传感器接线

- **实验目的：**

采集当前环境的光照强度并在 OLED 显示，显示方式为：Bright-强，Normal-中等，Weak-弱。

- **实验讲解：**

我们先来看看光敏传感器模块的介绍：

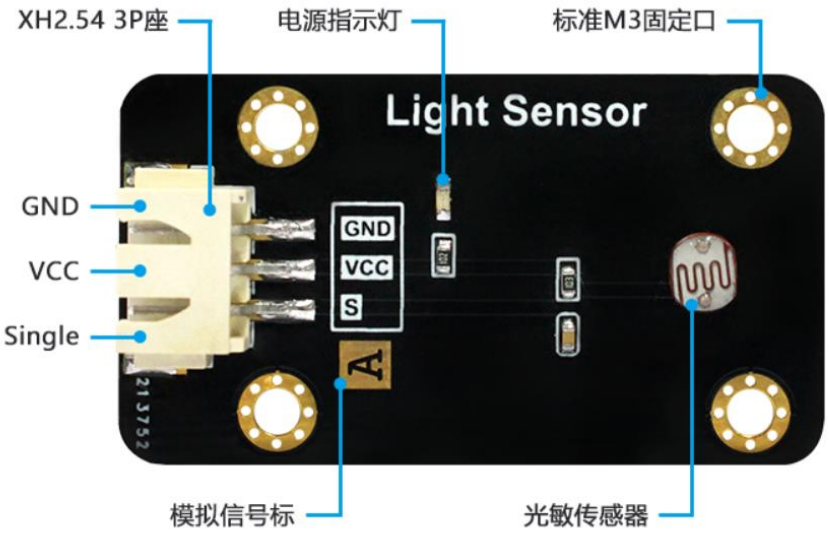


图 19-27 光敏传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	模拟信号：0-3.3V （VCC=3.3V 时）
模块尺寸	4.5*2.5 cm

表 19-2

从上表可以看到，光敏传感器输出的是模拟信号：0-3.3V，这代表了外界的光照强度。接近 0V 时光线最强，接近 3.3V 时，光线最弱。因此我们可以使用**基础实验-ADC** 学习过的内容来编程。

光敏传感器接在传感器接口 2，对应 pyBase 的引脚是“Y12”，即 pyBLE-NRF52840 的‘A3’引脚。由于开发板的 ADC 使用 16bit 精度，即最大值为 2^16-1=65535。因此我们将检测到的数值 0-65535 分成三段，分别代表光线强：【0-21840】，中等：【21840-43680】，弱：【43680-65535】。当然开发者可以根据自己实际情况来调整数值。编程流程图如下：

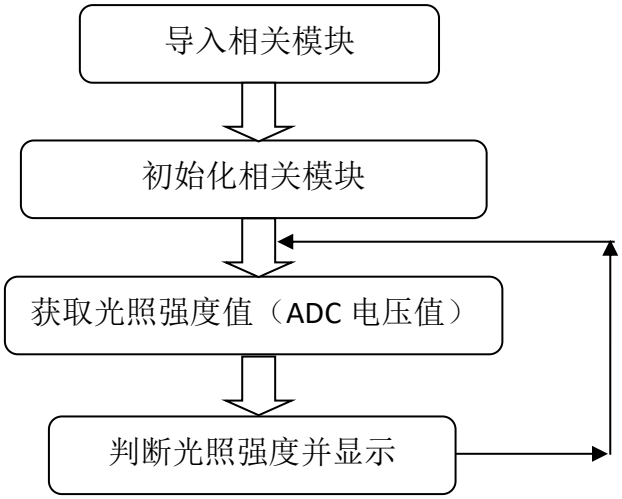


图 19-28 代码编写流程

参考程序代码如下：

```
'''
实验名称：光敏传感器
版本：v1.0
日期：2020.5
作者：01Studio（www.01studio.org）
'''

#导入相关模块
import time,board,busio
from analogio import AnalogIn
import adafruit_ssd1306

#构建 I2C 对象
i2c = busio.I2C(board.SCK, board.MOSI)

#构建 oled 对象,01Studio 配套的 OLED 地址为 0x3C
display = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c, addr=0x3C)
```

```
#清屏
display.fill(0)
display.show()

# 初始化光敏传感器对象，引脚为 A3.
light = AnalogIn(board.A3)

while True:

    #获取光照强度 AD 值
    value=light.value

    #基础信息显示
    display.fill(0) #清屏
    display.text('01Studio', 0,0, 1,font_name='font5x8.bin')
    display.text('Light Test', 0,15, 1,font_name='font5x8.bin')

    #电压信息显示
    display.text(str(value)+' (65535)', 0,40,
                 1,font_name='font5x8.bin')
    display.text(str(round(value*3.3/65535,2))+
                  ' V',0,55,1,font_name='font5x8.bin')

    #判断光照强度，分 3 档显示。
    if 0 < value <=21840:
        display.text('Bright', 60, 55, 1,font_name='font5x8.bin')
    if 21840 < value <= 43680:
        display.text('Normal', 60, 55, 1,font_name='font5x8.bin')
    if 43680 < value <= 65535:
```

```
        display.text('Weak', 60, 55, 1,font_name='font5x8.bin')

    display.show()

    time.sleep(0.5) #检测周期 0.5 秒
```

- **实验结果：**

用手遮挡住光敏传感器，可以见到 OLED 显示光线强势是弱 Weak：

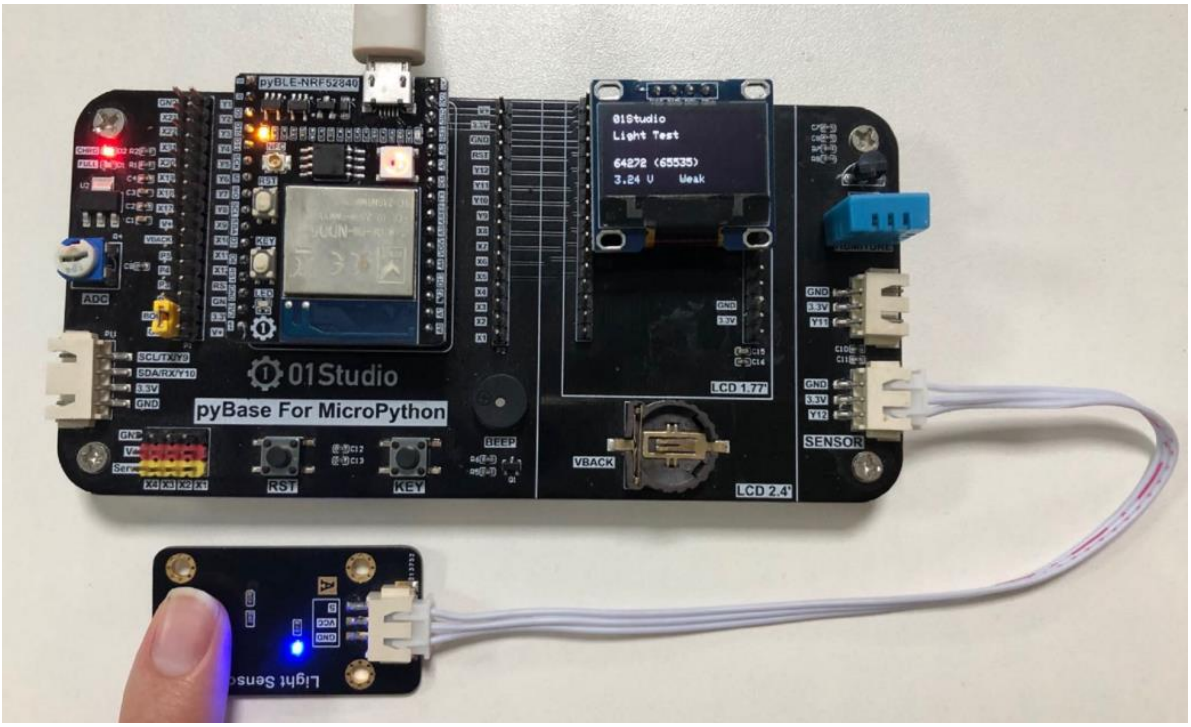


图 19-29 光线强度弱

打开手机手电筒，照在光敏传感器上，可以见到光线强度是强 Bright：

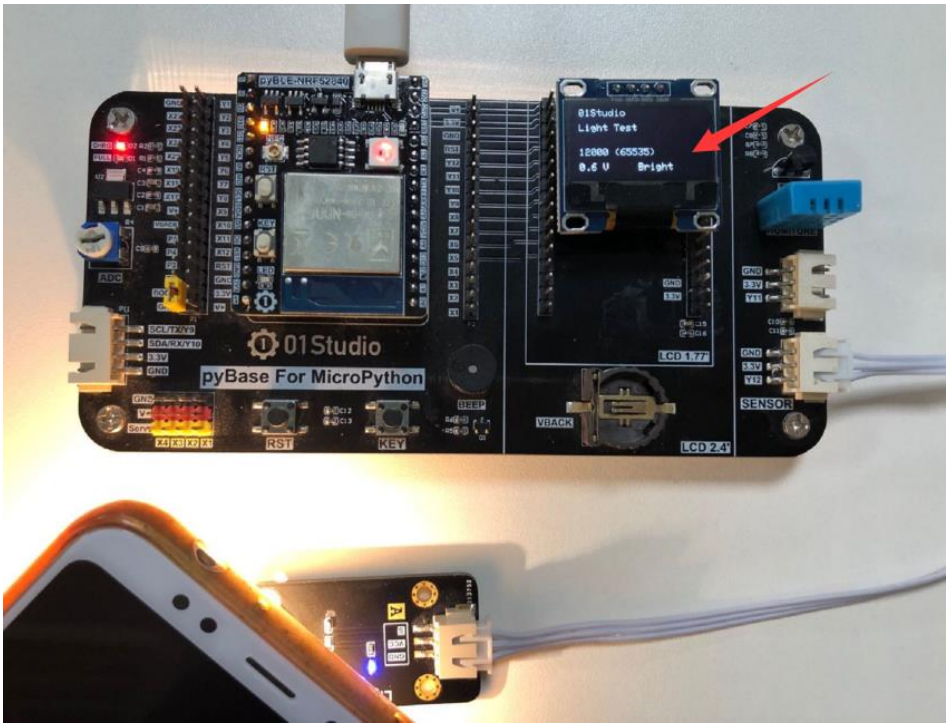


图 19-30 光线强度强

- **总结：**

从实验可以看到，光敏传感器背后的原理是对 ADC 的应用，实现了改功能后。我们可以自行扩展深入，制作自己喜欢的电子产品。

19.5 土壤湿度传感器

- **前言：**

土壤湿度传感器用于检测盆栽泥土的湿度，当泥土干枯时候，我们就需要给植物浇水了。这个用途非常广泛，如自动灌溉。

- **实验平台：**

pyBLE-NRF52840 开发套件和土壤湿度传感器模块。

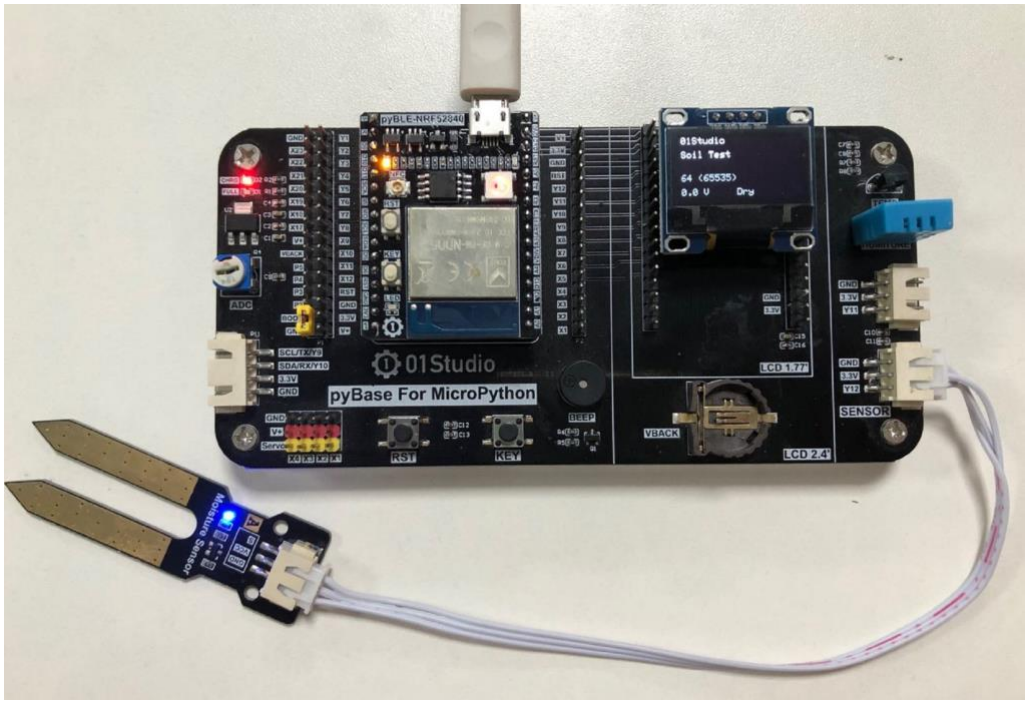


图 19-31 开发套件和传感器连接图

- **实验目的：**

采集盆栽土壤的的光照强度并在 OLED 显示，显示方式为：Dry-干，Normal-中等，Wet-湿。

- **实验讲解：**

我们先来看看土壤湿度传感器模块的介绍：

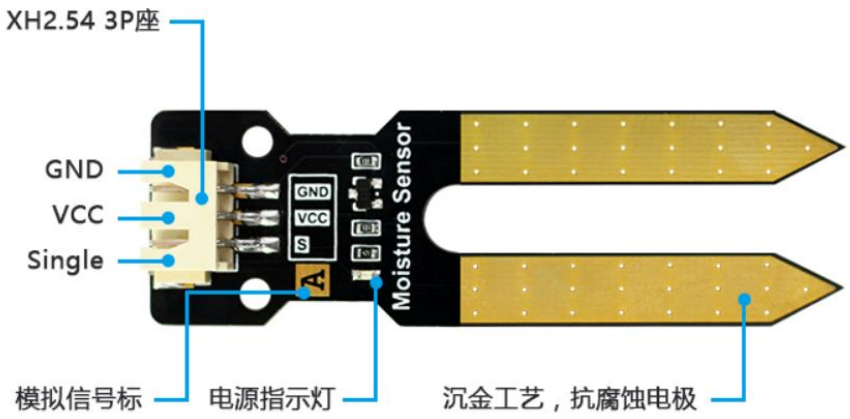


图 19-32 土壤湿度传感器模块

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-40 至 85℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	模拟信号：0-3.3V （VCC=3.3V 时）
模块尺寸	6.6*2.0cm

表 19-3

从上表可以看到，土壤湿度传感器输出的是模拟信号：0-3.3V，这代表土壤的湿度情况。接近 0V 时湿度为干燥，接近 3.3V 时，湿度情况为湿润。因此我们可以使用基础实验-ADC 学习过的内容来编程。

土壤湿度传感器接在传感器接口 2，对应 pyBase 的引脚是“Y12”，也就是 pyBLE-NRF52840 的‘A3’引脚，ADC 为 16bit 精度，即最大值为 2^16-1=4095。然后我们根据实际测试数据将检测到的数值 0-4095 分成三段，分别代表土壤干燥：【0-19952】，中等：【19952-35808】，湿润：【35808-65535】。当然开发者可以根据自己实际情况来调整数值。编程流程图如下：

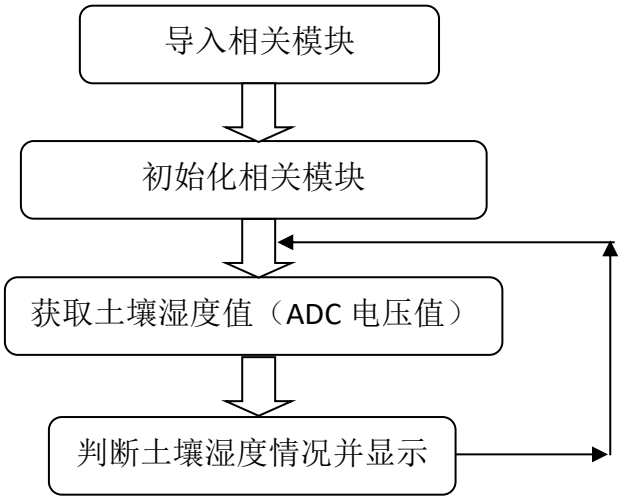


图 19-33 代码编写流程

参考程序代码如下：

```
'''
实验名称：土壤传感器
版本：v1.0
日期：2020.5
作者：01Studio（www.01studio.org）
'''

#导入相关模块
import time,board,busio
from analogio import AnalogIn
import adafruit_ssd1306

#构建 I2C 对象
i2c = busio.I2C(board.SCK, board.MOSI)

#构建 oled 对象,01Studio 配套的 OLED 地址为 0x3C
display = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c, addr=0x3C)
```

```
#清屏
display.fill(0)
display.show()

# 初始化土壤湿度传感器对象，引脚为 A3.
light = AnalogIn(board.A3)

while True:

    #获取光照强度 AD 值
    value=light.value

    #基础信息显示
    display.fill(0) #清屏
    display.text('01Studio', 0,0, 1,font_name='font5x8.bin')
    display.text('Soil Test', 0,15, 1,font_name='font5x8.bin')

    #电压信息显示
    display.text(str(value)+' (65535)', 0,40, 1,font_name='font5x8.bin')
    display.text(str(round(value*3.3/65535,2))+'
V',0,55,1,font_name='font5x8.bin')

    #判断土壤湿度，分 3 档显示。
    if 0 < value <=19952:
        display.text('Dry', 60, 55, 1,font_name='font5x8.bin')
    if 19952 < value <= 35808:
        display.text('Normal', 60, 55, 1,font_name='font5x8.bin')
    if 35808 < value <= 65535:
        display.text('Wet', 60, 55, 1,font_name='font5x8.bin')
```

```
display.show()

time.sleep(0.5) #检测周期 0.5 秒
```

- **实验结果：**

将传感器插到干燥的土壤中，可以见到 OLED 显示干燥-Dry：

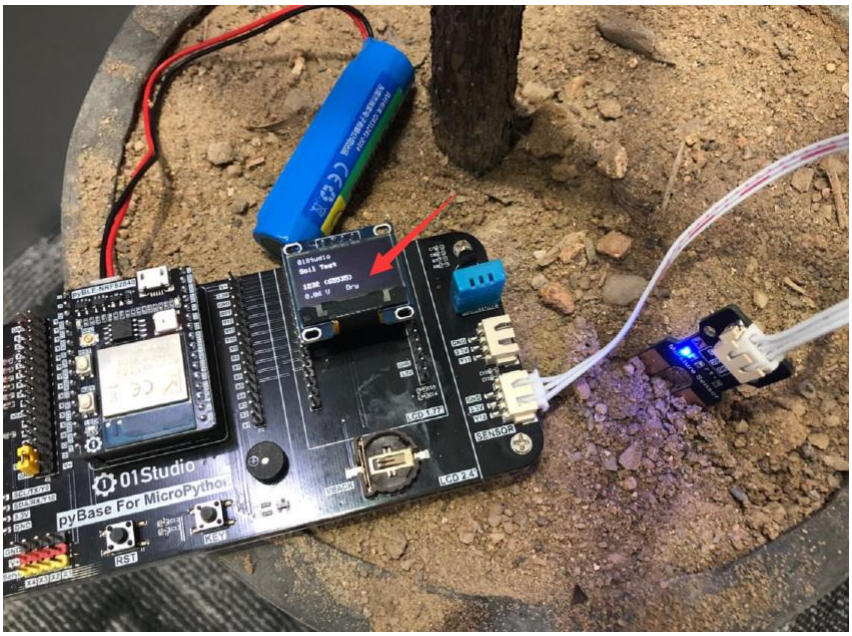


图 19-34 干燥的土壤

从传感器侧面往土壤浇水，让土壤变得湿润。**注意不要浇到传感器电路上！**

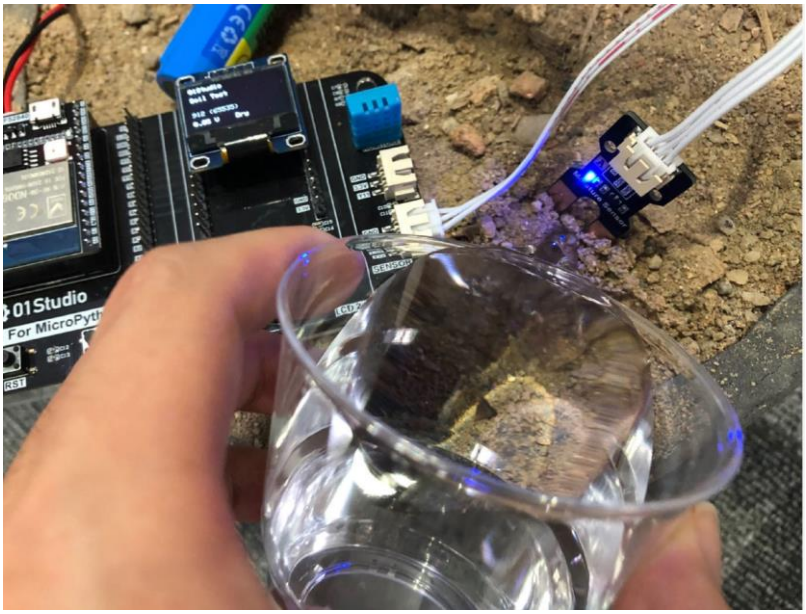


图 19-35 浇水

可以看到 OLED 显示湿润-Wet:

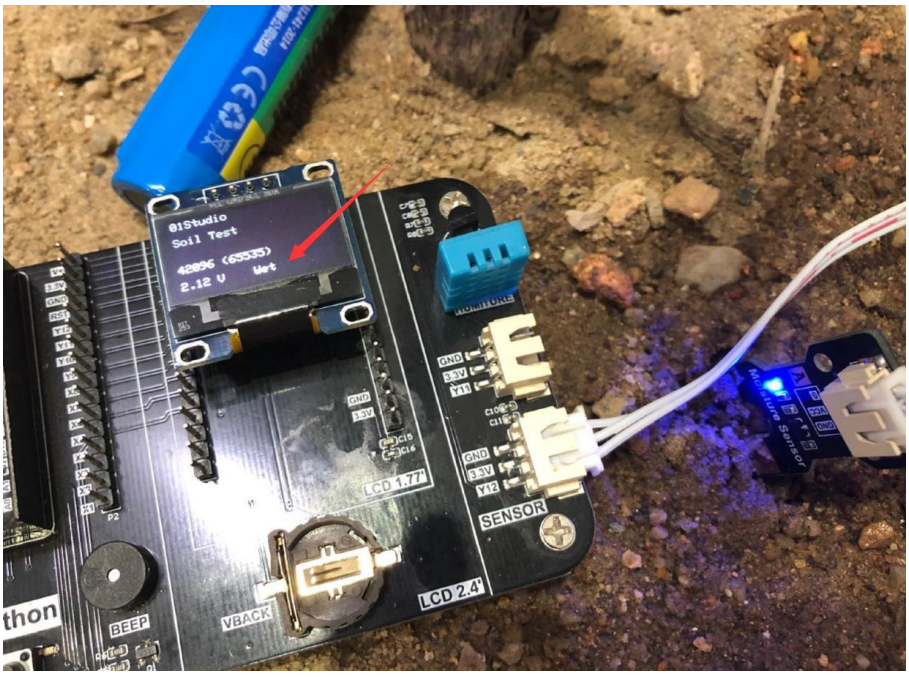


图 19-36 湿润的土壤

- **总结：**

从实验可以看到，土壤湿度传感器背后的原理是对 ADC 的应用，实现了该功能后。我们可以自行扩展深入，制作自己喜欢的电子产品。

19.6 水位传感器

- **前言：**

水位（液位）传感器是一款简单易用、小巧轻便、水位/水滴识别检测传感器，传感器通过一系列的暴露的平行导线线迹测量其水滴/水量大小使得传感器输出的模拟信号产生相应的变化，轻松完成水量到模拟信号的转换，从而判断水位。

- **实验平台：**

PyBLE-NRF52840 开发套件和水位传感器模块。

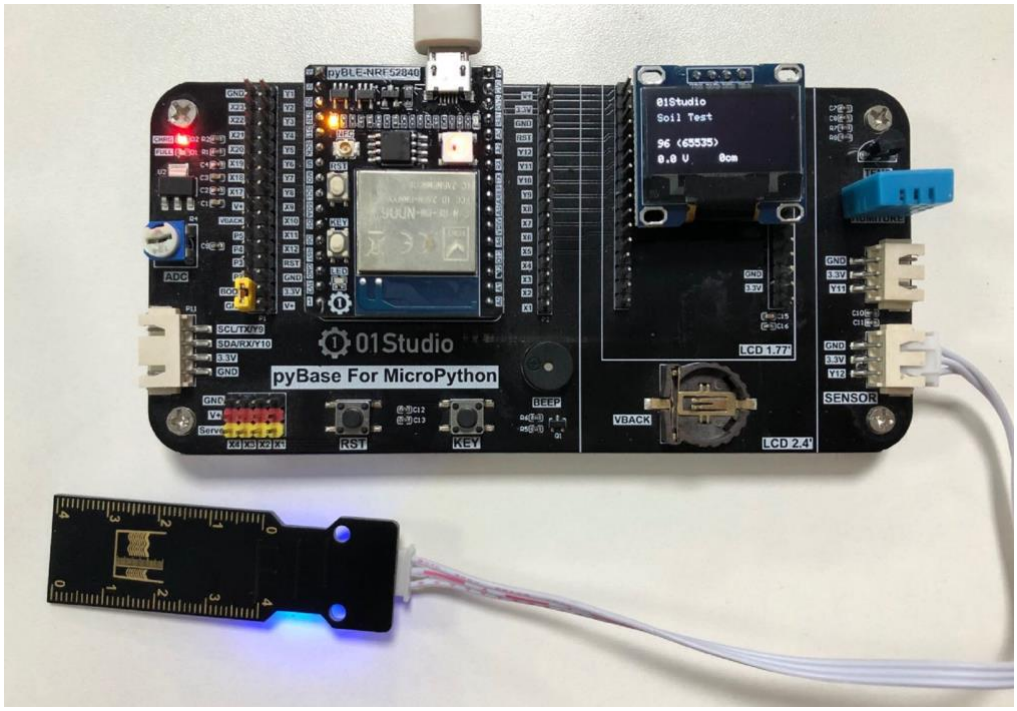


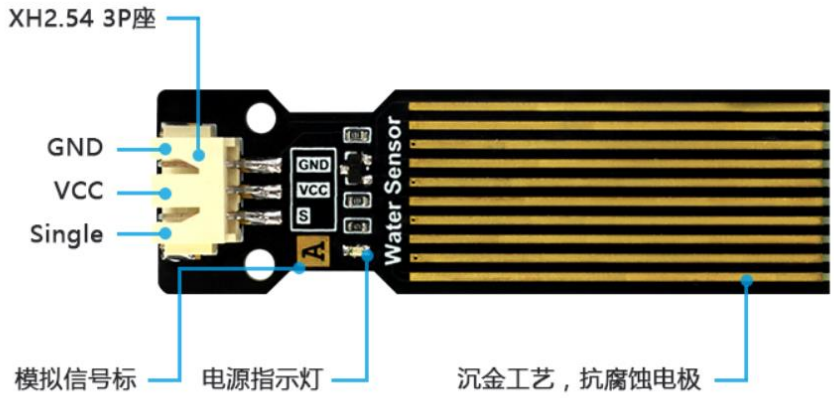
图 19-37 开发套件和水位传感器接线图

- **实验目的：**

测量水位高度并在 OLED 显示，显示方式为：0-4cm。

- **实验讲解：**

我们先来看看水位传感器的介绍：



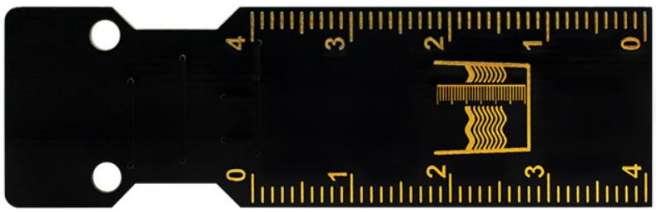


图 19-38 水位传感器模块正反面

功能参数	
供电电压	3.3-5V
工作电流	<20 mA
工作温度	0 至 60℃
接口定义	XH2.54 防呆接口（3Pin）【GND、VCC、Single】
输出信号	模拟信号：0-3.3V （VCC=3.3V 时）
模块尺寸	6.6*2.0cm
注意事项	水位传感器接口旁元件不能接触水，否则容易短路。所以测量时候要注意液体水位的高度。

图 19-39

从上表可以看到，水位传感器的量程为 4cm，输出的是模拟信号：0-3.3V（蒸馏水实测 0-1V）。这代表水位高度情况。接近 0V 时湿度为 0cm 或没有放进液体，电压升高，意味着水位升高。因此我们可以使用基础实验-ADC 学习过的内容来编程。

水位传感器接在传感器接口 2，对应 pyBase 的引脚是“Y12”。也就是 pyBLE-NRF52840 的引脚‘A3’。ADC 为 16bit 精度，即最大值为 2^16-1=65535。我们使用的是普通桶装水，根据实际测试数据将检测到的数值 0-65535 分成 5 档。分别代表水位高度 0cm：【0-9600】，1cm：【9600-14400】，2cm：【14400-19200】，3cm：【19200-20800】，4cm：【>20800】。

当然开发者可以根据自己测量液体不同或其它情况来调整数值。编程流程图如下：

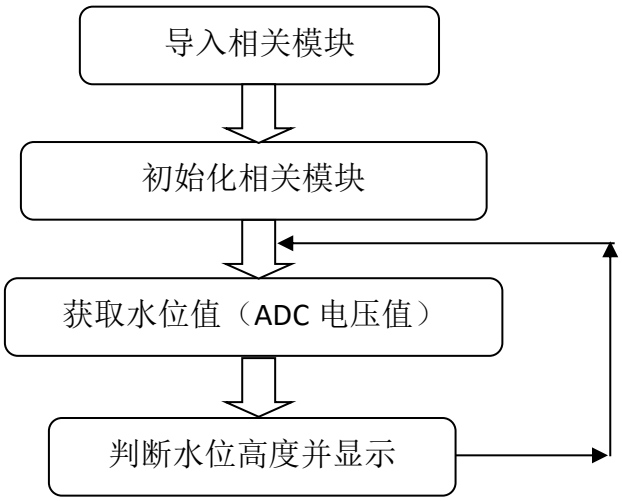


图 19-40 代码编写流程

参考程序代码如下：

```
'''
实验名称：土壤传感器
版本：v1.0
日期：2020.5
作者：01Studio（www.01studio.org）
'''

#导入相关模块
import time,board,busio
from analogio import AnalogIn
import adafruit_ssd1306
```

```
#构建 I2C 对象
i2c = busio.I2C(board.SCK, board.MOSI)

#构建 oled 对象,01Studio 配套的 OLED 地址为 0x3C
display = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c, addr=0x3C)

#清屏
display.fill(0)
display.show()

# 初始化土壤湿度传感器对象，引脚为 A3.
light = AnalogIn(board.A3)

while True:

    #获取光照强度 AD 值
    value=light.value

    #基础信息显示
    display.fill(0) #清屏
    display.text('01Studio', 0,0, 1,font_name='font5x8.bin')
    display.text('Soil Test', 0,15, 1,font_name='font5x8.bin')

    #电压信息显示
    display.text(str(value)+' (65535)', 0,40, 1,font_name='font5x8.bin')
    display.text(str(round(value*3.3/65535,2))+'
V',0,55,1,font_name='font5x8.bin')

    #判断水位高度，分 5 档显示。
```

```
if 0 <= value <=9600:
    display.text('0cm', 60, 55, 1,font_name='font5x8.bin')

if 9600 < value <= 14400:
    display.text('1cm', 60, 55, 1,font_name='font5x8.bin')

if 14400 < value <= 19200:
    display.text('2cm', 60, 55, 1,font_name='font5x8.bin')

if 19200 < value <= 20800:
    display.text('3cm', 60, 55, 1,font_name='font5x8.bin')

if 20800 < value:
    display.text('4cm', 60, 55, 1,font_name='font5x8.bin')

display.show()

time.sleep(0.5) #检测周期 0.5 秒
```

- **实验结果：**

  当没插入水中时，OLED 显示当前水位为 0cm:

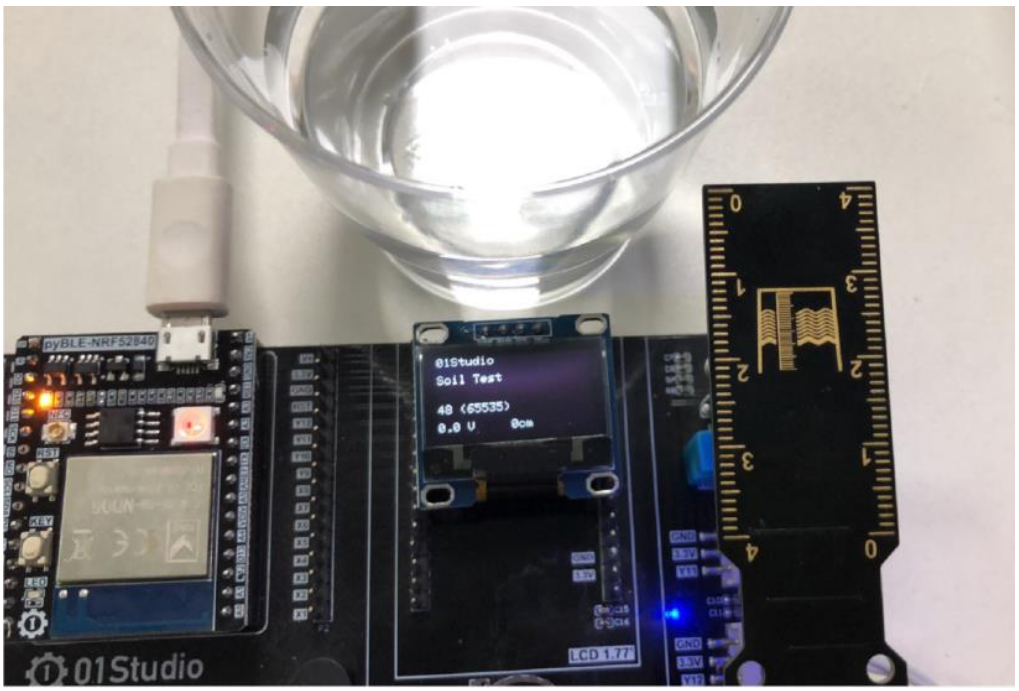


图 19-41

将水位传感器模块慢慢放入水中，可以看到 OLED 显示从 1cm-4cm 变化。

（注意量程是 4cm，请勿将超出的电路部分放置水中以免发生短路。）

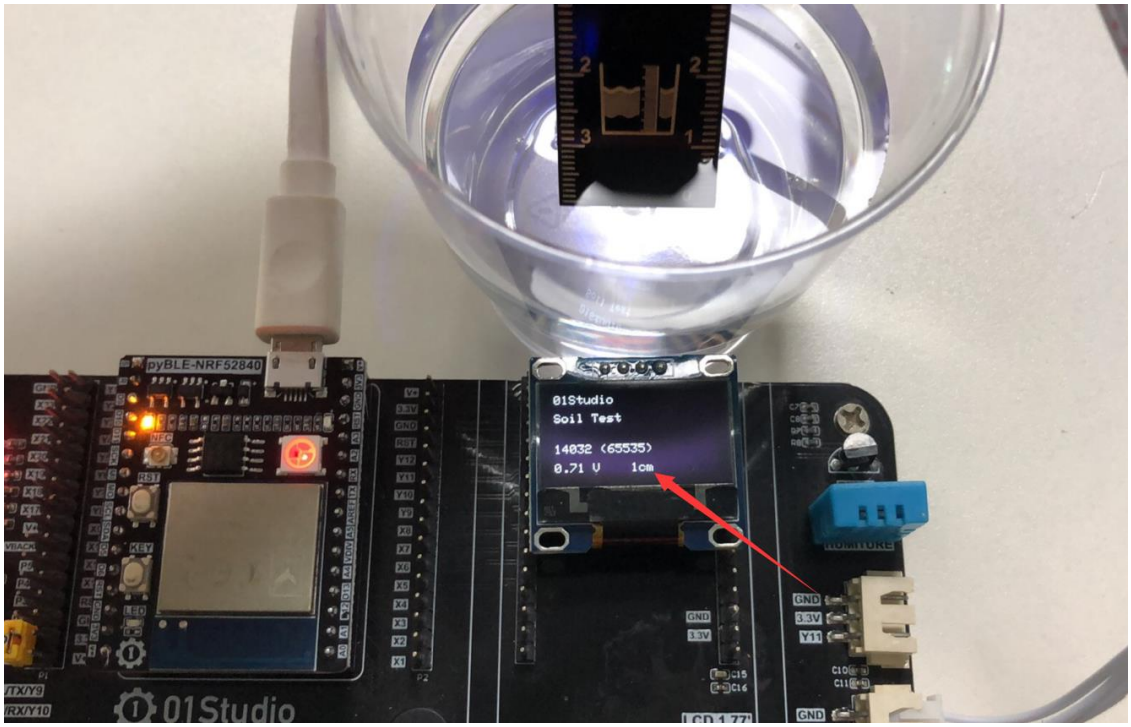


图 19-42 水位接近 1cm

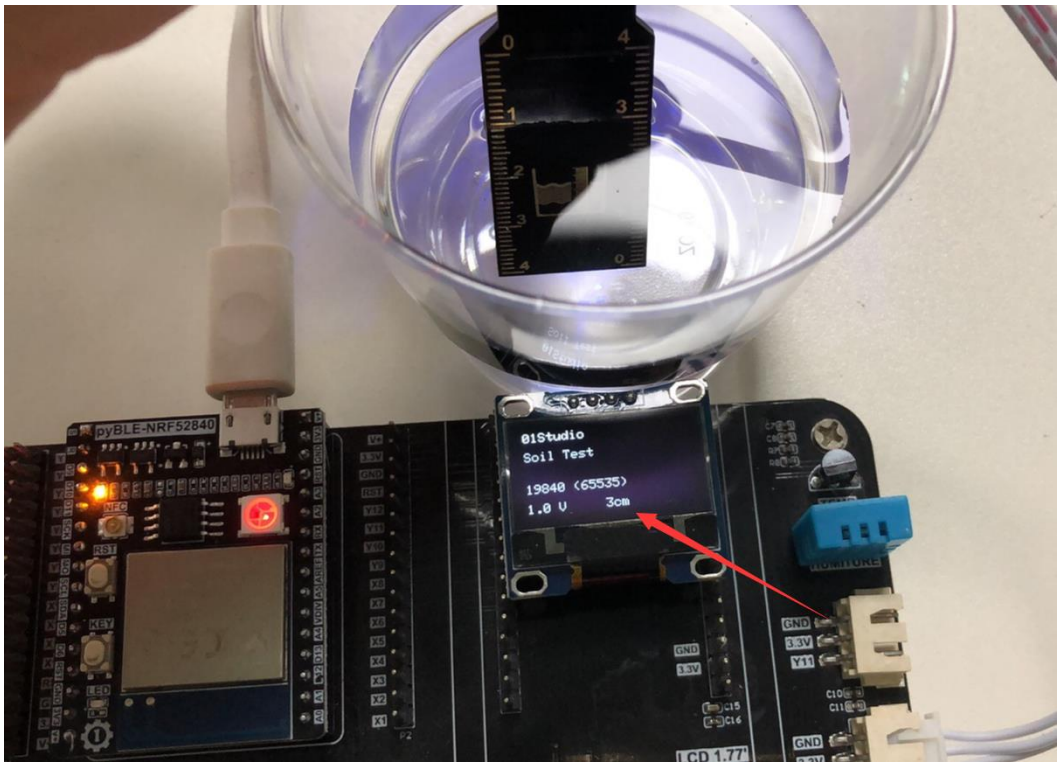


图 19-43 水位接近 3cm

- **总结：**

从实验可以见到，这款水位传感器出了可以用来测量水位（液位）高度外，还可以用来测量室外下雨情况和家里的淹水情况。有兴趣的小伙伴可以动手制作一下！

19.7 大气压强传感器

- **前言：**

本实验中大气压强传感器模块使用的是 BMP280，这是一款专为移动应用而设计的高精度大气压传感器， 传感器模块采用极其紧凑的封装，其小尺寸和低功耗可以应用在手机、 GPS 模块、 手表的电池供电设备中。 BMP280 传感器除了可以测量大气压强，还可以测量温度（温度精度不高）以及通过计算公式来换算出海拔高度。

- **实验平台：**

pyWiFi-ESP32 开发套件和大气压强传感器模块，传感器连接到 I2C/UART 扩展接口。

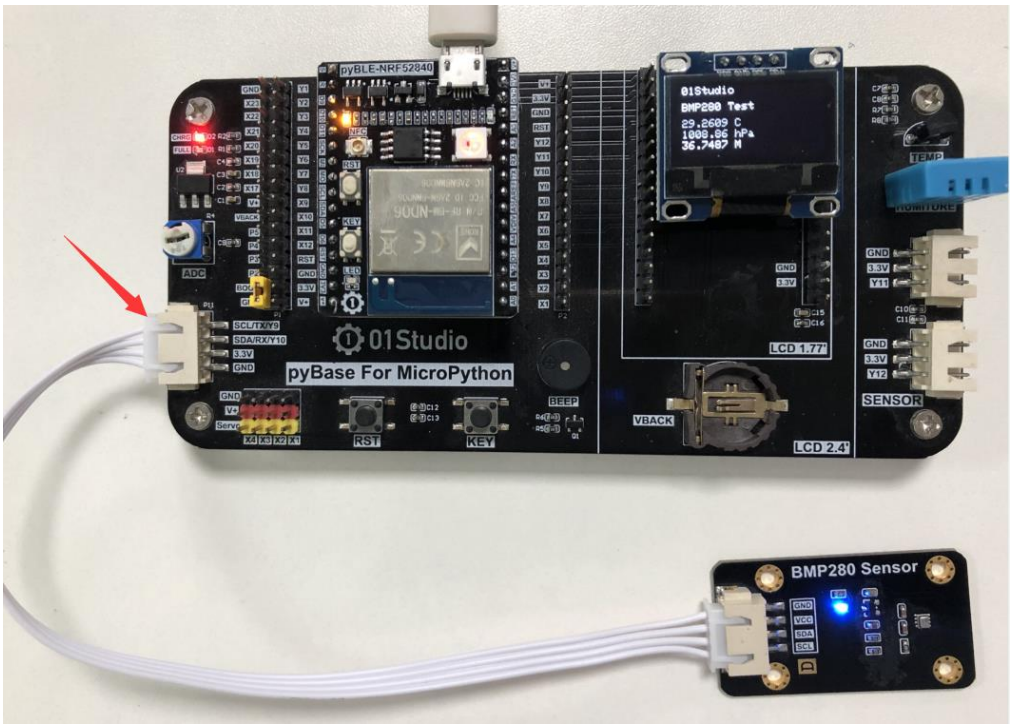


图 19-44 开发套件和大气压强传感器接线方法

- **实验目的：**

编程测量当前环境的大气压强、温度信息，将大气压强通过公式计算出当前海拔高度，并在 OLED 显示。打造自己的气压计！

- **实验讲解：**

我们先来看看大气压强传感器模块的介绍：

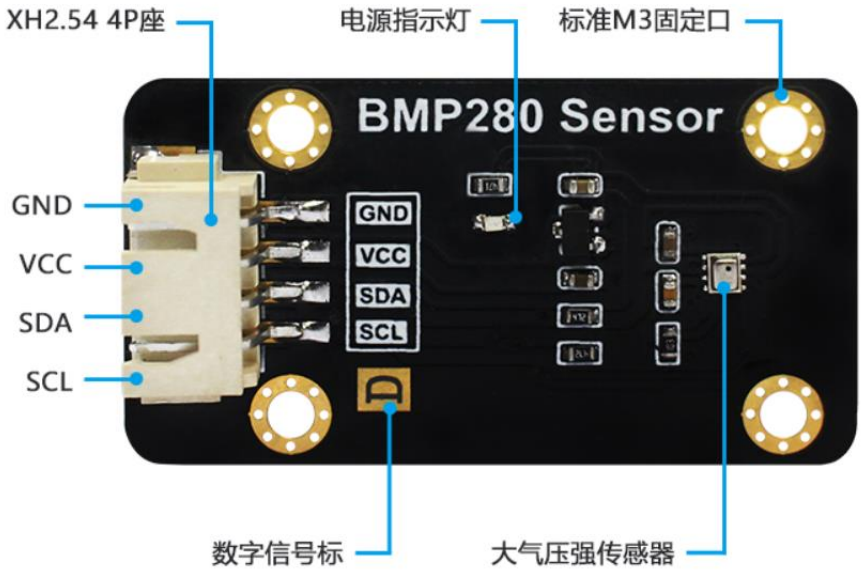


图 19-45 大气压强传感器模块

功能参数	
传感器型号	BMP280
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85℃
接口定义	XH2.54 防呆接口（4Pin）【GND、VCC、SDA、SCL】
通讯信号	I2C 总线
I2C 地址	0x76（BMP280 的 SDO 默认下拉）； 当 BMP280 的 SDO 引脚上拉时，I2C 地址为：0x77
模块尺寸	4.5*2.5cm

图 19-46

从上面介绍可以看到，BMP280 是一款通过 I2C 接口驱动的传感器。连接到 pyBase 左下角的 I2C 外扩接口上。我们通过前面学习的 I2C 接口使用的方式，即可以对该模块实现数据通讯。

标准大气压是指把温度为 0℃、纬度 45 度海平面（海拔为 0 米）上的气压称为 1 个大气压，其数值为 101325 帕斯卡（Pa）。

大气压同海拔高度的关系：P=P0×（1-H/44300)^5.256

因此计算高度公式为：H=44300*(1- (P/P0)^(1/5.256))

式中：H 为海拔高度，P0=大气压（0℃，101325Pa）

从上面公式可以看到，高度是通过大气压强换算出来的，从物理学的角度我们可以知道，高度越高的地方，空气越稀薄，大气压强越低。通过气压的变化我们就可以计算出海拔高度；但是这存在特定条件，那就是温度为 0℃的时候，而温度越高的地方，空气越稀薄，大气压强就越低。因此高度数据理论上需要做温度补偿，因此本实验的高度值换算存在轻微误差。有兴趣的小伙伴可以自行深入研究。

MicroPython 的强大之处是其有丰富的模块和函数库，一旦模块建立了，那么后来者使用起来就非常简单，无须再去做底层驱动的开发。从而实现面向对象的编程，而当有需要的时候又可以去改底层代码，可以说是进可攻退可守，非常灵活。在这里我们直接调用已经编写好的驱动模块，该文件实现了对大气压、温度、高度的测量和计算。用户直接使用即可，具体如下：

构造函数
bmp280= adafruit_bmp280.Adafruit_BMP280_I2C(i2c,address=0x77)
构建 BMP280 传感器对象。 【i2c】定义的 I2C 总线； 【address】传感器总线地址，默认是 0x77
使用方法
bmp280.temperature
获取温度；
bmp280.pressure
获取大气压强值；

bmp280.altitude
获取高度值

表 19-4 大气压强传感器 BMP280 对象

理解了传感器原理和模块使用方法后，我们可以整理出编程思路，流程图如下：

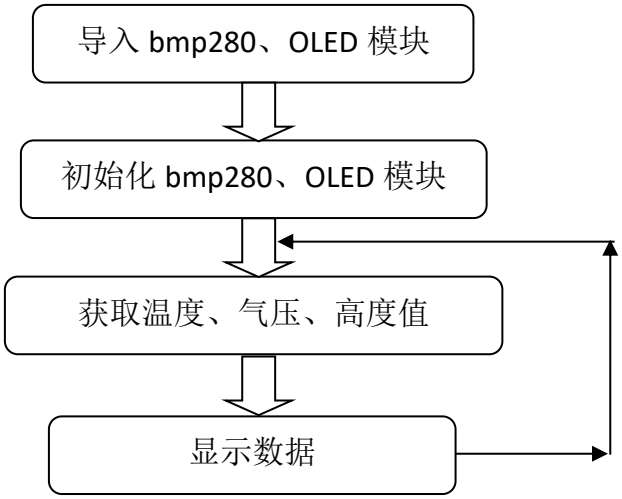


图 19-47 代码编写流程

参考程序代码如下：

```
'''
实验名称：大气压强传感器 BMP280
版本：v1.0
日期：2020.5
作者：01Studio（www.01studio.org）
'''

#导入相关模块
import time,board,busio
from analogio import AnalogIn
import adafruit_ssd1306,adafruit_bmp280
```

```
#构建 I2C 对象
i2c = busio.I2C(board.SCK, board.MOSI)
i2c2 = busio.I2C(board.TX, board.RX)

#构建 oled 对象,01Studio 配套的 OLED 地址为 0x3C
display = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c, addr=0x3C)

#清屏
display.fill(0)
display.show()

#构建 BMP280,注意 01studio 模块地址为 0x76.
bmp280 = adafruit_bmp280.Adafruit_BMP280_I2C(i2c2,address=0x76)

# 当地海平面标准大气压
bmp280.sea_level_pressure = 1013.25



while True:

    #获取光照强度 AD 值
    temperature=bmp280.temperature
    pressure=bmp280.pressure
    altitude=bmp280.altitude

    #基础信息显示
    display.fill(0) #清屏
    display.text('01Studio', 0,0, 1,font_name='font5x8.bin')
    display.text('BMP280 Test', 0,15, 1,font_name='font5x8.bin')
```

```
#电压信息显示
display.text(str(temperature)+' C', 0,30, 1,font_name='font5x8.bin')
display.text(str(pressure)+' hPa', 0,40, 1,font_name='font5x8.bin')
display.text(str(altitude)+' M', 0,50, 1,font_name='font5x8.bin')
display.show()

time.sleep(2) #检测周期 2 秒
```

- **实验结果：**

可以看到 oled 上显示采集到的相关信息。

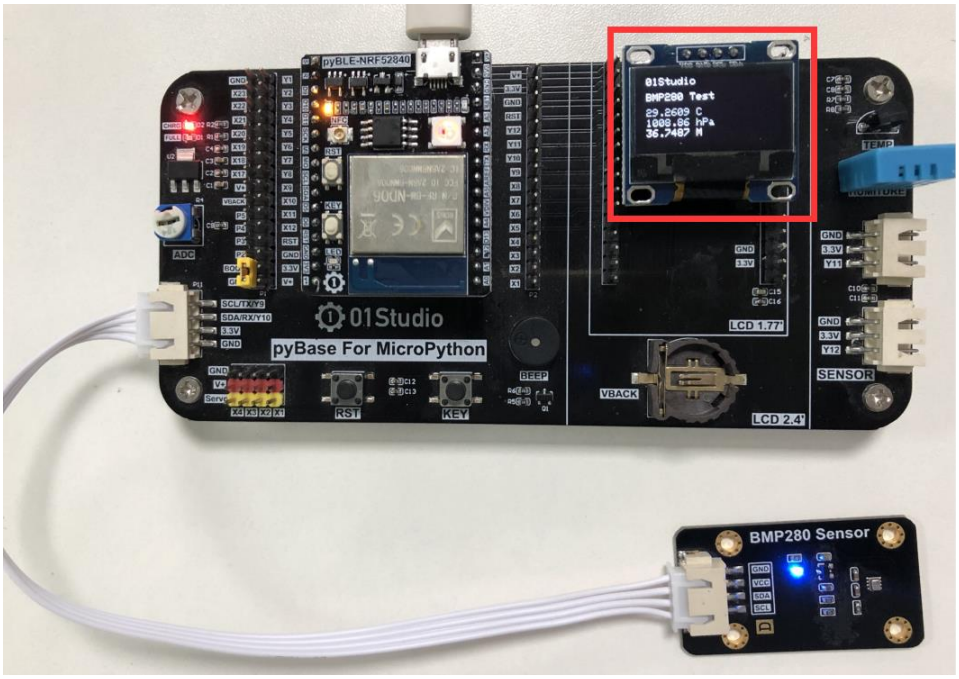


图 19-48 气压测量

开发板接上锂电池供电，尝试上下楼，会发现气压和海拔数值相应的变化。

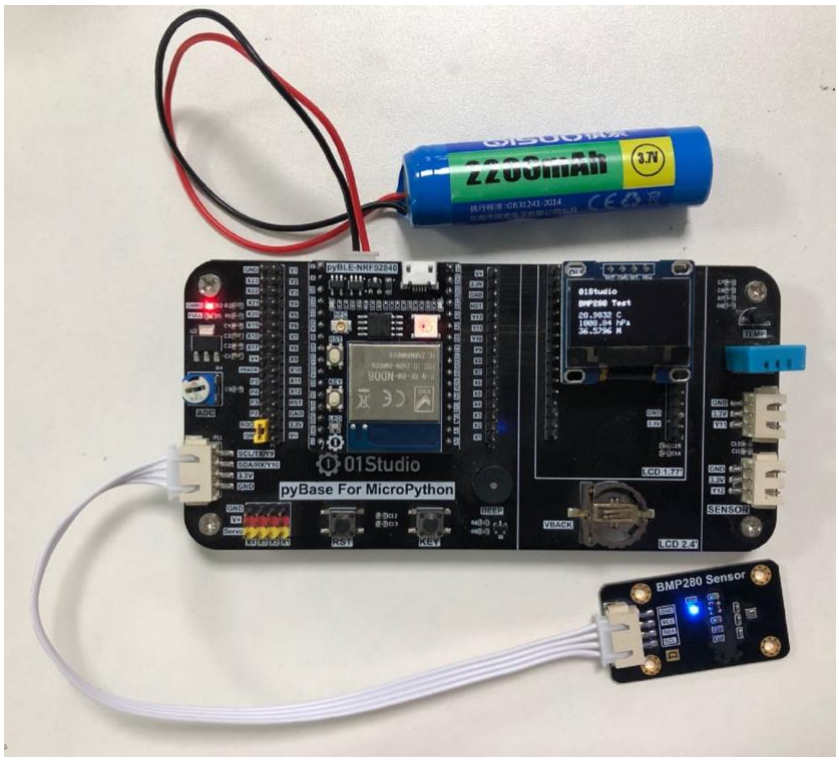


图 19-49 锂电池供电

- **总结：**

本节实现了大气压强传感器模块 BMP280 的应用，而且实验结果的精度很高，有兴趣的小伙伴可以结合自己情况，打造一个属于自己的气压计，然后到户外暴走！

19.8 超声波传感器

- **前言：**

超声波传感器是一款测量距离的传感器。其原理是利用声波在遇到障碍物反射接收结合声波在空气中传播的速度计算的得出。在测量、避障小车，无人驾驶等领域都有相关应用。

- **实验平台：**

PyBLE-NRF52840 开发套件和超声波传感器模块，传感器模块连接到 I2C/UART 扩展接口。

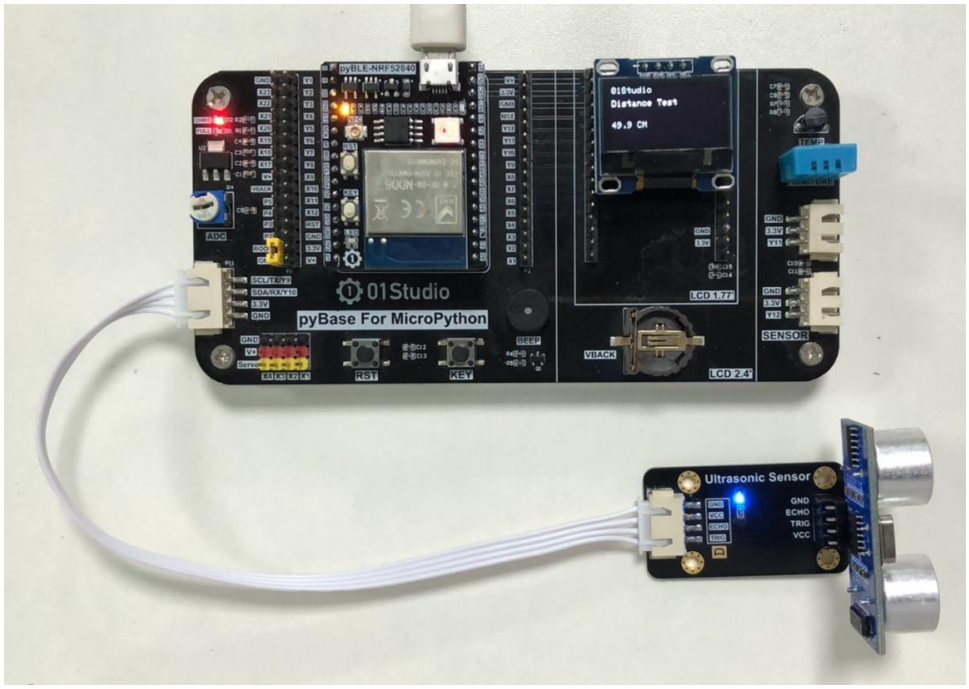


图 19-50 开发套件与传感器连接

- **实验目的：**

测量距离并在 OLED 上显示相关数据。

- **实验讲解：**

我们先来看看超声波传感器模块的介绍：

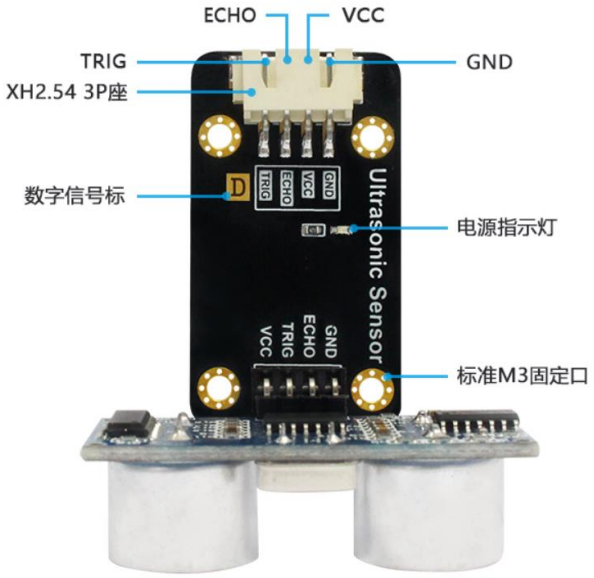


图 19-51 超声波模块

功能参数	
传感器型号	HCSR04
供电电压	3.3-5V
工作电流	<20 mA
工作温度	-20 至 85℃
接口定义	XH2.54 防呆接口（4Pin） 【GND、VCC、ECHO、TRIG】
通讯信号	IO 数字接口
测量距离	2-450 cm
测量精度	0.5 cm
整体尺寸	5.5*4.5*3.0 cm

表 19-5

超声波传感器模块使用两个 IO 口分别控制超声波发送和接收，工作原理如下：

11. 给超声波模块接入电源和地；
12. 给脉冲触发引脚（trig）输入一个长为 20us 的高电平方波；

13. 输入方波后，模块会自动发射 8 个 40KHz 的声波，与此同时回波引脚（echo）端的电平会由 0 变为 1；（此时应该启动定时器计时）
14. 当超声波返回被模块接收到时，回波引 脚端的电平会由 1 变为 0；（此时应该停止定时器计数），定时器记下的这个时间即为超声波由发射到返回的总时长；
15. 根据声音在空气中的速度为 340 米/秒，即可计算出所测的距离。

要学习和应用传感器，学会看懂传感器的时序图是很关键的，所以我们来看一下超声波传感器 HCSR04 的时序触发图。

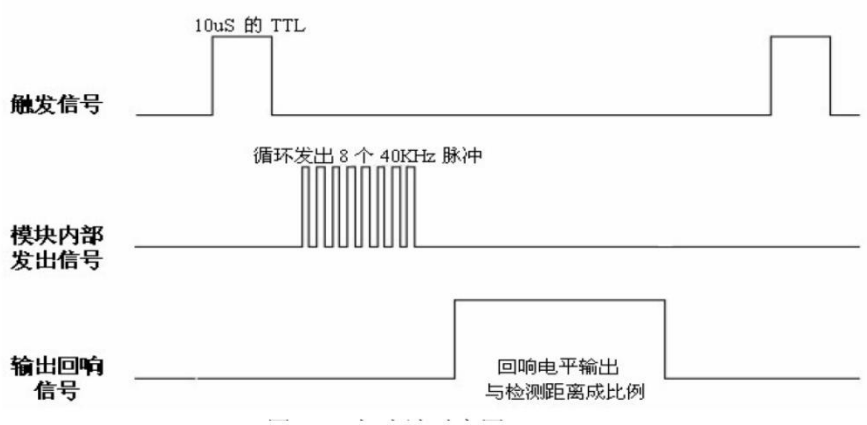


图 19-52 超声波传感器 HCSR04 时序图

以上普及了超声波传感器的原理，我们已经将其集成在 adafruit_hcsr04.py 文件，如想了解代码原理可以打开 adafruit_hcsr04.py 文件查看代码实现原理。使用 MicroPython 开发的用户只需要直接使用即可。使用方法如下：

构造函数
sonar = adafruit_hcsr04.HCSR04(trigger_pin, echo_pin)
构建 HCSR04 超声波传感器对象。 【trigger_pin】定义的 trigger 引脚； 【echo_pin】定义 echo_pin 引脚；
使用方法
`sonar.distance`
获取测量距离，单位 cm

表 19-6 超声波传感器 HCSR04 对象

从上表看到，超声波传感器的使用方法非常简单，使用 2 个代码就完成对距离信息的采集，代码编写流程如下：

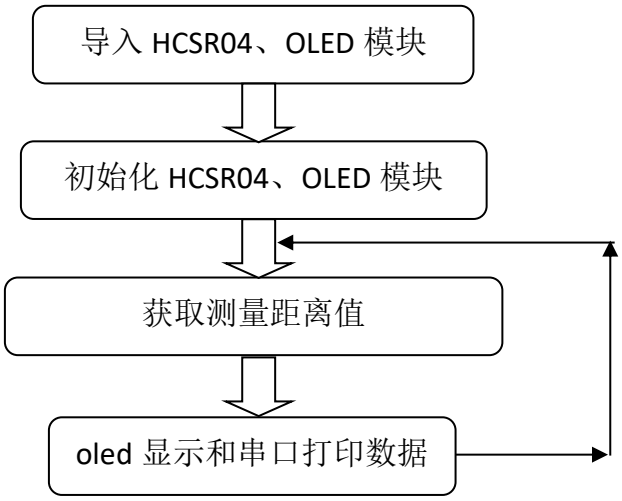


图 19-53 代码编写流程

主程序 main.py 参考代码如下：

```
'''
实验名称：大气压强传感器 BMP280
版本：v1.0
日期：2020.5
作者：01Studio（www.01studio.org）
'''

#导入相关模块
import time,board,busio
from analogio import AnalogIn
import adafruit_ssd1306,adafruit_hcsr04

#构建 I2C 对象
i2c = busio.I2C(board.SCK, board.MOSI)
```

```
#构建 oled 对象,01Studio 配套的 OLED 地址为 0x3C
display = adafruit_ssd1306.SSD1306_I2C(128, 64, i2c, addr=0x3C)

#清屏
display.fill(0)
display.show()

#构建超声波传感器对象
sonar = adafruit_hcsr04.HCSR04(trigger_pin=board.TX, echo_pin=board.RX)

while True:

    #获取距离值
    distance=sonar.distance

    #基础信息显示
    display.fill(0) #清屏
    display.text('01Studio', 0,0, 1,font_name='font5x8.bin')
    display.text('Distance Test', 0,15, 1,font_name='font5x8.bin')

    #距离信息显示
    display.text(str(round(distance,1))+' CM', 0,40,
1,font_name='font5x8.bin')
    display.show()

    print(str(round(distance,1))+' CM')

    time.sleep(0.2) #检测周期 0.2 秒
```

- **实验结果：**

在超声波传感器 30cm 外放置障碍物，可以看到 OLED 显示屏实时显示距离数据约为 30cm。

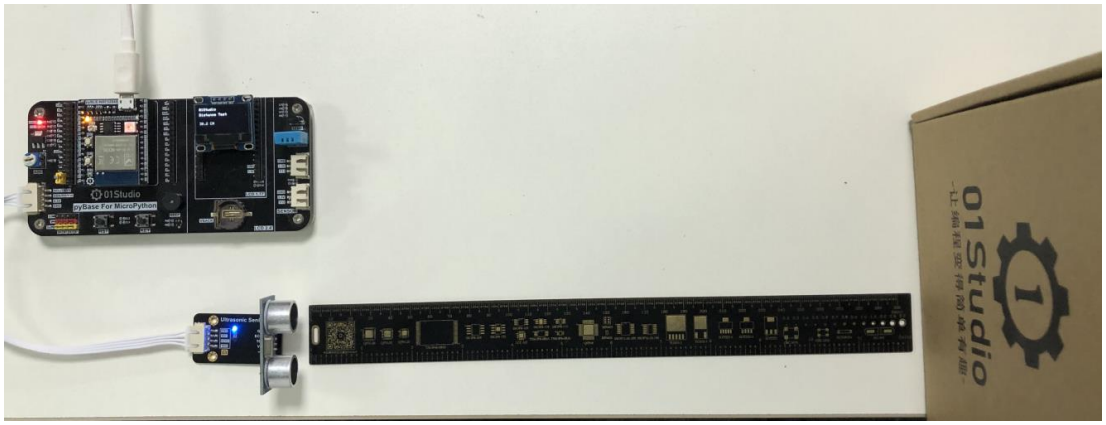


图 19-54 障碍物距离为 30cm

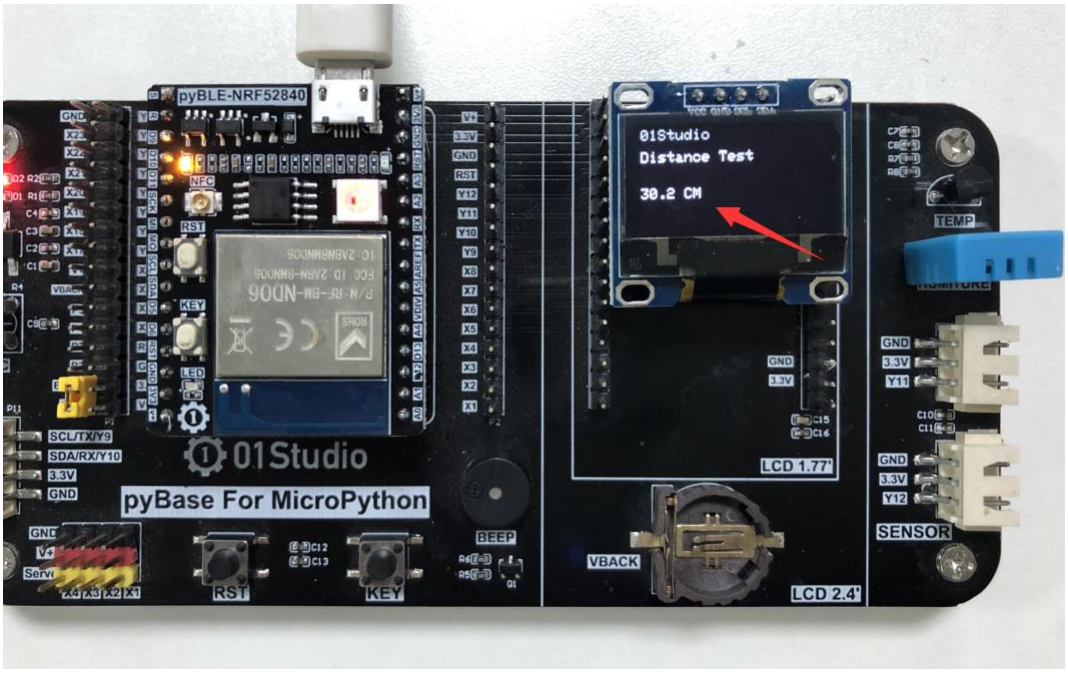


图 19-55 实验结果约为 30cm

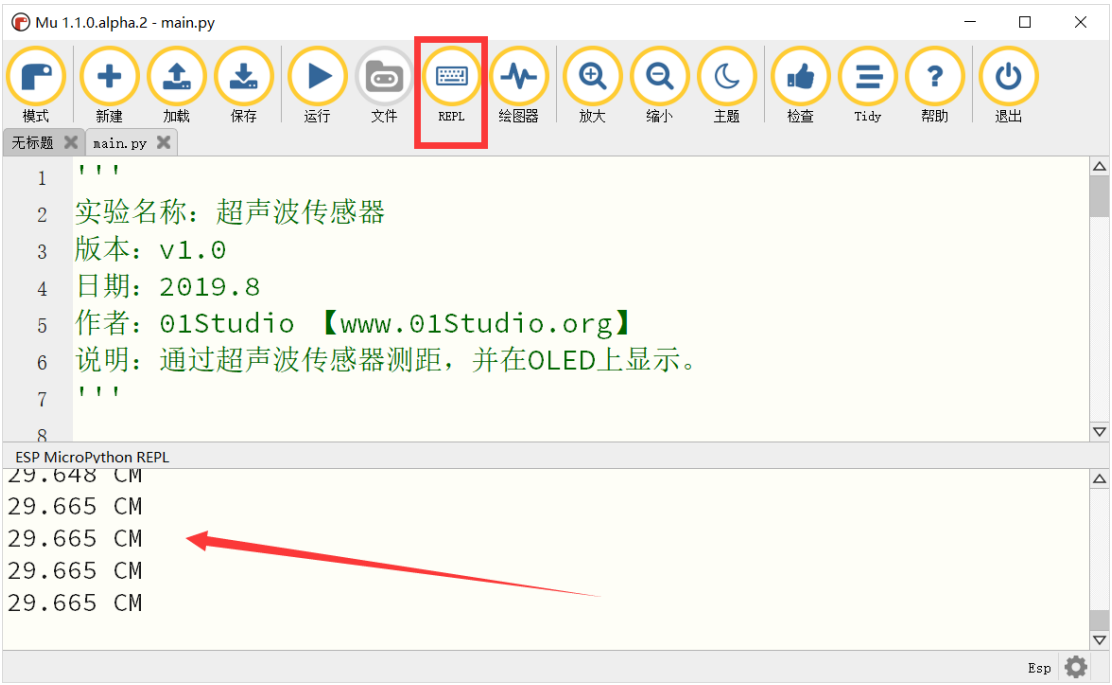


图 19-56　REPL 同步打印实验结果

- **总结：**

出去 OLED 显示代码，我们实际上只用了 2 行代码：初始化和调用测量函数就实现了对超神波传感器测距的应用。这让我们再一次感受到了 MicroPython 的魅力。赶快动作制作自己的避障小车和其他好玩的创作吧。

第20章　拓展实验

在学习了 MicroPython 前面内容后，相信我们已经对其编程方法有了一定的认识，那么除了较为简单的基础实验外，我们还可以使用 MicroPython 开发板连接非常多的外接设备，如常见的舵机、LCD 等。

这些外设的编程难度分两块：如果是仅仅面向应用的话，那么我们可以使用非常简单的方式就用起来；由于代码是完全开源，所以如果想深入学习，则可以看一下驱动代码的实验原理，甚至自行编写驱动程序。从而一起丰富 MicroPython 的周边的生态产品。

20.1 舵机

- **前言：**

舵机又叫伺服电机，是一个可以旋转特定角度的电机，可转动角度通常是 90°、180° 和 360°（360° 可以连续旋转）。我们看到的机器人身上就有非常多的舵机，它们抬手或者摇头的动作往往是通过舵机完成，因此机器人身上的舵机越多，意味着动作越灵活。

- **实验平台：**

PyBLE-NRF52840 开发套件和舵机。

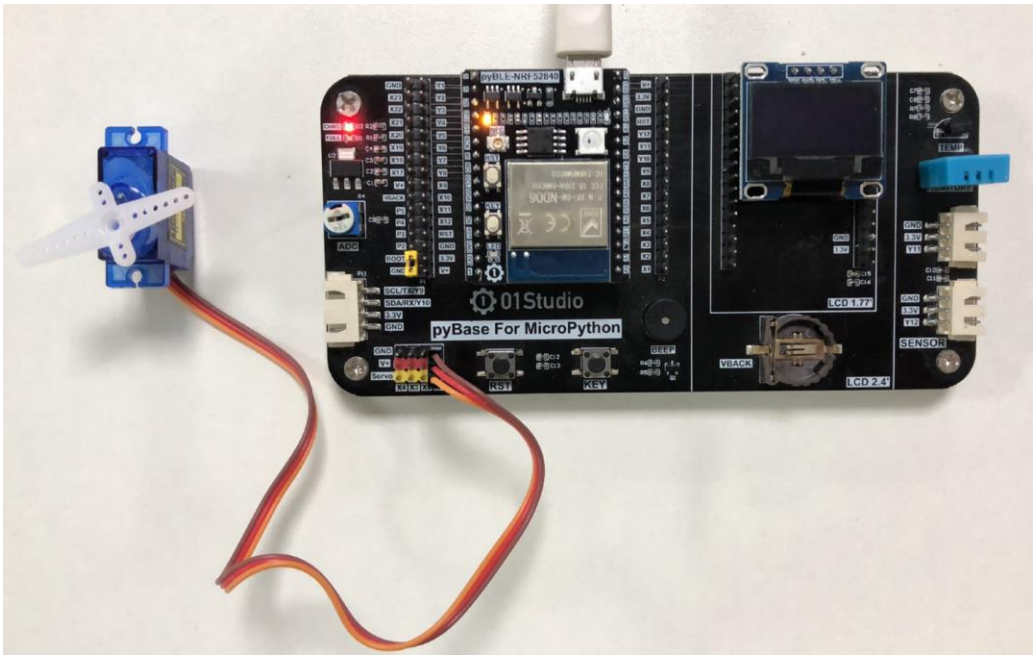


图 20-1 舵机接线

通常情况下：**黑色表示 GND，红色表示 VCC，橙色表示信号线。**

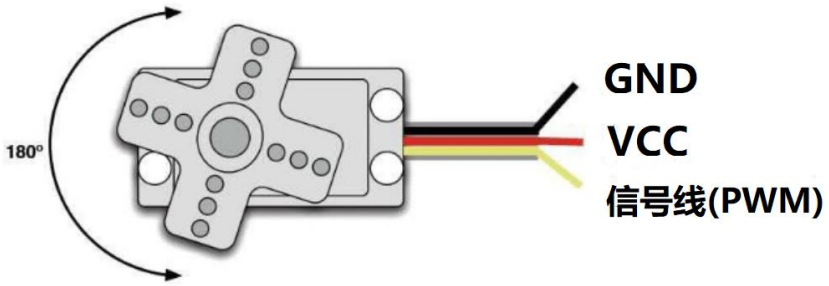


图 20-2 舵机示意图

- **实验目的：**

通过编程实现对舵机的控制。

- **实验讲解：**

伺服电机对象通过 3 线（信号，电源，地）控制，pyBase 上有 4 个位置可以插这些电机，分别是 X1-X4 引脚。对应 pyBLE-NRF52840 的 A0、A1、D12、D13 引脚。

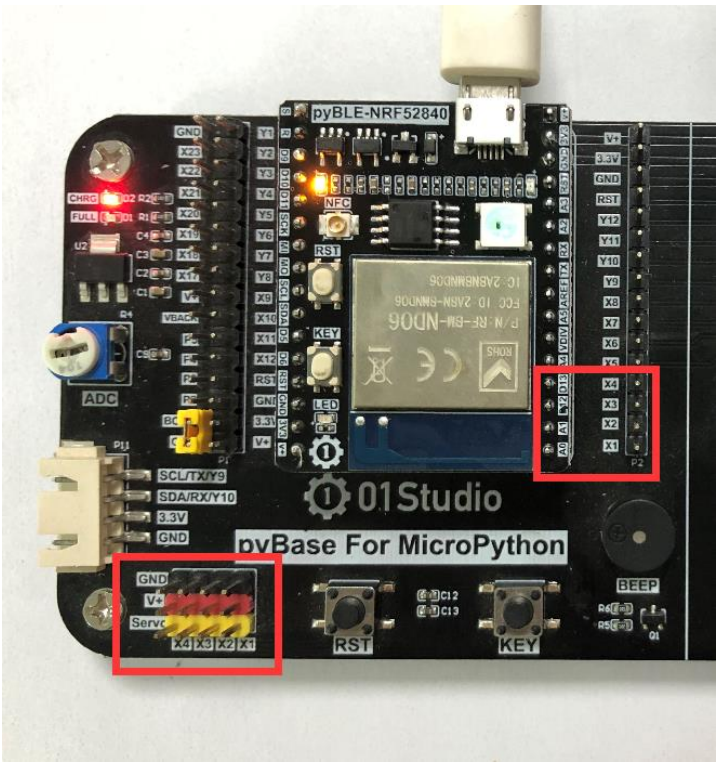


图 20-3　pyBase 的 4 路舵机接口

180°舵机的控制一般需要一个 20ms 左右的时基脉冲，该脉冲的高电平部分一般为 0.5ms-2.5ms 范围内的角度控制脉冲部分，总间隔为 2ms。以 180 度角度伺服为例，在 MicroPython 编程对应的控制关系是从-90°至 90°，示例图如下：

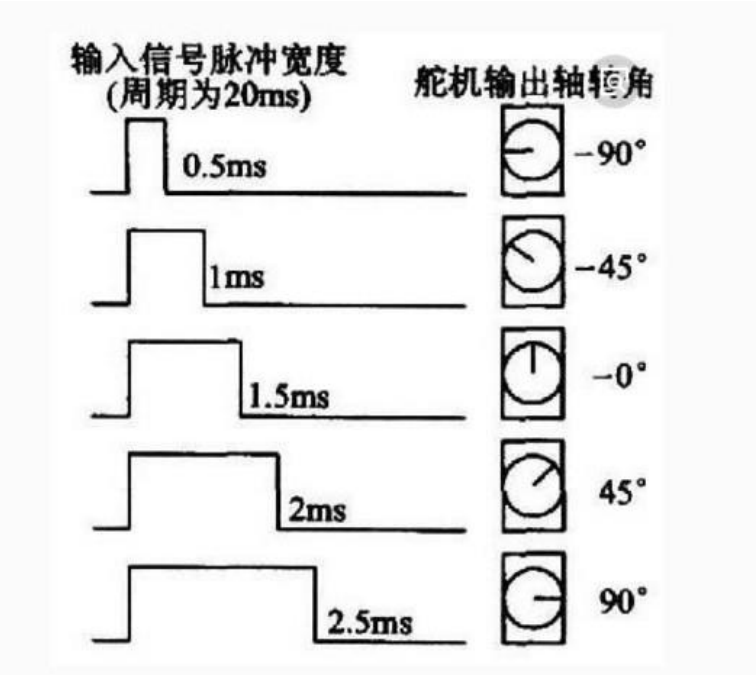


图 20-4　PWM 与角度关系

而对于 360°连续旋转舵机，上面的脉冲表则对应从正向最大速度旋转到反向最大速度旋转的过程。

如果过你学习过前面基于 STM32 平台的舵机实验，那就知道在 STM32 平台集成了舵机模块，使用起来非常方便。而当前的 NRF2840 也一样，原理是通过产生不同的 PWM 信号来控制舵机，可以通过写好的库函数来应用。代码编程流程图如下：

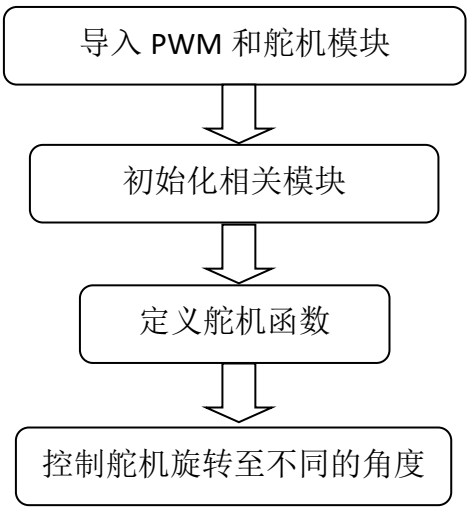


图 20-5 编程流程图

参考代码如下：

```
'''
实验名称：舵机控制
版本：v1.0
日期：2020.5
作者：01Studio 【www.01Studio.org】
说明：通过编程控制 180°舵机旋转到不同角度（360°连续旋转舵机表现为不同旋转方向和速度）
'''

import time,board,pulseio
from adafruit_motor import servo

#构建 PWM 对象，A0 引脚。
pwm = pulseio.PWMOut(board.A0, duty_cycle=2 ** 15, frequency=50)

#构建 Servo 对象 Servo1。参数适用于 SG90 舵机
Servo1 = servo.Servo(pwm,min_pulse=500, max_pulse=2500)

#180°舵机表现为旋转到不同角度；360°连续旋转舵机表现为不同旋转方向和速度
while True:

    Servo1.angle = 0
    time.sleep(1)

    Servo1.angle = 45
    time.sleep(1)
```

```
Servo1.angle = 90
time.sleep(1)


Servo1.angle = 135
time.sleep(1)


Servo1.angle =180
time.sleep(1)
```

- **实验结果：**

将 180° 舵机插到 pyBase 的 X1 的三线接口，下载程序。可以看到舵机依次旋转至不同角度。

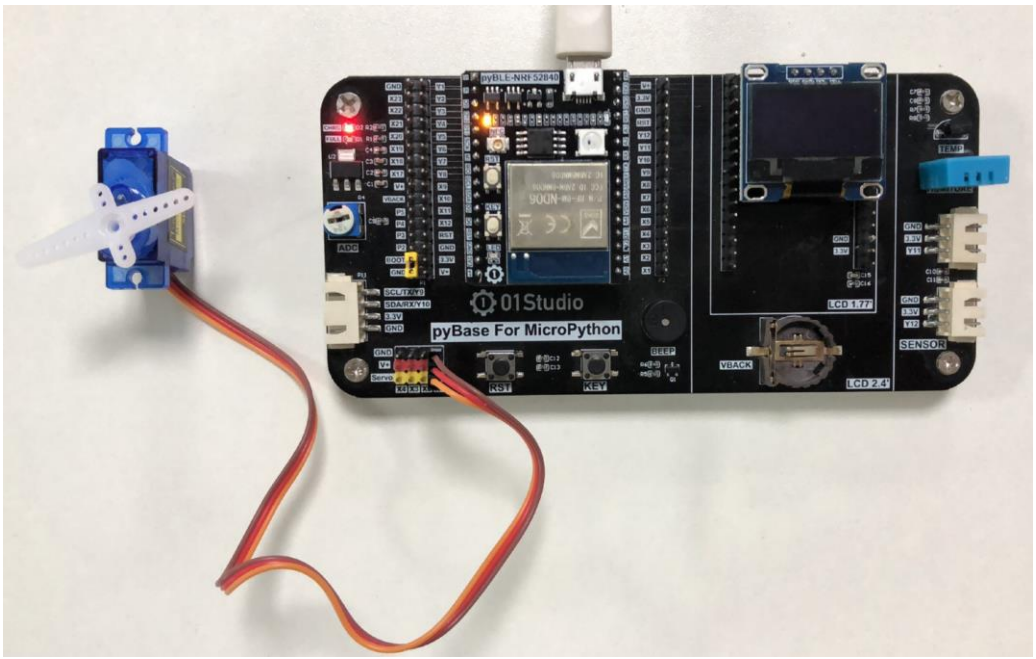


图 20-6 舵机角度旋转至不同角度

- **实验拓展：**

我们刚刚实现了 180° 舵机的角度控制，现在来做一下 360° 连续旋转舵机的实验，360° 连续旋转舵机可以实现直流减速电机功能，用在小车或者航模上。

实验的代码不变，角度【0 至 180】代表旋转方向和速度值大小。插上 360° 连续旋转舵机。可以看到舵机的旋转速度和方向逐渐变变化。

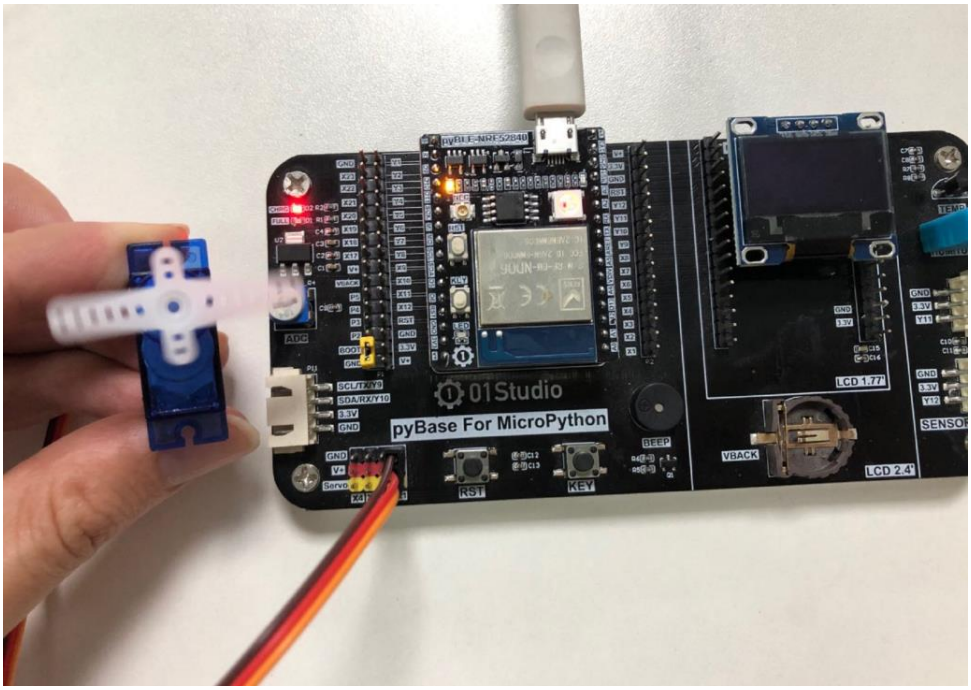


图 20-7　360°连续旋转舵机

- **总结：**

通过本节我们学会了使用不同类型的舵机，通过多路电机的组合使用，可以实现模型、航模、小车、机器人的实验。

20.2 RGB 彩灯（Neopixel）

- **前言：**

RGB 灯带是彩灯的一种，我们看到长长的灯带以及 LED 彩色显示屏，都是采用一个个灯珠组合而成。在本章节我们将通过编程实现 RGB 灯带的控制，可以应用到家居布置、节日气氛的场景中去。

- **实验平台：**

PyBLE-NRF52840 开发套件（可选配 RGB 灯带，连接 pyBase 到“Y11”接口。）

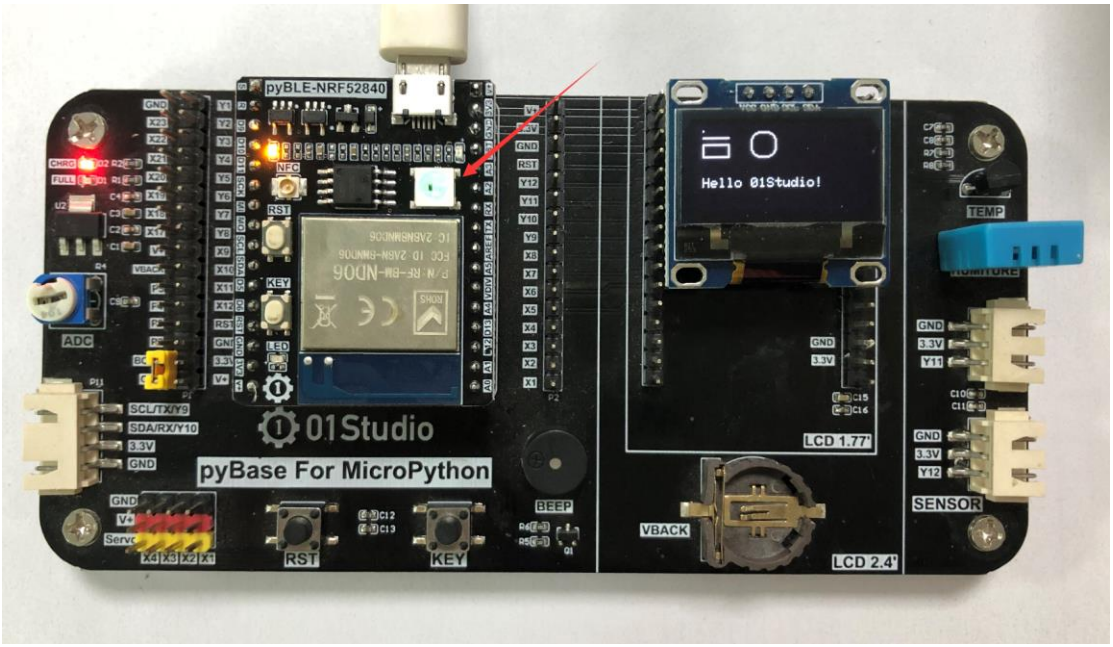


图 20-8 RGB 灯带接线方式

- **实验目的：**

通过编程实现 RGB 彩灯循环显示红色（RED）、绿色（GREEN）和蓝色（BLUE）。

- **实验讲解：**

pyBLE-NRF52840 上一个 NEOPIXEL 灯珠，可以直接编程实现相关功能，非常方便，而外接灯带的话只需要修改 IO 口和灯珠数量即可，我们先来先来介绍一下本实验用到的 RGB 灯带。

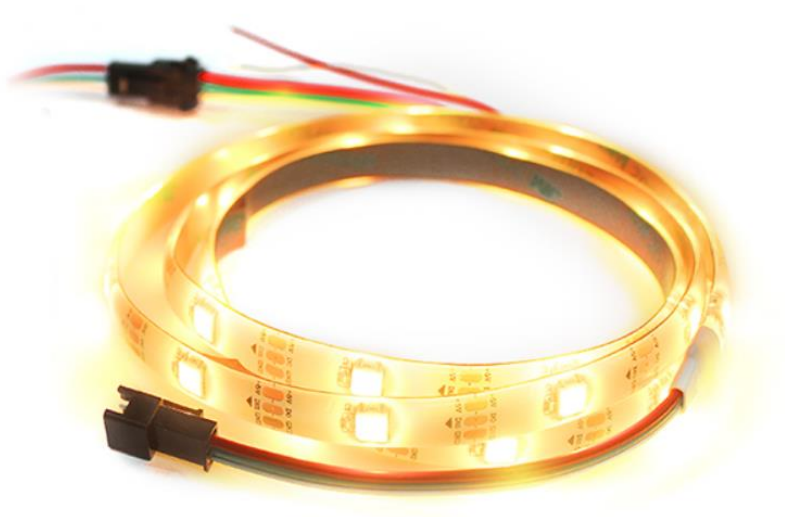


图 20-9 RGB 灯带

该灯带为 3 线接口，长 1 米，有 30 颗灯珠，每个灯珠里面都有一个驱动 IC，型号是 WS2812B，单总线驱动。每个灯珠首尾相连。灯带末端预留了接口可以串联更多的灯带。也就是说只需要通过 1 个 GPIO 口就可以控制了数十上百的灯珠了。

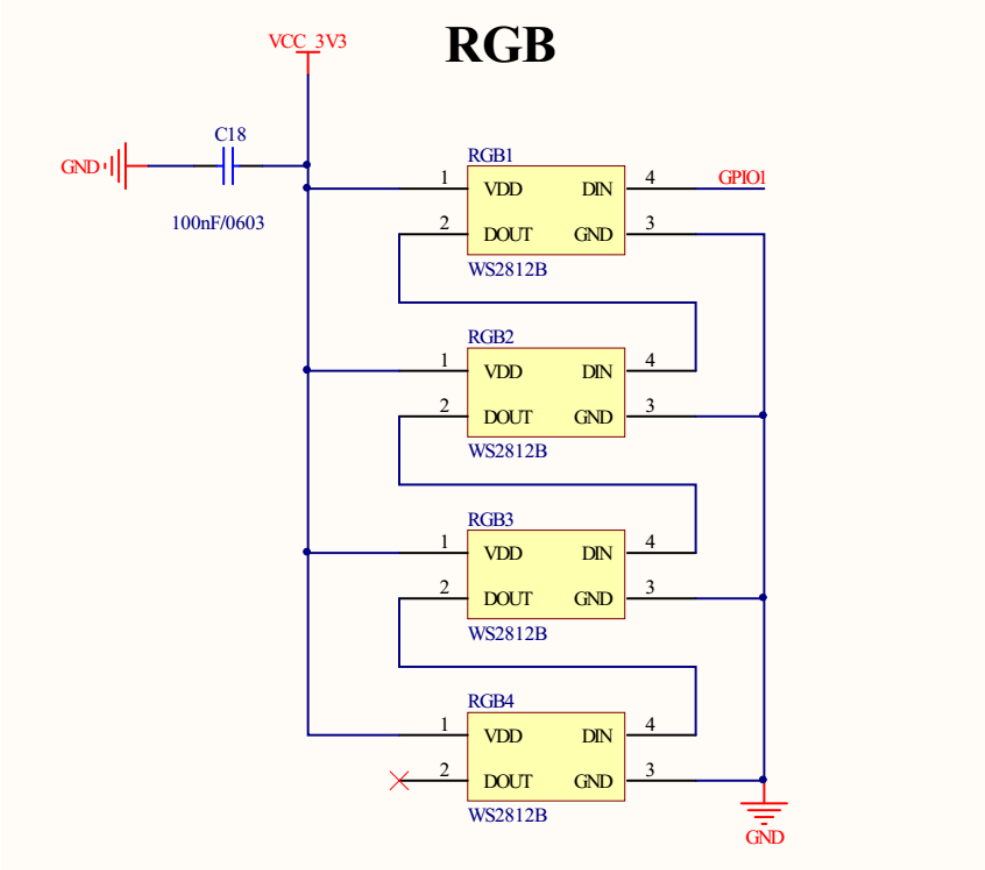


图 20-10 灯珠串联接线方式

了解了接线方式后，我们知道颜色是由最基本的三种颜色的不同亮度混合出新颜色。这 3 个最基本的颜色顺序分别是红，绿，蓝（RGB）。这里每个颜色的亮度级别从 0-255,0 表示没有，255 表示最亮。如（255,0,0）则表示红色最亮。GPIO 口就是将这些数据逐一发送给灯带。

彩灯驱动模块是 NeoPixel，适用于 WS2812B 驱动的灯珠。因此我们可以直接使用。说明如下：

构造函数
`pixels = neopixel.NeoPixel(pin, n, brightness=1.0)`
构建 Neopixel 对象。 【pin】灯带控制引脚；板载灯珠引脚为 board.NEOPIXEL。 【n】灯珠数量； 【brightness】亮度，默认 1.0。
使用方法
`pixels[i]=(R,G,B)`
设置第 i 个灯珠参数。
pixels.fill((R,G,B))
全部灯珠填充。
pixels.show()
显示刷新。

表 20-1 灯带对象

编程流程图如下：

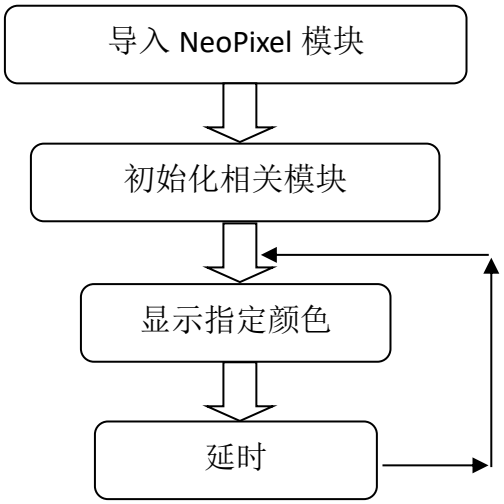


图 20-11 编程流程图

main.py 参考程序代码如下：

```
'''
实验名称：RGB 彩灯控制（NeoPixel）
版本：v1.0
日期：2020.5
作者：01Studio 【www.01Studio.org】
说明：通过编程实现灯带不同颜色的变化。
'''

import time
import board
import neopixel

#pyBLE-NRF52840 板载 NEOPIXEL(1 个灯珠)
pixels = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.2,
                                    auto_write=False)
```

```
#外接 01Studio RGB 灯带(30 个灯珠)
#pixels = neopixel.NeoPixel(board.A2, 30, brightness=0.2,
                                    auto_write=False)

while True:

    #红色
    pixels.fill((255, 0, 0))
    pixels.show()
    time.sleep(1)

    #绿色
    pixels.fill((0, 255, 0))
    pixels.show()
    time.sleep(1)

    #蓝色
    pixels.fill((0, 0, 255))
    pixels.show()
    time.sleep(1)
```

- **实验结果：**

将程序下载到开发套件，可以看到 pyBLE-NRF52840 板载灯珠的颜色在红、绿、蓝三色中循环变化。

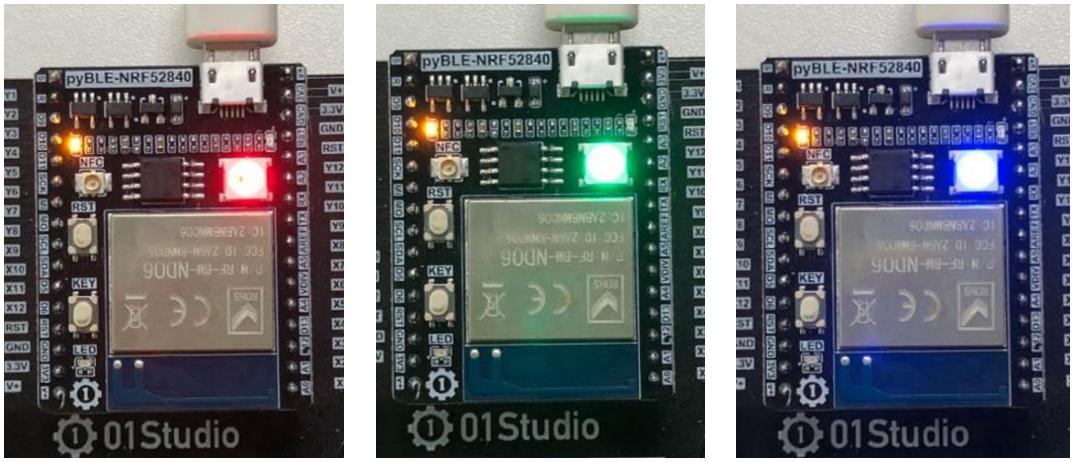


图 20-12 板载 Neopixel

将示例代码的构建对象改成外接 RGB 灯带，可以看到 RGB 灯带出现同样的实验结果。

```
#外接 01Studio RGB 灯带(30 个灯珠)
#pixels = neopixel.NeoPixel(board.A2, 30, brightness=0.2,
                                    auto_write=False)
```

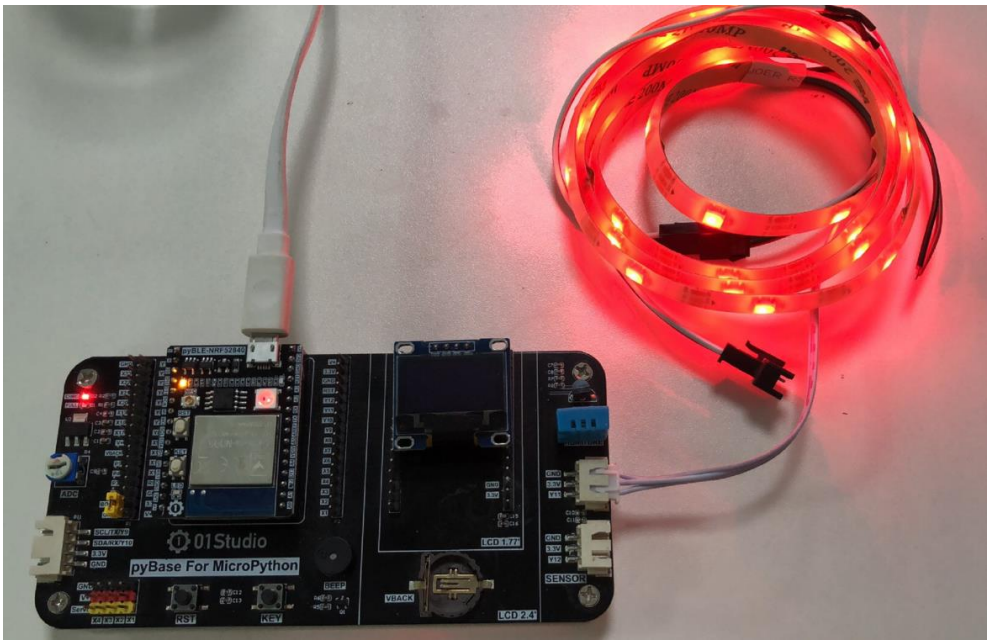


图 20-13 实验结果

- **总结：**

NEOPEXEL 的单 IO 口驱动的特点使得我们可以轻易的增加和减少灯带的数量而无需过多的修改程序。但需要注意的是如果灯带数量过多，那么需要外接电源供电，否则会因为电流不足而影响使用。

20.3 LCD

20.3.1 1.77 寸 LCD

- **前言：**

LCD 液晶显示屏是非常常见的一个外接显示设备，跟前面的 OLED 显示屏相比，LCD 会更常用一些，我们看到的手持设备、小型电器，很多都用到 LCD，部分配合触摸屏应用，能实现非常多的功能。

- **实验平台：**

pyBLE-NRF52840 开发套件、1.77 寸 LCD，驱动 ST7735S。

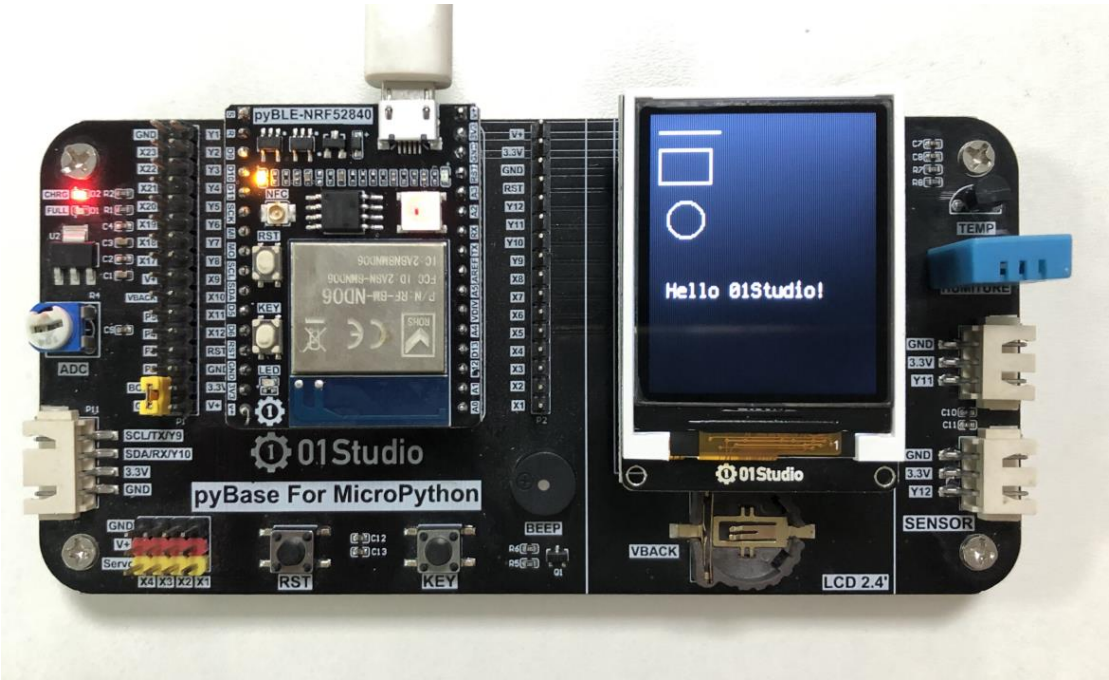


图 20-14 1.8 寸 LCD

- **实验目的：**

通过 MicorPython 编程方式实现 LCD 的各种显示功能，包括画点、线、矩形、圆形、显示英文、显示图片等。

- **实验讲解：**

我们先来看看实验所使用 LCD 的参数介绍：

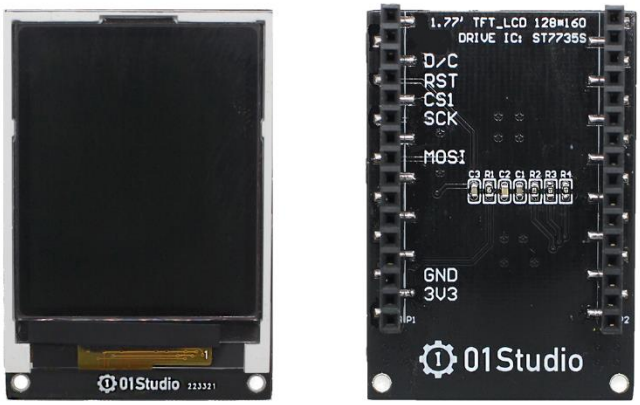


图 20-15 1.77 寸 LCD 显示屏

功能参数	
供电电压	3.3V
屏幕尺寸	1.77 寸
分辨率	128*160
颜色参数	TFT 彩色
驱动芯片	ST7735S
通讯方式	SPI 总线
接口定义	2.54mm 排母（兼容 pyboard v1.1 接口）
整体尺寸	5.0*3.5 cm

表 20-2

1.8 寸 LCD 跟 pyBLE-NRF52840 的引脚连线如下表。

pyboard	LCD	LCD 引脚说明
Y1	PEN	触摸屏中断
Y2	CS2	触摸屏片选
Y3	D/C	数据/命令选择

Y4	RST	复位
Y5	CS1	LCD 片选
Y6	SCK	SPI 接口
Y7	MISO	SPI 接口
Y8	MOSI	SPI 接口
3.3V	VCC	电源（3.3V 供电）
GND	GND	地

图 20-16 LCD 引脚连接和说明

实验用的 LCD 是 1.8 寸，驱动是常见的 ST7735S，使用 SPI 接口跟开发板通信，按以往嵌入式 C 语言开发，我们需要对 ST7735S 进行编程实现驱动，然后再建立各种划线、图案和字符函数。

使用 MicroPython 其实也需要做以上工作，但由于可读性和移植性强的特点，我们只需要搞清各个对象函数使如何使用即可。总的来说和之前一样，有构造函数和功能函数。构造函数解决的是初始化问题，告诉 pyBLE-NRF52840 外设是怎么接线，是什么样的；而功能函数解决的则是使用问题，我们基于自己的需求直接调用相关功能函数，实现自己的功能即可！

我们管这些函数的集合叫驱动，主要是基于 circuitpython 的 displayio 对象、shape 对象和字符 Label 对象，使用说明如下：

构造函数
`display_bus = displayio.FourWire(spi, command, chip_select, reset)`
构建 SPI 对象 【spi】SPI 总线； 【command】LCD 的 D/C 引脚; 【chip_select】LCD 的 CS 引脚； 【reset】LCD 的复位引脚。
`display = ST7735R(bus, width, height,  rotation=0, bgr=True)`
构建 ST7735R 显示屏对象。

【bus】总线对象; 【width】LCD 宽度; 【height】LCD 高度; 【rotation】旋转角度：0,90,180,270; 【bgr】附加初始化系列，01Studio 的 LCD 需要初始化为 True
splash = displayio.Group(max_size=20, scale=1)
display 会把每个显示的元素归到组； 【max_size】最大元素数量。 【scale】字体大小，默认 1
使用方法
display.show(splash)
显示组元素。
splash.append(group)
增加元素； 【group】所增加要显示的元素
更多用法，请看官方文档： https://circuitpython.readthedocs.io/en/5.0.x/shared-bindings/displayio/__init__.html

表 20-3 displayio 对象

构造函数
Line(x0,y0,x1,y1,outline)
画线。模块在 adafruit_display_shapes.line 下。 【x0,y0】起始坐标 【x1,y1】终点坐标 【outline】颜色，例：0Xffffff 白色
Rect(x,y,width,height,fill=None,outline=None)
画矩形。模块在 adafruit_display_shapes.rect 下。 【x,y】起始坐标

【width】宽度 【height】高度 【fill】填充颜色 【outline】外框颜色
`Circle(x0,y0,r,fill=None,outline=None)`
画圆。 【x0,y0】圆心 【r】半径 【fill】填充颜色 【outline】外框颜色

表 20-4 shape 对象

构造函数
Label(font, x=0,y=0,text=None, color=0xFFFFFF, background_color=None)
标签字符对象。模块 adafruit_display_text.label 下。 【font】字体； 【x,y】坐标； 【text】显示字符； 【color】字体颜色； 【background_color】背景颜色。

表 20-5 Label 对象

上面的对象构造函数和使用说明，由于 Circuitpython 务求在不同开发板的操作方式一致，因此有些库使用起来有点绕，但好处是一旦熟悉后，代码一致性较强，结合上方对象讲解我们来看看编程思路：

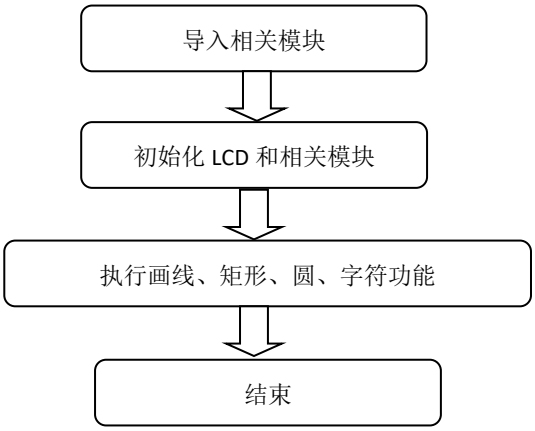


图 20-17 编程流程图

参考 main.py 函数代码如下：

```
'''
实验名称：1.8 寸 LCD 显示（ST7735S）
版本：v1.0
日期：2020.4
作者：01Studio
说明：LCD 显示实验。
'''

import board,time
import displayio
import terminalio
from adafruit_st7735r import ST7735R
from adafruit_display_shapes.line import Line
from adafruit_display_shapes.rect import Rect
from adafruit_display_shapes.circle import Circle
from adafruit_display_text import label

#释放显示总线
```

```
displayio.release_displays()

#定义 LCD 接口，SPI
spi = board.SPI()
tft_cs = board.D11
tft_dc = board.D9
tft_rst = board.D10

#构建 spi 显示总线
display_bus = displayio.FourWire(spi, command=tft_dc,
chip_select=tft_cs, reset=tft_rst)

#构建显示屏对象,默认自动刷新
display = ST7735R(display_bus, width=128, height=160,  rotation=0,
bgr=True)

#设置显示元素个数，最大 20 个
splash = displayio.Group(max_size=20,scale=1)
display.show(splash)

########################################
# 添加元素，添加后默认自动显示
########################################

#画线
splash.append(Line(5, 10, 40, 10, 0xFFFFFF))

#画矩形
rect = Rect(5, 20, 30, 20, outline=0xFFFFFF,fill=0x0)
splash.append(rect)
```

```
#画圆
circle = Circle(20, 60, 10, fill=0x0, outline=0xFFFFFF)
splash.append(circle)

# 字符显示,使用 Label 对象
text_area = label.Label(terminalio.FONT, text="Hello 01Studio!",
color=0xFFFFFF,x=10, y=100)
splash.append(text_area)

while True:
    pass
```

- **实验结果：**

将示例程序所有库和函数文件拷贝到开发板 U 盘。可以看到 LCD 执行相关显示功。

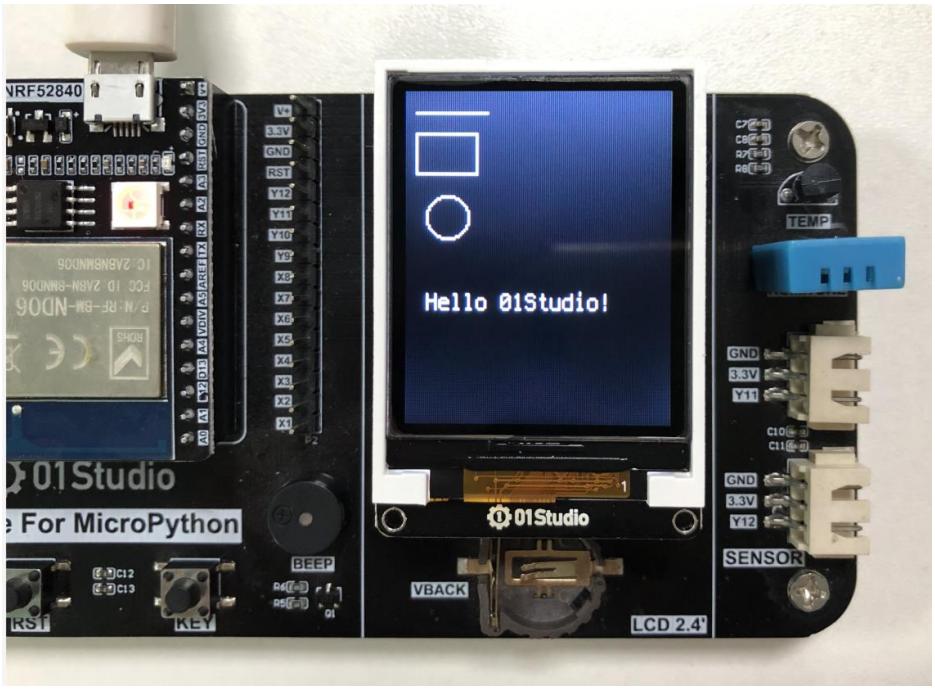
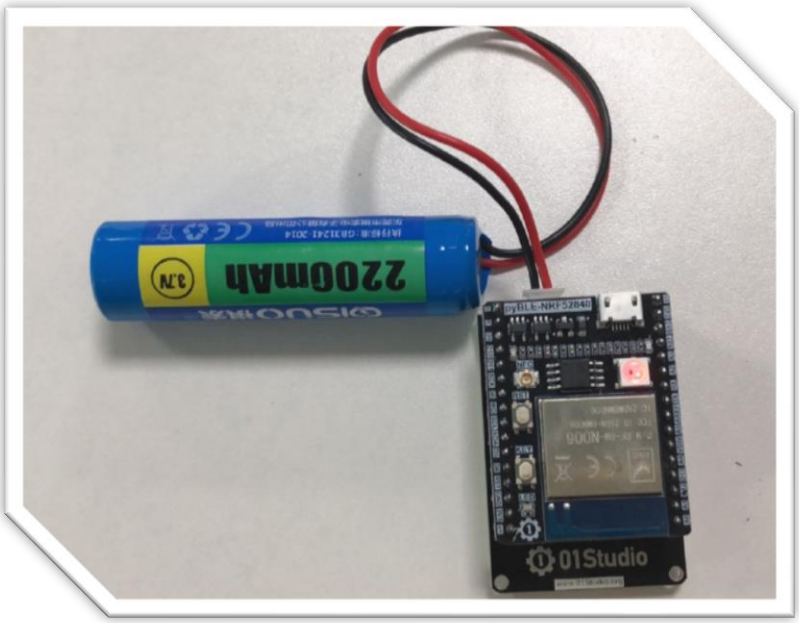


图 20-18

由于我们 LCD 接口兼容 pyboard，因此也可以直接将 pyBLE-NRF52840 插到 LCD 背面，使用锂电池供电。如下图所示：



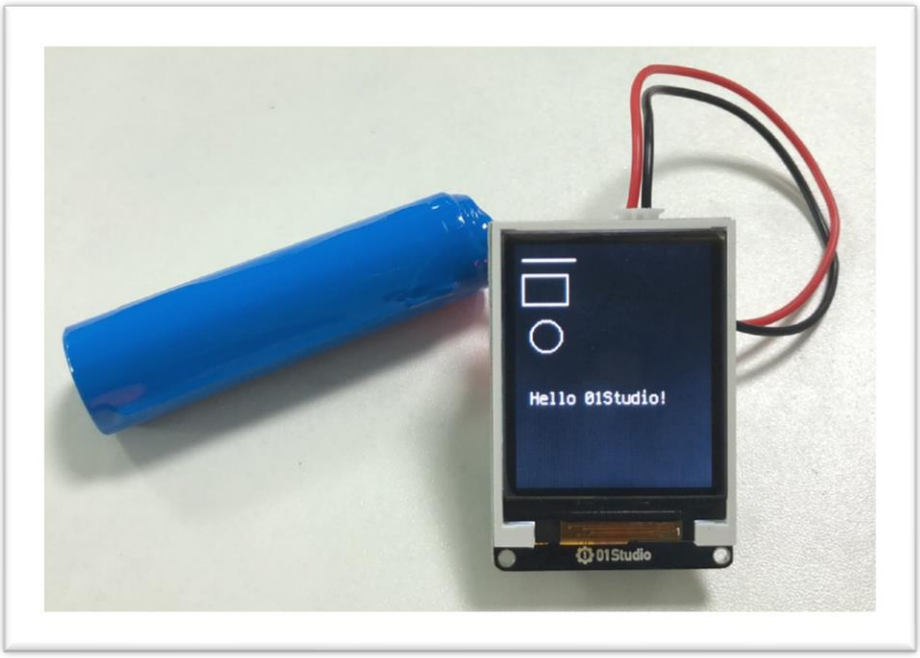


图 20-19　LCD 直接连接 pyboard v1.1-CN

- **总结：**

本实验用到的 LCD 和其驱动芯片 ST7735S 是常见的产品，可以看到 MicroPython 编程只关注屏幕的驱动和分辨率，对于不用尺寸和型号的 LCD 来说，移植性非常强，有兴趣的朋友可以自行驱动其它系列的 LCD。

第21章　BLE 应用

我们先来看看 BLE 的简介，以下内容来源：百度百科。

- **概念：**

低功耗蓝牙（Bluetooth Low Energy），是蓝牙技术联盟设计和销售的一种个人局域网技术，旨在用于医疗保健、运动健身、信标、安防、家庭娱乐等领域的新兴应用。相较经典蓝牙，低功耗蓝牙旨在保持同等通信范围的同时显著降低功耗和成本。

- **兼容性：**

BLE 和经典蓝牙协议不兼容。

- **目标市场：**

蓝牙技术联盟的低功耗技术瞄准多个市场，特别是智能家庭、健康、运动健身部分。主要的优点包括：

1. 低功耗，使用纽扣电池就可运行数月至数年；
2. 小体积、低成本；
3. 与现有的大部分手机、平板电脑和计算机兼容。

看了上面相关概念，你可能还是会感觉模糊，，简单来说低功耗蓝牙技术就是专门为低功耗设备而设的，与经典蓝牙（蓝牙音箱、蓝牙耳机）协议有很大区别。例如我们日常生活中的手环就是用到 BLE 技术，BLE 甚至往 Mesh 组网的方向发展。

通过接下来内容的学习，相信你会对 BLE 技术有更全面的认识。

21.1 蓝牙广播

- **前言：**

BLE 是物联网中非常重要的角色，相对于 WiFi 连接而言，BLE 可以做到超低功耗，快速连接，通过连接智能手机从而接入到互联网，成了很多智能产品普遍的选择，手环就是一个应用。

跟传统蓝牙音箱一样，要想连接就得先让手机搜索到 BLE 设备的发出的信号（广播），本节我们就来学习一下 BLE 蓝牙是如何广播的。

- **实验平台：**

pyBLE-NRF52840 开发套件。

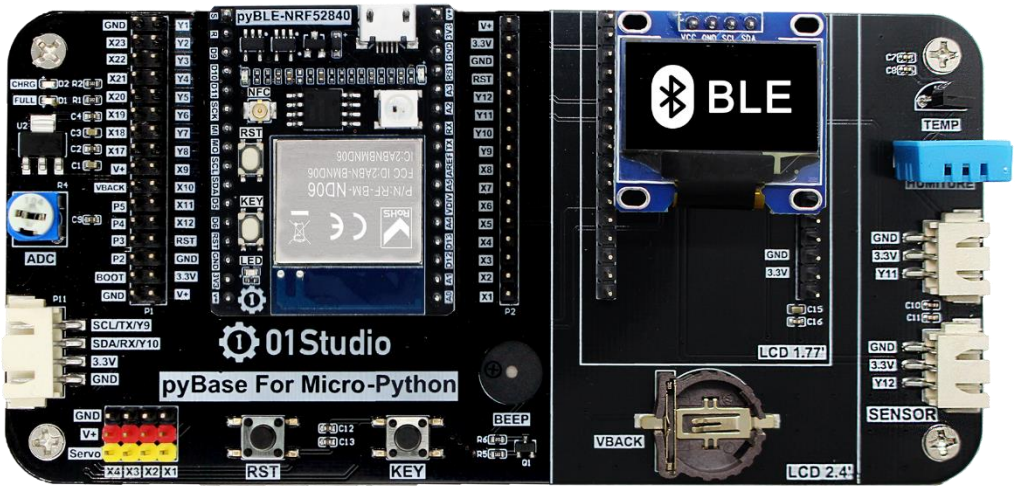


图 21-1 pyWiFi-ESP32 开发套件

- **实验目的：**

编程实现连接路由器，将 IP 地址等相关信息通过 OLED 显示（只支持 2.4G 网络）。

- **实验讲解：**

通常情况下，蓝牙设备（蓝牙从机）只有发出广播信号，才能让手机（蓝牙主机）搜索到从而发起连接。

MicroPython 已经集成了 BLE 开发模块，开发者使用内置的 BLE 模块函数可

以非常方便地连接上对蓝牙开发板做各种蓝牙无线操作。

我们先来看看 Micropython 的 BLE 模块（adafruit_ble）的构造函数和使用方法。

构造函数
ble = adafruit_ble.BLERadio()
构建 ble 对象。
使用方法
`ble.name = 'name'`
广播名称。 【name】广播名字。
`ble.start_advertising(advertisement)`
开始广播。 【advertisement】广播类型，可以携带蓝牙服务
更详细说明，请看官方文档： https://circuitpython.readthedocs.io/projects/ble/en/latest/api.html

表 21-1　BLE 对象

从上表可以看到 MicroPython 通过模块封装，让 BLE 广播变得非常简单。代码编写流程如下：

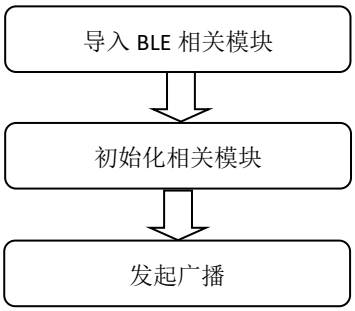


图 21-2 代码编写流程

实验参考代码如下：

```
'''
实验名称：蓝牙广播
版本：v1.0
日期：2020.5
作者：01Studio
说明：编程实现无线广播。
'''

import time
from adafruit_ble import BLERadio
from adafruit_ble.advertising.standard import
ProvideServicesAdvertisement

#构建 BLE 对象
ble = BLERadio()

#定义广播名称
ble.name='01Studio'

#服务类型，本例程暂时不提供蓝牙服务
advertisement = ProvideServicesAdvertisement()

#开始广播
ble.start_advertising(advertisement)

#保持程序运行
while True:
    pass
```

- **实验结果：**

BLE 和经典蓝牙不一样，是不能直接在手机设备里面进行连接的，我们需要下载一个 APP 来扫描广播的蓝牙设备和以及进行连接。这里可以下载比较常用的 LightBlue 软件，可以在 APP Store 搜索 LightBlue 下载安装：



图 21-3 蓝牙 APP 助手 LightBlue

下载程序，打开安装好的 LightBlue APP，可以看到 APP 搜索到广播名称为 01Studio 的设备，点击进去--showdata 可以看到设备相关信息。

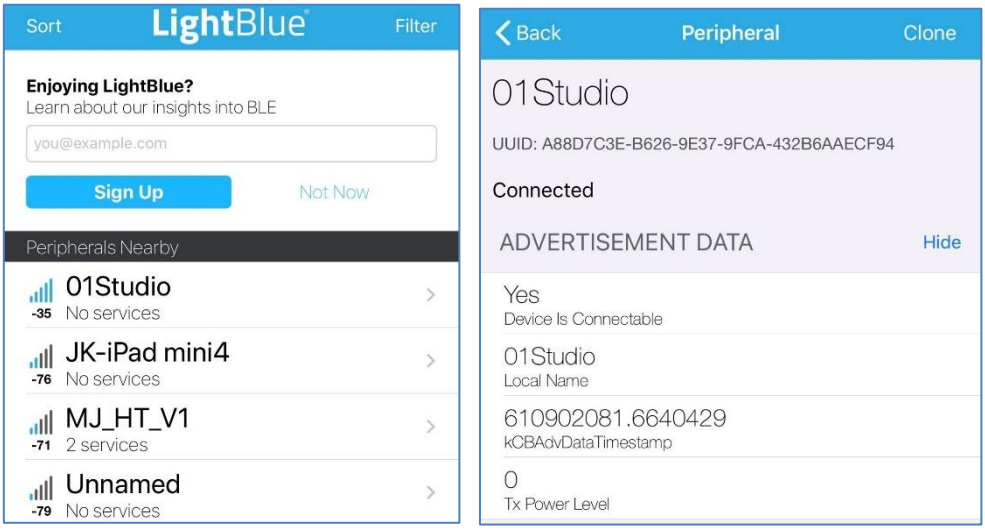


图 21-4

可以看到广播名称 01Studio 下发有一个值，称为 RSSI（信号强度），范围是 0 至-99，越接近 0 表示信号质量好，反之信号质量差。

将手机靠近开发板蓝牙模块（天线位置），RSSI 绝对值变小。表示信号变好。

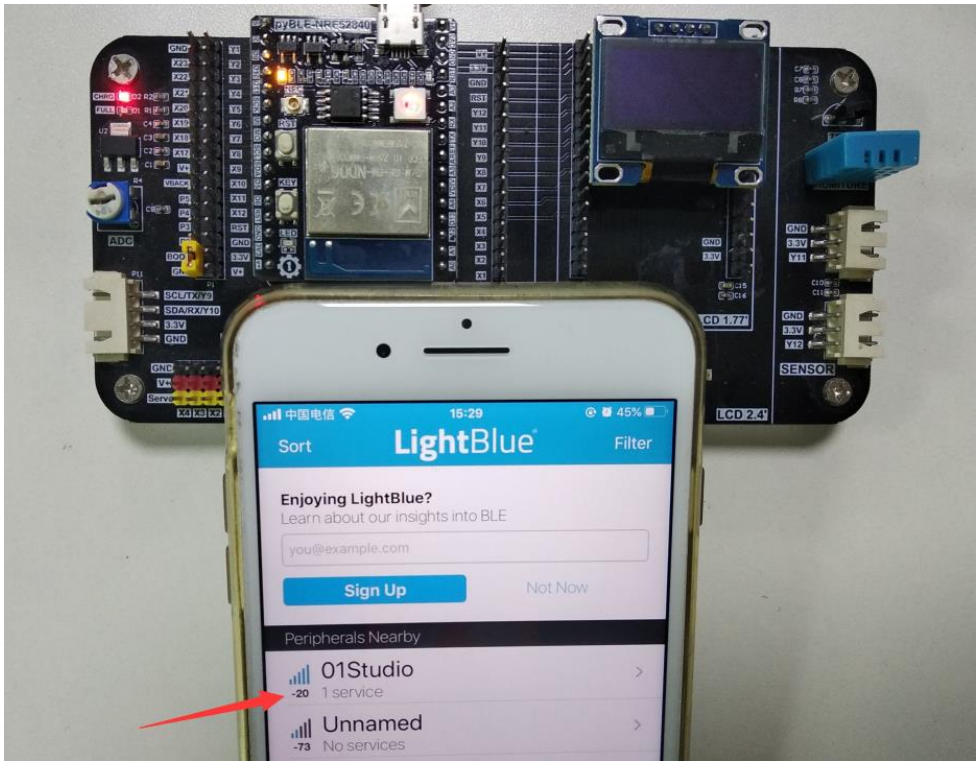


图 21-5 RSSI 变强

- **总结：**

本节是广播是 BLE 应用的基础，手机成功连接到蓝牙开发板好，后面就可以做各类服务通讯实验了。

21.2 设备信息

- **前言：**

上一节我们学习了最基础的广播，然而可以看到广播的信息量是非常少的，通常情况下我们往往会想了解更多蓝牙设备的信息，如厂商信息、软件版本、硬件版本等。本节就来学习一下如果为自己的 BLE 开发板添加硬件设备信息。

- **实验平台：**

pyBLE-NRF52840。

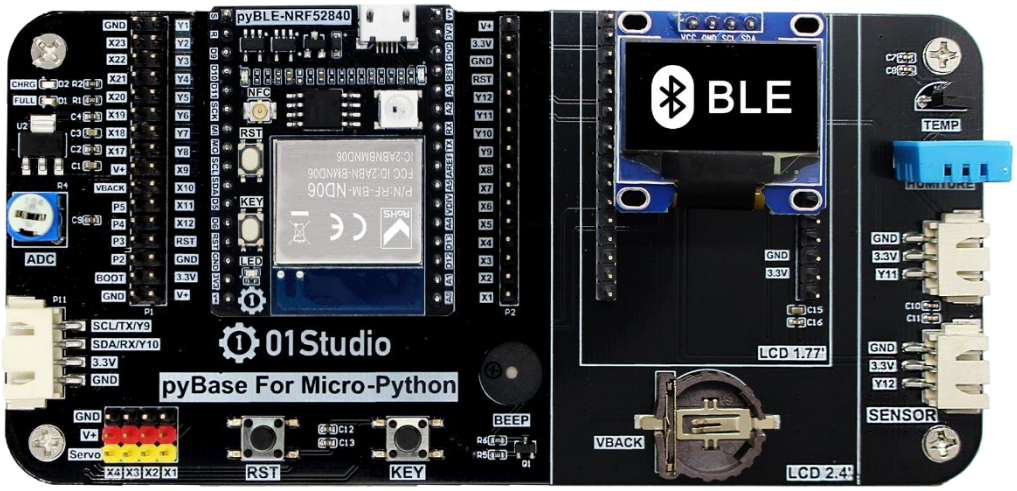


图 21-6 pyWiFi-ESP32 开发套件

- **实验目的：**

为蓝牙开发板添加设备信息。

- **实验讲解：**

设备信息可以理解成是蓝牙的一个服务，关于服务我们在下一节会详细展开讲解，本章先通过代码体验一下，可以简单理解成定义的一组信息，当手机连接上蓝牙设备后后可以获取相关信息的过程。我们来看看设备信息对象的定义和用法。

构造函数
dev_info = DeviceInfoService(manufacturer=None, software_revision=None, model_number=None, serial_number=None, firmware_revision=None, hardware_revision=None, service=None)
标准设备信息服务构建。 设备参数信息均可自行填写，如果没定义则使用固件默认的信息。
使用方法
advertisement = ProvideServicesAdvertisement(dev_info)
增加设备信息服务。
ble.start_advertising(advertisement)
开始广播。 【advertisement】广播类型，可以携带蓝牙服务

表 21-2 标准设备信息用法

从上面标准设备信息用法来看，也就是在原来广播基础上加入了一组信息，代码编写流程如下：

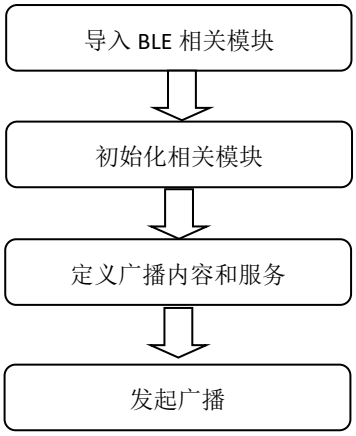


图 21-7 代码编写流程图

参考代码如下：

```
'''
实验名称：设备信息
版本：v1.0
日期：2020.5
作者：01Studio
说明：编程实现无线广播。
'''

import time
from adafruit_ble import BLERadio
from adafruit_ble.advertising.standard import
ProvideServicesAdvertisement
from adafruit_ble.services.standard.device_info import
DeviceInfoService

#构建 BLE 对象
ble = BLERadio()

#定义广播名称
ble.name = '01Studio'

#定义设备信息
dev_info = DeviceInfoService(manufacturer='01Studio',
                             software_revision='v1.0',
                             model_number='pyBLE-NRF52840',
                             serial_number=None,
                             firmware_revision=None,
                             hardware_revision='v1.0',
```

```
                            service=None)


#广播添加设备信息服务
advertisement = ProvideServicesAdvertisement(dev_info)


#发起广播
ble.start_advertising(advertisement)


while True:
    pass
```

- **实验结果：**

跟上一节一样，我们用 lightblue 软件连接 pyBLE-NRF52840，可以看到出来广播信息外还多了设备信息。

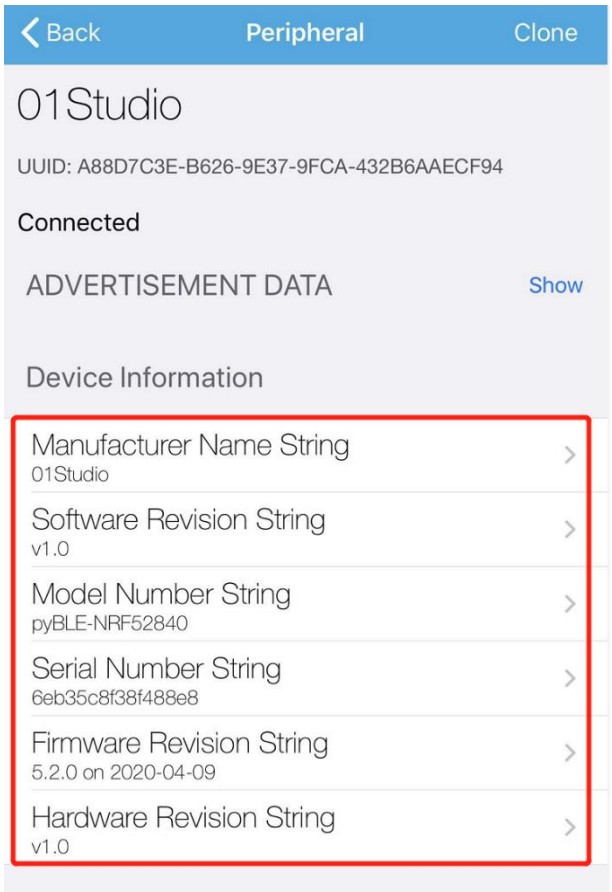


图 21-8 显示自定义的广播信息

- **总结：**

这一节是在蓝牙广播基础上添加了标准设备信息服务，让 app 可以观察到设备信息。这让蓝牙设备变得更加专业。而强大的 python BLE 模块，让我们轻松实现了这个功能。

21.3 无线数据收发

- **前言：**

BLE 数据收发（UART 服务），可能是最通用的功能和服务了，基本上所有的 BLE 数据传输设备都使用 UART 服务。这里的 UART 并非指硬件串口，而是 BLE 的一个服务，有了它，手机 APP 和蓝牙设备就能建立数据传输，事不宜迟，我们马上来开始学习。

- **实验平台：**

pyBLE-NRF52840 开发板。

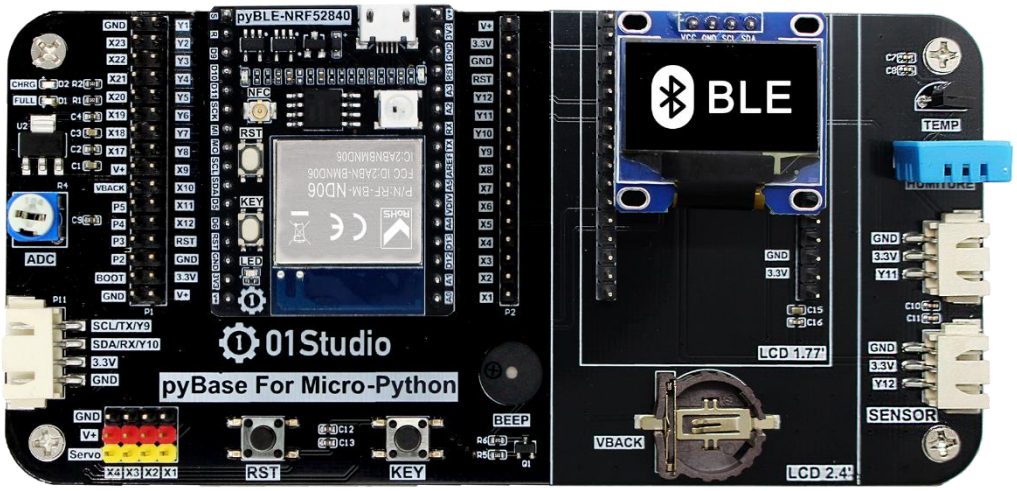


图 21-9 pyWiFi-ESP32 开发套件

- **实验目的：**

通过 python 编程实现手机与蓝牙开发板数据收发。

- **实验讲解：**

本实验默认以 pyBLE-NRF52840 为从机，手机为主机。其实上一节我们就已经接触到服务的概念了，蓝牙设备和手机通讯主要依赖“服务”，简单来说服务就是一个指定编号的黑盒子。

黑盒子的故事：

蓝牙设备和手机约定了一个黑盒子，名称为“AAA”，在双方建立连接后，这个黑盒子只允许手机写入数据，蓝牙设备从里面读取数据，当手机往黑盒子写入数据 123，蓝牙设备从黑盒子读取，这便完成了一个从手机发送数据到蓝牙设备的过程！

反过来再约定一个黑盒子，名称为“BBB”，这个黑盒子规定则和前面“AAA”相反，允许蓝牙设备写入数据，手机读取数据。这样便完成了一个从蓝牙设备发送数据到手机的过程。

就这样，各式各样的黑盒子构成了整个**蓝牙服务**，后来人们又考虑到为了让不同厂家之间的蓝牙设备尽量兼容，于是便制定了一系列固定编号的黑盒子。并约定了很多细节的东西，这便是**蓝牙标准协议**。

本节使用的 UART 服务，便是 Nordic 的 UART 协议。我们以此为例说明一下手机和 BLE 设备通讯真实过程，先看下表，Nordic 先定义了串口服务的服务 UUID（Service UUID），长度为 128bit（有 16bit 和 128bit 两个标准）：6E400001-B5A3-F393-E0A9-E50E24DCCA9E；在这个服务 UUID 下有 2 个特征值，分别是 RX 和 TX，这些特征值也是通过 128bit UUID 来识别。每个特征值下有权限和 value，权限决定写和读（通知），value 则是要写和读的数据。

Nordic UART (NUS) GATT Service (串口服务)			
Service UUID (服务UUID)	6E400001-B5A3-F393-E0A9-E50E24DCCA9E		
	Characteristics (特征UUID)	RX	6E400002-B5A3-F393-E0A9-E50E24DCCA9E
		Permission (权限):	Write or Write Without Response （写）
		Value (值):	最大512字节
	Characteristics (特征UUID)	TX	6E400003-B5A3-F393-E0A9-E50E24DCCA9E
		Permission (权限):	Notify（通知）
		Value (值):	最大512字节

表 21-3 Nordic UART Service

在串口服务中，手机和 BLE 就是通过上面协议来实现数据传输，如下图所示：

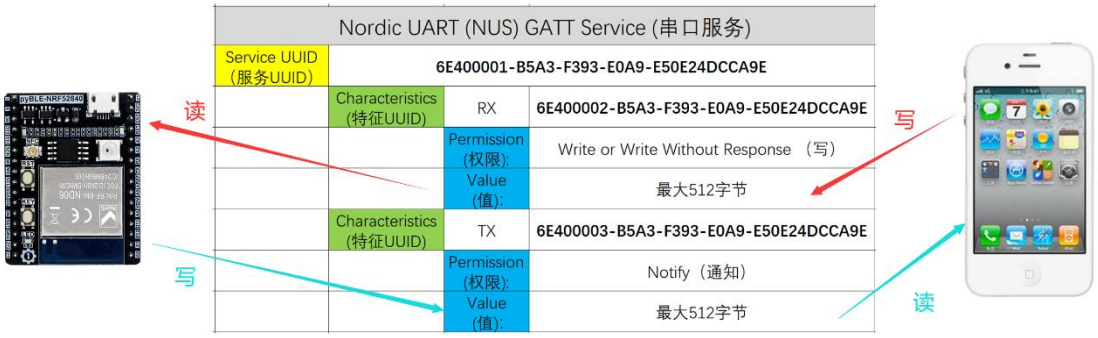


图 21-10 通讯 UART 服务读写数据

讨论到这，我们都是基于应用层，大部分蓝牙 IC 原厂和提供蓝牙开发协议栈，做好了底层包括射频传输等功能，开发的开发变得简单。而 MicroPython 的出现，可以说是如虎添翼，学会原理后直接面向对象编程，几行代码即可实现蓝牙数据通讯。我们先来看看相关对象：

构造函数
Uart_Service = UARTService()
构建 UART 服务。在模块 adafruit_ble.services.nordic 下。
使用方法
advertisement = ProvideServicesAdvertisement(Uart_Service)
广播添加 UART 服务。
ble.start_advertising(advertisement)
开始广播。 【advertisement】广播类型，可以携带蓝牙服务
Uart_Service.read(num_bytes)
串口服务读数据; 【num_bytes】读取字节个数，如果没数据，则返回 b''。
Uart_Service.write(buf)
写数据 【buf】要写的数据，类型需要是字节数组。

表 21-4 UART 服务对象和使用方法

从上表看到，nordic 的 UART 服务直接被打包好了，我们连 UUID 相关信息都无需关心，直接应用即可。我们也可以打开本节示例程序 lib\adafruit_ble\services 的 nordic.py ，找到 UART 服务初始化函数，可以看到定义的 UUID 和官方标准的一致：

main.py nordic.py

```
47
48      See ``examples/ble_uart_echo_test.py`` for a usage example.
49      """
50
51      # pylint: disable=no-member
52      uuid = VendorUUID("6E400001-B5A3-F393-E0A9-E50E24DCCA9E")
53      _server_tx = StreamOut(
54          uuid=VendorUUID("6E400003-B5A3-F393-E0A9-E50E24DCCA9E"),
55          timeout=1.0,
56          buffer_size=64,
57      )
58      _server_rx = StreamIn(
59          uuid=VendorUUID("6E400002-B5A3-F393-E0A9-E50E24DCCA9E"),
60          timeout=1.0,
61          buffer_size=64,
62      )
```

图 21-11 nordic.py 文件

代码编写流程如下：

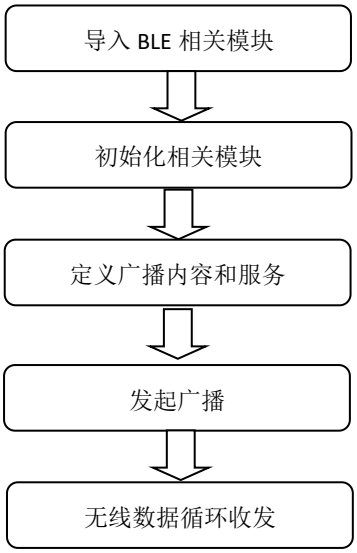


图 21-12 代码编写流程图

参考代码如下：

```
'''
实验名称：无线数据收发（UART 服务）
版本：v1.0
日期：2020.5
作者：01Studio
说明：编程实现手机和蓝牙开发板数据传输。
'''

#导入相关模块
import time
from adafruit_ble import BLERadio
from adafruit_ble.advertising.standard import
ProvideServicesAdvertisement
from adafruit_ble.services.nordic import UARTService

#构建蓝牙对象
ble = BLERadio()

#定义广播名称
ble.name='01Studio'

#构建 UART 服务
Uart_Service = UARTService()

#广播添加 UART 服务
advertisement = ProvideServicesAdvertisement(Uart_Service)

while True:

    #发起广播
```

```
ble.start_advertising(advertisement)


#等待连接
while not ble.connected:
    pass


#连接蔡成功
while ble.connected:


    # 读取 128 个字节数据，如果没数据，则返回 b''
    one_byte = Uart_Service.read(128)


    #收到信息，REPL 打印并回发
    if one_byte:
        print(one_byte)
        Uart_Service.write(one_byte)
```

- **实验结果：**

运行程序，打开 MU 的串口。打开手机 light blue，连接 01Studio，可以看到出现了 UART 服务和特征 UUID。

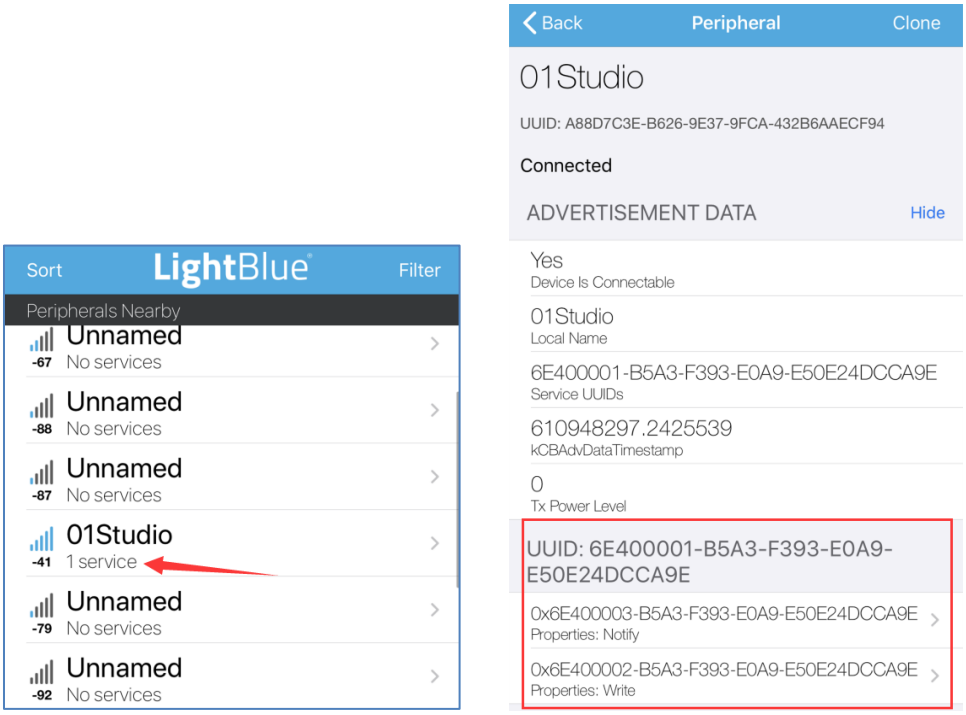


图 21-13

接下来我们测试一下手机往蓝牙开发板写数据，点击写特征：

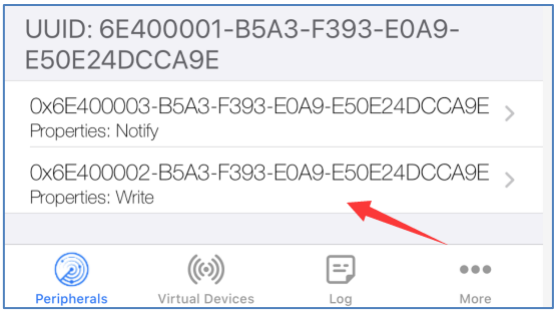


图 21-14

进入后可以在右上角修改发送数据类型，这里选择字符：

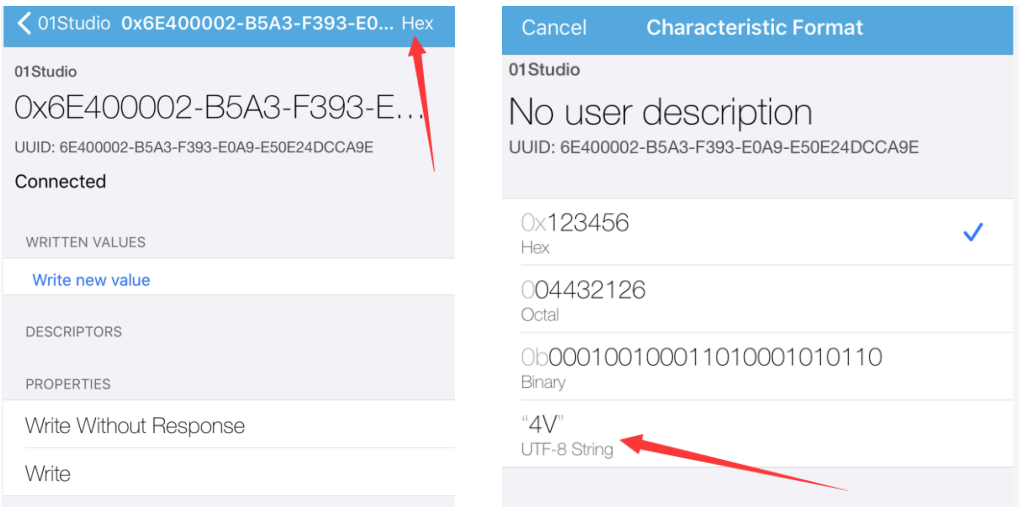


图 21-15 选择字符类型

输入“hello 01studio”，按 Done，数据便发出：

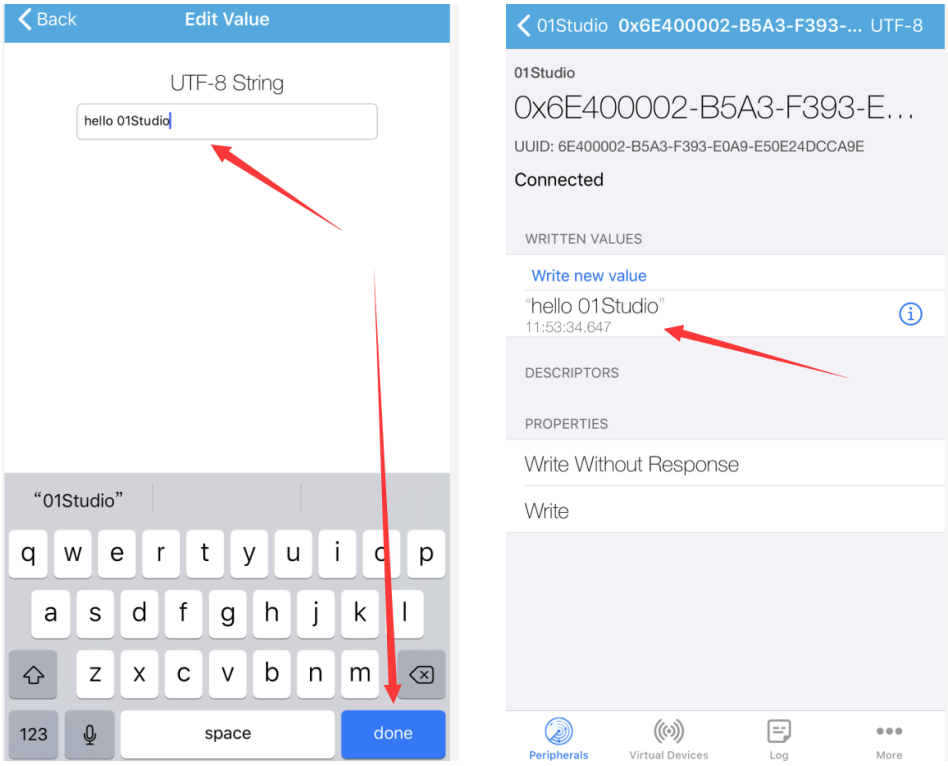


图 21-16

这时候可以看到连接开发板的 PC 端，REPL 收到了手机发来的信息：

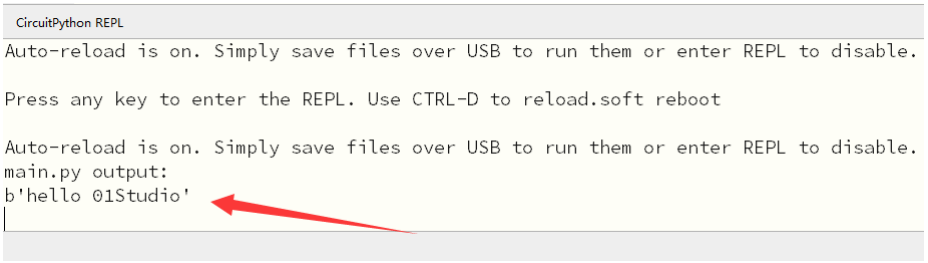


图 21-17

细心的用户或许发现，开发板程序是将收到的数据发送回给手机的，怎么查看呢，light blue 是比较通用的 APP，我们可以使用其它 APP 来测试了。

nrf toolbox：

nrf toolbox，一款 nordic 开发专门用于 nordic 服务的 APP。



图 21-18 下载 nRF Toolbox

进入 APP 后点击串口服务，然后点击 connect 连接 pyBLE-NRF52840 开发板：

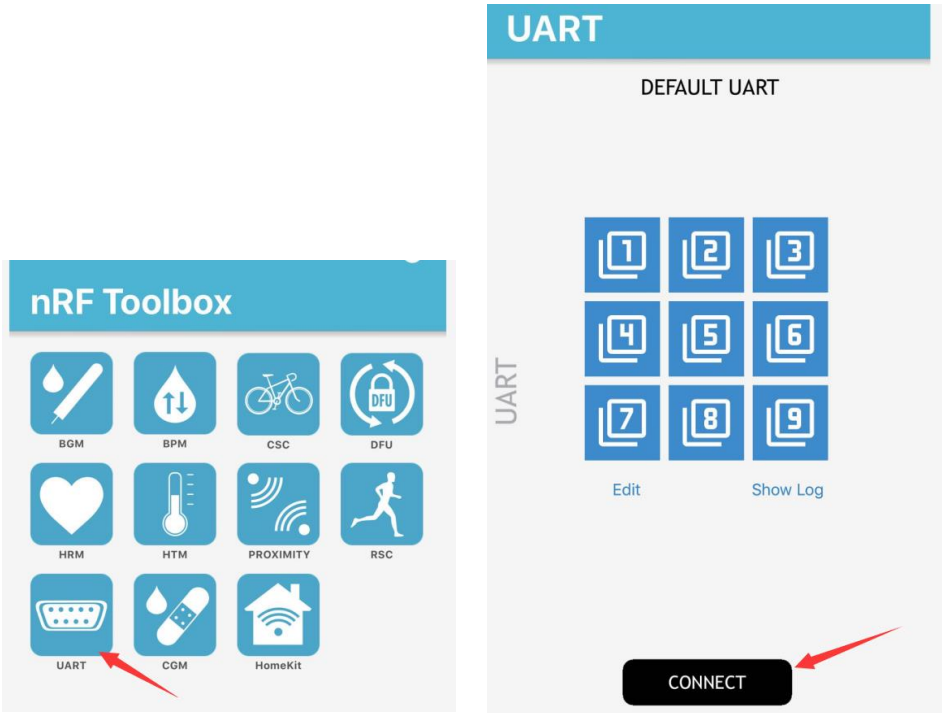


图 21-19

可以通过点击 edit，编辑 9 个色块任意一个，配置每个色块的外观和发送数据，比如 9。

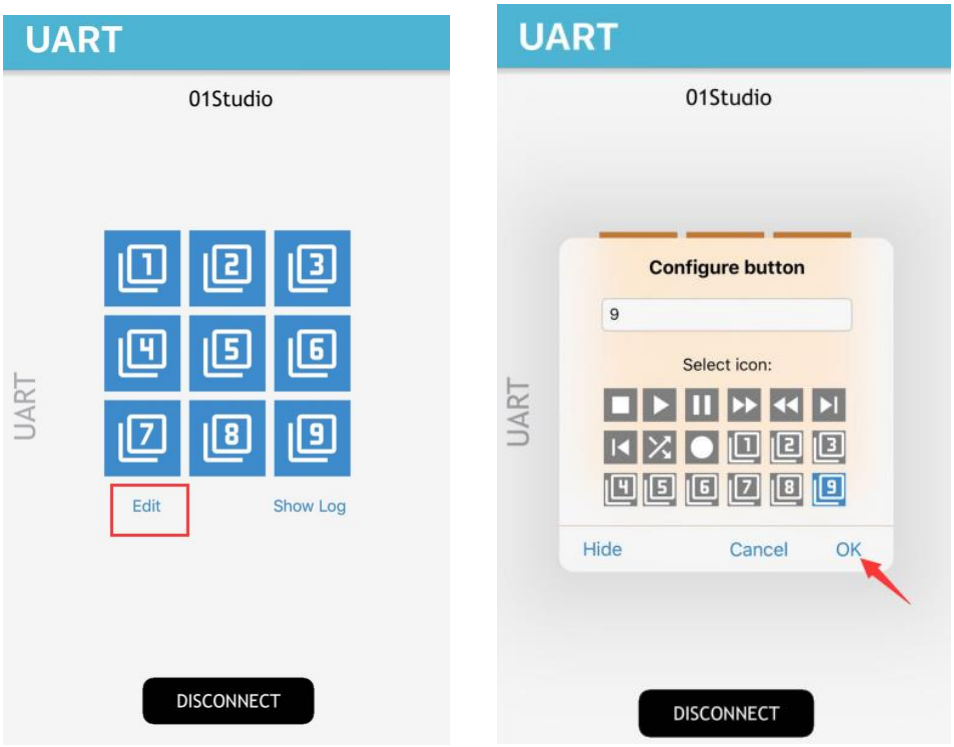


图 21-20

配置好后点击方块 9，便可以发送数据 9 给蓝牙开发板。

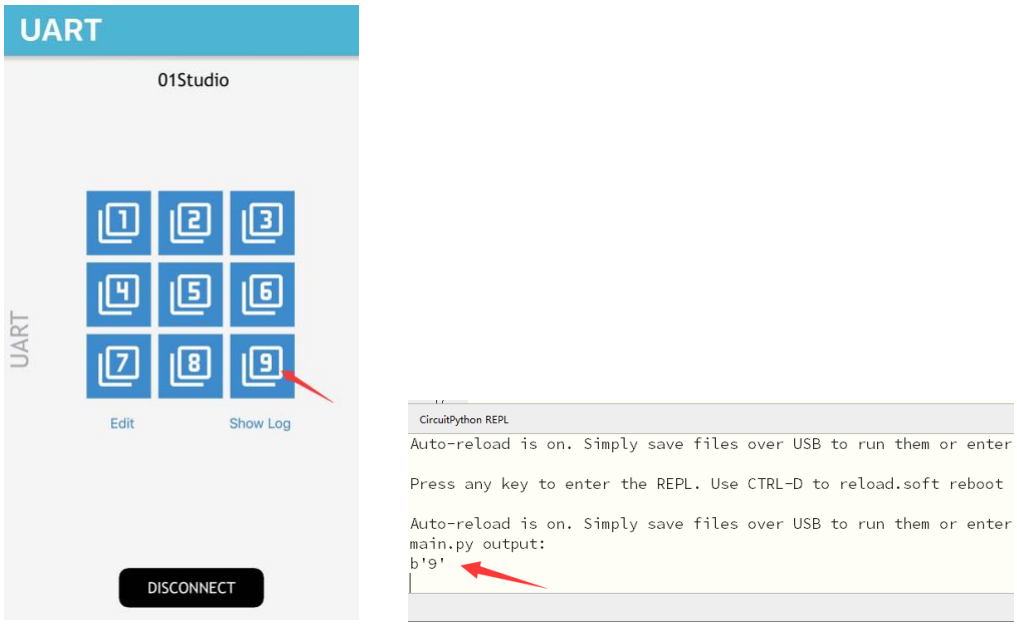


图 21-21 手机成功发送数据给开发板

点击 showlog，可以看到刚刚发送字符给开发板结果，同时也收到了开发板发回来的字符 9。

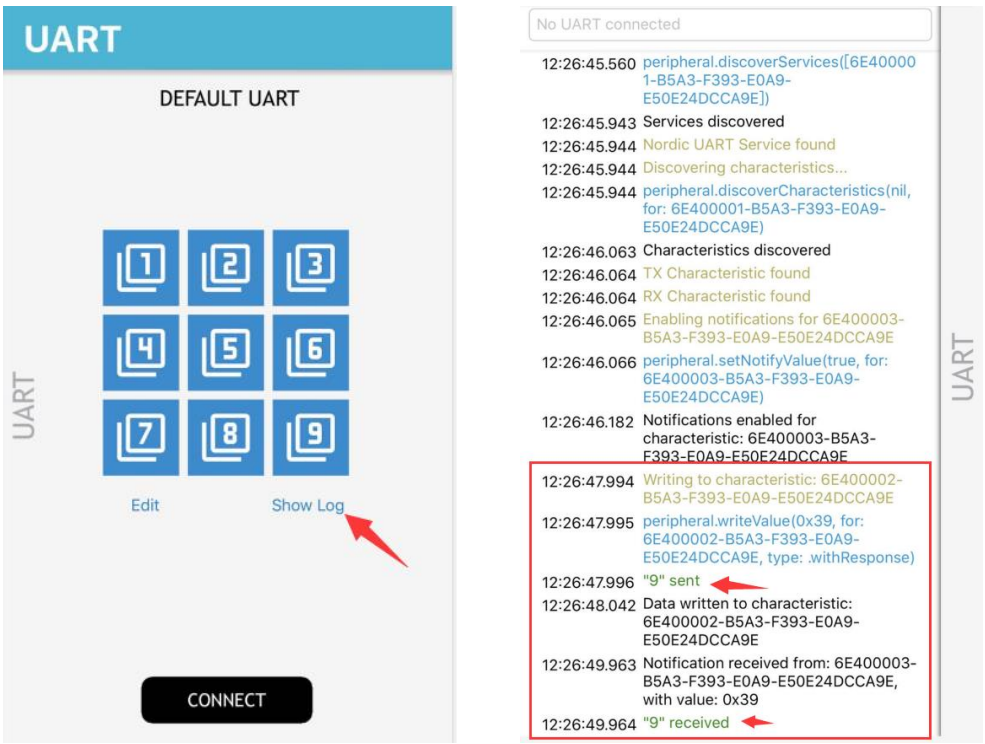


图 21-22 查看收发日志

BluefruitConnect：

BluefruitConnect 是 adafruit 开发的一款蓝牙测试 APP，拥有更直观的功能，下载后进入 UART 界面，可以直接进行可视化收发数据。

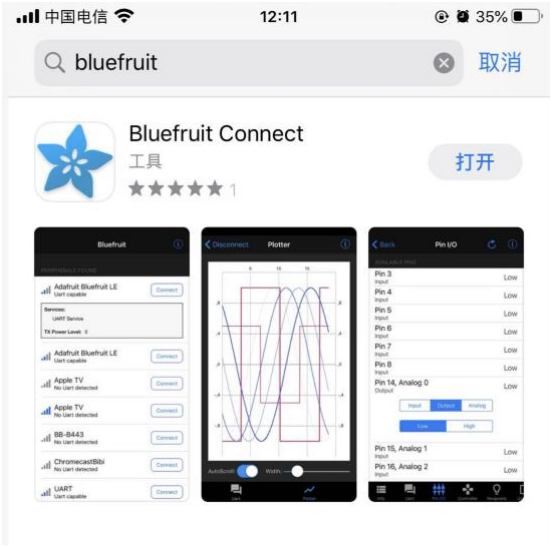


图 21-23 APPStore 下载

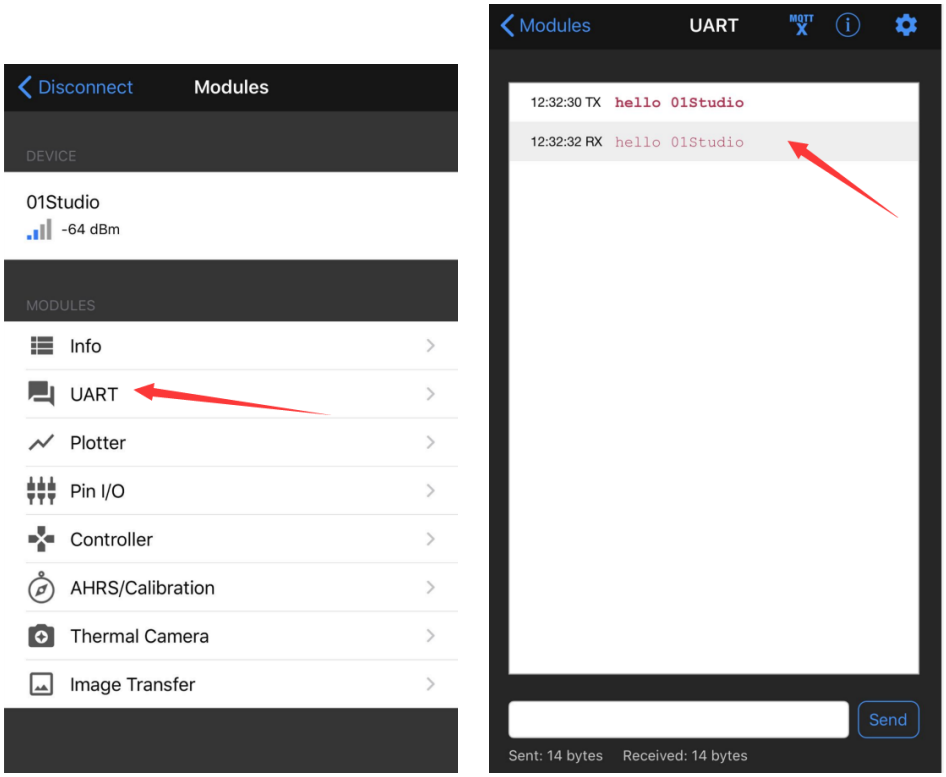


图 21-24 串口功能

- **总结：**

正如我们开头所说，数据传输是一个非常基础但实用的功能，它实现了手机和 BLE 设备的数据交互，从而可以开发各种各样的 BLE 产品和项目。

21.4 彩灯控制

- **前言：**

蓝牙彩色球泡灯相信大家都知道了，今天我们就来打造一个自己的蓝牙球泡灯原型，简单来说就是利用前面 BLE 应用学习到的知识，来控制 RGB 彩灯（NEOPIXEL）。

- **实验平台：**

pyBLE-NRF52840 开发板。

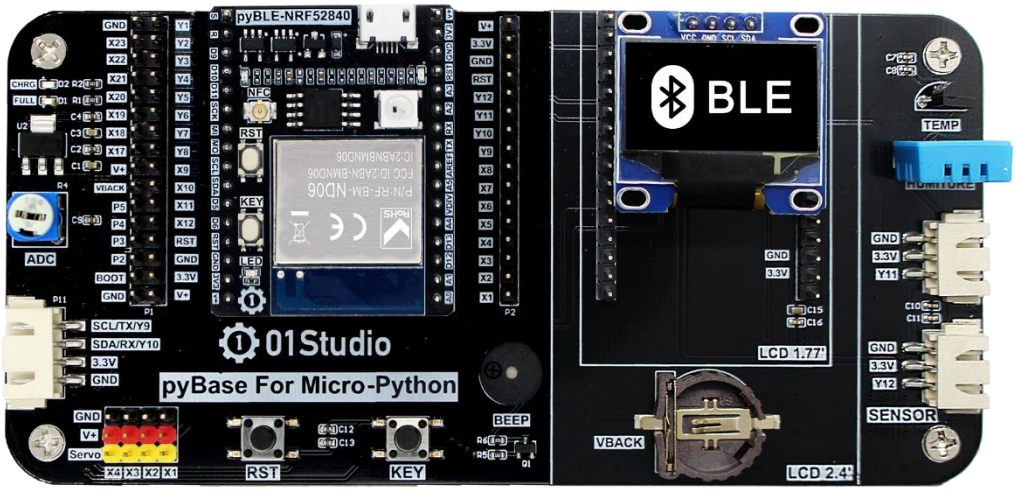


图 21-25 pyBLE-NRF52840 开发套件

- **实验目的：**

通过蓝牙控制开发板上的 Neopixel 彩灯。

- **实验讲解：**

Neopixel 在前面 RGB 彩灯章节已经学习过了，就是由一组颜色值（R,G,B），每个值分别为 0-255 从而组合出各种各样颜色。Neopixel（WS2812B）只需要一个 IO 引脚控制，支持灯珠直接串联，使用和扩展非常方便。

关于 neopixel 章节内容，请看：**20.2 RGB 彩灯（Neopixel）**

关于 BLE 数据收发章节内容，请看：**21.3 无线数据收发**

有了上面的基础，我们可以直接编程了，代码编写流程如下：

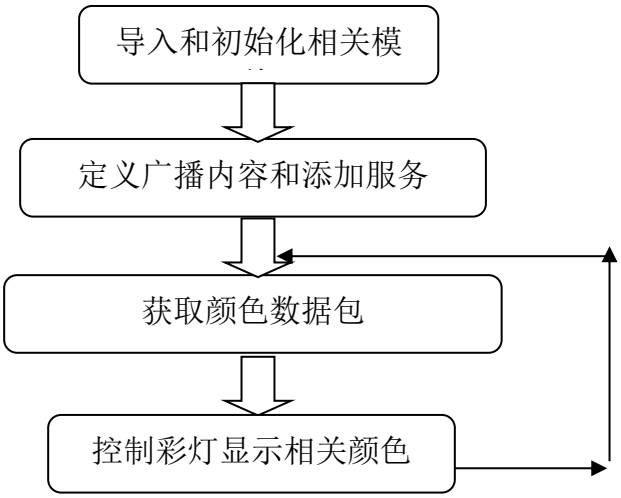


图 21-26 代码编写流程

参考代码：

```
'''
实验名称：无线彩灯控制
版本：v1.0
日期：2020.5
作者：01Studio
说明：编程实现手机控制彩灯。
'''

#导入相关模块
import board,neopixel
from adafruit_ble import BLERadio
from adafruit_ble.advertising.standard import
ProvideServicesAdvertisement
from adafruit_ble.services.nordic import UARTService
from adafruit_bluefruit_connect.packet import Packet
from adafruit_bluefruit_connect.color_packet import ColorPacket
```

```
#构建蓝牙对象
ble = BLERadio()


#定义广播名称
ble.name = '01Studio'


#构建 UART 服务
uart_server = UARTService()


#广播添加 UART 服务
advertisement = ProvideServicesAdvertisement(uart_server)


#定义 neopixel 引脚，默认使用板载 neopixel
pixels = neopixel.NeoPixel(board.NEOPIXEL, 1, brightness=0.1)


while True:
    # 广播
    ble.start_advertising(advertisement)


    #等待连接
    while not ble.connected:
        pass


    #已连接
    while ble.connected:


        #获取 UART 服务数据
        packet = Packet.from_stream(uart_server)


        #判断收到的包和颜色包类型是否一致
```

```
    if isinstance(packet, ColorPacket):
        print(packet.color)
        pixels.fill(packet.color) #修改灯珠颜色
```

- **实验结果：**

本实验可以使用 adafruit 开发的 bluefruitConnect APP，改 APP 已经将颜色数据打包好，连接后可以通过颜色控盘直接测试。

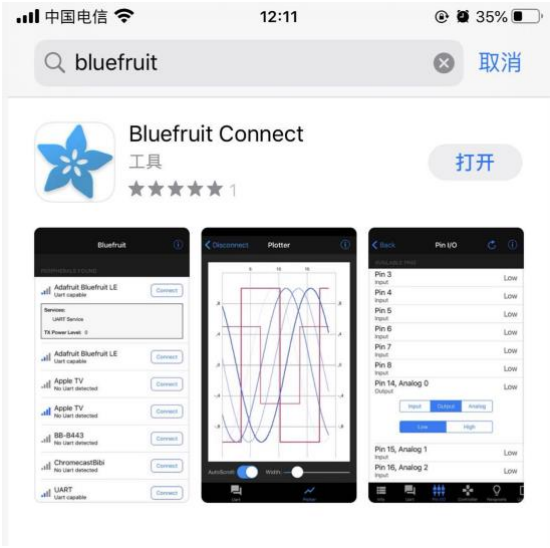
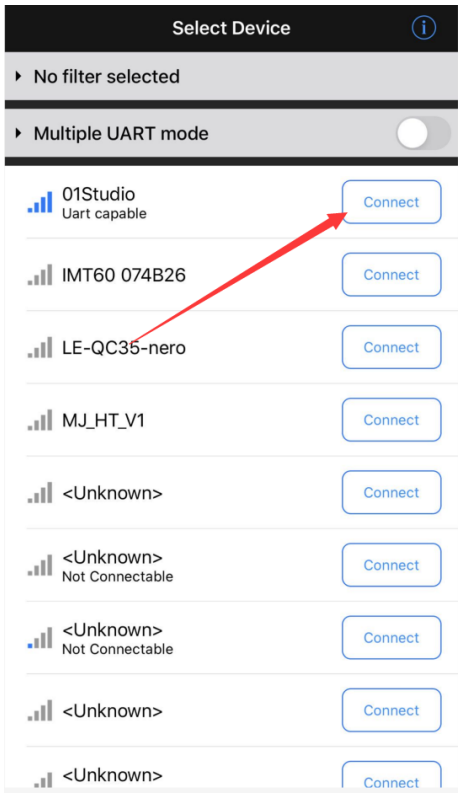
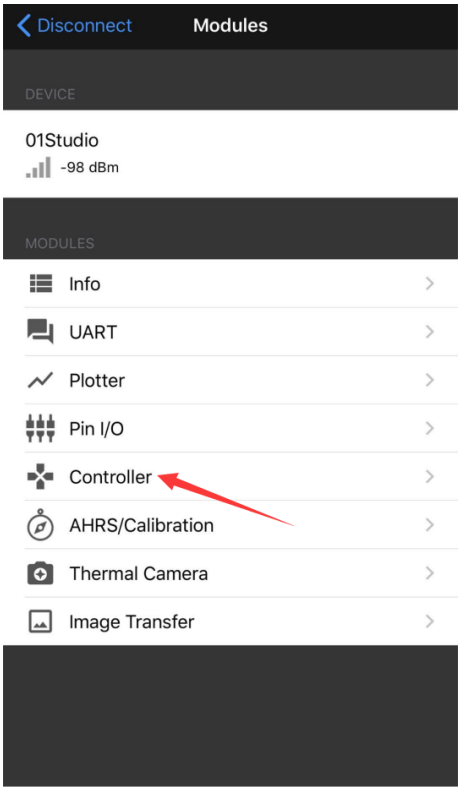
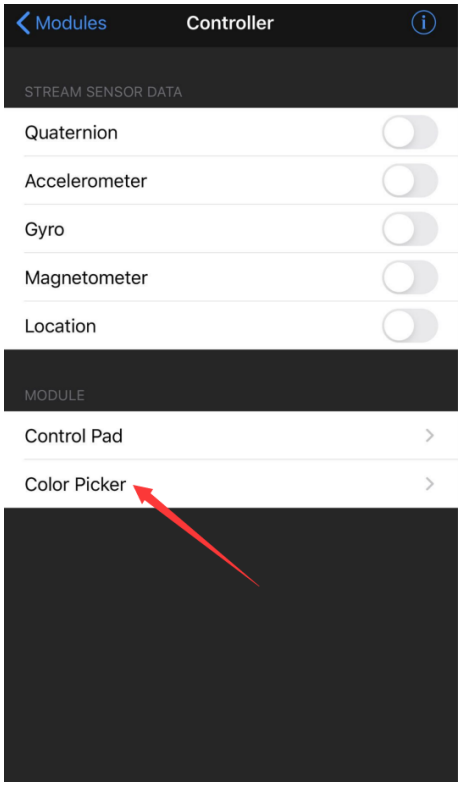


图 21-27 通过 APP Store 下载安装

开发板运行程序，手机连接上后打开 RGB 彩灯控制面板：







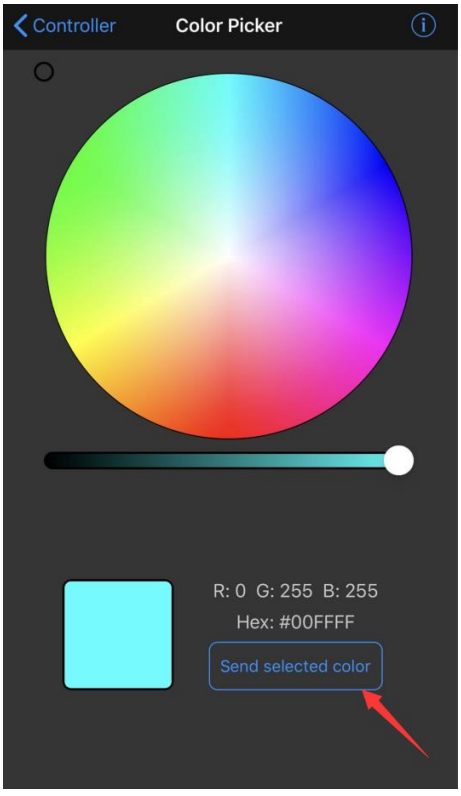


图 21-28 彩灯控制面板位置

打开后点击选中颜色，点击发送数据，即可看到开发板的 neopixel 颜色发生变化。

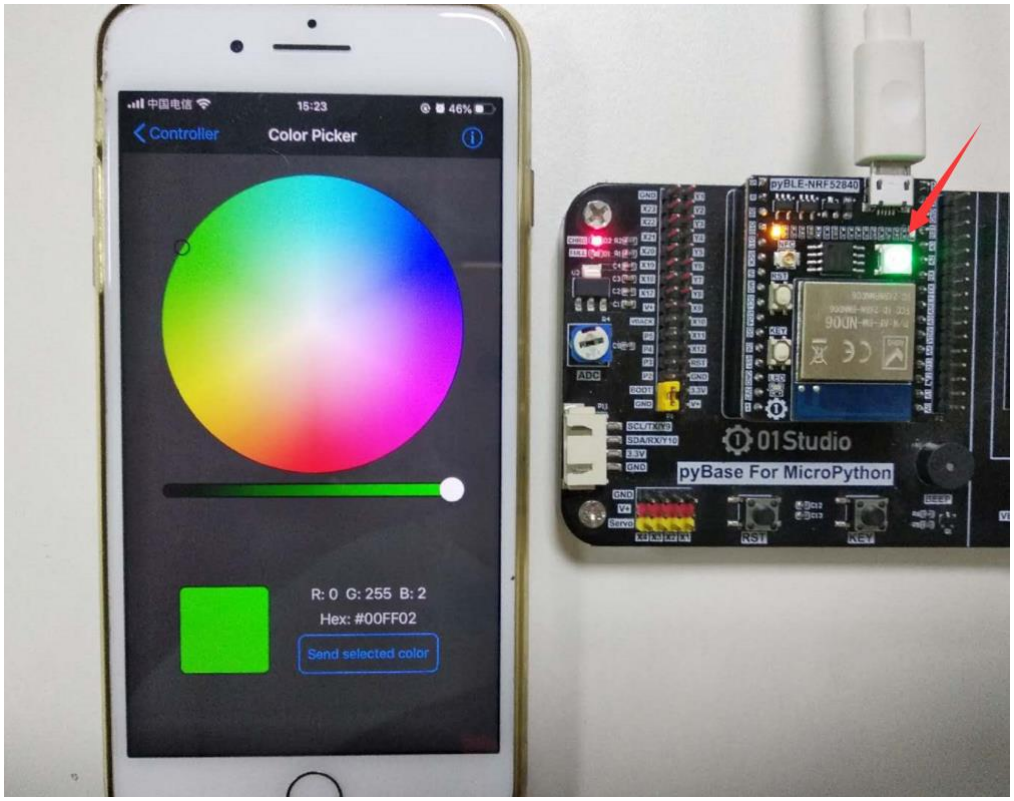


图 21-29 实验结果

- **总结：**

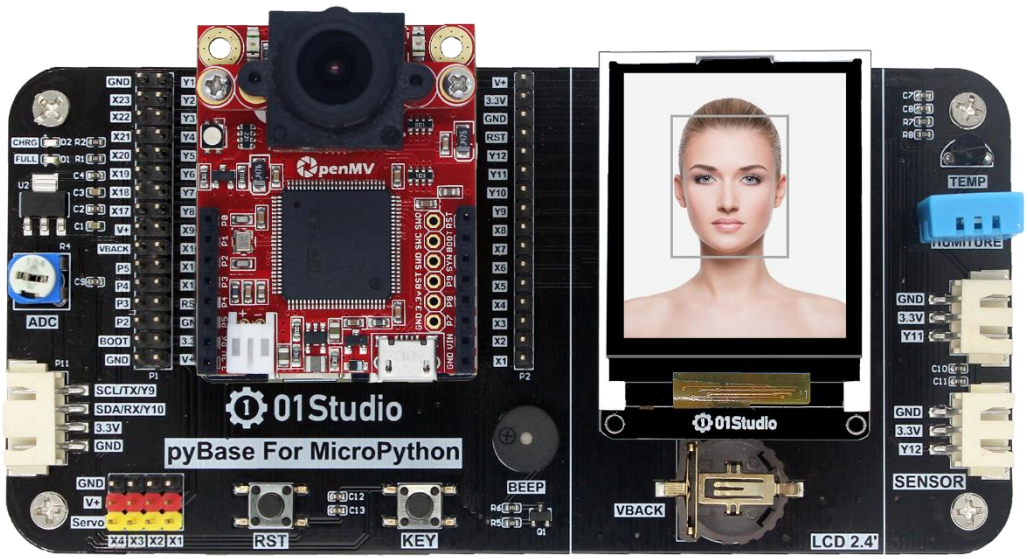
RGB 彩灯是非常常见的设备，本实验通过简单的代码就实现了无线控制，有兴趣的用户可以参考前面章节，改造成控制 RGB 灯带，打造自己的无线灯带控制器。

第六部分　基于 OpenMV4 平台

OpenMV（开源机器视觉）是美国克里斯·安德森及其团队基于 MicroPython 发起的开源项目，该项目旨在创建低成本，可扩展，使用 Python 驱动的机器视觉模块，旨在成为“机器视觉的 Arduino”。开源目标是使机器视觉算法更接近制造商和业余爱好者。该开源项目完成了困难且耗时的算法工作，为我们的创作留出了更多时间！

OpenMV Cam 就像一个功能强大的 Arduino，带有一个用 Python 编程的板载摄像头。我们可以轻松地在 OpenMV Cam 所看到的图像上运行机器视觉算法，因此您可以在几秒钟内跟踪颜色，检测面部等等，然后在实际环境中控制 I / O 引脚。

如果你学习了前面的平台，那么恭喜你，对于 OpenMV 你会非常快速上手。如果你没有学习过任何一个 MicroPython 平台那也没关系，你可以借助 OpenMV4 平台的学习从而入门 MicroPython。



pyAI-OpenMV4 开发套件

第22章　开发环境快速建立

22.1　基于 Windows

22.1.1　安装 python3

说明：我们提供安装 python3 教程只是为了在 PC 上体验一下 python3 安装和使用过程，由于 micropython 固件集成了 python 库，所以电脑并非一定要安装，本节可以跳过。

请阅读 第二部分　　3.1.1　　安装 python3 章节内容，这里不再重复。

22.1.2　安装开发软件 OpenMV IDE

与其它平台不一样，由于 OpenMV 拥有成熟的团队，他们为该平台量身定做了 IDE，可以在官网下载，我们使用该 IDE 可以轻松进行开发。而且拥有 Windows、Mac OS、Linux、树莓派等版本。

下载地址：https://openmv.io/pages/download ，下载界面如下图。用户可以根据自己系统选择合适的版本下载安装！

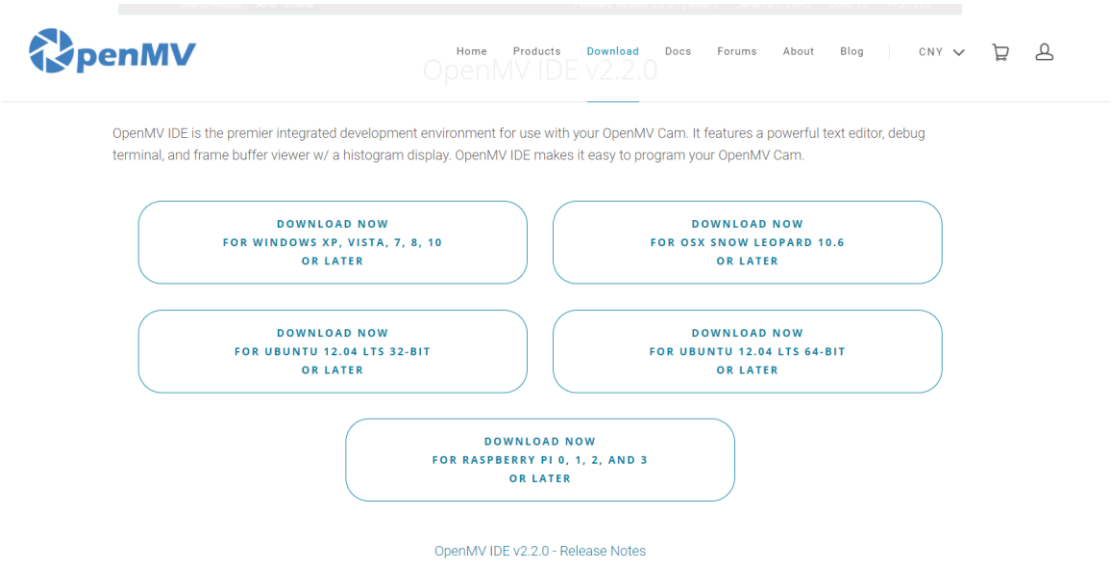


图 22-1 OpenMV IDE 下载界面

安装完成后打开软件，如下图所示，至此安装完成。

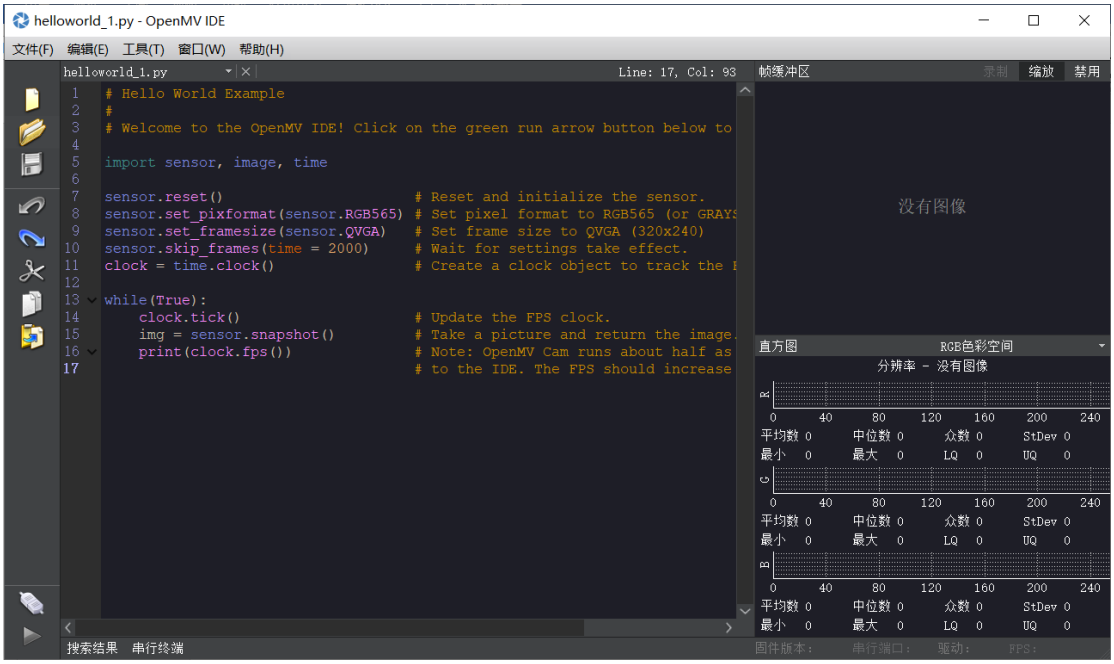


图 22-2 OpenMV IDE

22.1.3 开发套件使用

22.1.3.1 驱动安装

我们在上一节安装完 OpenMV IDE 后，驱动也一起安装了。我们将 pyAI-OpenMV4 开发板通过 MicroUSB 数据线连接到电脑：

我们将 MicroPython 开发板通过 MicroUSB 数据线连接到电脑：

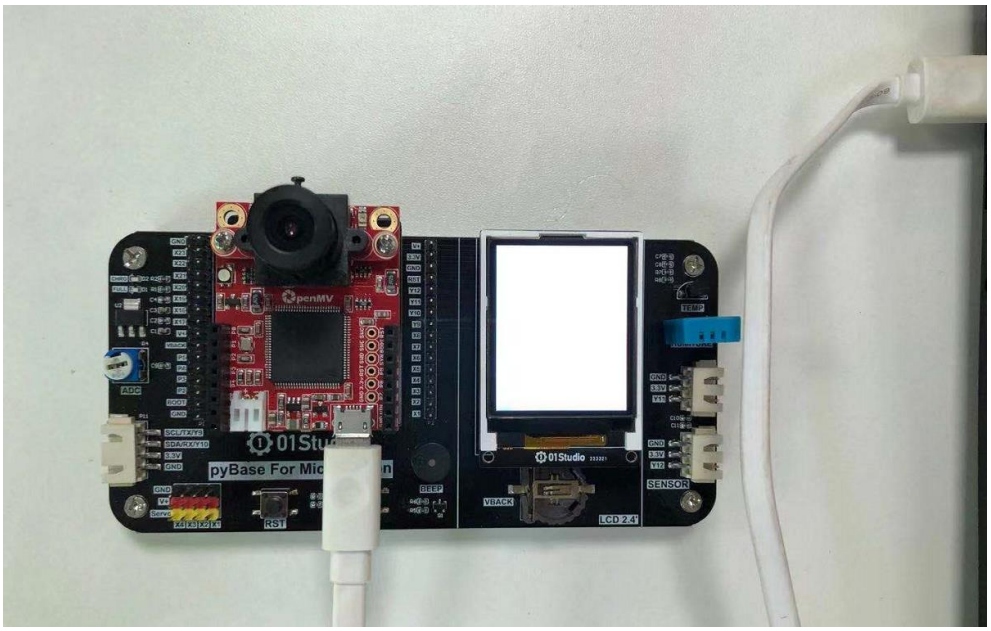


图 22-3 通过 MicroUSB 线连接到电脑

连接电脑后，会发现出现一个 U 盘，点击打开，看到以下文件。

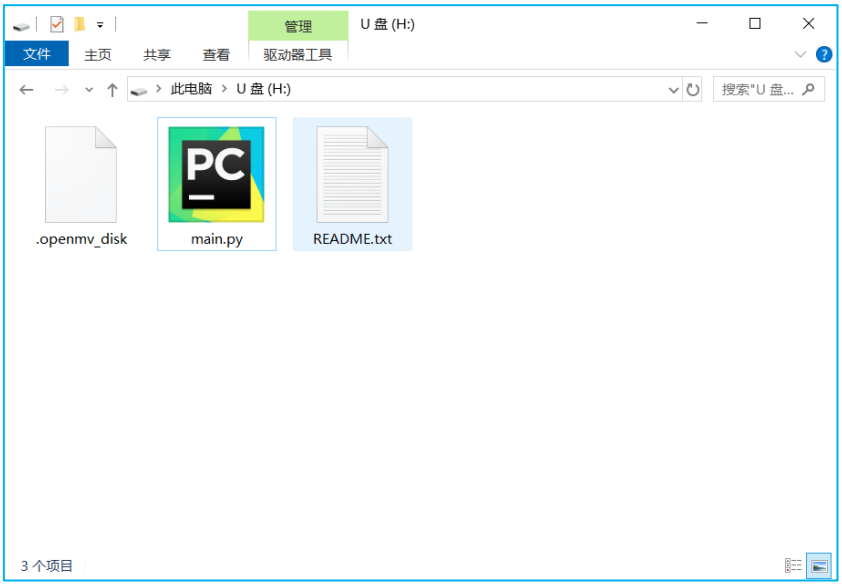


图 22-4 OpenMV4 的系统文件

下面是对主要文件的简介：

1. **main.py：主函数代码文件，上电后首先执行；**
2. **README.txt：说明文件；**

另外系统还自动安装一个 USB 转串口的驱动，用于调试，鼠标右键点击 “我的电脑”—属性—设备管理器：能找到 OpenMV4 虚拟串口新设备:

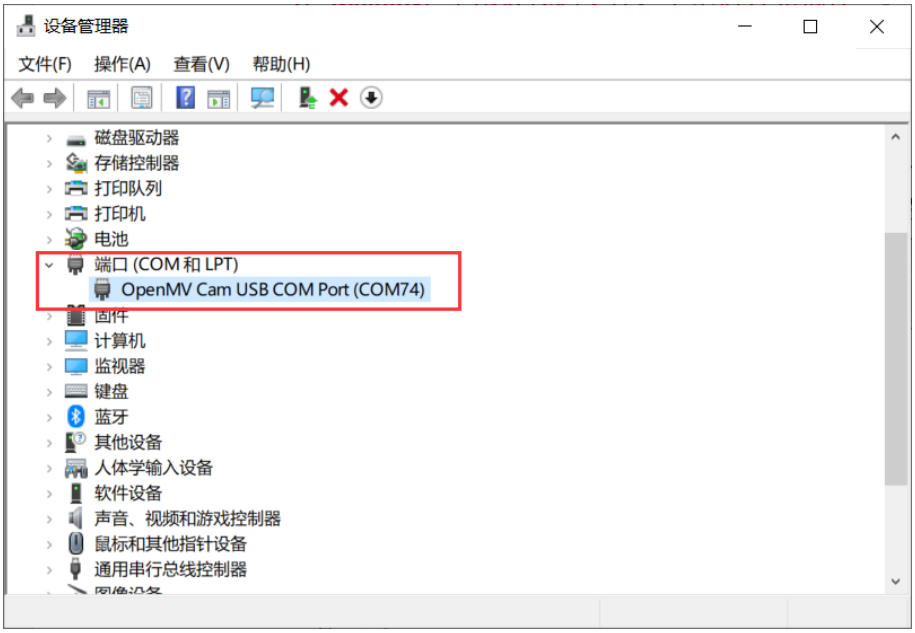


图 22-5 OpenMV 串口驱动

注意事项：

部分用户的 Win7 系统由于使用的是 Ghost 版系统，Ghost 系统精简了很多系统文件，有可能导致驱动无法安装。我们不建议开发者安装 Ghost 的系统。这时候建议用户采取以下措施。

（1）使用 360 的系统重装功能，实测重装后驱动可以安装；

（2）升级成 Win10 系统。

我们更推荐方法（2），因为微软已经宣布停止对 Win7 进行更新，另外就是我们的教程都会基于 Win10 系统。

22.1.3.2 例程测试

我们使用 OpenMV IDE 来进行我们的第一个实验，借此来熟悉开发环境。将 pyAI-OpenMV4 开发板通过 MicroUSB 线连接到电脑。打开 OpenMV IDE，点击左下角连接按钮：

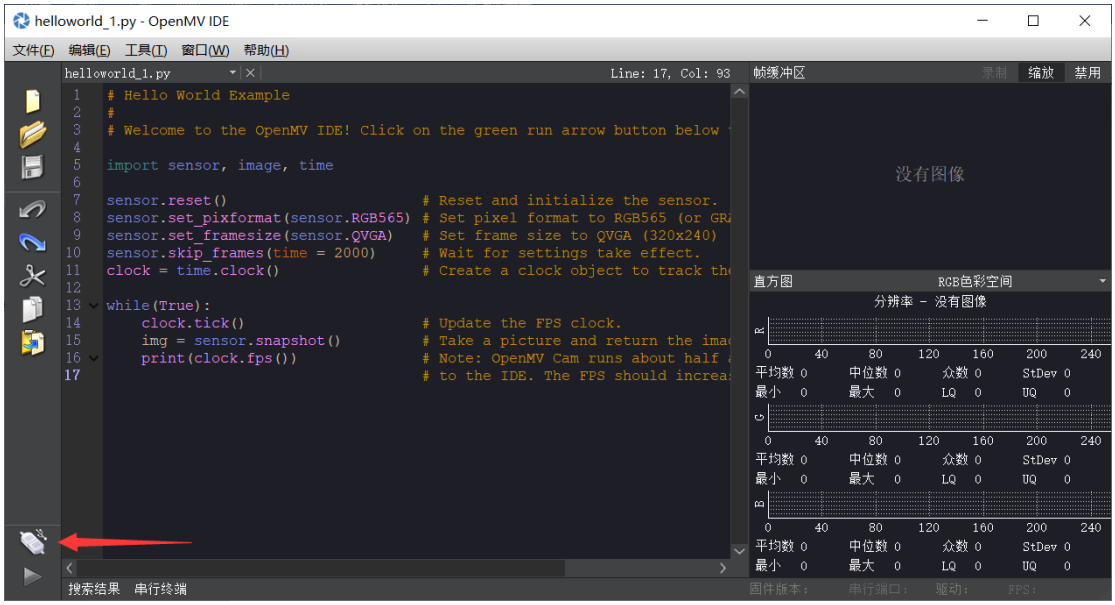


图 22-6 点击连接按钮

连接成功后，运行按钮变成绿色。

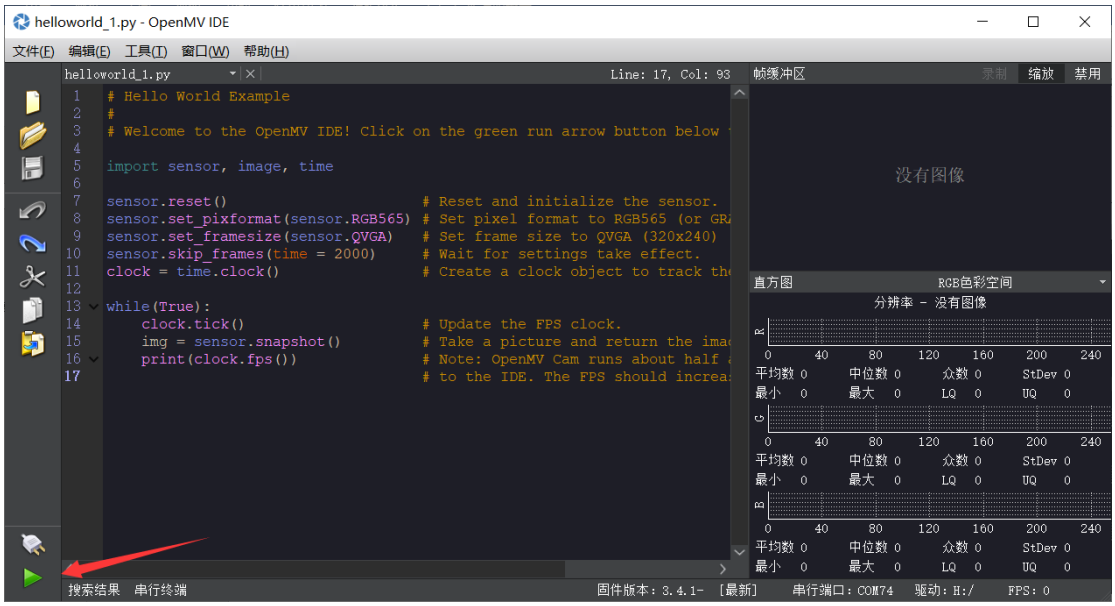


图 22-7 连接成功

当前的文件是 IDE 自带的 hello_world.py 程序文件，我们点击绿色按键“运行”按钮（记得打开摄像头盖），可以看到程序在运行，右上角出现当前摄像头实时采集图像和色彩直方图，说明实验成功:

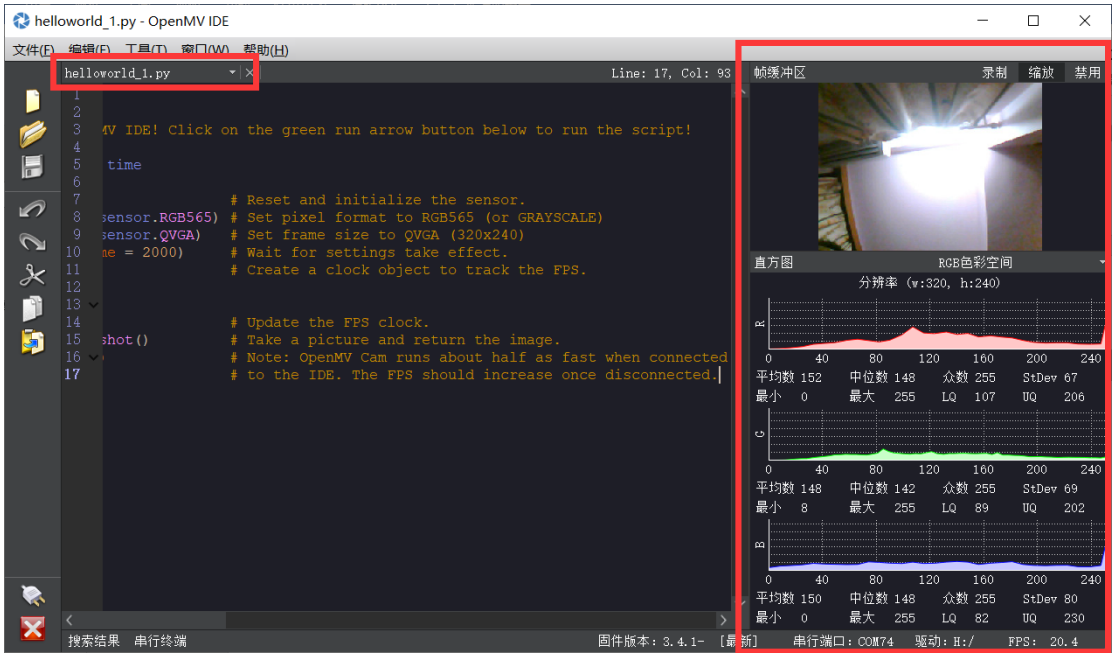


图 22-8 第一个实验

以上测试是基于 IDE 的运行功能，当然你也可以直接将 py 文件保存到 U 盘文件系统中，这样程序不再依赖 IDE 就可以直接运行。具体方法如下：

将代码放在 main.py 文件中（main.py 为上电自动运行的文件）。然后将 main.py 复制到设备的文件系统中，设备红色 LED 灯亮表示文件正在重写，需要等待红色灯灭掉，然后按下复位按键。程序即重新运行。

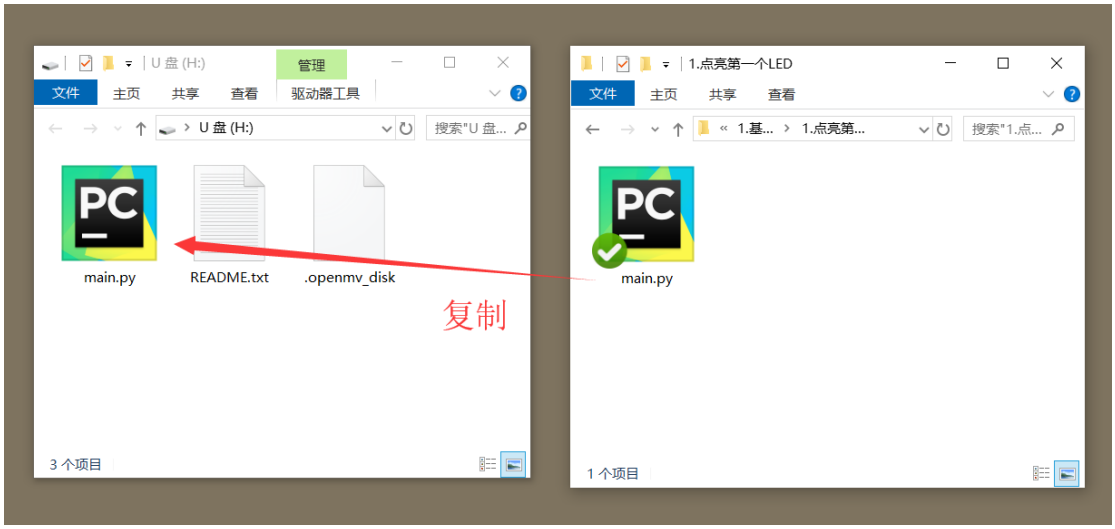


图 22-9 复制文件到 OpenMV4 文件系统

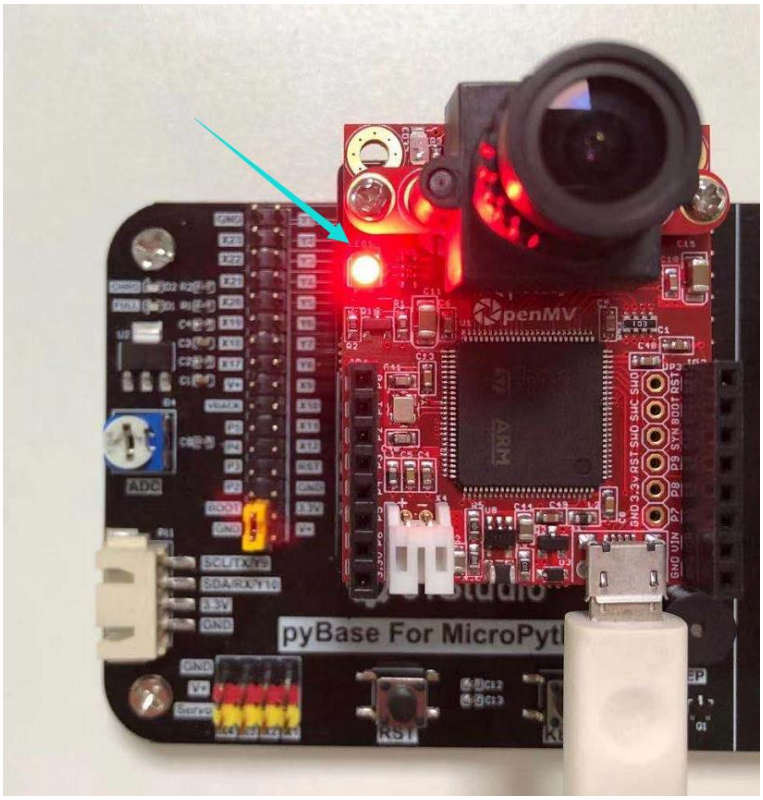


图 22-10 等待红灯熄灭再操作

OpenMV IDE 集成了串口调试窗口，点击 IDE 左下角“串口终端”，即可观察到串口打印的信息。

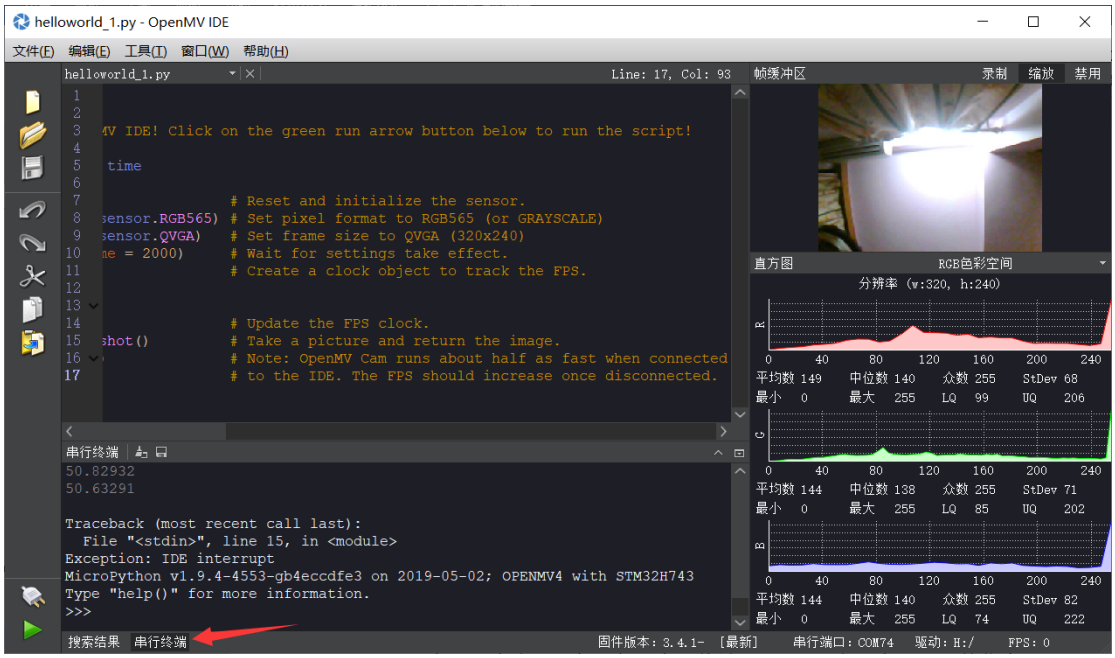


图 22-11 串口终端调试

但这个串口终端只能显示数据，并不能实现交互，如果需要 REPL【读取(Read)-运算(Eval)-输出(Print)-循环(Loop)】串口交互调试，请使用 putty 或相串口工具。具体请参阅：**第二部分　　3.1.3.3　　REPL 串口交互调试** 章节内容。

22.1.3.3　固件更新

当 OpenMV4 上的固件意外丢失或者我们希望升级到较新版本固件时候，就需重新烧录固件。OpenMV4 上面是一个 STM32 单片机，所以这个操作相当于给 STM32 单片机重新烧录程序。

我们采用 DFU+OpenMV IDE 结合的烧写方式，DFU 烧写方式优势是不需要 ST-LINK 一类仿真器，只需要安装一个软件即可。我们打开 MicroPython 开发套件配套资料\开发工具\Windows\固件更新工具\DfuSe_Demo_V3.0.5 目录下的 DfuSe 安装软件。DFU 软件安装完成后相当于电脑所需要的驱动也安装了。

将开发板上的默认连接的 BOOT--GND 跳线帽连接到 3.3V，让 OpenMV4 进入 DFU 烧写模式，如下图所示：

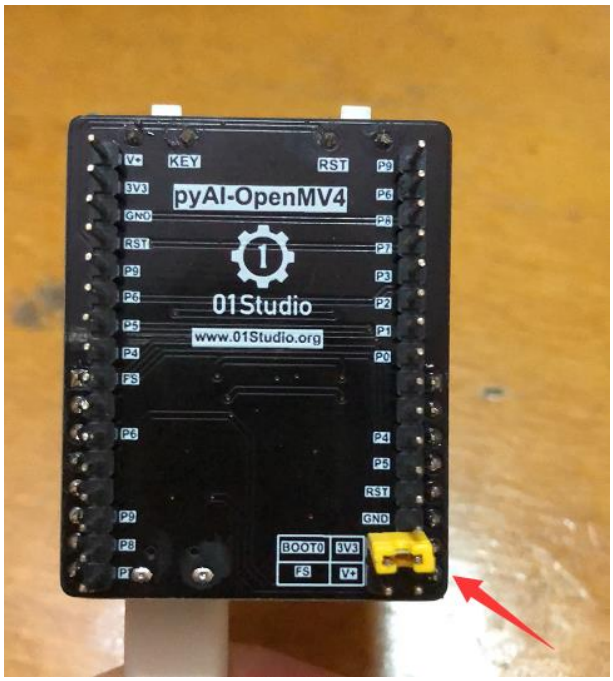


图 22-12　pyAI-OpenMV4 进入 DFU（BOOT0 与 3.3V 相连）

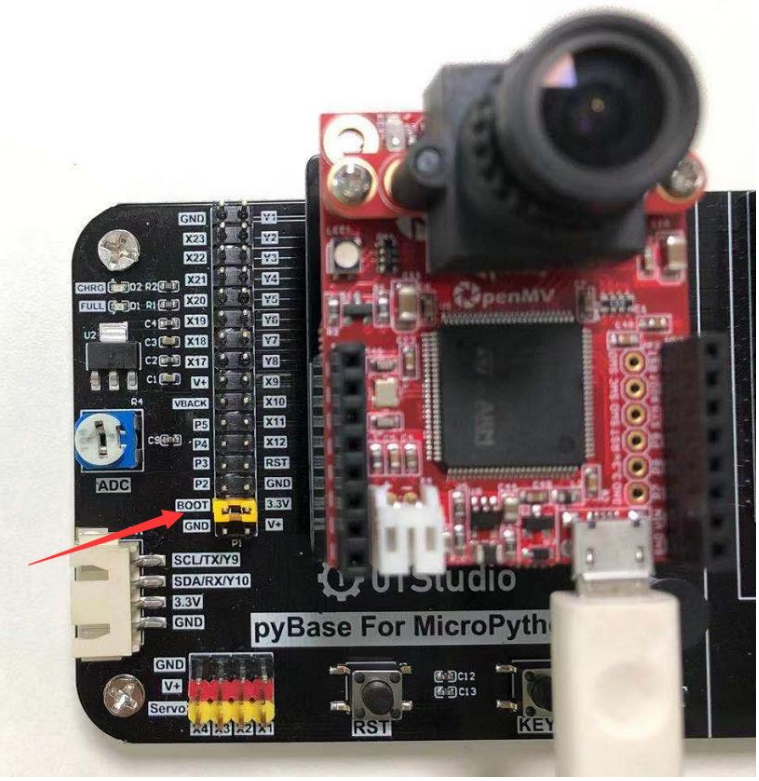


图 22-13 开发套件进入 DFU（BOOT0 与 3.3V 相连）

我们先烧写 bootloader.dfu，打开刚刚安装的 DFU 软件，按下开发板 RST 复位键，可以见到 DFU 软件成功识别出 ST 的芯片。

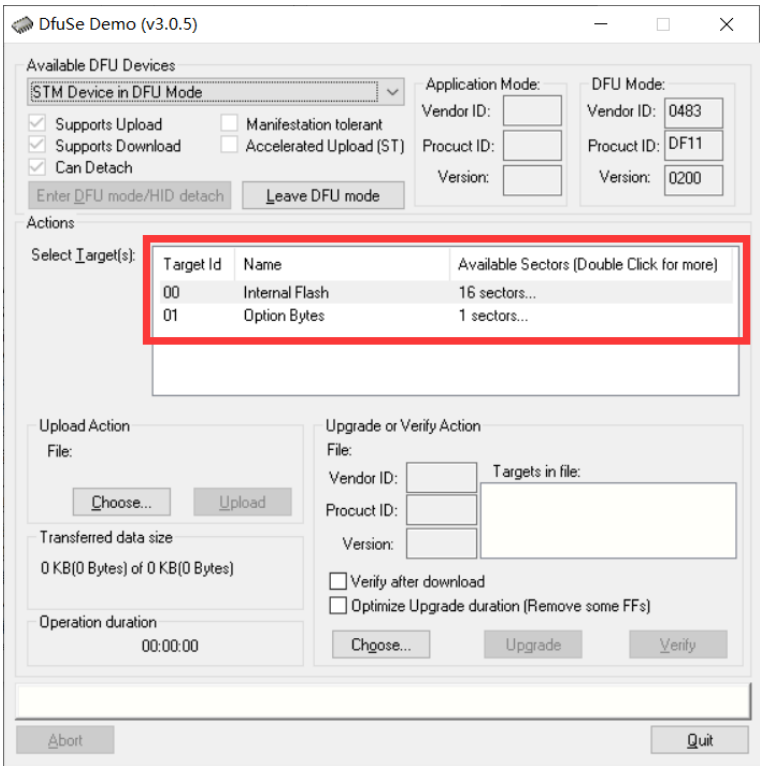


图 22-14

点击 choose，在 MicroPython 开发板配套资料\相关固件\pyAI-OpenMV 目录下的 bootloader.dfu。

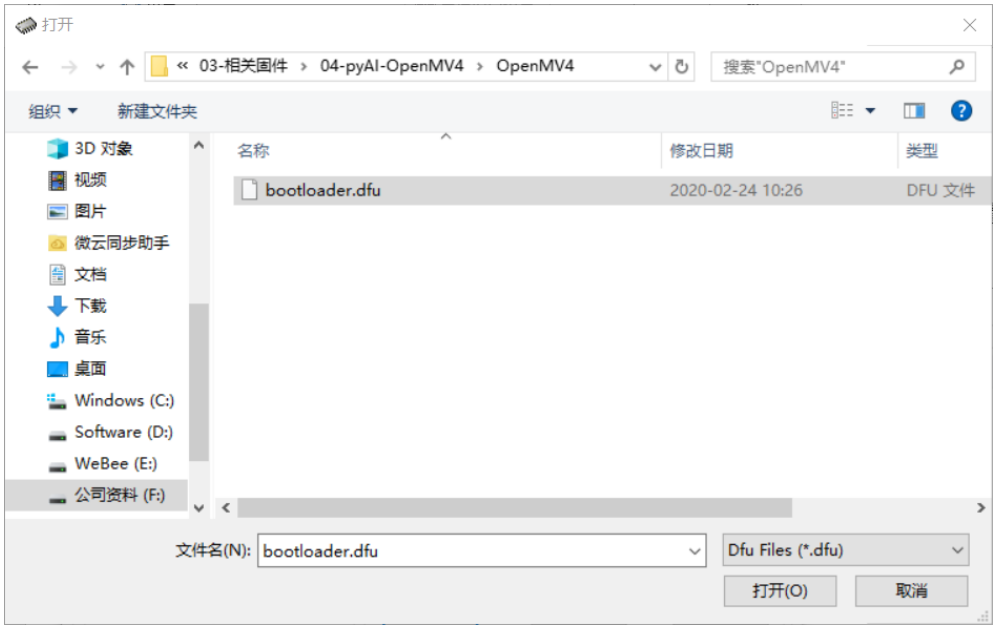


图 22-15 选择固件

点击 Upgrade 开始烧录。

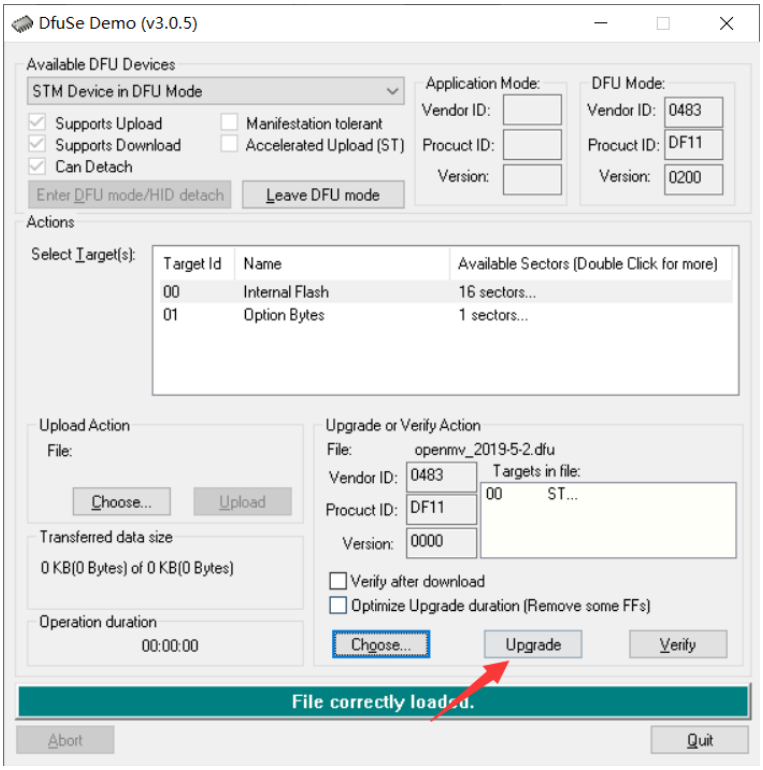


图 22-16

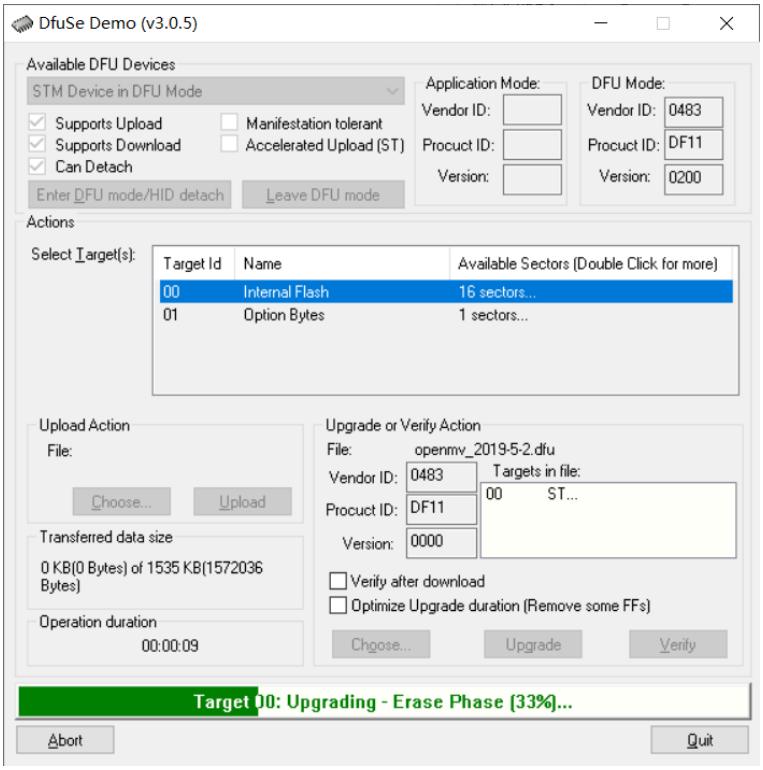


图 22-17 正在烧录

烧录完成后如下图所示：

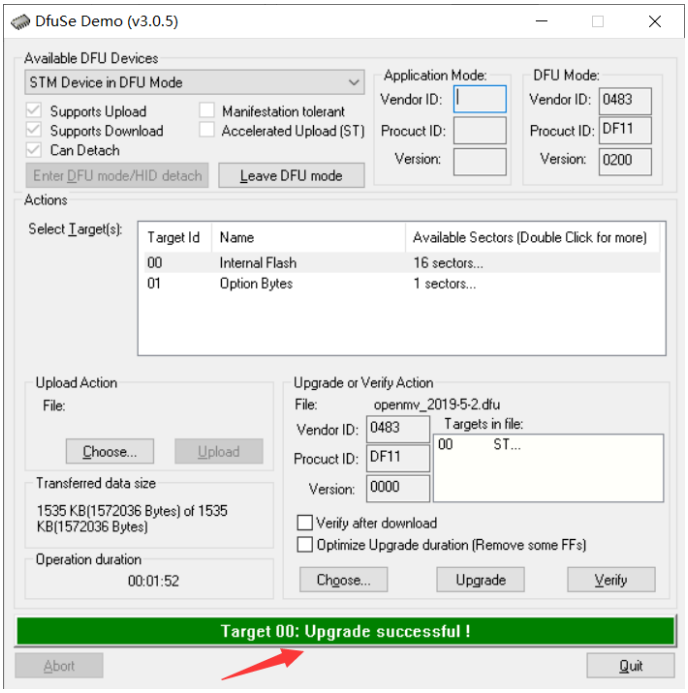


图 22-18 烧录完成

保持 BOOT 与 3.3V 用跳线帽相连，将 pyAI-OpenMV4 通过 USB 线连接到电脑，打开 OpenMV IDE，点击左下角连接按钮：

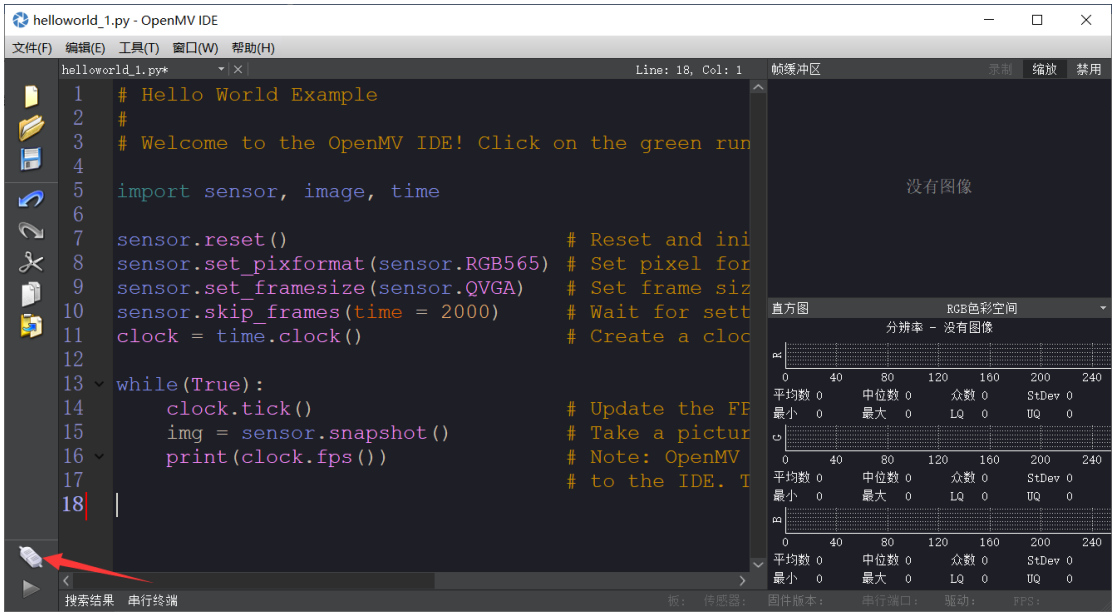


图 22-19

在弹出对话框“没有找到 OpenMV Cams!”,直接点击 OK：

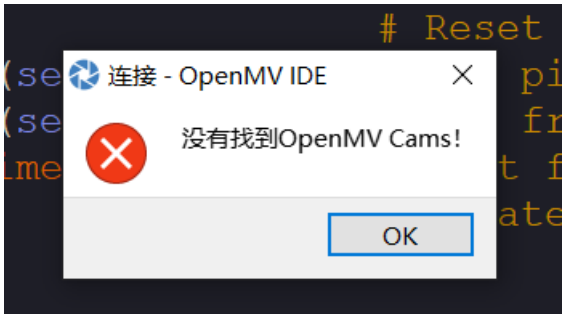


图 22-20

询问 OpenMV Cam 是否变砖了，点击 Yes:

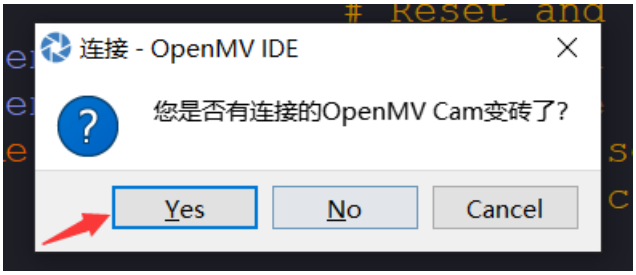


图 22-21

选择开发板类型，OpenMV4 就是下图的 H7（**如果你使用 openmv plus，则选择 plus 主板类型**）：

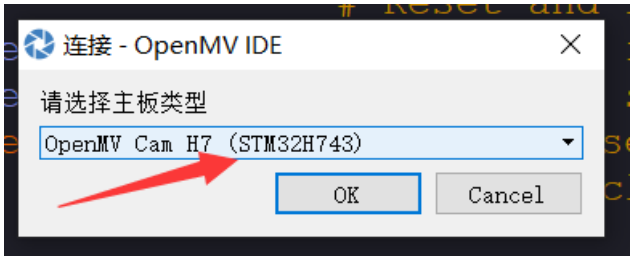


图 22-22

询问是否擦除文件系统，点击 Yes：

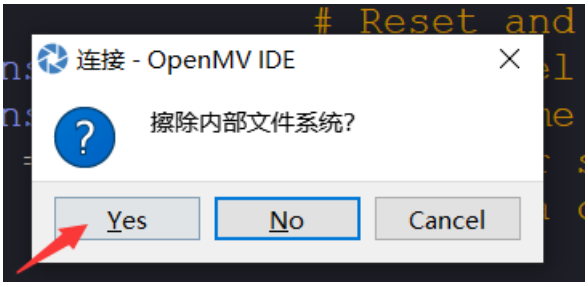


图 22-23

出现以下界面：

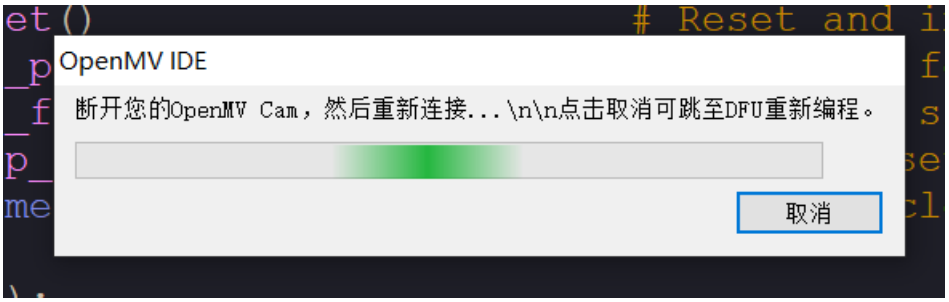


图 22-24

这时候将 BOOT 与 3.3V 连接的跳线帽拔掉或者连接到开发板上的 GND 备用。

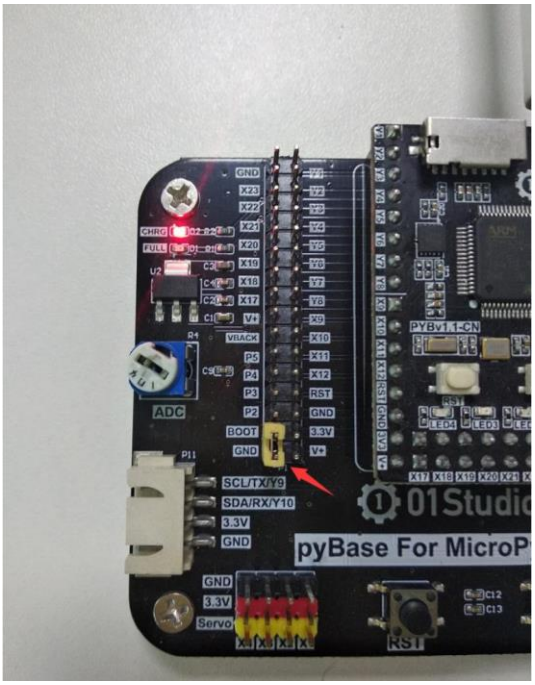


图 22-25

按下开发板复位键（RST），或者拔掉 USB 线重新插入，开始擦除：



图 22-26

擦除后开始烧写固件：



图 22-27

烧写完成后出现下面界面，然后模块进入自检，等开发板蓝色灯闪亮再点击 OK 即可：

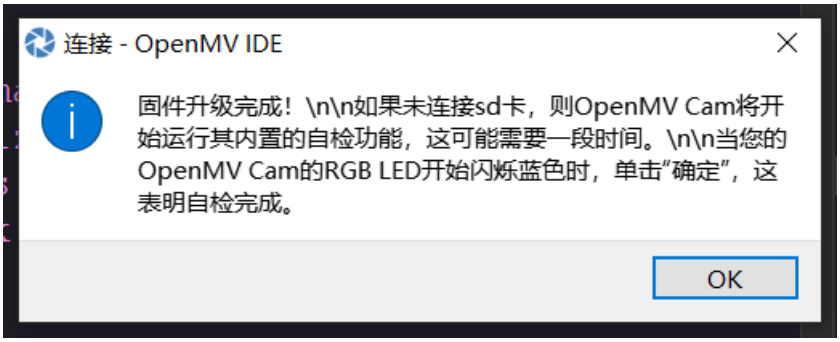


图 22-28

弹出 U 盘盘符，至此，重烧固件完成。



图 22-29

22.1.3.4 正版激活

OpenMV4 的官方团队为 OpenMV4 打造了完善的开发环境 OpenMV IDE，但并不是无偿的。需要购买官方激活码，俗称（KEY）来对设备进行激活，如果你的 OpenMV4 设备尚未激活，那么使用官方 OpenMV4 IDE 将在每次连接设备时候弹出 3 次提示激活，影响用户体验。**（01Studio 官方 openmv 系列产品出厂均带官方正版 KEY）**

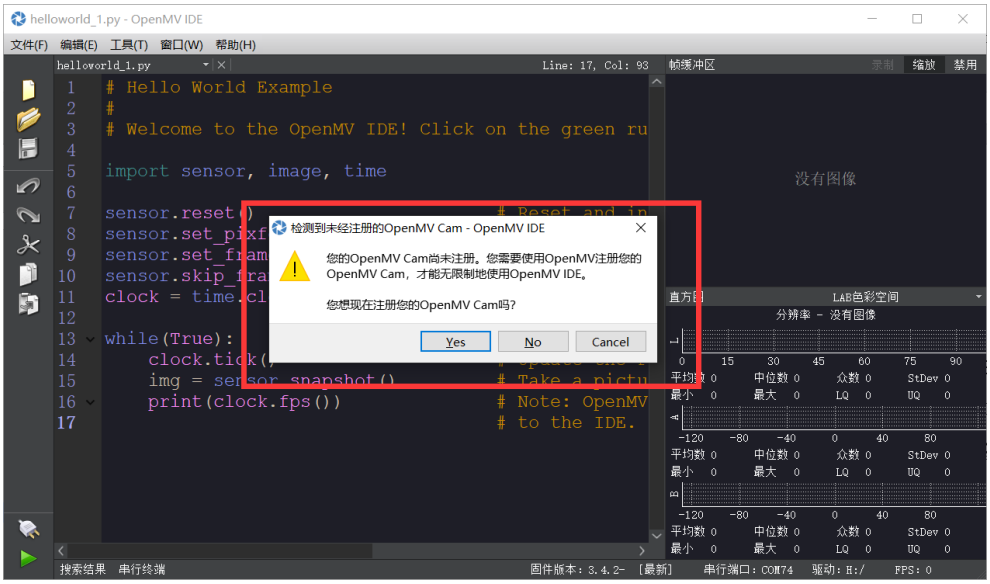


图 22-30 提示激活

这时候你需要到 OpenMV 官网或者联系 01Studio 购买 KEY，KEY 为一串 25 个字符的编码，一旦激活后跟当前 OpenMV 设备唯一绑定，IDE 即可正常使用。

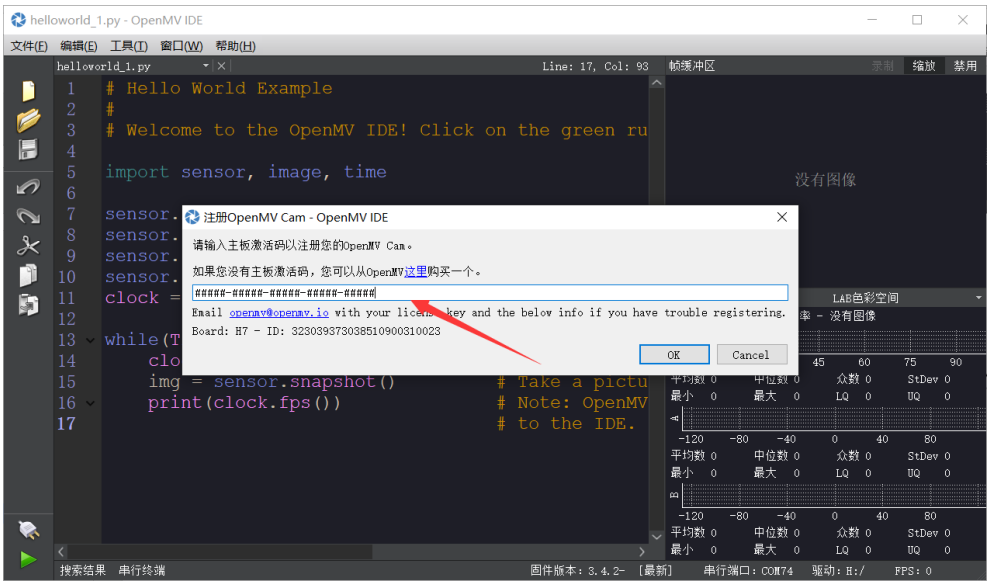


图 22-31 激活 OpenMV 设备

尽管 **OpenMV** 是开源项目，官方在开发环境和升级更新花了不少精力，因此我们 **01Studio** 出品的产品均为已激活产品以表示对其团队支持，用户如果是在市面上其它渠道购买 **OpenMV** 设备而没激活的建议购买 **KEY** 激活以支持官方。

22.2 基于 Mac OS

22.2.1 安装 python3

具体请参考：第二部分 3.2.1 安装 python3 这里不再重复。

22.2.2 安装开发软件 OpenMV IDE

这个跟 Windows 环境安装方法完全一样。具体请参考：**22.2.2 安装开发软件 OpenMV IDE** 章节内容，这里不再重复。

下载地址：https://openmv.io/pages/download ，下载界面如下图。用户可以根据自己系统选择合适的版本下载安装！

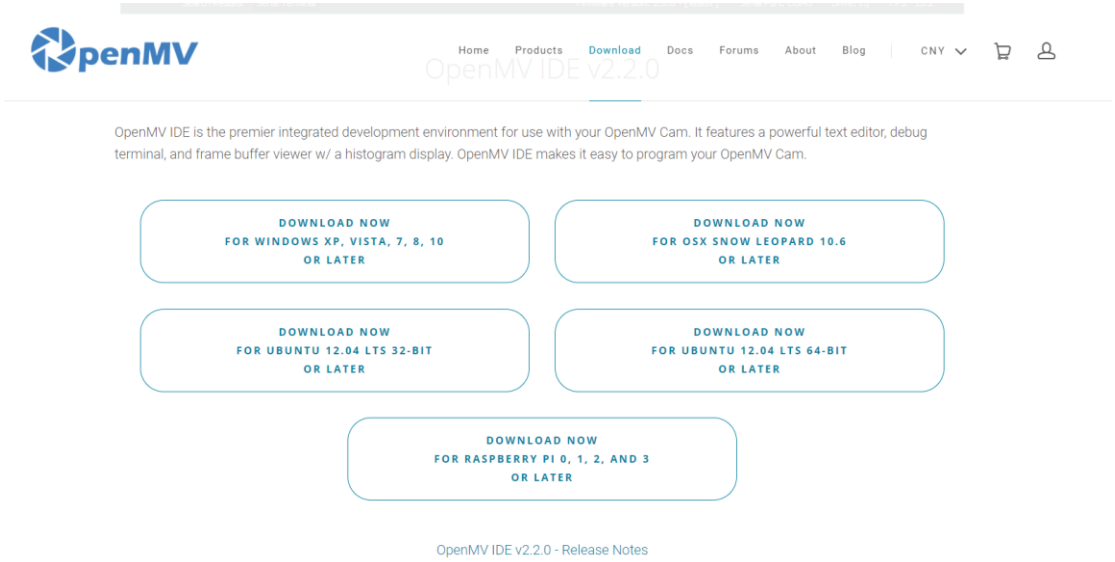


图 22-32 OpenMV IDE 下载界面

22.3 基于 Linux（树莓派）

OpenMV IDE 提供 Linux 和树莓派版本，下载压缩包看 Readme 文件按步骤安装即可，使用方法请参考：**22.2.2 安装开发软件 OpenMV IDE** 章节内容，这里不再重复。

下载地址：https://openmv.io/pages/download ，下载界面如下图。用户可以根据自己系统选择合适的版本下载安装！

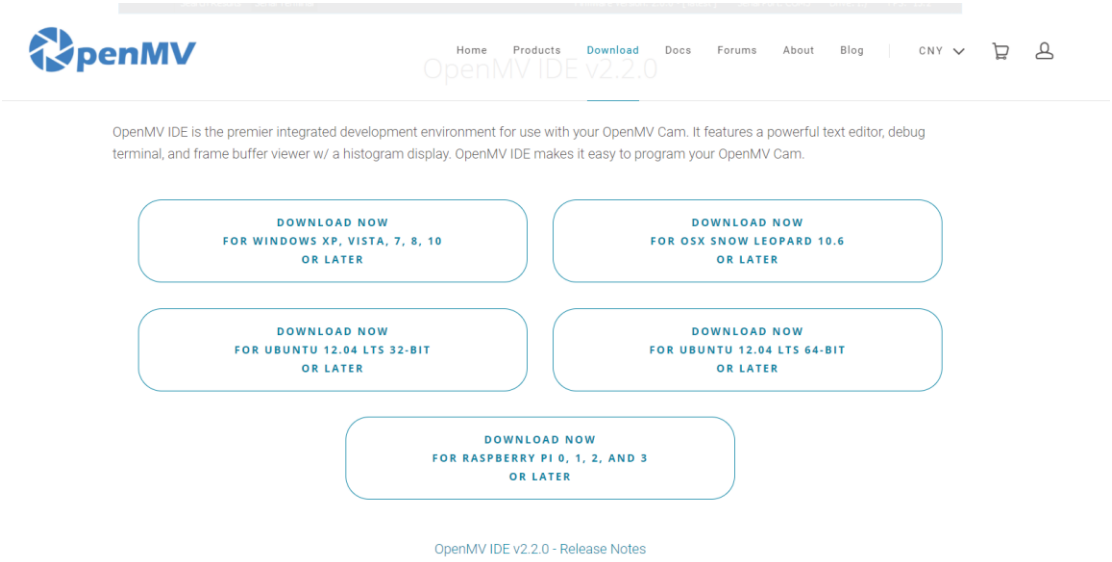


图 22-33 OpenMV IDE 下载界面

第23章　基础实验

MicroPython 更强调的是针对应用的学习，强大的底层库函数让我们可以直接关心功能的实现，也就是说我们只要理解和熟练相关的函数用法，就可以很好的玩转 MicroPython。它让我们可以做到不关心硬件和底层原理（当然有兴趣和能力的小伙伴可以深入研究）而直接跑起硬件。

基础实验是针对一些简单的实验学习讲解，可以让我们更快和更直接感受到 MicroPython 针对嵌入式开发的强大之处， 我后面的学习和开发夯实基础。

请先看实验讲解格式预览，每一节我们都会以以下形式讲解，图文并茂，务求达到快速理解和学习的作用：

（1） 前言：简单介绍这个实验；

（2） 实验平台：实验所用到的开发板、配件和接线说明；

（3） 实验目的：本实验要实现的功能；

（4） 实验讲解：对函数、代码、编程方法以及实验的详细讲解；

（5） 实验结果：记录程序下载到开发板上的图片示例；

（6） 总结:对实验进行总结。

23.1 点亮第一个 LED

- **前言：**

相信大部分人开始学习嵌入式单片机编程都会从点亮 LED 开始，基于 OpenMV4 平台的 MicroPython 的学习也不例外，通过点亮第一个 LED 能让你对编译环境和程序架构有一定的认识，为以后的学习和更大型的程序打下基础，增加信心。

- **实验平台：**

pyAI-OpenMV4。



图 23-1 pyAI-OpenMV4

- **实验目的：**

学习 LED 的点亮，点亮 LED3（蓝灯）。

- **实验讲解：**

pyAI-OpenMV4 上总共有 4 个 LED，分别是 LED1(红色)、LED2(绿色)、LED3(蓝色)和 LED4(红外拍照灯)；我们可以直接通过 MicroPython 对象编程来控制。控制 LED 使用到 LED 对象。LED 的构造函数和使用方法如下表所示：

构造函数
pyb.LED(id)
LED 对象在 pyb 模块下。
使用方法
LED.on()
点亮 LED，输出低电平。
LED.off()
关闭 LED，输出高电平。
LED.toggle()
LED 亮灭状态翻转，输出电平翻转

表 23-1 LED 对象

从上表可以看到，LED 通过对象编程让使用变得非常简单。我们先来看看 OpenMV4 的 LED 的布局和接线图。

RGB 为三色一体 LED，在下图黄色框内。IR 红外 LED 有 2 个在摄像头的两旁，下图蓝色框内。

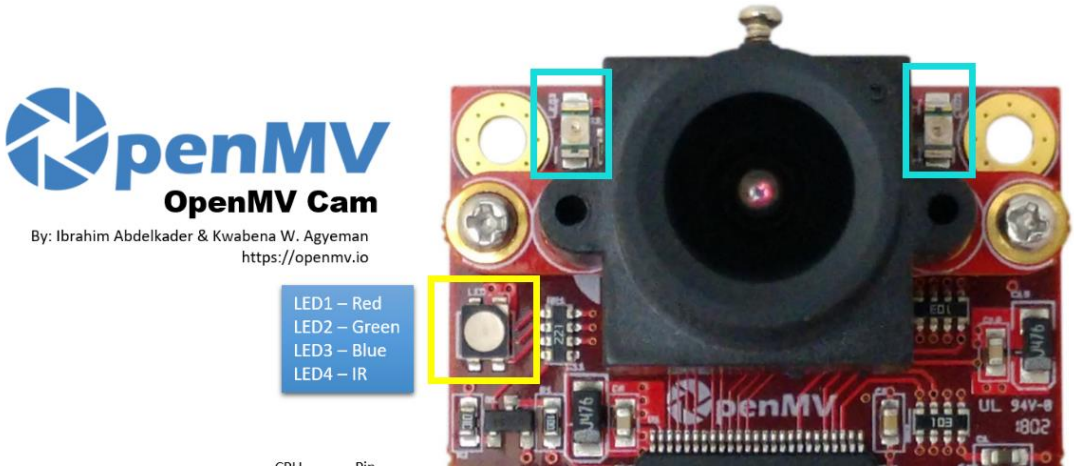


图 23-2 LED 布局

从下面原理图可以看到，LED 的正极都是接在 VCC（3.3V），因此当 IO 为低电平点亮。

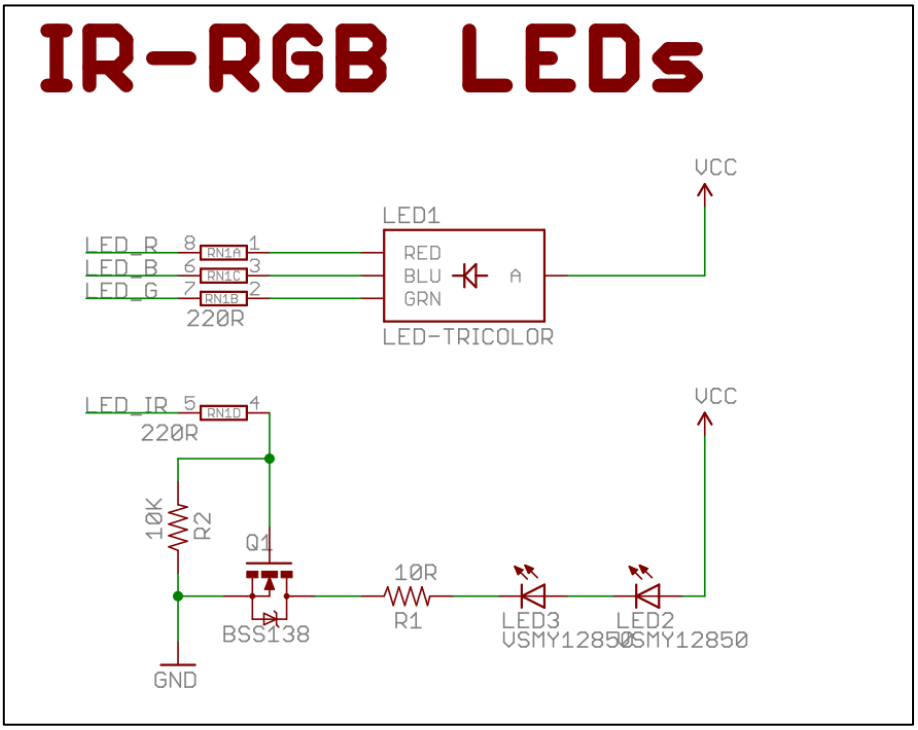


图 23-3 LED 原理图

上表对 MicroPython 的 LED 对象做了详细的说明，pyb 是大模块，LED 是 pyb 下面的其中一个小模块，在 python 编程里有两种方式引用相关模块:

方式 1 是：import pyb，然后通过 pyb.LED 来操作；

方式 2 是：from pyb import LED,意思是直接从 pyb 中引入 LED 模块，然后直接通过 LED 来操作。显然方式 2 会显得更直观和方便，本实验也是使用方式 2 来编程。代码编写流程如下：

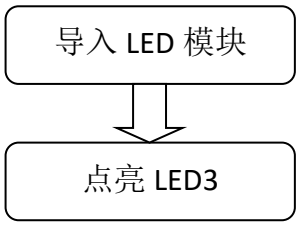


图 23-4 代码编写流程

参考代码：

```
'''
实验名称：点亮 LED(3)蓝灯
版本：v1.0
日期：2019.9
作者：01Studio
实验目的：学习 led 点亮。
'''

from pyb import LED  #导入 LED 模块
LED(3).on()          #点亮 LED3，蓝灯
```

- **实验结果：**

将以上代码文件 main.py 拷贝到 OpenMV4 文件系统，等待红色灯从亮到灭（烧写完成）。按下开发板复位键重新启动。可以看到 LED 蓝灯被点亮。

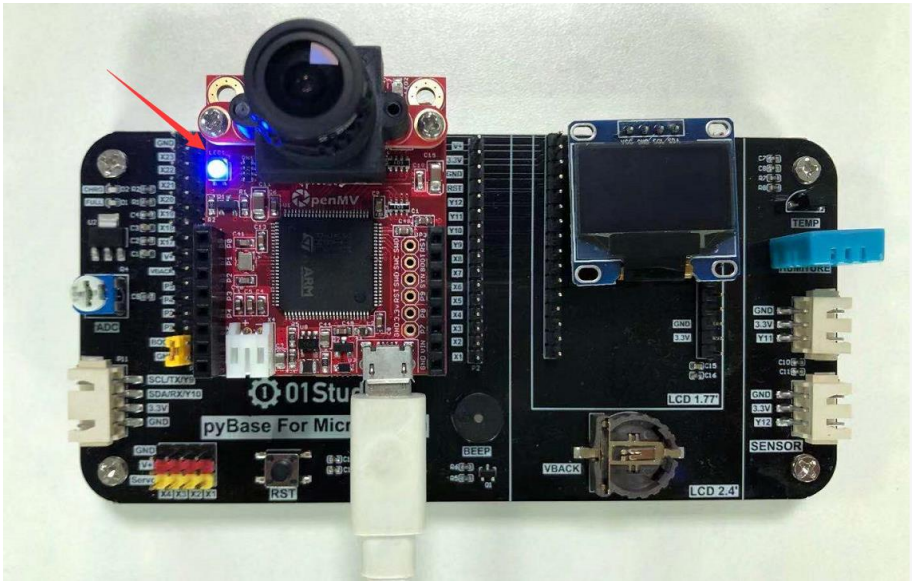


图 23-5 点亮 LED3（蓝灯）

注意：上述代码无法直接通过 OpenMV ID 来运行，因为里面没有 while True() 死循环。

- **总结：**

从第一个实验我们可以看到，使用 MicroPython 来开发关键是要学会构造函数和其使用方法，便可完成对相关对象的操作，在强大的模块函数支持下，实验只用了简单的两行代码便实现了点亮 LED 灯。

23.2 流水灯

- **前言：**

通过上一节点亮 LED 灯的学习，我们已经对 OpenMV4 使用 micropython 的编程有了初步的了解，这一节来做一个功能稍微复杂一点的实验，流水灯。流水灯也叫跑马灯，也就是让几个 LED 来回亮灭，达到好像流水的效果。也是单片机开发学习的典型例子。

- **实验平台：**

pyAI-OpenMV4。



图 23-6 pyAI-OpenMV4

- **实验目的：**

流水灯。让 LED2-LED3 循环亮灭，达到像流水一样的效果。

- **实验讲解：**

OpenMV4 上总共有 4 个 LED，分别是 LED1(红色)、LED2(绿色)、LED3(蓝色)、LED4(IR 红外灯)；控制 LED 使用到 LED 对象。上一章节我们已经学习过 LED 点亮，这里要实现固定时间来亮灭，需要用到 utime 模块中的延时的函数。具体如下：

构造函数
utime()
时间模块，直接使用。
使用方法
`utime.sleep_ms(ms)`
毫秒级延时。`ms`：延时毫秒数。
`utime.sleep_us(us)`
微秒级延时。`us`：延时微秒数。
*更多用法请阅读 openmv 官方文档： 文档链接：http://docs.openmv.io/library/utime.html

表 23-2　utime 时间对象

知道了延时函数的使用方法后，我们可以简单的梳理一下流程，首先导入 LED 和 utime 模块，程序开始先让 LED2-LED3 灭掉，开启循环，依次点亮每个 LED，延时 1 秒，关闭 LED。流程如下：

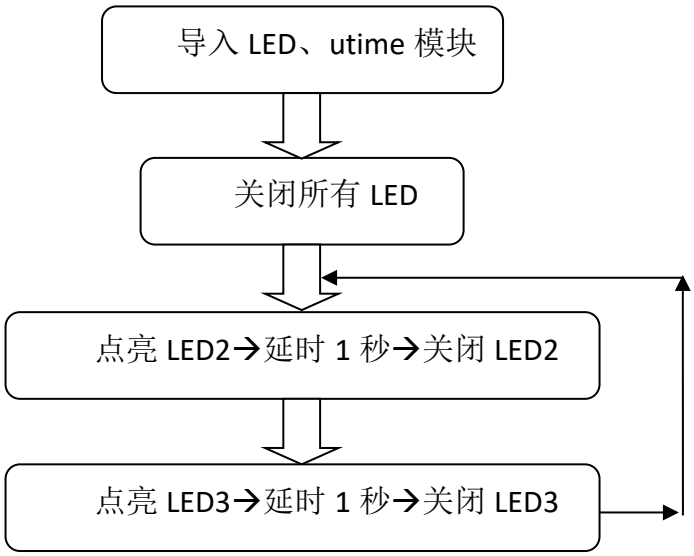


图 23-7 代码编写流程

以下是本实验参考程序代码：

```
'''
实验名称：流水灯
版本：v1.0
日期：2019.9
作者：01Studio
'''

from pyb import LED  #从 pyb 导入 LED 模块
import utime

#关闭全部 LED
LED(2).off()
LED(3).off()

#while True 表示一直循环
while True:
    #LED2 亮 1 秒
    LED(2).on()
    utime.sleep(1)
    LED(2).off()

    #LED3 亮 1 秒
    LED(3).on()
    utime.sleep(1)
    LED(3).off()
```

上述代码没错是完整的按照编程思路来编写，但可以见到有很多格式相似的地方，这显得代码非常冗余。我们可以通过 for 函数来编写程序，由于是对 LED2、

LED3 的操作，因此我们可以用 for i in range(2,4): 语句来修改。参考代码如下：

参考代码：

```
'''
实验名称：流水灯
版本：v2.0
日期：2019.9
作者：01Studio
'''

from pyb import LED  #从 pyb 导入 LED 模块
import utime

# 相当于 for i in [2, 3]，LED(i).off()执行 2 次，分别是 LED 2，3
for i in range(2,4):
    LED(i).off()

while True:
    #使用 for 循环
    for i in range(2,4):
        LED(i).on()
        utime.sleep(1) #延时 1 秒
        LED(i).off()
```

- **实验结果：**

将上述代码对应的 main.py 复制到文件系统，或者直接在 IDE 中运行，可以看到 LED2 和 LED3 也就是绿灯和蓝灯交替闪烁，像流水的效果。

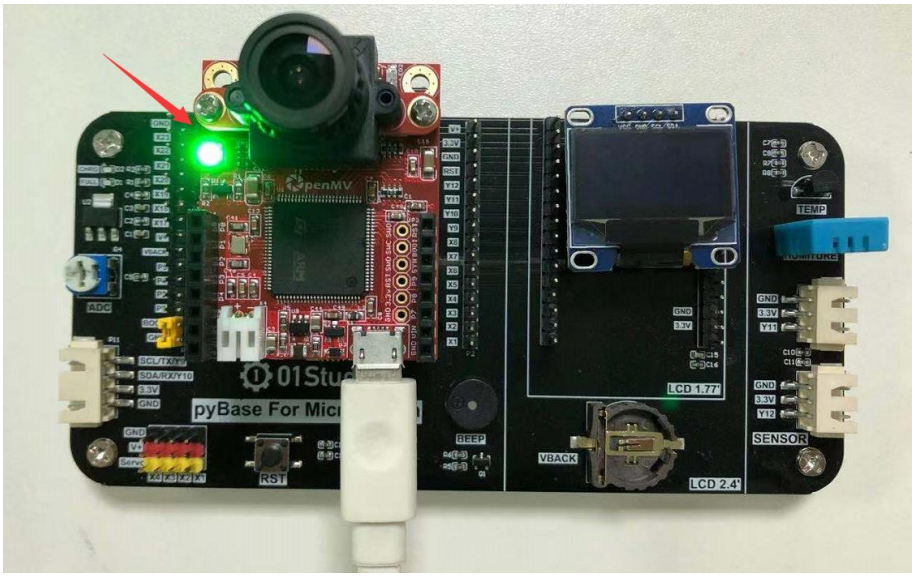


图 23-8

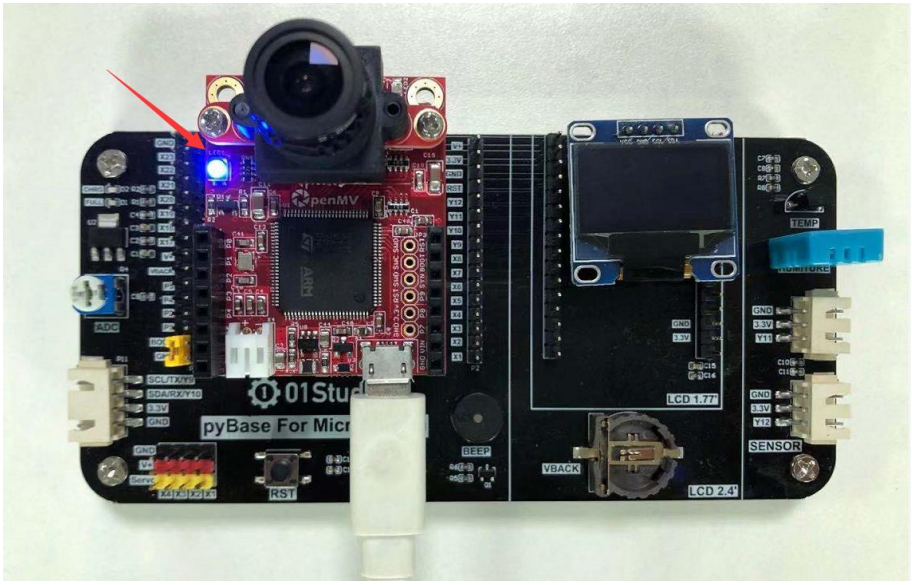


图 23-9

- **总结：**

本节实验相对于第一节相比增加了延时函数的应用，可见 MicroPython 编程功能增加就是各类函数的叠加，当开发复杂功能时候，因为底层函数已经封装好，所以面向应用的开发使用起来非常方便。除此之外我们也体验了 Python 代码的简洁和高效，使得程序更好的阅读。

23.3 GPIO（按键）

- **前言：**

按键是最简单也最常见的输入设备，很多产品都离不开按键，包括早期的 iphone，今天我们就来学习一下如何使用 MicroPython 来编写按键程序。有了按键输入功能，我们就可以做很多好玩的东西了。

- **实验平台：**

pyAI-OpenMV4 开发板。



图 23-10 pyAI-OpenMV4

- **实验目的：**

学会使用微处理器的 GPIO 的输入/输出，使用 GPIO 方式来控制 LED 和按键。

- **实验讲解：**

OpenMV4 上引出了 P0-P9 共 10 个 GPIO（General Purpose Input/Output，通用输入输出口），基本上每个 GPIO 口都可以配置成特定的 GPIO 方式来进行应用。我们先来了解一下 GPIO（Pin 对象）的构造函数和使用方法：

构造函数
pyb.Pin(id,mode,pull_mode)
Pin 对象在 pyb 模块下。 【id】引脚号，如“P0”，“P1”； 【mode】输入输出模式选择 Pin.IN 输入模式， Pin.OUT_PP 输出带推挽； 【pull_mode】上下拉电阻配置 Pin.PULL_UP 上拉电阻， Pin.PULL_DOWN 下拉电阻， Pin.PULL_NONE 没上下拉电阻； ***例： KEY = Pin('P9', Pin.IN, Pin.PULL_UP) #将按键"P9"配置为输入方式***
使用方法
`Pin.high()`
引脚输出高电平。
`Pin.low()`
引脚输出低电平。
`Pin.value()`
获取当前引脚的输入电平，返回 `0`（低电平）或者 `1`（高电平）。仅在引脚配置为输入模式有效。

表 23-3 Pin 对象（GPIO）

我们来看看 pyAI-OpenMV4 的原理图，看看按键连接的引脚为 P9。

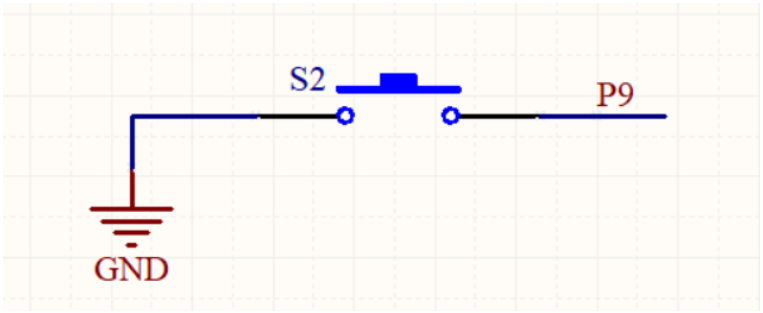


图 23-11 按键接线图

GPIO 对象使用非常简单，我们将按键即“P9”引脚配置成输入，实现当检测到按键被按下时候点亮 LED（2），松开时关闭 LED（2）。代码编写流程如下：

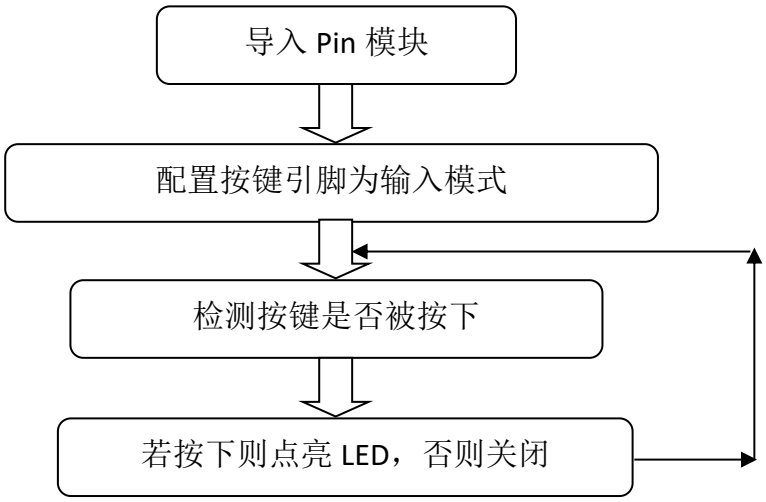


图 23-12 代码编写流程

参考代码：

```
'''
实验名称：GPIO（按键）
版本：v1.0
日期：2019.9
作者：01Studio
社区：www.01studio.org
'''

from pyb import Pin,LED



#将 KEY 按键"P9"配置为输入方式
KEY = Pin('P9', Pin.IN, Pin.PULL_UP)
```

```
while True:


    if KEY.value()==0: #按键被按下接地
        LED(3).on()    #点亮 LED（3）蓝灯


    else:
        LED(3).off()     #关闭 LED（3）蓝灯
```

- **实验结果：**

运行代码，可以看到当按键被按下时候，LED(3)蓝灯点亮，松开时熄灭。按键可以按下 pyAI-OpenMV4 顶部的按键或者 pyBase 开发底板上的 KEY 按键。

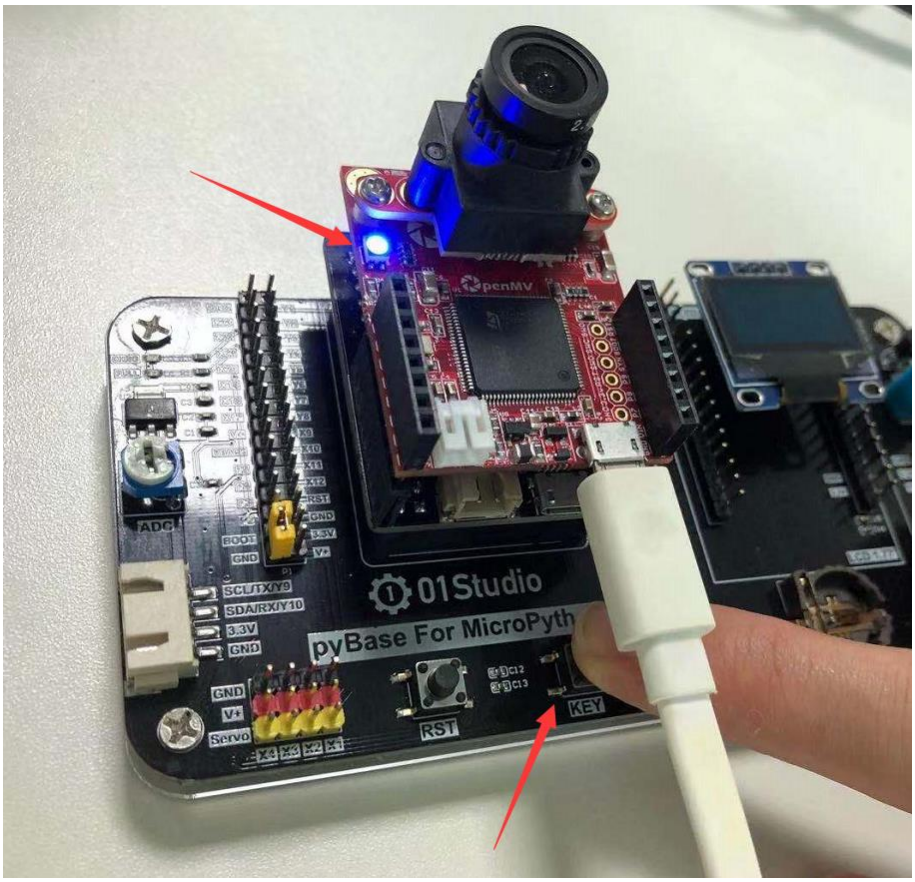


图 23-13　GPIO 按键实验

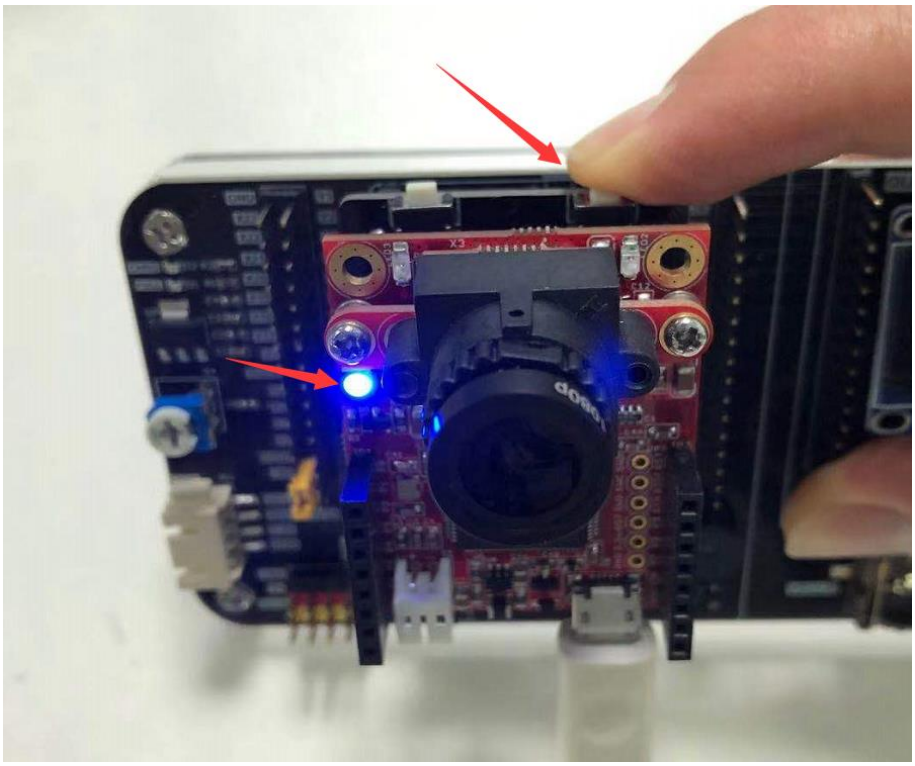


图 23-14 上方的按键

- **总结：**

GPIO 是非常通用的实验和功能，学会了 GPIO，就可以把开发板所有的引脚为自己所用，灵活性很强。

23.4 外部中断

- **前言：**

前面我们在做普通的 GPIO 时候，虽然能实现 IO 口输入输出功能，但代码是一直在检测 IO 输入口的变化，因此效率不高，特别是在一些特定的场合，比如某个按键，可能 1 天才按下一次去执行相关功能，这样我们就浪费大量时间来实时检测按键的情况。

为了解决这样的问题，我们引入外部中断概念，顾名思义，就是当按键被按下(产生中断)时，我们才去执行相关功能。中断的应用非常普遍，而 OpenMV4 上基本每个 IO 口都具备中断功能。

- **实验平台：**

pyAI-OpenMV4。



图 23-15 pyAI-OpenMV4

- **实验目的：**

利用中断方式来检查按键 KEY 状态，被按键被按下（产生外部中断）后使 LED（2）蓝灯的亮灭状态反转。

- **实验讲解：**

从上一节 GPIO 实验我们知道按键是连接到 P9 引脚，我们先来看看外部中断的用法。

构造函数
pyb.ExtInt(pin,mode,pull_mode,callback)
外部中断 ExtInt 对象在 pyb 模块下。 【pin】引脚号，如“P0”，“P1”； 【mode】中断触发方式 ExtInt.IRQ_RISING：上升沿触发 ExtInt.IRQ_FALLING：下降沿触发 ExtInt.IRQ_RISING_FALLING：上升或下降沿触发 【pull_mode】上下拉电阻设置 Pin.PULL_NONE：无上下拉电阻； Pin.PULL_UP：启用上拉电阻； Pin.PULL_DOWN：启用下拉电阻； 【callback】中断后执行的回调函数。

表 23-4 外部中断对象

我们先来了解一下上升沿和下降沿的概念，由于按键 KEY 引脚是通过按键接到 GND，也就是我们所说的低电平“0”，所以当按键被按下再松开时，引脚先获得下降沿，再获得上升沿，如下图所示：

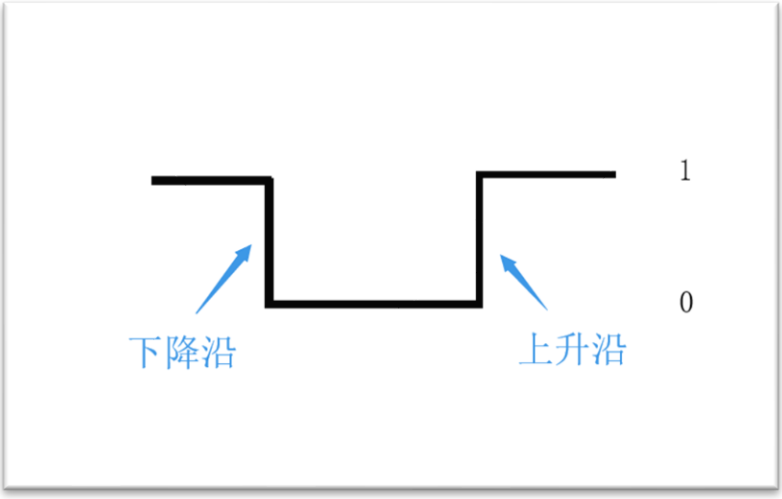


图 23-16 上升沿和下降沿

由此可见，我们可以选择下降沿方式触发外部中断，也就是当按键被按下的时候立即产生中断。

编程思路中断跟 GPIO 按键章节类似，在初始化中断后，当系统检测到外部中断时候，执行 LED 状态反转的代码即可。

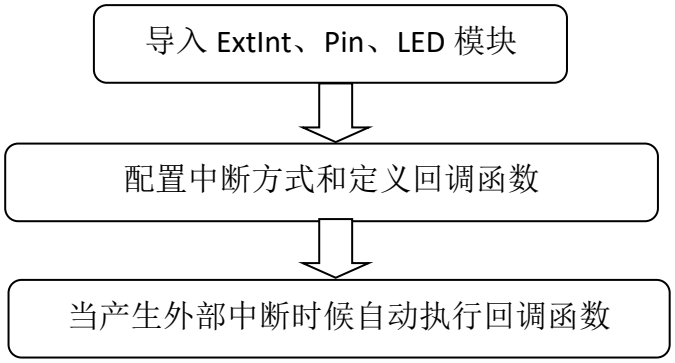


图 23-17 代码编写流程

参考代码：

```
'''
实验名称：外部中断
版本：v1.0
日期：2019.9
作者：01Studio
社区：www.01studio.org
'''

from pyb import Pin,ExtInt,LED

callback = lambda e: LED(3).toggle()
#下降沿触发，打开上拉电阻
ext = ExtInt(Pin('P9'), ExtInt.IRQ_FALLING, Pin.PULL_UP, callback)
```

- **实验结果：**

烧录程序，可以看到每次按下按键时候，LED3 蓝灯的亮灭状态翻转。按键可以使用 pyAI-OpenMV4 顶部的按键或者 pyBase 开发底板上的 KEY 按键。**（注：中断无法通过 IDE 运行来执行。）**

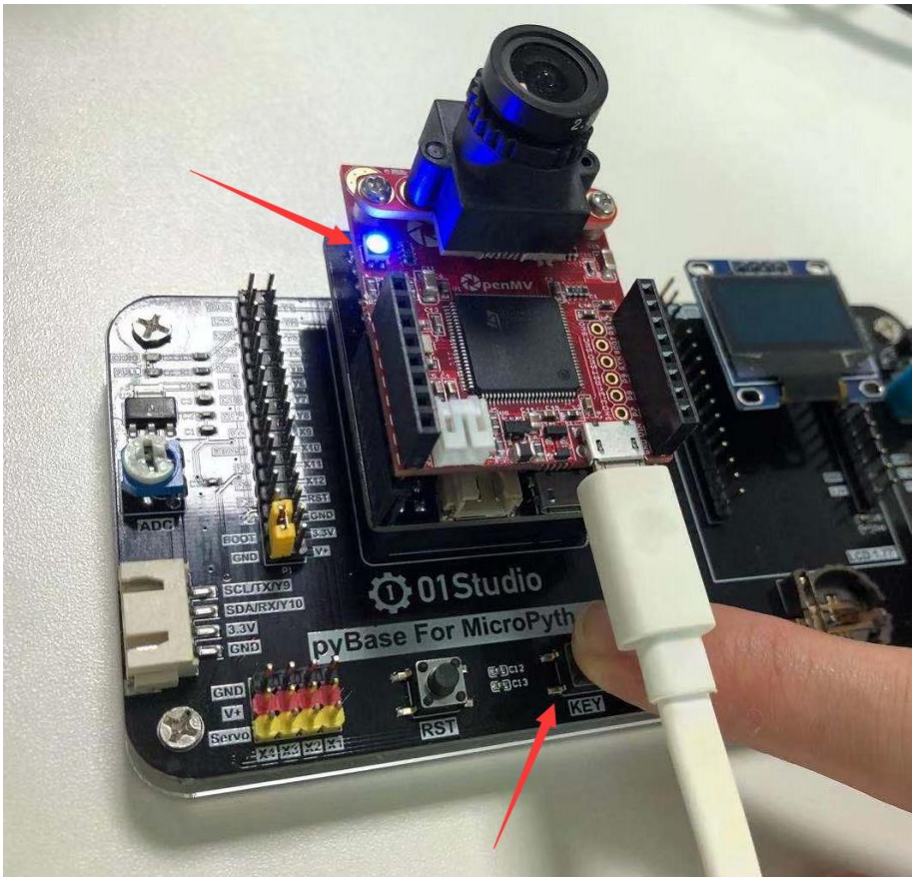


图 23-18　GPIO 按键实验

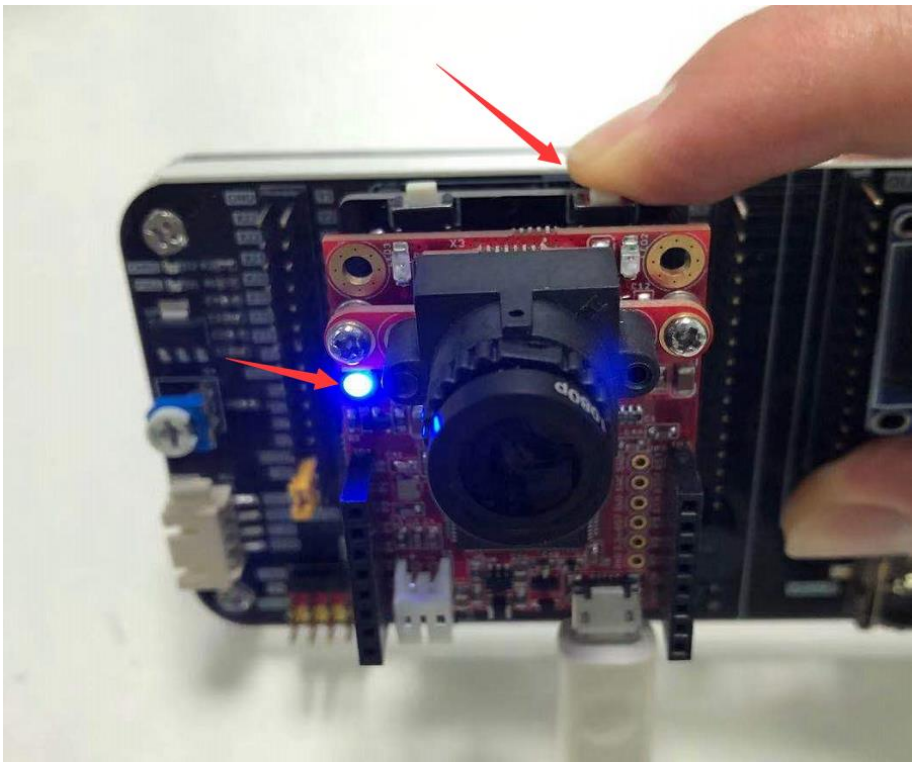


图 23-19 上方的按键

- **总结：**

从参考代码来看，只是用了 3 行代码就实现了实验功能，而且相对于使用 while True 实时检测函数来看，代码的效率大大增强。外部中断的应用非常广，除了普通的按键输入和电平检测外，很大一部分输入设备，比如传感器也是通过外部中断方式来实时检测，从而提供 CPU 的利用率。

23.5 定时器

- **前言：**

定时器，顾名思义就是用来计时的，我们常常会设定计时或闹钟，然后时间到了就告诉我们要做什么了。单片机也是这样，通过定时器可以完成各种预设好的任务。

- **实验平台：**

pyAI-OpenMV4。



图 23-20 pyAI-OpenMV4

- **实验目的：**

通过定时器让 LED(3)蓝灯周期性每秒闪烁 1 次。

- **实验讲解：**

定时器模块 Timer 在 pyb 模块中。对象说明如下：

构造函数
pyb.Timer(id,freq,……)
定时器对象 Timer 对象在 pyb 模块下。

【id】定时器编号，1-14; 【freq】定时器中断频率
使用方法
Timer.callback(fun)
定义执行的回调函数。
Timer.deinit()
终止定时器。
*更多用法请阅读 openmv 官方文档： 文档链接：http://docs.openmv.io/library/pyb.Timer.html

表 23-5　定时器对象

定时器到了预设指定时间后，也会产生中断，因此跟外部中断的编程方式类似，代码编程流程图如下：

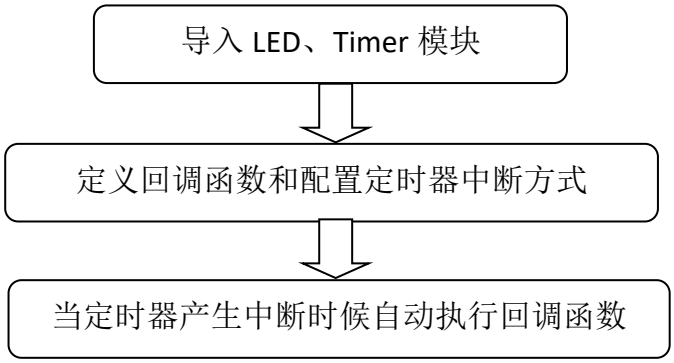


图 23-21 代码编写流程

参考代码如下：

```
'''
实验名称：定时器（蓝灯周期性闪烁）
版本：v1.0
日期：2019.9
作者：01Studio
'''
```

```
import pyb

tim = pyb.Timer(4,freq=1)       # 使用定时器 4 创建定时器对象,频率 1Hz
#定时器中断回调函数，执行 LED（3）蓝灯状态反转
tim.callback(lambda t:pyb.LED(3).toggle())
```

- **实验结果：**

烧录程序，可以看到 LED(2)蓝灯每隔 1 秒闪烁 1 次。**（注：中断无法通过 IDE 运行来执行。）**

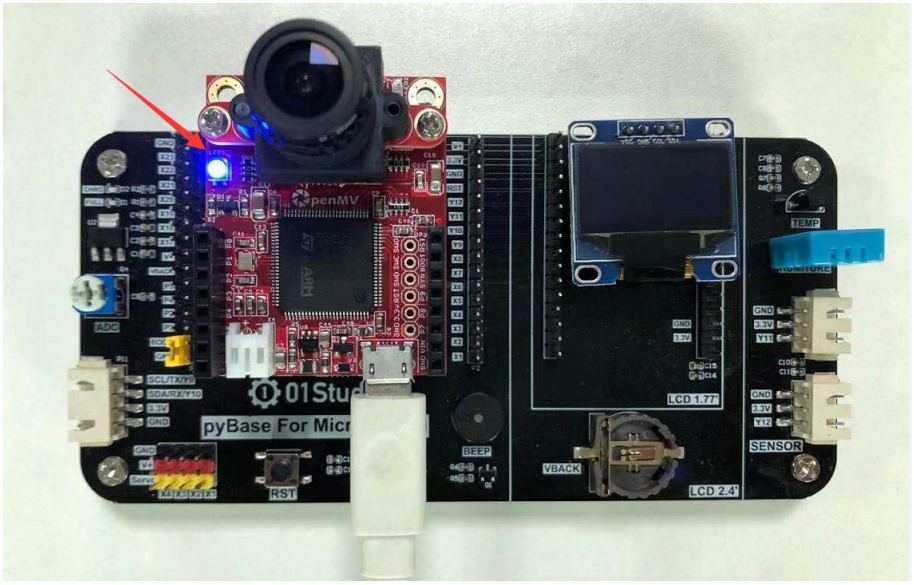


图 23-22

- **总结：**

本节实验介绍了定时器的使用方式，有用户可能会认为使用延时延时也可以实现这个功能，但相比于延时函数，定时器的好处就是不占用过多的 CPU 资源。

有兴趣的用户也可以多定义几个定时器对象 tim2,tim3，通过不同的参数配置实现多任务的操作。

23.6 I2C 总线（OLED 显示屏）

- **前言：**

在上一节学习了按键输入设备后，我们这一节先来学习输出设备 OLED 显示屏，其实之前的 LED 灯也算是输出设备，因为它们确切地告诉了我们硬件的状态。只是相对于只有亮灭的 LED 而言，显示屏可以显示更多的信息，体验更好。

本章节的 OLED 显示屏学习，实际上是在使用 I2C 的总线接口，pyAI-OpenMV4 是通过 I2C 总线与 OLED 显示屏通讯的。这章稍微复杂一点，我们把它放在了前面来学习，是因为学会了显示屏的使用，那么在后面的实验中可玩性就更强了。

- **实验平台：**

pyAI-OpenMV4 开发套件。

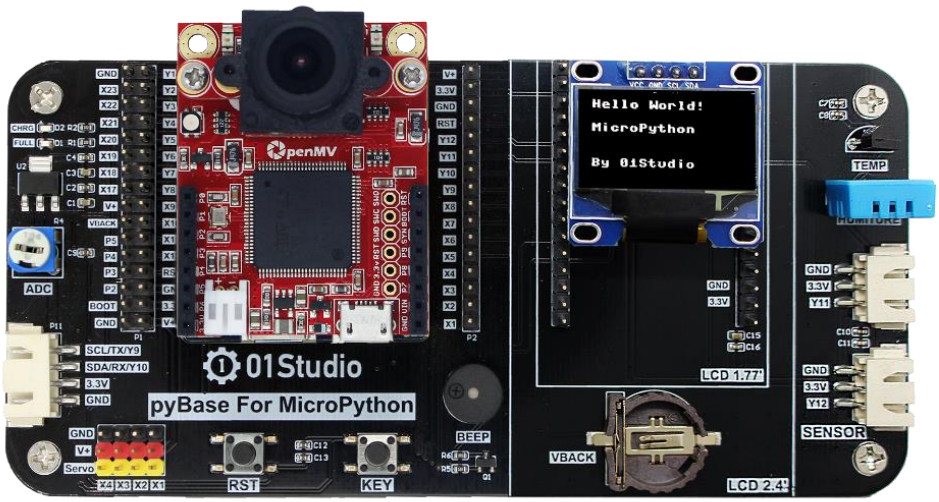


图 23-23 pyAI-OpenMV4 开发套件

- **实验目的：**

学习使用 MicroPython 的 I2C 总线通讯编程和 OLED 显示屏的使用。

- **实验讲解：**

什么是 I2C?

I2C 是用于设备之间通信的双线协议，在物理层面，它由 2 条线组成：SCL 和 SDA，分别是时钟线和数据线。也就是说不通设备间通过这两根线就可以进行通信。

什么是 OLED 显示屏？

OLED 的特性是自己发光，不像 TFT LCD 需要背光，因此可视度和亮度均高，其次是电压需求低且省电效率高，加上反应快、重量轻、厚度薄，构造简单，成本低等特点。简单来说跟传统液晶的区别就是里面像素的材料是由一个个发光二极管组成，因为密度不高导致像素分辨率低，所以早期一般用作户外 LED 广告牌。随着技术的成熟，使得集成度越来越高。小屏也可以制作出较高的分辨率。



图 23-24 I2C 接口的 OLED

在了解完 I2C 和 OLED 显示屏后，我们先来看看开发板的原理图，也就是 MicroPython 上的 OLED 接口是如何连线的。下图是 pyBase 开发底板的原理图。

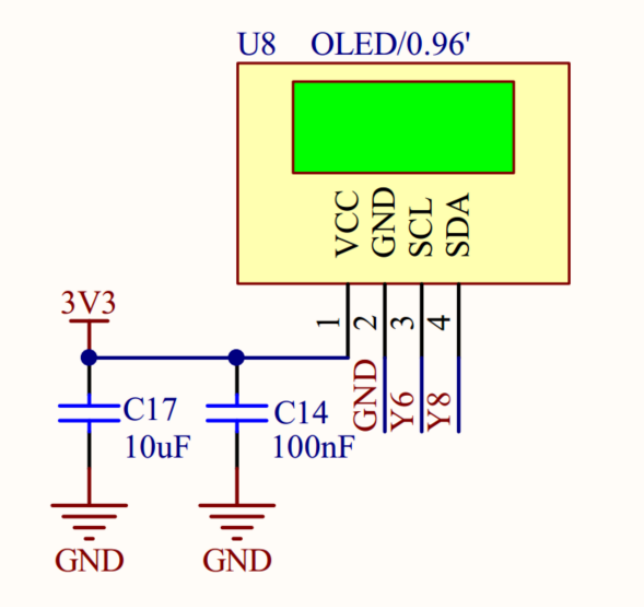


图 23-25 OLED 接口原理图

我们再来看看 pyAI-OpenMV4 转接板的原理图接口部分。

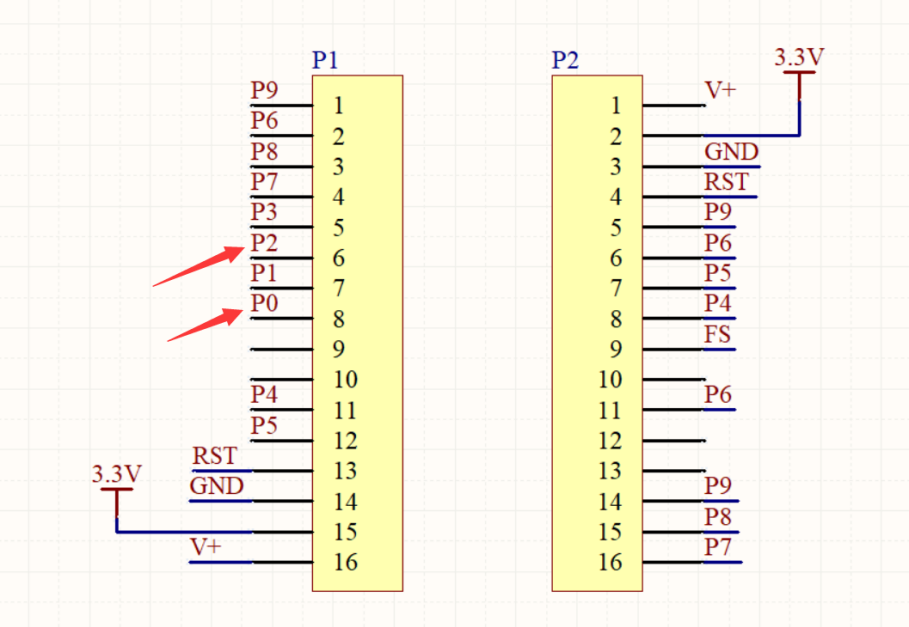


表 23-6　pyAI-OpenMV4 转接板接口图

结合以上可以得知 pyBase 底板连接到 OLED 的对应关系是 Y6→SCL 和 Y8→SDA。对应 OpenMV4 的关系是：P2→Y6→SCL，P0→Y8→SDA。尽管没有连接到自带的 I2C 接口，我们也可以通过软件模拟 I2C 方式来实现 OLED 的数据通信和使用。本例程将使用 MicroPython 的 Machine 模块来定义 Pin 口和 I2C 初始化。具体如下：

构造函数
i2c = machine.I2C(scl,sda,freq)
构建 I2C 对象。 【scl】时钟引脚； 【sda】数据引脚； 【freq】通信频率，即速度。
使用方法
i2c.scan()
扫描 I2C 总线的设备。返回地址，如：0x3c；

i2c.readfrom(addr,nbytes)
从指定地址读数据。addr:指定设备地址；nbytes:读取字节数；
i2c.write(buf)
写数据。buf:数据内容；
*其它更多用法请阅读官方文档： 文档链接：http://docs.openmv.io/library/pyb.I2C.html

表 23-7 I2C 对象

定义好 I2C 后，还需要驱动一下 OLED。这里我们已经写好了 OLED 的库函数，在 ssd1306x.py 文件里面。开发者只需要将改 py 文件拷贝到 OpenMV4 文件系统里面，然后在 main.py 里面调用函数即可。人生苦短，我们学会调用函数即可，也就是注重顶层的应用，想深入的小伙伴也可以自行研究 ssd1306.py 文件代码。OLED 显示屏对象介绍如下：

构造函数
oled = SSD1306_I2C(width, height, i2c, addr)
构 OLED 显示屏对象。 【width】屏幕宽像素； 【height】 屏幕高像素； 【i2c】定义好的 I2C 对象; 【addr】显示屏设备地址。
使用方法
oled.text(string,x,y)
将 string 字符写在指定为位置。string：字符；x:横坐标；y:纵坐标。
oled.show()
执行显示。
oled.fill(RGB)
清屏。RGB：0 表示黑色，1 表示白色。

表 23-8 OLED 对象

学习了 I2C、OLED 对象用法后我们通过编程流程图来理顺一下思路：

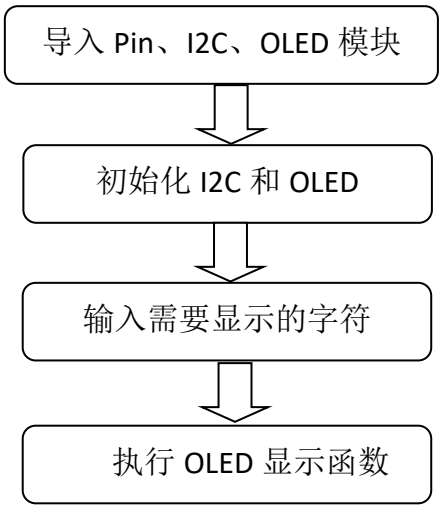


图 23-26 代码编写流程

参考代码如下：

```
'''
实验名称：OLED 显示屏（I2C 总线）
版本：v1.0
日期：2019.9
作者：01Studio
'''

from machine import I2C,Pin        #从 machine 模块导入 I2C、Pin 子模块
from ssd1306x import SSD1306_I2C   #从 ssd1306 模块中导入 SSD1306_I2C 子模块

#I2C 初始化：sda--> P0, scl --> P2,频率 8MHz
i2c = I2C(sda=Pin("P0"), scl=Pin("P2"),freq=80000 )
#OLED 显示屏初始化：128*64 分辨率,OLED 的 I2C 地址是 0x3c
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

oled.text("Hello World!", 0,  0)      #写入第 1 行内容
oled.text("MicroPython",  0, 20)      #写入第 2 行内容
```

```
oled.text("By 01Studio",  0, 50)      #写入第 3 行内容


oled.show()   #OLED 执行显示
```

上述代码中 OLED 的 I2C 地址是 0x3C,不同厂家的产品地址可能预设不一样，具体参考厂家的说明书。或者也可以通过 I2C.scan()来获取设备地址。

另外记得将我们提供的示例代码中的 ssd1306x.py 驱动文件拷贝到 OpenMV4 的文件系统下，跟 main.py 保持同一个路径。

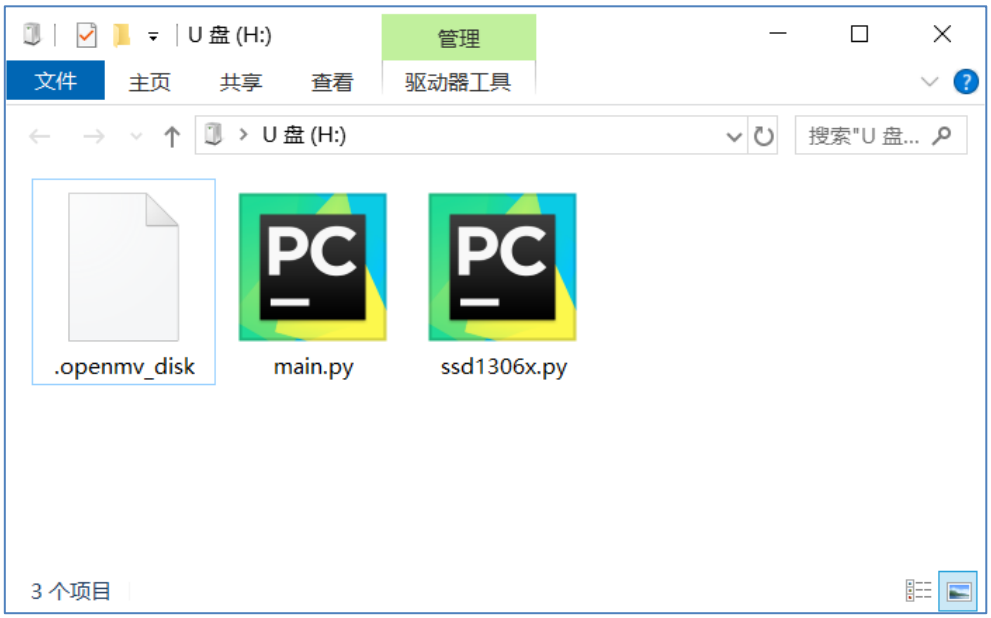


图 23-27 将 main.py、ssd1306x.py 拷贝到文件系统

- **实验结果：**

运行代码，可以看到实验现象如下：

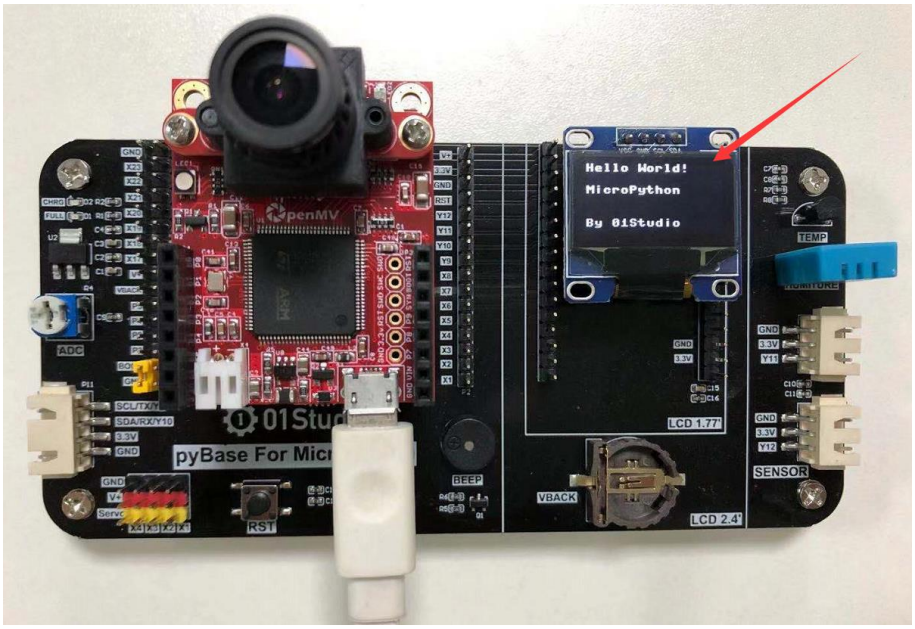


图 23-28 实验结果

- **总结：**

这一节我们学会了驱动 OLED 显示屏，换着以往如果从使用单片机从 0 开发的话你需要了解 I2C 总线原理，了解 OLED 显示屏的使用手册，编程 I2C 代码，有经验的嵌入式工程师搞不好也要弄个几天。现在基本半个小时解决问题。当然前提是别人已经给你搭好桥了，有了强大的底层驱动代码支持，我们只做好应用就好。

这一节学习的意义不仅在完成实验。在学习完 OLED 显示屏实验后，接下来我们的实验都可以使用这个 OLED 来跟用户交互了，这大大提高了实验的可观性。

23.7 RTC 实时时钟

- **前言：**

时钟可以说我们日常最常用的东西了，手表、电脑、手机等等无时无刻不显示当前的时间。

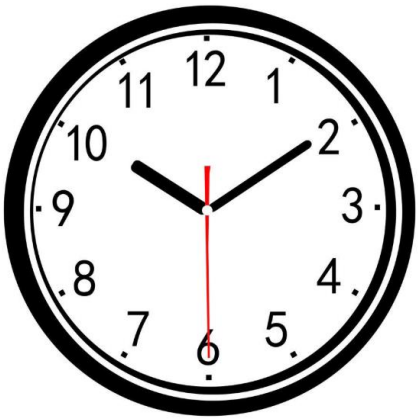


图 23-29 时钟

可以说每一个电子爱好者心中都希望拥有属于自己制作的一个电子时钟，接下来我们就用 MicroPython 开发板来制作一个属于自己的电子时钟。

- **实验平台：**

pyAI-OpenMV4 开发套件。

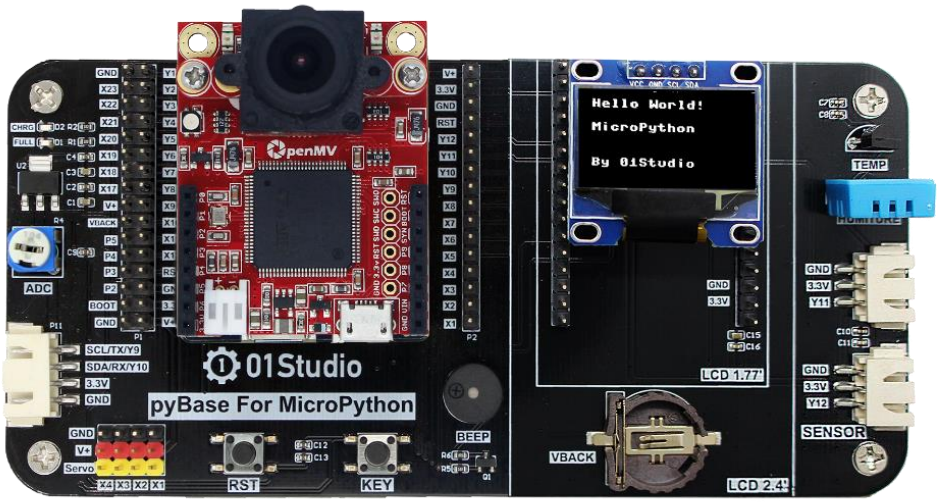


图 23-30 pyAI-OpenMV4 开发套件

- **实验目的：**

学习 RTC 编程和制作电子时钟，使用 OLED 显示。

- **实验讲解：**

实验的原理是读取 RTC 数据，然后通过 OLED 显示。毫无疑问，强大的 pyBoard 已经集成了内置时钟函数模块。具体如下：

构造函数
pyb.RTC()
RTC()是内置的实时时钟函数模块。
使用方法
RTC().datetime((2019,4,1,1,0,0,0,0))
设置日期时间： 顺序分别是年，月，日，星期，时，分，秒，次级秒。 星期：1-7 表示周一至周日； 次级秒：255-0 倒着计数。
RTC().datetime()
获取当前日期时间数据；

表 23-9　RCT 实时时钟对象

从上表可以看到 RTC（）的使用方法，我们需要做的就是先设定时间，然后再获取当前芯片里的时间，通过 OLED 显示屏显示，如此循环。在循环里，如果一直获取日期时间数据会造成资源浪费，所以可以每隔第一段时间获取一次数据，又由于肉眼需要看到至少每秒刷新一次即可，这里每隔 300ms 获取一次数据，所以具体流程如下：

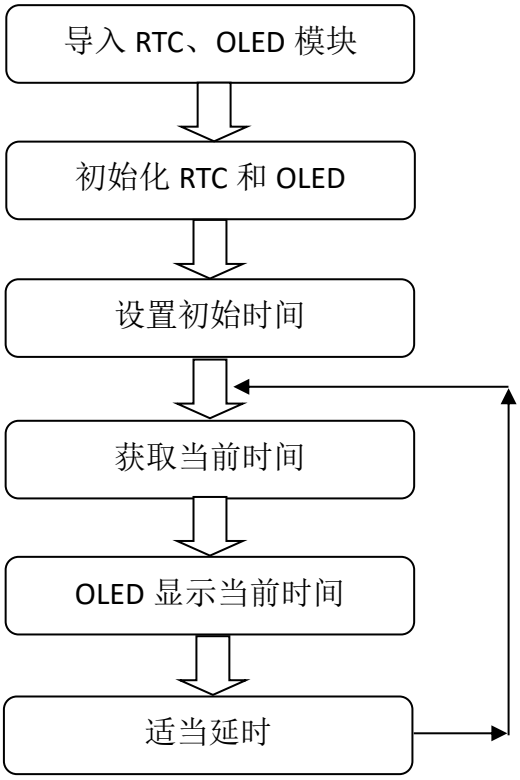


图 23-31 代码编写流程

参考代码如下：

```
'''
实验名称：RTC 实时时钟
版本：v1.0
日期：2019.9
作者：01Studio
'''

# 导入相关模块
from pyb import RTC
from machine import Pin, I2C
from ssd1306x import SSD1306_I2C
import utime
```

```
# 定义星期和时间（时分秒）显示字符列表
week = ['Mon', 'Tues', 'Wed', 'Thur', 'Fri', 'Sat', 'Sun']
time = ['', '', '']

# 初始化所有相关对象
i2c = I2C(sda=Pin("P0"), scl=Pin("P2"), freq=80000) #频率 8MHz
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
rtc = RTC()

# 首次上电配置时间，按顺序分别是：年，月，日，星期，时，分，秒，次秒级；这里做
#了一个简单的判断，检查到当前年份不对就修改当前时间，用户可以根据自己实际情况来
# 修改。
if rtc.datetime()[0] != 2019:
    rtc.datetime((2019, 4, 1, 1, 0, 0, 0, 0))

while True:

    datetime = rtc.datetime()  # 获取当前时间

    oled.fill(0)  # 清屏显示黑色背景
    oled.text('01Studio', 0, 0)    # 首行显示 01Studio
    oled.text('RTC Clock', 0, 15)  # 次行显示实验名称

    # 显示日期，字符串可以直接用“+”来连接
    oled.text(str(datetime[0]) + '-' + str(datetime[1]) + '-' +
            str(datetime[2]) + ' ' + week[(datetime[3] - 1)], 0, 40)

  # 显示时间需要判断时、分、秒的值否小于 10，如果小于 10，则在显示前面补“0”以达
  # 到较佳的显示效果
    for i in range(4, 7):
```

```
        if datetime[i] < 10:
            time[i - 4] = "0"
        else:
            time[i - 4] = ""

    # 显示时间
    oled.text(time[0] + str(datetime[4]) + ':' + time[1] +
str(datetime[5]) + ':' + time[2] + str(datetime[6]), 0, 55)

    oled.show()
    utime.sleep_ms(300)  #延时 300ms
```

- 实验结果：

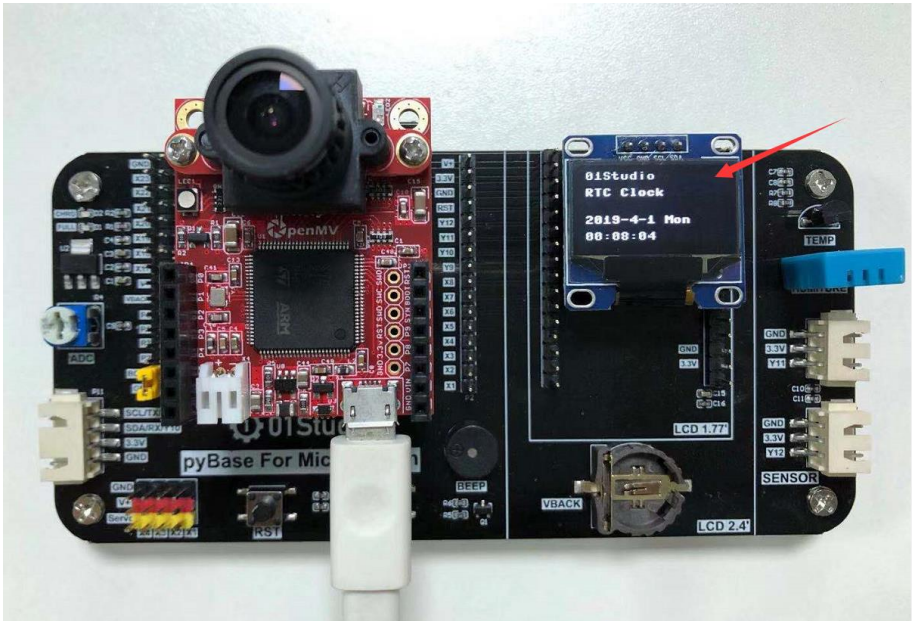


图 23-32 实验结果

由于 OpenMV4 没有引出后备电池引脚，所以不支持掉电保存。因此 pybase 上面的纽扣电池是不起作用的。

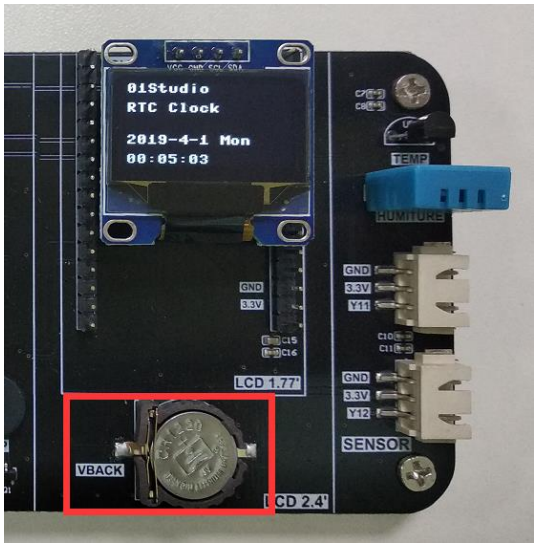


图 23-33 后备电池（VBACK）

- **总结：**

RTC 实时时钟的可玩性很强，我们还可以根据自己的风格来设定数字显示位置，以及加上一些属于自己的字符标识。打造自己的电子时钟。

23.8 ADC

- **前言：**

ADC(analog to digital conversion) 模拟数字转换。意思就是将模拟信号转化成数字信号，由于单片机只能识别二级制数字，所以外界模拟信号常常会通过 ADC 转换成其可以识别的数字信息。常见的应用就是将变化的电压转成数字信号。

- **实验平台：**

pyAI-OpenMV4 开发套件。

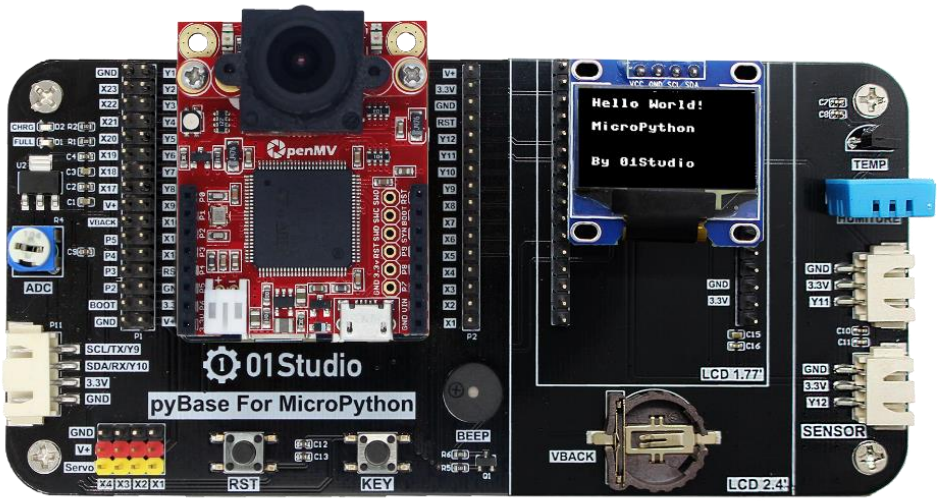


图 23-34 pyAI-OpenMV4 开发套件

- **实验目的：**

通过编程调用 MicroPython 的内置 ADC 函数，实现测量 0-3.3V 电压，并显示到 OLED 屏幕上。

- **实验讲解：**

pyBase 开发底板的的 X7 引脚连接到了电位器，通过电位器的调节可以使得 X7 引脚上的电压变化范围实现从 0-3.3V。

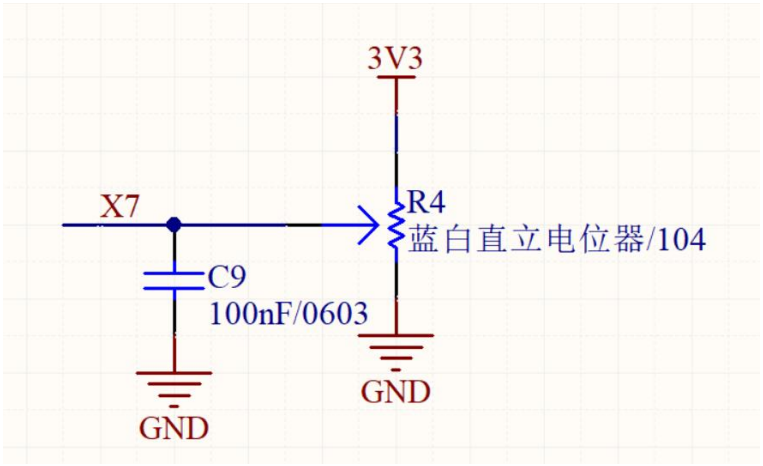


图 23-35 电位器原理图

我们再来看看 pyAI-OpenMV4 的引脚情况，其对用的 X7 是悬空，但 P6 是接在 pyBase 的 X6 上。

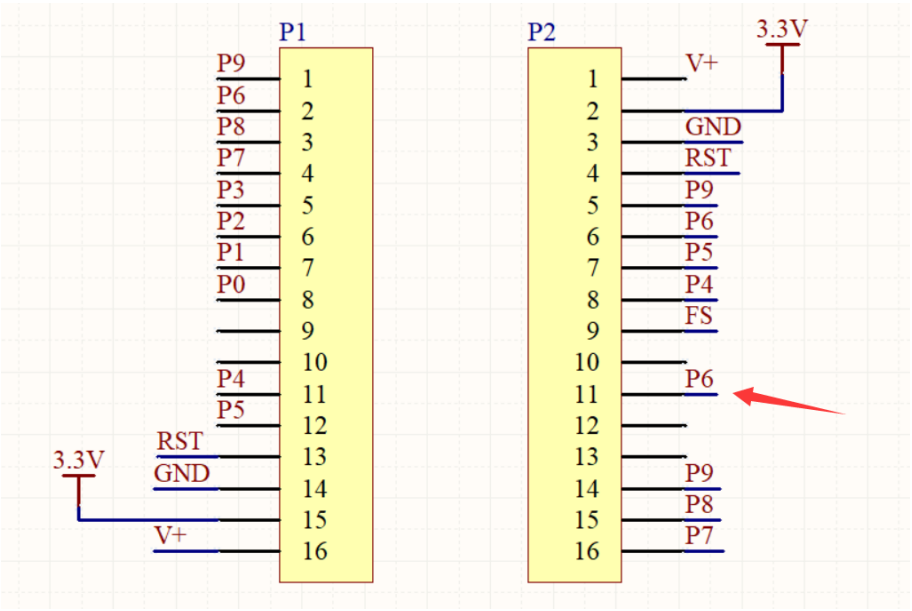


图 23-36 P6 连接到 X6

因此我们可以用一个跳线帽或者跳线将 pyBase 开发底板上的 X6 跟 X7 相连接，从而使用 P6 引脚编程实现 AD 数据采集。

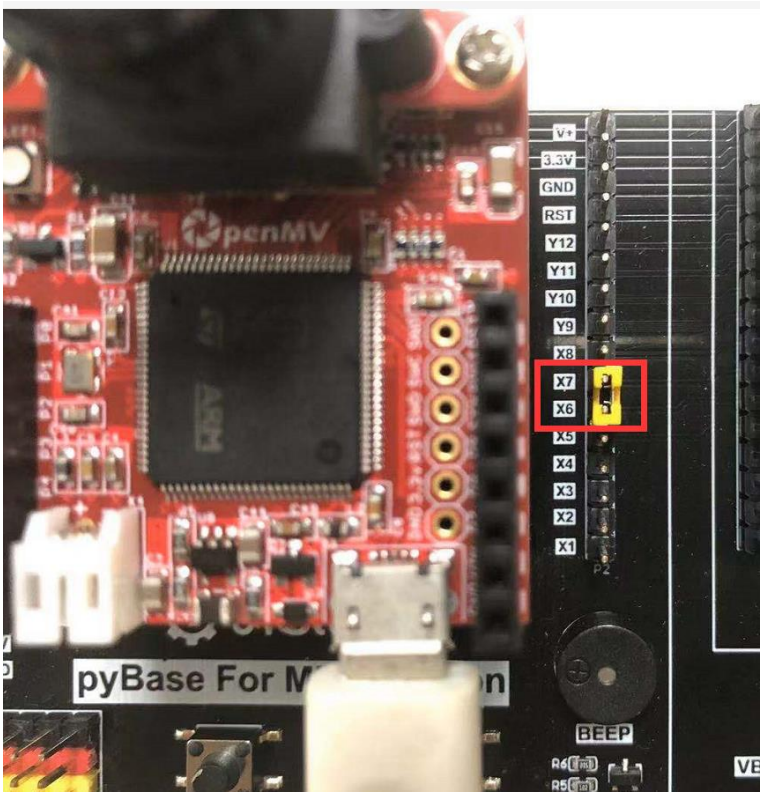


图 23-37 使用跳线帽将 X6 和 X7 短接

在了解了接线后，我们来看看内置 ADC 模块的函数使用方法。

构造函数
pyb.ADC(pin)
ADC 对象在 pyb 模块下。pin：ADC 输入引脚
使用方法
ADC.read()
读取 AD 值，返回 0-4095（0v-3.3V）,自带 ADC 的测量精度是 12 位，2^12=4096

表 23-10 ADC 对象

你没看错，就这么简单。两句函数就可以获得 AD 数值，让我们来理顺一下编程逻辑。先导入相关模块，然后初始化模块。在循环中不断读取 ADC 的值，转化成电压值后在 OLED 上面显示，每隔 1 秒读取一次，具体如下：

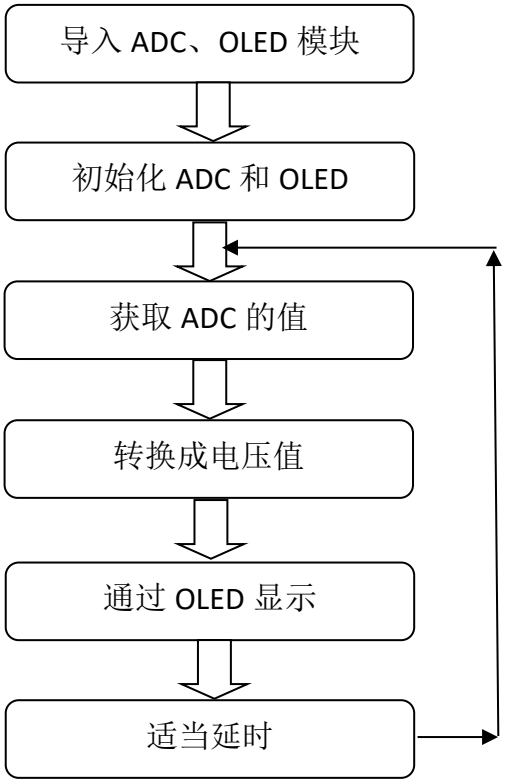


图 23-38 代码编写流程

实验参考代码：

```
'''
实验名称：ADC-电压测量
版本：v1.0
日期：2019.9
作者：01Studio
说明：通过对 ADC 数据采集，转化成电压在显示屏上显示。ADC 精度 12 位，电压 0-
3.3V。
'''

#导入相关模块
import pyb,utime
from machine import Pin,I2C
from ssd1306x import SSD1306_I2C
```

```
#初始化相关模块
i2c = I2C(sda=Pin("P0"), scl=Pin("P2"),freq=80000)
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)
adc = pyb.ADC('P6') #Pin='P6'

while True:

    oled.fill(0)  # 清屏显示黑色背景
    oled.text('01Studio', 0, 0)  # 首行显示 01Studio
    oled.text('ADC', 0, 15)      # 次行显示实验名称

    #获取 ADC 数值
    oled.text(str(adc.read()),0,40)
    oled.text('(4095)',40,40)

    #计算电压值，获得的数据 0-4095 相当于 0-3.3V，（'%.2f'%）表示保留 2 位小数
    oled.text(str('%.2f'%(adc.read()/4095*3.3)),0,55)
    oled.text('V',40,55)

    oled.show()
    utime.sleep(1) #延时 1 秒
```

- **实验结果：**

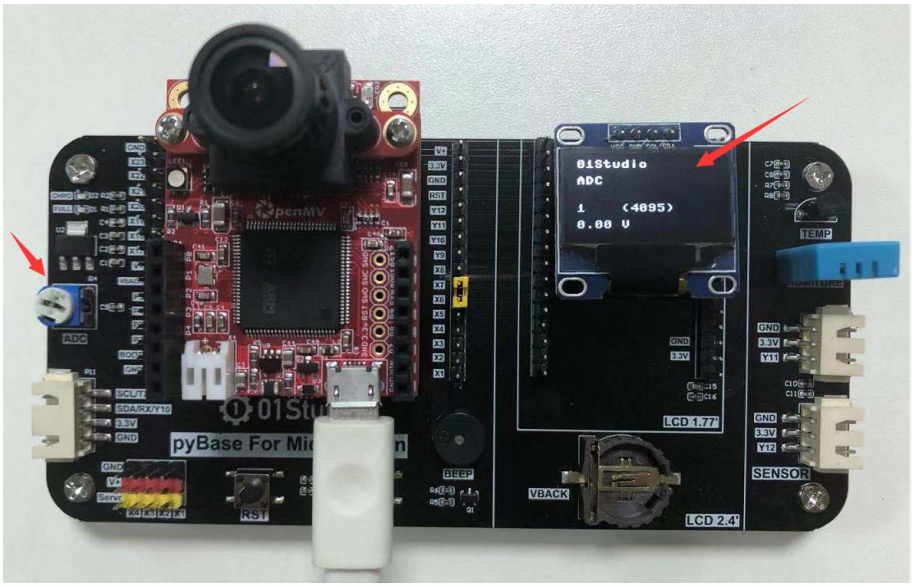


图 23-39 电位器顺时钟拧到尽头是 0V

调节电位器，发现电压测量值不断变化。

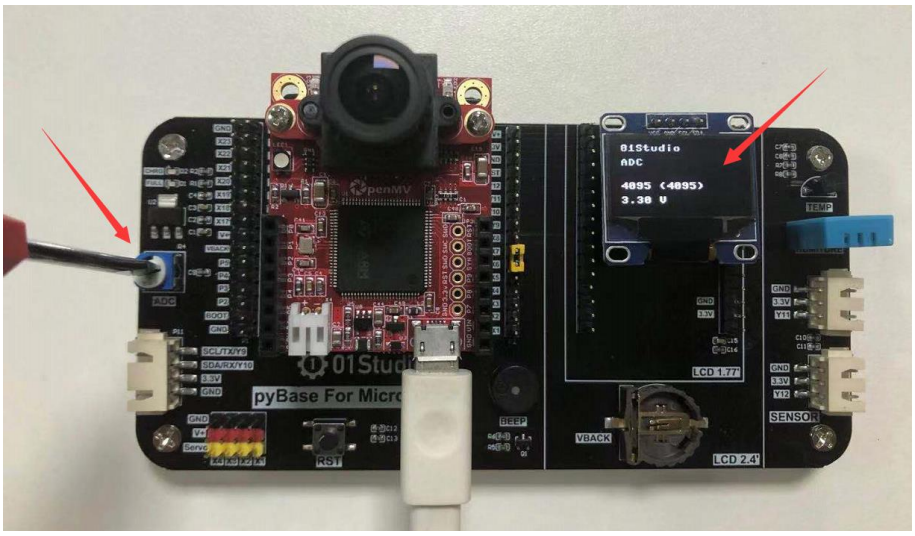


图 23-40 调节电位器来改变输入电压

- **总结：**

这一节我们学习了 ADC 的应用。主要用于电压的检测、模拟信号输入（传感器）检测等应用。

23.9 DAC

- **前言：**

DAC（**Digital-to-Analog Converter**）数字模拟转换器。我们前面做的 ADC 实验，是将模拟信号转化为了数字信号。这一节做的实验刚好是反转，我们将特定的数字信号通过 DAC 输出。输出的形式多种，如特定电压、信号波形（正弦波、方波、三角波等），有条件的用户可以用示波器来观看输出。本章节利用开发板上的无源蜂鸣器，通过 DAC 输出不同的频率来改变其音色。

- **实验平台：**

pyAI-OpenMV4 开发套件。

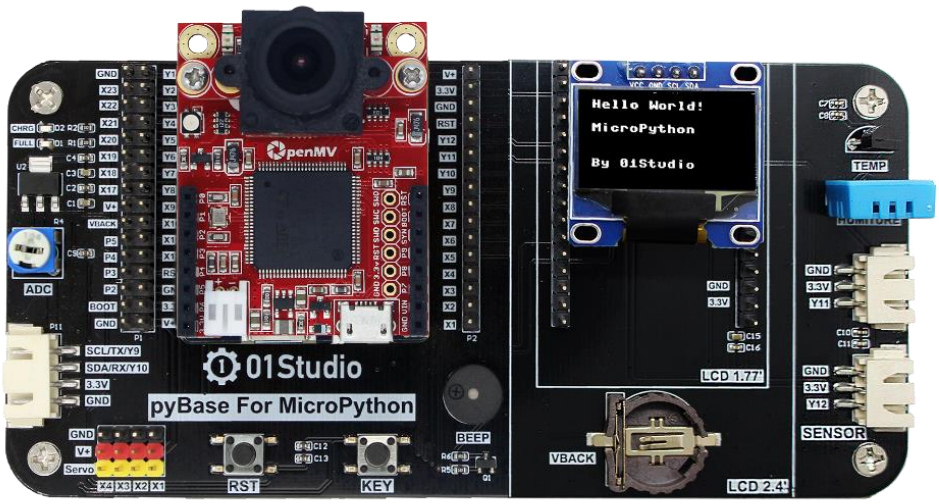


图 23-41 pyAI-OpenMV4 开发套件

- **实验目的：**

通过 DAC 输出不同频率的方波来驱动蜂鸣器。

- **实验讲解：**

蜂鸣器分有源蜂鸣器和无源蜂鸣器，有源蜂鸣器的使用方式非常简单，只需要接上电源，蜂鸣器就发声，断开电源就停止发声。而本实验用到的无源蜂鸣器，是需要给定指定的频率，才能发声的，而且可以通过改变频率来改变蜂鸣器的发声音色，以此来判定 OpenMV4 的 DAC 输出频率是在变化的。

从下图可以看到，pyBase 开发底板的 X5 引脚连接了无源蜂鸣器。如下图所示：

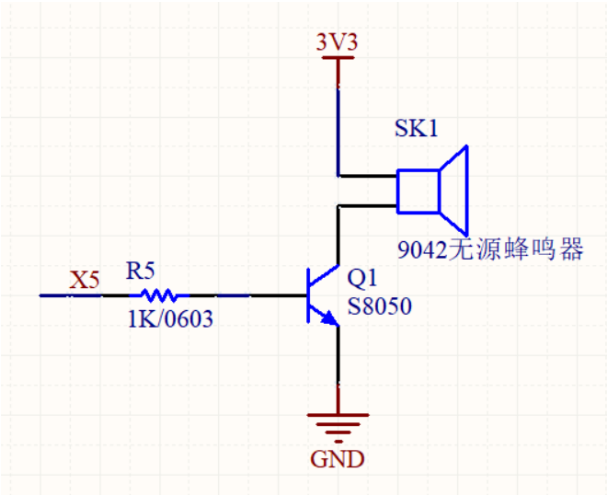


图 23-42 蜂鸣器原理图

而 pyAI-OpenMV4 的 DAC 引脚 P6 则是连接到 pyBase 开发底板的 X6 引脚，因此我们可以用跳线帽或者跳线来连接 pyBase 的 X5 和 X6 引脚。相当于将无源蜂鸣器接到 OpenMV4 的 P6 引脚。

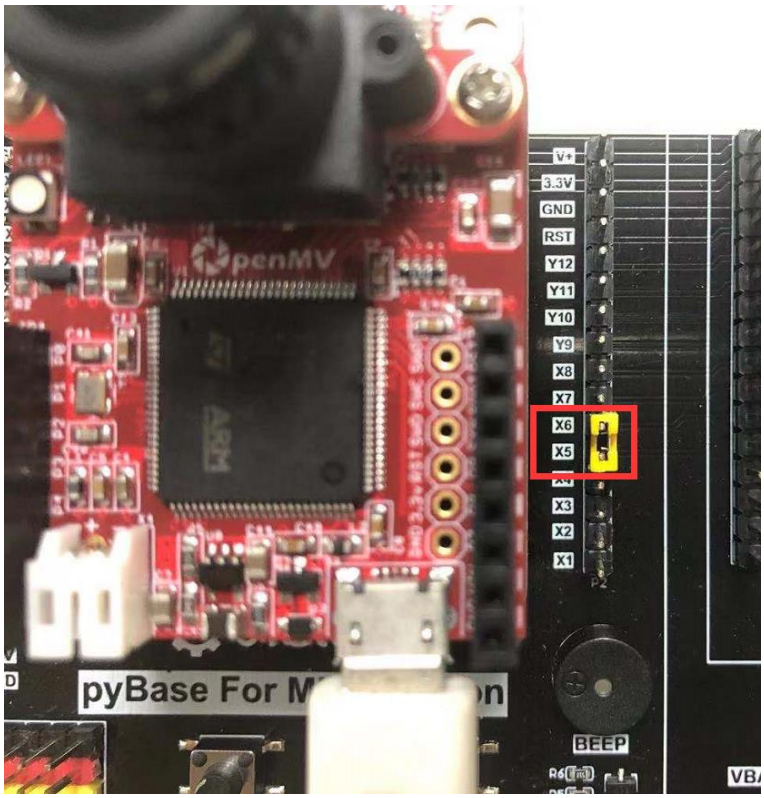


图 23-43 使用跳线帽将 X6 和 X5 短接

在了解了接线后，我们来看看 DAC 模块的函数使用方法。

构造函数
pyb.DAC(port,bits=8)
DAC 对象在 pyb 模块下。 【port】: 1，对应“P6”； 【bits】：8 或者 12
使用方法
DAC.noise(freq)
产生特定频率的噪声信号
DAC.triangle(freq)
产生特定频率三角波信号
DAC.write(value)
输出特定电压(0-VCC)。 【value】范围：0~2^bits-1（例：8bits 那么范围就是 0-255）
DAC.write_timed(data,freq,mode)
输出特定信号。 【data】字节数组; 【freq】频率输出； 【mode】模式： DAC.NORMAL：单次模式； DAC.CIRCULAR：循环模式。

表 23-11　DAC 对象

无源蜂鸣器我们可以用特定频率的方波来驱动，方波的原理很简单，就是一定频率的高低电平转换，使用 **DAC.write(0)**和 **DAC.write(255)**交替输出即可，可以通过延时函数来控制输出变换的次数。实际过程中我们没必要用低效率的延时函数，因为看到 DAC 模块里面已经定义好了特定格式输出的函数 **DAC.write_timed(data,freq,mode)**，将 mode 设定成循环模式，就可以循环不断地输出指定频率的信号了。

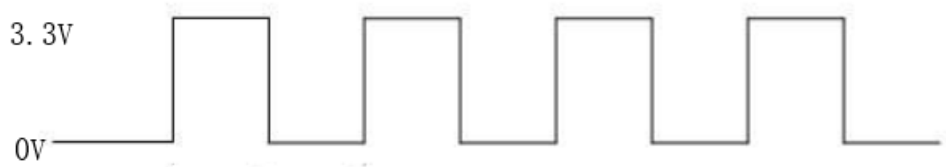


图 23-44 方波信号

结合我们前面的按键实验，我们可以通过 KEY 按键按下来切换输出频率，并在 OLED 上直观体验。具体流程如下：

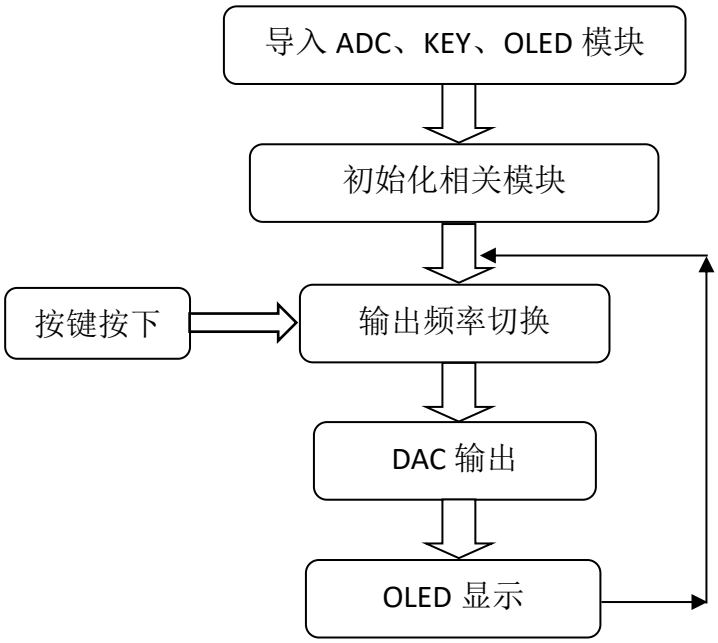


图 23-45 代码编程流程

实验参考代码如下：

```
'''
实验名称：DAC-蜂鸣器
版本：v1.0
日期：2019.9
```

```
作者：01Studio
说明：通过 KEY 按键让 DAC 输出不同频率的方波来驱动蜂鸣器。
'''

#导入相关模块
from pyb import DAC,ExtInt
from machine import Pin,I2C
from ssd1306x import SSD1306_I2C

#初始化相关模块
i2c = I2C(sda=Pin("P0"), scl=Pin("P2"),freq=80000)
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

dac = DAC(Pin("P6"))     #定义 DAC 对象名字为 dac，输出引脚为 P6

#定义 4 组频率值：1Hz、200Hz、1000Hz、5000Hz
freq=[1,200,1000,5000]

# 定义 8 位精度下方波的值。0、255 分别对应输出 0V、3.3V。需要定义成字节数组。
buf = bytearray(2)
buf[0]=0
buf[1]=255

key_node = 0  #按键标志位
i = 0         #用于选择频率数组

################################################
#  按键和其回调函数
################################################
def key(ext):
```

```
    global key_node
    key_node = 1

#下降沿触发，打开上拉电阻
ext = ExtInt(Pin('P9'), ExtInt.IRQ_FALLING, Pin.PULL_UP, key)

################################################
#  OLED 初始显示
################################################
oled.fill(0)  # 清屏显示黑色背景
oled.text('01Studio', 0, 0)  # 首行显示 01Studio
oled.text('DAC-Beep', 0, 15)  # 次行显示实验名称
oled.text('Pls Press USER', 0, 40)  # 显示当前频率
oled.show()

while True:

    if key_node==1: #按键被按下
        i = i+1
        if i == 4:
            i = 0
        key_node = 0 #清空按键标志位

        #DAC 输出指定频率
        dac.write_timed(buf, freq[i]*len(buf), mode=DAC.CIRCULAR)

        #显示当前频率
        oled.fill(0)  # 清屏显示黑色背景
        oled.text('01Studio', 0, 0)  # 首行显示 01Studio
        oled.text('DAC-Beep', 0, 15)  # 次行显示实验名称
```

```
    oled.text(str(freq[i]) + 'Hz', 0, 40)  # 显示当前频率
    oled.show()
```

以上代码中 DAC 的函数使用非常简单，但为了让实验能更直观，加入按键和 OLED 模块后需要更多的代码来支持，但编程逻辑不变。以上代码有几个关键的地方需要补充说明一下：

（1） 由于底层独特的操作方式，使得输入的数据 buf 必须是字节数组，所以需要加入 bytearray()来定义。

（2） Key_node 是全局变量，因此在 key()函数里面用该变量必须添加 global key_node 代码，否则会在函数里面新建一个样的变量。

（3） 变量 i 的作用是为了让 DAC 选择指定的频率数组 freq 输出

- **实验结果：**

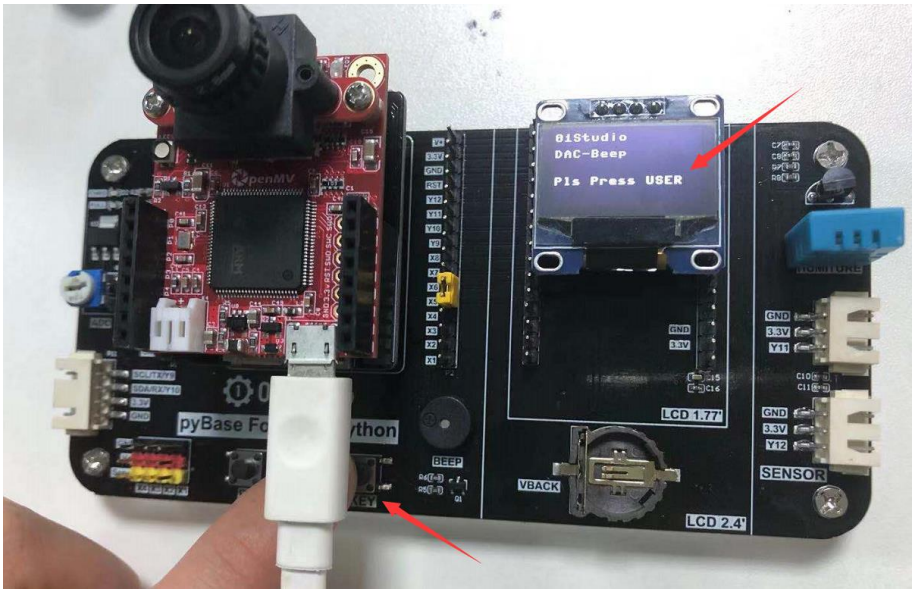


图 23-46 按下按键 KEY 开始

每次按下后发现 OLED 上的频率示意值发生变化，蜂鸣器的叫声也跟着变化。

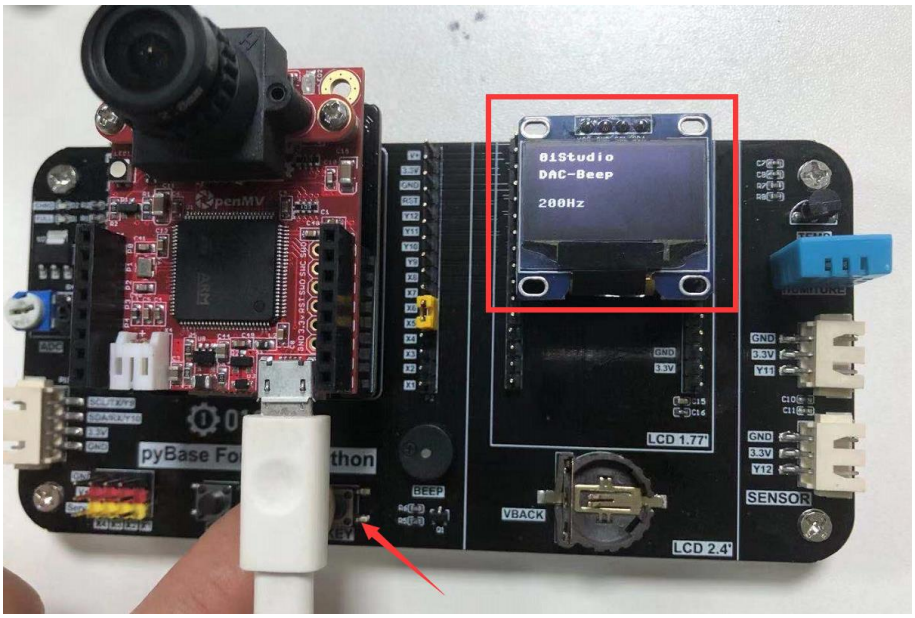


图 23-47 通过按键切换输出频率

有条件的朋友可以使用示波器连接 X5，观察信号波形的变化：

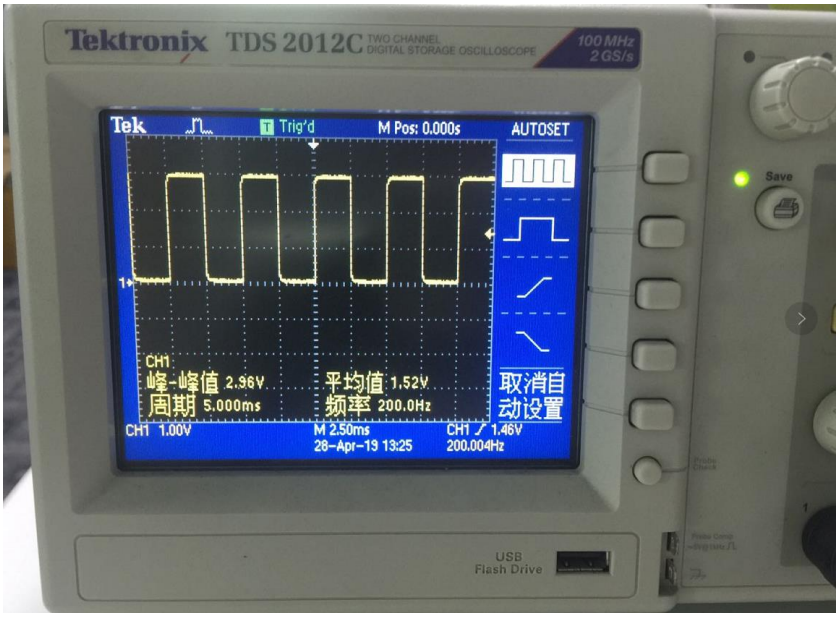


图 23-48 示波器显示

- **总结：**

到了这一节，我们发现实验对象函数使用方法非常简单，而随着要实现功能的复杂化让编程的代码数量变多，逻辑也略显复杂。这是好事，让我们可以将更多精力放在应用上，做出更多好玩的创意。而不需要过多的关注复杂的底层代码开发。

第24章　机器视觉

24.1　摄像头应用

- **前言：**

从前面开发环境建立章节我们测试了 IDE 自带的 hello world 例程，看到了 IDE 成功显示摄像头实时采集的图像以及颜色直方图的显示。今天我们就通过示例代码来看看 OpenMV4 的摄像头是如何使用的。

- **实验平台：**

pyAI-OpenMV4。



图 24-1 pyAI-OpenMV4

- **实验目的：**

学习官方自带的 hello wrold 例程，理解 OpenMV4 摄像头基本编程和配置原理。

- **实验讲解：**

我们打开 IDE 中的 hello world 示例代码，其位置在文件---示例---01-Basics---helloworld.py。

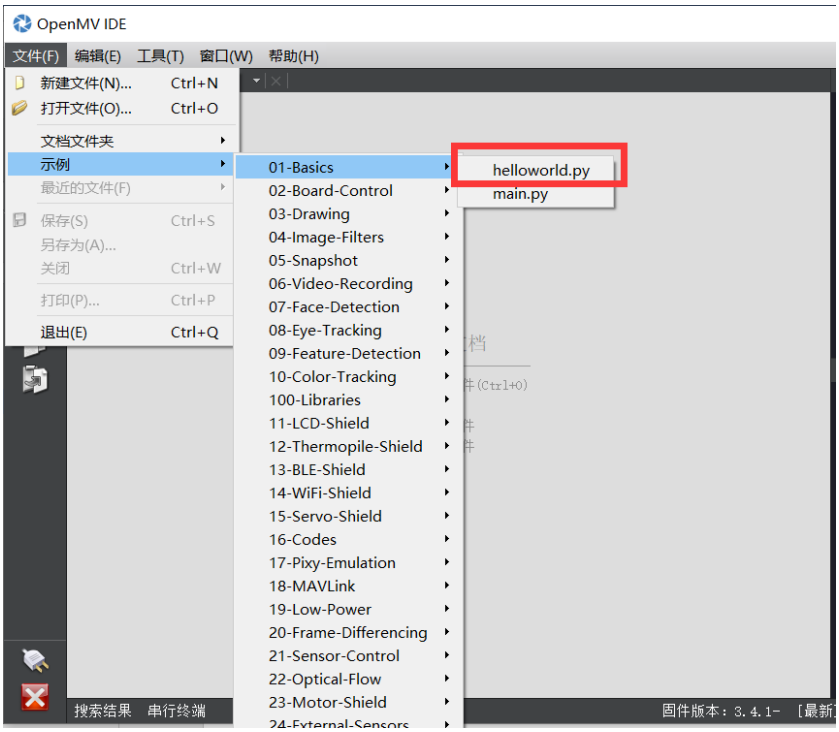


图 24-2 helloworld 例程位置

打开后发现编辑框出现了相关代码，我们可以先直接跑一下代码看看实验现象，连接 pyAI-OpenMV4，点击运行，可以发右图上方出现了摄像头实时采集的图像。

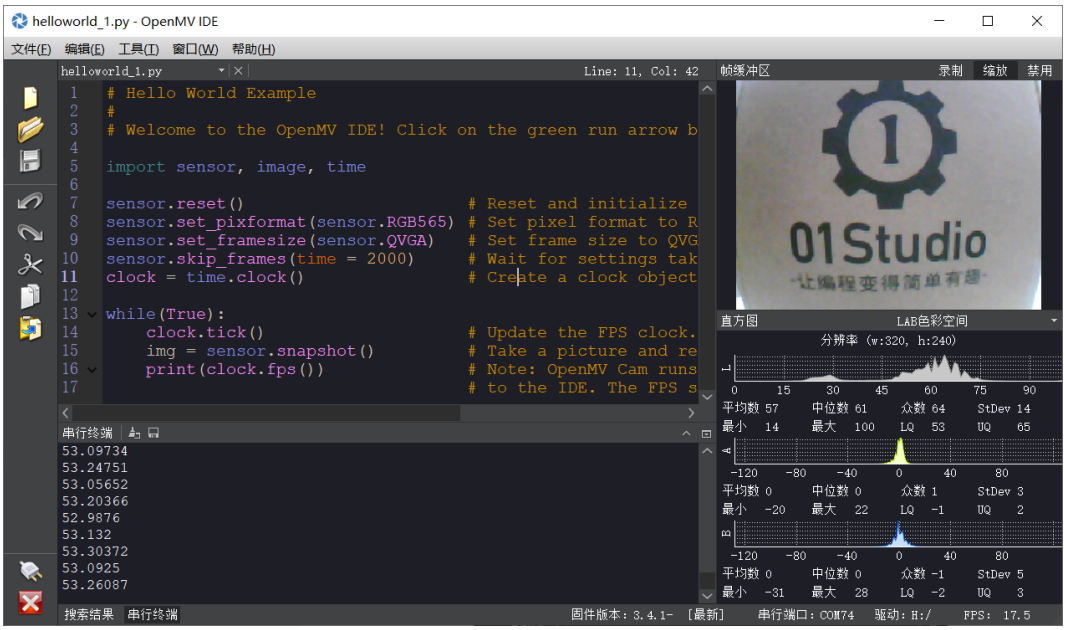


图 24-3

OpenMV 已经将所有的摄像头功能封装到 sersor 模块中，用户可以通过调用轻松使用。这也是使用 MicroPython 编程的魅力所在。

构造函数
sensor
摄像头对象，通过 import 直接调用
使用方法
sensor.reset()
初始化摄像头
sensor.set_pixformat(*pixformat*)
设置像素格式。pixformat 有 3 个参数。 sensor.GRAYSCAL：灰度图像，每像素 8 位（1 字节），处理速度快； sensor.RGB565：每像素为 16 位（2 字节），5 位用于红色，6 位用于绿色，5 位用于蓝色，处理速度比灰度图像要慢。 sensor.BAYER:占空间小，仅捕捉图像用，不可做图像处理。
sensor.set_framesize(*framesize*)
设置每帧大小（即图像尺寸）。常用的 *framesize* 参数有下面这些： sensor.QQVGA: 160*120; sensor.QVGA: 320*240; sensor.VGA: 640*480; sensor.LCD: 128x160 (LCD 模块用) sensor.QQVGA2: 128x160 (LCD 模块用) （更多参数请看表格末尾官方文档链接。）
sensor.skip_frames([*n*, *time*])
摄像头配置后跳过 n 帧或者等待时间 time 让其变稳定。n:跳过帧数；time：等待时间,单位 ms。 （如果 n 和 time 均没指定，则默认跳过 300 毫秒的帧。）
sensor.snapshot()
使用相机拍摄一张照片，并返回 image 对象。
*其它更多用法请阅读 openmv 官方文档： 文档链接：http://docs.openmv.io/library/omv.sensor.html

表 24-1　sensor 摄像头对象

我们再来看看本例程用于计算 FPS（每秒帧数）的 CLOCK 对象模块。

构造函数
clock=time.clock()
创建一个时钟。
使用方法
clock.tick()
开始追踪运行时间。
clock.fps ()
停止追踪运行时间，并返回当前 FPS（每秒帧数）。 在调用该函数前始终首先调用 tick 。
*其它更多用法请阅读 MaixPy 官方文档： 文档链接：https://maixpy.sipeed.com/zh/libs/standard/utime.html#clocktick

图 24-4 Clock 对象

我们来看看 helloword 代码的编写流程图：

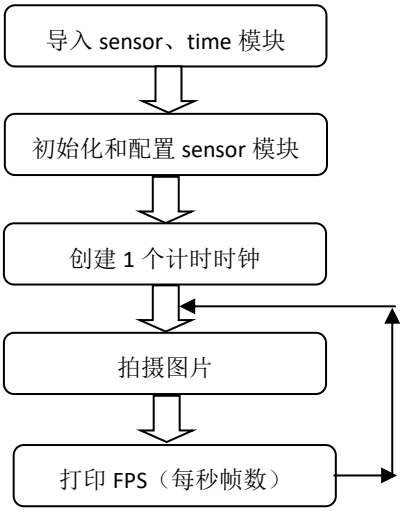


图 24-5 helloword 代码编写流程

这个实验运行的就是编辑框里面的 helloworld 代码：

```
# Hello World 例程
#
#翻译和注释：01Studio

#导入相关模块
import sensor, image, time, lcd

lcd.init(freq=15000000)   # LCD 初始化
sensor.reset()             # 复位和初始化摄像头设备，执行 sensor.run(0)停止

sensor.set_pixformat(sensor.RGB565) # 设置像素格式为彩色 RGB565 (或灰色
GRAYSCALE)
sensor.set_framesize(sensor.QVGA)   # 设置帧大小为 QVGA (320x240) 默认 LCD
大小
sensor.skip_frames(time = 2000)     # 等待设置生效
clock = time.clock()                # 创建一个时钟来追踪 FPS（每秒拍摄帧数）.

while(True):
    clock.tick()                    # 更新 FPS 时钟.
    img = sensor.snapshot()         # 拍摄图像并返回 image.
    lcd.display(img)                # 在 LCD 上显示
    print(clock.fps())              # 注意：当 MaixPy 连接到 IDE 时候，运行速
度减半，
                                    # 因此当脱机运行时，FPS 会提升。
```

- **实验结果：**

点击运行，可以看到在右边显示摄像头实时拍摄情况，下方则显示 RGC 颜色直方图。

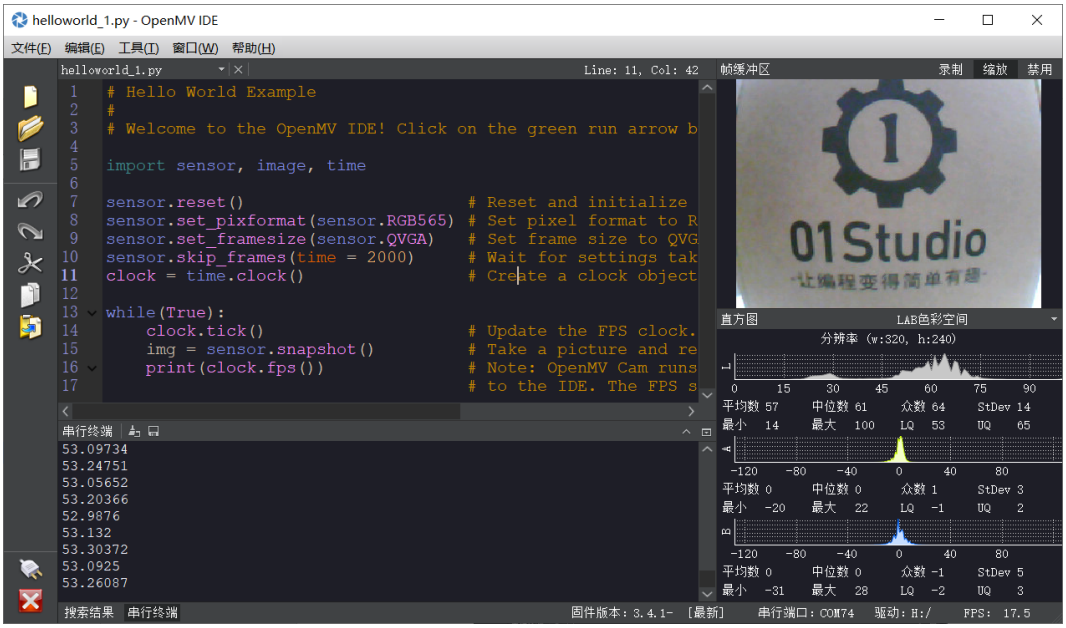


图 24-6 实验现象

点击左下角串口终端，可以看到软件弹出串口打印串口，实时显示当前的 FPS(没秒帧数)值为 50 多帧。

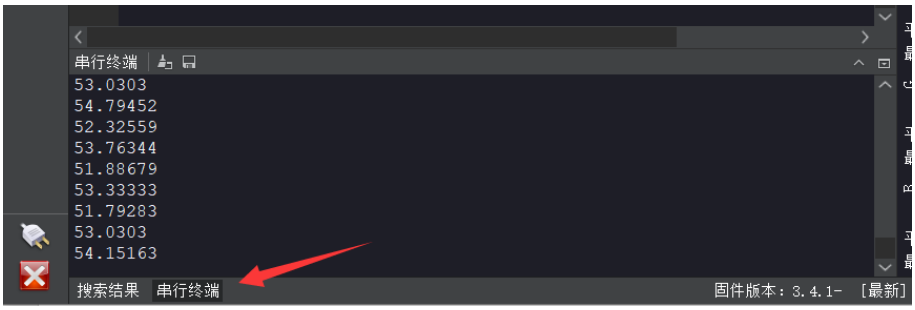


图 24-7 串口终端显示 FPS

- **总结：**

通过本实验，我们了解了摄像头 sensor 模块以及时间 time 模块的原理和应用，可以看到 OpenMV 将摄像头功能封装成 sensor 模块，用户不必关注底层代码编可以轻松使用。

24.2 画图

- **前言：**

通过摄像头采集到照片后，我们会进行一些处理，而这时候往往需要一些图形来指示，比如在图片某个位置标记箭头、人脸识别后用矩形框提示等。本节就是学习在图形上画图的使用功能。

- **实验平台：**

pyAI-OpenMV4。



图 24-8 pyAI-OpenMV4

- **实验目的：**

在拍摄的图片上画各种图形。

- **实验讲解：**

在 OpenMV IDE 的示例程序里面有各类画图的示例程序，大家可以自行阅读。

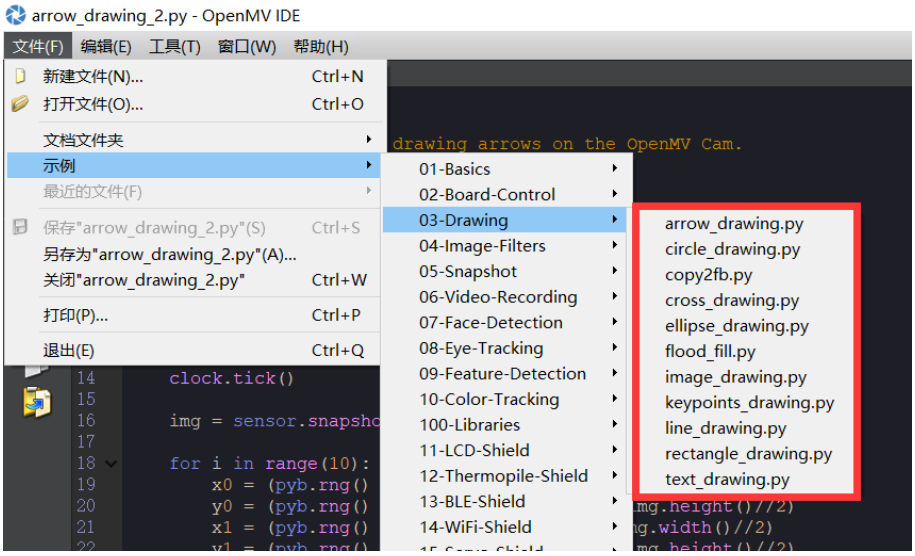


图 24-9 画图示例程序

上一节我们学习了摄像头 sensor 模块应用，通过摄像头实时采集到的是图片 image，没错，本节实验就是建立在非常重要的 image 模块上面。openmv 已经将图片处理（包含画图）封装成各类模块，我们只需要熟悉其构造函数和使用方法即可，具体如下：

构造函数
img=sensor.snapshot() 或 img=image.Image(*path*[, *copy_to_fb=False*])
创建图像，通过拍摄或者读取文件路径获取。 *copy_to_fb=True：可以加载大图片；* copy_to_fb=False：不可以加载大图片。 示例：img = image.Image("/01Studio.bmp", copy_to_fb=True),表示加载根目录下的 01Studio.bmp 图片。
使用方法
image.draw_line(*x0*, *y0*, *x1*, *y1*[, *color*[, *thickness=1*]])
画线段。（x0,y0）:起始坐标；（x1,y1）:终点坐标；color:颜色，如（255,0,0）表示红色；thickness：粗细。
image.draw_rectangle(*x*, *y*, *w*, *h*[, *color*[, thickness=*1*[, *fill=False*]]])
画矩形。（x,y）:起始坐标；w:宽度；h:长度；color：颜色；thickness：边框粗

细；fill:是否填充。
image.draw_circle(*x*, *y*, *radius*[, *color*[, thickness=*1*[, *fill=False*]]])
画圆。（x,y）:圆心； radius:半径； color：颜色；thickness:线条粗细；fill：是否填充。
image.draw_arrow(*x0*, *y0*, *x1*, *y1*[, *color*[, size,[thickness=*1]*]])
画箭头。（x0,y0）:起始坐标；（x1,y1）:终点坐标；color:颜色；size:箭头位置大小。thickness：线粗细。
image.draw_cross(*x*, *y*[, *color*[, *size=5*[, *thickness=1*]]])
画十字交叉。（x,y）：交叉坐标；color:颜色；size:尺寸；thickness：线粗细。
image.draw_string(*x*, *y*, *text*[, *color*[, *scale*=1[,*mono_space=True*…]]])
写字符。(x,y)：起始坐标；text:字符内容；color：颜色；scale：字体大小；mono_space:强制间距。
*其它更多用法请阅读 OpenMV 官方文档： 文档链接：http://docs.openmv.io/library/omv.image.html

表 24-2 image 对象画图功能

熟悉了 image 对象的画图功能后，我们尝试在摄像头采集到的画面依次画出线段、矩形、圆形、箭头、十字交叉和字符。具体编程思路如下：

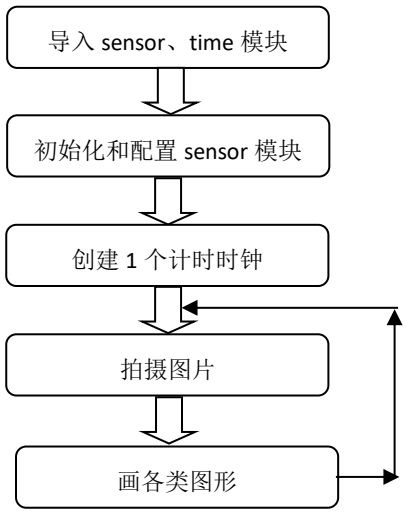


图 24-10 代码编写流程

参考代码：

```
# 在这里写上你的代码 :-)
'''
实验名称：画各种图形和写字符
版本：v1.0
日期：2019.10
作者：01Studio
'''
import sensor, image, time, pyb


sensor.reset()
sensor.set_pixformat(sensor.RGB565) # or GRAYSCALE...
sensor.set_framesize(sensor.QVGA) # or QQVGA...
sensor.skip_frames(time = 2000)
clock = time.clock()


while(True):


    clock.tick()


    img = sensor.snapshot()


    # 画线段：从 x0, y0 到 x1, y1 坐标的线段，颜色红色，线宽度 2。
    img.draw_line(20, 20, 100, 20, color = (255, 0, 0), thickness = 2)


    #画矩形：绿色不填充。
    img.draw_rectangle(150, 20, 100, 30, color = (0, 255, 0),
                       thickness = 2, fill = False)


    #画圆：蓝色不填充。
```

```
    img.draw_circle(60, 120, 30, color = (0, 0, 255), thickness = 2,
                    fill = False)

    #画箭头：白色。
    img.draw_arrow(150, 120, 250, 120, color = (255, 255, 255), size =
                    20, thickness = 2)

    #画十字交叉。
    img.draw_cross(60, 200, color = (255, 255, 255), size = 20,
                    thickness = 2)

    #写字符。
    img.draw_string(150, 200, "Hello 01Studio!", color = (255, 255,
                    255), scale = 2,mono_space = False)

    print(clock.fps())
```

- **实验结果：**

在 OpenMV IDE 中打开例程文件，或者直接复制到新建的文件中，点击运行。可以看到在图像缓冲区中画上了各种图形。

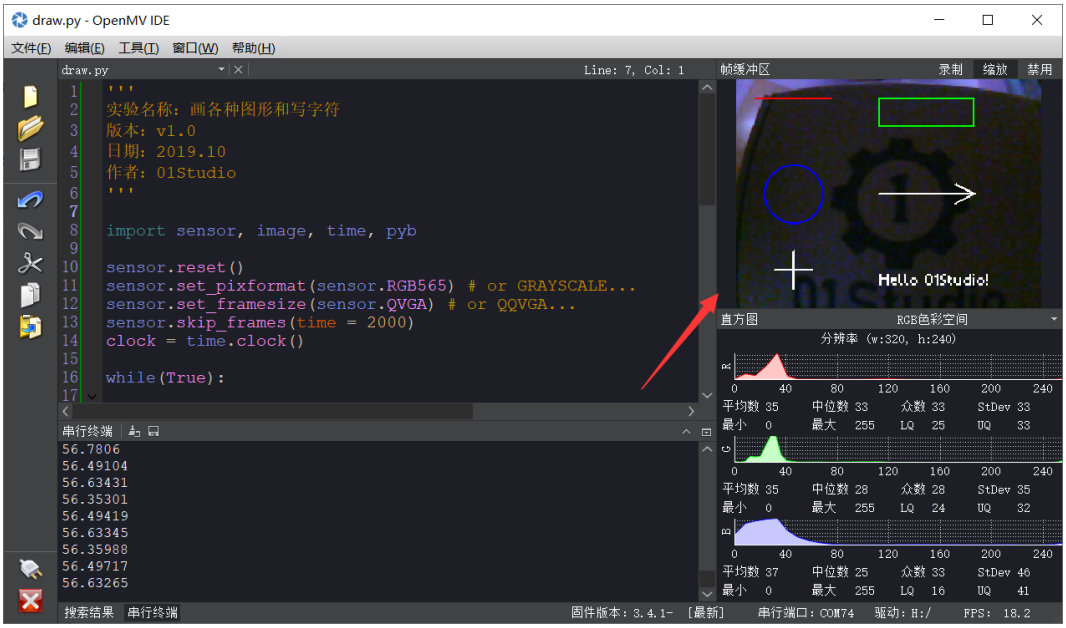


图 24-11 画图形

- **总结：**

画图形是很基础的功能，但在以后的实验中特别是指示识别内容时候会经常用到。

24.3 特征检测

从本节起，可以说是正式进入视觉检测的学习。

特征检测，要做的就是对图像进行特定的特征检测，如本节包含的边缘检测、各种形状识别、特征点识别、线性回归等都是非常有代表性的实验。特征检测的全部官方例程位于 Feature-Detection 例程下，本节挑选重点实验进行讲解。

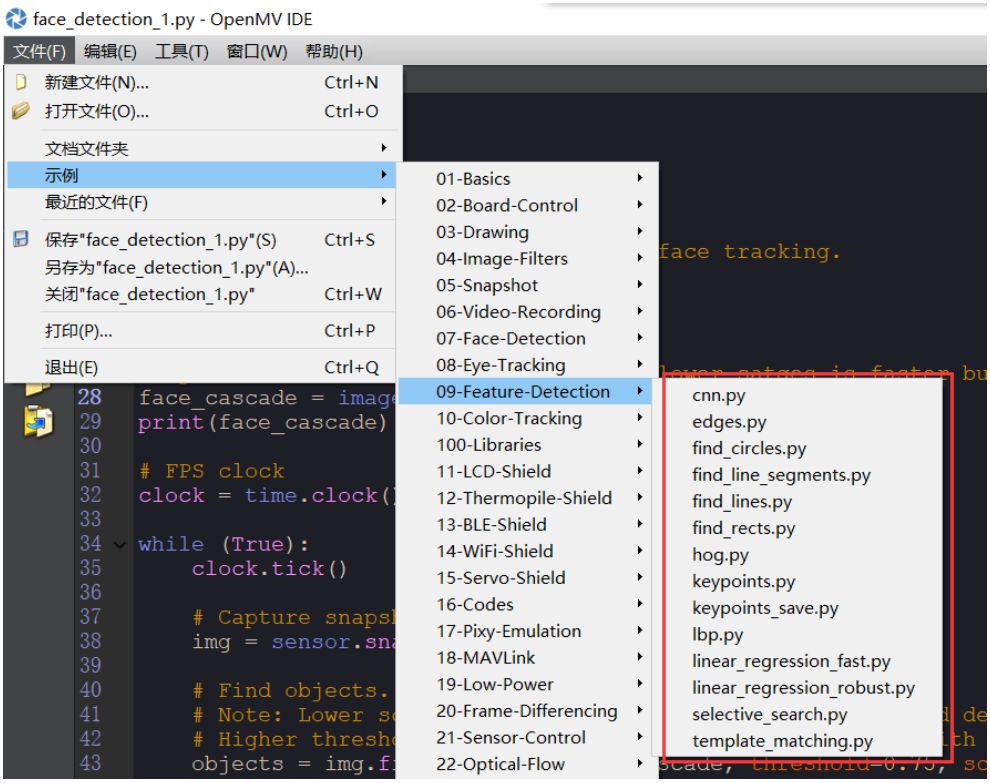


图 24-12 特征检测官方例程位置

24.3.1 边缘检测

- **前言：**

生活中每个物体都有一个边缘。简单来说就是轮廓，本节学习的是使用 MicroPython 结合 OpenMV4 自带的库来做图像轮廓检测。

- **实验平台：**

pyAI-OpenMV4。



图 24-13 pyAI-OpenMV4

- **实验目的：**

识别摄像头采集的图像轮廓并显示。

- **实验讲解：**

OpenMV4 集成了边缘识别 find_edges 函数，位于 image 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下边缘处理函数相关说明，具体如下：

构造函数
image.find_edges(*edge_type*[, *threshold*])
边缘检测，将图像变为黑白，边缘保留白色像素。 【edge_type】处理方式： image.EDGE_SIMPLE - 简单的阈值高通滤波算法 image.EDGE_CANNY - Canny 边缘检测算法 【threshold】包含高、低阈值的二元组，默认是（100,200），仅支持灰度图像。
使用方法
直接调用该函数。

表 24-3　边缘处理函数

由此可见边缘处理的方法非常简单，我们结合前面摄像头的应用，整理一下编程思路如下：

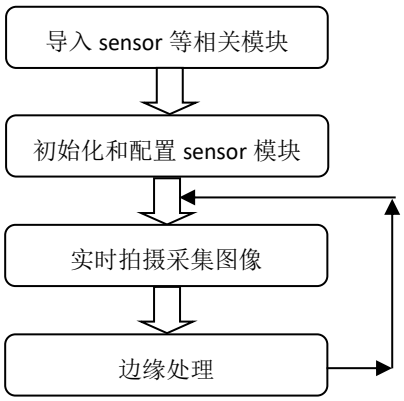


图 24-14 代码编写流程

官方参考代码路径【示例→Feature-Detection→edges.py】，具体如下：

```
# 使用 Canny 算法做边缘检测：
#
# 这个例子演示了 Canny 边缘检测器.
#
```

```
#翻译：01Studio

import sensor, image, time

#初始化摄像头
sensor.reset() # 初始化摄像头模块.
sensor.set_pixformat(sensor.GRAYSCALE) # 或者使用 sensor.RGB565 彩色
sensor.set_framesize(sensor.QQVGA) # 或者使用 sensor.QVGA (or others)
sensor.skip_frames(time = 2000) #延时让摄像头文稳定.
sensor.set_gainceiling(8) #设置增益，这是官方推荐的参数

clock = time.clock() # Tracks FPS.

while(True):

    clock.tick() # 用于计算 FPS（每秒帧数）.
    img = sensor.snapshot() # 拍摄并返回图像.

    #使用 Canny 边缘检测器
    img.find_edges(image.EDGE_CANNY, threshold=(50, 80))

    # 也可以使用简单快速边缘检测，效果一般，配置如下
    #img.find_edges(image.EDGE_SIMPLE, threshold=(100, 255))

    print(clock.fps()) #显示 FPS（每秒帧数）
```

- **实验结果：**

使用 IDE 运行上面的代码，可以看到实验现象如下：



图 24-15 边缘检测实验：原图（左）和实验图片（右）

- **总结：**

本节做了特征检测的边缘检测实验，可以看到使用 MicroPython 开发是如此的简单美妙。本节仅仅是一个开始，接下来我们将学习更多的机器视觉应用。

24.3.2 圆形识别

- **前言：**

  本节学习的是对图像中的圆形进行识别。

- **实验平台：**

  pyAI-OpenMV4。



图 24-16 pyAI-OpenMV4

- **实验目的：**

  识别图像中的圆形并用画圆功能指示出来。

- **实验讲解：**

  OpenMV4 集成了圆形识别 find_circles 函数，位于 image 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下圆形识别函数相关说明，具体如下：

构造函数
image.find_circles([*roi*[, *x_stride=2*[, *y_stride=1*[, *threshold=2000*[, *x_margin=10*[, *y_margin=10*[, *r_margin=10*[, *r_min=2*[, *r_max*[, *r_step=2*]]]]]]]]]])

找圆函数。返回一个 image.circle 圆形对象，该圆形对象有 4 个值： x, y（圆心）, r （半径）和 magnitude（量级）；量级越大说明识别到的圆可信度越高。 【roi】识别区域（x,y,w,h），未指定则默认整张图片； 【threshold】阈值。返回大于或等于 threshold 的圆，调整识别可信度； 【x_stride】【y_stride】霍夫变换时跳过 x，y 像素的量； 【x_margin】【y_margin】【r_margin】控制所检测圆的合并； 【r_min】【r_max】控制识别圆形的半径范围； 【r_step】控制识别步骤
使用方法
直接调用该函数。**（大部分参数使用默认即可，不支持压缩图像和 bayer 图像）**

表 24-4 圆形识别函数

圆形检测的编程思路如下：

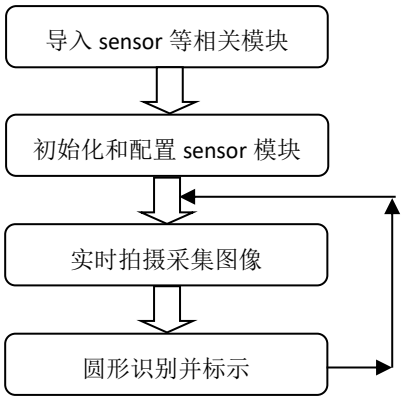


图 24-17 代码编写流程

官方参考代码路径【示例→Feature-Detection→find_circles.py】，具体如下：

```
# 找圆形例程
#
# 这个例程使用 Hough 算法（霍夫变换）
# 参考链接： https://en.wikipedia.org/wiki/Circle_Hough_Transform
#
# 需要注意的是该例程只适用于识别在图像内部完整的圆形，超出图像或者识别区域
```

```
#（roi）的无效。
#
#翻译：01Studio

import sensor, image, time

#摄像头模块初始化
sensor.reset()
sensor.set_pixformat(sensor.RGB565) # 使用 grayscale 灰度图像处理速度会更快
sensor.set_framesize(sensor.QQVGA)
sensor.skip_frames(time = 2000)
clock = time.clock()

while(True):

    clock.tick()

    #lens_corr 为了去除畸变，1.8 是默认参数，可以根据自己实际情况调整
    img = sensor.snapshot().lens_corr(1.8)

    # 圆形类有 4 个参数值： 圆心(x, y), r (半径)和 magnitude（量级）;
    # 量级越大说明识别到的圆可信度越高。

    # `threshold` 参数控制找到圆的数量，数值的提升会降低识别圆形的总数。
    # `x_margin`, `y_margin`, and `r_margin`控制检测到接近圆的合并调节.
    # r_min, r_max, and r_step 用于指定测试圆的半径范围。

    for c in img.find_circles(threshold = 2000, x_margin = 10, y_margin
              = 10, r_margin = 10,r_min = 2, r_max = 100, r_step = 2):
```

```
        #画红色圆做指示
        img.draw_circle(c.x(), c.y(), c.r(), color = (255, 0, 0))
        print(c) #打印圆形的信息

    print("FPS %f" % clock.fps()) #打印 FPS（每秒采集帧数）
```

- **实验结果：**

使用 IDE 运行以上代码，可以看到实验现象如下：



图 24-18 圆形识别实验：原图（左）和实验图片（右）

在串行终端可以看到代码 print(c) 通过串口打印出来圆形的信息。

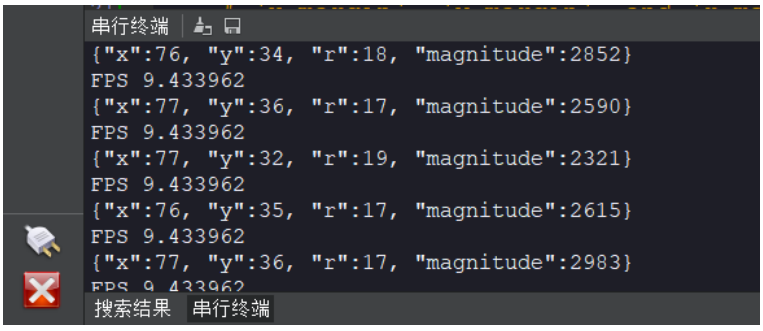


图 24-19 串口终端

- **总结：**

本节学习了圆形识别，可以使用该例程来识别圆形图案或者球类物体。

24.3.3 线段识别

- **前言：**

本节学习的是对图像中的线段进行识别。

- **实验平台：**

pyAI-OpenMV4。



图 24-20 pyAI-OpenMV4

- **实验目的：**

识别图像中的线段并用画线段功能指示出来。

- **实验讲解：**

OpenMV4 集成了线段识别 find_line_segments 函数，位于 image 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下线段识别函数相关说明，具体如下：

构造函数
image.find_line_segments([*roi*[, *merge_distance=0*[, *max_theta_difference=15*]]])
线段识别函数。返回一个 image.line 线段对象列表。

【roi】识别区域（x,y,w,h），未指定则默认整张图片； 【merge_distance】两条线段间可以相互分开而不被合并的最大像素； 【max_theta_difference】将少于这个角度值的线段合并。
使用方法
直接调用该函数。（**大部分参数使用默认即可，不支持压缩图像和 bayer 图像**）

表 24-5　圆形识别函数

线段检测的编程思路如下：

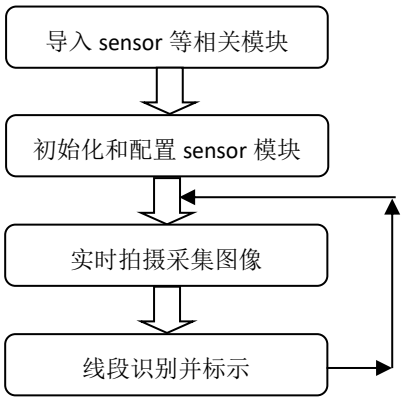


图 24-21 代码编写流程

官方参考代码路径【示例→Feature-Detection→find_line_segments.py】，具体如下：

```
# 线段识别例程
#
# 该例程演示了如果在图像中找出线段，每个被找出的线段都会返回一个对象，包括线端
# 旋转的线对象。


# 使用 find_line_segments() 可以找特定的线段 (速度比较慢).
# 使用 find_line_segments() 找任意线段 (速度比较快).
#
#翻译：01Studio
```

```
enable_lens_corr = False # 打开以获得更直的线段

import sensor, image, time

#摄像头初始化
sensor.reset()
sensor.set_pixformat(sensor.RGB565) # grayscale 灰度图像处理速度更快
sensor.set_framesize(sensor.QQVGA)
sensor.skip_frames(time = 2000)
clock = time.clock()

# 所有线段对象都可以通过 `x1()`, `y1()`, `x2()`, and `y2()` 方法获取其端
# 点，当然也可以使用`line()` 获取上面 4 个值以用于 `draw_line()`画线.

while(True):

    clock.tick()

    img = sensor.snapshot()
    if enable_lens_corr: img.lens_corr(1.8) # for 2.8mm lens...

# `merge_distance` 控制相近的线段是否合并.  数值 0 (默认值)表示不合并。数值
#为 1 时候表示相近 1 像素的线段被合并。因此你可以通过改变这个参数来控制检测到线
#段的数量。

# `max_theta_diff` 控制相差一定角度的线段合并，默认是 15 度，表示 15 度内的线
# 段都会合并

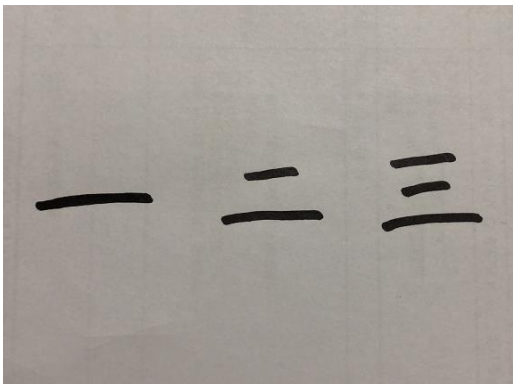
    for l in img.find_line_segments(merge_distance = 0,
                                    max_theta_diff = 5):
```

```
    img.draw_line(l.line(), color = (255, 0, 0))
    print(l)

print("FPS %f" % clock.fps())
```

- **实验结果：**

使用 IDE 运行以上代码，可以看到实验现象如下：



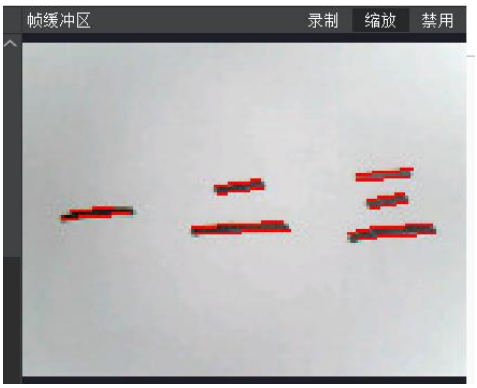


图 24-22 线段识别实验：原图（左）和实验图片（右）

- **总结：**

本节学习了线段识别，可以使用该例程来识别线段图案或者相关物体。

24.3.4 直线识别

- **前言：**

本节学习的是对图像中的直线进行识别。

- **实验平台：**

pyAI-OpenMV4。



图 24-23 pyAI-OpenMV4

- **实验目的：**

识别图像中的直线并用画直线功能指示出来。

- **实验讲解：**

OpenMV4 集成了直线识别 find_lines 函数，位于 image 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下识别函数相关说明，具体如下：

构造函数
image.find_lines([*roi*[, *x_stride=2*[, *y_stride=1*[, *threshold=1000*[, *theta_margin=25*[, *rho_margin=25*]]]]]])
直线识别函数。返回一个 image.line 直线对象；

【roi】识别区域（x,y,w,h），未指定则默认整张图片； 【threshold】阈值。返回大于或等于 threshold 的直线，调整识别可信度； 【x_stride】【y_stride】霍夫变换时跳过 x，y 像素的量； 【theta_margin】【rho_margin】控制所检测直线的合并规则；
使用方法
直接调用该函数。（**大部分参数使用默认即可，不支持压缩图像和 bayer 图像**）

表 24-6　直线识别函数

圆形检测的编程思路如下：

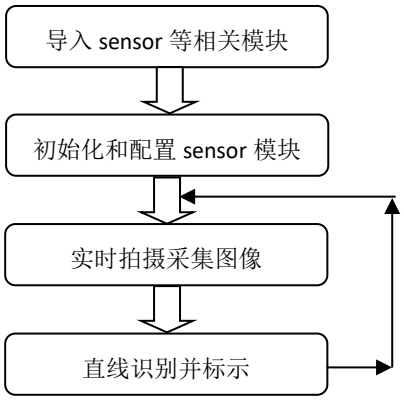


图 24-24 代码编写流程

官方参考代码路径【示例→Feature-Detection→find_lines.py】，具体如下：

```
# 直线识别例程
#
# 该例程演示了如何在图像中找出直线，每个被找出的直线都会返回一个对象，包括直线
#的线路定位。
#
# 直线识别使用了霍夫转换算法:
# 参考链接：http://en.wikipedia.org/wiki/Hough_transform
# 你可以从上面链接更深入了解 `theta` 和 `rho` 参数的概念.
#
# 使用 find_lines() 找直线 (速度比较慢).
# 使用 find_line_segments() 找线段 (速度比较快).
```

```
#
#翻译：01Studio



enable_lens_corr = False  # 打开以获得更直的线段


import sensor, image, time


sensor.reset()
sensor.set_pixformat(sensor.RGB565) # grayscale is faster
sensor.set_framesize(sensor.QQVGA)
sensor.skip_frames(time = 2000)
clock = time.clock()


# 所有直线都可以通过 `theta()` 函数来获取跟水平线的夹角。
# 你也可以通过滤波算法来平滑角度.


min_degree = 0
max_degree = 179


# 所有直线对象都可以通过 `x1()`, `y1()`, `x2()`, and `y2()` 方法获取其端
#点，当然也可以使用`line()` 获取上面 4 个值以用于 `draw_line()`画线.


while(True):


    clock.tick()
    img = sensor.snapshot()
    if enable_lens_corr: img.lens_corr(1.8) # for 2.8mm lens...


    # `threshold` 参数控制找到直线的数量，数值的提升会降低识别直线的总数。
```

```
    # `theta_margin` and `rho_margin` 控制直线是否合并。若少于预设值则合并


    for l in img.find_lines(threshold = 1000, theta_margin = 25,
                                rho_margin = 25):


   #直线角度在 0 至 180 度范围内
        if (min_degree <= l.theta()) and (l.theta() <= max_degree):
            img.draw_line(l.line(), color = (255, 0, 0))
            print(l)


    print("FPS %f" % clock.fps())


# 关于 rho 出现负数:
# [theta+0:-rho] 元组与 [theta+180:+rho]相同.
```

- **实验结果：**

使用 IDE 运行以上代码，可以看到实验现象如下：

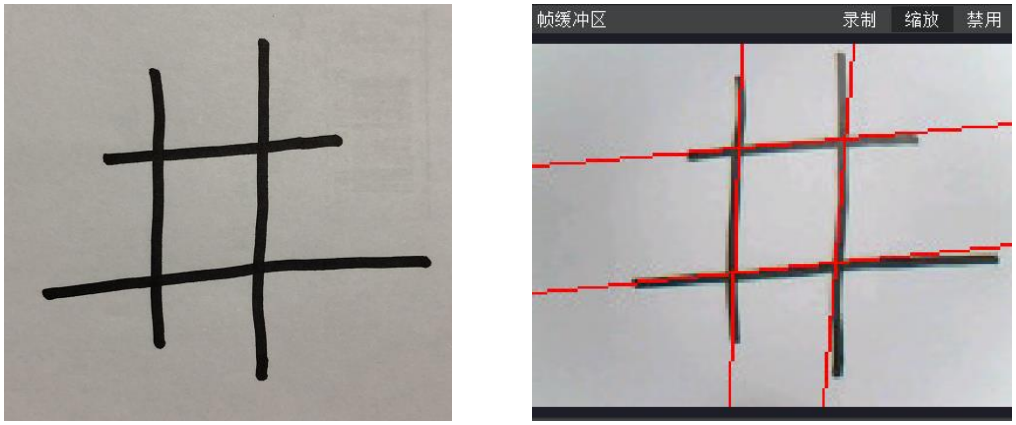


图 24-25 直线识别实验：原图（左）和实验图片（右）

- **总结：**

本节学习了直线识别，可以使用该例程来识别直线图案或者相关物体。

24.3.5 矩形识别

- **前言：**

  本节学习的是对图像中的矩形进行识别。

- **实验平台：**

  pyAI-OpenMV4。



图 24-26 pyAI-OpenMV4

- **实验目的：**

  识别图像中的矩形并用画图功能指示出来。

- **实验讲解：**

  OpenMV4 集成了矩形识别 find_rects 函数，位于 image 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下识别函数相关说明，具体如下：

构造函数
`image.find_rects([roi=Auto, threshold=10000])`
矩形识别函数。返回一个 image.rect 矩形对象列表； 【roi】识别区域（x,y,w,h），未指定则默认整张图片； 【threshold】阈值。返回大于或等于 threshold 的矩形，调整识别可信度；

使用方法
直接调用该函数。（**大部分参数使用默认即可，不支持压缩图像和 bayer 图像**）

表 24-7　矩形识别函数

圆形检测的编程思路如下：

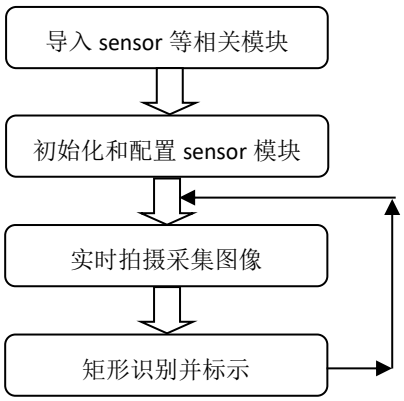


图 24-27 代码编写流程

官方参考代码路径【示例→Feature-Detection→find_rects.py】，具体如下：

```
#  矩形识别例程
#
#这个例子演示了如何使用 quad 阈值在图像中找到矩形
#四元阈值检测算法检测矩形使用鲁棒的方式，是远远好于霍夫
#转换的方法。例如，它仍然可以检测矩形，即使镜头
#扭曲会使这些矩形看起来弯曲。所以圆角矩形是没有问题的!
#(鉴于此，代码也将检测小半径圆)…
#
#翻译：01Studio

import sensor, image, time

sensor.reset()
# grayscale is faster (160x120 max on OpenMV-M7)
```

```
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA)
sensor.skip_frames(time = 2000)
clock = time.clock()

while(True):
    clock.tick()
    img = sensor.snapshot()

    # `threshold` 需要设置一个比价大的值来过滤掉噪声。
    #这样在图像中检测到边缘亮度较低的矩形。矩形
    #边缘量级越大，对比越强…

    for r in img.find_rects(threshold = 10000):
        img.draw_rectangle(r.rect(), color = (255, 0, 0)) #画矩形显示
        for p in r.corners(): #四角画小圆形
            img.draw_circle(p[0], p[1], 5, color = (0, 255, 0))
        print(r)

    print("FPS %f" % clock.fps())
```

- **实验结果：**

使用 IDE 运行以上代码，可以看到实验现象如下：

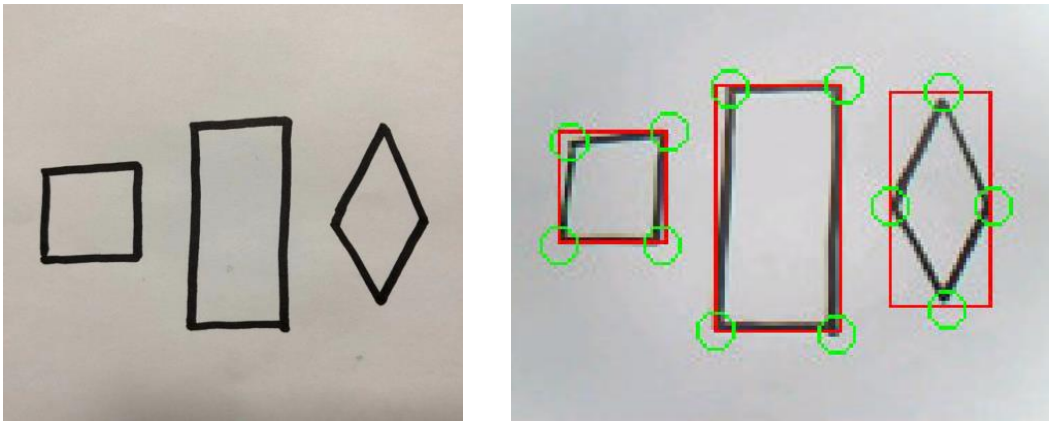


图 24-28 矩形识别实验：原图（左）和实验图片（右）

- **总结：**

本节学习了矩形识别，可以使用该例程来识别矩形图案或者相关物体。

24.3.6 特征点识别

- **前言：**

在本章节中我们前面学习了几个常见的特征形状识别，都是检测跟预设图案特征来对比，而本节会有点区别，本节是先对采集到的图案或物体进行学习识别，然后当摄像头重新铺抓到跟学习对象一样的图片，就标识出来。

也就是说有一个先学习对象物体，再重新识别的过程，理论上这个方式能识别任何物体。

- **实验平台：**

pyAI-OpenMV4。



图 24-29 pyAI-OpenMV4

- **实验目的：**

编程实现学习和记录图像的特征点，当识别到相关特征点后用方框指示出来。

- **实验讲解：**

OpenMV4 集成了特征点识别 find_keypoints 函数，位于 image 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下识别函数相关说明，具体如下：

构造函数
image.find_keypoints([*roi*[, *threshold=20*[, *normalized=False*[, *scale_factor=1.5*[, *max_keypoints=100*[, *corner_detector=image.CORNER_AGAST*]]]]]])
特征点识别函数。返回一个 image.rect 矩形对象列表； 【roi】识别区域（x,y,w,h），未指定则默认整张图片； 【threshold】控制提取的数量的数字（取值 0-255）。对于默认的 AGAST 角点检测器，该值应在 20 左右。 对于 FAST 角点检测器，该值约为 60-80。阈值越低，您提取的角点越多； 【normalized 】布尔值。若为 True，在多分辨率下关闭提取键点。 若您不关心处理扩展问题，且希望算法运行更快，就将之设置为 True。 【scale_factor】是一个必须大于 1.0 的浮点数。较高的比例因子运行更快，但其图像匹配相应较差。理想值介于 1.35-1.5 之间。 【max_keypoints】是一个键点对象所能容纳的键点最大数量。若键点对象过大可能会导致内存问题，需要降低该值。 *更多参数说明请阅读 OpenMV 官方文档： 文档链接：http://docs.openmv.io/library/omv.image.html
使用方法
直接调用该函数。**（仅支持灰度图像）**

表 24-8 特征点识别函数

image 同时包含了特征点对比函数 match_descriptor，用于比较 2 个特征点的相似程度，也就是判断是否一样，说明如下：

构造函数
image.match_descriptor(*descritor0*, *descriptor1*[, *threshold=70*[, *filter_outliers=False*]])

特征点对比函数。返回 kptmatch 对象； 【descritor0】，【descriptor1】：要对比的 2 个特征点； 【threshold】：控制匹配程度。
使用方法
直接调用该函数。

表 24-9　特征点对比函数

从上表可以知道 image.match_descriptor()函数返回的是 kptmatch 对象，对象说明如下：

构造函数
image.kptmatch()
特征点对象。由 image.match_descriptor()函数返回
使用方法
kptmatch.rect()
返回特征点边界框。是一个矩形元组（x,y,w,h）
kptmatch.cx()
返回特征点中心 x 坐标位置（int）。也可以通过索引[0]取得这个值
kptmatch.cy()
返回特征点中心 y 坐标位置（int）。也可以通过索引[1]取得这个值
*更多参数说明请阅读 OpenMV 官方文档： http://docs.openmv.io/library/omv.image.html#class-kptmatch-keypoint-object

表 24-10　特征点对象

可以看到尽管特征识别相对于之前的实验变得复杂，但是由于 MicroPython 面向对象编程的优势，使用起来并没有复杂很多。依然是直接使用库。编程思路如下：

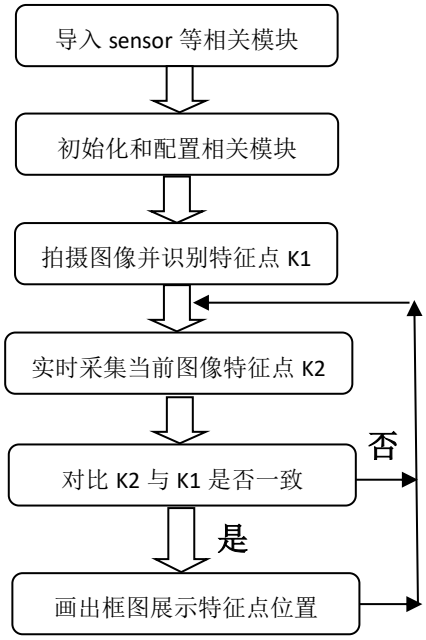


图 24-30 代码编写流程

官方参考代码路径【示例→Feature-Detection→find_keypoints.py】，具体如下：

```
#通过特征点识别对象例程
#
#给摄像头展示一个对象，然后运行程序，程序将提取一组特征点。
#提取完成后将将自动跟踪设备对象。如果需要重新识别请重新运行程序。
#注意：可以在文档中查看 find_keypoints 和 match_keypoints 两个调优参数。
#
#翻译：01Studio

import sensor, time, image

# 重启摄像头
sensor.reset()

# 摄像头初始化配置
sensor.set_contrast(3)
```

```
sensor.set_gainceiling(16)
sensor.set_framesize(sensor.VGA)
sensor.set_windowing((320, 240))
sensor.set_pixformat(sensor.GRAYSCALE)

sensor.skip_frames(time = 2000)
sensor.set_auto_gain(False, value=100)

def draw_keypoints(img, kpts):
    if kpts:
        print(kpts)
        img.draw_keypoints(kpts)
        img = sensor.snapshot()
        time.sleep(1000)

kpts1 = None
# 提示: 也可以使用下面语句从文件导入特征点。
#kpts1 = image.load_descriptor("/desc.orb")
#img = sensor.snapshot()
#draw_keypoints(img, kpts1)

clock = time.clock()

while (True):

    clock.tick()
    img = sensor.snapshot()
    if (kpts1 == None):
        # 提示: 默认情况下 find_keypoints 会返回多种尺寸的特征点，识别灵活。
        kpts1 = img.find_keypoints(max_keypoints=150, threshold=10,
```

```
                                    scale_factor=1.2)
        draw_keypoints(img, kpts1)


    else:
# 提示: 当提取关键字以匹配第一个描述符时，我们使用 normalization =True
#方式来提取。关键点只来自第一个刻度，它将匹配第一个描述符中的一个刻度。
        kpts2 = img.find_keypoints(max_keypoints=150, threshold=10,
                                    normalized=True)
        if (kpts2):
            match = image.match_descriptor(kpts1, kpts2, threshold=85)
            if (match.count()>10):
                # >10 判断为特征点一致。用户可以自行调整
                # 画圆和交叉用于展示特征点.
                img.draw_rectangle(match.rect())
                img.draw_cross(match.cx(), match.cy(), size=10)

                print(kpts2, "matched:%d dt:%d"%(match.count(),
                                                 match.theta()))
                # 提示: 如果你想绘制观点的，请取消下一行语句的注释
                #img.draw_keypoints(kpts2, size=KEYPOINTS_SIZE, matched=True)

    # Draw FPS
    img.draw_string(0, 0, "FPS:%.2f"%(clock.fps()))
```

- **实验结果：**

将摄像头正对要识别的对象，在 IDE 中运行代码，可以看到 OpenMV4 很快就识别了特征点 K1 并在 IDE 上显示。

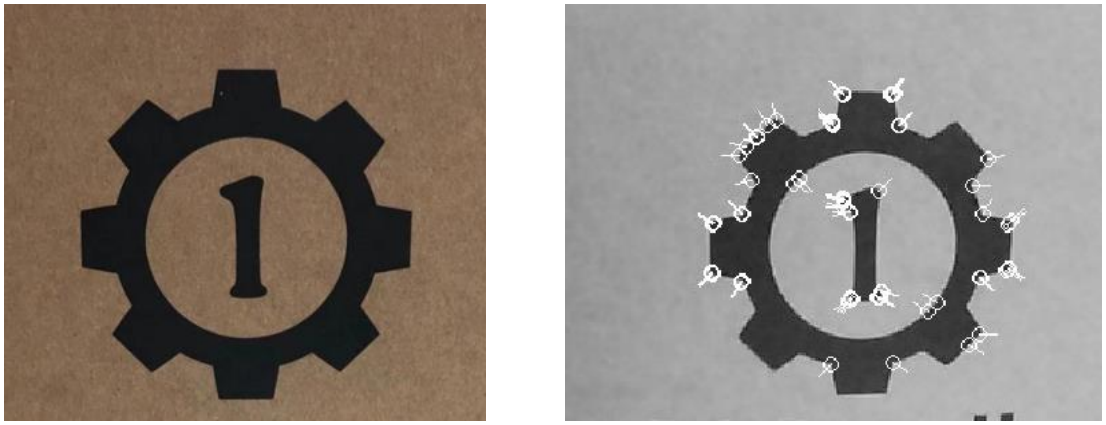


图 24-31 原图（左）和特征点学 K1 学习（右）

学习完成后，我们将摄像头朝向不同的物体，可以看到当摄像头拍摄到我们学习的特征点 K1 时候，会通过矩形框和十字画图方式展示出来。

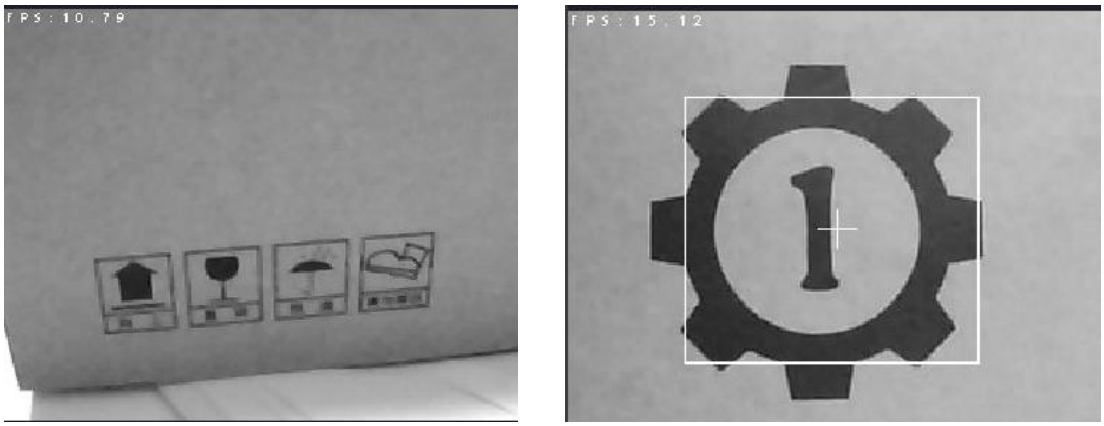


图 24-32 无效识别（左）与有效识别（右）

- **总结：**

通过使用特征点学习函数和特征点对比函数，我们可以非常快速实现特征点的识别。

24.3.7 快速线性回归（巡线）

- **前言：**

快速线性回归的用途非常广泛，如比赛经常用到的小车、机器人巡线，可以通过线性回归的方式判断虚线和实线的轨迹。从而做出判断和响应。本节我们就来学习一下 OpenMV4 线性回归的编程方法。

- **实验平台：**

pyAI-OpenMV4。



图 24-33 pyAI-OpenMV4

- **实验目的：**

编程实现快速线性回归。

- **实验讲解：**

OpenMV4 集成了快速线性回归 get_regression 函数，位于 image 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下函数相关说明，具体如下：

构造函数
image.get_regression(*thresholds*[, *invert=False*[, *roi*[, *x_stride=2*[, *y_stride=1*[, *area_threshold=10*[, *pixels_threshold=10*[, *robust=False*]]]]]]])
线性回归计算。对图像所有阈值像素进行线性回归计算，通过最小二乘法进行，通常速度较快，但不能处理任何异常值； 【thresholds】 必须是元组列表。 (lo, hi) 定义你想追踪的颜色范围。对于灰度图像，每个元组需要包含两个值：最小灰度值和最大灰度值。 *更多参数说明请阅读 OpenMV 官方文档： 文档链接：http://docs.openmv.io/library/omv.image.html
使用方法
直接调用该函数。

表 24-11　线性回归函数

为了提高处理效果，我们可以先将图像变成二值（黑白）图像。方法如下：

构造函数
image.binary(*thresholds*[, *invert=False*[, *zero=False*[, *mask=None*[, *to_bitmap=False*[, *copy=False*]]]]])
图像二值（黑白）化。 【thresholds】 必须是元组列表。 (lo, hi) 定义你想追踪的颜色范围。对于灰度图像，每个元组需要包含两个值：最小灰度值和最大灰度值。 **例：thresholds=(0,100) ，则该函数表示将(0,100)灰度值范围变成白色。** *更多参数说明请阅读 OpenMV 官方文档： 文档链接：http://docs.openmv.io/library/omv.image.html
使用方法
直接调用该函数。

表 24-12　图像二值化函数

编程思路如下：

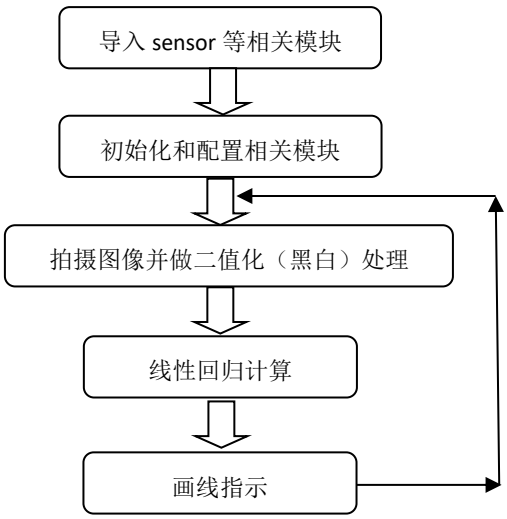


图 24-34 代码编写流程图

官方参考代码路径【示例→Feature-Detection→ linear_regression_fast.py】，具体如下：

```
# 快速线性回归例程
#
#本例程展示了如何在 OpenMV Cam 上使用 get_regression()方法
#得到 ROI 的线性回归。使用此方法可以轻松构建
#一种机器人，它可以跟踪所有方向相同的线，在适当的线上使用 find_blobs() ，以获得更好的过滤选项和控制。
#
#这方法被称为快速线性回归，因为我们使用了最小二乘
#方法来拟合直线。但是，这种方法并非适用于任何图像
#有很多异常点会破坏线的识别。
#
#翻译：01Studio

THRESHOLD = (0, 100)  # 黑白图像的灰度阈值
```

```
BINARY_VISIBLE = True # 使用二值化图像你可以看到什么是线性回归。
                      # 这可能降低 FPS（每秒帧数）.

import sensor, image, time

sensor.reset()
sensor.set_pixformat(sensor.GRAYSCALE)
sensor.set_framesize(sensor.QQVGA)
sensor.skip_frames(time = 2000)
clock = time.clock()

while(True):

    clock.tick()

    #image.binary([THRESHOLD])将灰度值在 THRESHOLD 范围变成了白色
    img = sensor.snapshot().binary([THRESHOLD]) if BINARY_VISIBLE else
                                                    sensor.snapshot()

    # 返回一个类似 find_lines() 和 find_line_segments()的对象.
    # 有以下函数使用方法： x1(), y1(), x2(), y2(), length(),
    # theta() (rotation in degrees), rho(), and magnitude().
    #
    # magnitude() 代表线性回归的指令，其值为(0, INF]。
    # 0 表示一个圆，INF 数值越大，表示线性拟合的效果越好。

    line = img.get_regression([(255,255) if BINARY_VISIBLE else
                                   THRESHOLD])

    if (line): img.draw_line(line.line(), color = 127)
```

```
    print("FPS %f, mag = %s" % (clock.fps(), str(line.magnitude()) if
                                                (line) else "N/A"))


# 有关 rho 负值说明:
#  [theta+0:-rho] 元组和  [theta+180:+rho]是一样的.
```

- **实验结果：**

运行代码，我们分别来测试一下实线和虚线线性回归效果，结果如下：

虚线测试：



图 24-35 虚线直线测试原图（左）和实验图片（右）

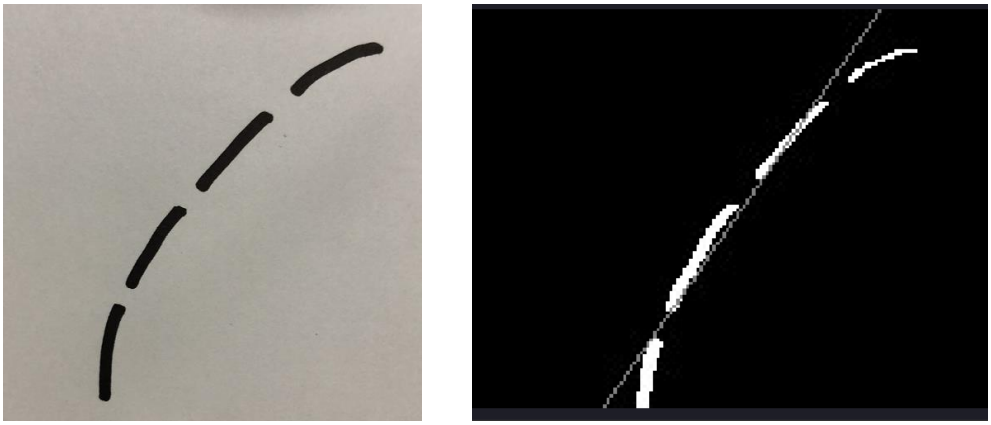


图 24-36 虚线曲线测试原图（左）和实验图片（右）

实线测试：

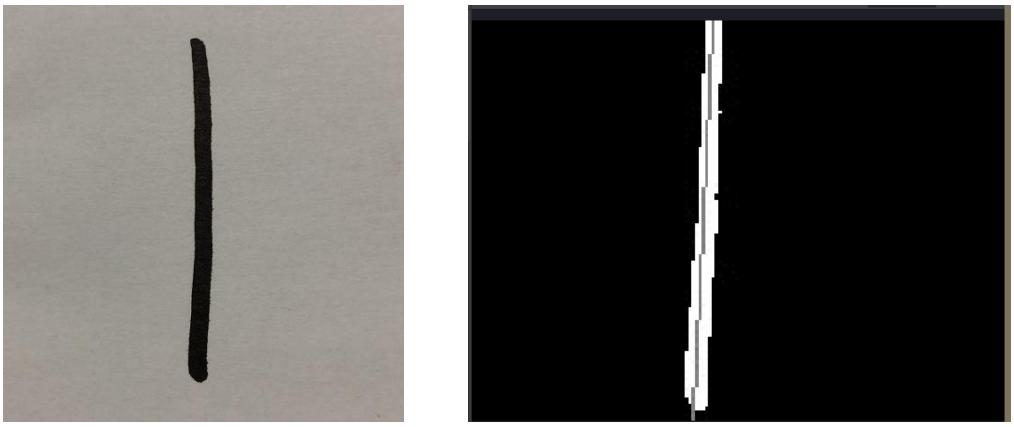


图 24-37 实线直线测试原图（左）和实验图片（右）

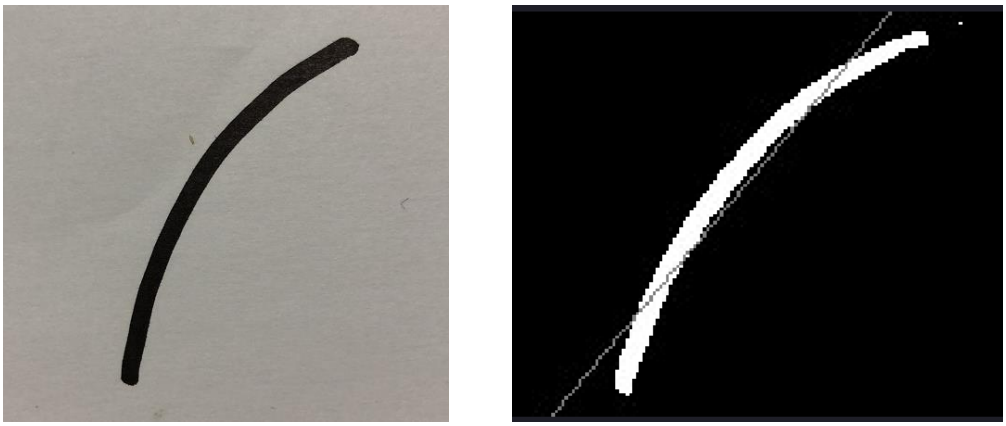


图 24-38 实线曲线测试原图（左）和实验图片（右）

- 总结：

可以看到通过 MicroPython 编程实现的线性回归实验非常容易，效果也是非常不错。

24.4 颜色追踪

颜色追踪，要做的就是对图像指定的颜色进行检测识别，如本节包含的 RGB565 彩色自动追踪、机器人巡线、颜色识别等都是非常有代表性的实验。特征检测的全部官方例程位于 Color-Tracking 例程下，本节挑选重点实验进行讲解。

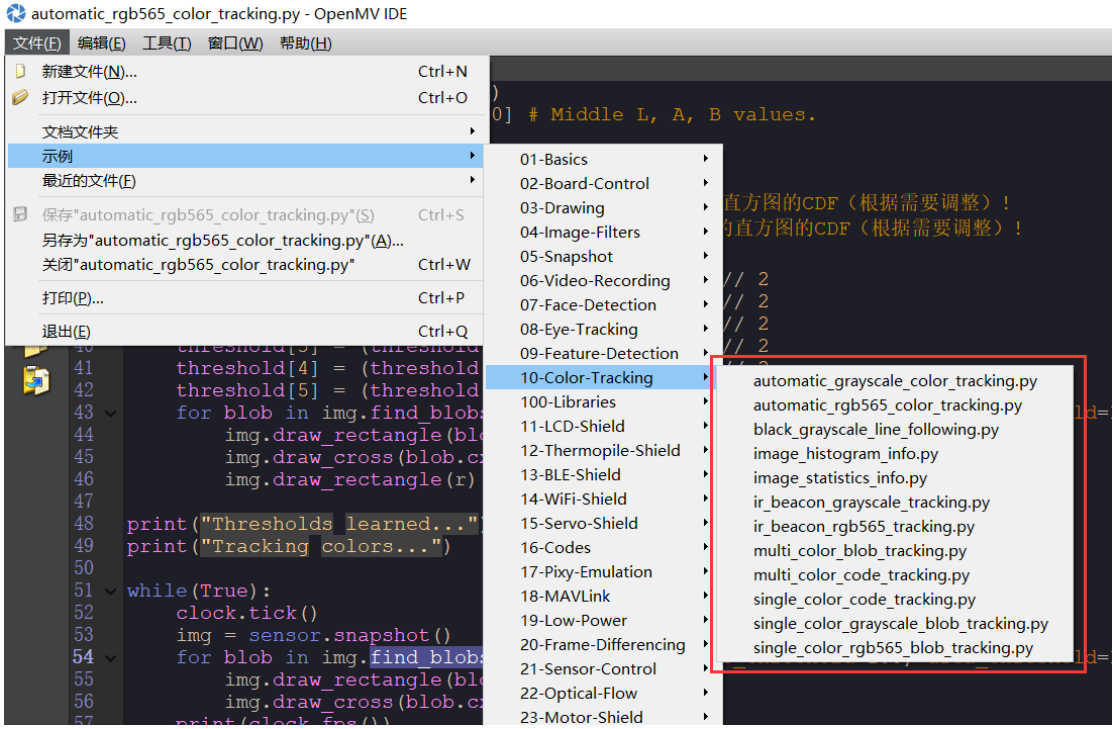


图 24-39 特征检测官方例程位置

24.4.1 RGB565 彩色自动追踪

- **前言：**

本节讲述颜色识别，我们会让 OpenMV4 先学习一种摄像头上拍摄到的颜色，然后再找出新拍摄图像中所有相同颜色的色块。这可以说让摄像头从色盲到开始认识颜色的过程，非常有趣。

- **实验平台：**

pyAI-OpenMV4。



图 24-40 pyAI-OpenMV4

- **实验目的：**

通过编程实现 OpenMV4 识别图像中学习到的颜色色块。

- **实验讲解：**

OpenMV4 集成了 RGB565 颜色块识别 find_blobs 函数，主要是基于 LAB 颜色模型（有兴趣的用户可以自行查阅相关模型资料）。其位于 image 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下本实验相关对象和函数说明，具体如下：

构造函数
*image.find_blobs(*thresholds*[,* invert=False*[,* roi*[,* x_stride=2*[,* y_stride=1*[,* area_threshold=10*[,* pixels_threshold=10*[,* merge=False*[,* margin=0*[,* threshold_cb=None*[,* merge_cb=None*]]]]]]]]]]*)
查找图像中指定的色块。返回 image.blog 对象列表； 【thresholds】 必须是元组列表。 [(lo, hi), (lo, hi), ..., (lo, hi)] 定义你想追踪的颜色范围。 对于灰度图像，每个元组需要包含两个值 - 最小灰度值和最大灰度值。 仅考虑落在这些阈值之间的像素区域。 对于 RGB565 图像，每个元组需要有六个值(l_lo，l_hi，a_lo，a_hi，b_lo，b_hi) - 分别是 LAB L，A 和 B 通道的最小值和最大值。 【area_threshold】若色块的边界框区域小于此参数值，则会被过滤掉； 【pixels_threshold】若色块的像素数量小于此参数值，则会被过滤掉； 【merge】若为 True,则合并所有没有被过滤的色块； 【margin】调整合并色块的边缘。 *更多参数说明请阅读 OpenMV 官方文档： 文档链接：http://docs.openmv.io/library/omv.image.html
使用方法
以上函数返回 image.blob。
blob.rect()
返回一个矩形元组（x,y,w,h）,如色块边界。
blob.cx()
返回色块(int)的中心 x 位置。
blob.cy()
返回色块(int)的中心 y 位置。
*更多使用说明请阅读 OpenMV 官方文档： 文档链接：http://docs.openmv.io/library/omv.image.html#class-blob-blob-object

表 24-13 find_blobs 函数说明

了解了找色块函数应用方法后，我们可以理清一下编程思路，上电后先让OpenMV4 学习摄像头当前图像指定小区域的颜色，学习完成后，进入循环检测，当检测到新的图像中色块颜色跟所学习色块颜色一致时，用矩形和十字交叉表示出来，从而实现颜色追踪。代码编写流程如下：

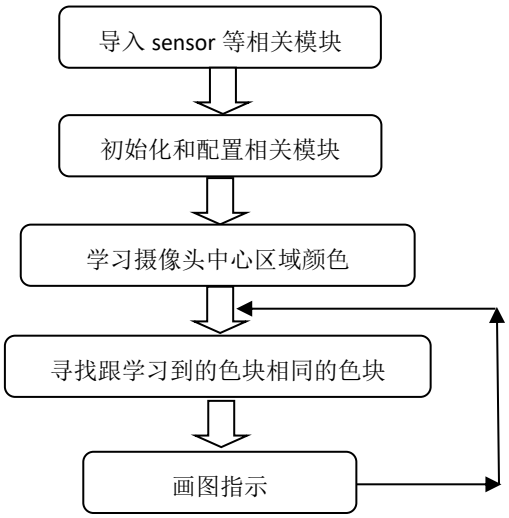


图 24-41 代码编写流程图

官方参考代码路径【示例→Color-Tracking→automatic_rgb565_color_tracking.py】，具体如下：

```
# RGB565 彩色自动追踪
#
# 本例程展示了单个颜色追踪.
#
#翻译：01Studio

import sensor, image, time
print("Letting auto algorithms run. Don't put anything in front of the
camera!")
```

```
#摄像头初始化
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 2000)
sensor.set_auto_gain(False) # 颜色追踪必须关闭摄像头的自动增益
sensor.set_auto_whitebal(False) # 颜色追踪必须关闭自动白平衡
clock = time.clock()

#捕获摄像头中心区域的颜色阈值.50*50 中心区域。
r = [(320//2)-(50//2), (240//2)-(50//2), 50, 50]

#将需要识别的颜色放在摄像头中心
print("Auto algorithms done. Hold the object you want to track in front
      of the camera in the box.")
print("MAKE SURE THE COLOR OF THE OBJECT YOU WANT TO TRACK IS FULLY
       ENCLOSED BY THE BOX!")
for i in range(60):
    img = sensor.snapshot()
    img.draw_rectangle(r)

print("Learning thresholds...")
threshold = [50, 50, 0, 0, 0, 0] # Middle L, A, B values.
for i in range(60):
    img = sensor.snapshot()
    hist = img.get_histogram(roi=r)
   #  获取 1%范围的直方图的 CDF（根据需要调整）!
    lo = hist.get_percentile(0.01)

   #  获取 99%范围的直方图的 CDF（根据需要调整）!
```

```
    hi = hist.get_percentile(0.99)

    # 百分位数平均值.
    threshold[0] = (threshold[0] + lo.l_value()) // 2
    threshold[1] = (threshold[1] + hi.l_value()) // 2
    threshold[2] = (threshold[2] + lo.a_value()) // 2
    threshold[3] = (threshold[3] + hi.a_value()) // 2
    threshold[4] = (threshold[4] + lo.b_value()) // 2
    threshold[5] = (threshold[5] + hi.b_value()) // 2
    for blob in img.find_blobs([threshold], pixels_threshold=100,
                        area_threshold=100, merge=True, margin=10):
        img.draw_rectangle(blob.rect())
        img.draw_cross(blob.cx(), blob.cy())
        img.draw_rectangle(r)

print("Thresholds learned...")
print("Tracking colors...")

while(True):
    clock.tick()
    img = sensor.snapshot()

    #跟学习到的色块 LAB 模型[Threshold]做对比
    for blob in img.find_blobs([threshold], pixels_threshold=100,
                        area_threshold=100, merge=True, margin=10):
        img.draw_rectangle(blob.rect())
        img.draw_cross(blob.cx(), blob.cy())
    print(clock.fps())
```

- **实验结果：**

运行代码，当摄像头中心出现小矩形区域时候，将区域对准要学习的颜色色块，这里学习黄色：



图 24-42 学习黄色区域

注意学习过程需要保存摄像头不动，直到今日检测判断模式，串口出现打印 FPS（每秒帧数），如下图。

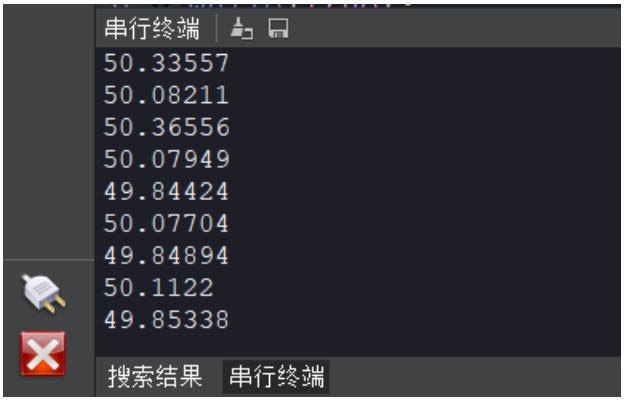


图 24-43

学习完成后，我们将摄像头拍照相关颜色物体，可以看到检测到学习的色块（黄色）。具体如下：

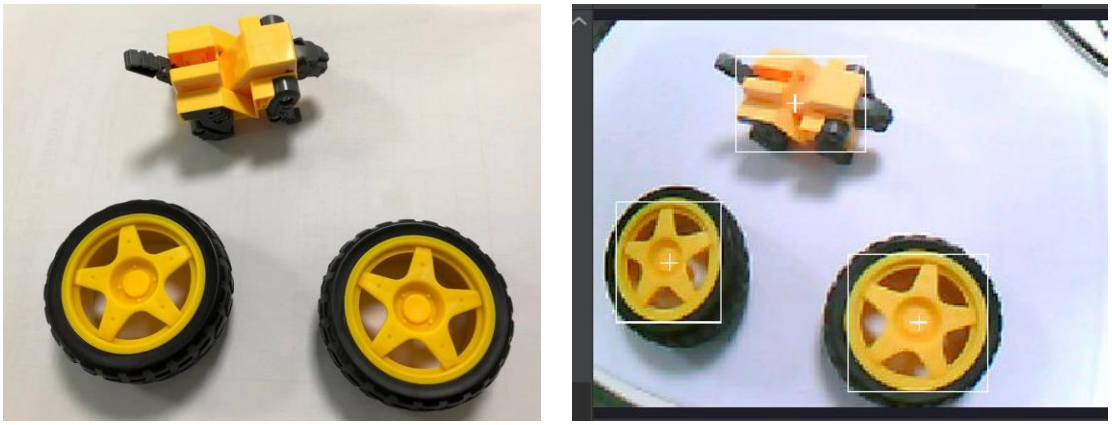


图 24-44 色块追踪 原图（左）和实验图片（右）

- **总结：**

本节学习了通过 MicroPython 编程在 OpenMV4 上实现色块追踪。本实验主要是基于 LAB 颜色模型来判断。有兴趣的小伙伴可以自行查阅 LAB 模块相关资料，再结合打印 threshold 查看其元组数据来深入学习。

24.4.2 机器人巡线（实线巡线）

- **前言：**

之前我们做过巡线实验，是利用线性回归方法测量的。本节我们将学习一个新方法，那就是直接根据摄像头采集到的图像直线与中心偏离的位置计算出偏离角度，这个方法让选线变得更容易。如果你的 OpenMV4 连接到机器人（或相关设备），那么机器人（设备）可以直接关键计算角度偏离结果做出相应调整。

- **实验平台：**

pyAI-OpenMV4。



图 24-45 pyAI-OpenMV4

- **实验目的：**

编程实现 OpenMV4 计算画面中黑线的偏离角度。

- **实验讲解：**

本实验对画面是有一定要求的，也就是摄像头采集图像一定要出现唯一一条连续的直线。程序通过对画面切割成三部分，计算每个部分黑色线的中心点 X 坐标，然后采用加权平均算法估算出直线的偏离位置。通常情况下越靠近底部的地方离摄像头越近，顶部表示远方线段。因此底部的图形权重高。以下下是示意图

讲解：

假设摄像头当前画面的像素是 180（宽）X160（高），左上角坐标为（0,0），然后当前出现直线坐标为（80,120）至（160,0）偏右的直线。上中下三个部分的权重分别为 0.1、0.2、0.7（底部图像靠近机器人，权重大，权重总和不一定为 1），我们来计算一下其中心值。

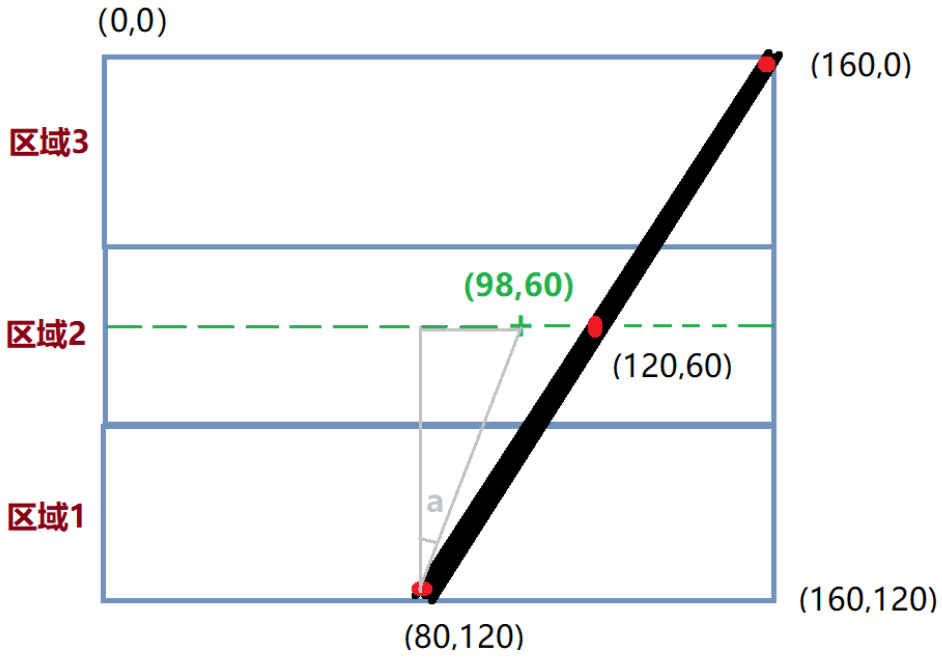


图 24-46 直线偏离计算示意图

我们默认了 Y 轴的中点坐标就是 60，X 坐标加权平均值计算如下：

X'=（80*0.7+120*0.3+160*0.1）/（0.7+0.3+0.1）=98

那么直线偏离坐标可以认为是（98,60），图中绿色“+”位置。那么利用反正切函数可以求出偏离角度：a = atan((98-80)/60)=16.7°，机器人相当于实线的位置往左偏了，所以加一个负号，即 -16.7°；偏离角度就是这么计算出来的。

本实验主要采用 find_blobs 函数编程，这个函数在上一节已经有讲述，具体使用方法如下表：

构造函数
*image.find_blobs(*thresholds*[,* invert=False*[,* roi*[,* x_stride=2*[,* y_stride=1*[,* area_threshold=10*[,* pixels_threshold=10*[,* merge=False*[,* margin=0*[,* threshold_cb=None*[,* merge_cb=None*]]]]]]]]]]*)*

查找图像中指定的色块。返回 image.blog 对象列表； 【thresholds】 必须是元组列表。 [(lo, hi), (lo, hi), ..., (lo, hi)] 定义你想追踪的颜色范围。 对于灰度图像，每个元组需要包含两个值 - 最小灰度值和最大灰度值。 仅考虑落在这些阈值之间的像素区域。 对于 RGB565 图像，每个元组需要有六个值(l_lo，l_hi，a_lo，a_hi，b_lo，b_hi) - 分别是 LAB L，A 和 B 通道的最小值和最大值。 【area_threshold】若色块的边界框区域小于此参数值，则会被过滤掉； 【pixels_threshold】若色块的像素数量小于此参数值，则会被过滤掉； 【merge】若为 True,则合并所有没有被过滤的色块； 【margin】调整合并色块的边缘。 *更多参数说明请阅读 OpenMV 官方文档： 文档链接：http://docs.openmv.io/library/omv.image.html
使用方法
以上函数返回 image.blob。
blob.rect()
返回一个矩形元组（x,y,w,h）,如色块边界。
blob.cx()
返回色块(int)的中心 x 位置。
blob.cy()
返回色块(int)的中心 y 位置。
*更多使用说明请阅读 OpenMV 官方文档： 文档链接：http://docs.openmv.io/library/omv.image.html#class-blob-blob-object

表 24-14

编程思路如下：

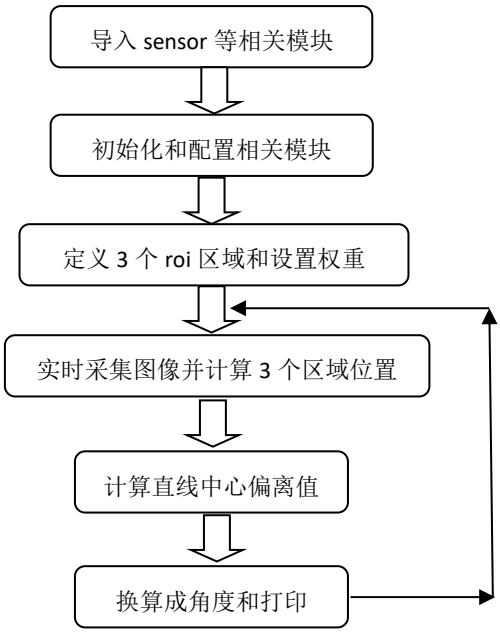


图 24-47 代码编写流程图

官方参考代码路径【示例→Color-Tracking→black_grayscale_line_following.py】，具体如下：

```
# 黑色灰度线巡线跟踪示例
#
#做一个跟随机器人的机器人需要很多的努力。这个示例脚本
#演示了如何做机器视觉部分的线跟随机器人。你
#可以使用该脚本的输出来驱动一个差分驱动机器人
#跟着一条线走。这个脚本只生成一个表示的旋转值（偏离角度）
#你的机器人向左或向右。
#
# 为了让本示例正常工作，你应该将摄像头对准一条直线（实线）
#并将摄像头调整到水平面 45 度位置。请保证画面内只有 1 条直线。
#
#翻译：01Studio


import sensor, image, time, math
```

```
# 追踪黑线。使用 [(128, 255)] 追踪白线.
GRAYSCALE_THRESHOLD = [(0, 64)]

# 下面是一个 roi【区域】元组列表。每个 roi 用 (x, y, w, h)表示的矩形。
# 本例程的采样图像为 160*120，列表把 roi 把图像分成 3 个矩形，越靠近的摄像头
#视野（通常为图像下方）的矩形权重越大。
ROIS = [ # [ROI, weight]
       (0, 100, 160, 20, 0.7), # 可以根据不同机器人情况进行调整。
       (0,  50, 160, 20, 0.3),
       (0,   0, 160, 20, 0.1)
      ]

# 计算以上 3 个矩形的权值【weight】的和，和不需要一定为 1.
weight_sum = 0
for r in ROIS: weight_sum += r[4]  # r[4] 为矩形权重值.

# 摄像头配置
sensor.reset() # 初始化.
sensor.set_pixformat(sensor.GRAYSCALE) # 使用灰度图像.
sensor.set_framesize(sensor.QQVGA) # 使用 QQVGA 分辨率（160*120）.
sensor.skip_frames(time = 2000) # 等待摄像头稳定.
sensor.set_auto_gain(False) # 必须关闭自动增益
sensor.set_auto_whitebal(False) # 必须关闭白平衡
clock = time.clock() # Tracks FPS.

while(True):
    clock.tick() # Track elapsed milliseconds between snapshots().
    img = sensor.snapshot() # 实时拍照.
```

```
centroid_sum = 0


for r in ROIS:
    blobs = img.find_blobs(GRAYSCALE_THRESHOLD, roi=r[0:4],
                           merge=True) # r[0:4] 是上面定义的 roi 元组.


    if blobs:
        # Find the blob with the most pixels.
        largest_blob = max(blobs, key=lambda b: b.pixels())


        # Draw a rect around the blob.
        img.draw_rectangle(largest_blob.rect())
        img.draw_cross(largest_blob.cx(),
                       largest_blob.cy())


        # r[4] 是每个 roi 的权重值.
        centroid_sum += largest_blob.cx() * r[4]


center_pos = (centroid_sum / weight_sum) # 确定直线的中心.


# 将直线中心位置转换成角度，便于机器人处理.
deflection_angle = 0


# 使用反正切函数计算直线中心偏离角度。可以自行画图理解
#权重 X 坐标落在图像左半部分记作正偏，落在右边部分记为负偏，
#所以计算结果加负号。
deflection_angle = -math.atan((center_pos-80)/60)


# 将偏离值转换成偏离角度.
deflection_angle = math.degrees(deflection_angle)
```

```
# 计算偏离角度后可以控制机器人进行调整.
print("Turn Angle: %f" % deflection_angle)


print(clock.fps()) # 连接计算机后 FPS 速度减半。
```

- **实验结果：**

运行程序，分别观察摄像头采集到没偏移、左偏和右偏各个直线的实验结果。

机器人无偏移：

可以看到偏移角度接近 0°；

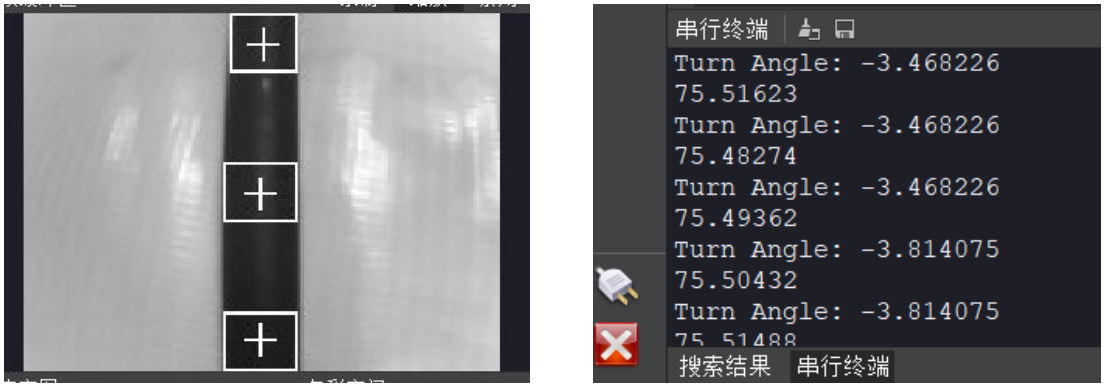


图 24-48 实验图片（左）和偏移角度（右）

机器人左偏：

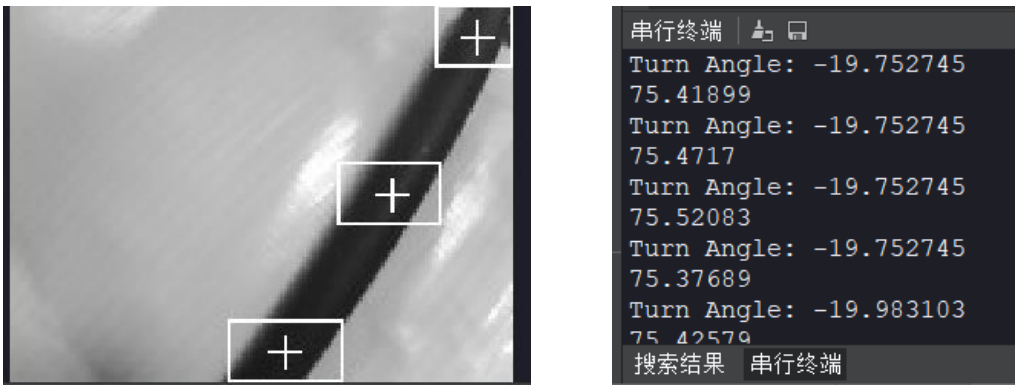


图 24-49 实验图片（左）和偏移角度（右）

机器人右偏：

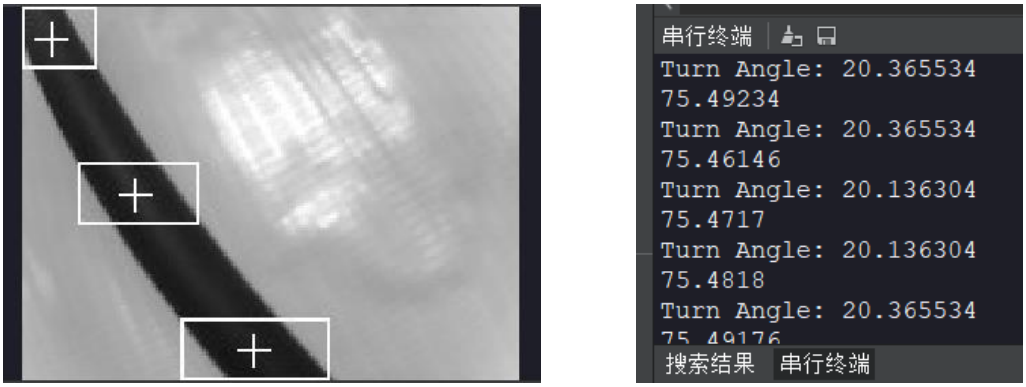


图 24-50 实验图片（左）和偏移角度（右）

- 总结：

从本节可以看到，OpenMV4 算法应用简单便捷，我们通过库可以轻松的实现图像特定的采集方式和计算，以便将精力都放在数学模型上。

24.4.3 单个颜色识别

- **前言：**

我们前面学习了颜色自动追踪，这一节的区别就是指定 OpenMV4 去识别系统预先设定的颜色，如红、绿、蓝。这样当摄像头打开后就能自动识别了。

- **实验平台：**

pyAI-OpenMV4 开发板。



图 24-51 pyAI-OpenMV4

- **实验目的：**

通过编程实现 OpenMV4 识别程序预先设定的颜色色块，分别是红、绿、蓝三种颜色。

- **实验讲解：**

前面已经学习过 OpenMV4 集成了颜色块识别 find_blobs 函数，主要还是基于 LAB 颜色模型（有兴趣的用户可以自行查阅相关模型资料）。其位于 image 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下本实验相关对象和函数说明，具体如下：

构造函数
*image.find_blobs(*thresholds*[,* invert=False*[,* roi*[,* x_stride=2*[,* y_stride=1*[,* area_threshold=10*[,* pixels_threshold=10*[,* merge=False*[,* margin=0*[,* threshold_cb=None*[,* merge_cb=None*]]]]]]]]]]*)
查找图像中指定的色块。返回 image.blog 对象列表； 【thresholds】 必须是元组列表。 [(lo, hi), (lo, hi), ..., (lo, hi)] 定义你想追踪的颜色范围。 对于灰度图像，每个元组需要包含两个值 - 最小灰度值和最大灰度值。 仅考虑落在这些阈值之间的像素区域。 对于 RGB565 图像，每个元组需要有六个值(l_lo，l_hi，a_lo，a_hi，b_lo，b_hi) - 分别是 LAB L，A 和 B 通道的最小值和最大值。 【area_threshold】若色块的边界框区域小于此参数值，则会被过滤掉； 【pixels_threshold】若色块的像素数量小于此参数值，则会被过滤掉； 【merge】若为 True,则合并所有没有被过滤的色块； 【margin】调整合并色块的边缘。 *更多参数说明请阅读 OpenMV 官方文档： 文档链接：http://docs.openmv.io/library/omv.image.html
使用方法
以上函数返回 image.blob。
blob.rect()
返回一个矩形元组（x,y,w,h）,如色块边界。
blob.cx()
返回色块(int)的中心 x 位置。
blob.cy()
返回色块(int)的中心 y 位置。
blob.elongation()
用于块形状识别。返回介于 0-1 的值，0 接近圆、>0.5 可以看成矩形、1 表示线；
blob.min_corners()
始终返回一个矩形边界 4 个（x,y）元组。常用不规则形状补偿成矩形框展示；

blob.major_axis_line()
返回一个线元组(x1,y1,x2,y2)。可以看做 blob.min_corners()的最长边；
blob.minor_axis_line()
返回一个线元组(x1,y1,x2,y2)。可以看做 blob.min_corners()的最短边；
*更多使用说明请阅读 OpenMV 官方文档： 文档链接：http://docs.openmv.io/library/omv.image.html#class-blob-blob-object

表 24-15 find_blobs 函数说明

了解了找色块函数应用方法后，我们可以理清一下编程思路，代码编写流程如下：

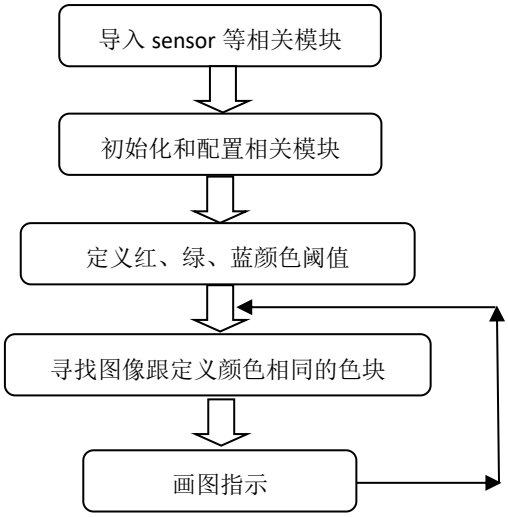


图 24-52 代码编写流程图

官方参考代码路径【示例→Color-Tracking→single_color_rgb565_blog_tracking.py】，具体如下：

```
# 单个颜色块识别例程
#
# 这个例程使用 OpenMV 来识别单种颜色。
#
#翻译：01Studio


import sensor, image, time, math
```

```
threshold_index = 0 # 0 红色, 1 绿色, 2 蓝色

# 颜色识别阈值 (L Min, L Max, A Min, A Max, B Min, B Max) LAB 模型
# 下面的阈值元组是用来识别 红、绿、蓝三种颜色，当然你也可以调整让识别变得更好。
thresholds = [(30, 100, 15, 127, 15, 127), # 红色阈值
              (30, 100, -64, -8, -32, 32), # 绿色阈值
              (0, 30, 0, 64, -128, 0)] # 蓝色阈值

#摄像头初始化
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 2000)
sensor.set_auto_gain(False) # 摄像头必须关闭自动增益
sensor.set_auto_whitebal(False) # 摄像头必须关闭白平衡
clock = time.clock()

# "pixel_threshold" 和 "area_threshold" 参数用于控制返回确定色块的数量。
# "merge=True" 用于合并找到的色块。

while(True):

    clock.tick()
    img = sensor.snapshot()

    for blob in img.find_blobs([thresholds[threshold_index]],
                  pixels_threshold=200, area_threshold=200, merge=True):

        # blob.elongation() > 0.5 来判断颜色块不是圆形的时候.
```

```
        if blob.elongation() > 0.5:
            img.draw_edges(blob.min_corners(), color=(255,0,0))
            img.draw_line(blob.major_axis_line(), color=(0,255,0))
            img.draw_line(blob.minor_axis_line(), color=(0,0,255))

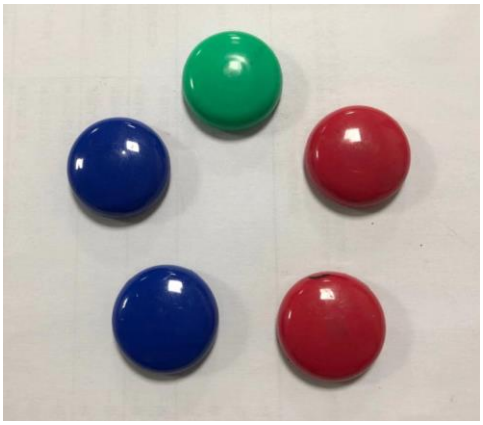
        # 下面这些画图展示一直会出现.
        img.draw_rectangle(blob.rect())
        img.draw_cross(blob.cx(), blob.cy())
        # 颜色块旋转角度只有 0-180 度.
        img.draw_keypoints([(blob.cx(), blob.cy(),
                             int(math.degrees(blob.rotation())))], size=20)
        print(int(math.degrees(blob.rotation())))
    print(clock.fps())
```

- **实验结果：**

我们可以将代码中的 threshold_index 变量修改成 0、1、2 然后运行，这分别代表识别红色、绿色和蓝色。

```
threshold_index = 0 # 0 红色, 1 绿色, 2 蓝色
```

红色识别：



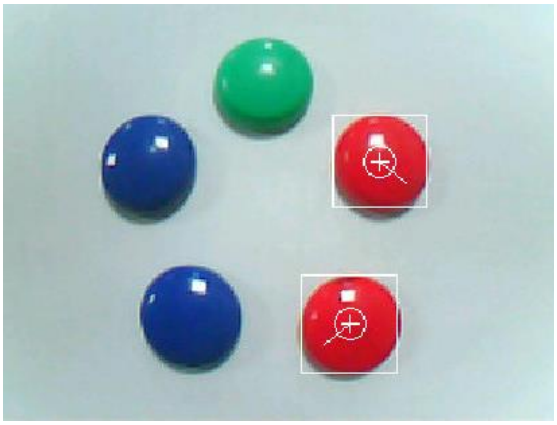


图 24-53 红色识别原图（左）和实验图片（右）

绿色识别：

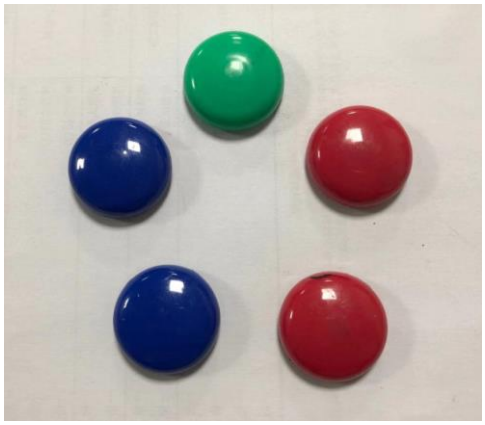
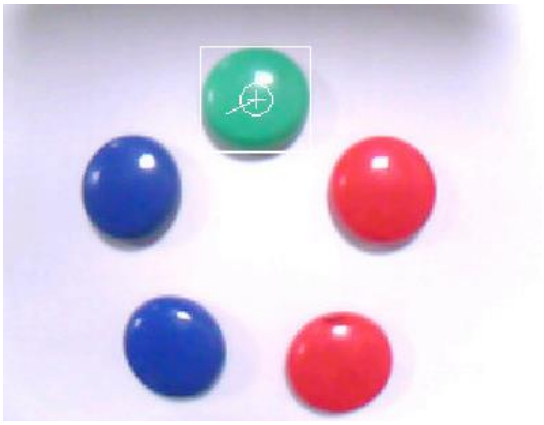



图 24-54 绿色识别原图（左）和实验图片（右）

蓝色识别：

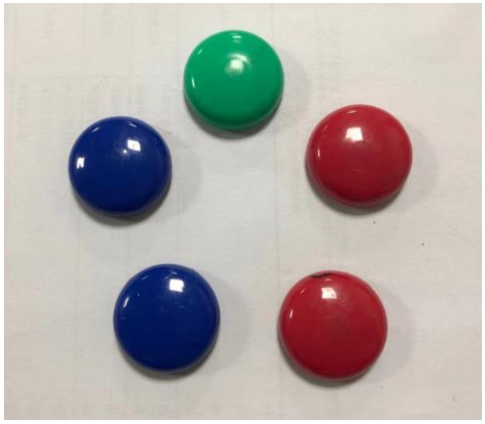
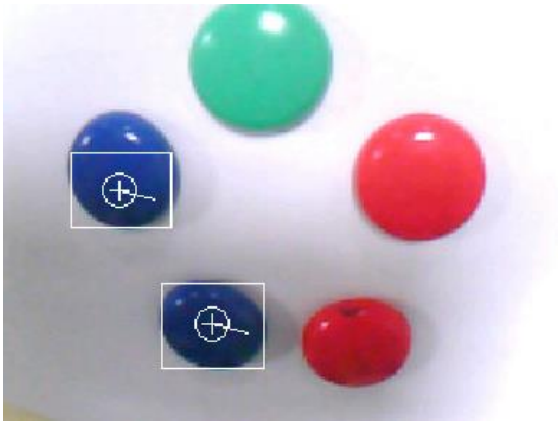



图 24-55 蓝色识别原图（左）和实验图片（右）

- 总结：

本节学习了通过 MicroPython 编程在 OpenMV4 上实现单种颜色识别。本实验主要是基于 LAB 颜色模型来判断。有兴趣的小伙伴可以自行查阅 LAB 模块相关资料，再结合打印 threshold 查看其元组数据来深入学习。

还有就是可以找多种不规则形状的颜色物体对比学习。

24.5 人脸检测

人脸检测类实验主要是人脸检测和人脸特征识别，这是在安防、支付等领域非常流行的应用。全部官方例程位于 Face-Detection 例程下，本节挑选重点实验进行讲解。

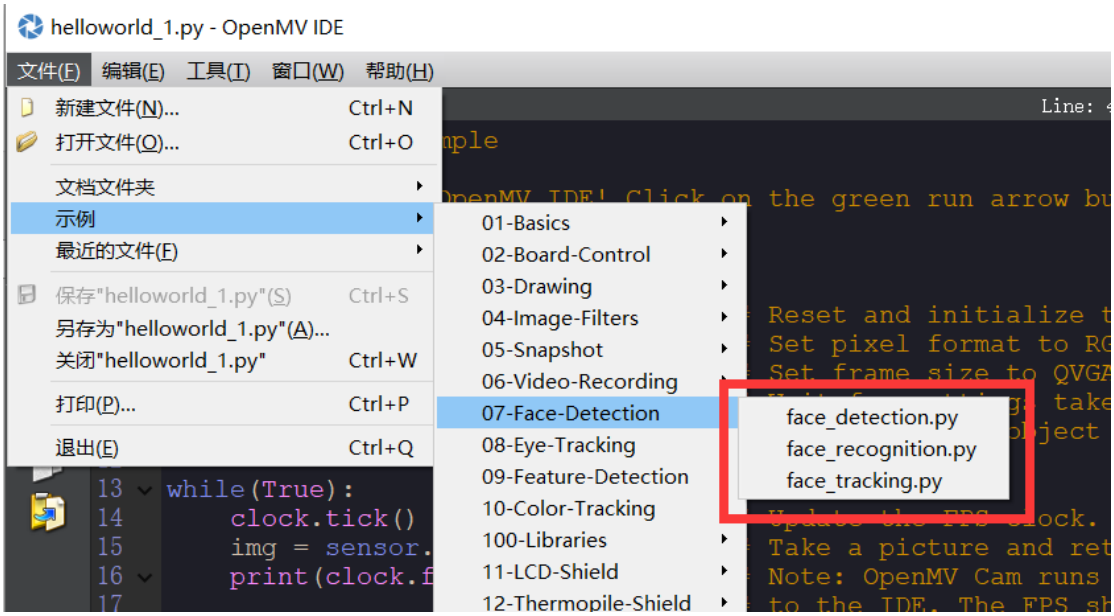


图 24-56 人脸检测官方例程位置

24.5.1 人脸检测

- **前言：**

人脸检测，简单来说就是告诉你将一幅图片中哪些是人脸。今天我们来学习一下如何通过 MicroPython 编程快速实现人脸识别。

- **实验平台：**

pyAI-OpenMV4。



图 24-57 pyAI-OpenMV4

- **实验目的：**

将摄像头拍摄到的画面中的人脸用矩形框表示出来。

- **实验讲解：**

人脸识别本质是特征识别，OpenMV4 已经集成了非常多的特征库和算法库，人生苦短，通过 MicroPython 编程我们可以快速实现人脸识别应用。其使用的是 image 模块下的 find_features()特征寻找函数，说明如下：

构造函数
*image.find_features(*cascade*[, threshold=0.5[, scale=1.5[, roi]]])*
搜索和 Haar Cascade 匹配的所有区域的图像，并返回一个关于这些特征的边界框矩形元组(x，y，w，h)的列表。若未发现任何特征，则返回一个空白列表。 【cascade】 Haar Cascade 对象 【threshold 】是浮点数（0.0-1.0），其中较小的值在提高检测速率同时增加误报率。相反，较高的值会降低检测速率，同时降低误报率。； 【scale】是一个必须大于 1.0 的浮点数。较高的比例因子运行更快，但其图像匹配相应较差。理想值介于 1.35-1.5 之间； 【roi 】指定识别区域的矩形元组(x，y，w，h)。如果未指定，roi 即整个图像的图像矩形。
使用方法
直接调用该函数。建议使用灰度图像。

表 24-16　寻找人脸特征函数

从上表可以看到人脸检测的模型和算法已经在 OpenMV4 里面内置，我们直接调用即可，具体编程思路如下：

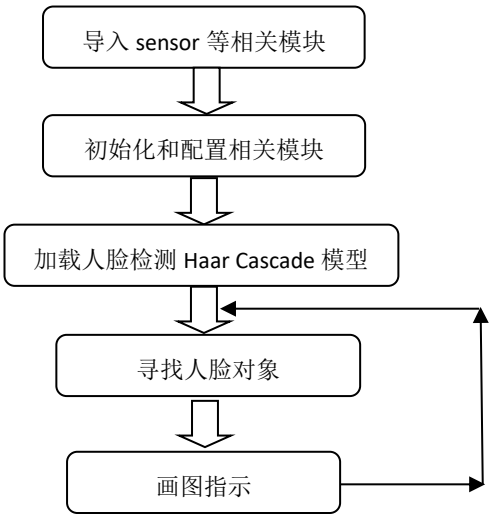


图 24-58 代码编写流程图

官方参考代码路径【示例→Face-Detection→ face_detection.py】，具体如下：

```
# 人脸检测例程
#
# 这个例子展示了 OpenMV 内置的人脸检测功能。
#
# 人脸检测是通过在图像上使用 Haar Cascade 特征检测器来实现的。
# 一个 Haar Cascade 是一系列简单区域的对比检查。
# 人脸识别有 25 个阶段，每个阶段有几百次检测。Haar Cascades 运行很快因为是逐个
# 阶段递进检测的。 此外，OpenMV Cam 使用一种称为积分图像的数据结构来在恒定时间
# 内快速执行每个区域的对比度检查(特征检测需要用灰度图像的原因是因为图像积分需
# 要更多空间)。
#
#翻译和注释：01Studio

import sensor, time, image

# Reset sensor
sensor.reset()

# Sensor settings
sensor.set_contrast(1) #设置相机图像对比度为 1
sensor.set_gainceiling(16) #设置相机图像增益上限为 16
# HQVGA 和 GRAYSCALE 是人脸检测最佳配置.
sensor.set_framesize(sensor.HQVGA)
sensor.set_pixformat(sensor.GRAYSCALE)

# 加载 Haar Cascade 模型
# 默认使用 25 个步骤，减少步骤会加快速度但会影响识别成功率.
face_cascade = image.HaarCascade("frontalface", stages=25)
print(face_cascade)
```

```
# FPS clock
clock = time.clock()

while (True):
    clock.tick()

    # Capture snapshot
    img = sensor.snapshot()

    # 找人脸对象.
    # threshold 和 scale_factor 两个参数控制着识别的效率和准确性.
    objects = img.find_features(face_cascade, threshold=0.75,
                                            scale_factor=1.25)

    # 画图展示对象（将人脸用矩形展示出来）
    for r in objects:
        img.draw_rectangle(r)

    # Print FPS.
    # 注意: 连接电脑会降低 FPS.
    print(clock.fps())
```

- **实验结果：**

运行代码，将摄像头正对自己，可以看到将自己的脸检测出来。我们用图片演示一下单个人脸和多个人脸检测的场景。

单个人脸检测：





图 24-59 单个人脸检测 原图（左）和实验图片（右）

多个人脸识别：



图 24-60 多个人脸检测原图

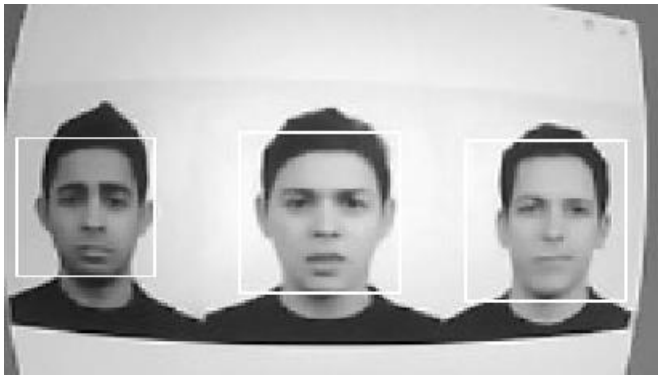


图 24-61 多个人脸检测实验图片

- **总结：**

本节学习了人脸检测，通过 OpenMV4 内置的模型和算法，结合 MicroPython 编程我们轻松完成了实验。而且本实验支持单个和多个人脸同时检测，是机器视觉中非常有代表性的实验。

24.5.2 人脸追踪

- **前言：**

人脸追踪，是通过短时间的人脸特征学习，再重新识别的过程。这节内容是基于上一节人脸检测基础上完成的。今天我们来学习一下如何通过 MicroPython 编程快速实现人脸追踪。

- **实验平台：**

pyAI-OpenMV4。



图 24-62 pyAI-OpenMV4

- **实验目的：**

追踪出学习记录下来的人脸。

- **实验讲解：**

本节是人脸检测和特征点识别的相结合，实验用到的函数和对象之前都有介绍过，这里再次列出来让大家重温一下：

构造函数
*image.find_features(*cascade*[, threshold=0.5[, scale=1.5[, roi]]])*
搜索和 Haar Cascade 匹配的所有区域的图像，并返回一个关于这些特征的边界框矩形元组(x，y，w，h)的列表。若未发现任何特征，则返回一个空白列表。 【cascade】 Haar Cascade 对象 【threshold 】是浮点数（0.0-1.0），其中较小的值在提高检测速率同时增加误报率。相反，较高的值会降低检测速率，同时降低误报率。; 【scale】是一个必须大于 1.0 的浮点数。较高的比例因子运行更快，但其图像匹配相应较差。理想值介于 1.35-1.5 之间; 【roi 】指定识别区域的矩形元组(x，y，w，h)。如果未指定，roi 即整个图像的图像矩形。
使用方法
直接调用该函数。建议使用灰度图像。

表 24-17　寻找人脸特征函数

构造函数
image.find_keypoints([*roi[, threshold=20[, normalized=False[, scale_factor=1.5[, max_keypoints=100[, corner_detector=image.CORNER_AGAST]]]]]])
特征点识别函数。返回一个 image.rect 矩形对象列表; 【roi】识别区域（x,y,w,h），未指定则默认整张图片; 【threshold】控制提取的数量的数字（取值 0-255）。对于默认的 AGAST 角点检测器，该值应在 20 左右。 对于 FAST 角点检测器，该值约为 60-80。阈值越低，您提取的角点越多; 【normalized 】布尔值。若为 True，在多分辨率下关闭提取键点。 若您不关心处理扩展问题，且希望算法运行更快，就将之设置为 True。 【scale_factor】是一个必须大于 1.0 的浮点数。较高的比例因子运行更快，但

其图像匹配相应较差。理想值介于 1.35-1.5 之间。 【max_keypoints】是一个键点对象所能容纳的键点最大数量。若键点对象过大可能会导致内存问题，需要降低该值。 *更多参数说明请阅读 OpenMV 官方文档： 文档链接：http://docs.openmv.io/library/omv.image.html
使用方法
直接调用该函数。（**仅支持灰度图像**）

表 24-18　特征点识别函数

image 同时包含了特征点对比函数 match_descriptor，用于比较 2 个特征点的相似程度，也就是判断是否一样，说明如下：

构造函数
image.match_descriptor(*descritor0, descriptor1*[, *threshold=70*[, *filter_outliers=False*]])
特征点对比函数。返回 kptmatch 对象； 【descritor0】，【descriptor1】：要对比的 2 个特征点； 【threshold】：控制匹配程度。
使用方法
直接调用该函数。

表 24-19　特征点对比函数

从上表可以知道 image.match_descriptor()函数返回的是 kptmatch 对象，对象说明如下：

构造函数
image.kptmatch()
特征点对象。由 image.match_descriptor()函数返回

使用方法
kptmatch.rect()
返回特征点边界框。是一个矩形元组（x,y,w,h）
kptmatch.cx()
返回特征点中心 x 坐标位置（int）。也可以通过索引[0]取得这个值
kptmatch.cy()
返回特征点中心 y 坐标位置（int）。也可以通过索引[1]取得这个值
*更多参数说明请阅读 OpenMV 官方文档： http://docs.openmv.io/library/omv.image.html#class-kptmatch-keypoint-object

表 24-20　特征点对象

以上函数和对象都是我们之前使用过的，对于人脸追踪实验，具体编程思路如下：

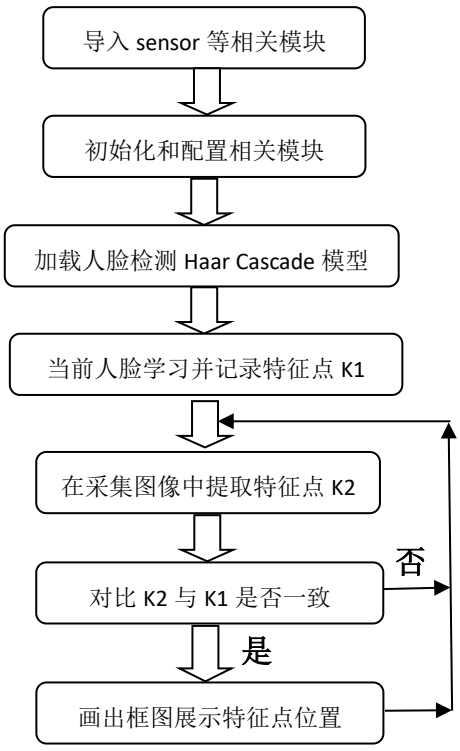


图 24-63 代码编写流程图

官方参考代码路径【示例→Face-Detection→ face_tracking.py】，具体如下：

```
# 人脸追踪例程
#
# 这个例程展示了如何使用关键特征来追踪一个已经使用 Haar Cascade 检测出来的人脸。
# 程序第一阶段先使用 Haar Cascade 找出人脸.然后使用关键特征来学习，最后不停的找这个人脸。
# 关键特征点可以用来追踪任何栋。
#
#翻译：01Studio

import sensor, time, image

# Reset sensor
sensor.reset()
sensor.set_contrast(3)
sensor.set_gainceiling(16)
sensor.set_framesize(sensor.VGA)
#在 VGA（640*480）下开个小窗口，相当于数码缩放。
sensor.set_windowing((320, 240))
sensor.set_pixformat(sensor.GRAYSCALE)

# 延时以便摄像头稳定工作
sensor.skip_frames(time = 2000)

# 加载 Haar Cascade 模型
# 默认使用 25 个步骤，减少步骤会加快速度但会影响识别成功率.
face_cascade = image.HaarCascade("frontalface", stages=25)
print(face_cascade)
```

```
# 特征 kpts1
kpts1 = None

# 找到人脸!
while (kpts1 == None):
    img = sensor.snapshot()
    img.draw_string(0, 0, "Looking for a face...")
    # Find faces
    objects = img.find_features(face_cascade, threshold=0.5, scale=1.25)
    if objects:
        # 将 ROI（x,y,w,h）往各个方向扩展 31 像素
        face = (objects[0][0]-31, objects[0][1]-31,objects[0][2]+31*2,
                objects[0][3]+31*2)
        # 使用扩展后的 ROI 区域（人脸）学习关键点
        kpts1 = img.find_keypoints(threshold=10, scale_factor=1.1,
                                   max_keypoints=100, roi=face)
        # 用矩形框展示人脸
        img.draw_rectangle(objects[0])

# 打印关键点
print(kpts1)
img.draw_keypoints(kpts1, size=24)
img = sensor.snapshot()
time.sleep(2000) #暂停以便观察特征

# FPS clock
clock = time.clock()

while (True):
    clock.tick()
```

```
img = sensor.snapshot()
# 从图像中提取关键点
kpts2 = img.find_keypoints(threshold=10, scale_factor=1.1,
                           max_keypoints=100, normalized=True)

if (kpts2):
    # 跟关键点 kpts1 匹配
    c=image.match_descriptor(kpts1, kpts2, threshold=85)
    match = c[6] # C[6] 为 matches 值，这个值越大表示匹配程度越高.
    if (match>5): #设置当大于 5 的时候为匹配成功，并画图标示。打印信息。
        img.draw_rectangle(c[2:6])
        img.draw_cross(c[0], c[1], size=10)
        print(kpts2, "matched:%d dt:%d"%(match, c[7]))

# Draw FPS
img.draw_string(0, 0, "FPS:%.2f"%(clock.fps()))
```

- **实验结果：**

运行代码，将摄像头正对自己学习，学习完成后移动摄像头，当再次拍到自己的时候会识别出来。

我们用图片演示一下人脸追踪实验。先学习人脸特征：



图 24-64 人脸学习图片 原图（左）和特征提取图（右）

学习后保存程序运行，拍摄一下干扰图片，发现没有成功匹配：**（有时候由于光线问题错误匹配，但可信度 match 值也不高。）**



图 24-65 干扰图片（左）和无法匹配（右）

再次拍摄原来的学习图片，发现轻松成功匹配。

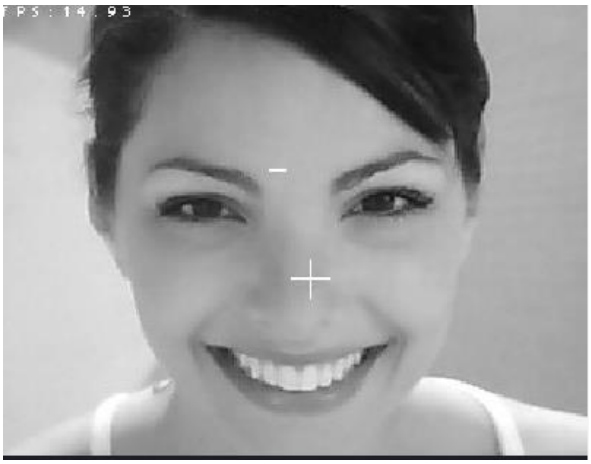


图 24-66 匹配成功

观察串口，发现匹配程度也非常高。matched 值大于 10。

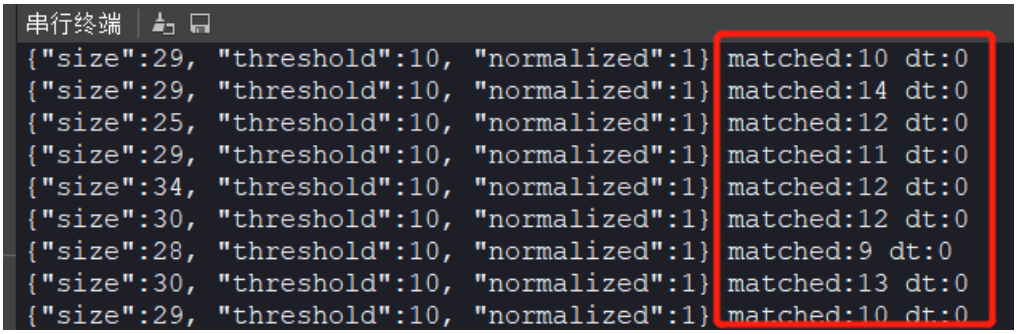


图 24-67

- **总结：**

本节学习了人脸追踪，本质是人脸检测和特征识别的相结合，因此我们可以加以扩展，指定区域学习相关物体特征，然后就可以追踪任何物体了。

24.6 眼球追踪

眼球追踪类实验主要是眼球检测和瞳孔特征识别追踪，这是在安防、支付等领域非常流行的应用。全部官方例程位于 Eye-Tracking 例程下，本节挑选重点实验进行讲解。

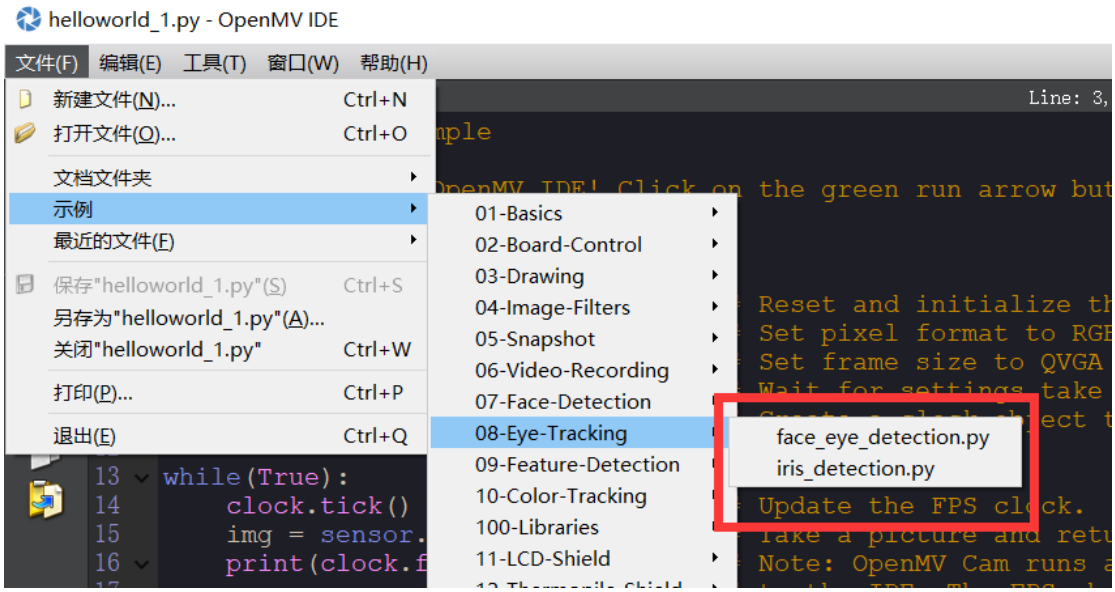


图 24-68 眼球官方例程位置

24.6.1 眼球检测

- **前言：**

前面我们学习过人脸检测，这一节我们来学习更细节的检测，那就是眼球的检测。

- **实验平台：**

pyAI-OpenMV4。



图 24-69 pyAI-OpenMV4

- **实验目的：**

将画面中人脸的眼球用检测出来并用矩形框表示。

- **实验讲解：**

眼球检测是建立在人脸检测的基础上的，我们可以先识别人脸，进而再识别人脸中的人眼，提高识别的效率和准确率。跟人脸特征模型一样，OpenMV4 也已经集成了眼球模型以及其处理函数。通过 MicroPython 编程我们可以快速实现人眼检测应用。本实验使用到的函数和对象说明如下：

构造函数
*image.find_features(*cascade*[,* threshold=0.5*[,* scale=1.5*[,* roi*]]])*
搜索和 Haar Cascade 匹配的所有区域的图像，并返回一个关于这些特征的边界框矩形元组(x，y，w，h)的列表。若未发现任何特征，则返回一个空白列表。 【cascade】 Haar Cascade 对象模型。可以是人脸或者眼球。 【threshold 】是浮点数（0.0-1.0），其中较小的值在提高检测速率同时增加误报率。相反，较高的值会降低检测速率，同时降低误报率。； 【scale】是一个必须大于 1.0 的浮点数。较高的比例因子运行更快，但其图像匹配相应较差。理想值介于 1.35-1.5 之间； 【roi 】指定识别区域的矩形元组(x，y，w，h)。如果未指定，roi 即整个图像的图像矩形。
使用方法
直接调用该函数。建议使用灰度图像。

表 24-21　寻找人脸和眼球特征函数

从上表可以看到人眼检测的模型和算法已经在 OpenMV4 里面内置，我们直接调用即可，具体编程思路如下：

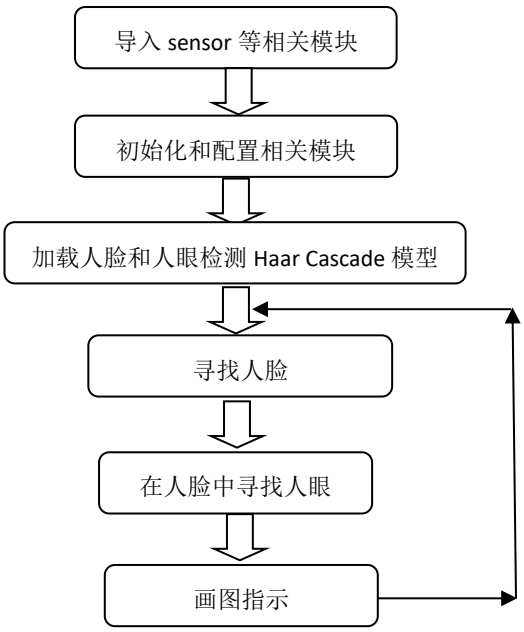


图 24-70 代码编写流程图

官方参考代码路径【示例→Eye-Tracking→ face_eye_detection.py】，具体如下：

```
# 人眼检测例程
#
# 这个程序是先利用人脸检测确定人的脸位置，然后在检测眼球位置。
# 如果你想检测瞳孔，请参考 iris_detection 例程.
#
#翻译和注释：01Studio

import sensor, time, image

# Reset sensor
sensor.reset()

# Sensor settings
sensor.set_contrast(1)
sensor.set_gainceiling(16)
sensor.set_framesize(sensor.HQVGA)
sensor.set_pixformat(sensor.GRAYSCALE)

# 加载 Haar Cascade 模型
# 默认使用全部 stages,提高识别的准确率.
face_cascade = image.HaarCascade("frontalface", stages=25) #定义人脸模型
eyes_cascade = image.HaarCascade("eye", stages=24) #定义眼球模型
print(face_cascade, eyes_cascade)

# FPS clock
clock = time.clock()

while (True):
    clock.tick()
```

```
# Capture snapshot
img = sensor.snapshot()

# 人脸检测
objects = img.find_features(face_cascade, threshold=0.5,
                                  scale_factor=1.5)

# 画出人脸方框
for face in objects:
    img.draw_rectangle(face)
    # 在人脸中寻找眼球.
    # 提示:使用一个较大的 threshold 值 (识别更多) 和 较小的
    # scale 值 (寻找较小区域的对象)
    eyes = img.find_features(eyes_cascade, threshold=0.5,
                                  scale_factor=1.2, roi=face)
    for e in eyes:
        img.draw_rectangle(e)

# Print FPS.
# Note: Actual FPS is higher, streaming the FB makes it slower.
print(clock.fps())
```

- **实验结果：**

运行代码，将摄像头正对自己脸，可以看到将自己眼睛检测出来。注意光线不能太亮。我们用图片演示一下人眼检测实验。

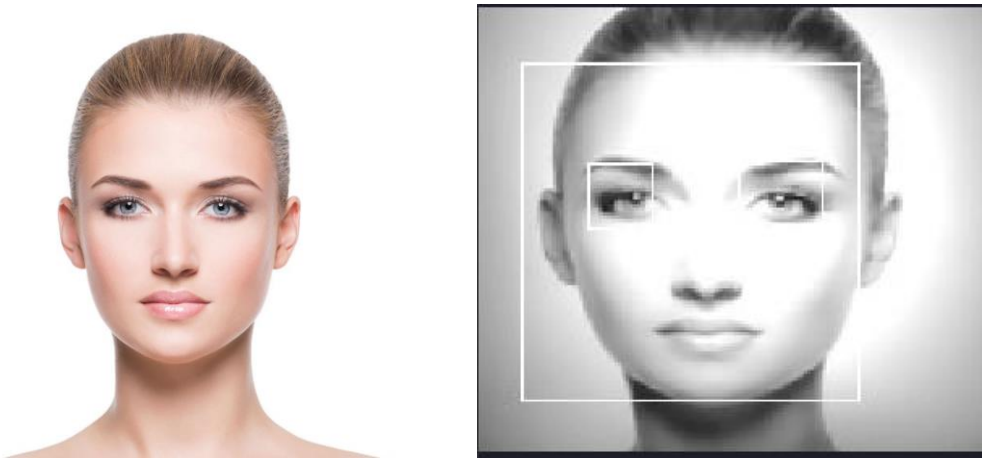


图 24-71 人眼检测 原图（左）和实验图片（右）

- **总结：**

通过本节人眼检测实验学习我们发现人眼和人脸检测原理非常相近，只是选择了不用的模型来判断。本质都是特征识别。

24.6.2 瞳孔追踪

- **前言：**

瞳孔，是动物或人眼睛内虹膜中心的小圆孔，为光线进入眼睛的通道。虹膜上瞳孔括约肌的收缩可以使瞳孔缩小，瞳孔开大肌的收缩使瞳孔散大，瞳孔的开大与缩小控制进入瞳孔的光量。

本节学习瞳孔的追踪。



图 24-72 瞳孔

- **实验平台：**

pyAI-OpenMV4。



图 24-73 pyAI-OpenMV4

- **实验目的：**

对人眼中的瞳孔进行追踪并画图标示。

- **实验讲解：**

瞳孔追踪是建立在眼球检测的基础上的，我们识别眼球后，再利用 OpenMV4 瞳孔追踪函数 find_eys 函数来追踪瞳孔即可。通过 MicroPython 编程我们可以快速实现瞳孔追踪的应用。本实验使用到的函数和对象说明如下：

构造函数
*image.find_features(*cascade*[,* threshold=0.5*[,* scale=1.5*[,* roi*]]])*
搜索和 Haar Cascade 匹配的所有区域的图像，并返回一个关于这些特征的边界框矩形元组(x，y，w，h)的列表。若未发现任何特征，则返回一个空白列表。 【cascade】 Haar Cascade 对象模型。可以是人脸或者眼球。 【threshold 】是浮点数（0.0-1.0），其中较小的值在提高检测速率同时增加误报率。相反，较高的值会降低检测速率，同时降低误报率。; 【scale】是一个必须大于 1.0 的浮点数。较高的比例因子运行更快，但其图像匹配相应较差。理想值介于 1.35-1.5 之间; 【roi 】指定识别区域的矩形元组(x，y，w，h)。如果未指定，roi 即整个图像的图像矩形。
使用方法
直接调用该函数。建议使用灰度图像。

表 24-22　寻找人脸和眼球特征函数

构造函数
*image.find_eye(*roi*)*
寻找瞳孔。返回中心坐标（x,y）。 【roi 】指定识别区域的矩形元组(x，y，w，h)。如果未指定，roi 即整个图像的图像矩形。
使用方法
直接调用该函数。建议使用灰度图像。

表 24-23　寻找瞳孔函数

从上表可以看到人眼和瞳孔检测的模型和算法已经在 OpenMV4 里面内置，我们直接调用即可，具体编程思路如下：

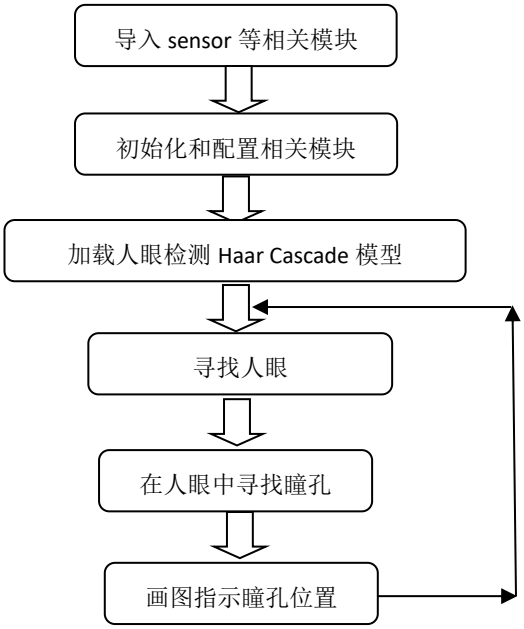


图 24-74 代码编写流程图

官方参考代码路径【示例→Eye-Tracking→iris_detection.py】，具体如下：

```
# 瞳孔追踪示例
#
#这个例子展示了如何找到眼睛的注视(瞳孔检测)
#图像中的眼睛。此脚本使用 find_eyes 函数来确定
#应该包含瞳孔的感兴趣区域的中心点。它基本上是这样做的
#找到眼睛最黑区域的中心，也就是瞳孔中心。
#
#注意:这个脚本不是先检测人脸，而是用长焦镜头。所以请将摄像头直接对准眼睛。
#
#翻译和注释：01Studio

import sensor, time, image
```

```
# Reset sensor
sensor.reset()

# Sensor settings
sensor.set_contrast(3)
sensor.set_gainceiling(16)

# Set resolution to VGA.
sensor.set_framesize(sensor.VGA)

# 裁剪图像到 200x100，这提供了更多的细节和更少的数据要处理
sensor.set_windowing((220, 190, 200, 100))

sensor.set_pixformat(sensor.GRAYSCALE)

# 加载 Haar Cascade 模型
# 使用默认参数步骤
eyes_cascade = image.HaarCascade("eye", stages=24)
print(eyes_cascade)

# FPS clock
clock = time.clock()

while (True):
    clock.tick()
    # Capture snapshot
    img = sensor.snapshot()
    # 找眼睛 !
    # 提示: 使用一个较大的 threshold 值 (识别更多) 和 较小的
    # scale 值 (寻找较小区域的对象)
```

```
    eyes = img.find_features(eyes_cascade, threshold=0.5,
                                    scale_factor=1.5)


    # 找瞳孔
    for e in eyes:
        iris = img.find_eye(e)
        print(iris)
        img.draw_rectangle(e)
        img.draw_cross(iris[0], iris[1])#瞳孔中心坐标 x,y


    # Print FPS.
    # Note: Actual FPS is higher, streaming the FB makes it slower.
    print(clock.fps())
```

- **实验结果：**

运行代码，将摄像头正对自己眼睛，可以看到 OpenMV4 可以将瞳孔位置检测出来，注意光线不能太亮以及眼镜的影响。我们用图片演示一下人眼检测实验。



图 24-75 原图

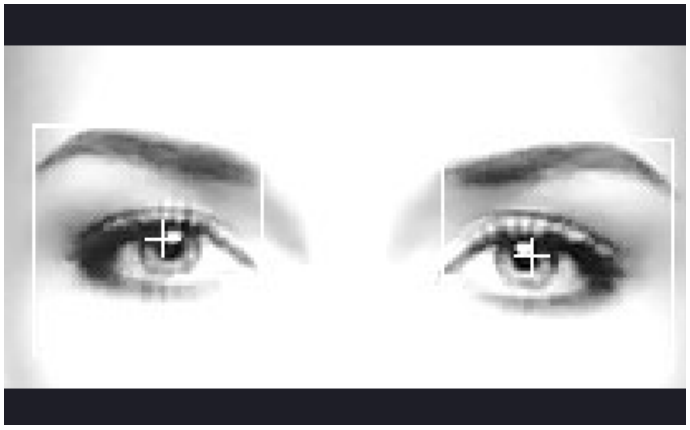


图 24-76 实验图片

- **总结：**

有了前面的基础，我们发现瞳孔追踪实验非常容易就完成了。我们转动眼球，OpenMV4 依然能捕捉到瞳孔位置的变化，这适用于需要对眼球瞳孔实时追踪的应用场景。

24.7 图片拍摄

拍摄可以说是最基本的功能了，本节内容有三部分，分别是普通拍摄、人脸追踪拍摄和移动物体抓拍。是我们摄像头安防监控常用的功能。图片拍摄官方例程在 Snapshot 中。

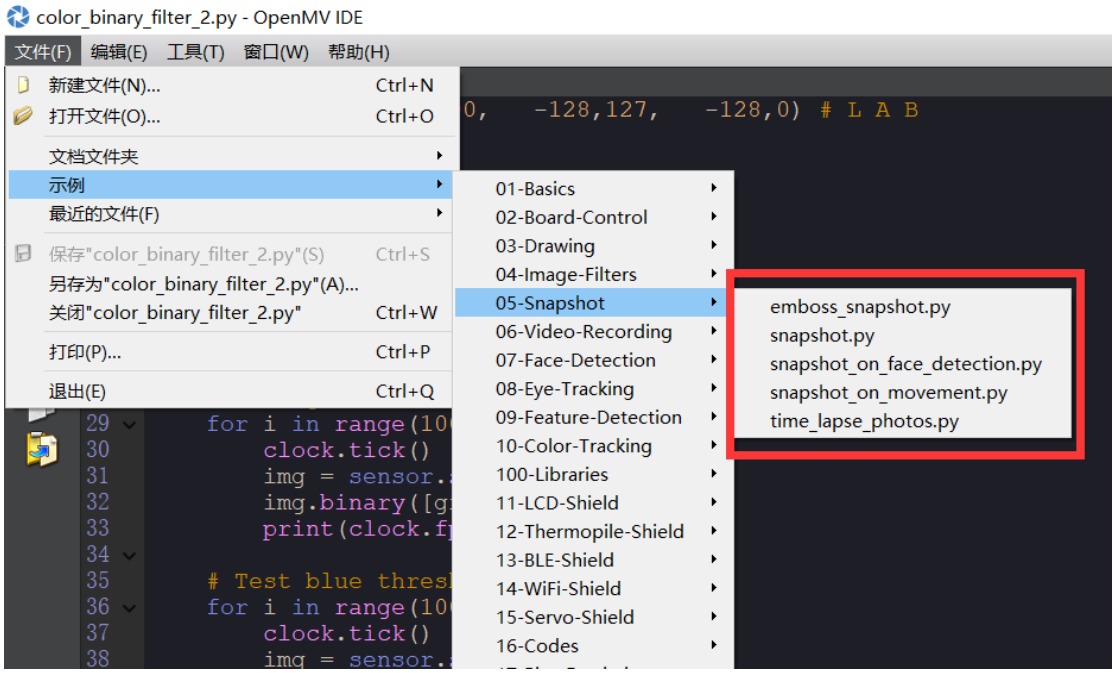


图 24-77 图片拍摄官方例程

24.7.1 普通拍摄

- **前言：**

普通拍摄是最基础的功能，OpenMV4 图像算法处理的前提就是实时连续采集多帧图片，然后进行分析。这个功能可以将 OpenMV4 变成一个随身微型照相机。

- **实验平台：**

pyAI-OpenMV4。另外需要配备 SD 卡进行图片储存。



图 24-78 pyAI-OpenMV4

- **实验目的：**

通过编程实现拍照并保存。

- **实验讲解：**

我们在前面摄像头应用章节已经学习过拍摄是使用 image=sensor.snapshot()函数模块，那么我们只需要学会将图片保存即可。保存也是可以直接使用 image 下的 save 模块，具体如下：

构造函数
img=sensor.snapshot()
通过拍摄创建图像 img

使用方法
image.save(*path*[, *roi*[, *quality=50*]])
保存图片。 `path`：保存路径； `roi`:指定保存区域`(x, y, w, h)`，默认全图保存； `quality`:仅针对 `JPEG` 格式的质量控制，有效值为 `0-100`。

表 24-24　摄像头对象 snapshot

掌握了拍照和保存功能，我们就可以编程实现了，官方例程编程代码流程图如下：

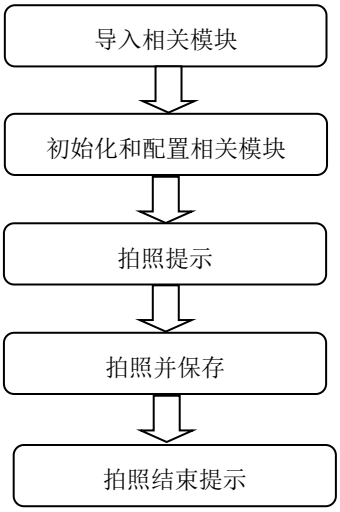


图 24-79 代码编写流程

24.7.2 人脸检测拍摄

- **前言：**

上一节普通拍摄是最基础的功能，本节我们将结合前面人脸检测实验，以便实现当摄像头检测到人脸时，拍照记录下来。

- **实验平台：**

pyAI-OpenMV4 开发板。另外需要配备 SD 卡进行图片储存。



图 24-80 pyAI-OpenMV4

- **实验目的：**

通过编程实现人脸检测并拍摄图片记录下来。

- **实验讲解：**

本节可以说是人脸检测和普通拍摄的相结合，人脸检测请参考 **24.5.1 人脸检测** 章节内容，这里不再重复。

结合上一节普通拍摄内容，我们的例程编程代码流程图如下：

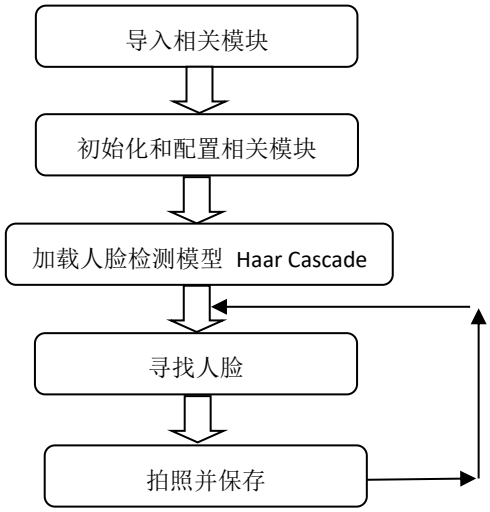


图 24-81 代码编写流程

官方参考代码路径【示例→Snapshot→snapshot_on_face_detection.py】，具体如下：

```
# 人脸检测拍摄示例
#
# 提示：本实验需要 SD 卡.
#
# 这个例子演示了如何在你的 OpenMV 摄像头上使用人脸跟踪然后拍照
#
#翻译和注释：01Studio


import sensor, image, pyb


RED_LED_PIN = 1
BLUE_LED_PIN = 3


#配置摄像头参数
sensor.reset() # Initialize the camera sensor.
sensor.set_pixformat(sensor.GRAYSCALE)
```

```
sensor.set_framesize(sensor.HQVGA) # or sensor.QQVGA (or others)
sensor.skip_frames(time = 2000) # Let new settings take affect.

# 加载人脸检测 HaarCascade 模型. 默认使用 stages=25 以提高识别正确率
face_cascade = image.HaarCascade("frontalface", stages=25)

while(True):

    #红灯亮提示开始识别
    pyb.LED(RED_LED_PIN).on()
    print("About to start detecting faces...")
    sensor.skip_frames(time = 2000) # 给时间用户准备.

    #人脸检测中。
    pyb.LED(RED_LED_PIN).off()
    print("Now detecting faces!")
    pyb.LED(BLUE_LED_PIN).on()

    diff = 10 # 成功识别 10 次后确认为识别到人脸.

    while(diff):
        img = sensor.snapshot()
        # Threshold 和 scale 两个参数控制着识别质量和效率。具体看文档说明。
        faces = img.find_features(face_cascade, threshold=0.5,
                                  scale_factor=1.5)

        if faces:
            diff -= 1
            for r in faces:
                img.draw_rectangle(r)
```

```
pyb.LED(BLUE_LED_PIN).off()
print("Face detected! Saving image...")
sensor.snapshot().save("snapshot-%d.jpg" % pyb.rng()) # 保存照片.
```

- **实验结果：**

运行代码，当摄像头捕捉到人脸时就会拍照并保存到 SD 卡：



图 24-82 扑捉到图片后开始拍摄

按下开发板 RST 复位键，可以看到 SD 卡的文件系统出现了刚刚拍照保存的图片文件。

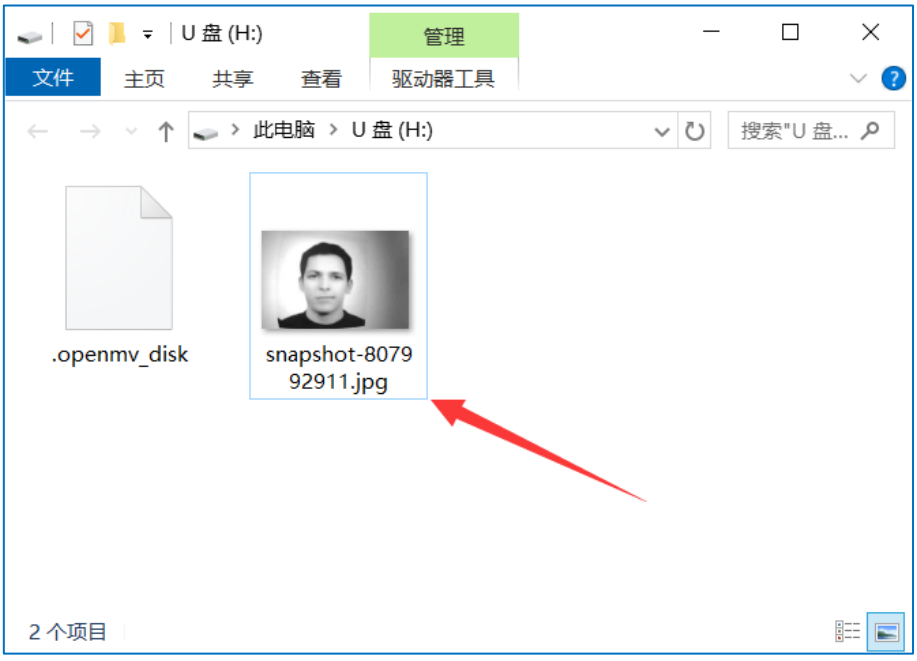


图 24-83 图片储存在 SD 里

- **总结：**

本节通过对人脸检测和拍照的整合，实验了人脸检测追踪拍摄，这适用于一些场合的安防应用。

24.7.3 移动物体抓拍

- **前言：**

移动物体抓拍，是指当摄像头判断到画面有变化时候（这可能是陌生人闯入或者动物活动）拍摄图片记录。这在安防摄像头上市非常常见的功能。今天我们就编程来实现这个功能。

- **实验平台：**

pyAI-OpenMV4 开发板。另外需要配备 SD 卡进行图片储存。



图 24-84 pyAI-OpenMV4

- **实验目的：**

检测移动物体（画面变化）并拍照保存。

- **实验讲解**

检测移动物体并拍照保存，简单来说就是判断画面的变化，换言之就是判断当前图片和最初记录的图片相比较，看有无变化。一旦发生变化，就判定为画面有东西移动，然后拍照保存。

对比函数 difference 在 image 下，具体说明如下：

构造函数
image.difference(*image*[, *mask=None*])
图像对比函数。将两个图像彼此按像素取绝对值。对于每个颜色通道而言，将每个像素替换为 ABS(this.pixel-image.pixel)并返回至当前 image 对象。 【image】:需要对比的图像 例：两张图像如果完全一样，则每个像素返回 0，返回的是全黑的图像。
使用方法
直接调用该函数。

表 24-25　图像对比函数

通过上面对比函数后 image 的信息会改变，可以检测 image 状态来判断图片是否相同。

构造函数
stats=image.statistics()
统计数据对象。 灰度统计数据有一个通道，使用非 l_* 、 a_* 或 b_* 方法； RGB565 百分比值有三个通道。使用 l_* 、 a_* 和 b_* 方法。
使用方法
stats.mean()
返回灰度均值（0-255）整型。
stats.median()
返回灰度中值（0-255）整型。
stats.mode()
返回灰度众值（0-255）整型。
stats.min()
返回灰度最小值（0-255）整型。
stats.max()
返回灰度最大值（0-255）整型。

以上说明均以灰度为例。

*更多使用说明请阅读 OpenMV 官方文档：

https://docs.singtown.com/micropython/zh/latest/openmvcam/library/omv.image.html#class-statistics

表 24-26　数据统计对象

实验原理就是先拍下当前画面并保存，然后摄像头不停的采集实时画面，与保存的图片对比，有差异的话即执行拍摄功能，代码编写流程如下：

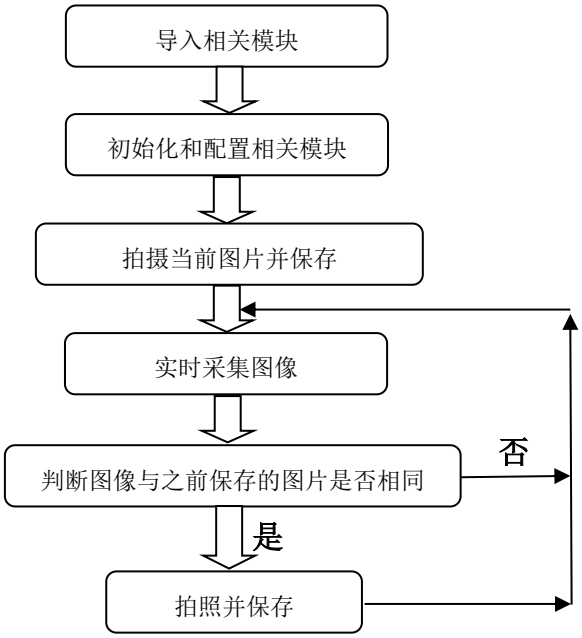


图 24-85 代码编写流程图

官方参考代码路径【示例→Snapshot→snapshot_on_movement.py】，具体如下：

```
# 移动物体抓拍示例
#
# 提示：本实验需要插入 SD 卡.
#
```

```
# 本示例是通过帧差异来检测移动物体。检测到移动物体后拍摄图片并保存。
#
#翻译和注释：01Studio

import sensor, image, pyb, os

RED_LED_PIN = 1
BLUE_LED_PIN = 3

#摄像头初始化
sensor.reset() # Initialize the camera sensor.
sensor.set_pixformat(sensor.RGB565) # or sensor.GRAYSCALE
sensor.set_framesize(sensor.QVGA) # or sensor.QQVGA (or others)
sensor.skip_frames(time = 2000) # Let new settings take affect.
sensor.set_auto_whitebal(False) # Turn off white balance.

if not "temp" in os.listdir(): os.mkdir("temp") # 建立 temp 文件夹

while(True):

    pyb.LED(RED_LED_PIN).on()
    print("About to save background image...")
    sensor.skip_frames(time = 2000) # 给一定时间用户做测试准备.

    pyb.LED(RED_LED_PIN).off()
    sensor.snapshot().save("temp/bg.bmp")
    print("Saved background image - Now detecting motion!")
    pyb.LED(BLUE_LED_PIN).on()

    diff = 10 # 连续测试 10 帧，减少误判.
```

```
while(diff):
    img = sensor.snapshot()
    #和开始拍摄图片对比，不变时界面全黑色，返回图像对象。
    img.difference("temp/bg.bmp")

    stats = img.statistics()
    # 下面 Stats[5] 返回 RGB5656 LAB 中 L 的最大值(0-255) (int)，大于 20
    # 判定为图片变化，即物体移动.
    if (stats[5] > 20):
        diff -= 1

pyb.LED(BLUE_LED_PIN).off()
print("Movement detected! Saving image...")
sensor.snapshot().save("temp/snapshot-%d.jpg" % pyb.rng())
# 保存图片.
```

- **实验结果：**

运行代码，可以看到过一会 IDE 的帧缓冲区是黑色。这是因为图像没变化，对比函数处理好图像的值变成 0（黑色）。

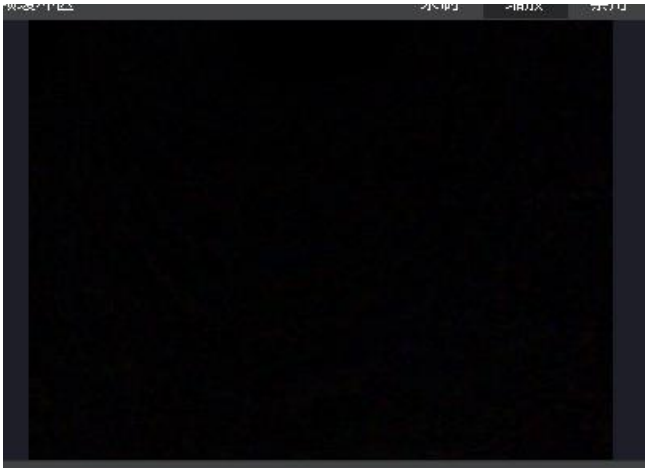


图 24-86 图像界面没变化时候显示情况

我们将手指伸到摄像头前，可以看到摄像头检测到变化并提示拍照保存。



图 24-87 让画面发生变化

按下开发板复位，打开 U 盘（SD 卡），看到 temp 文件夹下保存里面相关照片文件。bg.bmp 为上电首次拍摄图片，snapshot-xxx.jpg 为移动界面抓拍图片。



图 24-88 U 盘（SD 卡）保存的文件

- **总结：**

本节通过图像对比函数，轻松实现图像界面变化识别。从而实现简易的移动物体抓拍功能。

24.8 视频录制

视频录制的学习方式跟图片拍摄有点类似，本节内容有三部分，分别是普通视频拍摄、人脸追踪视频拍摄和移动物体视频抓拍。也是我们摄像头安防监控常用的功能。图片拍摄官方例程在 Video-Recording 中。

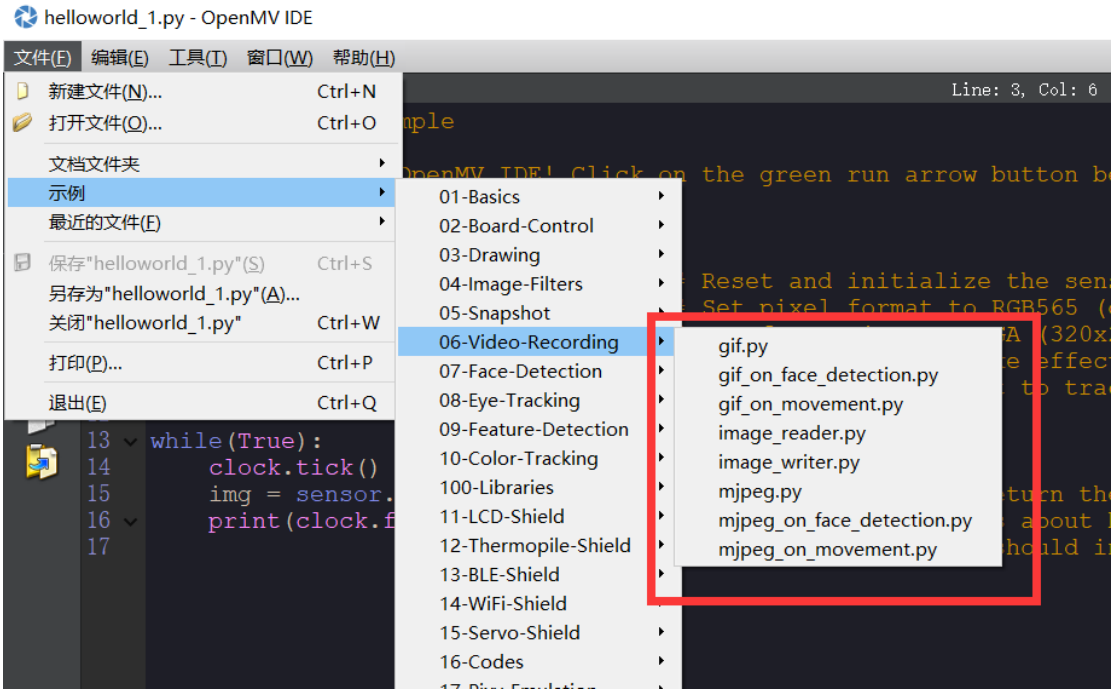


图 24-89 视频拍摄官方例程

24.8.1 普通 GIF 视频录制

- **前言：**

GIF 是非常流行的动图文件，我们平时在网上看到的表情包或短视频很多都是使用 GIP 格式。今天我们就来学习一下使用 OpenMV4 来录制视频。

- **实验平台：**

pyAI-OpenMV4。另外需要配备 SD 卡进行储存。



图 24-90 pyAI-OpenMV4

- **实验目的：**

录制 GIF 视频。

- **实验讲解：**

OpenMV4 集成了视频采集函数，也是使用拍摄功能，原理是将一张张拍摄的图片添加一起组合成动图。我们来看看 GIF 的对象：

构造函数
gif.Gif(*filename, width=Auto, height=Auto, color=Auto, loop=True*)
创建一个 Gif 对象。 【filename】保存的路径和文字； 【width】Auto 为图像传感器水平分辨率（除非显式覆盖）；

【height】Auto 为图像传感器垂直分辨率（除非显式覆盖）； 【color】Auto 为图像传感器颜色模式（除非显式覆盖）； 【loop】Ture 表示自动循环播放
使用方法
`gif.width()`
返回 gif 对象的宽度（水平分辨率）。
`gif.height()`
返回 gif 对象的高度（垂直分辨率）。
`gif.add_frame(image, delay=10)`
将一张图像添加到 gif 记录中。图像的宽度、高度和颜色模式必须与 gif 构造函数中使用的宽度、高度和颜色模式相同。 【image】添加的图像 【delay】是前一帧结束后等待播放此帧的厘秒数，默认是 10。
*更多使用说明请阅读 OpenMV 官方文档： https://docs.singtown.com/micropython/zh/latest/openmvcam/library/omv.gif.html#id3

表 24-27　GIF 对象

学习了 gif 的相关用法后，我们整理思路，代码编写流程如下：

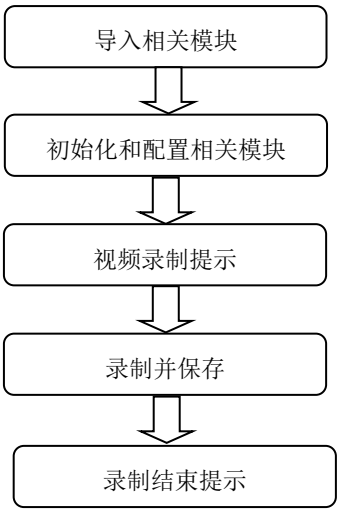


图 24-91 代码编写流程

官方参考代码路径【示例→Video Recording→gif.py】，具体如下：

```
# GIF 视频录制示例
#
# 提示：你需要插入 SD 卡来运行此程序.
#
# 你可以使用 OpenMV 来录制 gif 文件. 可以选择性录制 RGB565 或灰度图像.
# 使用像 GIMP 这样的照片编辑软件来压缩和优化 Gif，然后再上传到 web。
#
#翻译和注释：01Studio

import sensor, image, time, gif, pyb

RED_LED_PIN = 1
BLUE_LED_PIN = 3

#摄像头初始化
sensor.reset() # Initialize the camera sensor.
sensor.set_pixformat(sensor.RGB565) # or sensor.GRAYSCALE
sensor.set_framesize(sensor.QQVGA) # or sensor.QVGA (or others)
sensor.skip_frames(time = 2000) # Let new settings take affect.
clock = time.clock() # Tracks FPS.

pyb.LED(RED_LED_PIN).on()
sensor.skip_frames(time = 2000) # Give the user time to get ready.

pyb.LED(RED_LED_PIN).off()
pyb.LED(BLUE_LED_PIN).on()

g = gif.Gif("example.gif", loop=True)
```

```
print("You're on camera!")
for i in range(100):
    clock.tick()
    # clock.avg() 返回每帧平均运行时间,单位 ms;
    # gif 的延时是厘秒级别（百分之一秒）
    g.add_frame(sensor.snapshot(), delay=int(clock.avg()/10))
    print(clock.fps())

g.close()
pyb.LED(BLUE_LED_PIN).off()
print("Done! Reset the camera to see the saved recording.")
```

- **实验结果：**

运行代码，将摄像头对准要拍摄的物体，点击运行。



图 24-92 对准拍摄物体



图 24-93 拍摄效果

拍摄完成后，按下开发板复位键，可以看到 U 盘（SD 卡）上出现了刚刚拍摄的 gif 文件，文件名为：“example.gif”。点击打开就可以看到录制的视频。



图 24-94 gif 视频文件

- **总结：**

本节学习了视频录制，结合基础例程的按键可以打造按键录像机。有兴趣的用户可以自行编程实现。

24.8.2 人脸检测 GIF 视频录制

- **前言：**

上一节普通拍摄是最基础的功能，本节我们将结合前面人脸检测实验，以便实现当摄像头检测到人脸时，拍摄视频记录下来。

- **实验平台：**

pyAI-OpenMV4。另外需要配备 SD 卡进行图片储存。



图 24-95 pyAI-OpenMV4

- **实验目的：**

通过编程实现人脸检测并拍摄视频记录下来。

- **实验讲解：**

本节可以说是人脸检测和普通 GIF 视频拍摄的相结合，人脸检测请参考 **24.5.1 人脸检测** 章节内容，这里不再重复。

结合上一节普通 GIF 视频拍摄内容，我们的例程编程代码流程图如下：

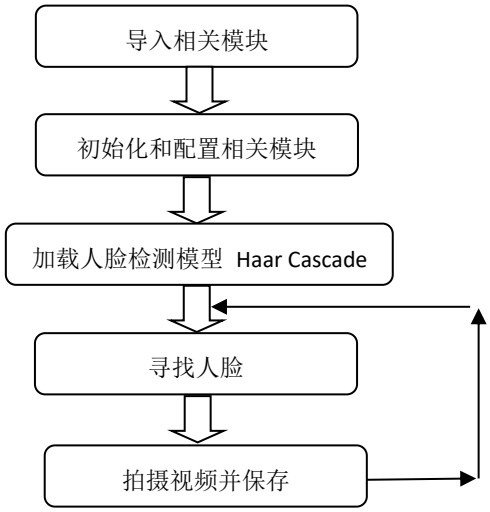


图 24-96 代码编写流程

官方参考代码路径【示例→Video Recording→gif_on_face_detection.py】，具体如下：

```
# 人脸检测 GIF 视频录制例程
#
# 提示：此程序需要使用 SD 卡.
#
# 你可以使用 OpenMV 来录制 gif 文件. 可以选择性录制 RGB565 或灰度图像.
# 使用像 GIMP 这样的照片编辑软件来压缩和优化 Gif，然后再上传到 web。
#
# 这个例程是实现当检测到人脸时候自动录制 GIF 视频
#
#翻译和注释：01Studio

import sensor, image, time, gif, pyb

RED_LED_PIN = 1
BLUE_LED_PIN = 3
```

```
#摄像头初始化
sensor.reset() # Initialize the camera sensor.
sensor.set_pixformat(sensor.GRAYSCALE) # or sensor.
sensor.set_framesize(sensor.QQVGA) # or sensor.HQVGA (or others)
sensor.skip_frames(time = 2000) # Let new settings take affect.

# 人脸检测是通过在图像上使用 Haar Cascade 特征检测器来实现的。
# 一个 Haar Cascade 是一系列简单区域的对比检查.人脸识别有 25 个阶段，每个阶段#
# 有几百次检测。Haar Cascades 运行很快因为是逐个阶段递进检测的。
# 此外，OpenMV Cam 使用一种称为积分图像的数据结构来在恒定时间内快速执行每个区
# 域的对比度检查(特征检测需要用灰度图像的原因是因为图像积分需要更多空间)。
face_cascade = image.HaarCascade("frontalface", stages=25)

while(True):

    pyb.LED(RED_LED_PIN).on()
    print("About to start detecting faces...")
    sensor.skip_frames(time = 2000) # Give the user time to get ready.

    pyb.LED(RED_LED_PIN).off()
    print("Now detecting faces!")
    pyb.LED(BLUE_LED_PIN).on()

    diff = 10 # 重复检测 10 帧，确认人脸.
    while(diff):
        img = sensor.snapshot()
        # threshold 和 scale_factor 两个参数控制着识别的效率和准确性.
        faces = img.find_features(face_cascade, threshold=0.5,
scale_factor=1.5)
```

```
    if faces:
        diff -= 1
        for r in faces:
            img.draw_rectangle(r)

# 创建 GIF 对象，pyb.rng()随机返回一个 30 位的二进制数值，
# 最终文件名转换成 10 进制。
g = gif.Gif("example-%d.gif" % pyb.rng(), loop=True)

clock = time.clock() # Tracks FPS.
print("You're on camera!")
for i in range(100):
    clock.tick()
# clock.avg() 返回每帧平均运行时间，单位 ms ；
# gif 的延时是厘秒级别（百分之一秒）
    g.add_frame(sensor.snapshot(), delay=int(clock.avg()/10))
    print(clock.fps())

g.close()
pyb.LED(BLUE_LED_PIN).off()
print("Restarting...")
```

- **实验结果：**

在 IDE 中运行代码，将摄像头对准人脸，检测到后开始拍摄 GIF 视频：

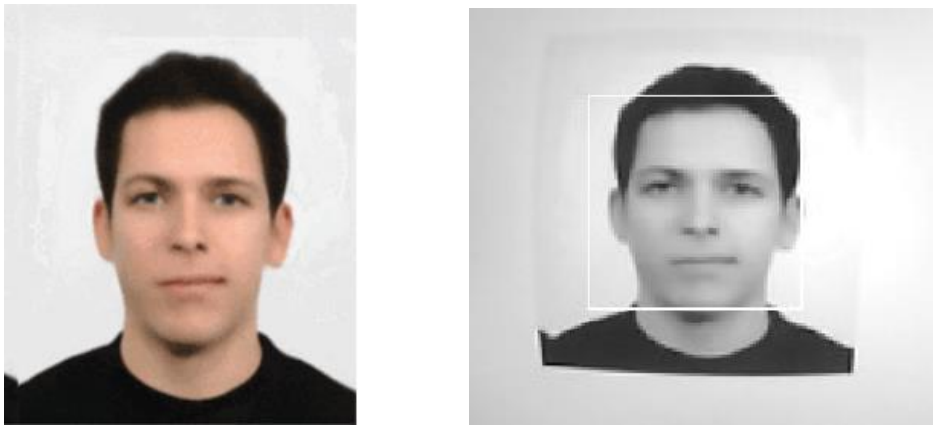


图 24-97 单个人脸检测 原图（左）和实验图片（右）

拍摄完成后，按下开发板复位键，可以看到 U 盘（SD 卡）上出现了刚刚拍摄的 gif 文件，文件名为：“example-xxx.gif”。点击打开就可以看到刚刚录制的视频。



图 24-98 GIF 视频文件

- **总结：**

本节通过对人脸检测和拍摄视频的整合，实验了人脸检测追踪拍摄视频，这适用于一些场合的安防应用。

24.8.3 移动物体 GIF 视频录制

- **前言：**

移动物体视频录制，是指当摄像头判断到画面有变化时候（这可能是陌生人闯入或者动物活动）拍摄一段小视频记录。这在安防摄像头上市非常常见的功能。今天我们就编程来实现这个功能。

- **实验平台：**

pyAI-OpenMV4。另外需要配备 SD 卡进行 GIF 视频文件储存。



图 24-99 pyAI-OpenMV4

- **实验目的：**

检测移动物体（画面变化）并拍摄视频保存。

- **实验讲解**

检测移动物体并拍摄视频保存，简单来说就是判断画面的变化，换言之就是判断当前图片和最初记录的图片相比较，看有无变化。一旦发生变化，就判定为画面有东西移动，然后拍摄视频保存。

本节内容可以说是前面学习过的移动物体抓拍和 GIF 视频拍摄内容相结合。移动问题判断可以在 **24.7.3 移动物体抓拍** 章节内容找到，这里不在重复。

结合通 GIF 视频拍摄内容，我们的例程编程代码流程图如下：

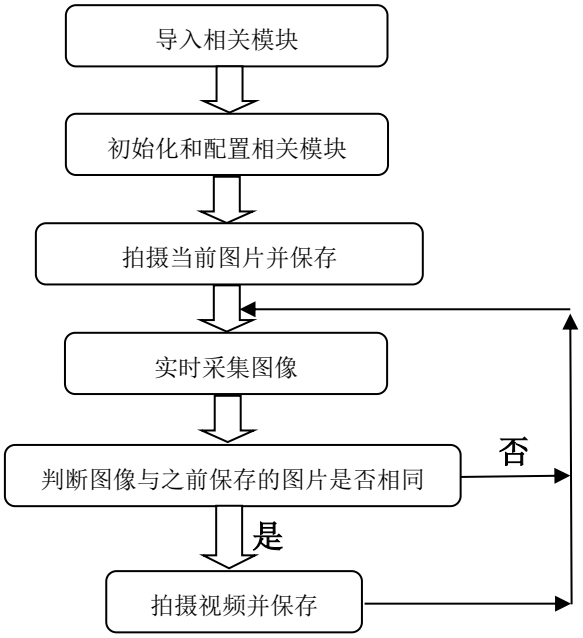


图 24-100 代码编写流程图

官方参考代码路径【示例→Video Recording→gif_on_movement.py】，具体如下：

```
# 移动物体 GIF 视频录制例程
#
# 提示：你需要插入 SD 卡来运行此程序.
#
# 你可以使用 OpenMV 来录制 gif 文件. 可以选择性录制 RGB565 或灰度图像.
# 使用像 GIMP 这样的照片编辑软件来压缩和优化 Gif，然后再上传到 web。
#
# 本示例是通过帧差异来检测移动物体。检测到移动物体后拍摄视频并保存。
#
#翻译和注释：01Studio


import sensor, image, time, gif, pyb, os
```

```
RED_LED_PIN = 1
BLUE_LED_PIN = 3

#摄像头初始化
sensor.reset() # Initialize the camera sensor.
sensor.set_pixformat(sensor.RGB565) # or sensor.GRAYSCALE
sensor.set_framesize(sensor.QQVGA) # or sensor.QVGA (or others)
sensor.skip_frames(time = 2000) # Let new settings take affect.
sensor.set_auto_whitebal(False) # Turn off white balance.

if not "temp" in os.listdir(): os.mkdir("temp") # 新建一个 temp 文件夹存放
视频

while(True):

    pyb.LED(RED_LED_PIN).on()
    print("About to save background image...")
    sensor.skip_frames(time = 2000) # Give the user time to get ready.

    pyb.LED(RED_LED_PIN).off()
    sensor.snapshot().save("temp/bg.bmp")
    print("Saved background image - Now detecting motion!")
    pyb.LED(BLUE_LED_PIN).on()

    diff = 10 # 连续测试 10 帧，减少误判.
    while(diff):
        img = sensor.snapshot()
        img.difference("temp/bg.bmp")
        stats = img.statistics()
```

```
        #下面 Stats[5] 返回 RGB5656 LAB 中 L 的最大值(0-255) (int)，大于 20 判定为图片变化，即物体移动.
        if (stats[5] > 20):
            diff -= 1


    g = gif.Gif("example-%d.gif" % pyb.rng(), loop=True)


    clock = time.clock() # Tracks FPS.
    print("You're on camera!")
    for i in range(100):
        clock.tick()
        # clock.avg() returns the milliseconds between frames - gif delay is in
        g.add_frame(sensor.snapshot(), delay=int(clock.avg()/10)) # centiseconds.
        print(clock.fps())


    g.close()
    pyb.LED(BLUE_LED_PIN).off()
    print("Restarting...")
```

- **实验结果：**

运行代码，可以看到过一会 IDE 的帧缓冲区是黑色。这是因为图像没变化，对比函数处理好图像的值变成 0（黑色）。

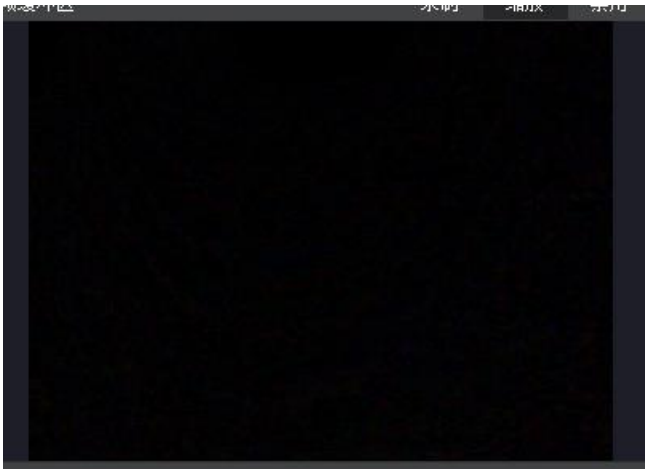


图 24-101 图像界面没变化时候显示情况

我们将手指伸到摄像头前，可以看到摄像头检测到变化并提示拍摄视频。



图 24-102 让画面发生变化

按下开发板复位，打开 U 盘（SD 卡），看到 temp 文件夹下保存的 bg.bmp 为上电首次拍摄图片。

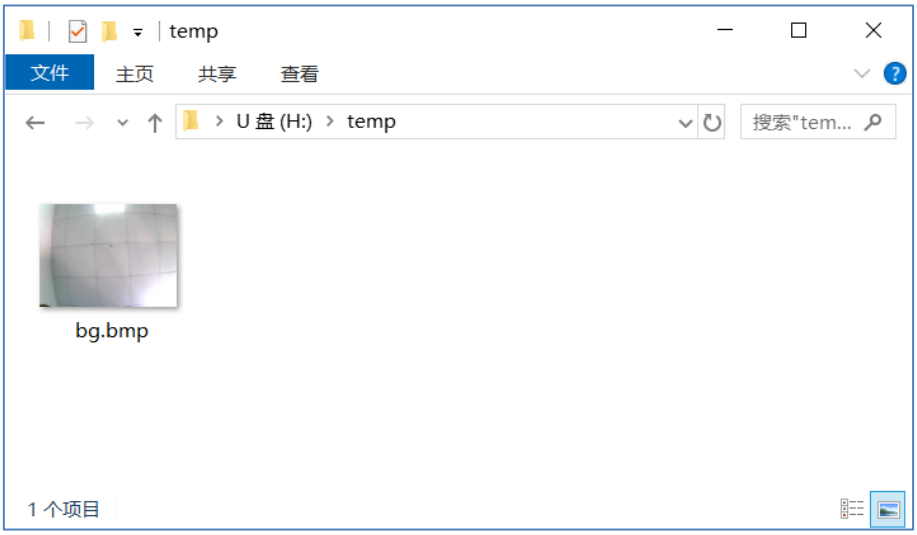


图 24-103 U 盘（SD 卡）temp 目录保存的图片文件

根目录里面保存相关 GIF 视频文件。

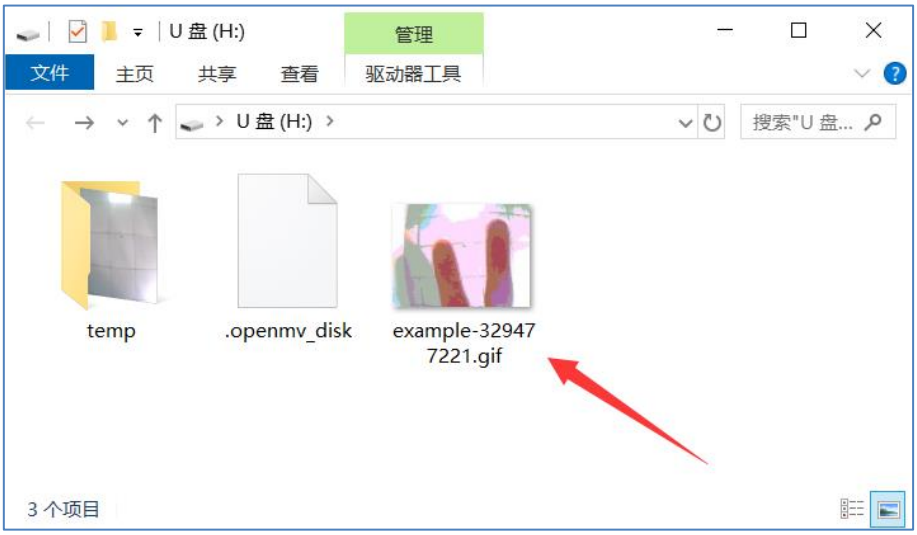


图 24-104 GIF 视频文件在 U 盘根目录下

- **总结：**

本节通过图像对比函数，轻松实现图像界面变化识别。从而实现简易的移动物体抓拍视频功能。

24.9 图像滤波

图片滤波处理可以说是整个机器视觉应用的基础，当我们通过摄像头取得拍摄图片时候，通过特定方式的滤波可以让图片去掉无相关信息，让后期图像处理变得更容易。日常生活中的拍照美颜，以及前面的机器视觉实验大多都运用了图像滤波算法。

OpenMV ID 集成了非常多的图像滤波例程，在 image-filters 中。我们挑选常用的例程来学习，以便了解其使用方法。

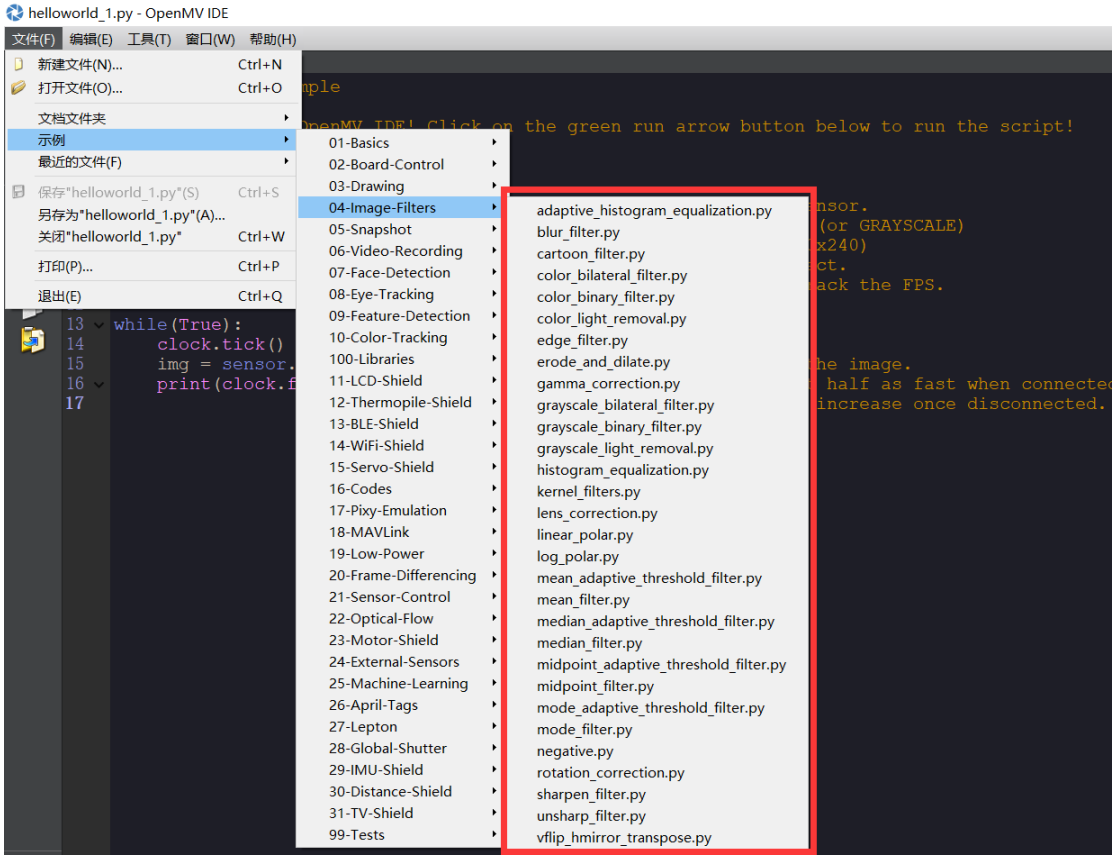


图 24-105 图像滤波相关例程

24.9.1 颜色二值化滤波

- **前言：**

颜色二值化是指将彩色图像黑白图像，而凸显出图像中某种颜色，比如我们想突出一幅图像中的红色，我们可以通过二值化将图像中红色部分变成白色，然后其余部分变成黑色。变化后的图像文件将变得非常小，以提高后期图片处理的速度。

- **实验平台：**

pyAI-OpenMV4。另外需要配备 SD 卡进行储存。



图 24-106 pyAI-OpenMV4

- **实验目的：**

将图像中特定颜色的物体进行二值化滤波。

- **实验讲解：**

图像二值化我们在特征检测章节学习过，这里重温一下其对象说明：

构造函数
image.binary(*thresholds*[, *invert=False*[, *zero=False*[, *mask=None*[, *to_bitmap=False*[, *copy=False*]]]]])

图像二值（黑白）化。 【thresholds】 必须是元组列表。 (lo, hi) 定义你想追踪的颜色范围。对于灰度图像，每个元组需要包含两个值：最小灰度值和最大灰度值。 **例：thresholds=(0,100) ，则该函数表示将(0,100)灰度值范围变成白色。** 【invert】如果为 True，则 0 1（黑 白）反转；默认 False 不反转。 *更多参数说明请阅读 OpenMV 官方文档： 文档链接：http://docs.openmv.io/library/omv.image.html
使用方法
直接调用该函数。

表 24-28 图像二值化应用

二值化的函数非常简单，我们定义好阈值即可，代码编写流程如下：

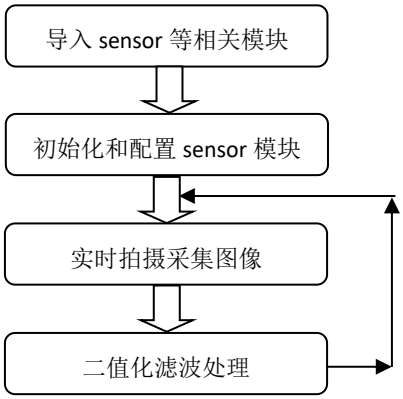


图 24-107 代码编写流程图

官方参考代码路径【示例→Image Filters→Color_Binary_Filter.py】，具体如下：为了方便测试，我们将蓝色和绿色的二值化用”’注释掉，所以以下代码仅进行红色和非红色两个情况的二值化滤波。

```
# 颜色二值化滤波示例
#
# 这个程序展示了图像二值化滤波．你可以自定义任何阈值来分割不同颜色的图像。
#
```

```
#翻译和注释：01Studio

import sensor, image, time

sensor.reset()
sensor.set_framesize(sensor.QVGA)
sensor.set_pixformat(sensor.RGB565)
sensor.skip_frames(time = 2000)
clock = time.clock()

# 可以使用  工具-> 机器视觉 -> 阈值编辑器 来调整阈值.
red_threshold = (0,100,   0,127,   0,127) # L A B
green_threshold = (0,100,   -128,0,   0,127) # L A B
blue_threshold = (0,100,   -128,127,   -128,0) # L A B

while(True):

    # Test red threshold
    for i in range(100):
        clock.tick()
        img = sensor.snapshot()
        img.binary([red_threshold])
        print(clock.fps())

    '''
    # Test green threshold
    for i in range(100):
        clock.tick()
        img = sensor.snapshot()
        img.binary([green_threshold])
```

```
    print(clock.fps())


# Test blue threshold
for i in range(100):
    clock.tick()
    img = sensor.snapshot()
    img.binary([blue_threshold])
    print(clock.fps())
'''
# Test not red threshold
for i in range(100):
    clock.tick()
    img = sensor.snapshot()
    img.binary([red_threshold], invert = 1)
    print(clock.fps())
'''
# Test not green threshold
for i in range(100):
    clock.tick()
    img = sensor.snapshot()
    img.binary([green_threshold], invert = 1)
    print(clock.fps())


# Test not blue threshold
for i in range(100):
    clock.tick()
    img = sensor.snapshot()
    img.binary([blue_threshold], invert = 1)
    print(clock.fps())
'''
```

- **实验结果：**

运行代码，我们将红色物体放在摄像头前，可以看到程序对图像进行红色和非红色二值化滤波（不断循环）。

对红色进行二值化滤波，红色物体会用白色显示：



图 24-108 红色二值化滤波 原图（左）和实验图片（右）

对红色进行二值化滤波，红色物体会用黑色显示（跟上面红色二值化滤波刚好相反）：

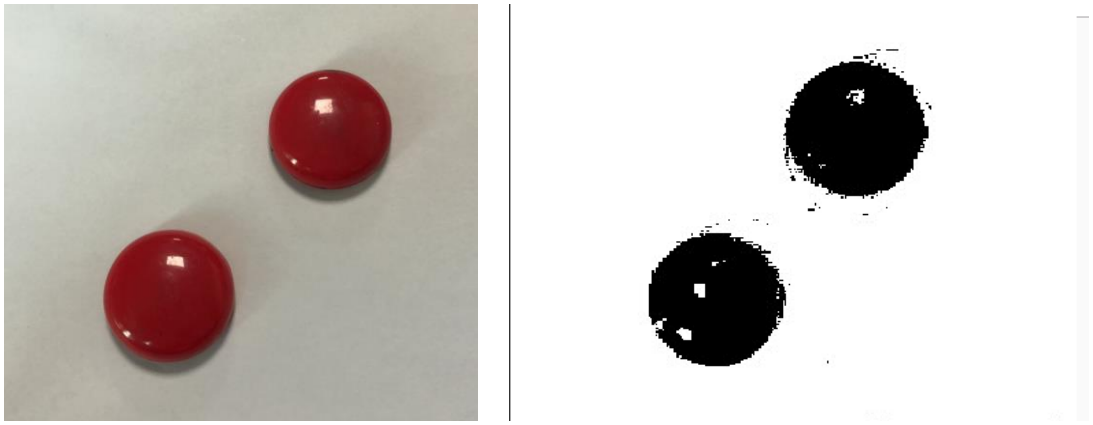


图 24-109 非红色二值化滤波 原图（左）和实验图片（右）

- **总结：**

本节学习了图像二值化滤波处理，这是应用非常广泛的滤波示例，特别适用于指定颜色追踪和处理。因为滤波后的图像变得更小，特征更明显，方便处理。

24.9.2 锐化滤波

- **前言：**

图像锐化*(image sharpening)*是补偿图像的轮廓，增强图像的边缘及灰度跳变的部分，使图像变得清晰，分为空间域处理和频域处理两类。图像锐化是为了突出图像上地物的边缘、轮廓，或某些线性目标要素的特征。这种滤波方法提高了地物边缘与周围像元之间的反差，因此也被称为边缘增强。

简答来说锐化就是为了让图像变得更清晰。

- **实验平台：**

pyAI-OpenMV4 开发板。



图 24-110 pyAI-OpenMV4

- **实验目的：**

编程实现将拍摄到的图像进行锐化处理。

- **实验讲解：**

本节实验是通过使用拉普拉斯滤波器来锐化图像。当然和以往一样，我们不需要再重复造轮子，OpenMV4 已经集成了 laplacian 函数算法。我们学会对象构建和使用即可，具体如下：

构造函数
image.laplacian(*size*[, *sharpen=False*[, *mul*[, *add=0*[, *threshold=False*[, *offset=0*[, *invert=False*[, *mask=None*]]]]]]])
通过边缘检测拉普拉斯核来对图像进行卷积 【size】内核大小。1（3X3 内核），2（5X5 内核）或更高值； 【sharpen】如果为 True，则为锐化图像。 *更多参数说明请阅读 OpenMV 官方文档： https://docs.singtown.com/micropython/zh/latest/openmvcam/library/omv.image.html
使用方法
直接调用该函数。

表 24-29 拉普拉斯函数

从上表可以看到，锐化使用方法只需要配置拉普拉斯函数终端 size 和 sharpen 即可，代码编写流程如下：

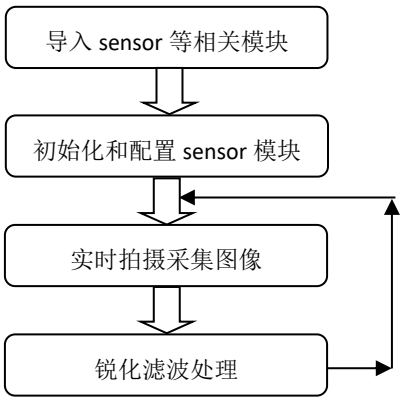


图 24-111 代码编写流程

官方参考代码路径【示例→Image Filters→ sharpen_filter.py】，具体如下：

```
# 锐化滤波示例
#
# 这个例子展示了如何使用拉普拉斯滤波器来锐化图像。
```

```
#
#翻译和注释:01Studio

import sensor, image, time

#摄像头初始化
sensor.reset() # Initialize the camera sensor.
sensor.set_pixformat(sensor.GRAYSCALE) # or sensor.RGB565
sensor.set_framesize(sensor.QQVGA) # or sensor.QVGA (or others)
sensor.skip_frames(time = 2000) # Let new settings take affect.
clock = time.clock() # Tracks FPS.

while(True):
    clock.tick() # Track elapsed milliseconds between snapshots().
    img = sensor.snapshot() # Take a picture and return the image.

    # 运行拉普拉斯内核.
    img.laplacian(1, sharpen=True)

    print(clock.fps()) #打印 FPS
```

- **实验结果：**

运行程序，我们可将代码 img.laplacian(1, sharpen=True) 注释，表示不进行锐化，与锐化进行比较。可见锐化后的图片更清晰。所得结果如下：

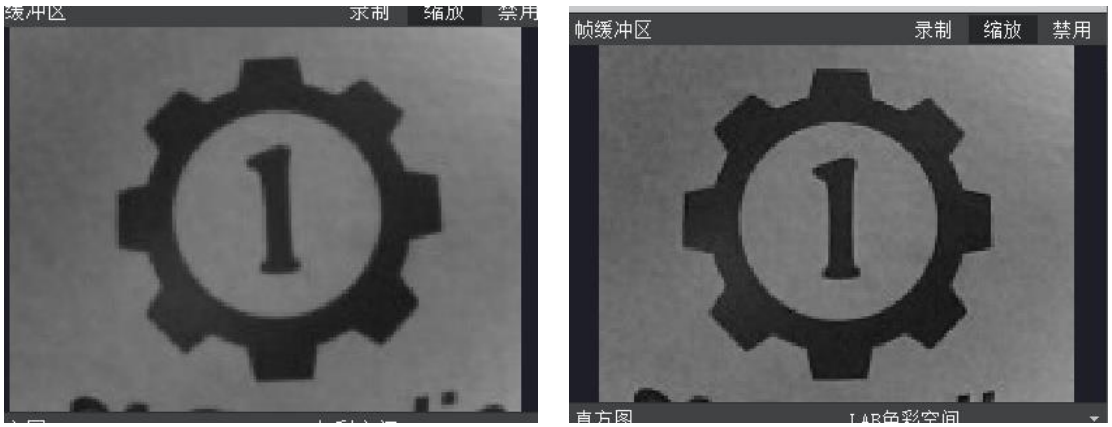


图 24-112 锐化前（左）与锐化后（右）比较

- **总结：**

本节学习了图像滤波中的锐化，主要实现的功能是让拍摄的图片变得更清晰。这是我们在 Photoshop 中经常使用的功能。

24.9.3 图像翻转

- **前言：**

图像旋转和镜像是经常用到的，比如某些场合我们需要将摄像头硬件反过来安装，如装在天花板上的监控，那么我们未来正确地显示图像就会使用图像旋转或者镜像功能。学会了这个功能我们足以应付摄像头各种不同安装时的应用场景。

- **实验平台：**

pyAI-OpenMV4。



图 24-113 pyAI-OpenMV4

- **实验目的：**

编程实现图像的各种旋转和镜像。

- **实验讲解：**

不难想象，一张图片有 0、90、180、270 四个角度一共有 4 种情况，加上镜像那么一共有八种显示情况，OpenMV4 就是通过垂直反转、水平镜像、转置 3 个参数（2^3=8）来确定图像的 8 种显示方式。使用的是 image 下的 replace 函数，具体如下：

构造函数
image.replace(*image*[, *hmirror=False*[, *vflip=False*[, *transpose=False*[, *mask=None*]]]])
图像翻转功能。 【image】 要操作的图像对象； 【hmirror】如果为 True，则水平镜像替换图像。 【vflip】如果为 True，则垂直翻转替换图像。 【transpose】如果为 True，则沿对角线翻转图像。
使用方法
直接调用该函数。 例： vflip=False, hmirror=False, transpose=False -> 旋转 0° vflip=True, hmirror=False, transpose=True -> 旋转 90° vflip=True, hmirror=True, transpose=False -> 旋转 180° vflip=False, hmirror=True, transpose=True -> 旋转 270°

表 24-30 图像翻转函数

可见图像的各种翻转就是通过配置函数参数实现，代码编写流程如下：

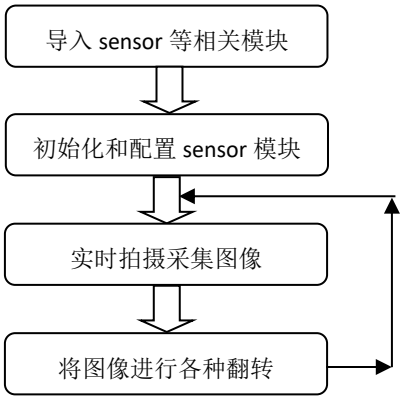


图 24-114 代码编写流程

官方参考代码路径【示例→Image Filters→ vflip_hmirror_transpose.py】，具体如下：

```
# 垂直反转 - 水平镜像 - 转置
#
# 这个示例展示了如何将图像实现 垂直反转 - 水平镜像 - 转置
# 具体参数如下：3 个参数一共有 8 中情况,以下罗列了 4 种无镜像情况
#
# vflip=False, hmirror=False, transpose=False -> 0 degree rotation
# vflip=True,  hmirror=False, transpose=True  -> 90 degree rotation
# vflip=True,  hmirror=True,  transpose=False -> 180 degree rotation
# vflip=False, hmirror=True,  transpose=True  -> 270 degree rotation
#
#翻译和注释：01Studio

import sensor, image, time, pyb

#摄像头初始化
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 2000)
clock = time.clock()

mills = pyb.millis()
counter = 0

while(True):
    clock.tick()



    img = sensor.snapshot().replace(vflip=(counter//2)%2,
                                    hmirror=(counter//4)%2,
```

```
                                transpose=(counter//8)%2)


# 也可以手动修改参数观察图像变化情况。
#img = sensor.snapshot().replace(vflip=False,
#                                hmirror=False,
#                                 transpose=False)


 if (pyb.millis() > (mills + 1000)):
     mills = pyb.millis()
     counter += 1


 print(clock.fps())
```

上面代码中通过 counter 变量和运算符//和%的结合，巧妙地实现了 8 种参数组合循环。我们也可以手动修改参数观察图像变化情况。

- **实验结果：**

运行代码，可以看到图像在 8 中显示方式中不断循环。





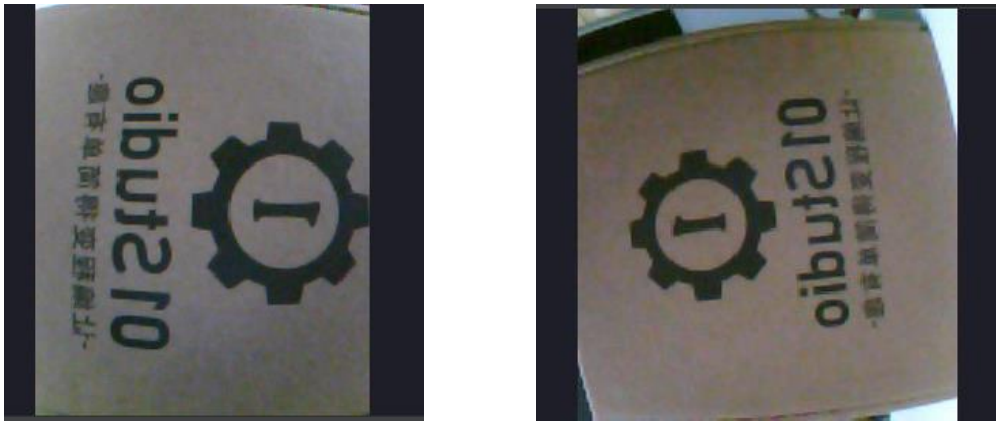


图 24-115 实验部分图片展示

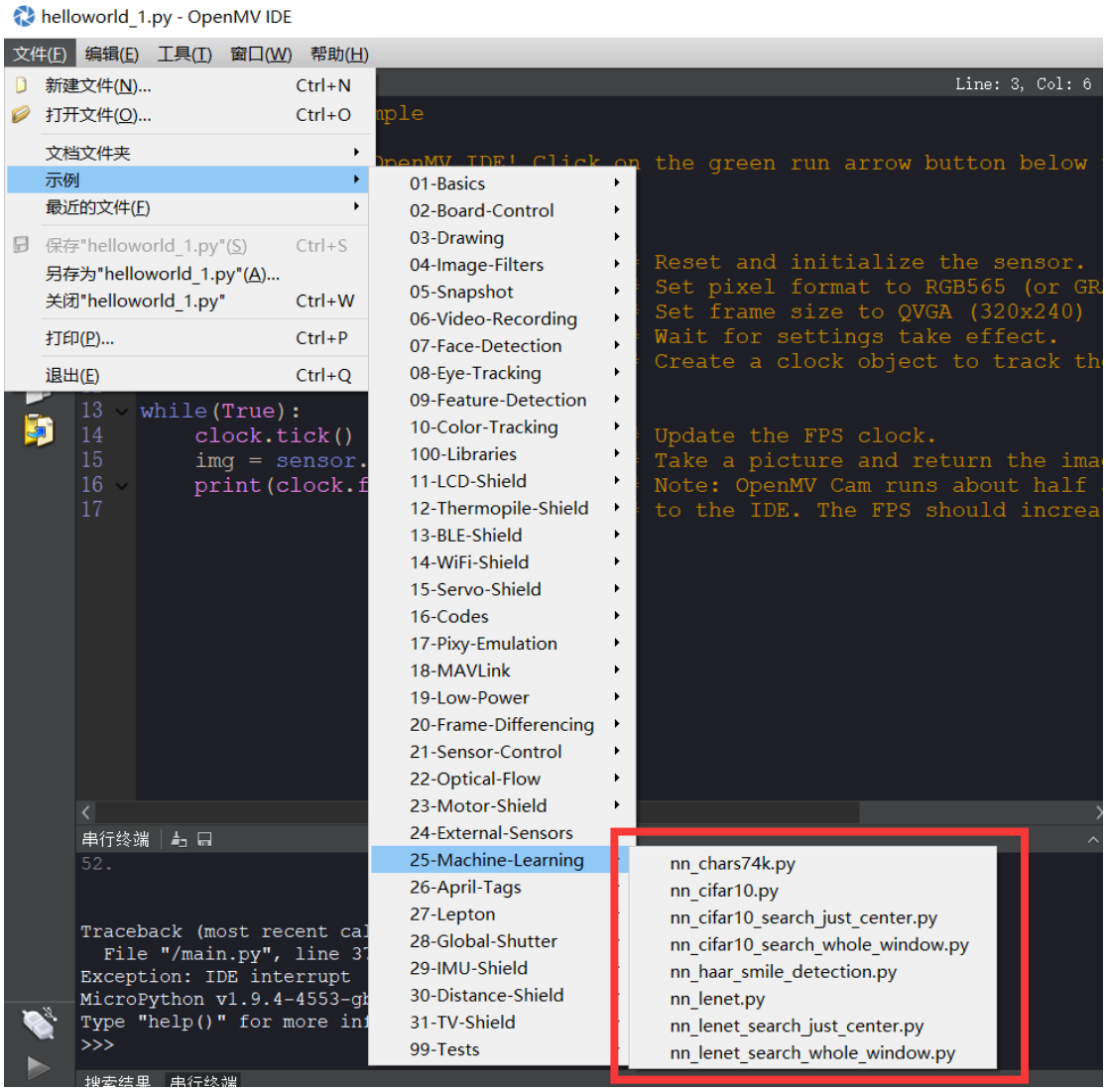
- **总结：**

本节学习了图像的各种旋转和镜像，有了这个技能，无论我们的 OpenMV4 如何安装，都可以轻松实现画面的调整。

第25章　机器学习

机器学习是一门多领域交叉学科，涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科。专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构使之不断改善自身的性能。

它是人工智能的核心，是使计算机具有智能的根本途径。OpenMV 全部官方例程在示例的 Machine-Learning 下，本章对重点实验进行讲解。



25.1 物体识别

- **前言：**

生活中当我们看到一个物体，我们会将它和我们大脑中对这个物体的记忆进行匹配，然后我们就知道了这是什么。物体识别，和这一人类行为非常类似，摄像头将拍摄到的图片，跟过去学习的神经网络模型进行匹配，然后告诉我们这是什么。今天，我们就来做这个非常有趣的实验。

- **实验平台：**

pyAI-OpenMV4 或 pyAI-OpenMV4 开发套件，需要使用 SD 卡。

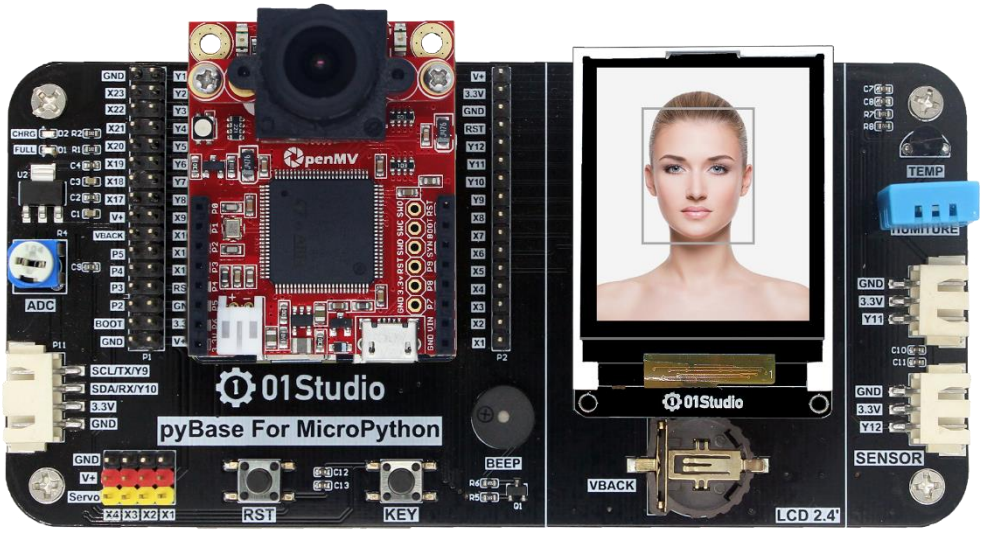


图 25-1 pyAI-OpenMV4 开发套件

- **实验目的：**

对拍摄的物体进行识别。

- **实验讲解：**

拍摄图片和机器视觉我们前面都学习了，也就是说我们只需要给 OpenMV4 一个神经网络模型就可以。关于物体识别的神经网络模型本实验使用一个叫做 CIFAR10 物体识别模型，这个模型有提供现成的文件，我们直接使用即可。我们先来简单认识一下 CIFAR10 神经网络模型。

CIFAR10 是一个拥有 10 种物体识别能力的神经模型，分别是“飞机”、“汽车”、

“鸟”、“猫”、“鹿”、“狗”、“青蛙”、“马”、“船”、“卡车”。 这个神经网络数据集共有 60000 张彩色图像，这些图像是 32*32，分为 10 个类，每类 6000 张图。这里面有 50000 张用于训练，构成了 5 个训练批，每一批 10000 张图；另外 10000 用于测试，单独构成一批。测试批的数据里，取自 10 类中的每一类，每一类随机取 1000 张。抽剩下的就随机排列组成了训练批。注意一个训练批中的各类图像并不一定数量相同，总的来看训练批，每一类都有 5000 张图。关于更多 CIFAR10 的实现原理大家可以自行网络搜索研究

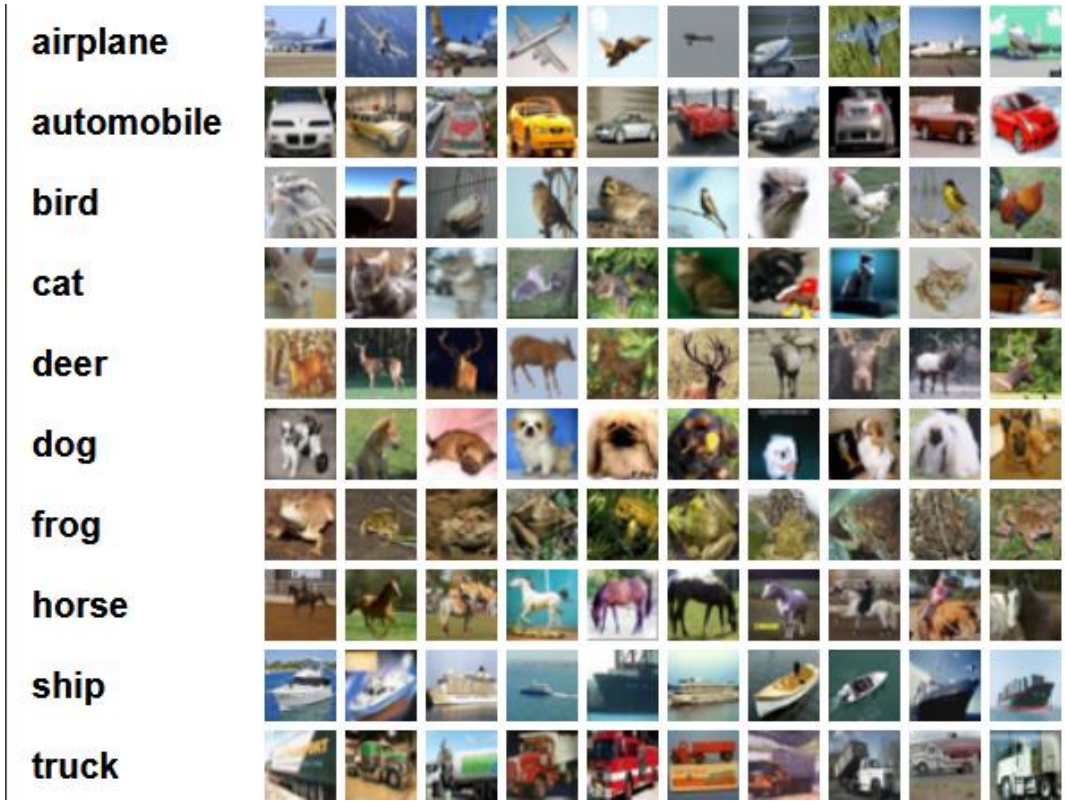


图 25-2 CIFAR10 训练库示例图片

OpenMV 的 IDE 有提供 CIFAR10 的神经网络模型，因此我们直接使用即可，位置在 IDE 的 **工具—机器视觉—CNN 网络模型库** 中。

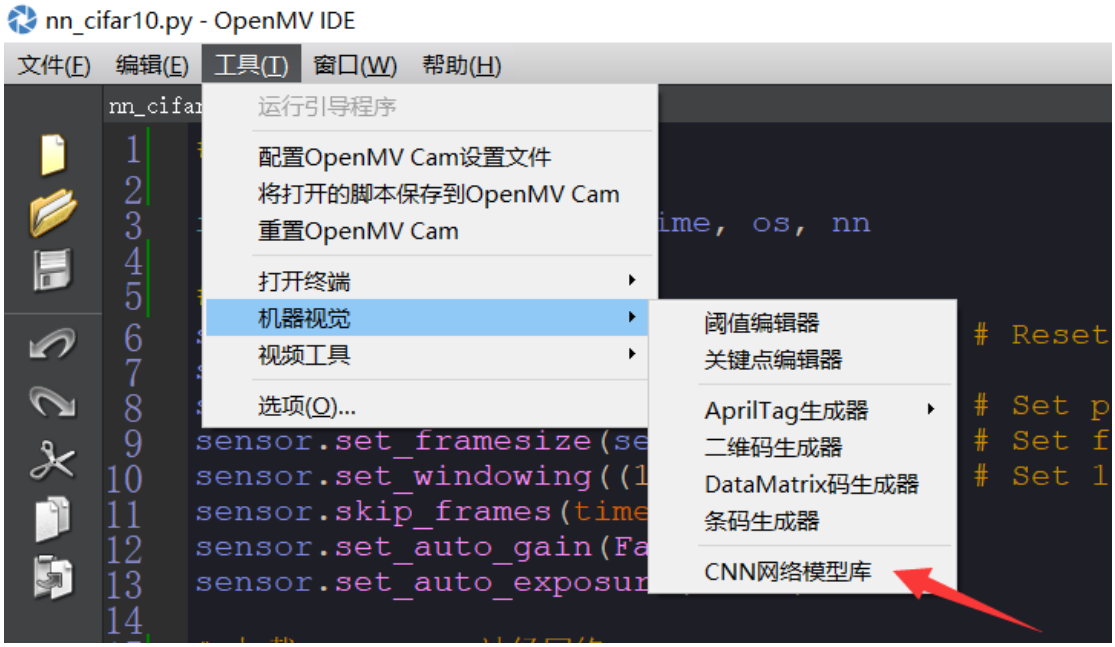


图 25-3 神经网络模型库位置

点击打开后看到有各类模型库，打开 CIFAR10 文件夹，看到了 cifar10.network 文件，这就是现成的模型。我们需要将它复制到 OpenMV4 的 U 盘（SD 卡）中去。

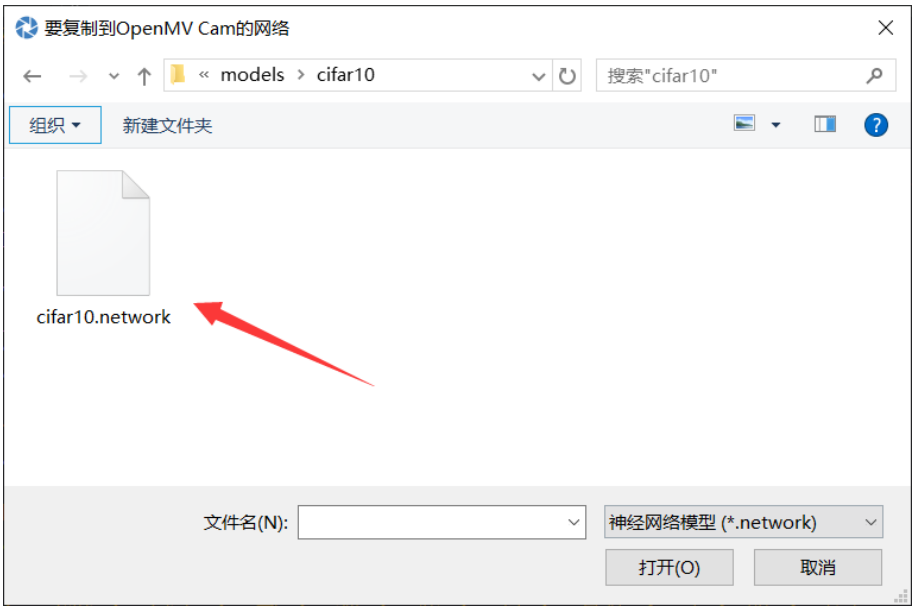


图 25-4 cifar10 神经网络模型文件

结合 MicroPython 的 OpenMV4 让机器学习变得非常简单易用，我们来看看

神经网络对象和使用方法：

构造函数
net = nn.load(*path*)
从指定路径中加载.network 的神经网络模型文件。 【path】 指定文件路径；根目录为 U 盘（SD 卡）。
使用方法
net.forward(*image*[, *roi*[, *softmax=False*[, *dry_run=False*]]])
在图像上运行神经网络并返回结果的浮点值列表；每个值的范围是 0 至 1，数值越大匹配度越高。 【image】指定要分析的图像； 【roi】指定图像区域(x,y,w,h)； 【softmax】设置 True，则列表中所有的输出总和为 1。否则，列表中的任何输出都可以在 0 和 1 之间。 【dry_run】设置为 True 以打印出正在执行的网络层而不是实际执行它们。这是为了调试 debug。

表 25-1　神经网络模型对象

本实验加载的是 CIFAR10 模型，因此使用神经网络函数处理会返回 10 个 0-1 的值，我们只需要找出最大值，就可判断当前图像和那个物品的匹配度最高了。代码编写流程如下：

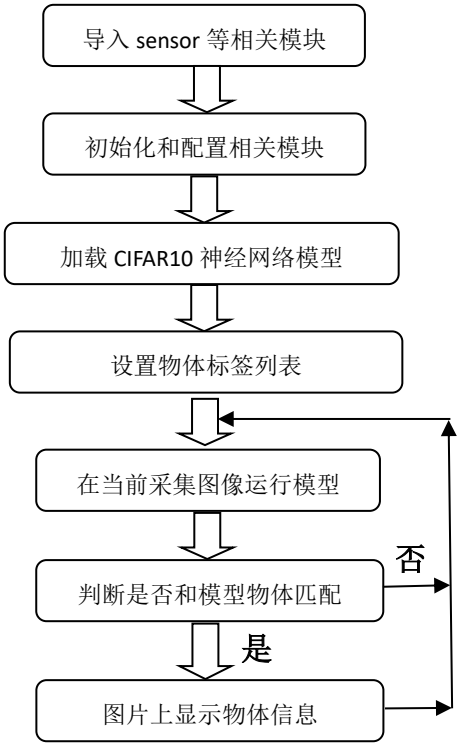


图 25-5 代码编写流程

官方参考代码路径【示例→Machine Learning→ nn_cifar10.py】，具体如下：

```
# CIFAR10 物体识别示例

import sensor, image, time, os, nn

#摄像头初始化
sensor.reset()                         # Reset and initialize the sensor.
sensor.set_contrast(3)
sensor.set_pixformat(sensor.RGB565)    # Set pixel format to RGB565
sensor.set_framesize(sensor.QVGA)      # Set frame size to QVGA
(320x240)
sensor.set_windowing((128, 128))       # Set 128x128 window.
sensor.skip_frames(time=1000)
```

```
sensor.set_auto_gain(False)
sensor.set_auto_exposure(False)

# 加载 cifar10 神经网络
net = nn.load('/cifar10.network')

# 更快、更小和计算量少
#net = nn.load('/cifar10_fast.network')

#10 种物体：“飞机”、“汽车”、“鸟”、“猫”、“鹿”、“狗”、“青蛙”,“马”,“船”,“卡车”
labels = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog',
'frog', 'horse', 'ship', 'truck']

clock = time.clock()        # Create a clock object to track the FPS.
while(True):
    clock.tick()                # Update the FPS clock.
    img = sensor.snapshot()     # Take a picture and return the image.

    #在当前图像运行神经网络，返回浮点值列表。列表共 10 个值，范围：0-1
    out = net.forward(img)

    #挑选列表中最大值
    max_idx = out.index(max(out))

    #将列表值乘以 100，方便计算，范围变成：0-100
    score = int(out[max_idx]*100)

    #70 分以上才认为识别成功
    if (score < 70):
        score_str = "??:??%"
```

```
else:
    score_str = "%s:%d%% "%(labels[max_idx], score)

#图像显示字符提示
img.draw_string(0, 0, score_str, color=(255, 0, 0))

print(clock.fps())
```

- **实验结果：**

记得将上文提到的 CIFAR10.network 文件拷贝到 OpenMV4 中。

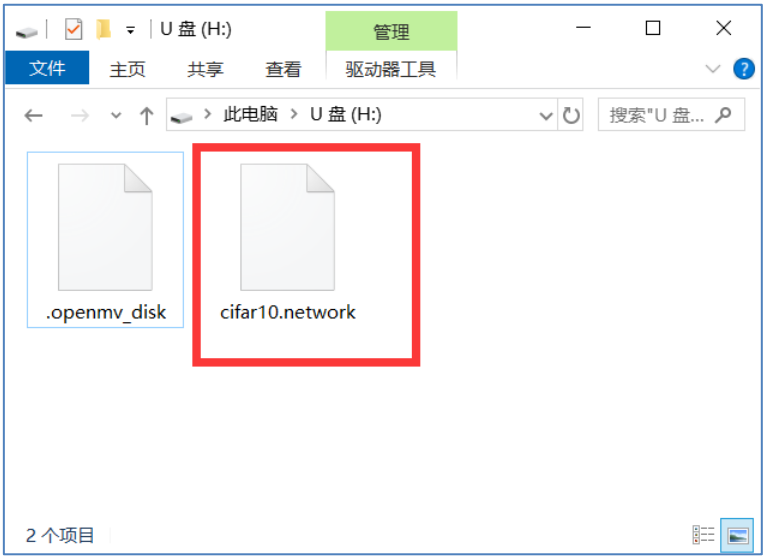


图 25-6

在 IDE 中运行程序，将摄像头对准要识别的物体，只能是默认的 10 类物体，可以看到当成功识别时候，图片上会显示当前物体信息和识别匹配率百分比。本实验使用飞机图片来进行识别测试：



图 25-7 飞机原图



图 25-8 实验图片

注意：由于受限于摄像头像素，实测本实验对大部分部分物体识别效果不是太好。测试可以尽量使用白底图片。

- **总结：**

物体识别作为 OpenMV4 的第一个个机器学习使用，可以看到使用 MicroPyhton 对象编程让机器学习变得非常简单，接下来我们会用通用的方式进行其它更多的机器学习实验。

25.2 笑脸识别

- **前言：**

人的脸部表情有很多，而笑脸是很常见的一个。本节我们来学习一下如何使用神经网络库结合前面人脸识别实验来识别笑脸。

- **实验平台：**

pyAI-OpenMV4 或 pyAI-OpenMV4 开发套件，需要使用 SD 卡。

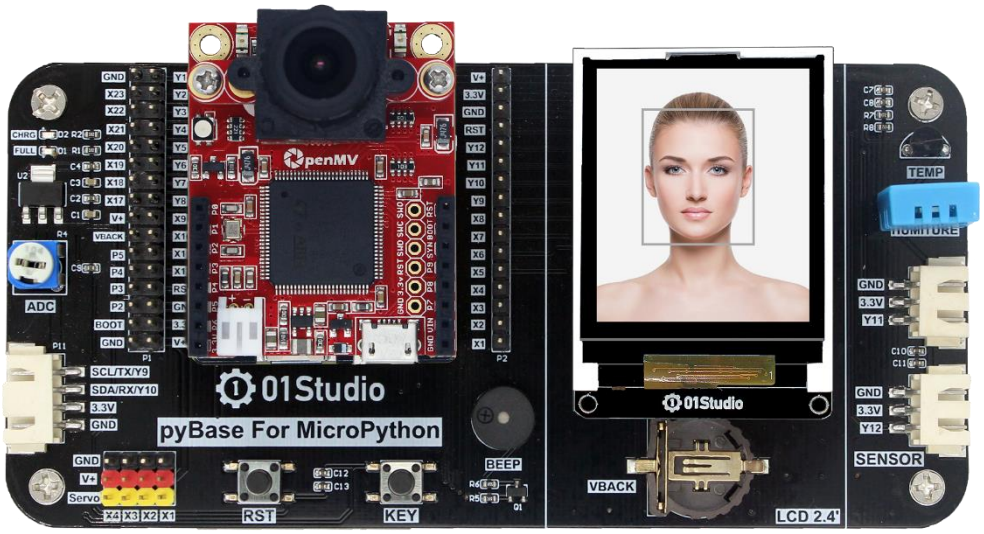


图 25-9 pyAI-OpenMV4 开发套件

- **实验目的：**

识别笑脸。

- **实验讲解：**

和上一节一样，OpenMV4 提供了现成的笑脸神经网络模型文件 smile.network。我们结合人脸检测实验先在图像中检测人脸并锁定区域，然后通过神经网络模型再判断当前人脸是不是一张笑脸即可。人脸检测请参考 **24.5.1 人脸检测**章节内容。

神经网络对象构建和使用方法如下：

构造函数
`net = nn.load(path)`
从指定路径中加载.network 的神经网络模型文件。

【path】 指定文件路径；根目录为 U 盘（SD 卡）。
使用方法
net.forward(*image*[, *roi*[, *softmax=False*[, *dry_run=False*]]])
在图像上运行神经网络并返回结果的浮点值列表；每个值的范围是 0 至 1，数值越大匹配度越高。 【image】指定要分析的图像； 【roi】指定图像区域(x,y,w,h)； 【softmax】设置 True，则列表中所有的输出总和为 1。否则，列表中的任何输出都可以在 0 和 1 之间。 【dry_run】设置为 True 以打印出正在执行的网络层而不是实际执行它们。这是为了调试 debug。

表 25-2　神经网络模型对象

本实验为人脸检测和 CNN（神经网络模型）的相结合，代码编写流程如下：

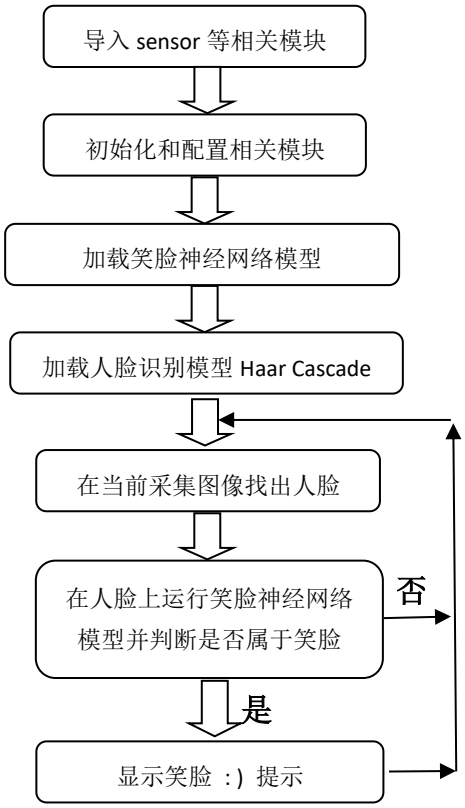


图 25-10 代码编写流程

官方参考代码路径【示例→Machine Learning→ nn_haar_smile_detection.py】，具体如下：

```
# 笑脸识别示例（ Haar Cascade + CNN 模型）.
#
#翻译和注释：01Studio

import sensor, time, image, os, nn

#设置摄像头
sensor.reset()                         # Reset and initialize the sensor.
sensor.set_contrast(2)
sensor.set_pixformat(sensor.RGB565)    # Set pixel format to RGB565
sensor.set_framesize(sensor.QVGA)      # Set frame size to QVGA
(320x240)
sensor.skip_frames(time=2000)
sensor.set_auto_gain(False)

# 加载笑脸识别神经网络模型
net = nn.load('/smile.network')

# 加载人脸识别模型 Haar Cascade
face_cascade = image.HaarCascade("frontalface", stages=25)
print(face_cascade)

# FPS clock
clock = time.clock()
while (True):
    clock.tick()
```

```
# Capture snapshot
img = sensor.snapshot()

# 找出图像中的人脸.
objects = img.find_features(face_cascade, threshold=0.75,
                                    scale_factor=1.25)

# 识别笑脸
for r in objects:

    # 重新构建识别的位置，缩小识别区域至人脸位置
    r = [r[0]+10, r[1]+25, int(r[2]*0.70), int(r[2]*0.70)]
    img.draw_rectangle(r)
    out = net.forward(img, roi=r, softmax=True)
    img.draw_string(r[0], r[1], ':)' if (out[0] > 0.8) else ':(',
                        color=(255), scale=2)

print(clock.fps())
```

- **实验结果：**

先将在 IDE 的 **工具—>机器视觉—>CNN 网络模型库** 中的笑脸模型文件 smile.network 复制到 U 盘（SD 卡）。

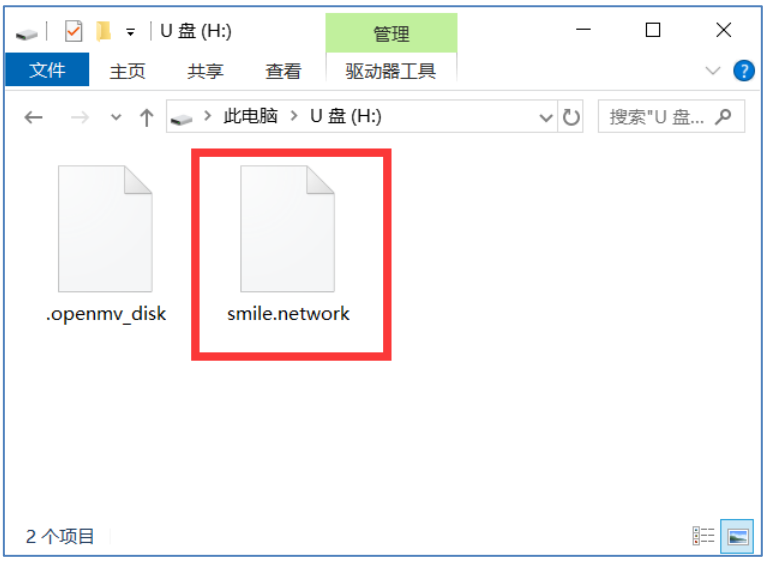


图 25-11 CNN 神经网络笑脸模型文件

在 IDE 上运行代码，可以看到当识别到笑脸时，图像上出现“：)”的提示。



图 25-12 笑脸原图



图 25-13 实验图片

- **总结：**

笑脸识别依赖于笑脸模型，本实验代码也同时支持多张人脸同时识别，有兴趣的用户可以自行测试。

25.3 数字识别

- **前言：**

数字识别，简单来说就是要识别阿拉伯数字 0-9，日常生活中非常多的数据都是用数字表示，本节我们就来学习如何使用 OpenMV4 进行简单的数字识别实验。

- **实验平台：**

pyAI-OpenMV4 或 pyAI-OpenMV4 开发套件，需要使用 SD 卡。

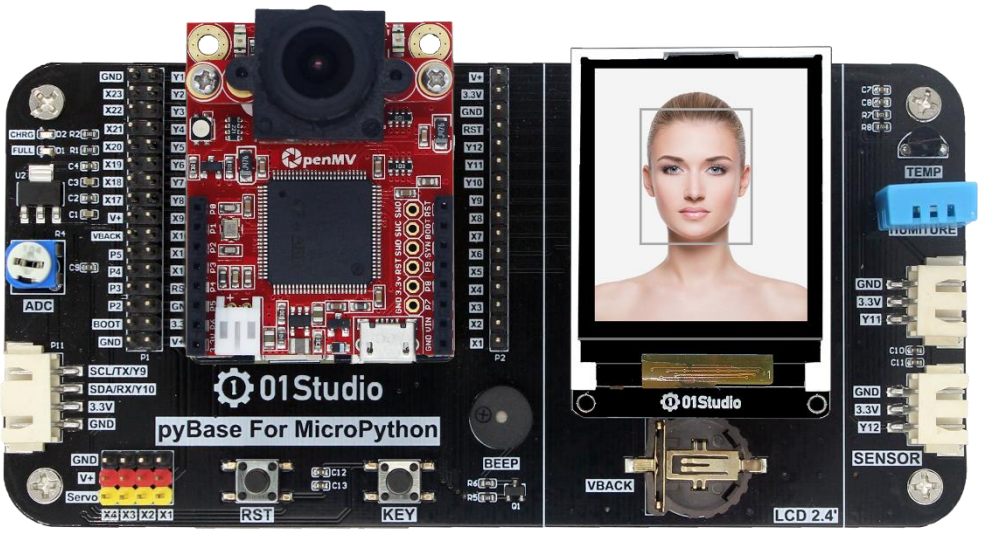


图 25-14 pyAI-OpenMV4 开发套件

- **实验目的：**

识别阿拉伯数字。

- **实验讲解：**

和前面的机器学习识别实验一样，数字识别也有自己的神经网络模型，本实验使用的是 lenet 模型。对应 IDE 菜单 **工具—机器视觉—CNN 网络模型库** 的 lenet.network 文件。

神经网络对象构建和使用方法如下：

构造函数
`net = nn.load(path)`
从指定路径中加载.network 的神经网络模型文件。

【path】 指定文件路径；根目录为 U 盘（SD 卡）。
使用方法
net.forward(*image*[, *roi*[, *softmax=False*[, *dry_run=False*]]])
在图像上运行神经网络并返回结果的浮点值列表；每个值的范围是 0 至 1，数值越大匹配度越高。 【image】指定要分析的图像； 【roi】指定图像区域(x,y,w,h)； 【softmax】设置 True，则列表中所有的输出总和为 1。否则，列表中的任何输出都可以在 0 和 1 之间。 【dry_run】设置为 True 以打印出正在执行的网络层而不是实际执行它们。这是为了调试 debug。

表 25-3　神经网络模型对象

编程方法和前面实验一样，导入 lenet 模型，然后对采集到的图像与模型数据进行匹配，判断当前图像中是否有出现数字和可信度。代码编写流程如下：

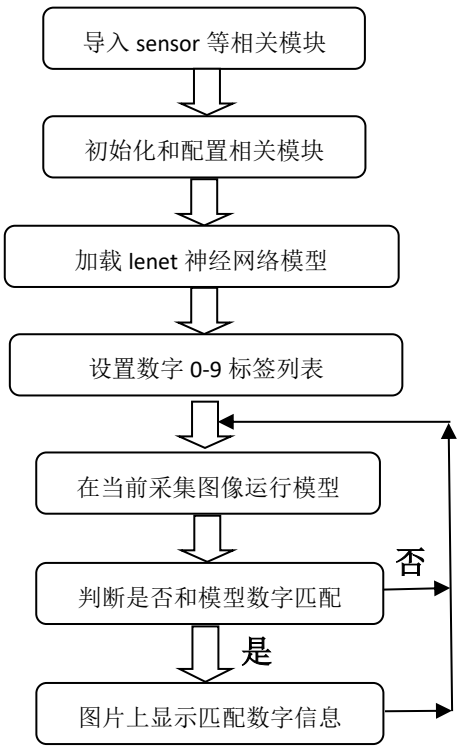


图 25-15 代码编写流程

官方参考代码路径【示例→Machine Learning→ nn_lenet.py】，具体如下：

```
# lenet 模型数字识别例程
#
#翻译和注释：01Studio

import sensor, image, time, os, nn

#摄像头初始化
sensor.reset()                      # Reset and initialize the sensor.
sensor.set_contrast(3)
sensor.set_pixformat(sensor.GRAYSCALE) # Set pixel format to RGB565 (or
GRAYSCALE)
sensor.set_framesize(sensor.QVGA)     # Set frame size to QVGA
(320x240)
sensor.set_windowing((128, 128))      # Set 128x128 window.
sensor.skip_frames(time=100)
sensor.set_auto_gain(False)
sensor.set_auto_exposure(False)

# 加载 lenet 神经网络模型
net = nn.load('/lenet.network')
labels = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']

clock = time.clock()        # Create a clock object to track the FPS.
while(True):
    clock.tick()              # Update the FPS clock.
    img = sensor.snapshot()    # Take a picture and return the image.

    #copy()表示创建一个图像副本储存在 MicroPython 堆中而不是帧缓冲区
```

```
#二值化是为了方便处理，阈值可以自己设定。
out = net.forward(img.copy().binary([(150, 255)], invert=True))
max_idx = out.index(max(out))
score = int(out[max_idx]*100)

#大于 70 分认为识别可信
if (score < 70):
    score_str = "??:??%"
else:
    score_str = "%s:%d%% "%(labels[max_idx], score)
img.draw_string(0, 0, score_str)

print(clock.fps())
```

- **实验结果：**

先将在 IDE 的 **工具—>机器视觉—>CNN 网络模型库** 中的 lenet 模型文件 lenet.network 复制到 U 盘（SD 卡）。

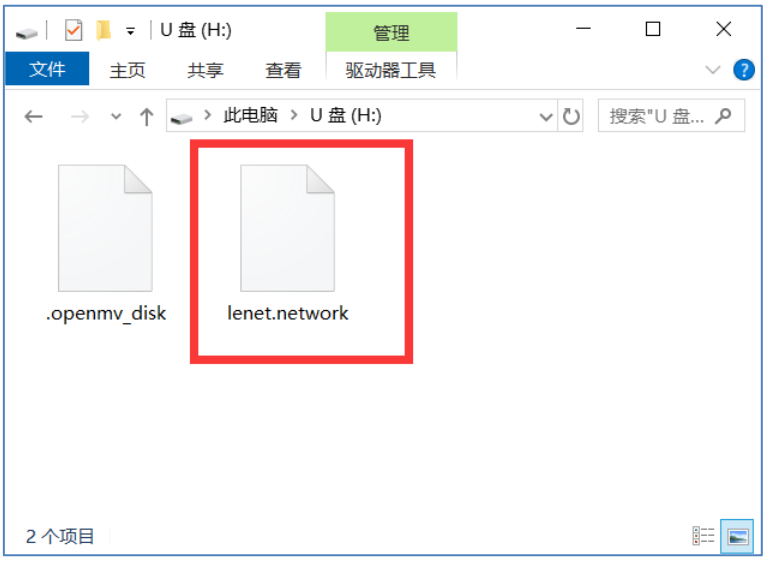


图 25-16

在 IDE 上运行程序，可以看到当 OpenMV4 识别到数字时，图像显示识别的数字和识别可信度。

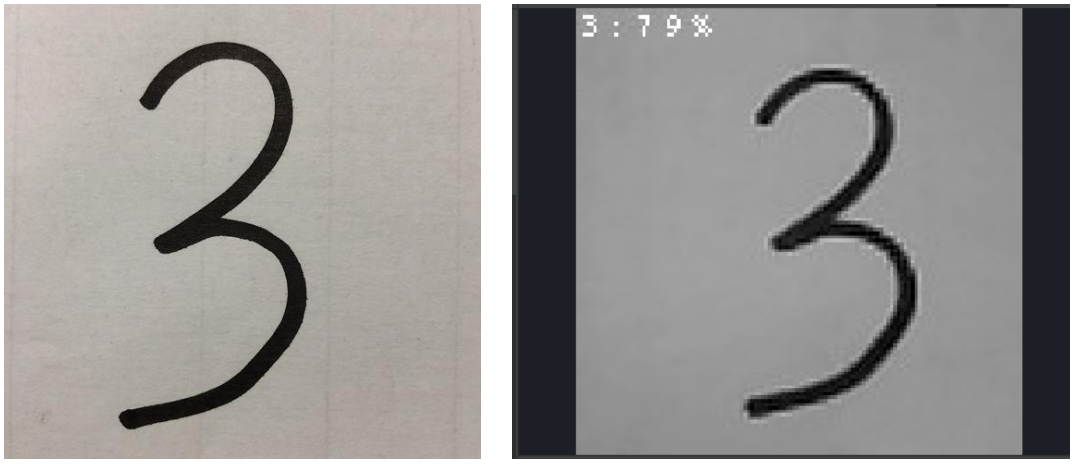


图 25-17 数字识别实验 原图（左）和实验图片（右）

- **总结：**

在几个机器学习实验下来，我们发现都是在使用同一个 CNN 神经网络对象函数来编程，只是加载了不同的模型。MicroPython 面向对象编程让我们可以不用管底层实现原理，直接调用即可实现相关机器学习功能，大大缩短了开发流程。

第26章　拓展模块

由于 OpenMV 的集成度非常高，因此为了完全更多的外扩功能，就需要通过拓展模块来实现。如 LCD 显示屏、舵机控制、WiFi 模块、摄像头镜头等。本章将对 OpenMV4 的常用拓展模块进行讲解。

26.1 LCD 显示屏

- **前言：**

我们前面的实验实时观察图像都是使用 OpenMV IDE 的实时缓冲区，有些时候我们需要离线观察，这时候可以使用 LCD，可以实现本地显示。这大大方便调试和应用。

- **实验平台：**

pyAI-OpenMV4 开发套件 + 01Studio 1.77（128x160） LCD 显示屏。

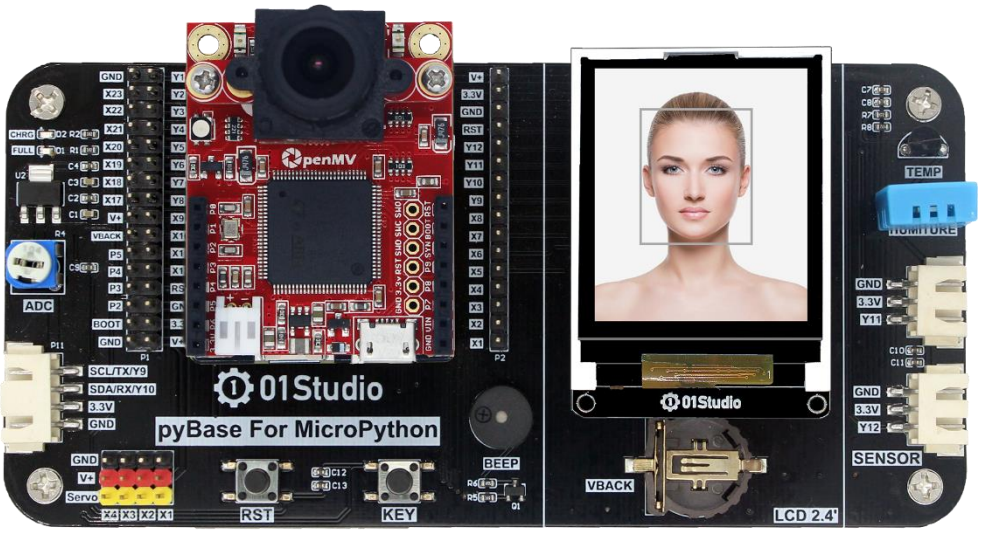


图 26-1 pyAI-OpenMV4 开发套件

- **实验目的：**

使用 LCD 实时显示摄像头采集的图像。

- **实验讲解：**

OpenMV IDE 集成了 LCD 驱动模块，目前只支持 1 款，分辨率必须为（128*160），SPI 接口的 LCD 模块。其对象说明如下：

构造函数
`import lcd`
导入 lcd 模块

使用方法
lcd.init(*type=1*)
初始化 lcd 模块。 【type】表明 lcd 扩展板的类型； 0: None 1: lcd（128*160） ***目前仅支持 1 中类型的 LCD**
lcd.deinit()
反初始化 lcd 模块，释放 GPIO 引脚。
lcd.display(*image, roi=Auto*)
在 lcd 上显示图片。 【image】显示的图片； 【roi】显示区域(x,y,w,h)。 *更多关于 LCD 使用方法说明，请阅读官方文档： http://docs.openmv.io/library/omv.lcd.html

表 26-1　LCD 对象

结合前面学习过的摄像头的应用，本实验的代码编写流程如下：

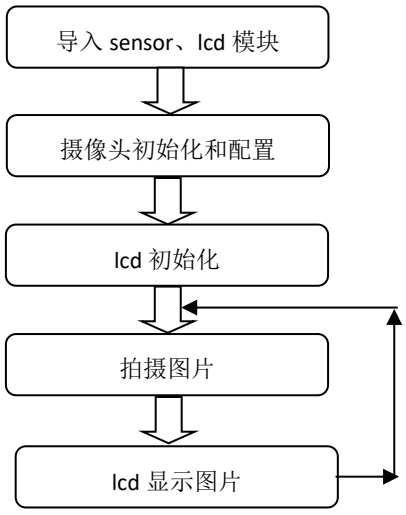


图 26-2 代码编写流程图

官方参考代码路径【示例→LCD-Shield→ lcd.py】，具体如下：

```
# LCD 示例程序
#
# 提示: 这个实验需要外接一个 LCD 模块.
#
# 本示例实现了 LCD 实时显示摄像头帧缓冲画面.
#
#翻译和注释：01Studio

import sensor, image, lcd

#摄像头初始化
sensor.reset() # 初始化摄像头.
sensor.set_pixformat(sensor.RGB565) # 或者 sensor.GRAYSCALE
sensor.set_framesize(sensor.QQVGA2) # LCD 的分辨率为 128x160 .

#LCD 初始化
lcd.init()

while(True):
    lcd.display(sensor.snapshot()) # 拍照和显示图像.
```

- **实验结果：**

在 IDE 中运行代码或者将代码文件修改成 main.py 拷贝到 OpenMV4 里面，可以看到 LCD 实时显示采集图像。

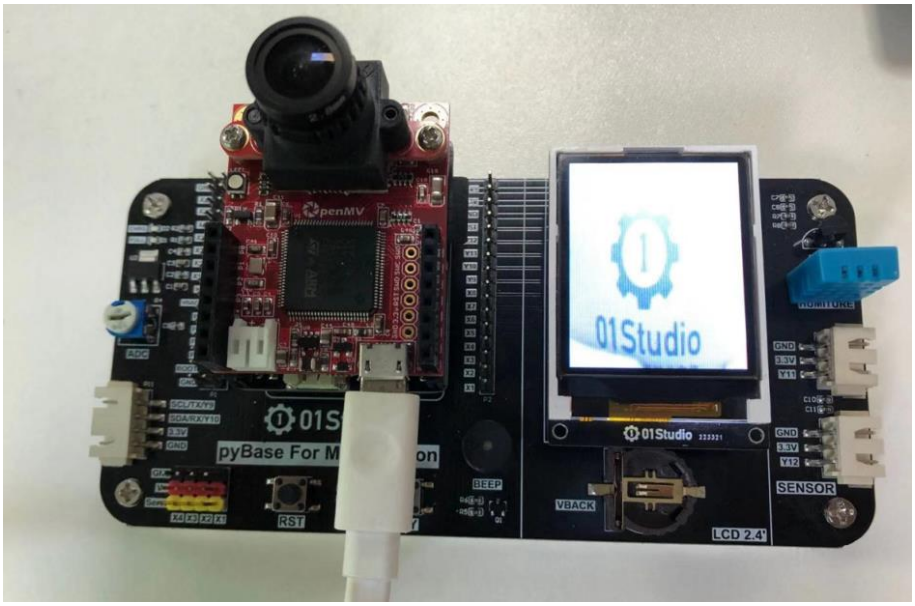


图 26-3 LCD 实时显示

你也可以直接将 LCD 显示屏插在 pyAI-OpenMV4 的背面来进行拍摄。

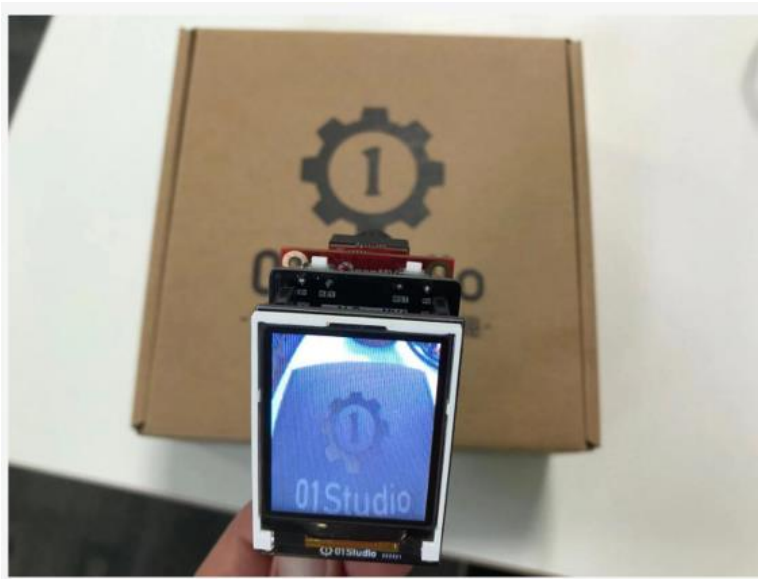


图 26-4 直接用 pyAI-OpenMV4 接 LCD 模块拍摄

- **总结：**

本节学习了 LCD 的使用，也是做了最基础的例子。其实 LCD 的用途非常广泛，我们前面学习的 OpenMV4 的大部分例程只要修改一下图像分辨率参数，全部都可以用这个 LCD 来实时显示。比如巡线实验，可以通过 LCD 实时显示线路轨迹。有兴趣的用户可以自行修改测试。

26.2 舵机

- **前言：**

舵机又叫伺服电机，是一个可以旋转特定角度的电机，可转动角度通常是 90°、180° 和 360°（360° 可以连续旋转）。我们看到的机器人身上就有非常多的舵机，它们抬手或者摇头的动作往往是通过舵机完成，因此机器人身上的舵机越多，意味着动作越灵活。

- **实验平台：**

pyAI-OpenMV4 开发套件和舵机。需要接上锂电池。

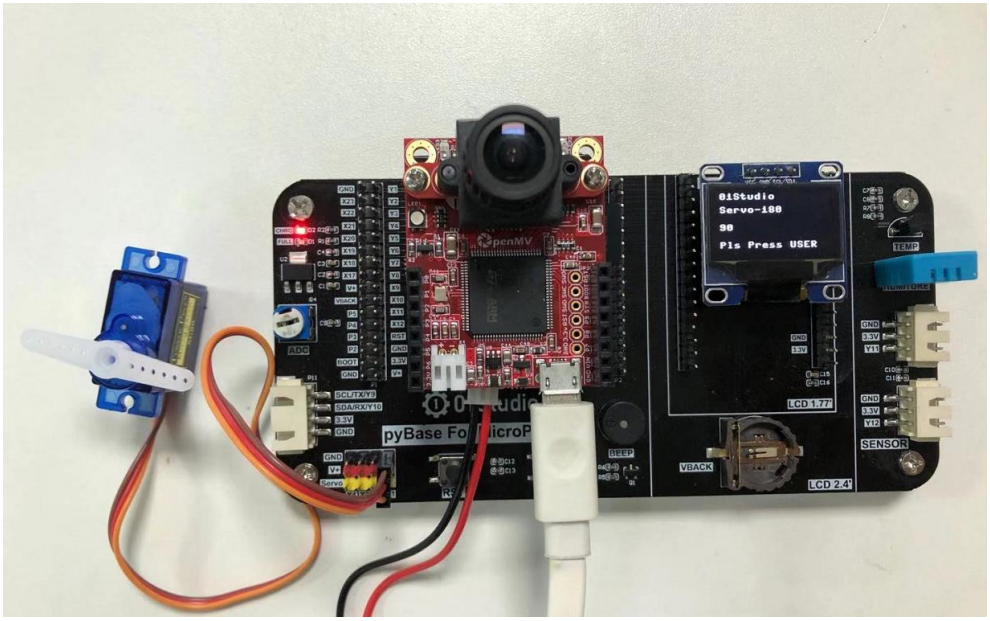


图 26-5 舵机接线

通常情况下：**黑色表示 GND，红色表示 VCC，橙色表示信号线。**

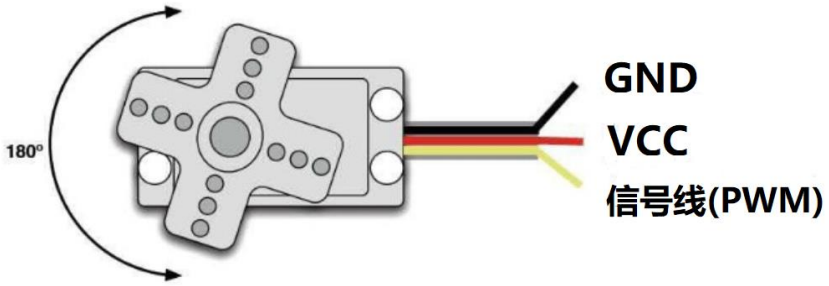


图 26-6 舵机示意图

- **实验目的：**

通过编程实现对舵机的控制。

- **实验讲解：**

伺服电机对象通过 3 线（信号，电源，地）控制，pyAI-OpenMV4 核心板有 3 个引脚可以用于直接控制舵机，分别连接到 pyBase 底板的 X1,X2,X3 引脚。对应关系是 P7→Servo1→X1，P8→Servo2→X2，P9→Servo3→X3。

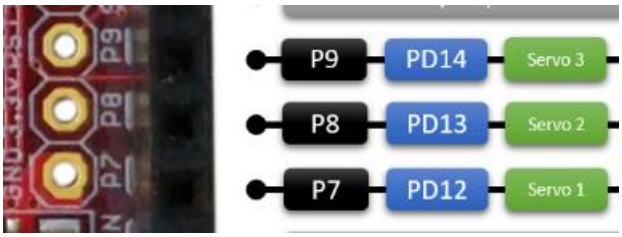


图 26-7　pyAI-OpenMV4 舵机控制引脚

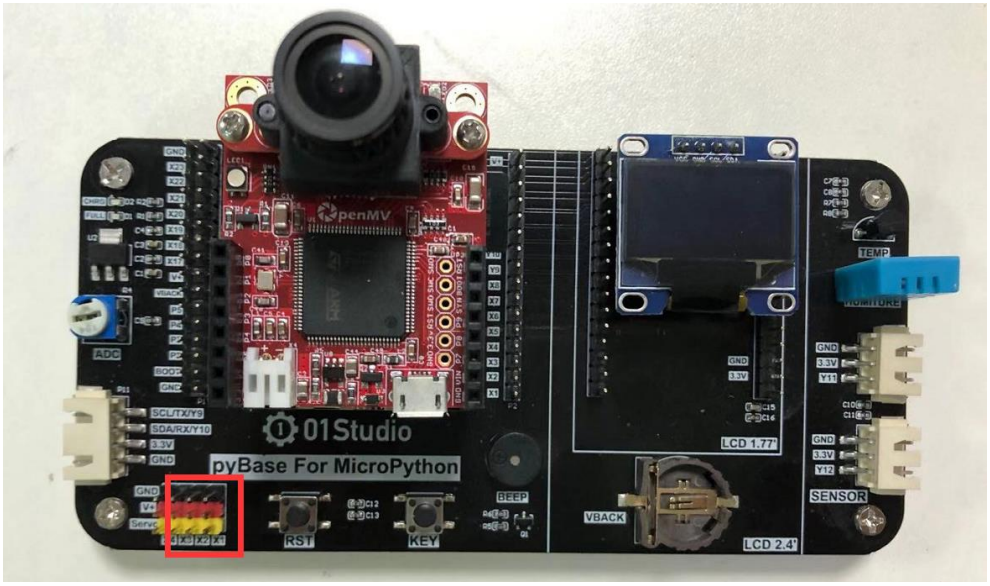


图 26-8 pyBase 对应的 3 路舵机接口

180°舵机的控制一般需要一个 20ms 左右的时基脉冲，该脉冲的高电平部分一般为 0.5ms-2.5ms 范围内的角度控制脉冲部分，总间隔为 2ms。以 180 度角度伺服为例，在 MicroPython 编程对应的控制关系是从-90°至 90°，示例图如下：

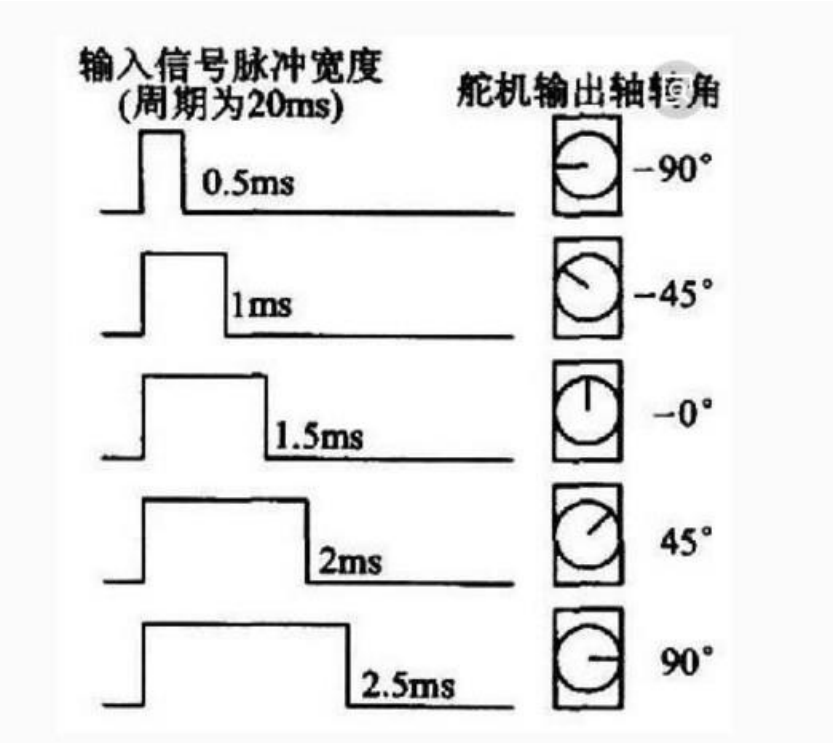


图 26-9 PWM 与角度关系

而对于 360° 连续旋转舵机，上面的脉冲表则对应从正向最大速度旋转到反向最大速度旋转的过程。

OpenMV 自带了舵机（Servo）的对象模块，我们可以直接调用，下面是 Servo 对象说明。

构造函数
s=pyb.Servo(id)
构建舵机对象。id 是 1-4，对应引脚 X1-X4
使用方法
s.angle (angle,time=0)
角度控制。angle:角度[-90 至 90]；time:旋转到指定角度的时间，单位 ms，默认 0 表示最快
s.speed(speed,time=0)
速度控制（针对 360 度连续旋转舵机）。speed:速度[-100 至 100]，time：开始启动到指定速度的时间，单位 ms，默认 0 表示以最快

表 26-2 舵机对象

可以看到 MicroPython 让舵机控制变得非常简单，代码编程流程图如下：

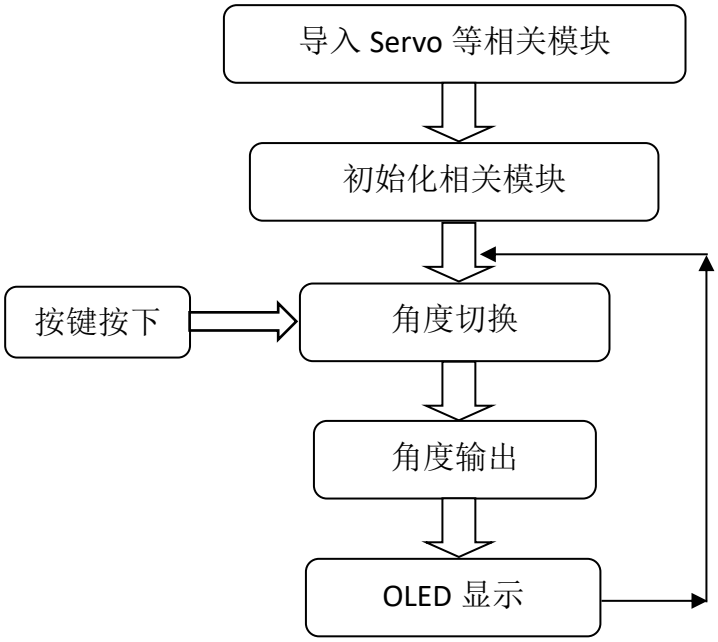


图 26-10 编程流程图

参考代码如下：

```
'''
实验名称：舵机(Servo)-180°
版本：v1.0
日期：2019.9
作者：01Studio
说明：通过 USER 按键让让舵机旋转不同角度。
'''

#导入相关模块
from pyb import Servo,ExtInt
from machine import Pin,I2C
from ssd1306x import SSD1306_I2C
```

```
#初始化相关模块
i2c = I2C(sda=Pin("P0"), scl=Pin("P2"), freq=80000)
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

s1 = Servo(1)     #构建舵机对象 s1，输出引脚为 X1

#定义 5 组角度：-90、-45、0、45、90
angle=[-90,-45,0,45,90]

key_node = 0  #按键标志位
i = 0         #用于选择角度

################################################
#  按键和其回调函数
################################################
def key(ext):
    global key_node
    key_node = 1

ext = ExtInt(Pin('P9'), ExtInt.IRQ_FALLING, Pin.PULL_UP, key) #下降沿触
发，打开上拉电阻

################################################
#  OLED 初始显示
################################################
oled.fill(0)  # 清屏显示黑色背景
oled.text('01Studio', 0, 0)  # 首行显示 01Studio
oled.text('Servo-180', 0, 15)  # 次行显示实验名称
oled.text(str(angle[i]) , 0, 35)  # 显示当前角度
```

```
oled.text('Pls Press USER', 0, 55)  #提示按按键
oled.show()

#X1 指定角度,启动时 i=0，默认-90°
s1.angle(angle[i])

while True:

    if key_node==1: #按键被按下
        i = i+1
        if i == 5:
            i = 0
        key_node = 0 #清空按键标志位

        #X1 指定角度
        s1.angle(angle[i])

        #显示当前频率
        oled.fill(0)  # 清屏显示黑色背景
        oled.text('01Studio', 0, 0)  # 首行显示 01Studio
        oled.text('Servo-180', 0, 15)  # 次行显示实验名称
        oled.text(str(angle[i]) , 0, 35)  # 显示当前角度
        oled.text('Pls Press USER', 0, 55)  #提示按按键
        oled.show()
```

- **实验结果：**

将实验代码拷贝到 OpenMV4。包括主函数代码和 OLED 驱动代码。

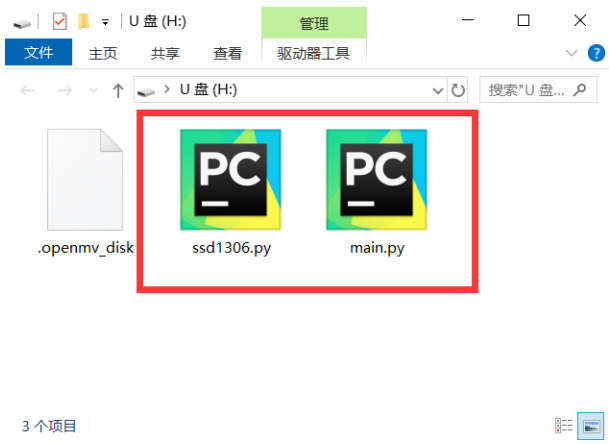


图 26-11

由于 OpenMV4 没有电源 V+输出引脚，因此需要让 pyAI-OpenMV4 接上锂电池，从而给 pyBase 的 V+供电。当然你可以将程序拷贝到 OpenMV4 后，将 USB 口插到 pyAI-OpenMV4 转接板。

将 180° 舵机插到 X1 三线接口，通过按键按下实现舵机的不同角度，同时在 OLED 上显示角度值。

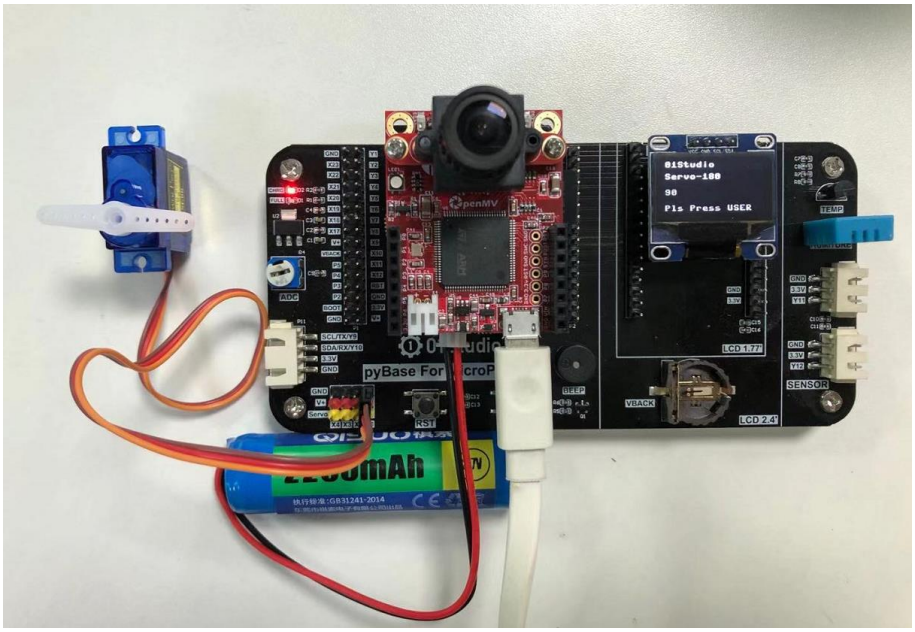


图 26-12 舵机角度 90°

- **实验拓展：**

我们刚刚实现了 180° 舵机的角度控制，现在来做一下 360° 连续旋转舵机

的实验，360°连续旋转舵机可以实现直流减速电机功能，用在小车或者航模上。

实验的基础条件和方式不变，我们只需要将角度参数【-90 至 90】改成速度【-100 至 100】即可。修改后的程序代码如下：

```
'''
实验名称：舵机(Servo)-360°连续旋转
版本：v1.0
日期：2019.9
作者：01Studio
说明：通过 USER 按键让舵机实现不同速度旋转。
'''

#导入相关模块
from pyb import Servo,ExtInt
from machine import Pin,I2C
from ssd1306x import SSD1306_I2C

#初始化相关模块
i2c = I2C(sda=Pin("P0"), scl=Pin("P2"), freq=80000)
oled = SSD1306_I2C(128, 64, i2c, addr=0x3c)

s1 = Servo(1)     #构建舵机对象 s1，输出引脚为 X1

#定义 5 组速度值
speed=[-100,-50,0,50,100]

key_node = 0  #按键标志位
i = 0         #用于选择角度

################################################
```

```
#  按键和其回调函数
################################################
def key(ext):
    global key_node
    key_node = 1

#下降沿触发，打开上拉电阻
ext = ExtInt(Pin('P9'), ExtInt.IRQ_FALLING, Pin.PULL_UP, key)

################################################
#  OLED 初始显示
################################################
oled.fill(0)  # 清屏显示黑色背景
oled.text('01Studio', 0, 0)  # 首行显示 01Studio
oled.text('Servo-360', 0, 15)  # 次行显示实验名称
oled.text(str(speed[i]) , 0, 35)  # 显示当前速度值
oled.text('Pls Press USER', 0, 55)  #提示按按键
oled.show()

#X1 指定速度，启动时 i=0，默认-100
s1.speed(speed[i])

while True:

    if key_node==1: #按键被按下
        i = i+1
        if i == 5:
            i = 0
        key_node = 0 #清空按键标志位
```

```
    #X1 指定角度
    s1.speed(speed[i])


    #显示当前频率
    oled.fill(0)  # 清屏显示黑色背景
    oled.text('01Studio', 0, 0)  # 首行显示 01Studio
    oled.text('Servo-360', 0, 15)  # 次行显示实验名称
    oled.text(str(speed[i]) , 0, 35)  # 显示当前速度值
    oled.text('Pls Press USER', 0, 55)  #提示按按键
    oled.show()
```

实验结果如下：

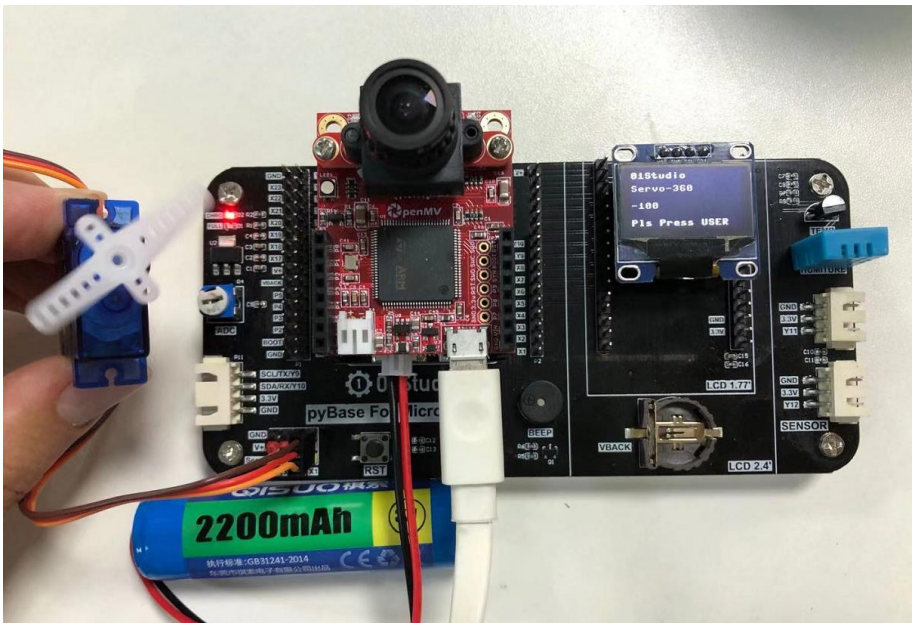


图 26-13　360°连续旋转舵机

- **总结：**

通过本节我们学会了使用不同类型的舵机，通过多路电机的组合使用，可以实现模型、航模、小车、机器人的实验。

26.3 多路舵机模块

- **前言：**

在上一节我们已经学习了舵机的应用。今天我们来学习一下如何同时控制多路舵机。

- **实验平台：**

pyAI-OpenMV4 开发套件、OpenMV 舵机模块和舵机。需要接上锂电池。

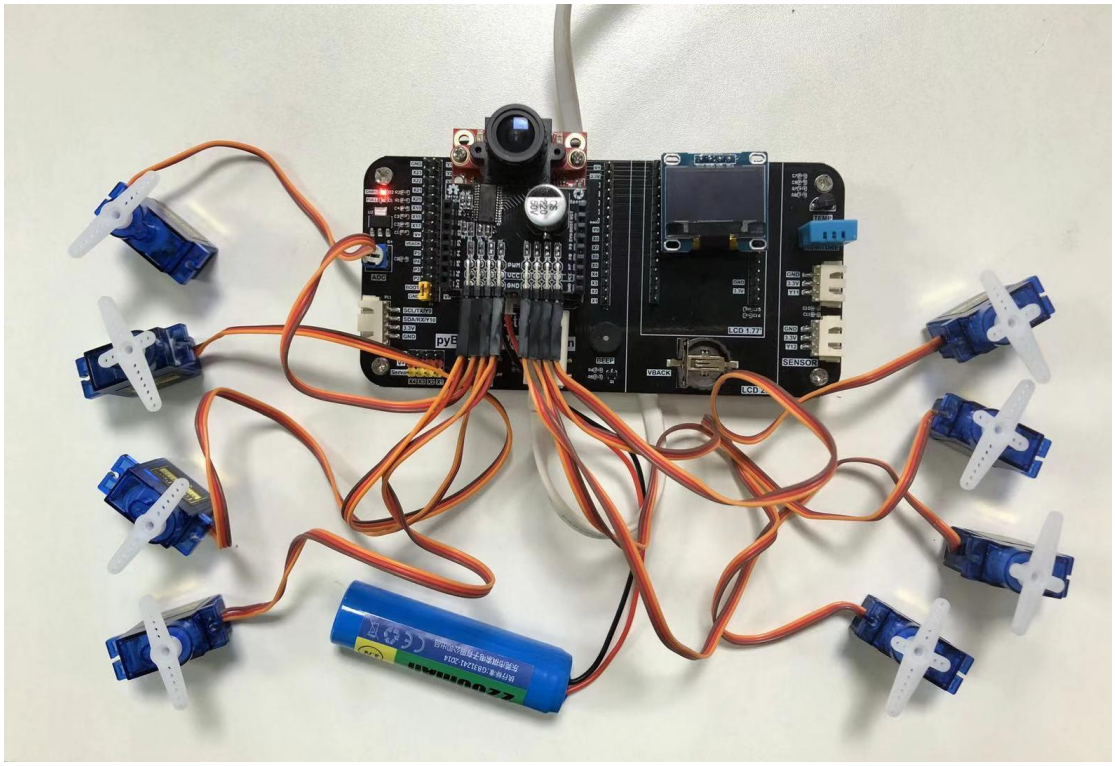


图 26-14 8 路舵机接线

- **实验目的：**

通过编程实现对 8 路舵机的控制。

- **实验讲解：**

我们这上一节已经学习过舵机的原理了，这里不再重复。OpenMV 舵机拓展模块是使用了 PCA9685 芯片，通过 I2C 跟 OpenMV 通讯，使得 OpenMV 只需要一个 I2C 接口就可以同时控制多路舵机。舵机模块功能参数说明如下：

功能参数	
主控芯片	PCA9685
控制方式	I2C 总线
输出接口	8 路舵机（PWM）
其他功能	支持级联，最多支持 16 路输出

表 26-3 舵机模块功能参数

OpenMV 集成了基于 PCA9685 舵机驱动的对象模块，我们可以直接调用，下面是对象相关说明。

构造函数
`servo = Servos(i2c, address=0x40, freq=50, min_us=500, max_us=2500, degrees=180)`
构建舵机模块对象。 【i2c】控制的 I2C 接口，默认接口是 I2C2,即 sda=Pin('P5'), scl=Pin('P4')； 【address】I2C 地址，默认是 0x40 （拓展板背部有焊点，焊接后地址为 0x60，方便同时控制 16 路舵机） 【freq】根据舵机配置，我们使用舵机周期是 20ms，即频率 50Hz； 【min_us】根据舵机配置，占空比最低 0.5ms,即 500us 【max_us】根据舵机配置，占空比最低 2.5ms,即 2500us 【degrees】舵机初始化角度。
使用方法
`servo.position(i, 0)`
舵机控制。 【i】0-7,分别对应舵机 S0-S7。总共 8 路； 【0】角度。0-180。

表 26-4 舵机模块对象

可以看到通过 I2C 可以轻松控制多路电机，代码编程流程图如下：

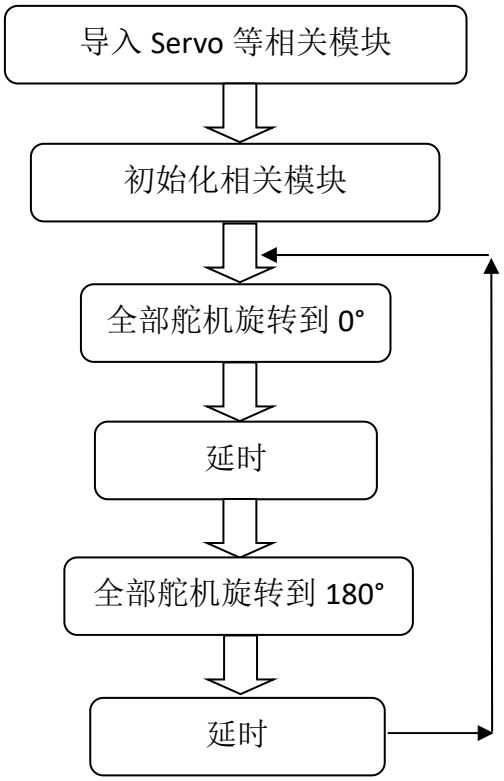


图 26-15 编程流程图

官方参考代码路径【示例→Servo-Shield→ main.py】，注意舵机初始化参数需要修改成我们使用的舵机参数。具体如下：

```
# 舵机模块例程.
#
# 本实验请遵循以下要求:
#
#   1. 将舵机连接到任意 PWM 输出口.
#   2. 需要连接 3.7V 锂电池 (或 5V 电源).
#   3. 将 pca9685.py 和 servo.py 文件拷贝到 OpenMV 的文件系统，然后复位.
#   4. 在 IDE 里运行本实验代码.
#
#翻译和注释：01Studio


import time
from servo import Servos
from machine import I2C, Pin
```

```
#初始化 I2C
i2c = I2C(sda=Pin('P5'), scl=Pin('P4'))

#初始化舵机模块
servo = Servos(i2c, address=0x40, freq=50, min_us=500, max_us=2500,
degrees=180)

while True:
    for i in range(0, 8):
        servo.position(i, 0) #8 路舵机旋转到 0°
    time.sleep(500)
    for i in range(0, 8):
        servo.position(i, 180)#8 路舵机旋转到 180°
    time.sleep(500)
```

- **实验结果：**

接上锂电池。将 pca9685.py 和 servo.py 文件拷贝到 OpenMV 的文件系统，然后复位。在 IDE 运行以上代码，可以看到 8 路舵机依次旋转到 0 度和 180 度。

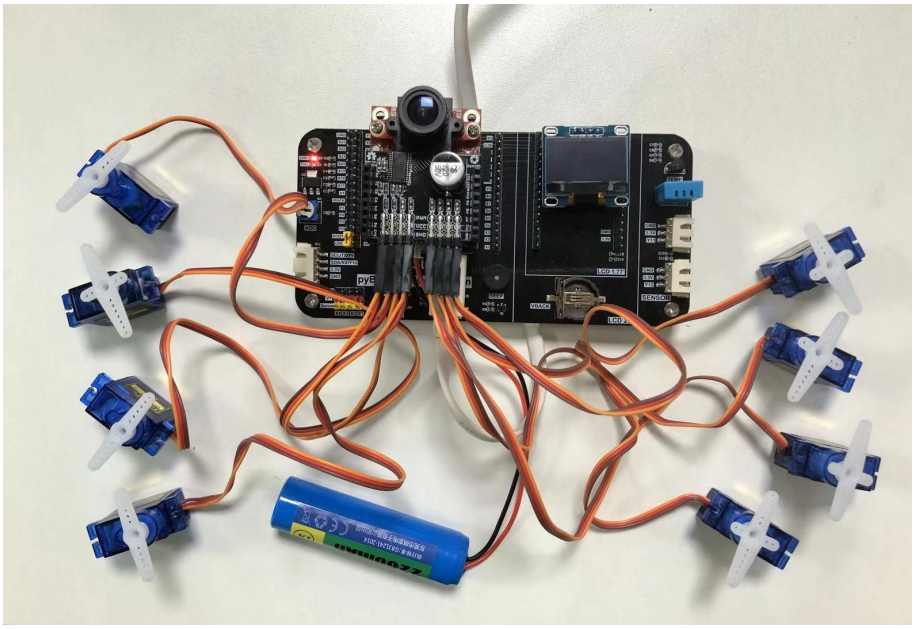


图 26-16

- **实验拓展：**

在 8 路舵机模块的背面有一个焊接点，将这个焊接上可以改变其 I2C 地址，从而实现 2 个 8 路舵机模块叠加，控制 16 路舵机。焊接焊点后的模块 I2C 地址为 0x60，有兴趣的用户可以自行测试。

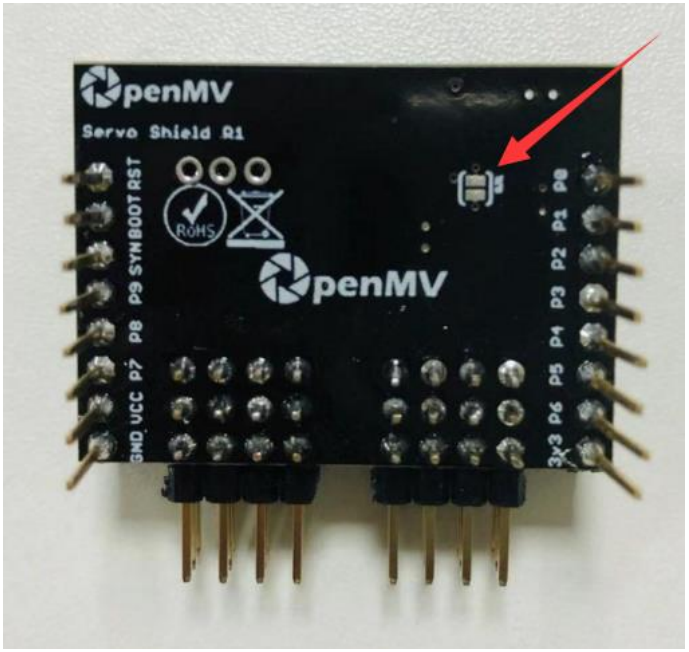


图 26-17　8 路舵机模块焊点

- **总结：**

本节学习了多路舵机的编程应用，可以见到通过简单的 I2C 实现了多路舵机控制，本模块也适用于其它平台的 MicroPython 开发板。

26.4 WiFi 模块

OpenMV 可以通过 WiFi 扩展模块连接无线路由器，从而实现 Socket、MQTT、图传等相关无线和物联网应用。模块的接口和功能参数介绍如下：



图 26-18 OpenMV WiFi 模块 PinOut 图

功能参数	
WiFi 主控	WINC1500
控制方式	SPI 接口
通信速率	IEEE 802.11b (up to 11Mbps) IEEE 802.11g (up to 48Mbps) IEEE 802.11n (up to 48Mbps)
加密类型	IEEE 802.11 WEP IEEE 802.11 WPA IEEE 802.11 WPA2
功耗	待机 80mA(3.3V) 工作 100mA(3.3V)
适用平台	pyAI-OpenMV4

表 26-5 OpenMV WiFi 模块功能参数

26.4.1 连接无线路由器

- **前言：**

WIFI 是物联网中非常重要的角色，现在基本上家家户户都有 WIFI 网络了，通过 WIFI 接入到互联网，成了智能家居产品普遍的选择。而要想上网，首先需要连接上无线路由器。这一节我们就来学习如何通过 WiFi 扩展模块让 OpenMV 通过 MicroPython 编程实现连上路由器。

- **实验平台：**

pyAI-OpenMV4 和 OpenMV WiFi 模块。（如果使用 pyBase 需要将 OLED 拔掉。避免 IO 冲突）

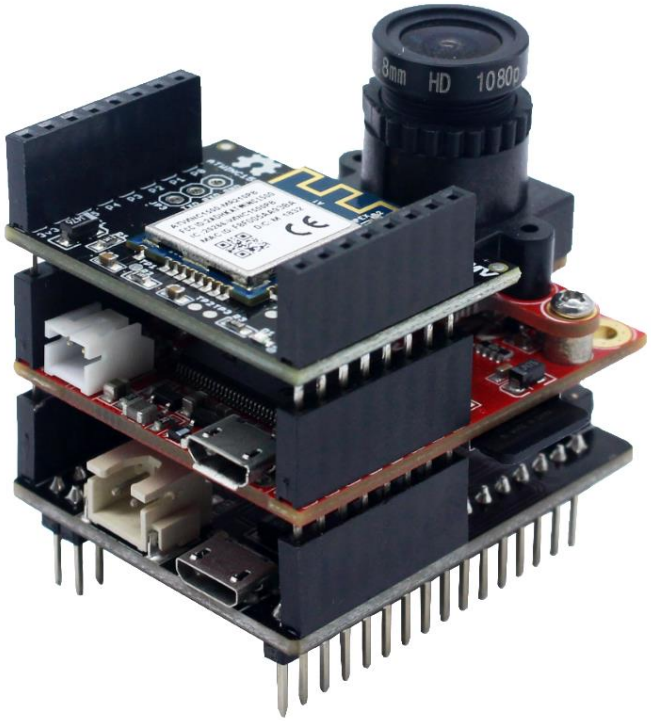


图 26-19 pyAI-OpenMV4

- **实验目的：**

编程实现连接路由器，将 IP 地址等相关信息通过串口终端打印（只支持 2.4G 网络）。

- **实验讲解：**

连接路由器上网是我们每天都做的事情，日常生活中我们只需要知道路由器

的账号和密码，就能使用电脑或者手机连接到无线路由器，然后上网冲浪。

OpenMV 已经集成了 network 模块，开发者使用内置的 network 模块函数可以非常方便地连接上路由器。

我们先来看看 network 基于 WiFi（WLAN 模块）的构造函数和使用方法。

构造函数
wlan = network.WINC()
构建 WIFI 模块连接对象。WINC 表示模块型号。
使用方法
wlan.scan ()
扫描允许访问的 SSID。
wlan.isconnected()
检查设备是否已经连接上。返回 Ture:已连接；False：未连接。
wlan.connected(ssid,passwork,security)
WIFI 连接。 【ssid】账号； 【passwork】密码； 【security】加密方式。
wlan.ifconfig([ip,subnet,gateway,dns])
设备信息配置。ip：IP 地址；subnet:子网掩码；gateway:网关地址；dns:DNS 信息。**（如果参数为空，则返回当前连接信息。）**
wlan.disconnected()
断开连接。
*更多参数说明请阅读 OpenMV 官方文档： 文档链接：http://docs.openmv.io/library/network.WINC.html

表 26-6　WINC 对象

从上表可以看到 MicroPython 通过模块封装，让 WIFI 联网变得非常简单。模块包含热点 AP 模块和客户端 STA 模式，热点 AP 是指电脑端直接连接 OpenMV 发出的热点实现连接，但这样你的电脑就不能上网了，因此我们一般情况下都是

使用 STA 模式。也就是电脑和设备同时连接到相同网段的路由器上。

代码编写流程如下：

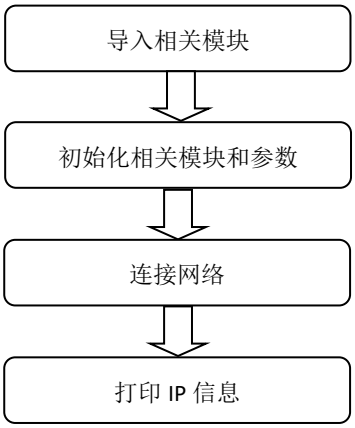


图 26-20 代码编写流程图

实验参考代码如下：

```
# WiFi 无线连接实验
#
# OpenMV 通过 WiFi 拓展模块连接无线网络
#
# 翻译和注释：01Studio

import network

SSID='01Studio' # WiFi 账号
KEY='88888888'  # WiFi 密码

# 打印提示
print("Trying to connect... (may take a while)...")

#构建 WiFi 对象
wlan = network.WINC()
```

```
#连接网络
wlan.connect(SSID, key=KEY, security=wlan.WPA_PSK)

#打印 IP 相关信息
print(wlan.ifconfig())
```

上面代码在 main.py 文件中，我们也可以新建一个 WIFI.py 文件，然后通过模块引用方式来供 main.py 调用，以提高程序的灵活性。

- **实验结果：**

运行程序，可以观察到连接成功后串口终端打印 IP 等信息。

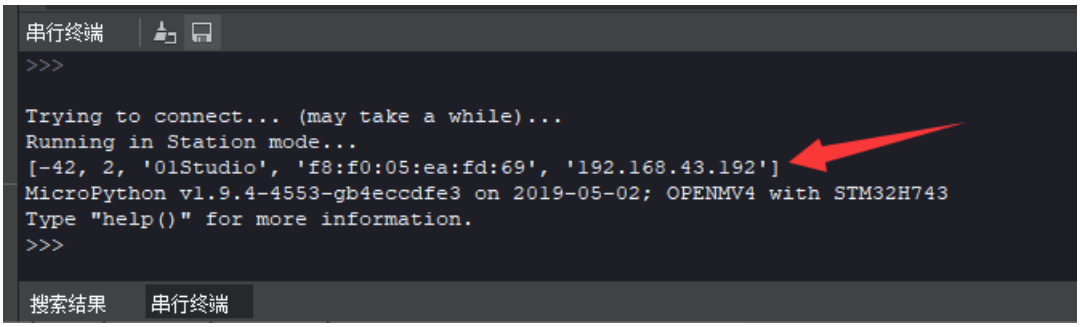


图 26-21

- **总结：**

本节是 WIFI 应用的基础，成功连接到无线路由器的实验后，后面就可以做 socket 和 MQTT 等相关网络通信的应用了。

26.4.2 Socket 通信

- **前言：**

上一节我们学习了如何通过 MicroPython 编程实现 pyAI-OpenMV4 结合 WiFi 拓展模块连接到无线路由器。这一节我们则来学习一下 Socket 通信实验。Socket 几乎是整个互联网通信的基础。

- **实验平台：**

pyAI-OpenMV4 和 OpenMV WiFi 模块。（如果使用 pyBase 需要将 OLED 拔掉。避免 IO 冲突）

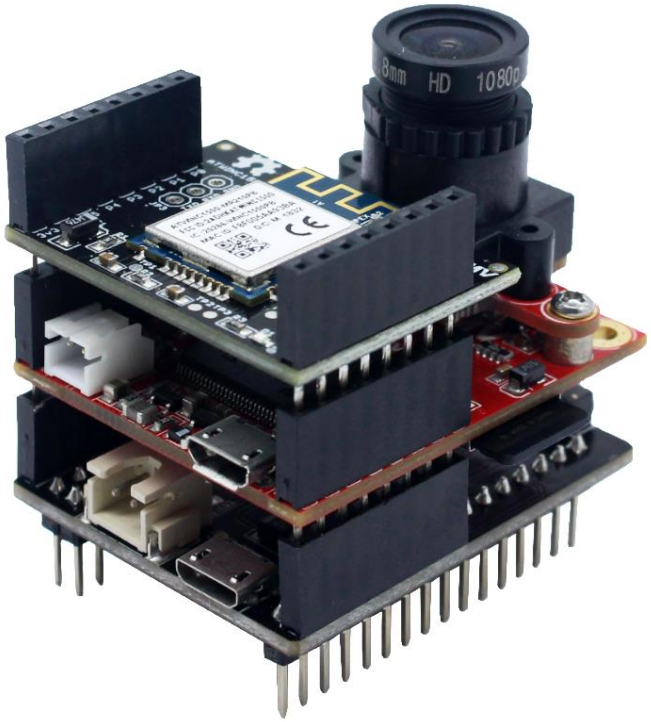


图 26-22 pyAI-OpenMV4 开发套件

- **实验目的：**

通过 Socket 编程实现 pyAI-OpenMV4 与电脑服务器助手建立连接，相互收发数据。

- **实验讲解：**

Socket 我们听得非常多了，但由于网络工程是一门系统工程，涉及的知识非

常广，概念也很多，任何一个知识点都能找出一堆厚厚的的书，因此我们经常会混淆。在这里，我们尝试以最容易理解的方式来讲述 Socket，如果需要全面了解，可以自行查阅相关资料学习。

我们先来看看网络层级模型图，这是构成网络通信的基础：

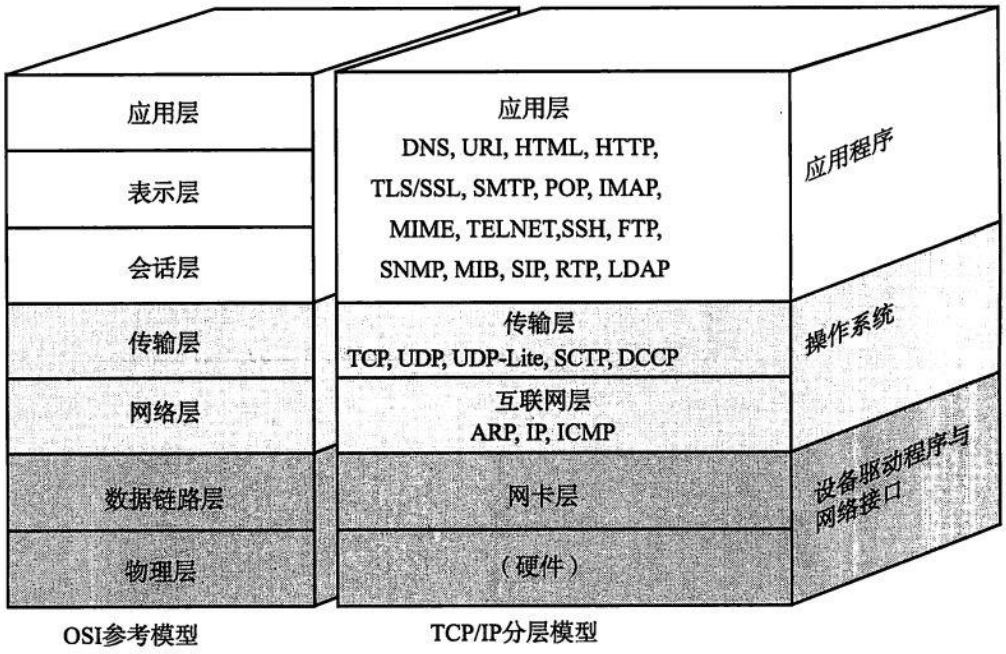


图 26-23 网络层级模型

我们看看 TCP/IP 模型的传输层和应用层，传输层比较熟悉的概念是 TCP 和 UDP，UPD 协议基本就没有对 IP 层的数据进行任何的处理了。而 TCP 协议还加入了更加复杂的传输控制，比如滑动的数据发送窗口（Slice Window），以及接收确认和重发机制，以达到数据的可靠传送。应用层中网页常用的则是 HTTP。那么我们先来解析一下这 TCP 和 HTTP 两者的关系。

我们知道网络通信是最基础是依赖于 IP 和端口的，HTTP 一般情况下默认使用端口 80。举个简单的例子：我们逛淘宝，浏览器会向淘宝网的网址（本质是 IP）和端口发起请求，而淘宝网收到请求后响应，向我们手机返回相关网页数据信息，实现了网页交互的过程。而这里就会引出一个多人连接的问题，很多人访问淘宝网，实际上接收到网页信息后就断开连接，否则淘宝网的服务器是无法支撑这么多人长时间的连接的，哪怕能支持，也非常占资源。

也就是应用层的 HTTP 通过传输层进行数据通信时，TCP 会遇到同时为多个应

用程序进程提供并发服务的问题。多个 TCP 连接或多个应用程序进程可能需要通过同一个 TCP 协议端口传输数据。为了区别不同的应用程序进程和连接，许多计算机操作系统为应用程序与 TCP / IP 协议交互提供了套接字(Socket)接口。应用层可以和传输层通过 Socket 接口，区分来自不同应用程序进程或网络连接的通信，实现数据传输的并发服务。

简单来说，Socket 抽象层介于传输层和应用层之间，跟 TCP/IP 并没有必然的联系。Socket 编程接口在设计的时候，就希望也能适应其他的网络协议。

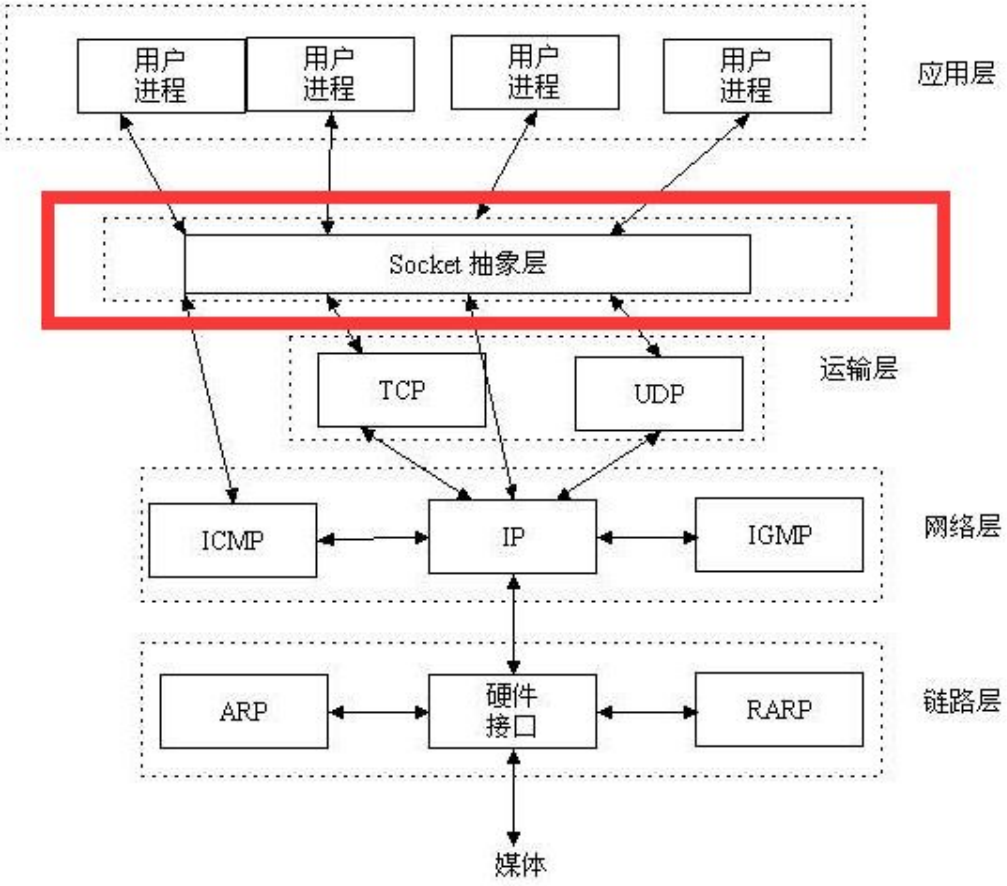


图 26-24　Socket 抽象层

套接字（socket）是通信的基石，是支持 TCP/IP 协议的网络通信的基本操作单元。它是网络通信过程中端点的抽象表示，包含进行网络通信必须的五种信息：**连接使用的协议（通常是 TCP 或 UDP），本地主机的 IP 地址，本地进程的协议端口，远地主机的 IP 地址，远地进程的协议端口。**

所以，socket 的出现只是可以更方便的使用 TCP/IP 协议栈而已，简单理解就是其对 TCP/IP 进行了抽象，形成了几个最基本的函数接口。比如 create，listen，

accept，connect，read 和 write 等等。以下是通讯流程：

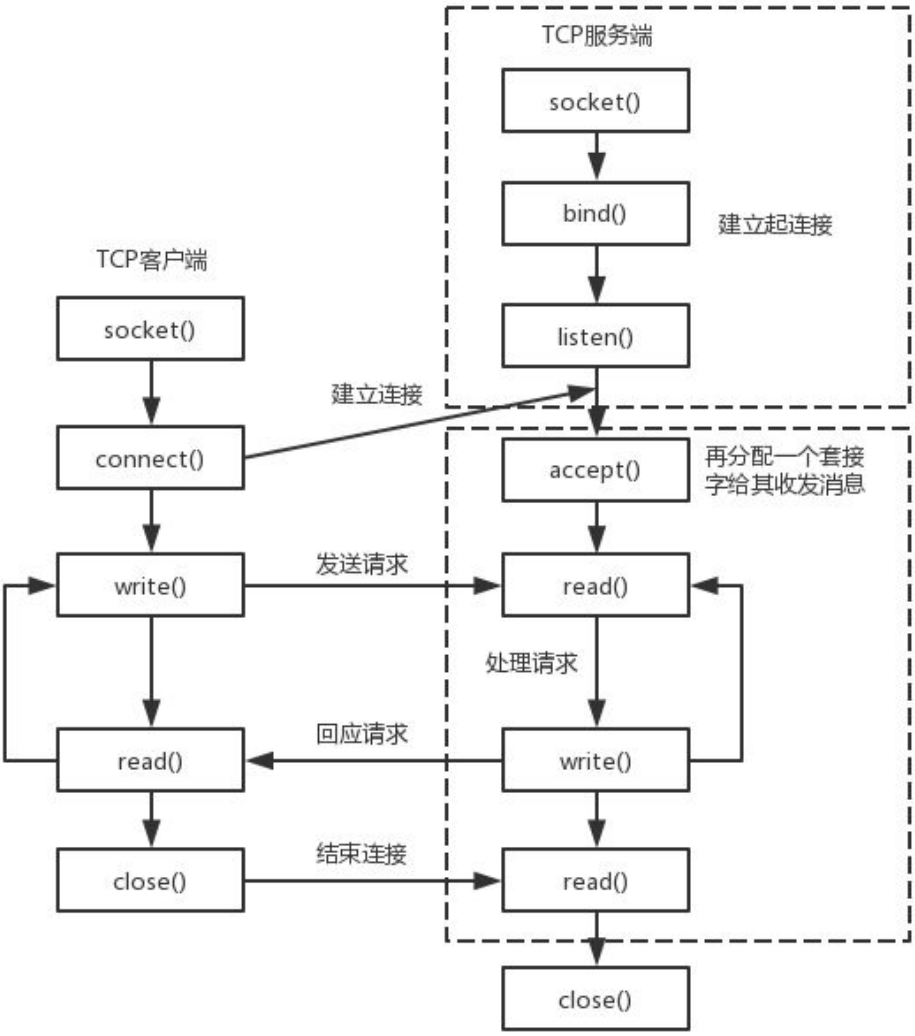


图 26-25　Socket 通信过程

从上图可以看到，建了 Socket 通信需要一个服务器端和一个客户端，以本实验为例，pyAI-OpenMV4 作为客户端，电脑使用网络调试助手作为服务器端，双方使用 TCP 协议传输。对于客户端，则需要知道电脑端的 IP 和端口即可建立连接。（端口可以自定义，范围在 0~65535，注意不占用常用的 80 等端口即可。）

以上的内容，简单来说就是如果用户面向应用来说，那么 OpenMV 只需要知道**通讯协议是 TCP 或 UDP、服务器的 IP 和端口号**这 3 个信息，即可向服务器发

起连接和发送信息。就这么简单。

MicroPython 已经封装好相关模块 usocket,跟传统的 socket 大部分兼容，两者均可使用，本实验使用 usocket，对象如下介绍：

构造函数
s=usocket.socekt(af=AF_INET, type=SOCK_STREAM,proto=IPPROTO_TCP)
构建 usocket 对象。 af: AF_INET→IPV4，AF_INET6 → IPV6； type: SCOK_STREAM→TCP，SOCK_DGRAM→UDP； proto: IPPROTO_TCP→TCP 协议，IPPROTO_UDP→UDP 协议。 （如果要构建 TCP 连接，可以使用默认参数配置，即不输入任何参数。）
使用方法
addr=usocket.getaddrinfo('www.01studio.org', 80)[0][-1]
获取 Socket 通信格式地址。返回：('47.91.208.161',80)
s.connect(address)
创建连接。address:地址格式为 IP+端口。例：('192.168.1.115',10000)
s.send(bytes)
发送。bytes：发送内容格式为字节
s.recv(bufsize)
接收数据。bufsize：单次最大接收字节个数。
s.bind(address)
绑定，用于服务器角色
s.listen([backlog])
监听，用于服务器角色。backlog:允许连接个数，必须大于 0。
s.accept()
接受连接，用于服务器角色。
*其它更多用法请阅读 MicroPython 文档：(搜索:usocket) 文档链接：http://docs.openmv.io/library/usocket.html

表 26-7 Socket 对象

本实验中 pyAI-OpenMV4 属于客户端，因此只用到客户端的函数即可。实验代码编写流程如下：

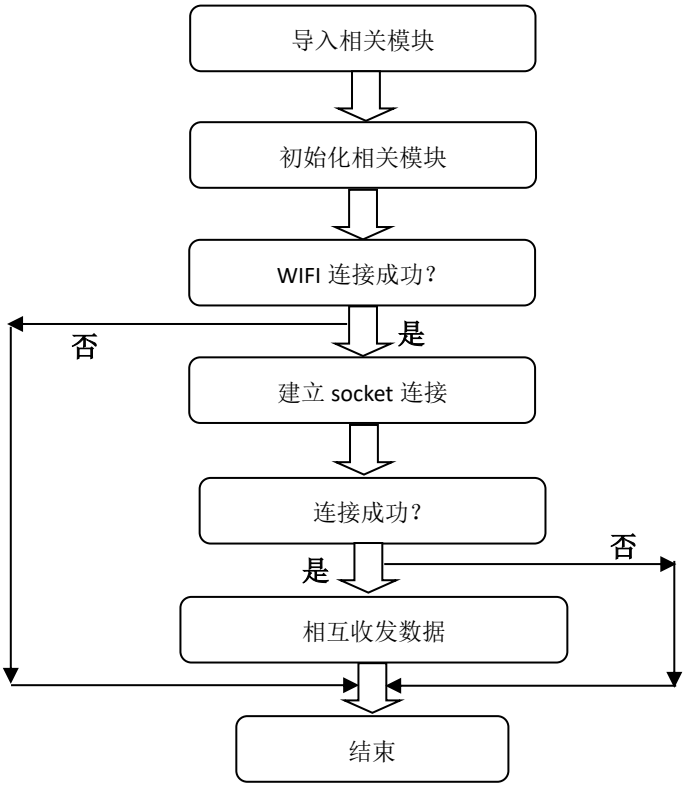


图 26-26 代码编写流程图

实验参考代码：

```
# TCP 客户端（Socket 通信） 实验
#
# 通过 WiFi 模块编程实现 OpenMV 的 Socket 通信，数据收发.
#
#作者：01Studio


import network, usocket,pyb


# WiFi 信息
SSID='WeBee_office_2.4G' # Network SSID
KEY='webee0123456789'  # Network key
```

```
#socket 数据接收中断标志位
socket_node = 0

# Init wlan module and connect to network
print("Trying to connect... (may take a while)...")

wlan = network.WINC()
wlan.connect(SSID, key=KEY, security=wlan.WPA_PSK)

# We should have a valid IP now via DHCP
print(wlan.ifconfig())

#创建 socket 连接，连接成功后发送“Hello 01Studio！”给服务器。
client=usocket.socket()
addr=('192.168.1.117',10000) #服务器 IP 和端口
client.connect(addr)
client.send('Hello 01Studio!')

#开启定时器，周期 100ms,重复执行 socket 通信接收任务
def fun(tim):
    global socket_node
    socket_node = 1
    pyb.LED(3).toggle()

tim = pyb.Timer(4,freq=10)
tim.callback(fun)



while True:
```

```
if socket_node:
    text=client.recv(128) #单次最多接收 128 字节
    if text == '':
        pass

    else: #打印接收到的信息为字节，可以通过 decode('utf-8')转成字符串
        print(text)
        client.send('I got:'+text.decode('utf-8'))

    socket_node=0
```

WIFI 连接代码在上一节已经讲解，这里不再重复，程序在连接成功后建了 Socket 连接，连接成功发送‘Hello 01Studio!’信息到服务器。另外 OpenMV 定时器设定了了每 100ms 处理从服务器接收到的数据。将接收到数据通过串口打印和重新发送给服务器。

- **实验结果：**

先在电脑端打开网络调试助手并建立服务器，软件在 零一科技（01Studio）MicroPython 开发套件配套资料_latest\01-开发工具\01-Windows\网络调试助手下的 NetAssist.exe ，直接双击打开即可！

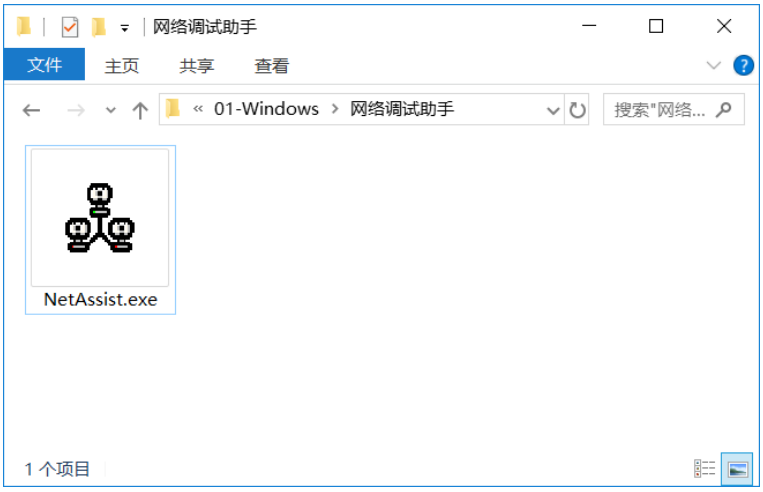


图 26-27 网络调试助手

以下是新建服务器的方法，打开网络调试助手后在左上角协议类型选择 TCP Server；中间的本地 IP 地址是自动识别的，不要修改，这个就是服务器的 IP 地址。然后端口写 10000（0-65535 都可以。），点击连接，成功后红点亮。如下图：



图 26-28 TCP 服务器配置

在时候服务器已经在监听状态！用户需要根据自己的实际情况自己输入 WIFI 信息和服务器 IP 地址+端口。即修改上面的代码以下部分内容。（服务器 IP 和端口可以在网络调试助手找到。）

```
# WiFi 信息

SSID='01Studio' # Network SSID

KEY='88888888'  # Network key
```

```
addr=('192.168.1.115',10000) #服务器 IP 和端口
```

下载程序，开发板成功连接 WIFI 后，发起了 socket 连接，连接成功可以可以看到网络调试助手收到了开发板发来的信息。在下方列表多了一个连接对象，点击选中：

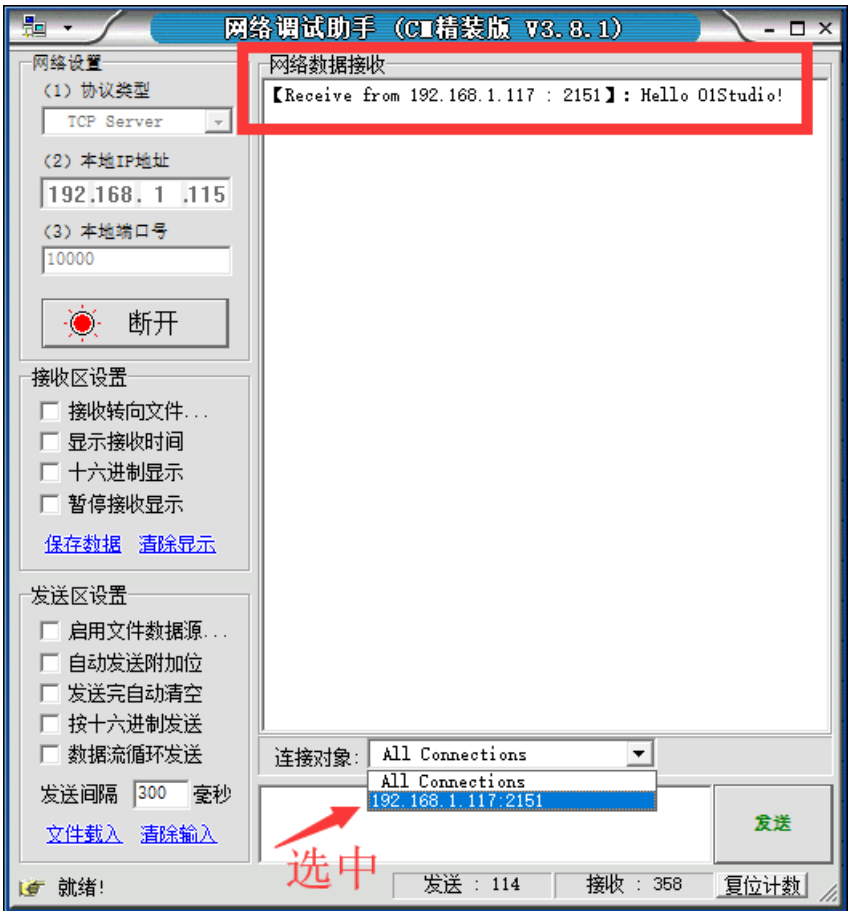


图 26-29 选中连接对象

选中后我们在发送框输入信息“Hi”，点击发送，可以看到开发板的 REPL 打印出来信息 Hi。为字节数据。另外由于程序将收到的信息发回给服务器，所以在网络调试助手中也接收到开发板返回的信息：I got:Hi。

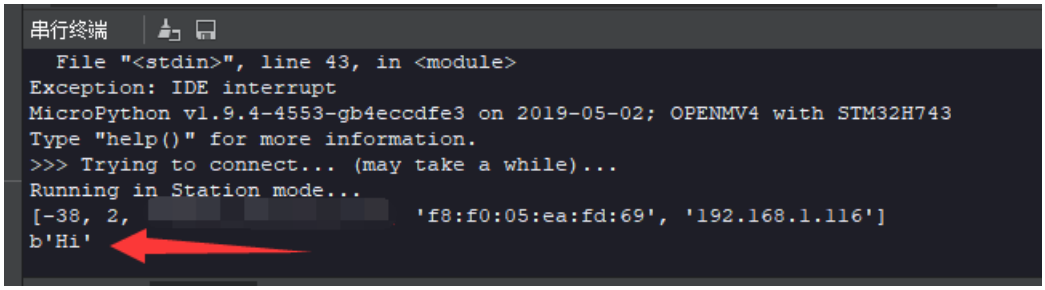


图 26-30

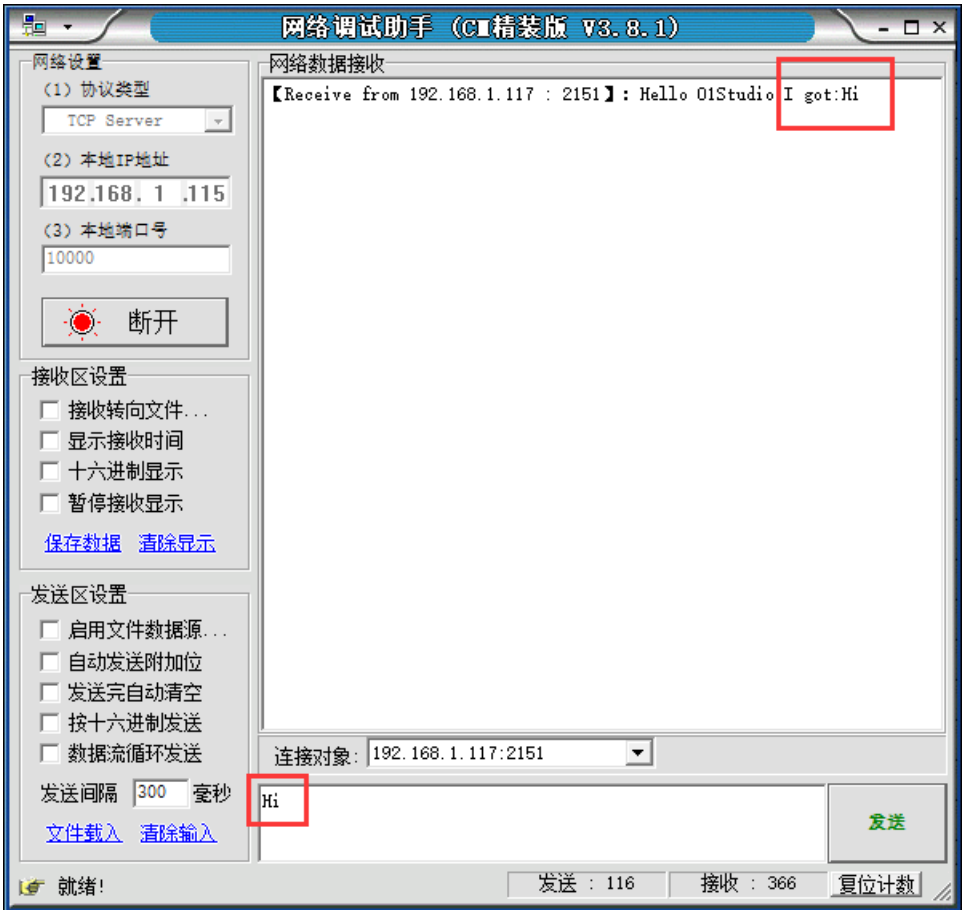


图 26-31 服务器发送数据

- **总结：**

通过本节学习，我们了解了 socket 通信原理以及使用 MicroPython 进行 socket 编程并且通信的实验。得益于优秀的封装，让我们可以直接面向 socket 对象编程就可以快速实现 socket 通信，从而开发更多的网络应用，例如将前面采集到的传感器数据发送到服务器。

26.4.3 MQTT 通信

- **前言：**

上一节，我们学习了 Socket 通信，当服务器和客户端建立起连接时，就可以相互通信了。在互联网应用大多使用 WebSocket 接口来传输数据。而在物联网应用中，常常出现这样的情况：海量的传感器，需要时刻保持在线，传输数据量非常低，有着大量用户使用。如果仍然使用 socket 作为通信，那么服务器的压力和通讯框架的设计随着数量的上升将变得异常复杂！

那么有无一个框架协议来解决这个问题呢，答案是有的。那就是 MQTT(消息队列遥测传输)。

- **实验平台：**

pyAI-OpenMV4 和 OpenMV WiFi 模块。（如果使用 pyBase 需要将 OLED 拔掉。避免 IO 冲突）

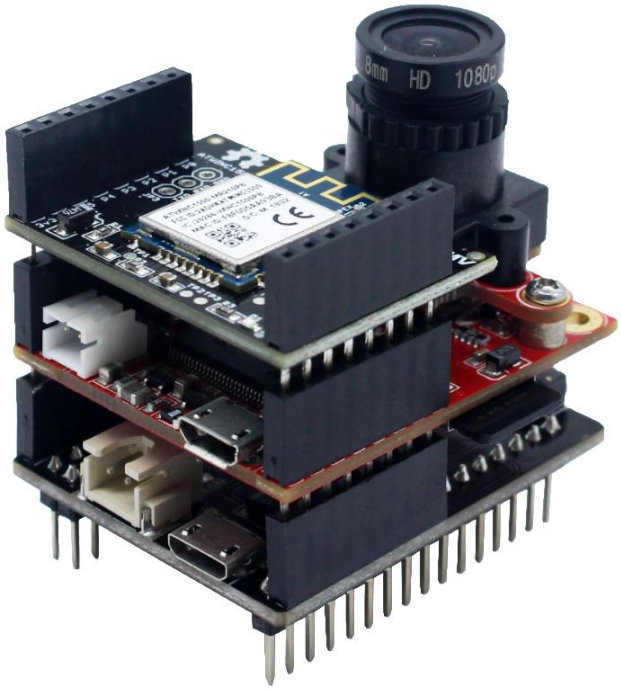


图 26-32 pyAI-OpenMV4

- **实验目的：**

通过编程让 pyAI-OpenMV4 实现 MQTT 协议信息的发布和订阅（接收）。

- **实验讲解：**

MQTT 是 IBM 于 1999 年提出的，和 HTTP 一样属于应用层，它工作在 TCP/IP 协议族上，通常还会调用 socket 接口。是一个基于客户端-服务器的消息发布/订阅传输协议。其特点是协议是轻量、简单、开放和易于实现的，这些特点使它适用范围非常广泛。在很多情况下，包括受限的环境中，如：机器与机器（M2M）通信和物联网（IoT）。其在，通过卫星链路通信传感器、偶尔拨号的医疗设备、智能家居、及一些小型化设备中已广泛使用。

总结下来 MQTT 有如下特性/优势：

- **异步消息协议**
- **面向长连接**
- **双向数据传输**
- **协议轻量级**
- **被动数据获取**

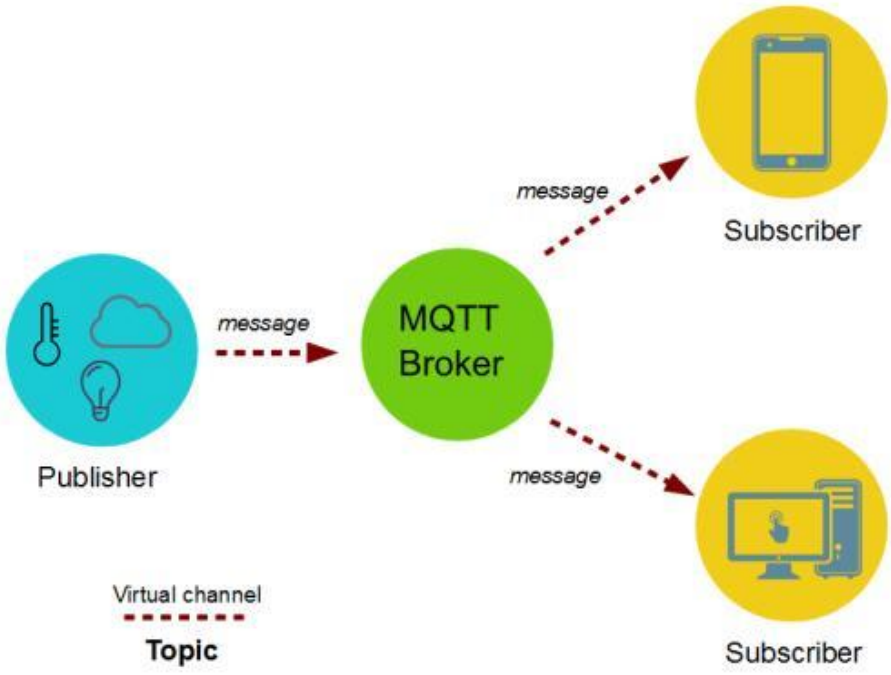


图 26-33 MQTT 通信流程

从上图可以看到，MQTT 通信的角色有两个，分别是服务器和客户端。服务器只负责中转数据，不做存储；客户端可以是信息发送者或订阅者，也可以同时是两者。具体如下图：

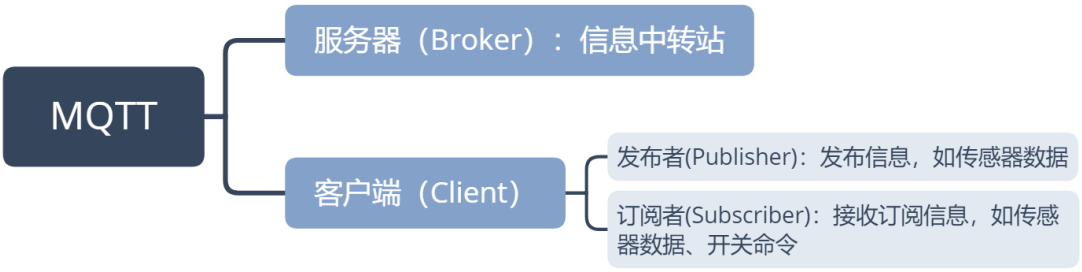


图 26-34 MQTT 角色说明

确定了角色后是如何传输数据呢？下表示 MQTT 最基本的数据帧格式，例如温度传感器发布主题“Temperature”编号,消息是“25”（表示温度）。那么所有订阅了这个主题编号的客户端（手机应用）就会收到相关信息，从而实现通信。如下表所示：

MQTT 数据帧格式	
Topic ID (主题编号)	Message (消息)
Temperature	25

图 26-35 MQTT 数据格式

由于特殊的发布/订阅机制，服务器不需要存储数据（当然也可以在服务器的设备上建立一个客户端来订阅保存信息），因此非常适合海量设备的传输。

人生苦短，而 MicroPython 已经封装好了 MQTT 客户端的库文件。让我们的应用变得简单美妙。OpenMV 的 MQTT 模块文件为例程文件夹里面的 mqtt.py 文件。使用方法如下：

构造函数
client=mqtt. MQTTClient (client_id, server, port)
构建 MQTT 客户端对象。 client_id: 客户端 ID，具有唯一性； server: 服务器地址，可以是 IP 或者网址； port:服务器端口。（默认是 1883，服务器通常采用的端口，也可以自定义。）
使用方法
`client.connect()`

连接到服务器。
client.publish(TOPIC,message)
发布。TOPIC：主题编号；message：信息内容，例：'Hello 01Studio!'
client.subscribe(TOPIC)
订阅。TOPIC：主题编号。
client.set_callback(callback)
设置回调函数。callback：订阅后如果接收到信息，就执行相名称的回调函数。
client.check_msg()
检查订阅信息。如收到信息就执行设置过的回调函数 callback。

表 26-8　MQTT 客户端对象

由于客户端分为发布者和订阅者角色，因此为了方便大家更好理解，本实验分开两个案例来编程，分别为发布者和订阅者。再结合 MQTT 网络调试助手来测试。代表编写流程图如下：

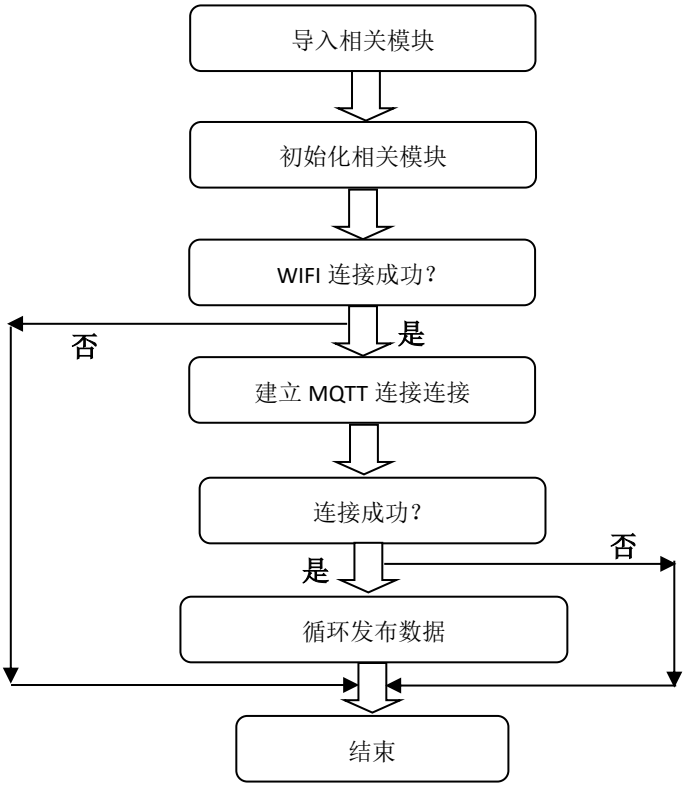


图 26-36 发布者代码编写流程



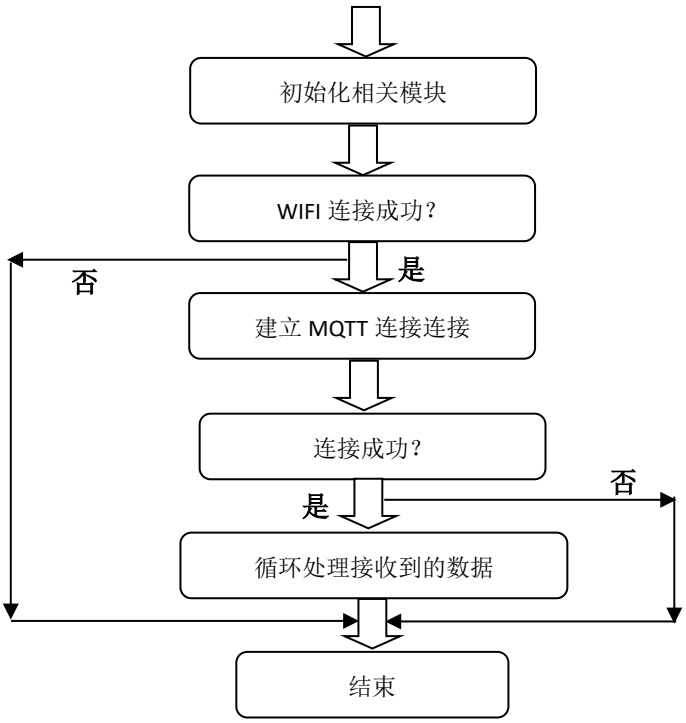


图 26-37 订阅者代码编写流程

发布者（publish）参考代码：

```
# MQTT 发布者（publish）例程.
#
#需要复制 mqtt.py 文件到 OpenMV 文件系统.
#
#翻译和注释：01Studio

import time, network
from mqtt import MQTTClient

SSID='WeBee_office_2.4G' # Network SSID
KEY='webee0123456789'  # Network key

# Init wlan module and connect to network
print("Trying to connect... (may take a while)...")
```

```
wlan = network.WINC()
wlan.connect(SSID, key=KEY, security=wlan.WPA_PSK)

# We should have a valid IP now via DHCP
print(wlan.ifconfig())

SERVER = 'mqtt.p2hp.com'
PORT = 1883
CLIENT_ID = '01Studio-OpenMV' # 客户端 ID
TOPIC = 'OpenMV/test' # TOPIC 名称

client = MQTTClient(CLIENT_ID, SERVER, PORT)
client.connect()

while (True):
    client.publish(TOPIC, "Hello 01Studio!") #发布消息
    time.sleep(1000)
```

订阅者（subscribe）参考代码：

```
# MQTT 订阅者（subscribe）例程.
#
#需要复制 mqtt.py 文件到 OpenMV 文件系统.
#
#翻译和注释：01Studio

import time, network
from mqtt import MQTTClient

SSID='WeBee_office_2.4G' # Network SSID
```

```
KEY='webee0123456789'  # Network key

# Init wlan module and connect to network
print("Trying to connect... (may take a while)...")

wlan = network.WINC()
wlan.connect(SSID, key=KEY, security=wlan.WPA_PSK)

# We should have a valid IP now via DHCP
print(wlan.ifconfig())

#设置 MQTT 回调函数,有信息时候执行
def MQTT_callback(topic, msg):
    print('topic: {}'.format(topic))
    print('msg: {}'.format(msg))

SERVER = 'mqtt.p2hp.com'
PORT = 1883
CLIENT_ID = '01Studio-OpenMV' # 客户端 ID
TOPIC = 'OpenMV/test' # TOPIC 名称



client = MQTTClient(CLIENT_ID, SERVER, PORT)
client.set_callback(MQTT_callback)  #配置回调函数
client.connect()
client.subscribe(TOPIC) #订阅主题

while (True):
    client.check_msg() #检测是否收到信息，收到则执行打印。
```

从以上代码可以看到发布者和订阅者的编程方式相近，另外本实验需要一个 MQTT 服务器（Broker），这里使用的是跟我们将用到的 MQTT 在线网络助手同一个服务器和端口。

```
SERVER = 'mqtt.p2hp.com'
PORT = 1883
```

- **实验结果：**

我们将 01Studio 示例程序中的 mqtt.py 文件拷贝到文件系统，然后在 OpenMV IDE 分别运行上述代码。

OpenMV IDE 可能有个 BUG，就是运行时候发现 mqtt.py 文件跟文件系统不一致，弹出提示。我们要选 No 忽略它。

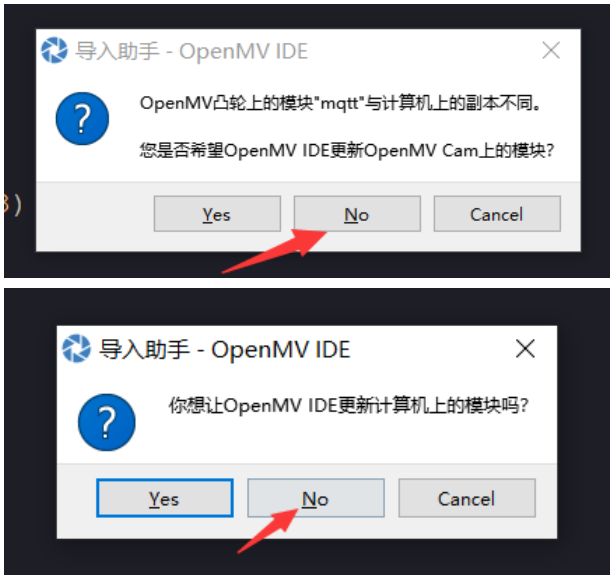


图 26-38 选择 No 忽略

为了方便测试，我们可以使用 MQTT 网络助手进行调试。这里推荐一个在线 MQTT 网络调试助手：http://mqtt.p2hp.com/websocket/

打开上面网址，即可看到 MQTT 在线调试助手。可以配置基本信息，这里默认即可，点击连接。

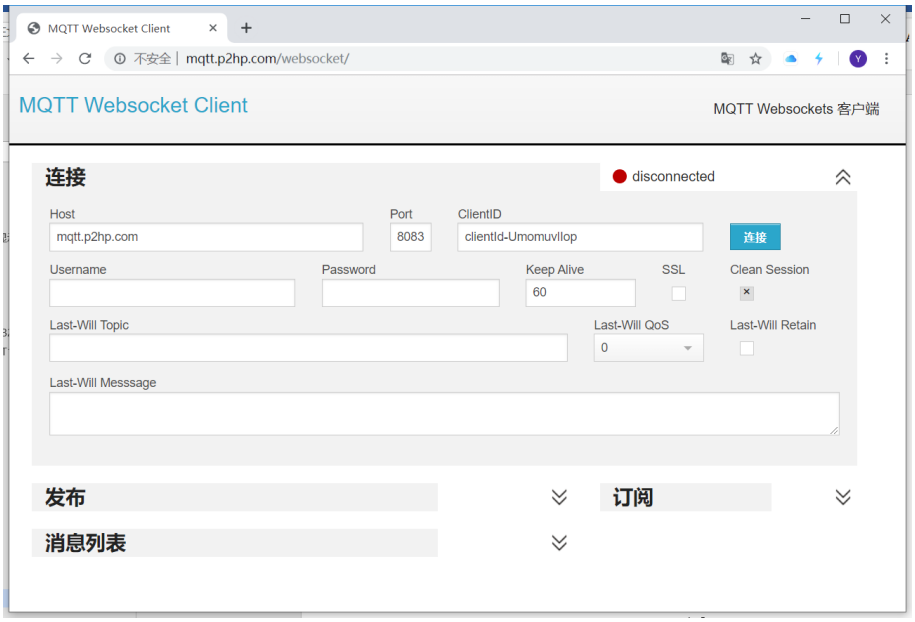


图 26-39 MQTT 助手

连接成功可以看到显示 Connected。

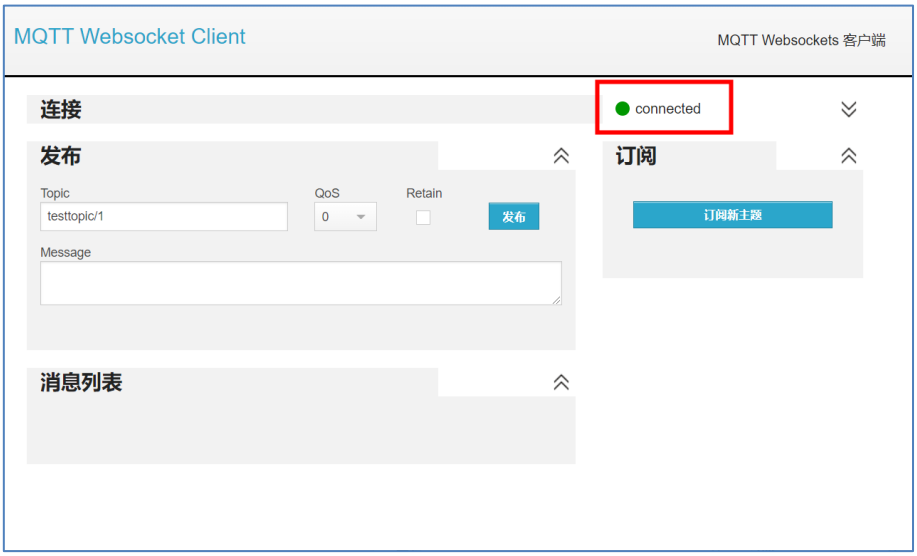


图 26-40 MQTT 助手连接成功

测试“发布者”代码：

我们先测试“发布者”代码。将“发布者”代码下载到开发板，然后在 MQTT 助手中订阅主题修改为：'/public/01Studio/1'（跟代码发布的主题一致），QOS 选择 0 即可。然后点击订阅主题。

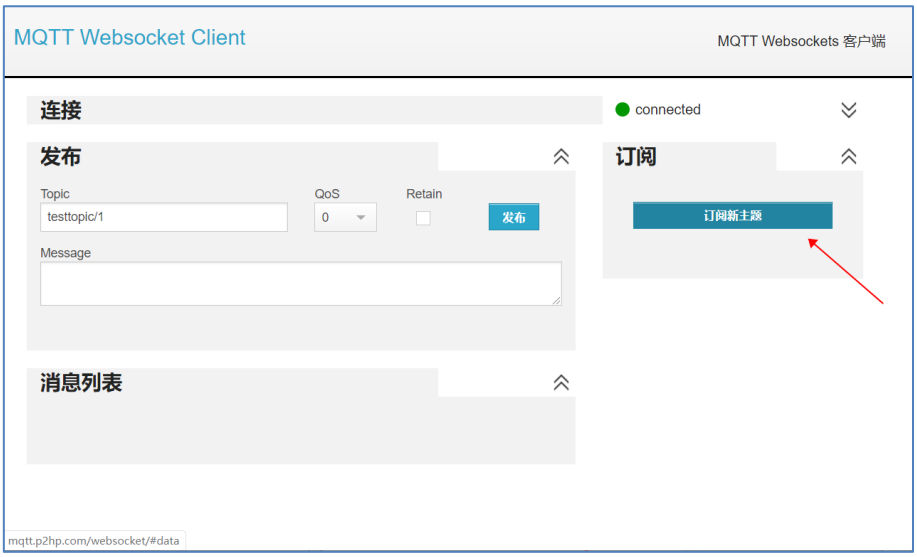


图 26-41 订阅主题

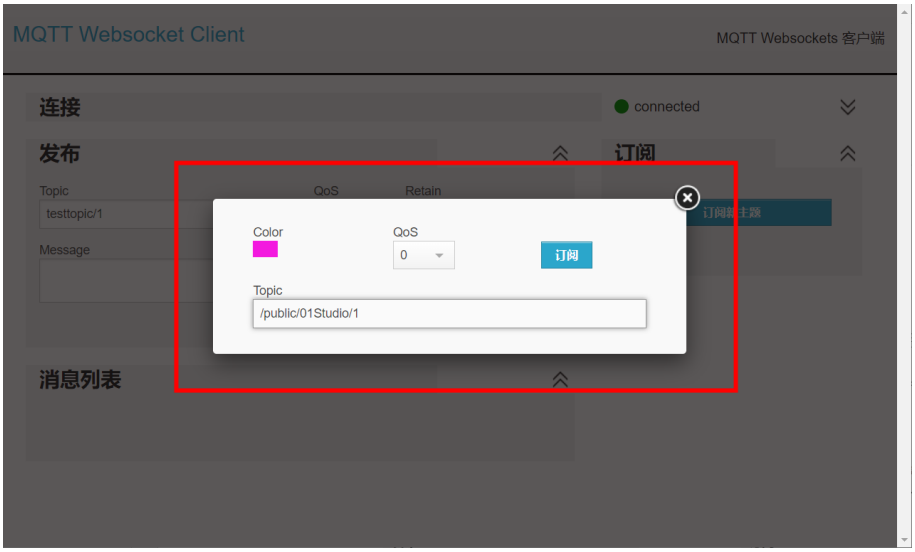


图 26-42

订阅后可以看到接收框收到来自开发板发布的信息。

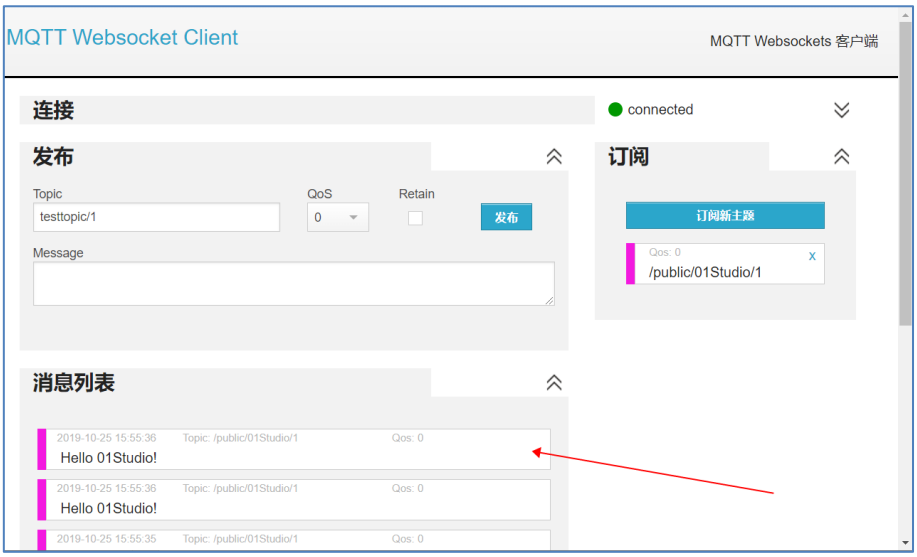


图 26-43 MQTT 助手收到开发板发布的信息

测试“订阅者”代码：

“订阅者”代码测试方法跟“发布者”相反。将“订阅者”代码下载到开发板，然后在电脑 MQTT 助手中发布主题修改为：'/public/01Studio/1'（跟代码发布的主题一致。）在下方空白框输入“Hello 01Studio!”。

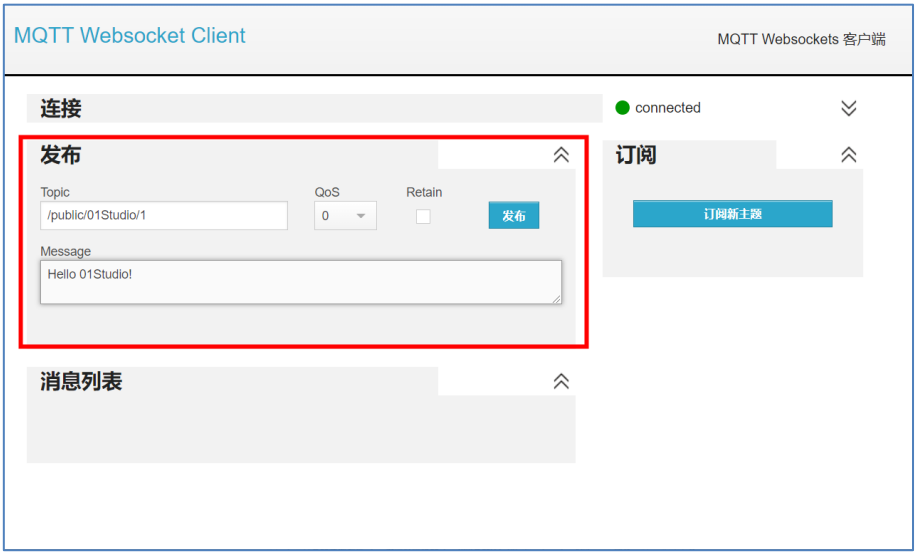


图 26-44 输入即将发布的主题和信息

将“订阅者”代码下载到开发板，成功连接后回到 MQTT 助手点击【发布】

发布信息，可以在开发板的 REPL 看到接收到的订阅信息“Hello 01Studio!”被打印出来了（数据格式为字节数据）。

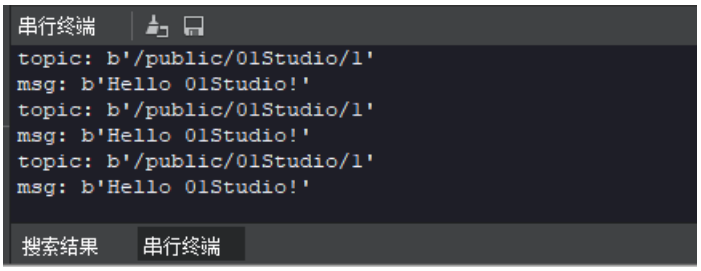


图 26-45 开发板接收到相关信息并打印

当使用 main.py 脱机运行时候，IDE 的串口终端和 putty 并不能打印信息。可以使用串口助手观察。波特率为 115200。

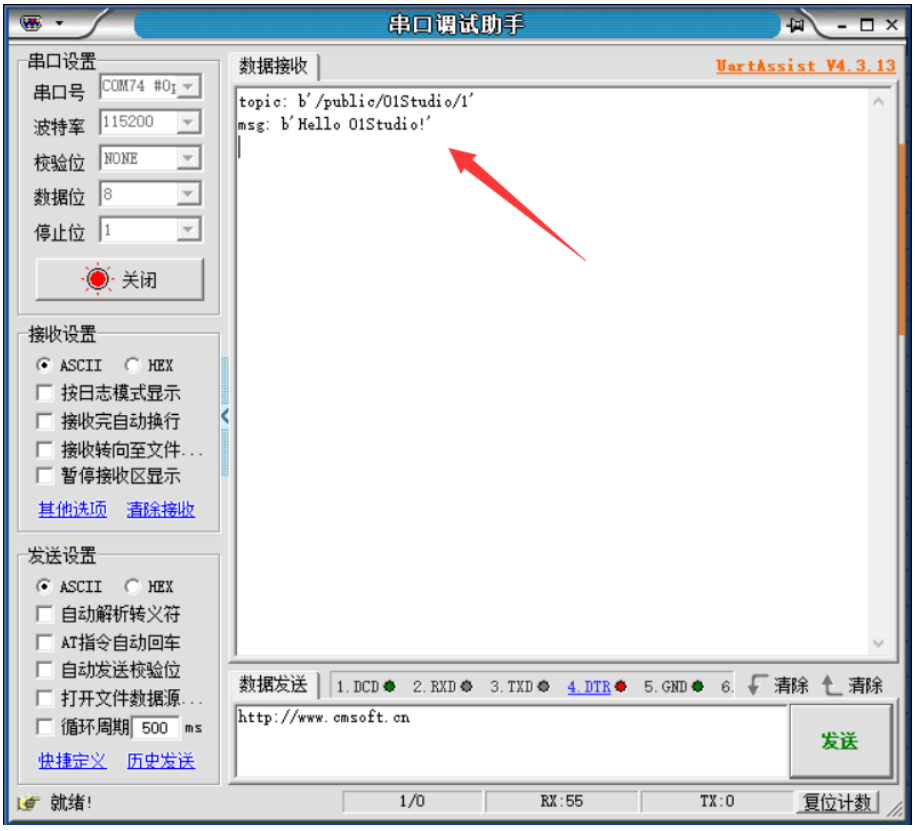


图 26-46 串口调试助手

当然你也可以在同一个 MQTT 在线助手下测试发布和订阅功能来做一些测试，只需要将主题设置一致即可，如下图所示：

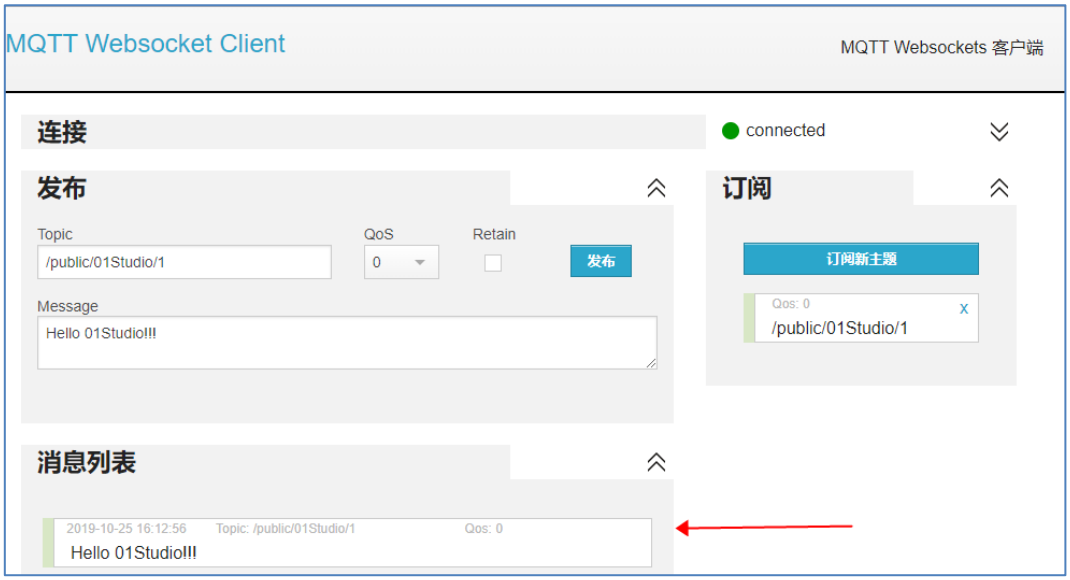


图 26-47 客户端同时发布和订阅信息

- **总结：**

通过本节我们了解了 MQTT 通信原理以及成功实现通信。目前市面上大部分物联网云平台支持 MQTT，原理大同小异。大家可以基于不同平台协议来开发，实现自己的物联网设备远程连接。

26.4.4 无线视频传输

- **前言：**

pyAI-OpenMV4 配合 WiFi 模块能实现浏览器视频传输，本节只讲述实验方法，关于原理和代码用户可以自行研究。

- **实验平台：**

pyAI-OpenMV4 和 OpenMV WiFi 模块。（如果使用 pyBase 需要将 OLED 拔掉。避免 IO 冲突）。另外最好配上锂电池，保证开发板供电的稳定性。

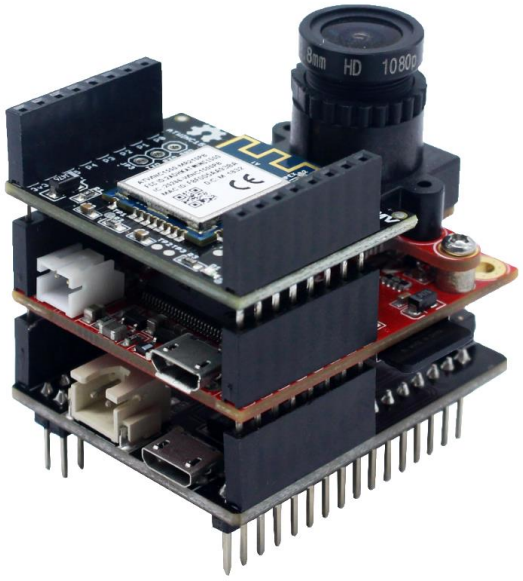


图 26-48 pyAI-OpenMV4

- **实验说明：**

我们打开官方例程【示例→WiFi-Shield→ mjpeg_streamer.py】，具体如下：

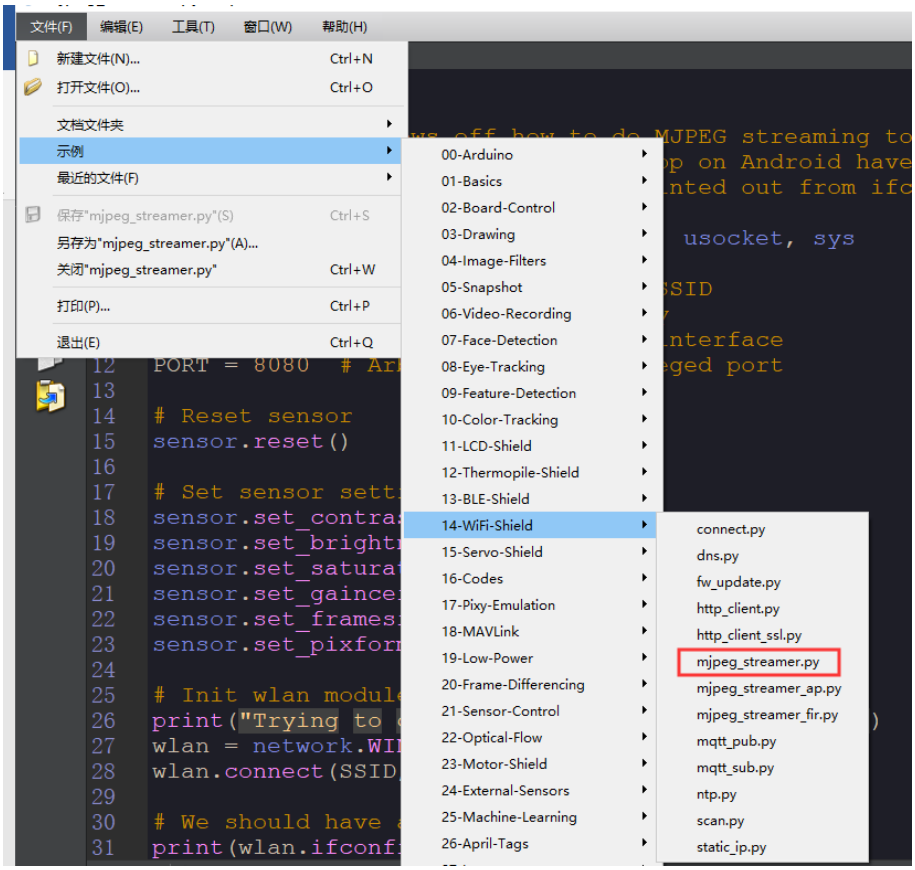


图 26-49 视频传输例程

我们只需要修改自己的无线网络,端口默认 8080 即可：

```
SSID='01Studio' # Network SSID

KEY='88888888'  # Network key

HOST =''     # Use first available interface

PORT = 8080  # Arbitrary non-privileged port
```

运行程序，可以看到串口中断打印出当前 OpenMV 设备的连接信息：

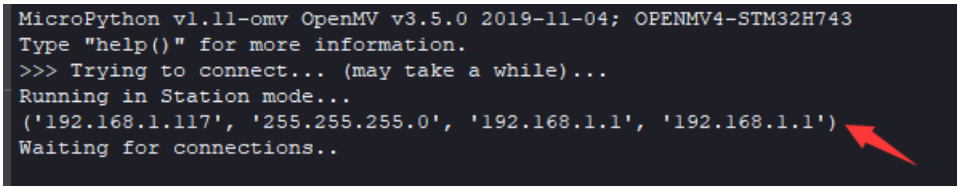


图 26-50 OpenMV4 WiFi 连接信息

可以看到 IP 是 192.168.1.117，我们在同一网络的电脑或者手机浏览器下打开 **192.1681.117:8080** 这个地址，可以看到连接成功后出现图像。（建议使用谷歌浏览器。）

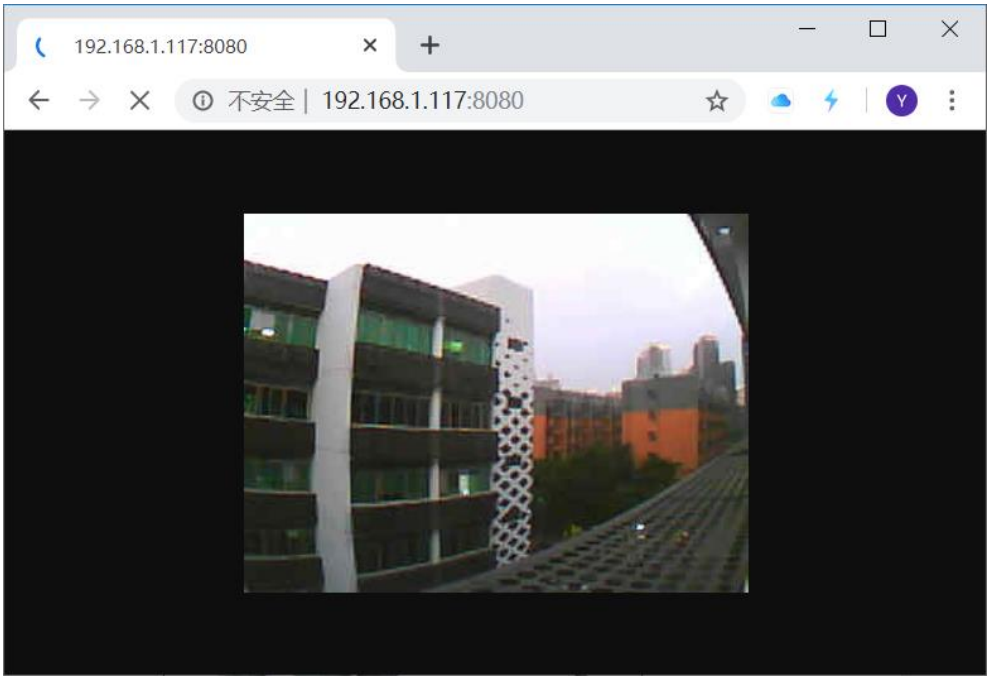


图 26-51 PC 浏览器

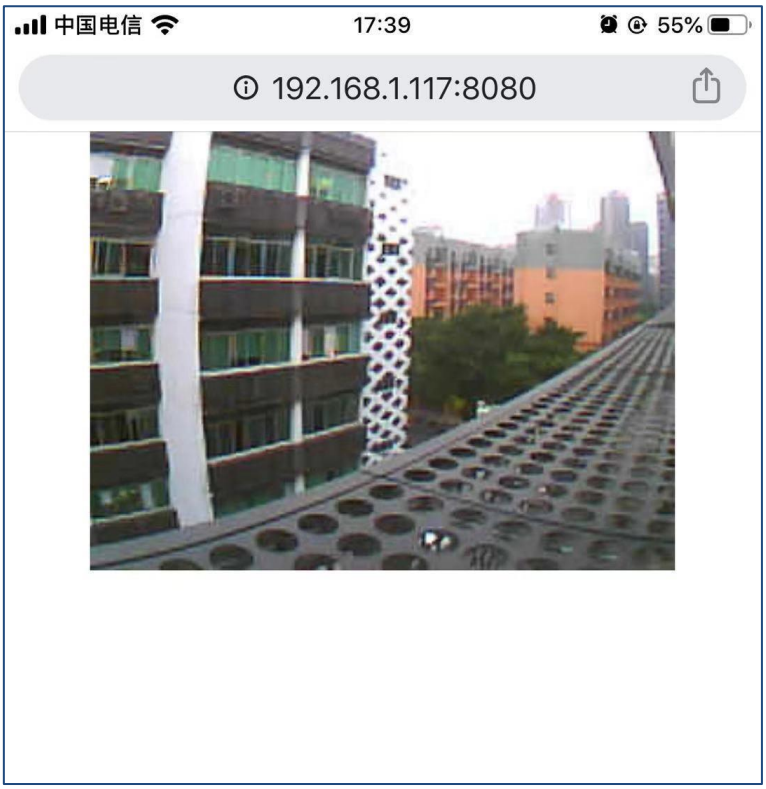


图 26-52 手机浏览器

26.5 摄像头模块

OpenMV 更换摄像头模块非常简单。可以将整个摄像头模块拆下更换。默认的摄像头模块是 OV7725 卷帘快门。

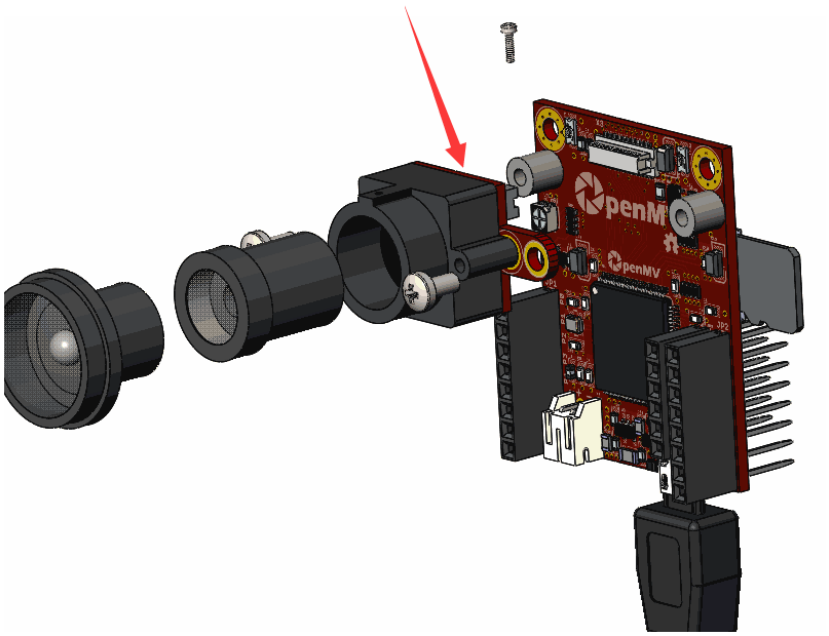


图 26-53

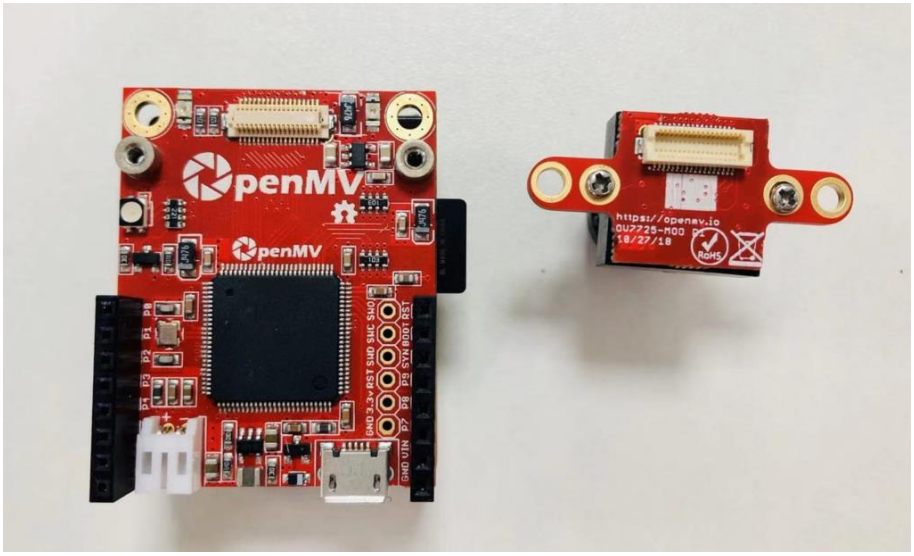


图 26-54 更换摄像头

26.5.1 MT9V034 全局快门

- **前言：**

MT9V034 全局快门适用于高速运动状态拍摄，支持工业高速运动的应用。每秒帧率相对于 OV7725 卷帘快门大大提升，仅支持灰度图像拍摄。

- **实验平台：**

pyAI-OpenMV4 开发板+MT9V034 全局快门模块。



图 26-55 pyAI-OpenMV4

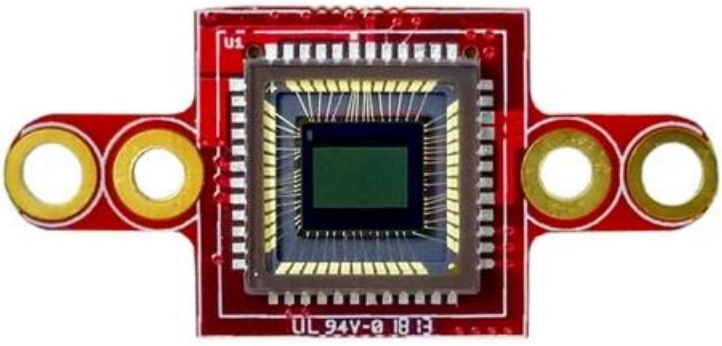


图 26-56 MT9V034 全局快门模组

- **实验目的：**

测试 MT9V034 全局快门在不同图像尺寸下的 FPS（每秒帧率）变化。

- **实验讲解：**

全局快门模块功能参数如下表：

功能参数	
型号	MT9V034
分辨率	752*480 像素
像素间距	6um
重量	10g
工作温度	-30℃至 70℃
使用场景	高速移动物体拍摄

表 26-9 功能参数

我们先来看看卷帘快门和全局快门的曝光方式和原理：

卷帘快门曝光方式：





图 26-57 卷帘快门曝光

全局快门曝光方式：



图 26-58 全局快门曝光

从上图可以知道，卷帘快门的曝光方式决定了它在拍摄高速运动物体会让图像出现变形，而全局快门很好解决了这个问题。以下是 MT 摄像头模组官方材料说明，分别是肉眼、卷帘快门和全局快门的拍摄效果。

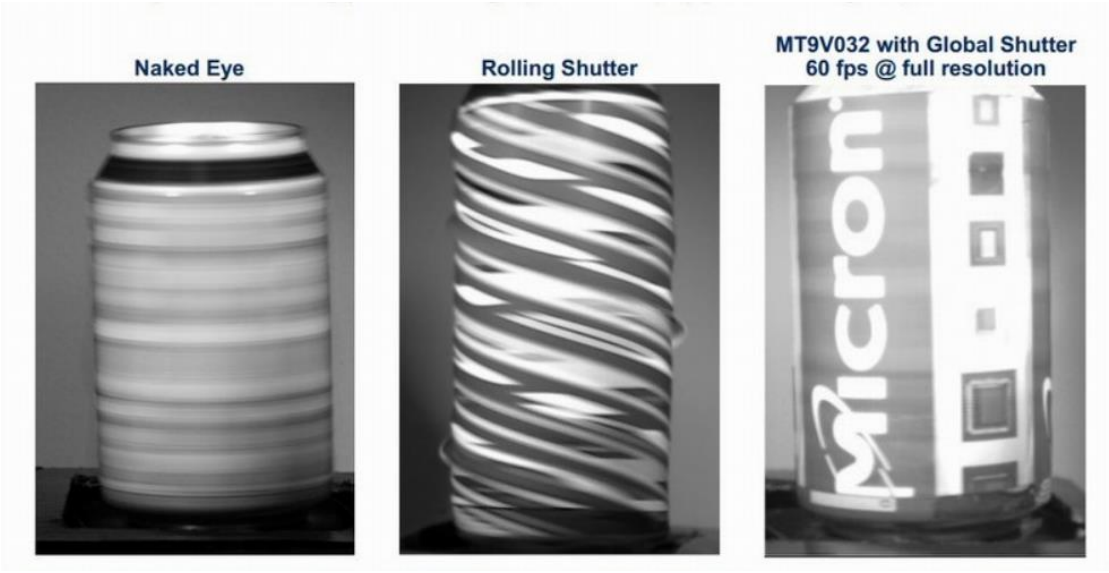


图 26-59　MT9V034 官方效果图（肉眼、卷帘快门、全局快门）

OpenMV 官方提供了全局快门高速 FPS（每秒帧率）的测试例程，参考代码路径【示例→Global Shutter→ high_fps.py】，具体如下：

```
# 高 FPS(每秒帧率)例程
#
#这个例程展示了全局快门的高帧率。你可以通过改变图像像素和曝光时间来调整帧率。
#
# 图像大小从 VGA 到 QVGA，FPS 提升 2 倍，从 QVGA 到 QQVGA，FPS 再提升 2 倍。
#
# 除此之前你也可以降低曝光时间来提供 FPS，但这样拍摄出来的图像会非常暗。因此你
# 需要额外补充光源。
#
#翻译和注释：01Studio

import sensor, image, time
```

```
#摄像头初始化
sensor.reset()                      # Reset and initialize the sensor.
sensor.set_pixformat(sensor.GRAYSCALE) # Set pixel format to GRAYSCALE
sensor.set_framesize(sensor.QQVGA)  # 可以测试 VGA/QVGA/QQVG 对比 FPS 效果
sensor.skip_frames(time = 2000)     # Wait for settings take effect.
clock = time.clock()                # Create a clock object to track the FPS.

sensor.set_auto_exposure(True, exposure_us=5000)
# 曝光时间，使用小的值提高速度。

while(True):
    clock.tick()                    # Update the FPS clock.
    img = sensor.snapshot()         # Take a picture and return the image.
    print(clock.fps())              # 使用 IDE 运行会降低 FPS。
```

- **实验结果：**

我们将以下代码的图像尺寸分别改成 VGA(640X480)、QVGA(320*240)和 QQVGA(160X120)。在串口中观察 FPS 变化。

```
sensor.set_framesize(sensor.QQVGA)  # 可以测试 VGA/QVGA/QQVG 对比 FPS 效果
```

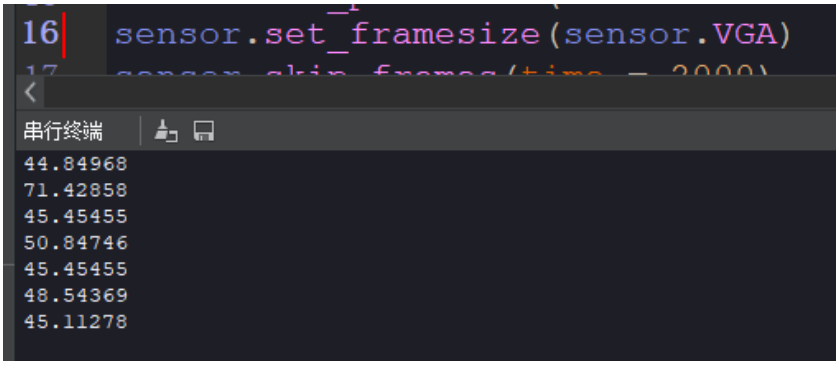


图 26-60 VGA 下的 FPS

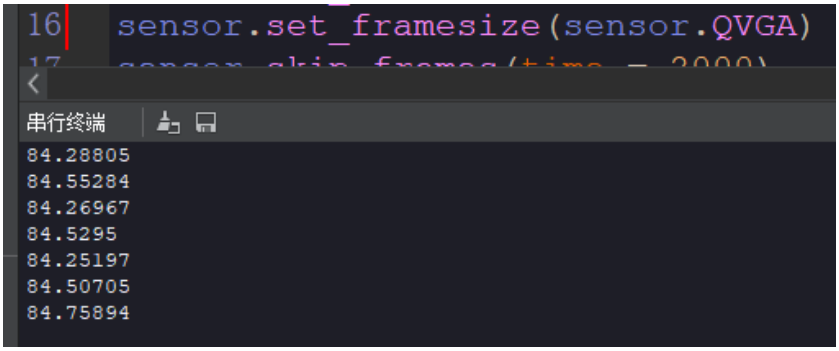


图 26-61　QVGA 下的 FPS

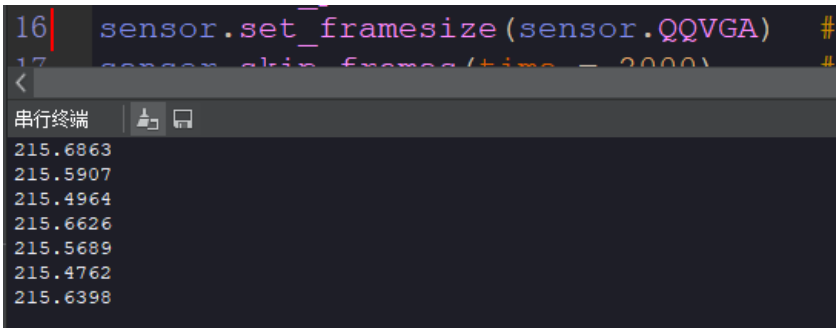


图 26-62　QQVGA 下的 FPS

- **实验总结：**

通过本实验，可以看到全局快门在不同图像尺寸下的 FPS（每秒帧率）变化，图像越小，FPS 越高。全局快门在用于曝光时间短的情景下，可以增加拍摄物的光线（补光）以获取更好的拍摄效果。

26.5.2 FLIR Lepton 红外热成像

- **前言：**

FLIR Lepton 红外热成像摄像头模块适用于识别人和动物、夜晚视觉、检测温度热量以及工业检测故障等应用，属于军工级产品。本节只讲述测试方法，具体代码内容用户可以自行深入研究。

- **实验平台：**

pyAI-OpenMV4 开发板+FLIR Lepton 3.5 红外热成像模块。

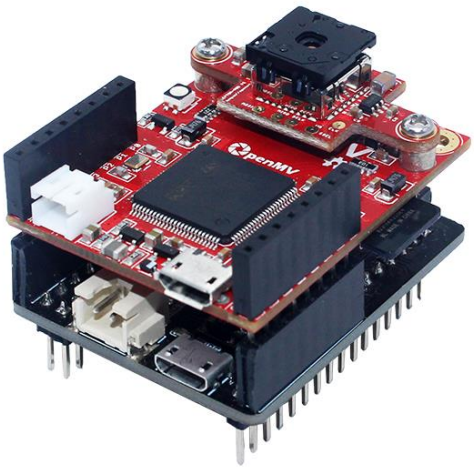


图 26-63 pyAI-OpenMV4

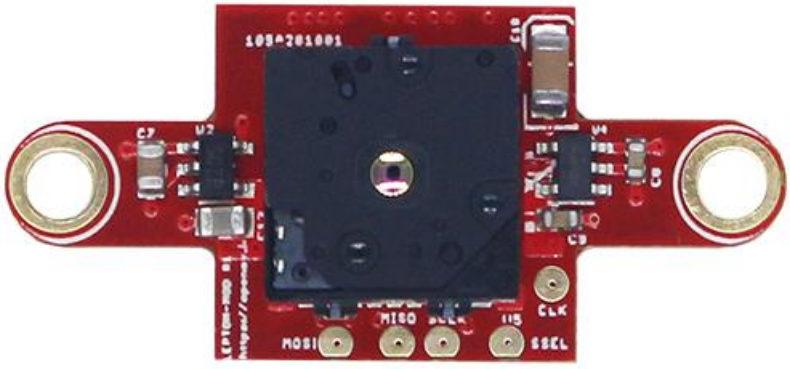


图 26-64 FLIR Lepton 3.5(160*120) 外热成像模组

- **实验目的：**

测试 FLIR Lepton 3.5(160*120) 外热成像模组。

- **实验讲解：**

热成像的工作原理是什么？

通俗地讲热像仪就是将物体发出的不可见红外能量转变为可见的热图像。热图像的上面的不同颜色代表被测物体的不同温度。通过查看热图像，可以观察到被测目标的整体温度分布状况，研究目标的发热情况，从而进行下一步工作的判断。 现代热像仪的工作原理是使用光电设备来检测和测量辐射，并在辐射与表面温度之间建立相互联系。所有高于绝对零度（-273℃）的物体都会发出红外辐射。热像仪利用红外探测器和光学成像物镜接受被测目标的红外辐射能量分布图形反映到红外探测器的光敏元件上，从而获得红外热像图，这种热像图与物体表面的热分布场相对应。

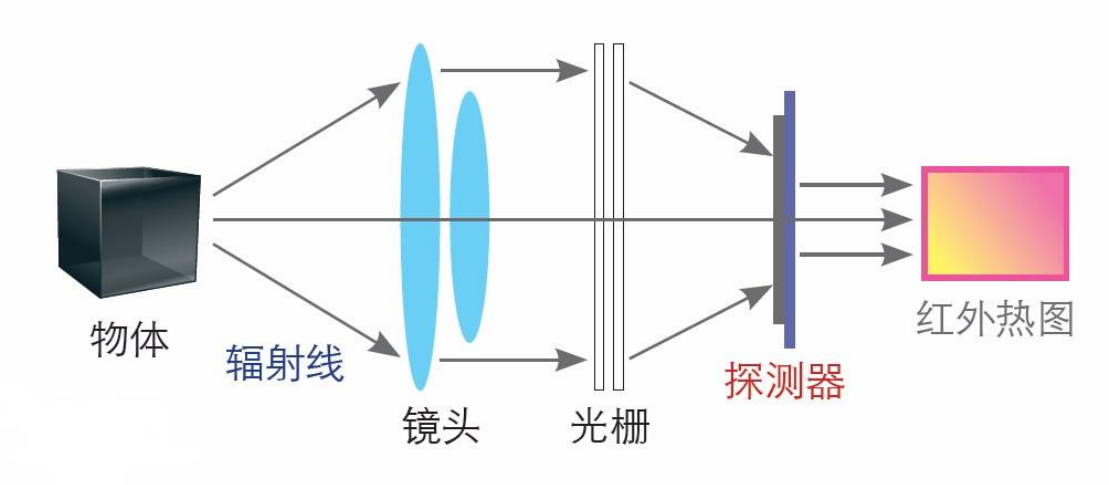


图 26-65 热成像工作原理

我们再来看看 Flir lepton 3.5 模块功能参数：

功能参数	
型号	FLIR Lepton 3.5
分辨率	160*120 像素
像素间距	12um
波长范围	8um – 14um
视场角	对角线 71°，水平 57°
其它功能	温度检测
适用场景	识别任何动物、夜晚视觉、检查温度热量及工业检测故障等。

表 26-10 功能参数

OpenMV 官方提供了非常全面的 Lepton 红外热成像模块测试例程，位于【示例—Lepton】下，带 lcd 的例程表示可以同时在 1.77 寸 LCD 液晶屏显示。

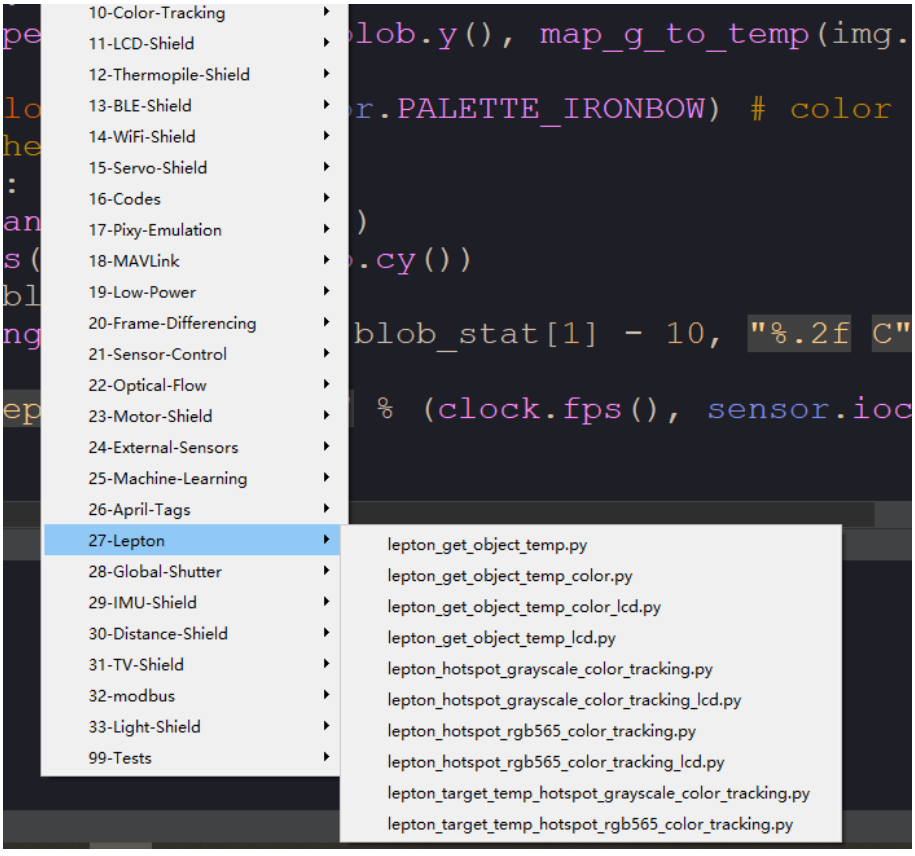


图 26-66 Lepton 测试例程

- **实验结果：**

Lepton 红外热成像具备了测温和成像功能。以下是部分实验测试图片。

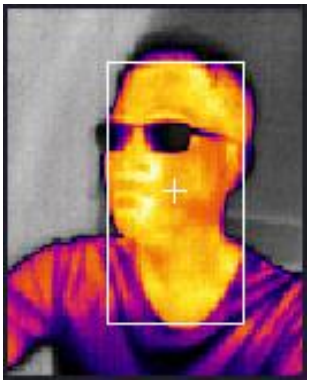


图 26-67

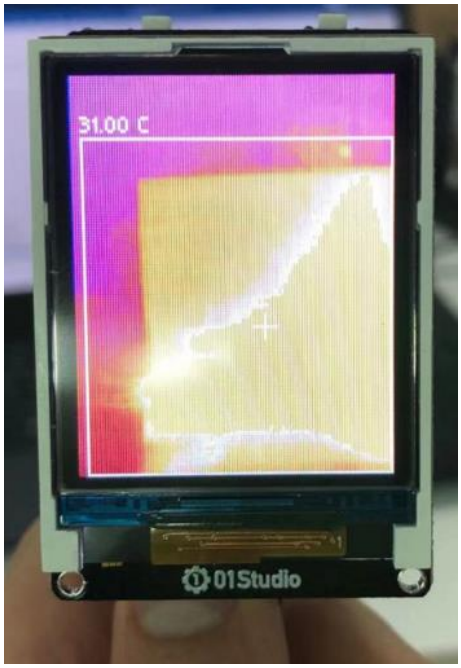


图 26-68

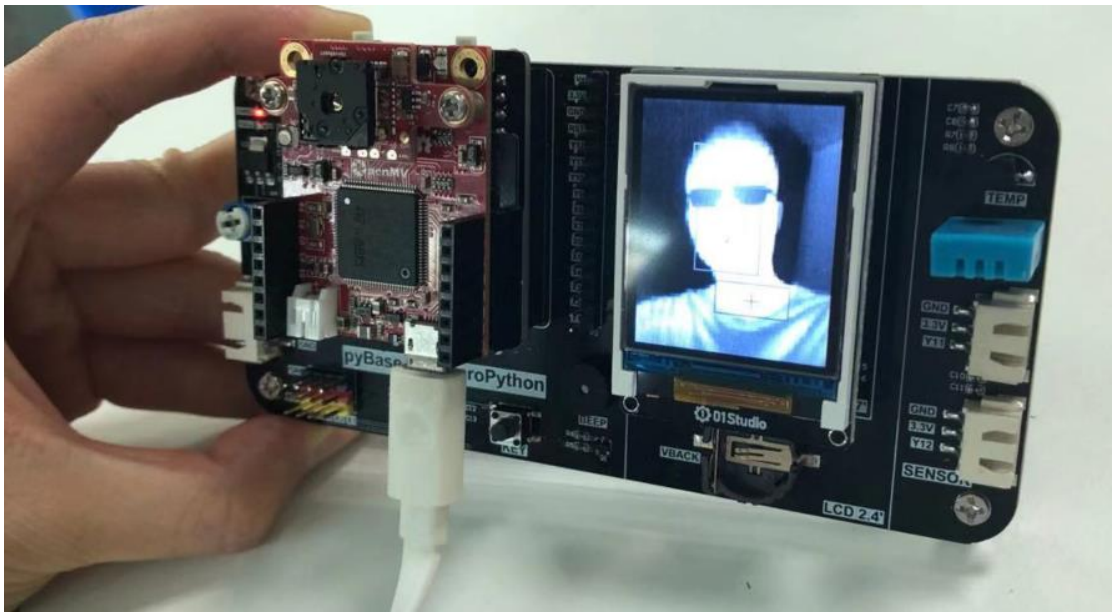


图 26-69

- **实验总结：**

通过本实验，可以看到 OpenMV 让 Lepton 红外成像变得非常简单有趣，你可以结合 pyAI-OpenMV4 打造属于自己的红外热成像仪器。

26.6 摄像头镜头

OpenMV4 摄像头镜头更换非常简单，只需要拧下原来镜头，将新的镜头拧上并调好焦距即可。

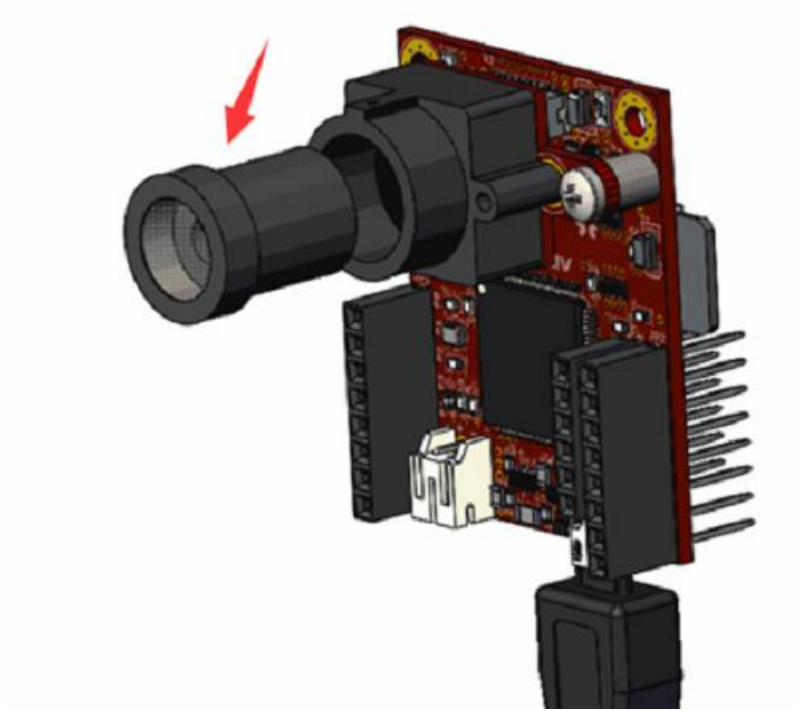


图 26-70

26.6.1 广角镜头

广角镜头是指可以让摄像头拍摄的范围变大，我们直接将广角镜头替换到 OpenMV4 上，通过旋转调好焦距即可使用。



图 26-71 广角镜头

功能参数	
焦距	1.7mm
像素	5mp（500 万）
视场角	1/2.5’’：185°
接口	M12*0.5 mm
功能	广角拍摄
其它	含滤光片

表 26-11

以下是标准镜头和广角镜头拍摄对比。

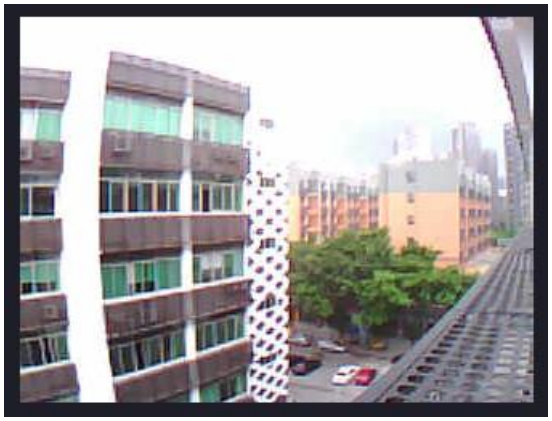


图 26-72 标准镜头



图 26-73 广角镜头

26.6.2 长焦镜头

长焦镜头可以理解成是望远镜，是指可以让摄像头拍摄远景，我们直接将长焦镜头替换到 OpenMV4 上，通过旋转调好焦距即可使用。



图 26-74 长焦镜头

功能参数	
焦距	2.5mm
像素	300 万
视场角	D*H*V（1/3’’）：13*11*8
畸变率	1/3’’：-0.55%
功能	长焦（远景）拍摄
其它	含滤光片

表 26-12

以下是标准镜头和长焦镜头拍摄对比。

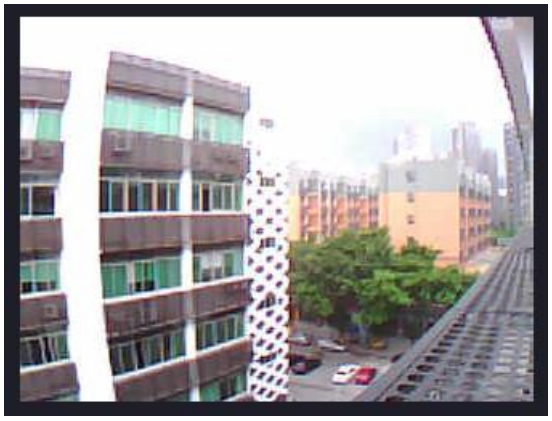


图 26-75 标准镜头



图 26-76 长焦镜头

26.6.3 无畸变镜头

标准镜头下的图片会出现一定的畸变（图像变形），而使用无畸变镜头可以解决这一问题，使拍摄出来的图片形状发生变形。我们直接将无畸变镜头替换到 OpenMV4 上，通过旋转调好焦距即可使用。



图 26-77 无畸变镜头

功能参数	
焦距	3.6mm
像素	500 万
视场角	D*H*V（1/2.5’’）：96*81.8*65.8
畸变率	1/2.5’’：<0.7%（无畸变）
光圈	F2.8
功能	无畸变拍摄
其它	含滤光片

表 26-13

以下是标准镜头和无畸变镜头拍摄对比。明细看到无畸变摄像头拍摄出来的图像两侧毫无变形。



图 26-78 标准镜头

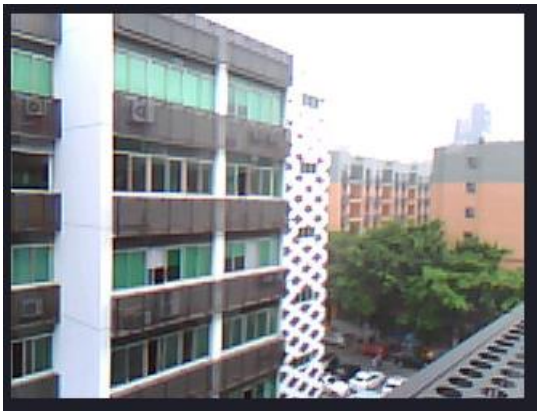


图 26-79 广角镜头

26.6.4 手动调焦镜头

手动调焦镜头可以让摄像头通过手动调节来获取最佳拍摄效果。我们直接将手动调焦镜头替换到 OpenMV4 上，通过旋转调好焦距即可使用。



图 26-80 手动调焦镜头

功能参数	
焦距	2.8mm – 12mm（4 倍变焦）
像素	300 万
光圈类型	固定
功能	手动调焦拍摄
其它	含滤光片

表 26-14

以下是标准镜头和手动调焦镜头拍摄对比。下面对比图片看到差别不大，但手动调焦的优势是针对不同距离的拍摄都可以任意调焦到最清晰的位置，特别是近景。支持 2.8-12mm。

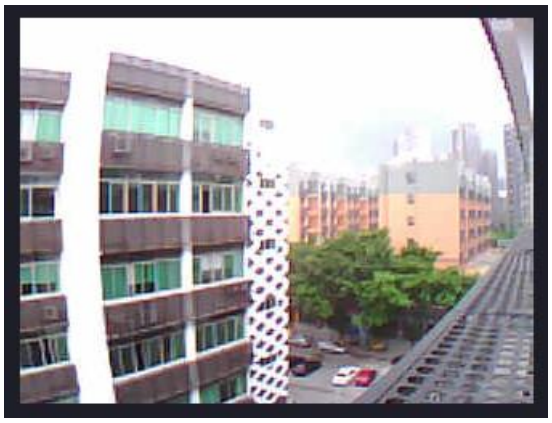


图 26-81 标准镜头

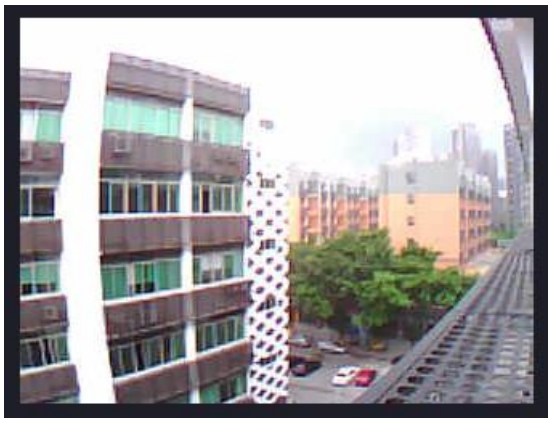


图 26-82 手动调节镜头

第27章　项目应用

27.1　照相机

- **前言：**

pyAI-OpenMV4 的外观非常酷，拥有按键功能，因此我们完全可以用来打造一台属于自己的照相机（尽管当前像素有点低）。现在无论数码相机还是手机都是带屏幕，主要是能实时显示方便拍摄，再配合锂电池，你就可以带上自制的照相机出去浪了！

- **实验平台：**

pyAI-OpenMV4 或 pyAI-OpenMV4 开发套件，需要使用 SD 卡和 LCD 显示屏。

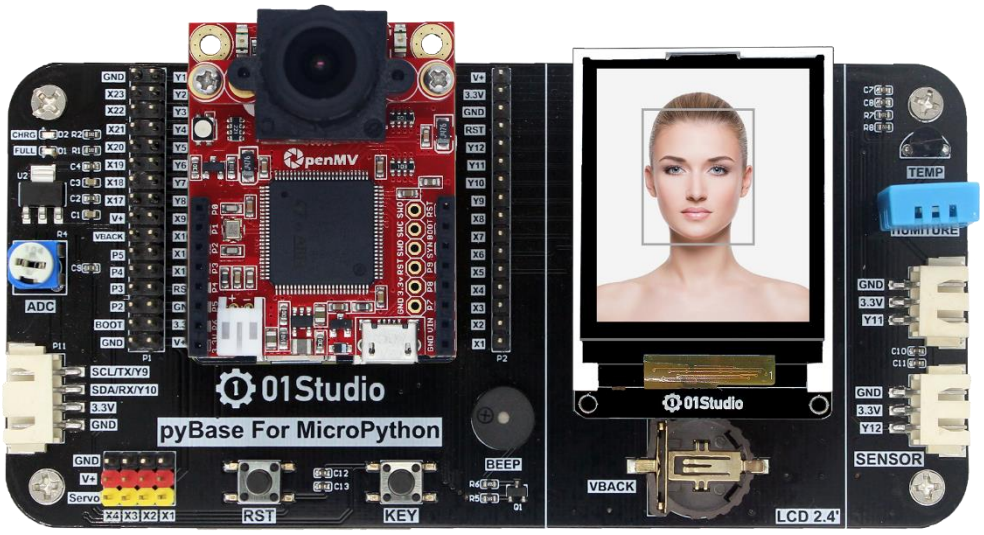


图 27-1 pyAI-OpenMV4 开发套件

- **实验目的：**

打造一台照相机，可以通过按键拍摄和 LCD 实时显示。

- **实验讲解：**

本项目主要是按键应用和拍照的相结合，这些内容可以在前面的实验找到，这里不再重复。

外部中断按键实验：请参阅 **23.4 外部中断** 章节内容；

拍摄照片实验：请参阅 **24.7.1 普通拍摄** 章节内容。

拍照后我们应该让图片停留一段时间，让用户观察照片的拍摄情况，然后再进行继续拍摄。代码编写流程如下：

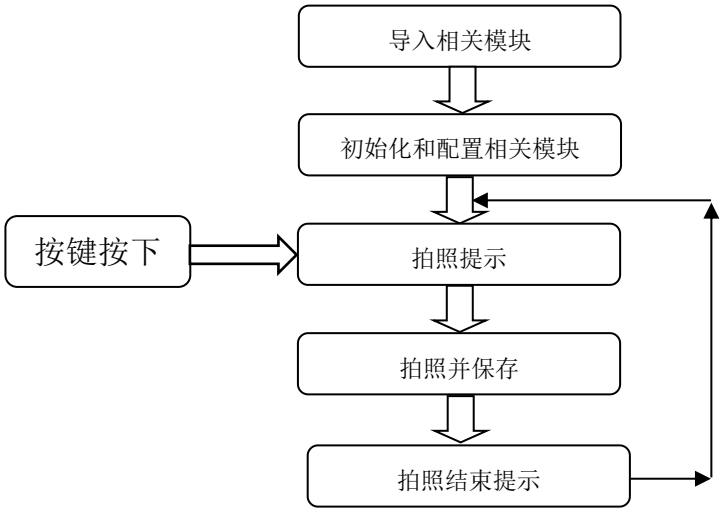


图 27-2 代码编写流程

参考代码如下：

```
'''
实验名称：照相机
版本：v1.0
日期：2019.10
作者：01Studio 【www.01Studio.org】
说明：通过按键拍照并在 LCD 上显示（本实验需要 SD 卡）。
'''



import sensor, image, pyb, lcd
from pyb import ExtInt,Pin,RTC

```

```
RED_LED_PIN = 1
BLUE_LED_PIN = 3

#摄像头初始化
sensor.reset() # Initialize the camera sensor.
sensor.set_pixformat(sensor.RGB565) # or sensor.GRAYSCALE
sensor.set_framesize(sensor.LCD) # or sensor.QQVGA (or others)
sensor.skip_frames(time = 2000) # Let new settings take affect.

#LCD 初始化
lcd.init()

#时间初始化，用于给图片命名
rtc=RTC()

key_node = 0  #按键标志位

################################################
#  按键和其回调函数
################################################
def key(ext):
    global key_node
    key_node = 1

ext = ExtInt(Pin('P9'), ExtInt.IRQ_FALLING, Pin.PULL_UP, key) #下降沿触发，打开上拉电阻

while True:

    lcd.display(sensor.snapshot()) # LCD 实时显示
```

```
if key_node==1: #按键被按下
    key_node = 0 #清空按键标志位

    #红灯亮提示用户看镜头
    pyb.LED(RED_LED_PIN).on()
    sensor.skip_frames(time = 2000)

    #红灯灭，蓝灯亮提示开始拍照
    pyb.LED(RED_LED_PIN).off()
    pyb.LED(BLUE_LED_PIN).on()

    print("You're on camera!")

    #拍照并保存，保存文件用时间来命名。
    lcd.display(sensor.snapshot().save(str(rtc.datetime())+".jpg"))

    pyb.LED(BLUE_LED_PIN).off()
    print("Done! Reset the camera to see the saved image.")

    #延时 3 秒，观看拍摄图片
    pyb.delay(3000)
```

- **实验结果：**

pyAI-OpenMV4 插入 SD 卡、接上 LCD 和锂电池。然后将以上代码编写好的 main.py 文件复制到 OpenMV4 文件系统中，按下复位即可运行。

然后对准要拍摄的东西，按下上方的 KEY 按键，即可拍摄！

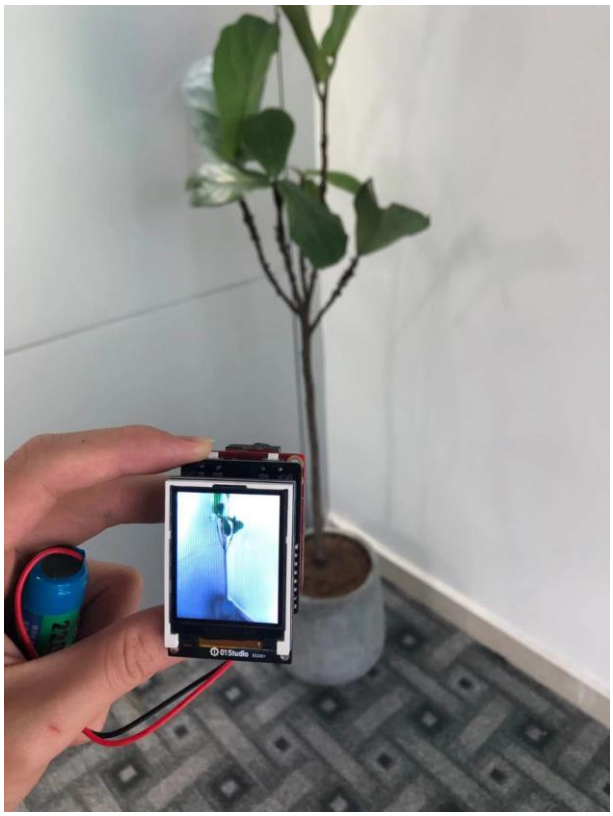


图 27-3 拍摄图片

当然你也可以结合 pyBase 底板来自拍：

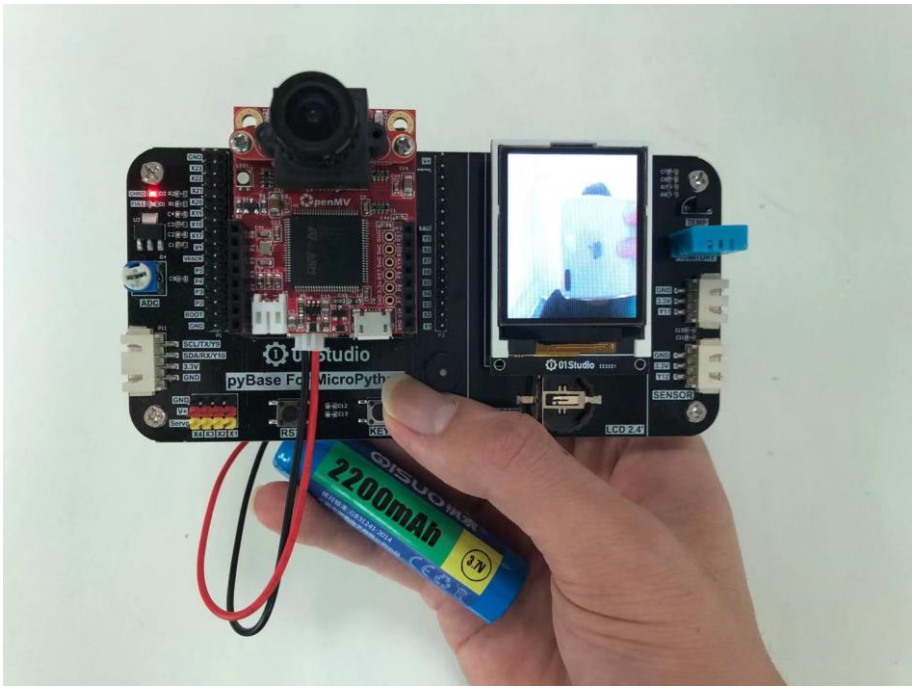


图 27-4 结合开发底板自拍

通过 Usb 线连接到 OpenMV4，按下复位，可以看到被保存的图片文件。（RTC 时间没做修改）

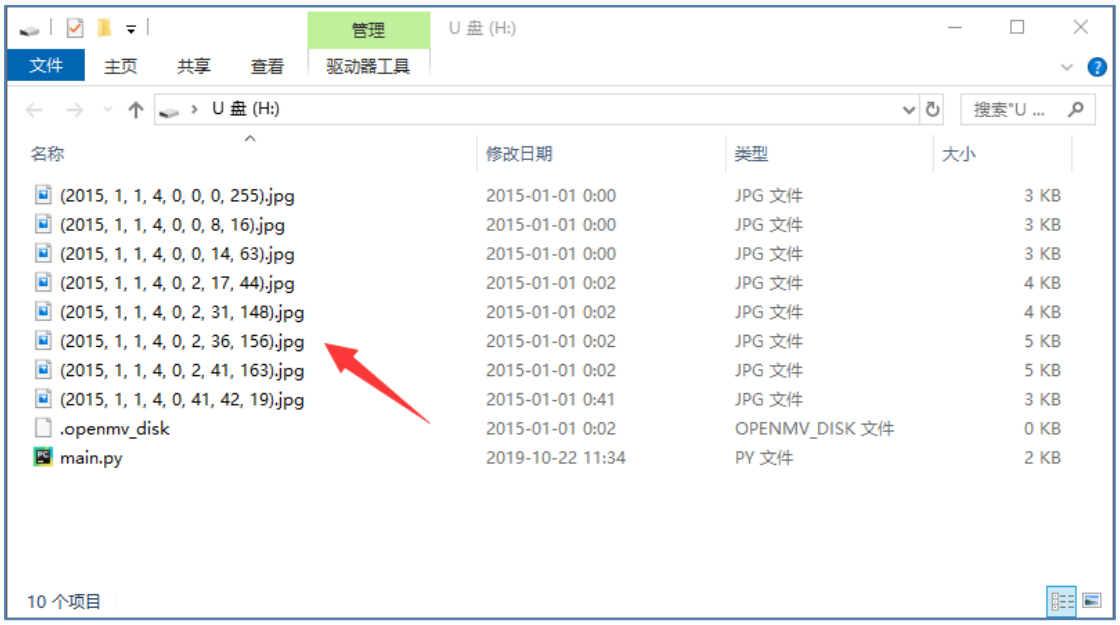


图 27-5 保存的图片文件

- **总结：**

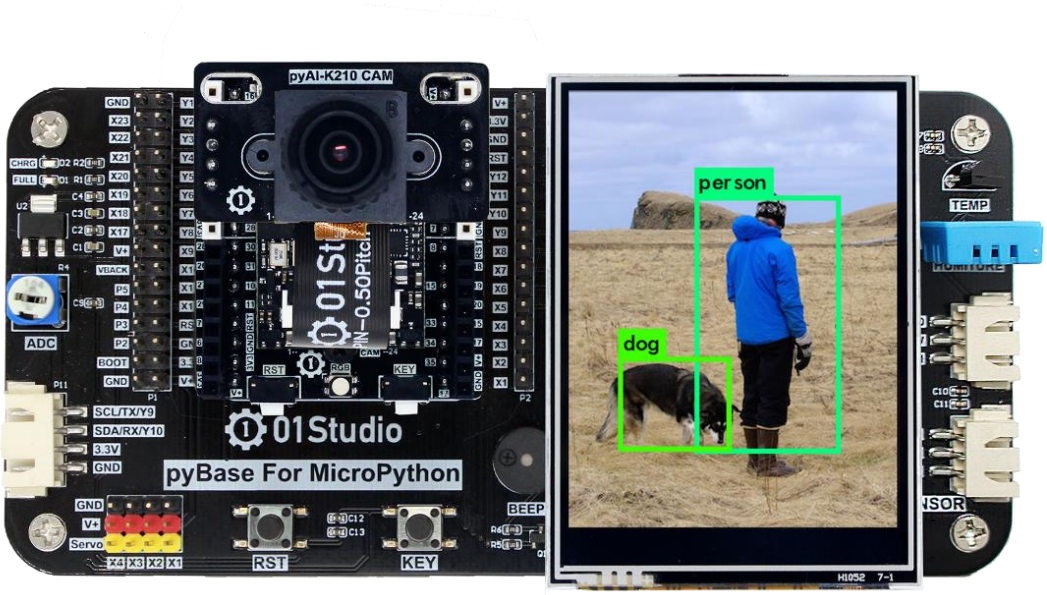
本节学习了一个简单的项目应用，照相机。可以看到项目是前面实验的一个综合，只要把基础知识掌握牢固了，就可以轻松使用 pyAI-OpenMV4 实现更多好玩有趣的项目。同时 OpenMV 也是不断升级的产品，相信不久就能用上更高清的摄像头。

第七部分　基于 K210 平台

K210 是嘉楠（Cannaan）科技的一款集成机器视觉与机器听觉能力的系统级芯片 (RSIC-V CPU)。使用台积电 (TSMC) 超低功耗的 28 纳米先进制程，具有双核 64 位处理器，拥有较好的功耗性能，稳定性与可靠性。该方案力求零门槛开发，可在最短时效部署于用户的产品中，赋予产品人工智能（AI）。可以说是集性能强劲与高性价比于一身。

这么强大的 MCU，配合 MicroPython 进行开发可以说是如虎添翼，那就是 MaixPy 开源项目。MaixPy 是中国 Spieed 团队发起将 MicroPython 移植到 K210 的开源项目， 通过 MicroPython 编程，我们可以轻松实现基于 K210 的 MCU 编程和各类 AI 算法，如机器视觉、机器听觉等。可以说这个平台是一个彻头彻尾的国产开源项目。在开源技术越来越流行的今天，我们 01Studio 希望国内能有更多更优秀的开源项目走向全球。

如果你学习了前面的平台，那么恭喜你，对于 K210 平台你会非常快速上手。如果你没有学习过任何一个 MicroPython 平台那也没关系，你可以借助 K210 平台和 Maxipy 的学习从而入门 MicroPython。



pyAI-K210 开发套件

第28章　开发环境快速建立

28.1　基于 Windows

28.1.1　安装 python3

说明：我们提供安装 python3 教程只是为了在 PC 上体验一下 python3 安装和使用过程，由于 micropython 固件集成了 python 库，所以电脑并非一定要安装，本节可以跳过。

请阅读 第二部分　3.1.1　安装 python3 章节内容，这里不再重复。

28.1.2　安装开发软件 MaixPy IDE

MaixPy 拥有自己官方的 IDE，可以在官网下载，我们使用该 IDE 可以轻松进行开发。而且拥有 Windows、Mac OS、Linux 等版本。

当前版本为 v0.2.4，官网下载地址：（如有更新请下载最新版）http://cn.dl.sipeed.com/MAIX/MaixPy/ide/_/v0.2.4，下载界面如下图。用户可以根据自己系统选择合适的版本下载安装。Window 用户选择 exe 格式文件下载安装。

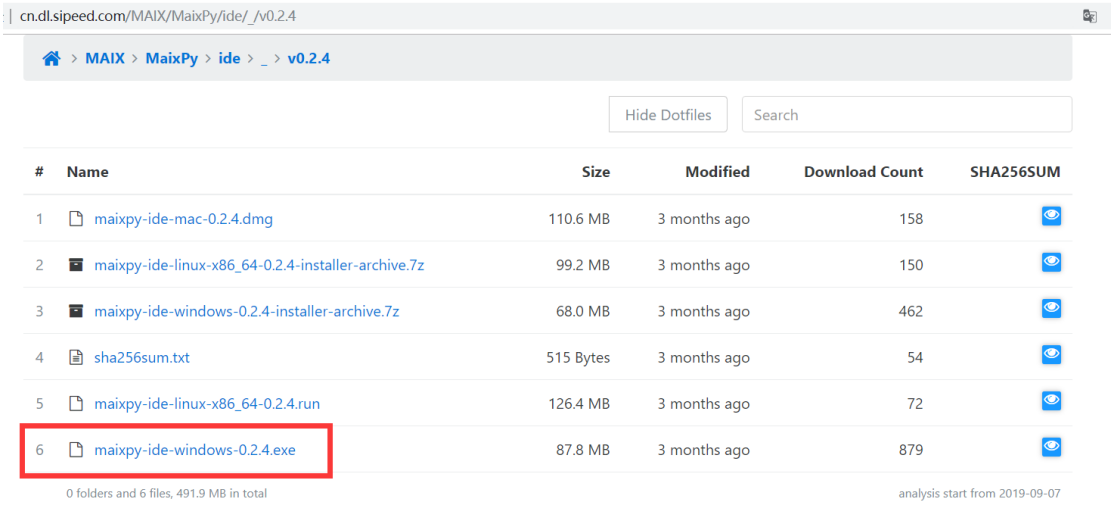


图 28-1 MaixPy IDE 下载界面

安装完成后如果在桌面没用找到该软件，可以在安装路径或者在系统搜索栏搜索 Maixpy 关键词找到该软件。

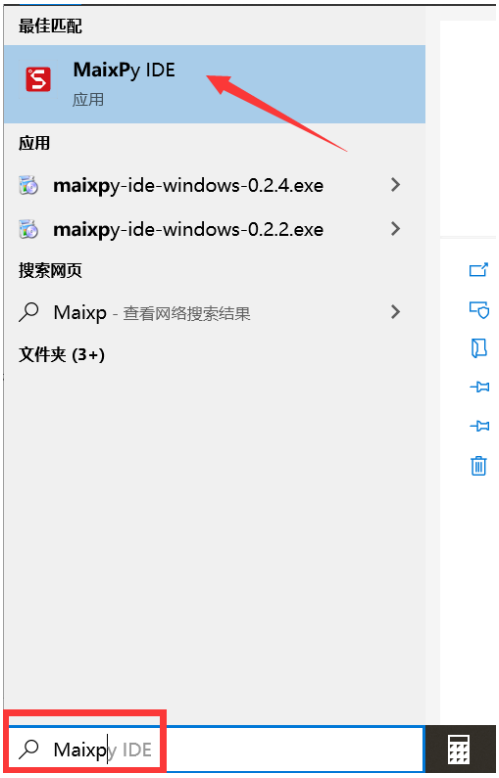


图 28-2 搜索 Maixpy IDE

打开软件，如下图所示，至此安装完成。可以看到跟 OpenMV IDE 长得很像，因为 MaixPy 就是基于 OpenMV IDE 衍生出来的。

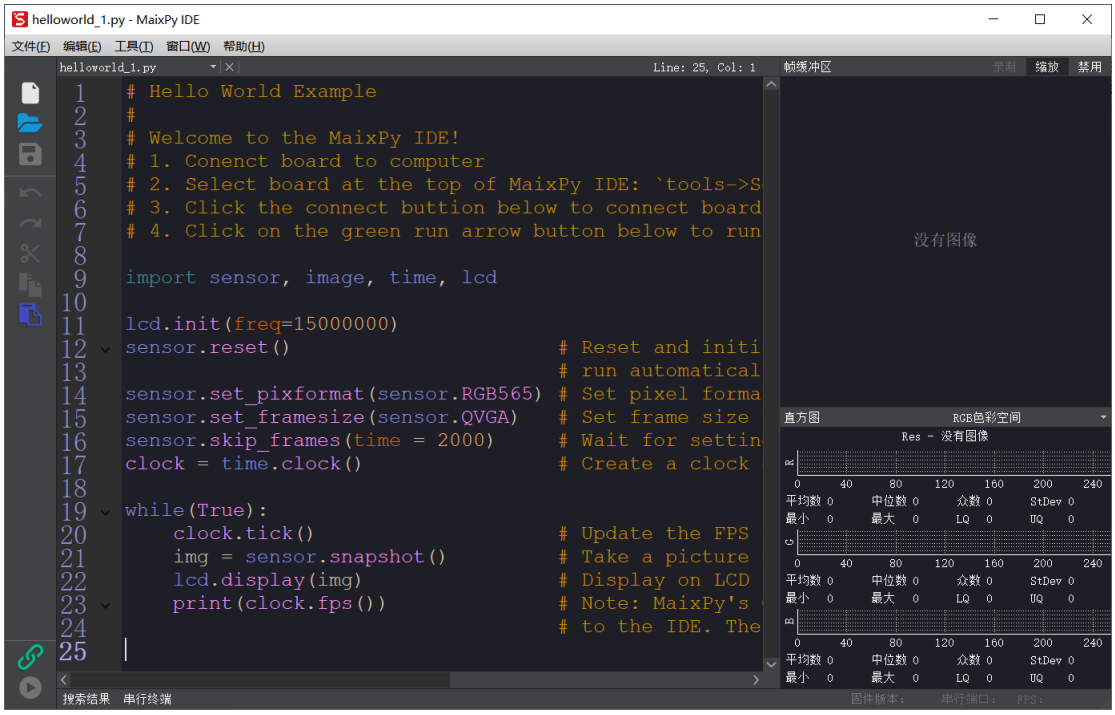


图 28-3 MaixPy IDE

28.1.3 开发套件使用

28.1.3.1 摄像头接线

摄像头模块建议使用配套的 6cm 长排线。

pyAI-K210 集成了 24P 摄像头接口，可以直接连接标准 OV2640 等 24P 摄像头模块，排线接线方式均为下接（排线金手指朝下），具体如下：**（注意核心板上的 FPC 座为后翻盖，也就是在后边翻开，插进排线，再锁紧）**

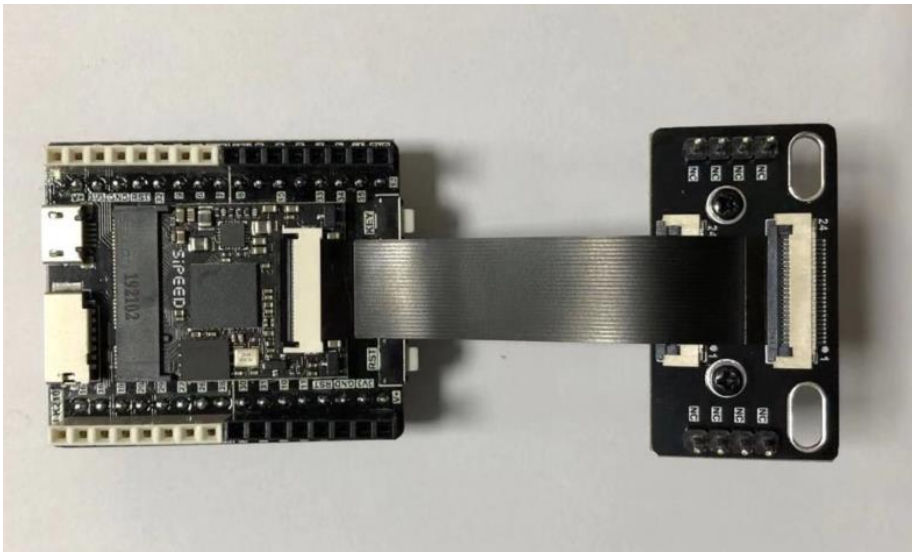


图 28-4 摄像头接线方式

接完后可以直接将 K210 摄像头模块插到 pyAI-K210 的拓展排母上，由于摄像头的排针均没有电气连接，所以可以根据自己需要任意摆放位置。

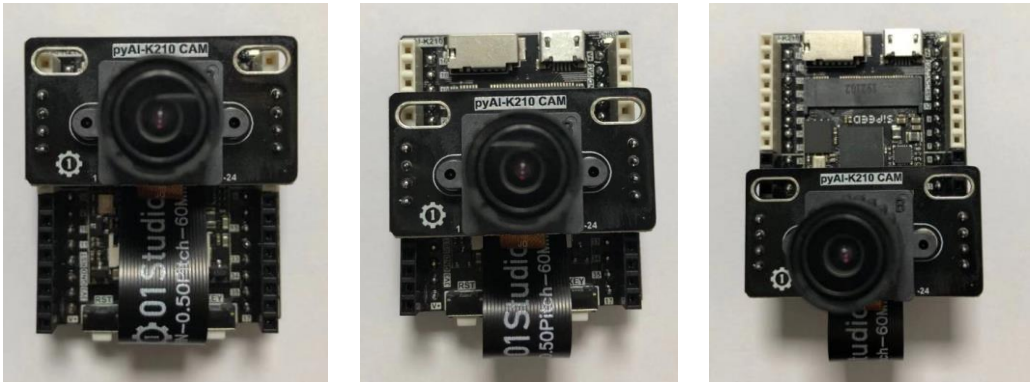


图 28-5 摄像头位置任意摆放

28.1.3.2 LCD 接线

LCD 模块建议使用配套的 20cm 长排线。

2.8 寸 LCD 跟 pyAI-K210 通过底部的 24P 排线连接，注意排线均为下接（排线金手指朝下）。将排线塞进座子，扣下即可。

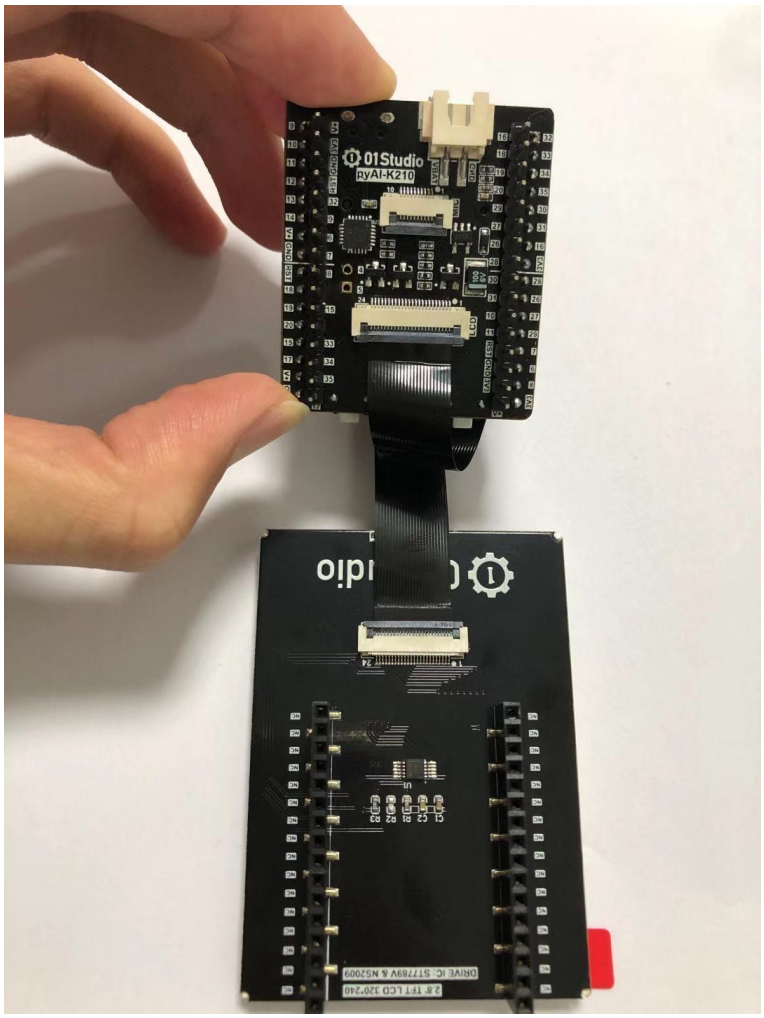


图 28-6 pyAI-K210 与 2.8 寸 LCD 接线方式

2.8 寸 LCD 背面的排母是没有电气连接的，仅仅是为了方便用户安装，可以接到 pyAI-K210 背面或者 pyBase 上，具体如下：

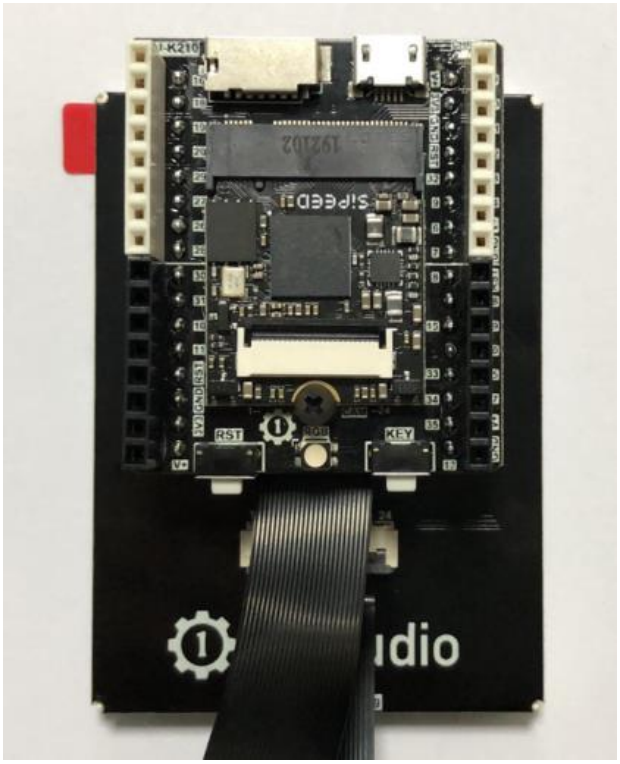


图 28-7　LCD 与 pyAI-K210

pyAI-K210 与 LCD 连接后，可以插到 pyBase 上，建议排线藏到后方，首次插进注意排线不要压着即可。组装完成如下图所示。

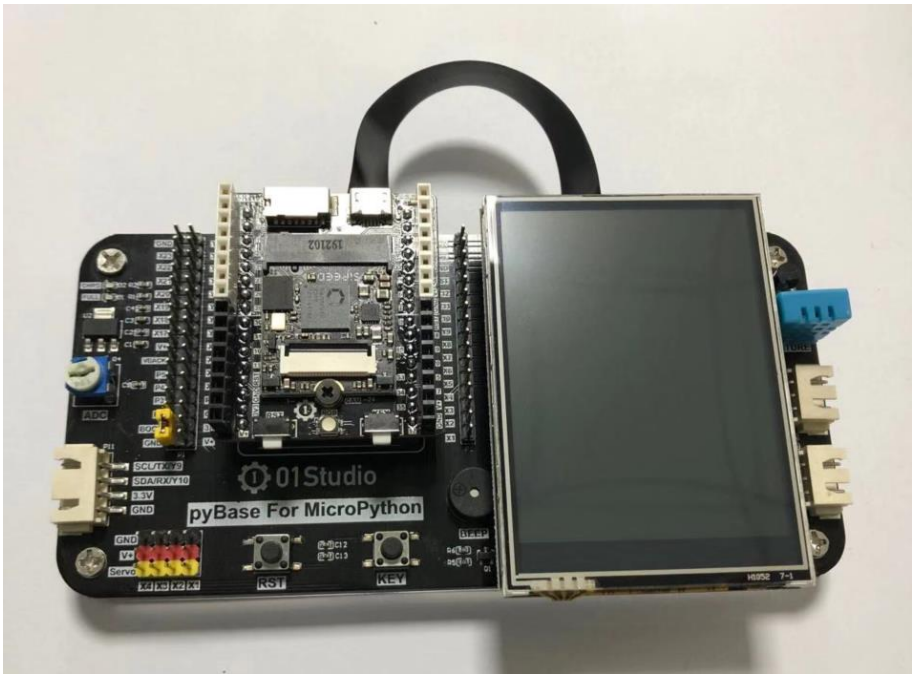


图 28-8 LCD 与 pyBase

28.1.3.3 驱动安装

pyAI-K210 通过串口烧写程序和通信，因此主主要是安装 USB 转串口驱动。我们将 pyAI-K210 开发板通过 MicroUSB 数据线连接到电脑：

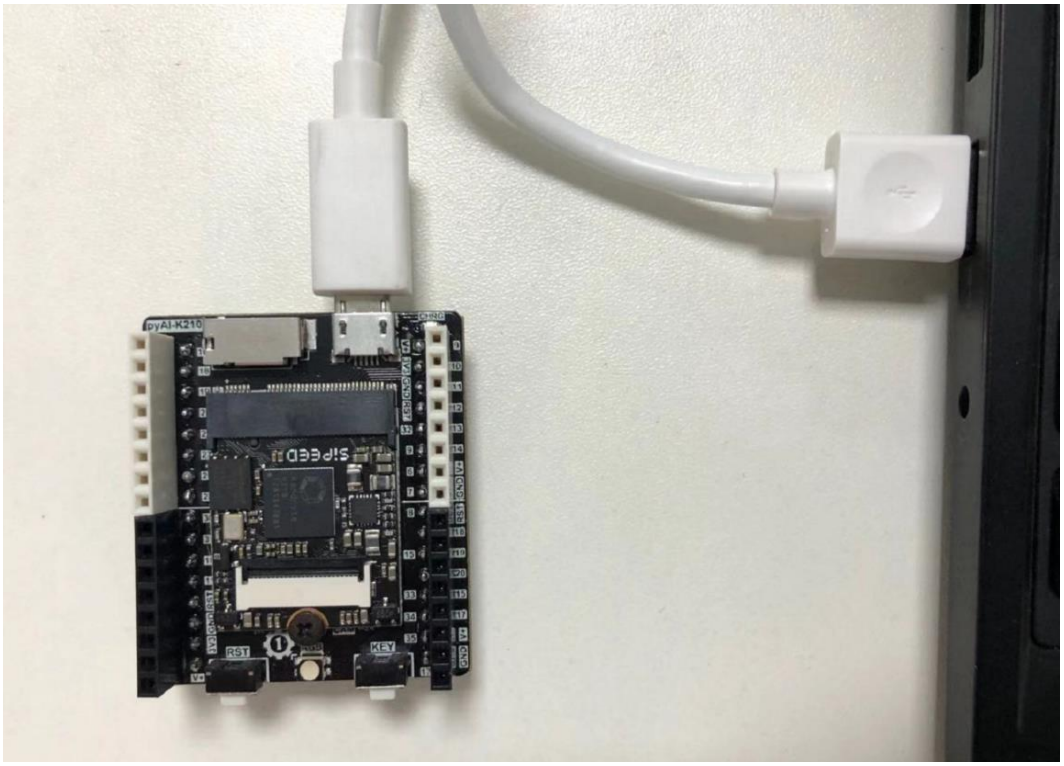


图 28-9 通过 MicroUSB 线连接到电脑

如果你的操作系统是 Win10,一般情况下能自动安装。鼠标右键点击“我的电脑”—属性—设备管理器： 出现串口号说明安装成功，如下图所示。

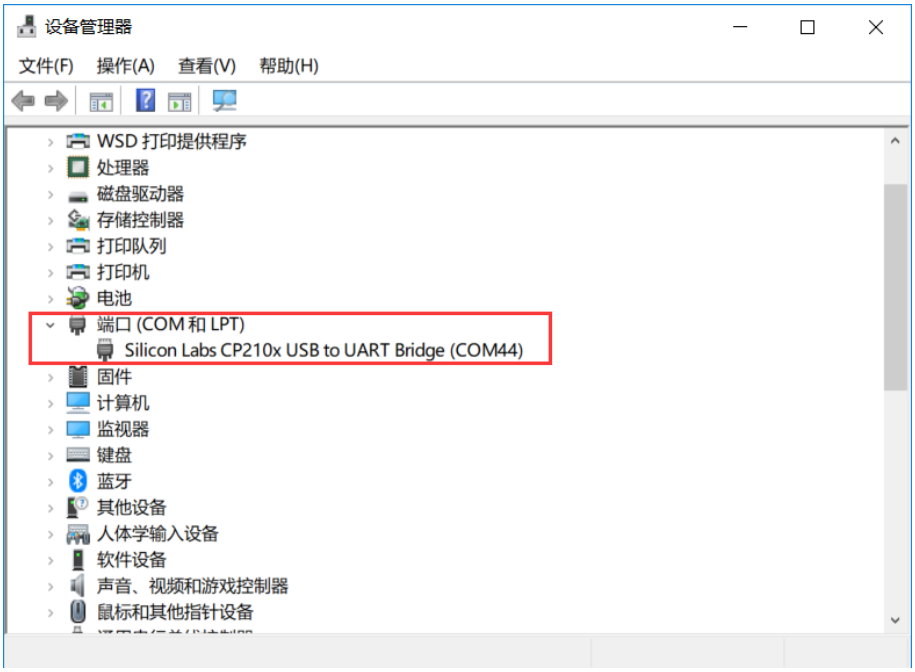


图 28-10 串口驱动成功安装

如果无法安装，请手动安装驱动，方法如下：

不能自动安装时候，设备会出现黄色叹号，这时候点击设备右键，选择“更新驱动程序”，选择“浏览计算机查找驱动”：

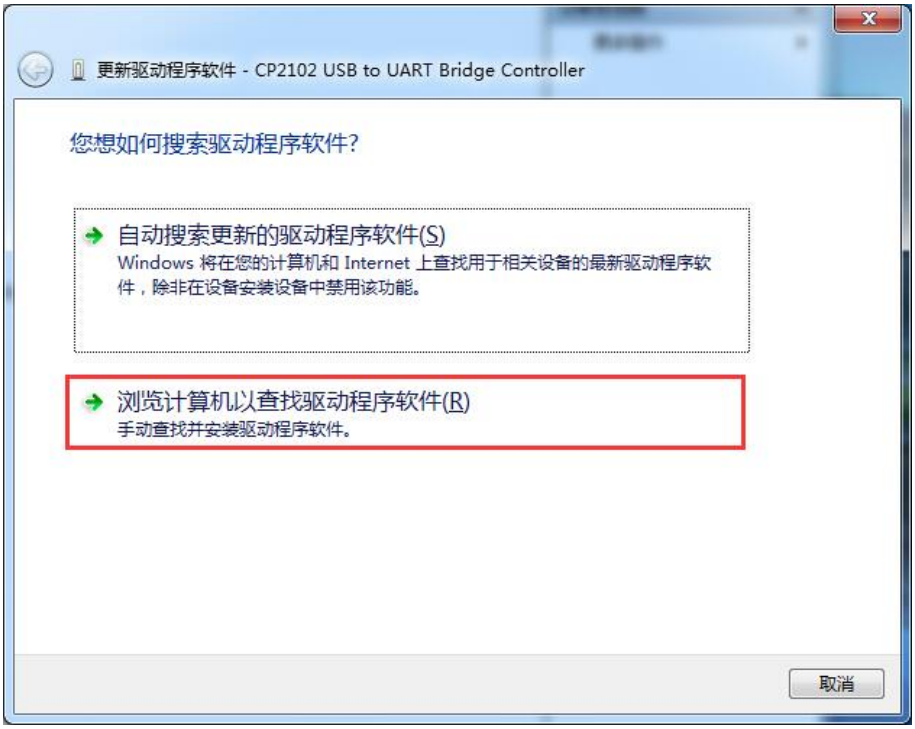


图 28-11

驱动路径选择：零一科技（01Studio）MicroPython 开发套件配套资料\01-开发工具\01-Windows\串口终端工具\CP210x 驱动，点击确认后即可自动安装：

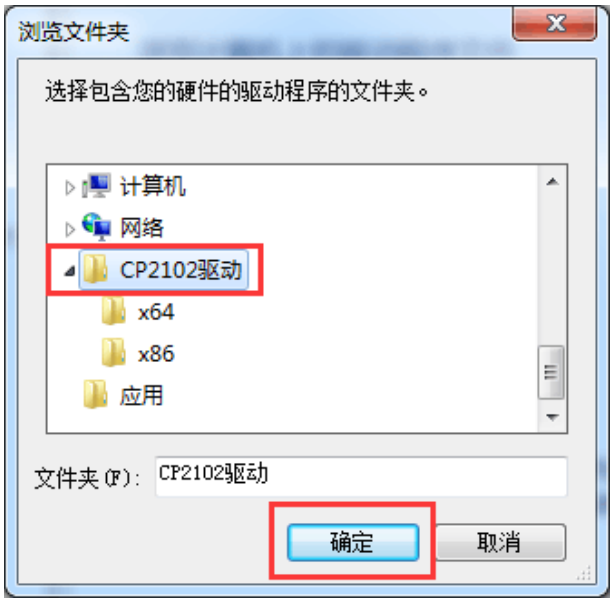


图 28-12

安装成功后如下图：

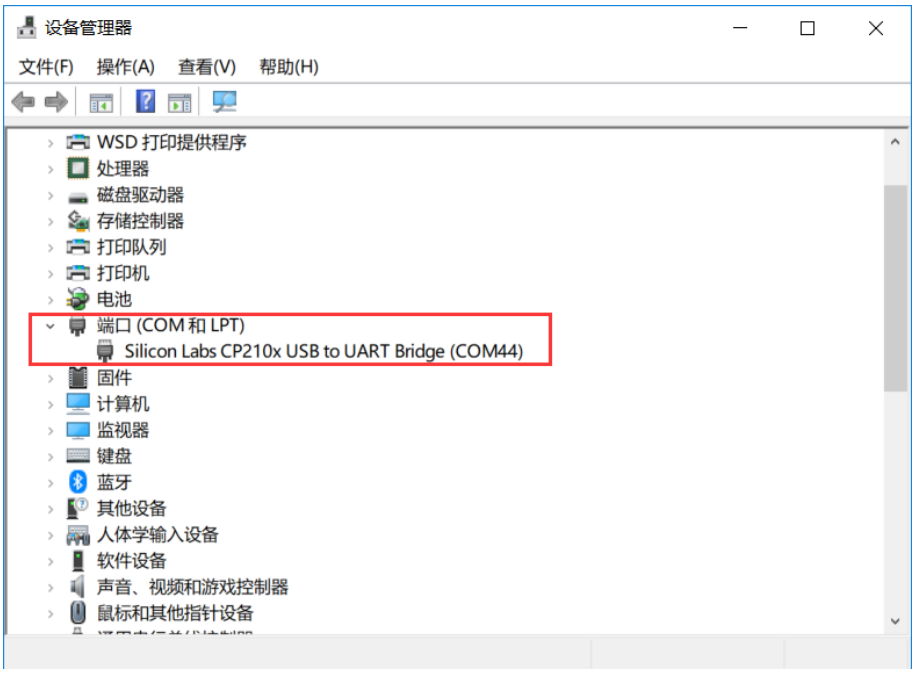


图 28-13 串口驱动安装成功

28.1.3.4 例程测试

我们使用 MaixPy IDE 来进行我们的第一个实验，借此来熟悉开发环境。将 pyAI-K210 开发板通过 MicroUSB 线连接到电脑。打开 MaixPy IDE，我们先打开配套资料包里面的例程代码（这里暂时不对代码进行讲解，后面章节会有详细内容）。

我们用最简单的 LED 程序来测试，在 MaixPy IDE 中打开 零一科技（01Studio）MicroPython 开发套件配套资料 latest\02-示例程序\5.pyAI-K210\1.基础实验\1.点亮第一个 LED 里面的 LED.py 例程（也可以直接拖动过去），如下图所示：



图 28-14

接下来我们需要连接开发板，pyAI-K210 的串口驱动芯片跟 Maix 的 DOCK 接近，因此可以在 IDE 顶部点击工具，选择 Dock。

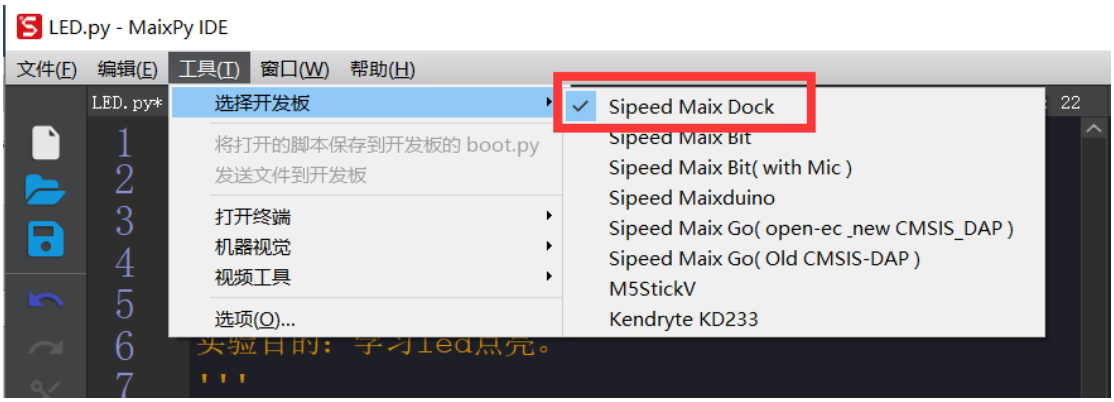


图 28-15

接下来点击左下角连接按钮：



图 28-16 点击连接按钮

连接成功后，运行按钮变成绿色。

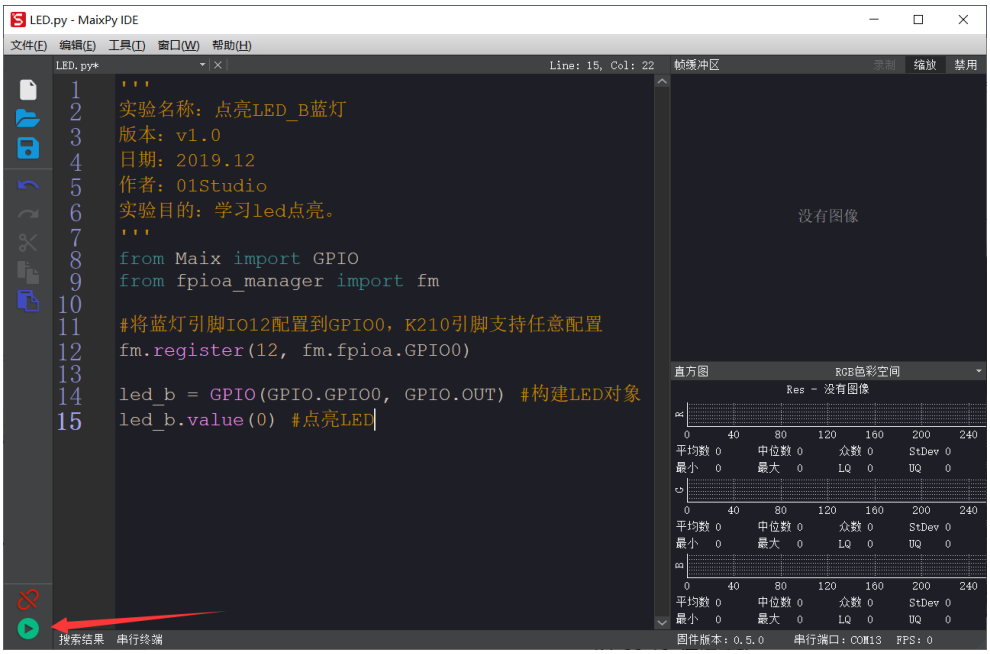


图 28-17 连接成功

当前的例程是点亮 LED 蓝灯，我们点击绿色按键“运行”按钮，当看到 pyAI-K210 开发板上的蓝灯亮时，说明实验成功:

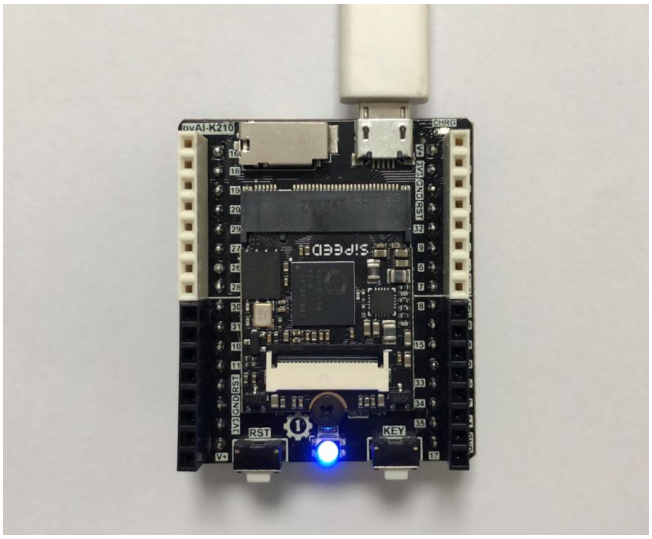


图 28-18 第一个实验

以上测试是基于 IDE 的在线运行功能，当然你也可以直接将 py 文件保存到开发板的文件系统中，这样程序不再依赖 IDE 就可以直接运行。具体方法如下：

在连接状态下点击工具—将打开的脚本保存到开发板的 boot.py，这里的意思是将当前编辑框的代码拷贝到开发板文件系统中的 boot.py，由于 boot.py 是 Maixpy 上电运行的第一个脚本文件，因此相当于实现了上电运行写入的程序。

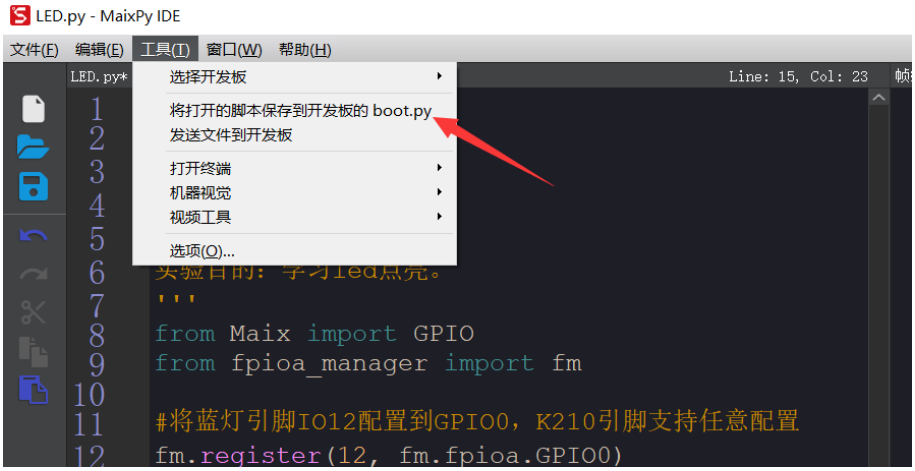


图 28-19 复制文件到 pyAI-K210 文件系统

点击后出现以下提示，点 yes 确认。

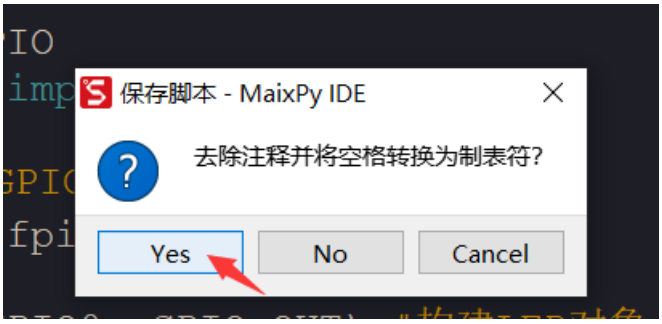


图 28-20 点击 YES

当出现保存脚本成功，说明程序烧写成功。

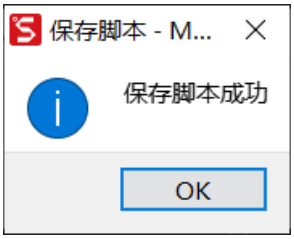


图 28-21

我们按下开发板的复位键(RST)，可以看到开发板上电后蓝灯亮，说明程序文件已经烧录进去了。

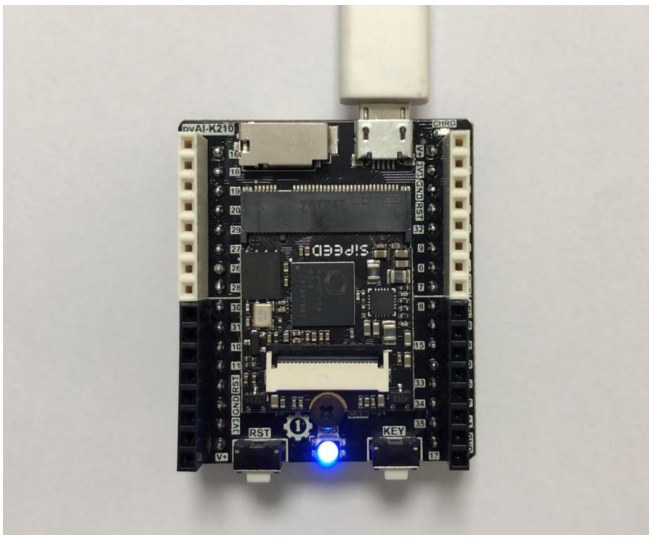


图 28-22 复位后程序运行

Maixpy IDE 集成了串口调试窗口，点击 IDE 左下角“串口终端”，即可观察到串口打印的信息。

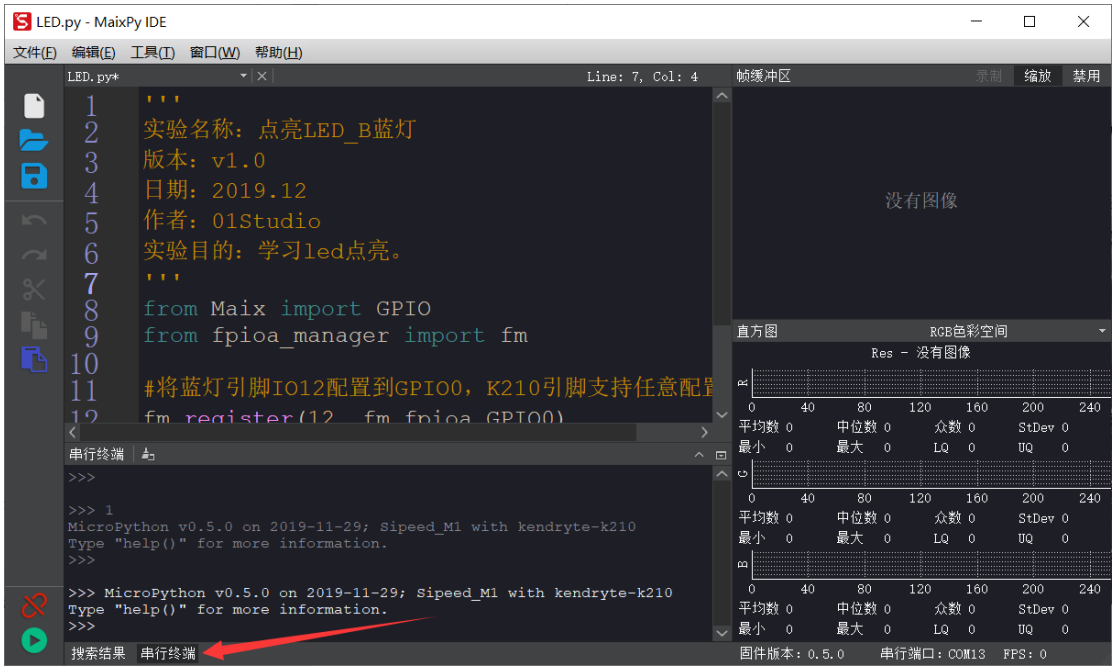


图 28-23 串口终端信息显示

28.1.3.1 REPL 串口调试

上一节在 IDE 中的串口终端打印调试数据，并不能实现交互，pyAI-K210 的 MicroPython 固件集成了交互解释器 REPL 【读取(Read)-运算(Eval)-输出(Print)-循环(Loop) 】，开发者可以直接通过串口终端来调试 pyboard 或 micropython 开发套件。我们使用的软件是一款免费的串口终端软件 putty。

将开发板连接到电脑，从我的电脑—属性—设备管理器中找到当前的串口号，这里是 COM4。

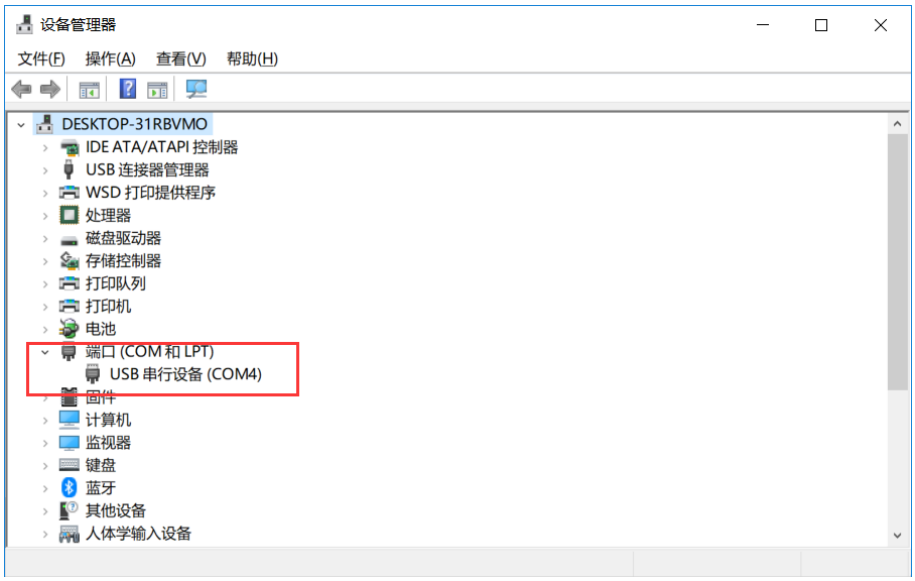


图 28-24 找到串口号

打开 MicroPython 开发套件配套资料\开发工具\串口终端工具\Putty.exe，选择左下角 Serial，配置信息如下：

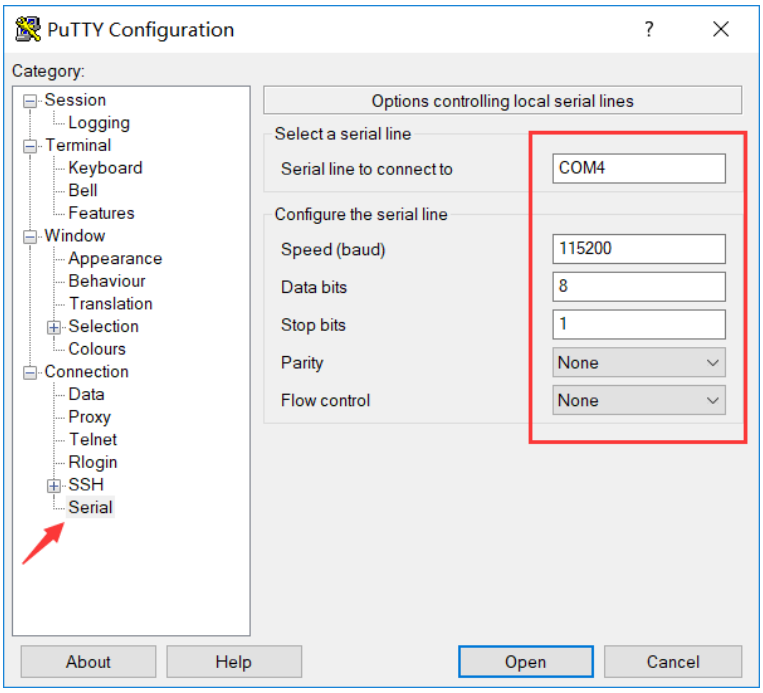


图 28-25 配置串行设备参数

配置好后不是点 open，而是点左边上方 Session，选择 Serial 后可看到刚刚的配置信息。串口号通常不会变化，我们在 Save Session 下方输入 COM4 或者自己喜欢的名称，点右边 Save，在空白框里面就出现 COM4 字样，以后可以直接使用。设置好后我们点击 Open。

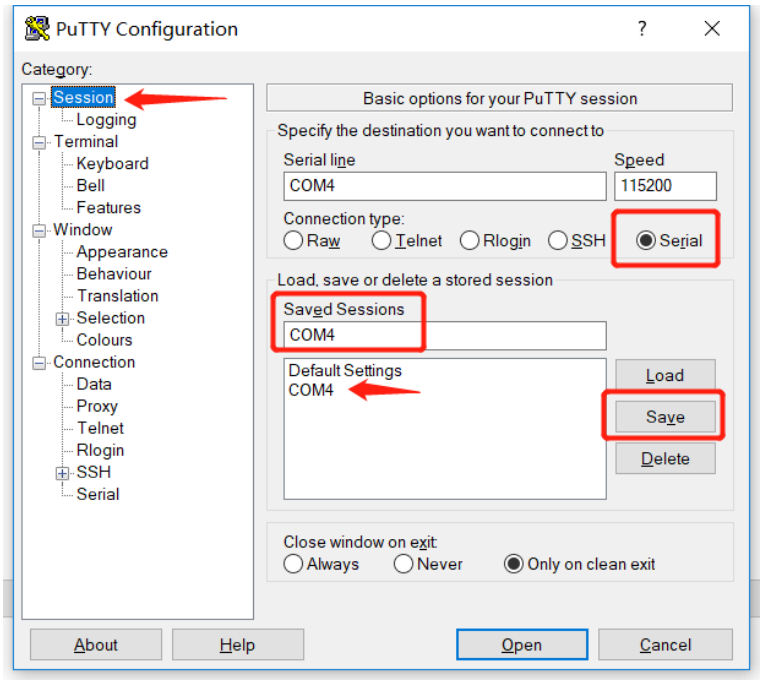


图 28-26

出现下面对话框，是不是似曾相识，没错，跟我们之前在计算机命令行执行 python 指令后出现类似的。上面主要是当前设备固件的版本号。

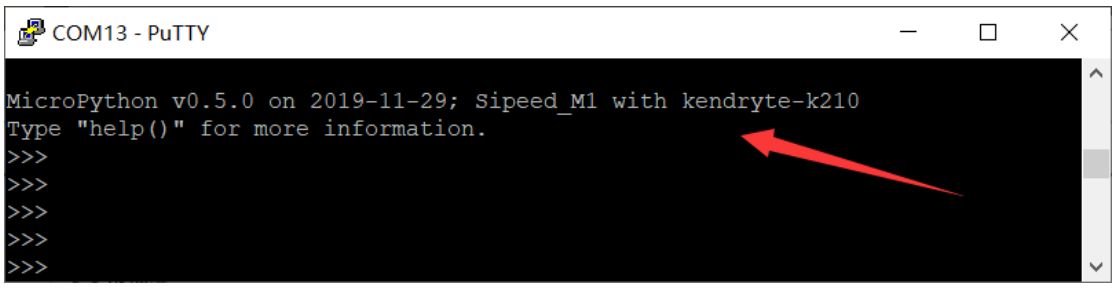


图 28-27

现在对话框相当于连接上了开发板上，由于 pyAI-K210 集成了 MicroPython 解析器。我们在这里可以进行调试和简单编程，接下来我们测试一下。在对话框输入下面代码，按回车，可以看到代码运行情况。

```
print(“Hello 01Studio!”)
```

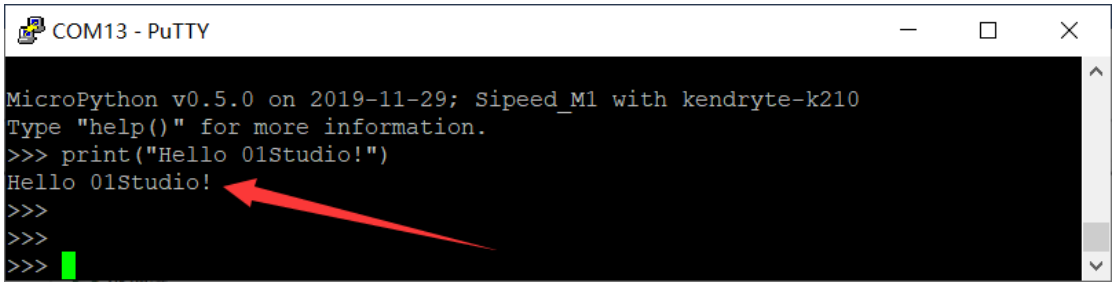


图 28-28

我们再输入 1+1 按回车，得到了计算结果 2。

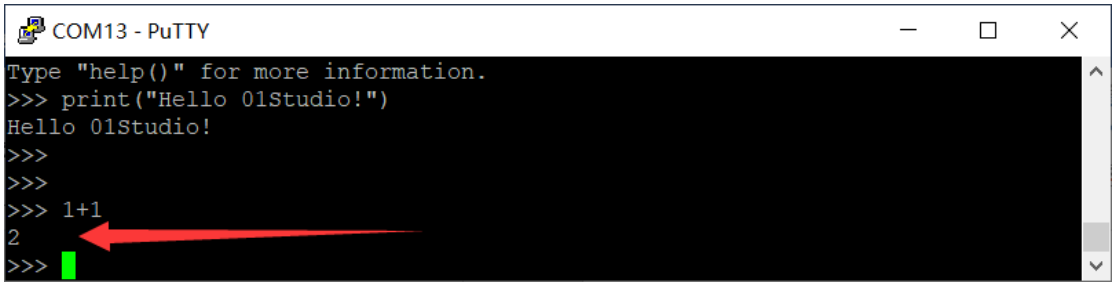


图 28-29

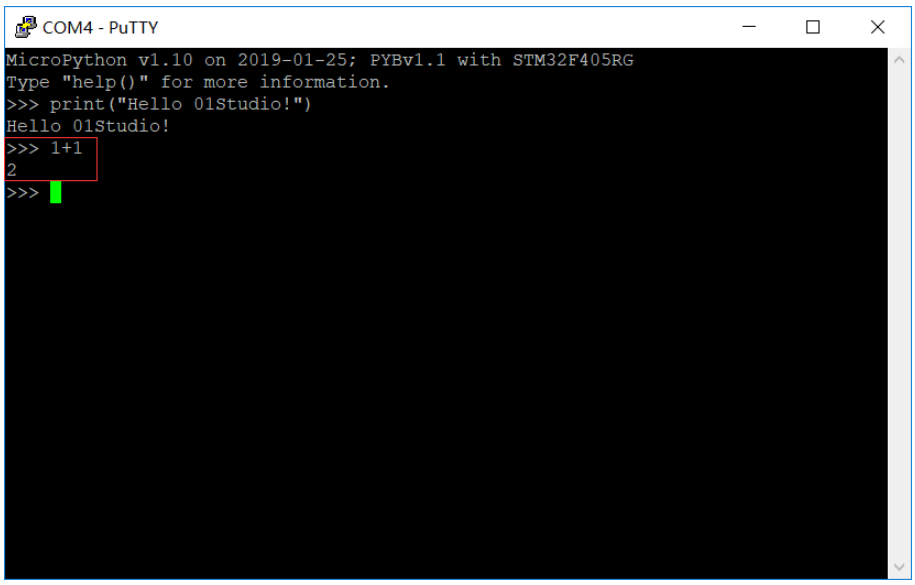


图 28-30

温馨提示：putty 在 windows 操作系统下，当按下开发板复位键的时候，由于系统重启过程中虚拟串口消失，会导致 putty 会死掉，需要重新开启。所以我们可以使用【Ctrl+D】组合键方式软件复位开发板。如下图所示：

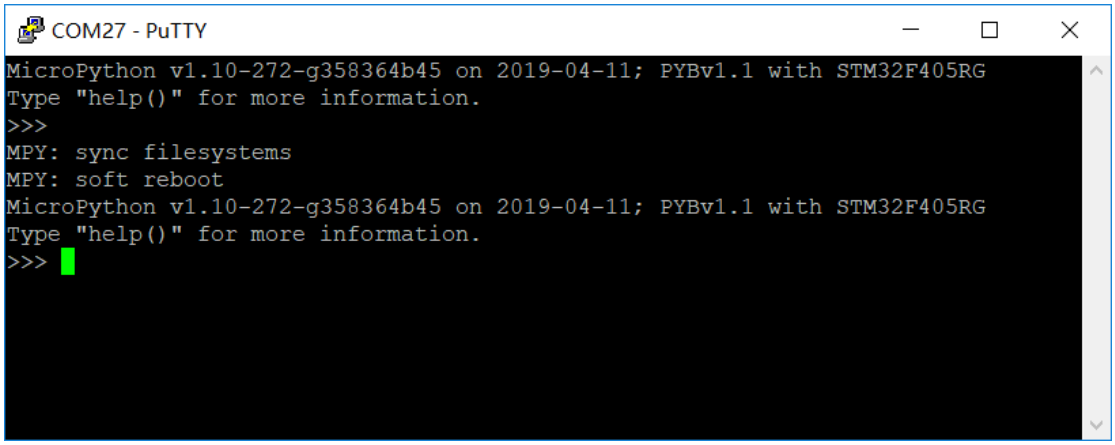


图 28-31 软件复位（soft reboot）

28.1.3.2 文件系统

pyAI-K210 里面内置了文件系统，可以简单理解成上电后运行的 python 文件，这个可以通过串口 REPL 结合命令来查看。

打开 putty 进入 REPL，依次输入下面命令，每行输入后按回车键。

```
import os

os.listdir()
```

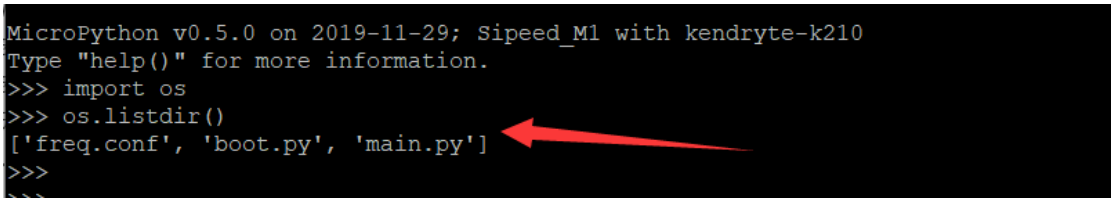


图 28-32 文件列表

可以看到一共有以下 3 个文件：freq.conf、boot.py、main.py。

【freq.conf】：CPU 参数配置

【boot.py】:上电运行的第一个脚本文件

【main.py】：上电运行的第二个脚本文件，也是主函数文件。

前面章节讲到的 MaixPy IDE 中 工具--将打开的脚本保存到开发板的 boot.py 实际上就是将脚本文件保存到开发板文件系统。当然我们也可以使用 IDE 中传输文件功能，非常方便：

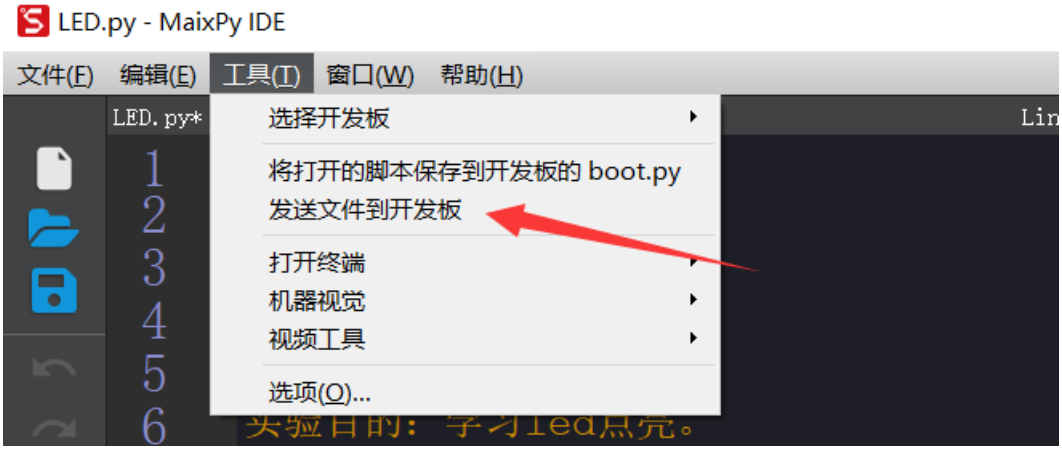


图 28-33

我们可以选择基础实验中的 LED.py 文件，保存到开发板。

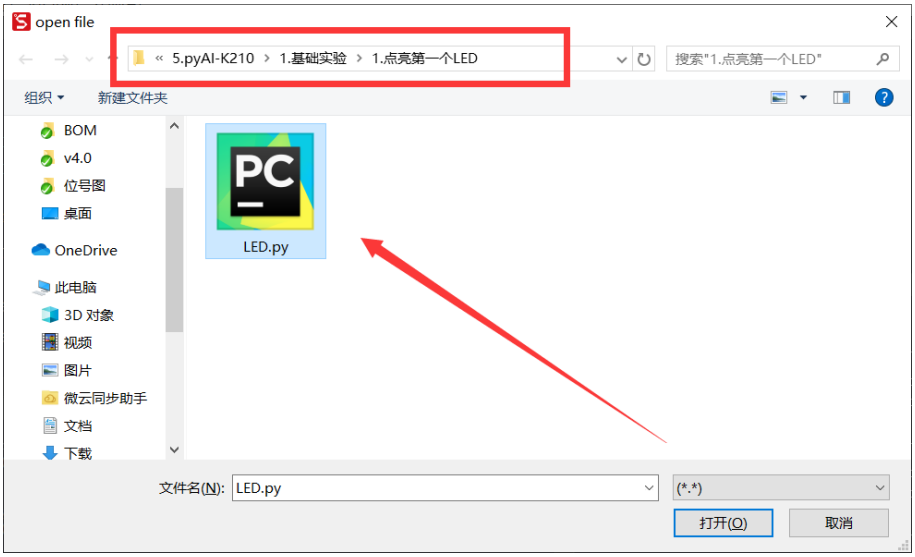


图 28-34

保存成功后再通过 REPL 查看文件，发现多了 LED.py。

```
MicroPython v0.5.0 on 2019-11-29; Sipeed_M1 with kendryte-k210
Type "help()" for more information.
>>> import os
>>> os.listdir()
['freq.conf', 'boot.py', 'LED.py', 'main.py']
>>>
```

图 28-35 文件拷贝成功

如果想删除某个文件，可以使用以下指令：

```
os.remove('xxx.py')
```

当插入 SD 卡时候，系统会自动以 SD 作为存储的文件系统。优先执行 SD 卡上面的脚本文件。

（**提示：**前面我们提到修改程序到 boot.py 用于运行，但这里更推荐将脚本文件改成 main.py，然后发送 main.py 文件到 pyAI-K210，这样更符合 MicroPython 使用习惯！）

28.1.3.3 固件更新

当 pyAI-K210 上的固件意外丢失或者我们希望升级到较新版本固件时候，就需重新烧录固件。pyAI-K210 上面是一个 K210 单片机，所以这个操作相当于给 K210 MCU 重新烧录 MicroPython 固件。

MaixPy 官方提供了免安装的烧录工具，通过板载 USB 转串口烧录的。我们打开 MicroPython 开发套件配套资料\开发工具\Windows\固件更新工具\kflash_gui 目录下的 kflash_gui.exe 烧录软件。

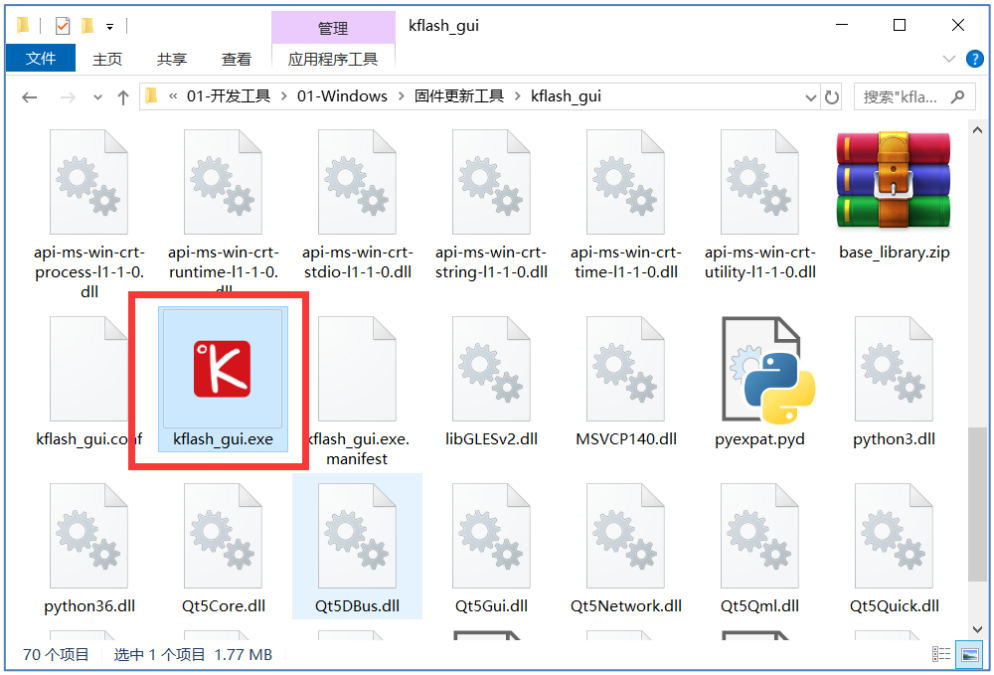


图 28-36 K210 固件烧录软件

打开软件后点击 open file:

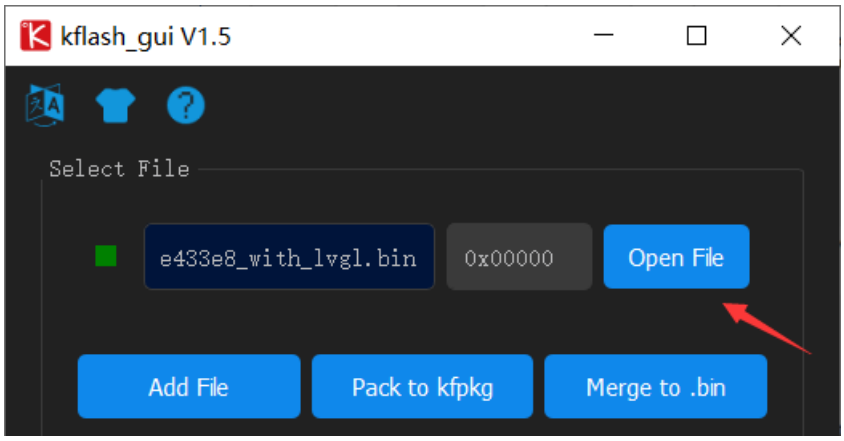


图 28-37

选择配套资料包路径_零一科技（01Studio）MicroPython 开发套件配套资料\03-相关固件\05-pyAI-K210_下的固件：

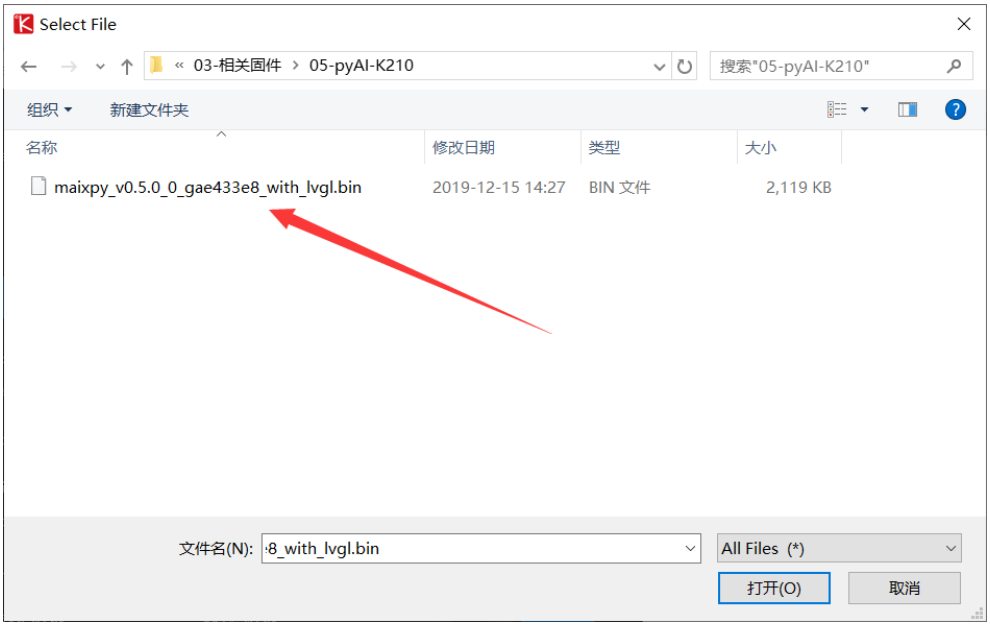


图 28-38 选择要更新的固件

烧录地址默认为 0x00000 即可。选择开发板和串口 COM，开发板可以选择跟 pyAI-K210 串口方案一样的 Maix Dock ，而串口则选择自己开发板对应的串口。

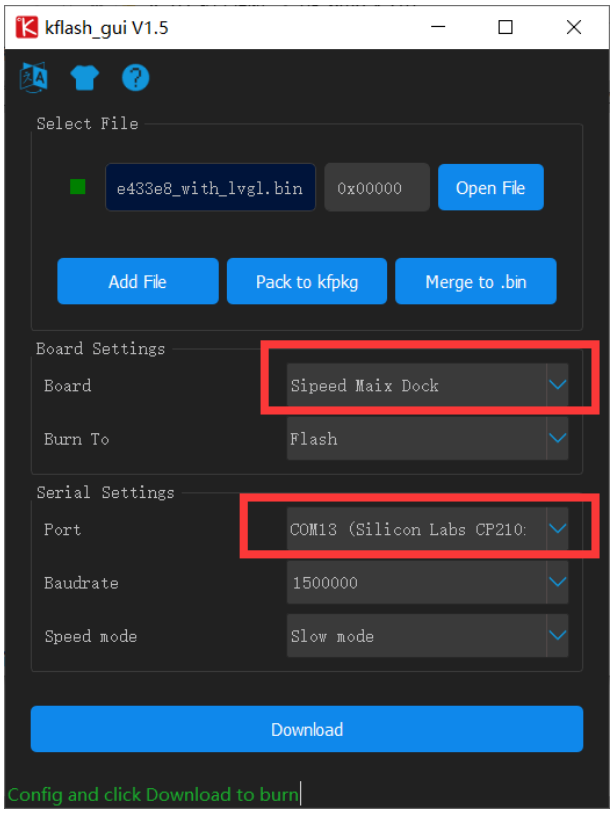


图 28-39 选择开发板和串口 COM

点击 Download 下载。

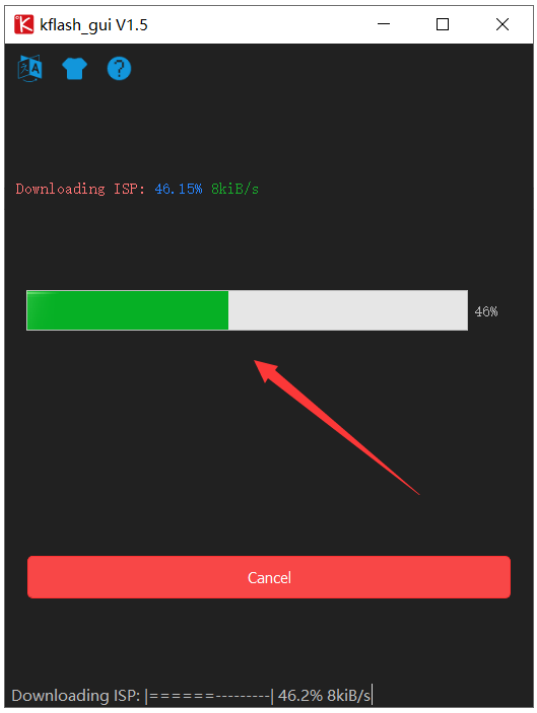


图 28-40 正在下载

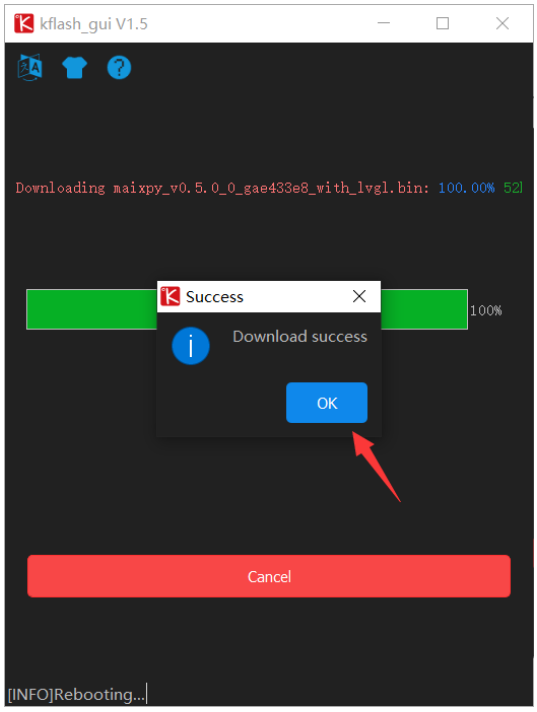


图 28-41 下载成功后弹出 success 对话框

28.2 基于 Mac OS

28.2.1 安装 python3

具体请参考：第二部分 3.2.1 安装 python3 这里不再重复。

28.2.2 安装开发软件 MaixPy IDE

MaixPy 拥有自己官方的 IDE，可以在官网下载，我们使用该 IDE 可以轻松进行开发。而且拥有 Windows、Mac OS、Linux 等版本。

当前版本为 v0.2.4，官网下载地址：（如有更新请下载最新版）
http://cn.dl.sipeed.com/MAIX/MaixPy/ide/_/v0.2.4，下载界面如下图。用户可以根据自己系统选择合适的版本下载安装。Mac OS 用户选择 dmg 格式文件下载安装。

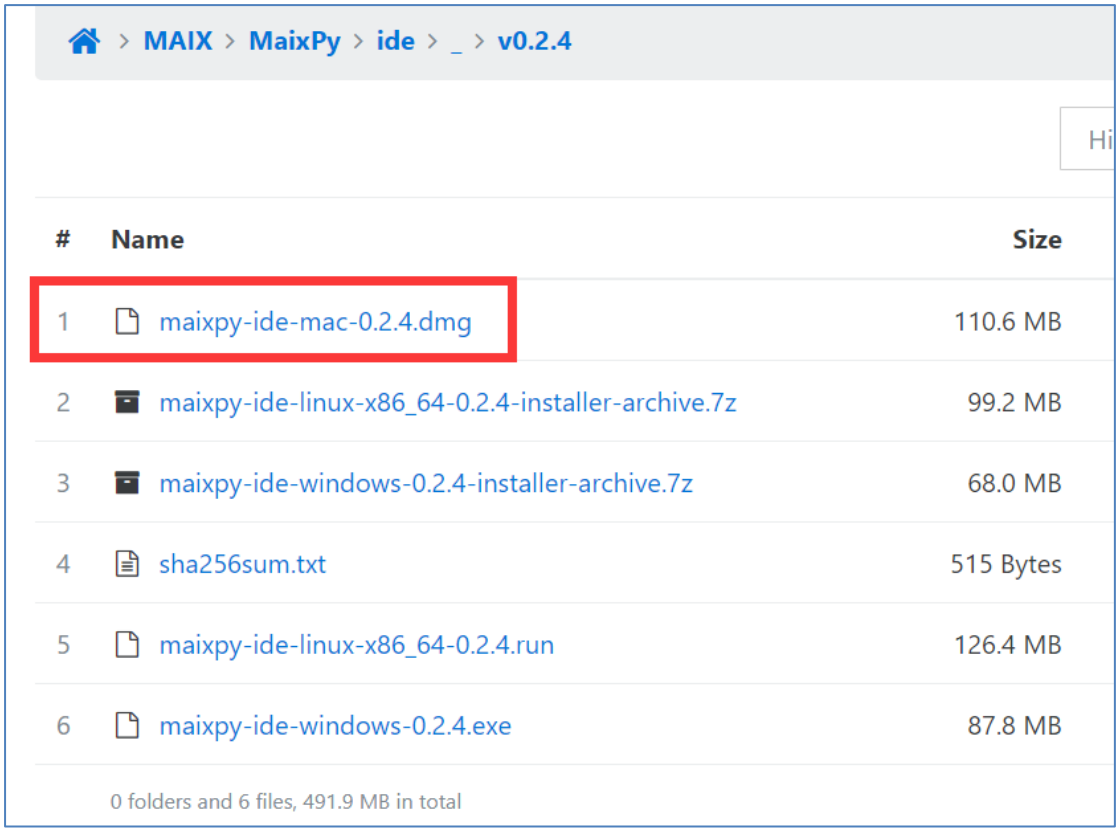


图 28-42 MaixPy IDE 下载界面

28.3 基于 Linux

28.3.1 安装 MaixPy IDE

MaixPy 拥有自己官方的 IDE，可以在官网下载，我们使用该 IDE 可以轻松进行开发。而且拥有 Windows、Mac OS、Linux 等版本。

当前版本为 v0.2.4，官网下载地址：（如有更新请下载最新版）

http://cn.dl.sipeed.com/MAIX/MaixPy/ide/_/v0.2.4，下载界面如下图。用户可以根据自己系统选择合适的版本下载安装。Linux 用户选择 run 格式文件下载安装。

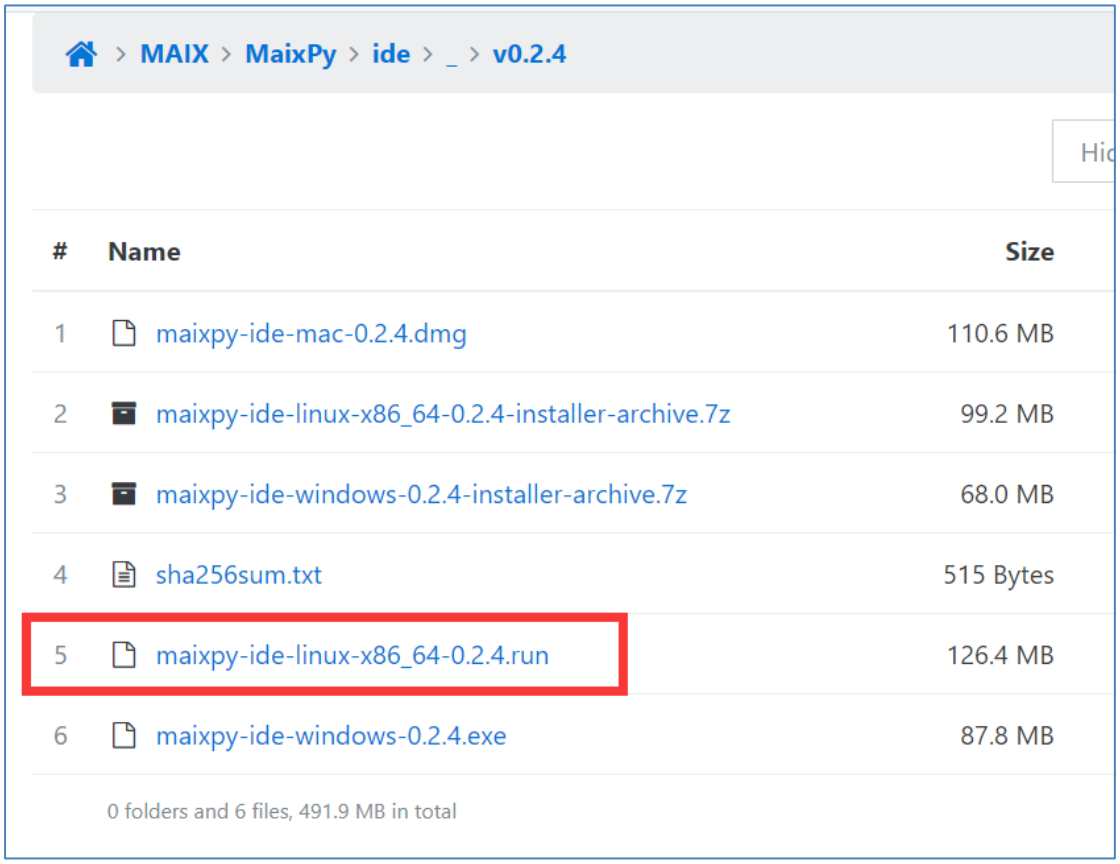


图 28-43 MaixPy IDE 下载界面

`Linux` 命令行给运行权限然后执行:

```
chmod +x maixpy-ide-linux-x86_64-0.2.2.run

./maixpy-ide-linux-x86_64-0.2.2.run
```

第29章　基础实验

MicroPython 更强调的是针对应用的学习，强大的底层库函数让我们可以直接关心功能的实现，也就是说我们只要理解和熟练相关的函数用法，就可以很好的玩转 MicroPython。它让我们可以做到不关心硬件和底层原理而直接跑起硬件（当然有兴趣和能力的小伙伴可以深入研究）。

基础实验是针对一些简单的实验学习讲解，可以让我们更快和更直接感受到 MicroPython 针对嵌入式开发的强大之处， 我后面的学习和开发夯实基础。

请先看实验讲解格式预览，每一节我们都会以以下形式讲解，图文并茂，务求达到快速理解和学习的作用：

（7） 前言：简单介绍这个实验；

（8） 实验平台：实验所用到的开发板、配件和接线说明；

（9） 实验目的：本实验要实现的功能；

（10） 实验讲解：对函数、代码、编程方法以及实验的详细讲解；

（11） 实验结果：记录程序下载到开发板上的图片示例；

（12） 总结:对实验进行总结。

29.1 点亮第一个 LED

- **前言：**

相信大部分人开始学习嵌入式单片机编程都会从点亮 LED 开始，基于 K210 平台的 MicroPython 的学习也不例外，通过点亮第一个 LED 能让你对编译环境和程序架构有一定的认识，为以后的学习和更大型的程序打下基础，增加信心。

- **实验平台：**

pyAI-K210。

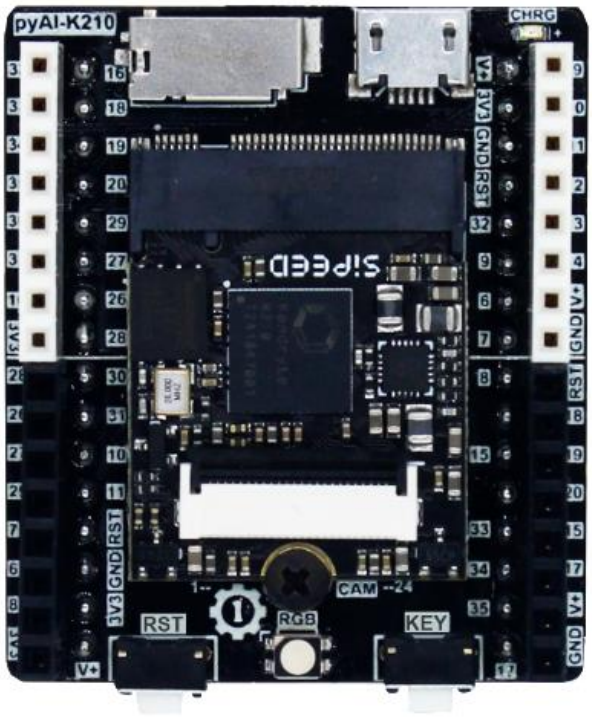


图 29-1 pyAI-K210 核心板

- **实验目的：**

学习 LED 的点亮，点亮 LED_B（蓝灯）。

- **实验讲解：**

pyAI-K210 上总共有 3 个 LED，位于 2 个按键中间，三色一体 LED。分别是 LED_B(蓝色)、LED_G(红色)、LED_R(红色)。

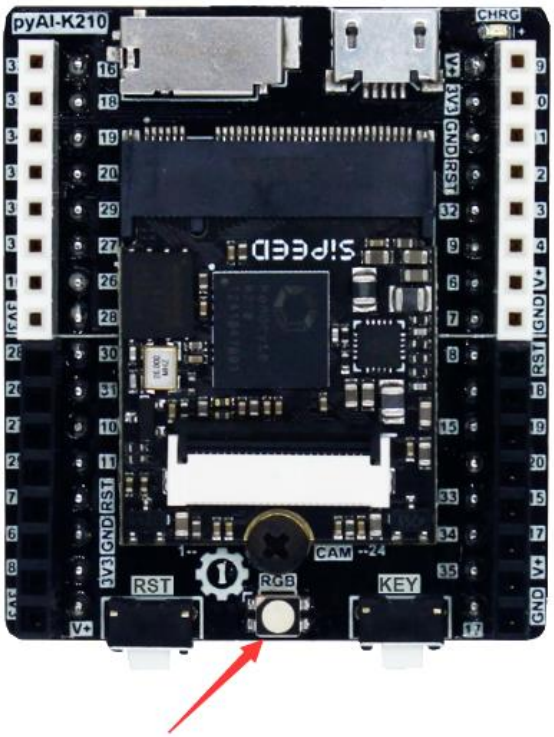


图 29-2 RGB 灯

其连接到 pyAI-K210 的外部 IO 引脚如下（可以看开发板原理图），LED 蓝灯对应的外部 IO 为 IO12，从电路可以看到当 IO12 为低电平时，蓝灯被点亮。

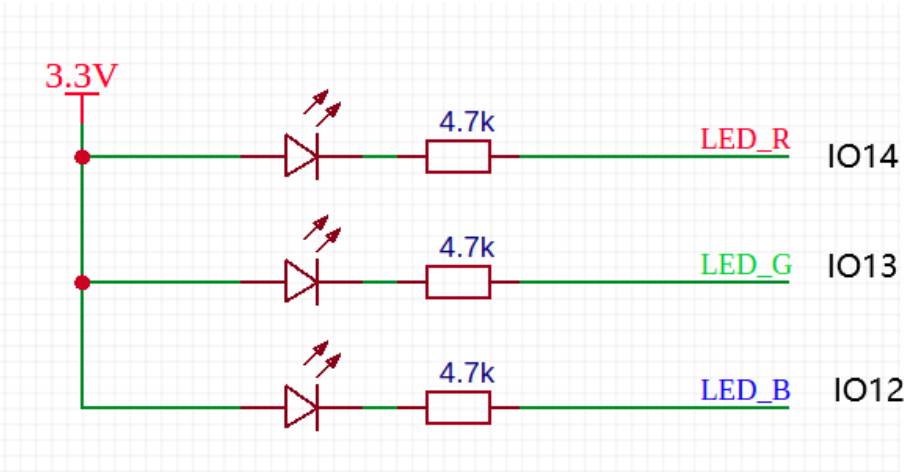


图 29-3 RGB LED 电路引脚图

K210 为外部 IO 和内部 IO，其片上外设（比如 GPIO、I2C 等）对应的引脚是可以任意设置的，而传统大部分 MCU 片上外设和引脚对应关系已经固定了， 只有部分引脚可以复用， 相比之下 K210 自由度更大。

因此我们在编程使用 GPIO 的时候需要注册一下硬件 IO 和 K210 内部 IO 的对应关系。注册方式使用 fpioa_manager：简称 fm，该模块用于注册芯片内部功能和引脚，帮助用户管理内部功能和引脚。

构造函数
fm.register(pin,function,force=False)
GPIO 注册函数；
使用方法
fm.register(pin,function,force=False)
【pin】芯片外部 IO 【function】芯片功能 【force】=True 则强制注册，清除之前的注册记录； 例：fm.register(12, fm.fpioa.GPIO0,force=True) 表示将外部 IO12 注册到内部 GPIO0
更多有关引脚和功能注册信息请看官方文档： https://maixpy.sipeed.com/zh/libs/builtin_py/fm.html

表 29-1 fm 引脚和功能注册模块

注册成功后我们就可以通过 GPIO 对象模块来控制外部 IO，从而控制 LED。GPIO 对象说明如下：

构造函数
GPIO(ID,MODE,PULL,VALUE)
GPIO 对象。 【ID】内部 GPIO 编号; 【MODE】GPIO 模式； GPIO.IN ： 输入模式 GPIO.OUT ： 输出模式 【PULL】 GPIO.PULL_UP ： 上拉

GPIO.PULL_DOWN ： 下拉 GPIO.PULL_NONE ： 无 【value】GPIO 初始化电平 1：高电平 0：低电平
使用方法
GPIO.value([value])
【value】GPIO 输出电平值； 1：高电平 0：低电平 *输入模式时候参数为空，表示获取当前 IO 输入电平值。

图 29-4 GPIO 对象输出功能

上表对 MicroPython 的 GPIO 对象做了详细的说明，GPIO 模块在 Maix 大模块下，而 fm 模块是在 fpioa_manager 大模块下面的其中一个小模块，在 python 编程里有两种方式引用相关模块:

方式 1 是：import Maix，然后通过 Maix.GPIO 来操作；

方式 2 是：from Maix import GPIO，意思是直接从 Maix 中引入 GPIO 模块，然后直接通过 GPIO 来操作。显然方式 2 会显得更直观和方便，本实验也是使用方式 2 来编程。代码编写流程如下：

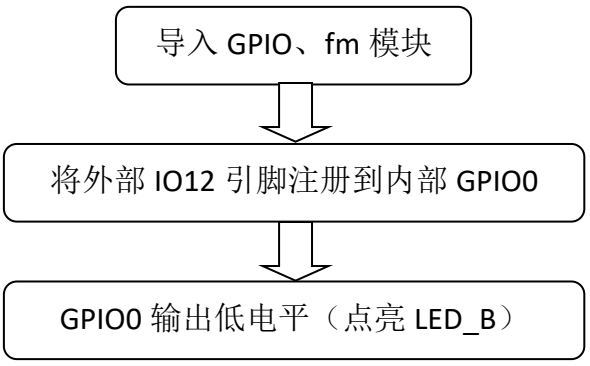


图 29-5 代码编写流程

参考代码如下：

```
'''
实验名称：点亮 LED_B 蓝灯
版本：v1.0
日期：2019.12
作者：01Studio
实验目的：学习 led 点亮。
'''
from Maix import GPIO
from fpioa_manager import fm

#将蓝灯引脚 IO12 配置到 GPIO0，K210 引脚支持任意配置
fm.register(12, fm.fpioa.GPIO0,force=True)

LED_B = GPIO(GPIO.GPIO0, GPIO.OUT) #构建 LED 对象
LED_B.value(0) #点亮 LED
```

- **实验结果：**

在 IDE 中运行上述代码，可以看到 LED_B 蓝灯被点亮。

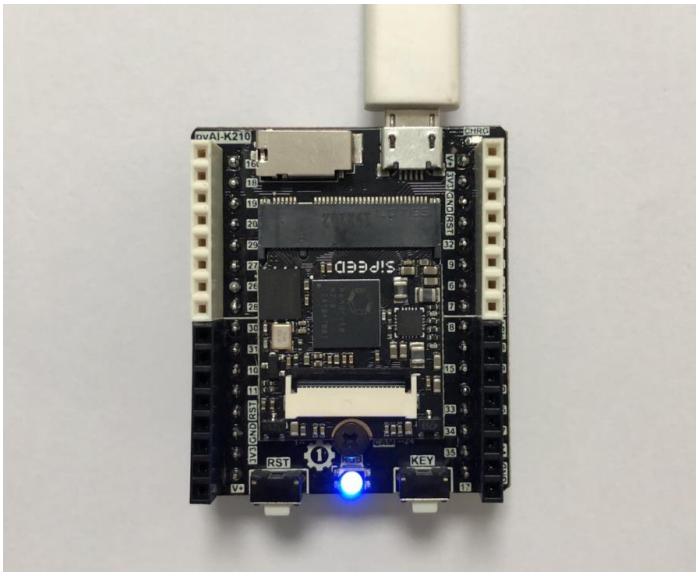


图 29-6 LED 蓝灯被点亮

- **总结：**

从第一个实验我们可以看到，使用 MicroPython 来开发关键是要学会构造函数和其使用方法，便可完成对相关对象的操作，在强大的模块函数支持下，实验只用了简单的两行代码便实现了点亮 LED 灯。

29.2 流水灯

- **前言：**

通过上一节点亮 LED 灯的学习，我们已经对 OpenMV4 使用 micropython 的编程有了初步的了解，这一节来做一个功能稍微复杂一点的实验，流水灯。流水灯也叫跑马灯，也就是让几个 LED 来回亮灭，达到好像流水的效果。也是单片机开发学习的典型例子。

- **实验平台：**

pyAI-K210。

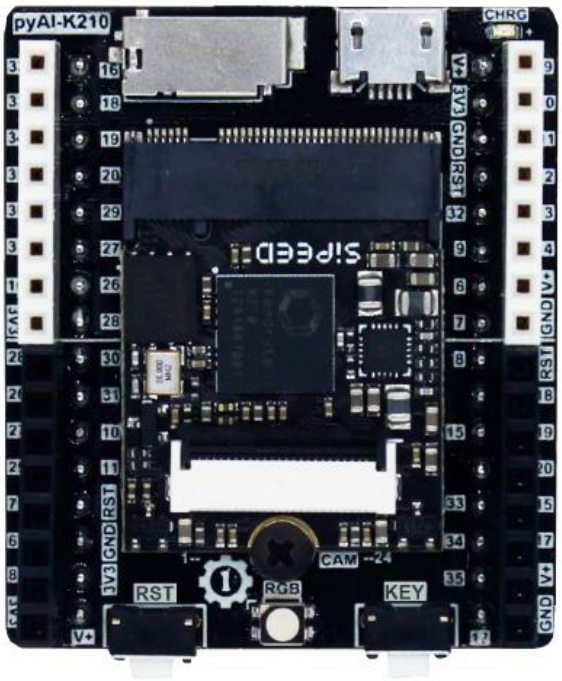


图 29-7

- **实验目的：**

流水灯。让 LED_B、LED_G、LED_R 循环亮灭，达到像流水一样的效果。

- **实验讲解：**

pyAI-K210 上总共有 3 个 LED，分别是 LED_B(蓝色)、LED_G(绿色)、LED_R(红色)；控制 LED 使用到 GPIO 对象。上一章节我们已经学习过 LED 点亮，这里要实现固定时间来亮灭，需要用到 utime 模块中的延时的函数。具体如下：

构造函数
utime()
时间模块，直接使用。
使用方法
utime.sleep(seconds)
秒级颜色。seconds:延时秒数
utime.sleep_ms(ms)
毫秒级延时。ms：延时毫秒数。
utime.sleep_us(us)
微秒级延时。us：延时微秒数。
*更多用法请阅读 MaixPy 官方文档： 文档链接：https://maixpy.sipeed.com/zh/libs/standard/utime.html

表 29-2 utime 时间对象

知道了延时函数的使用方法后，我们可以简单的梳理一下流程，首先导入 LED 和 utime 模块，程序开始先让 RGB LED 灭掉，开启循环，依次点亮每个 LED，延时 1 秒，关闭 LED。流程如下：

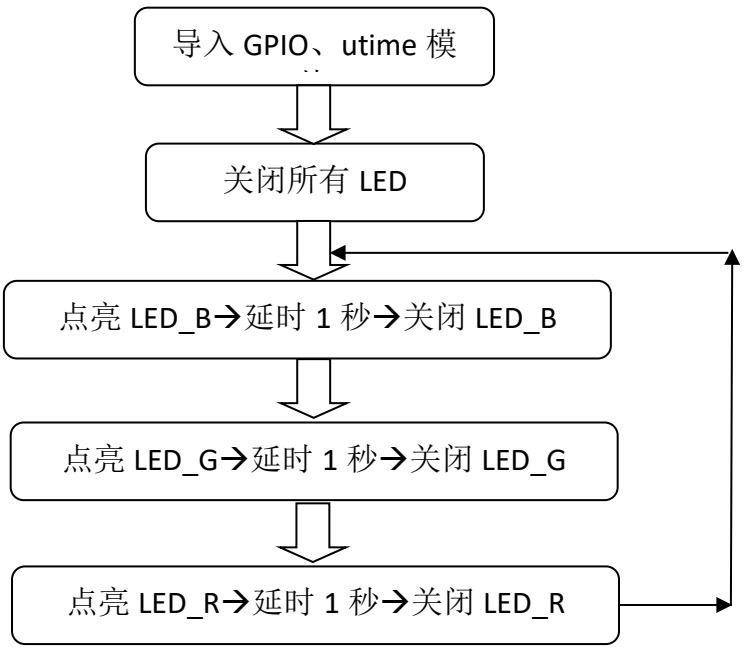


图 29-8 代码编写流程

参考例程代码如下：

```
'''
实验名称：流水灯
版本：v1.0
日期：2019.12
作者：01Studio
实验目的：让 RGB 灯循环闪烁。
'''
from Maix import GPIO
from fpioa_manager import fm
import utime

#将将 LED 外部 IO 注册到内部 GPIO，K210 引脚支持任意配置
fm.register(12, fm.fpioa.GPIO0)
fm.register(13, fm.fpioa.GPIO1)
fm.register(14, fm.fpioa.GPIO2)

LED_B = GPIO(GPIO.GPIO0, GPIO.OUT,value=1) #构建 LED 对象
LED_G = GPIO(GPIO.GPIO1, GPIO.OUT,value=1) #构建 LED 对象
LED_R = GPIO(GPIO.GPIO2, GPIO.OUT,value=1) #构建 LED 对象

while True:

    #蓝灯亮 1 秒
    LED_B.value(0) #点亮 LED
    utime.sleep(1)
    LED_B.value(1) #关闭 LED

    #绿灯亮 1 秒
    LED_G.value(0) #点亮 LED
```

```
    utime.sleep(1)
    LED_G.value(1) #关闭 LED

    #红灯亮 1 秒
    LED_R.value(0) #点亮 LED
    utime.sleep(1)
    LED_R.value(1) #关闭 LED
```

上述代码没错是完整地按照编程思路来编写，但可以见到有很多格式相似的地方，这显得代码非常冗余。我们可以通过 for 函数来编写程序，由于是对 3 个 LED 的操作，因此我们可以用 for i in range(0,3): 语句来修改，参考代码如下：

```
'''
实验名称：流水灯
版本：v1.0
日期：2019.12
作者：01Studio
实验目的：让 RGB 灯循环闪烁。
'''
from Maix import GPIO
from fpioa_manager import fm
import utime

#将将 LED 外部 IO 注册到内部 GPIO，K210 引脚支持任意配置
fm.register(12, fm.fpioa.GPIO0)
fm.register(13, fm.fpioa.GPIO1)
fm.register(14, fm.fpioa.GPIO2)

#构建 LED 对象，并初始化输出高电平，关闭 LED
```

```
LED_B = GPIO(GPIO.GPIO0, GPIO.OUT,value=1)
LED_G = GPIO(GPIO.GPIO1, GPIO.OUT,value=1)
LED_R = GPIO(GPIO.GPIO2, GPIO.OUT,value=1)

#定义数组方便循环语句调用
LED=[LED_B, LED_G, LED_R]

while True:

    for i in range(0,3):

        LED[i].value(0) #点亮 LED
        utime.sleep(1)
        LED[i].value(1) #关闭 LED
```

- **实验结果：**

在上述代码在 IDE 中运行，或者拷贝到 pyAI-K210 文件系统，可以看到 RGB LED 交替闪烁。

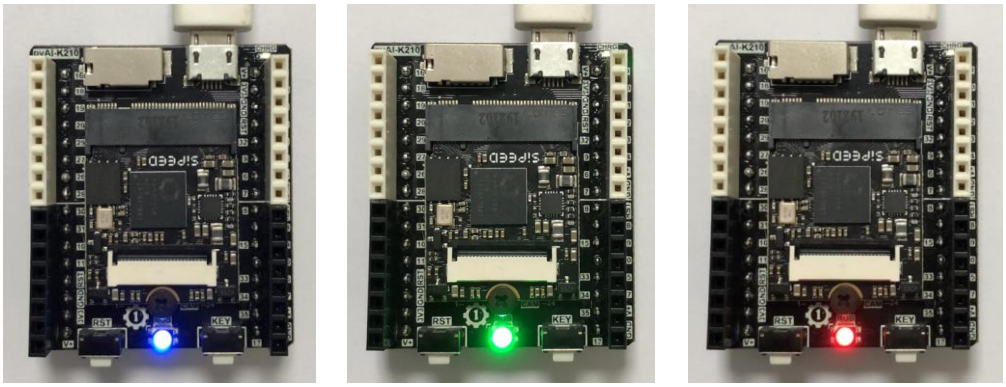


图 29-9 RGB 灯循环闪烁

- **总结：**

本节实验相对于第一节相比增加了延时函数的应用，可见 MicroPython 编程功能增加就是各类函数的叠加，当开发复杂功能时候，因为底层函数已经封装好，所以面向应用的开发使用起来非常方便。除此之外我们也体验了 Python 代码的简洁和高效，使得程序更好的阅读。

29.3 按键

- **前言：**

按键是最简单也最常见的输入设备，很多产品都离不开按键，包括早期的 iphone，今天我们就来学习一下如何使用 MicroPython 来编写按键程序。有了按键输入功能，我们就可以做很多好玩的东西了。

- **实验平台：**

pyAI-K210。

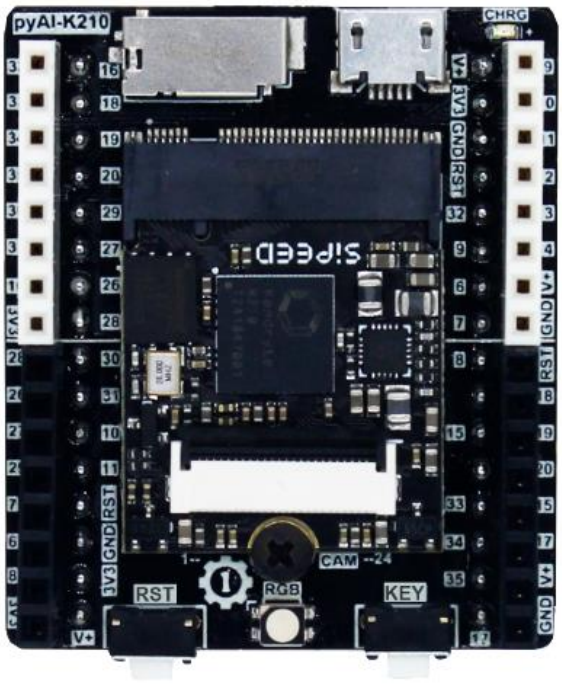


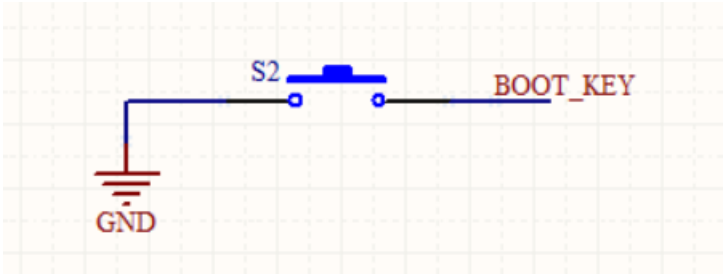
图 29-10

- **实验目的：**

学习 MicroPython 编程实现按键输入检测。

- **实验讲解：**

我们先来看看 pyAI-K210 的原理图，看看按键对应的外部 IO 引脚。



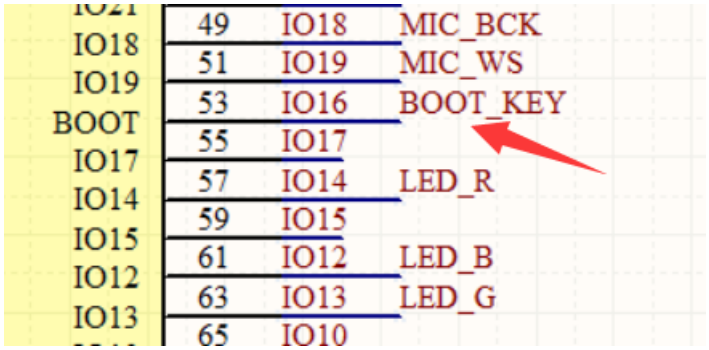


图 29-11 KEY 引脚图

从原理图可以看到，按键 KEY 的一端连接到 K210 的外部 IO16，另一端连接到 GND。所以按键在没按下时候输入高电平（1），按下时候输入低电平（0）。

和 LED 一样，按键的输入检测也是用到 GPIO 对象模块，具体如下：

构造函数
GPIO(ID,MODE,PULL,VALUE)
GPIO 对象。 【ID】内部 GPIO 编号; 【MODE】GPIO 模式; GPIO.IN ： 输入模式 GPIO.OUT ： 输出模式 【PULL】 GPIO.PULL_UP ： 上拉 GPIO.PULL_DOWN ： 下拉 GPIO.PULL_NONE ： 无 【value】GPIO 初始化电平 1：高电平 0：低电平

使用方法
GPIO.value([value])
【value】GPIO 输出电平值； 1：高电平 0：低电平 *输入模式时候参数为空，表示获取当前 IO 输入电平值。

表 29-3　GPIO 对象

GPIO 对象使用非常简单，我们将按键即外部“IO16”引脚配置成输入，实现当检测到按键被按下时候点亮 LED 蓝灯，松开时关闭 LED 蓝灯来做指示。代码编写流程如下：

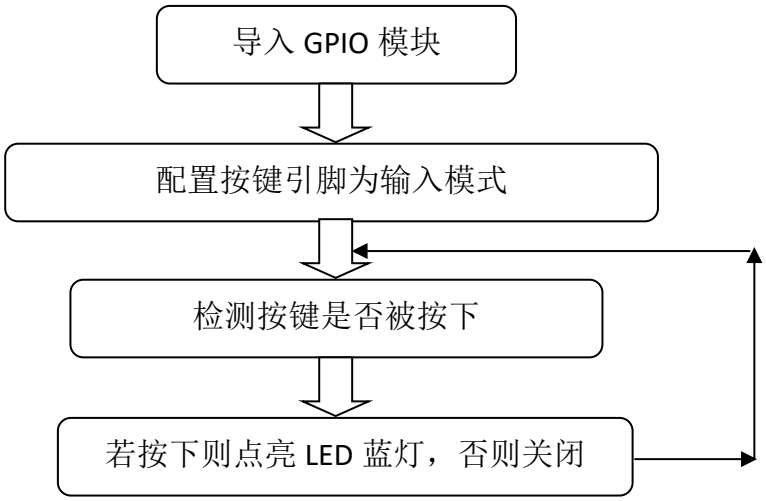


图 29-12 代码编写流程

参考代码如下：

```
'''
实验名称：  按键
版本：  v1.0
日期：  2019.12
作者：  01Studio
社区：  www.01studio.org
'''
```

```
from Maix import GPIO
from fpioa_manager import fm

#注册 IO，蓝灯-->IO12,KEY-->IO16
fm.register(12, fm.fpioa.GPIO0)
fm.register(16, fm.fpioa.GPIO1)

#初始化 IO
LED_B = GPIO(GPIO.GPIO0, GPIO.OUT)
KEY = GPIO(GPIO.GPIO1, GPIO.IN)

while True:

    if KEY.value()==0: #按键被按下接地
        LED_B.value(0) #点亮 LED_B,蓝灯
    else:
        LED_B.value(1) #熄灭 LED
```

- **实验结果：**

运行代码，可以看到当按键 KEY 被按下时候，LED_B 蓝灯点亮，松开时熄灭。按键可以按下 pyAI-K210 底部的按键或者 pyBase 开发底板上的 KEY 按键。

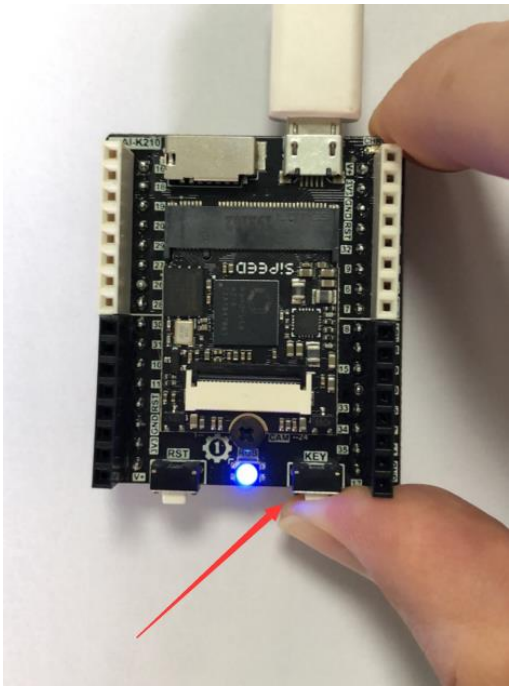


图 29-13 按键实验

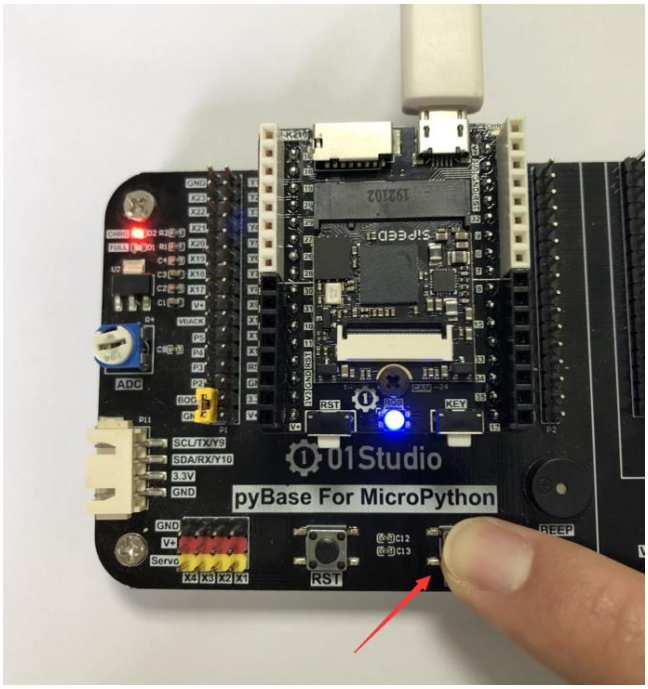


图 29-14 pyBase 上的按键

- **总结：**

GPIO 是非常通用的功能，学会了 GPIO，就可以把开发板所有的引脚为自己所用，灵活性很强。

29.4 外部中断

- **前言：**

前面我们在做普通的按键（GPIO）时候，虽然能实现 IO 口输入输出功能，但代码是一直在检测 IO 输入口的变化，因此效率不高，特别是在一些特定的场合，比如某个按键，可能 1 天才按下一次去执行相关功能，这样我们就浪费大量时间来实时检测按键的情况。

为了解决这样的问题，我们引入外部中断概念，顾名思义，就是当按键被按下(产生中断)时，我们才去执行相关功能。这大大节省了 CPU 的资源，因此中断的在实际项目的应用非常普遍。

- **实验平台：**

pyAI-K210。

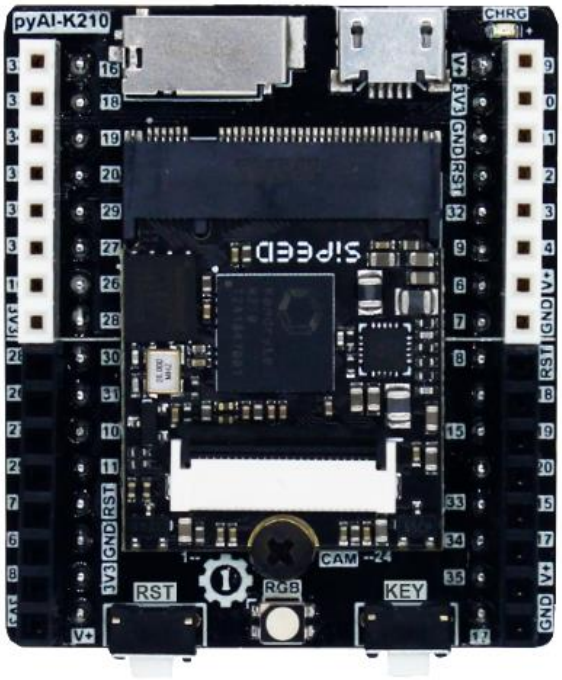


图 29-15

- **实验目的：**

利用中断方式来检查按键 KEY 状态，按键被按下（产生外部中断）后使 LED_B 蓝灯的亮灭状态反转。

- **实验讲解：**

外部中断也是通过 GPIO 模块来配置，我们先来看看其配构造函数和使用方法：

构造函数
GPIO(ID,MODE,PULL,VALUE)
GPIO 对象。 【ID】内部 GPIO 编号; 【MODE】GPIO 模式; GPIO.IN ： 输入模式 GPIO.OUT ： 输出模式 【PULL】 GPIO.PULL_UP ： 上拉 GPIO.PULL_DOWN ： 下拉 GPIO.PULL_NONE ： 无 【value】GPIO 初始化电平 1：高电平 0：低电平
使用方法
GPIO.irq(CALLBACK_FUNC,TRIGGER_CONDITION)
配置中断。 【CALLBACK_FUNC】中断执行的回调函数; 【TRIGGER_CONDITION】中断触发方式; GPIO.IRQ_RISING：上升沿触发 GPIO.IRQ_FALLING：下降沿沿触发 GPIO.IRQ_BOTH：都触发
GPIO.disirq()
关闭中断。

表 29-4 GPIO 外部中断对象

我们先来了解一下上升沿和下降沿的概念，由于按键 KEY 引脚是通过按键接到 GND，也就是我们所说的低电平“0”，所以当按键被按下再松开时，引脚先获得下降沿，再获得上升沿，如下图所示：

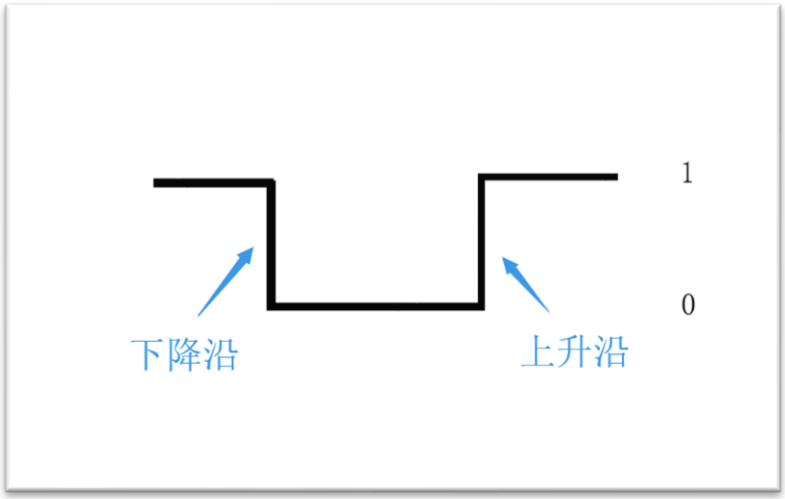


图 29-16 上升沿和下降沿

按键被按下时候可能会发生抖动，抖动如下图，有可能造成误判，因此我们需要使用延时函数来进行消抖：

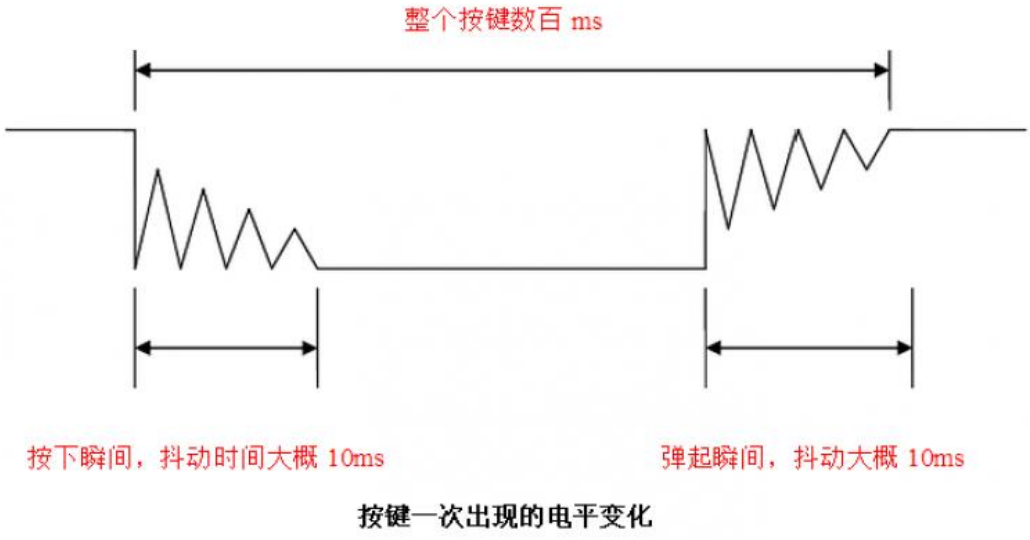


图 29-17 按键按下过程

我们可以选择下降沿方式触发外部中断，也就是当按键被按下的时候立即产生中断。

需要注意的是 K210 只有高速 GPIO 才有外部中断，GPIO 常量表如下：

K210 - GPIO
普通 GPIO
GPIO0 - GPIO7
高速 GPIO
GPIOHS0 – GPIOHS31

表 29-5 GPIO 表

编程思路中断跟 GPIO 按键章节类似，在初始化中断后，当系统检测到外部中断时候，执行 LED 状态反转的代码即可。流程图如下：

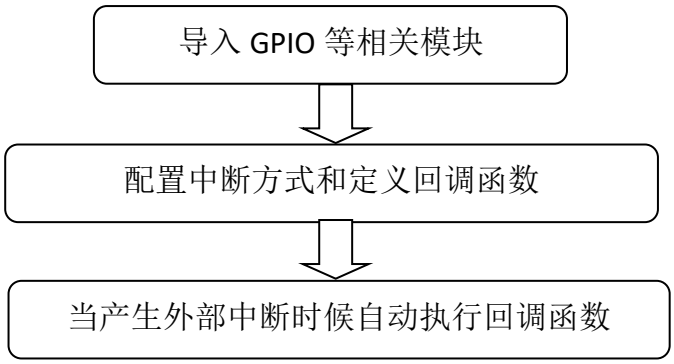


图 29-18 代码编写流程

参考代码如下：

```
'''
实验名称：外部中断
版本： v1.0
日期： 2019.12
作者： 01Studio
说明：通过按键改变 LED 的亮灭状态（外部中断方式）
'''

from Maix import GPIO
from fpioa_manager import fm
import utime
```

```
#注册 IO，注意高速 GPIO 口才有中断
fm.register(12, fm.fpioa.GPIO0)
fm.register(16, fm.fpioa.GPIOHS0)

#构建 lED 和 KEY 对象
LED_B=GPIO(GPIO.GPIO0,GPIO.OUT,value=1)
KEY=GPIO(GPIO.GPIOHS0, GPIO.IN, GPIO.PULL_UP)

#LED 状态表示
state = 1

#中断回调函数
def fun(KEY):
    global state
    utime.sleep_ms(10) #消除抖动
    if KEY.value()==0: #确认按键被按下
        state = not state
        LED_B.value(state)

#开启中断，下降沿触发
KEY.irq(fun, GPIO.IRQ_FALLING)
```

- **实验结果：**

可以看到每次按下按键 KEY 时候，LED_B 蓝灯的亮灭状态翻转。按键可以使用 pyAI-K210 底部的按键或者 pyBase 开发底板上的 KEY 按键。

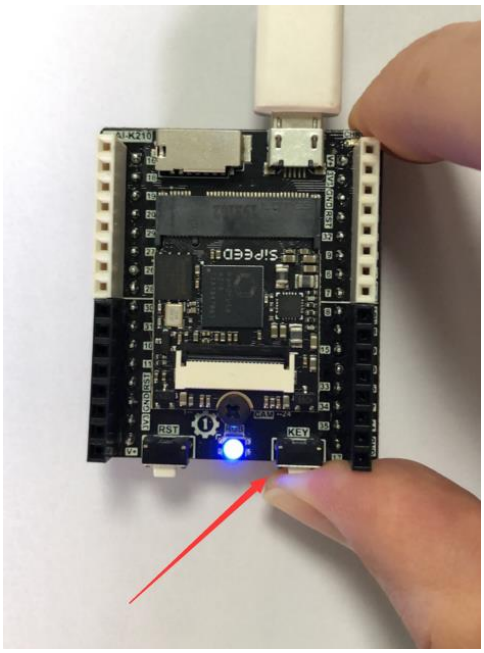


图 29-19 外部中断实验

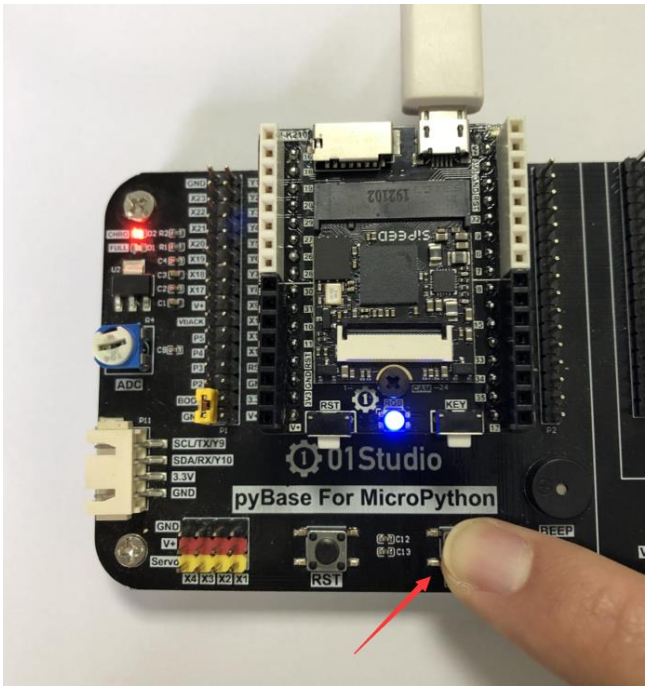


图 29-20 pyBase 上的按键 KEY

- **总结：**

从参考代码来看，只是用了几行代码就实现了实验功能，而且相对于使用 while True 实时检测函数来看，代码的效率大大增强。外部中断的应用非常广，除了普通的按键输入和电平检测外，很大一部分输入设备，比如传感器也是通过外部中断方式来实时检测，从而提供 CPU 的利用率。

29.5 定时器

- **前言：**

定时器，顾名思义就是用来计时的，我们常常会设定计时或闹钟，然后时间到了就告诉我们要做什么了。单片机也是这样，通过定时器可以完成各种预设好的任务。

- **实验平台：**

pyAI-K210。

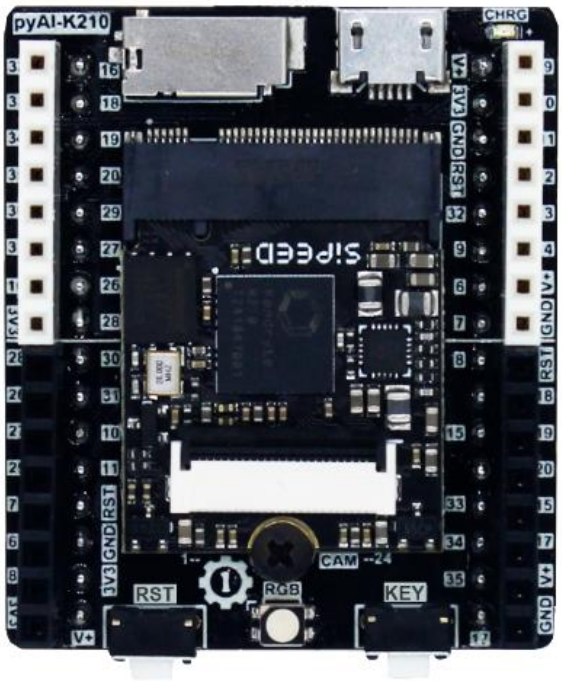


图 29-21

- **实验目的：**

通过定时器让 LED_B 蓝灯周期性每秒闪烁 1 次。

- **实验讲解：**

定时器是在 machine 的 Timer 模块。通过 MicroPython 可以轻松编程使用。我们也是只需要了解其构造对象函数和使用方法即可！

构造函数
machine.Timer(id,channel,mode=Timer.MODE_ONE_SHOT,period=1000, unit=Timer.UNIT_MS, callback=None, arg=None, start=True, priority=1, div=0)
定时器对象 Timer 对象在 machine 模块下。 【id】定时器编号, [Timer.TIMER0~TIMER2] 定时器 0-2; 【channel】Timer 通道，[Timer.CHANNEL0~Timer.CHANNEL3] 【mode】定时器模式 MODE_ONE_SHOT: 一次性 MODE_PERIODIC: 周期性 MODE_PWM 【period】定时器为周期性模块时每个周期时间值 【unit】周期的单位 Timer.UNIT_S：秒 Timer.UNIT_MS：毫秒 Timer.UNIT_US：微妙 Timer.UNIT_NS：纳秒 【callback】定时器中断执行的回调函数；**注意：回调函数是在中断中调用的，所以在回调函数中请不要占用太长时间以及做动态内存分配开关中断等动作。** 【arg】回调函数第 2 个参数 【start】是否在构建对象后立即开始定时器， =True: 立即开始； =False: 不立即开始，需要调用 start()来开启。 【priority】硬件中断优先级，在 K210 中，取值范围是[1,7],值越小优先级越高 【div】硬件分频器。
使用方法
Timer.callback(fun)
定义回调函数。
Timer.period([value])

配置周期。
Timer.start()
启动定时器。
Timer.stop()
停止定时器。
Timer.deinit()
注销定时器。
*更多用法请阅读 MaixPy 官方文档： 文档链接：https://maixpy.sipeed.com/zh/libs/machine/timer.html

表 29-6 Timer 定时器对象

定时器到了预设指定时间后，也会产生中断，因此跟外部中断的编程方式类似，代码编程流程图如下：

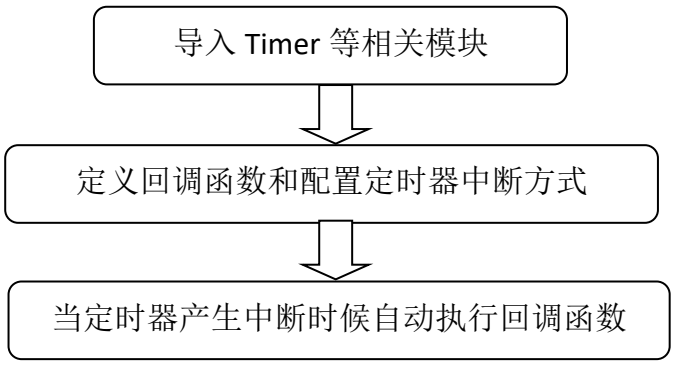


图 29-22 代码编写流程

参考代码如下：

```
'''
实验名称：定时器
版本： v1.0
日期： 2019.12
作者： 01Studio
说明：通过定时器让 LED 周期性每秒闪烁 1 次
```

```
'''
from Maix import GPIO
from fpioa_manager import fm
from machine import Timer

#注册 IO 和构建 LED 对象
fm.register(12, fm.fpioa.GPIO0)
LED_B = GPIO(GPIO.GPIO0, GPIO.OUT)

#计数变量
Counter=0

#定时器回调函数
def fun(tim):
    global Counter
    Counter = Counter + 1
    print(Counter)
    LED_B.value(Counter%2)#LED 循环亮灭。

#定时器 0 初始化，周期 1 秒
tim = Timer(Timer.TIMER0, Timer.CHANNEL0, mode=Timer.MODE_PERIODIC,
             period=1000, callback=fun)
```

- **实验结果：**

  运行程序，可以看到 LED_B 蓝灯每隔 1 秒闪烁 1 次。

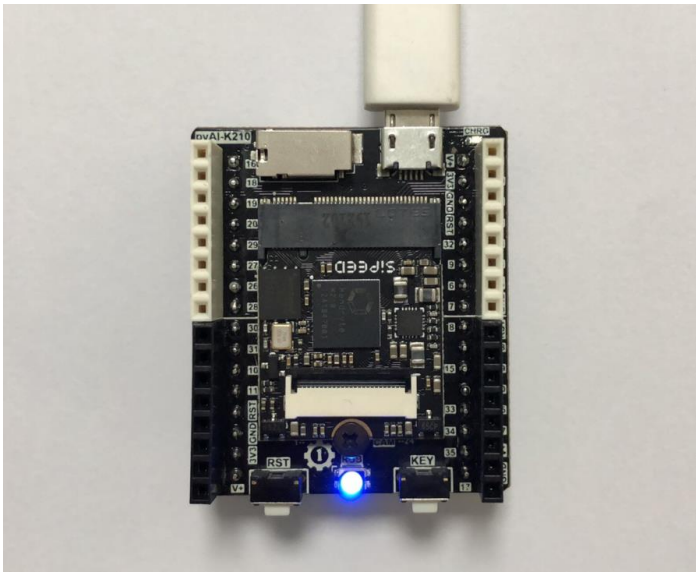


图 29-23　LED 定时闪烁

- **总结：**

本节实验介绍了定时器的使用方式，有用户可能会认为使用延时延时也可以实现这个功能，但相比于延时函数，定时器的好处就是不占用过多的 CPU 资源。

有兴趣的用户也可以多定义几个定时器对象 tim2,tim3，通过不同的参数配置实现多任务的操作。

29.6 PWM

- **前言：**

PWM（脉冲宽度调制）就是一个特定信号输出，主要用于输出不同频率、占空比（一个周期内高电平出现时间占总时间比例）的方波。以实现固定频率或平均电压输出。

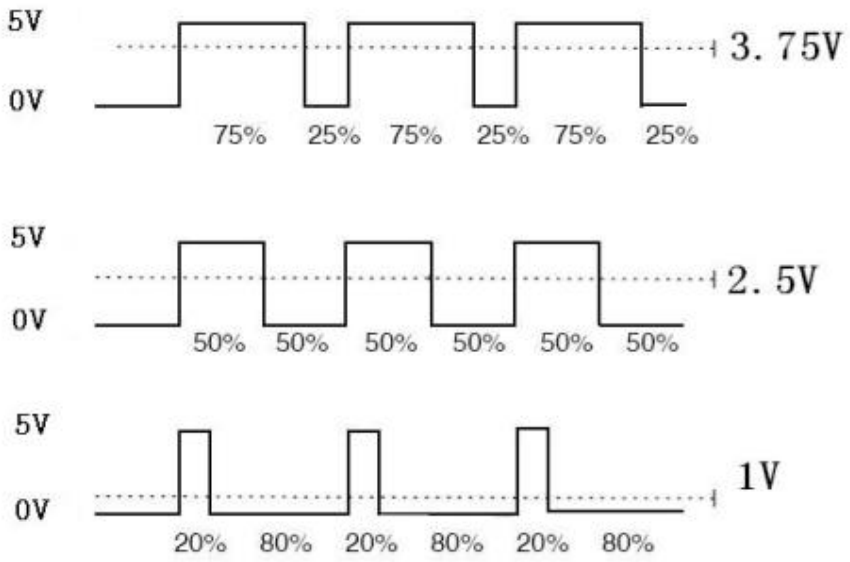


图 29-24 PWM 波形示例（右边为等效电压）

- **实验平台：**

pyAI-K210 开发套件。

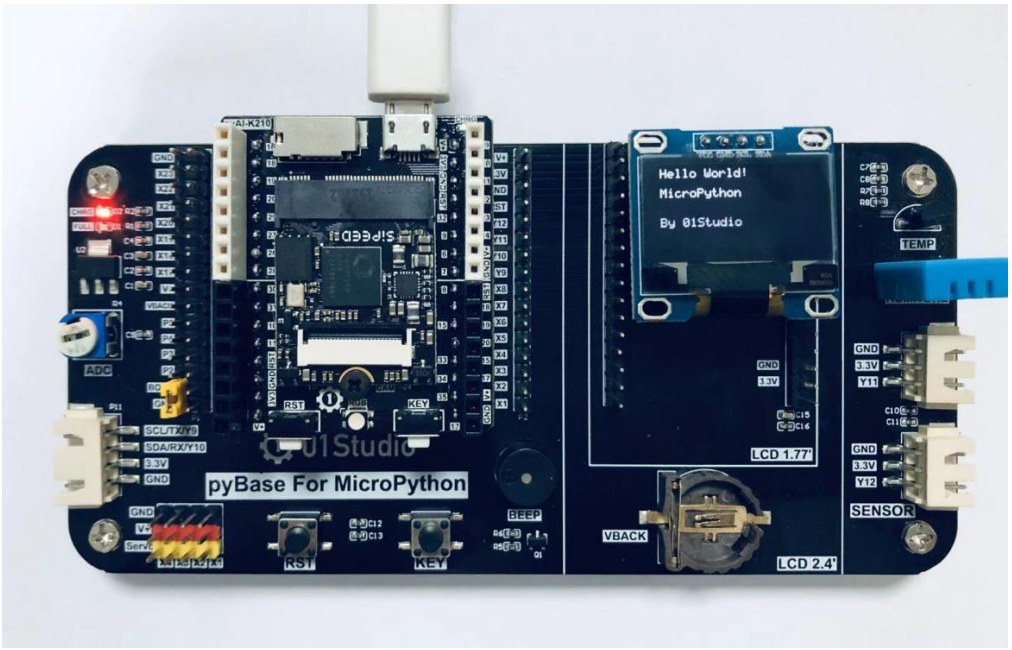


图 29-25

- **实验目的：**

通过不同频率的 PWM 信号输出，驱动无源蜂鸣器发出不同频率的声音。

- **实验讲解：**

蜂鸣器分有源蜂鸣器和无源蜂鸣器，有源蜂鸣器的使用方式非常简单，只需要接上电源，蜂鸣器就发声，断开电源就停止发声。而本实验用到的无源蜂鸣器，是需要给定指定的频率，才能发声的，而且可以通过改变频率来改变蜂鸣器的发声音色，以此来判定 pyAI-K210 的 PWM 输出频率是在变化的。

pyBase 开发底板上的无源蜂鸣器连接到引脚 X5。如下图所示：

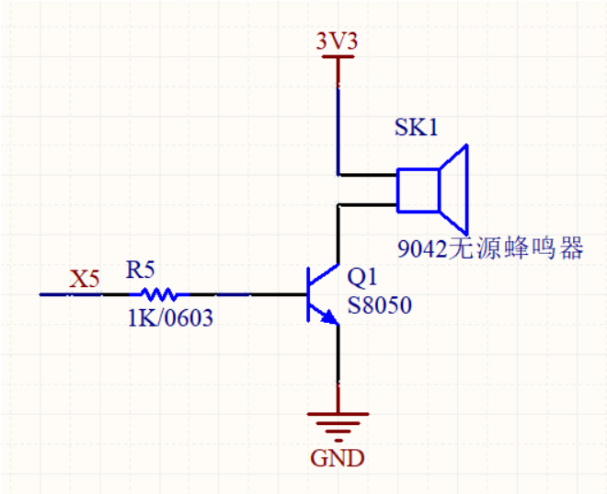


图 29-26 蜂鸣器原理图

而 pyAI-K210 并没有引脚直接连接到 pyBase 的 X5（主要避免影响 IO 复用。）而 IO15 连接到 pyBase 开发底板的 X6 引脚，因此我们可以用跳线帽或者跳线来连接 pyBase 的 X5 和 X6 引脚。相当于将无源蜂鸣器接到 pyAI-K210 的外部 IO15 引脚。

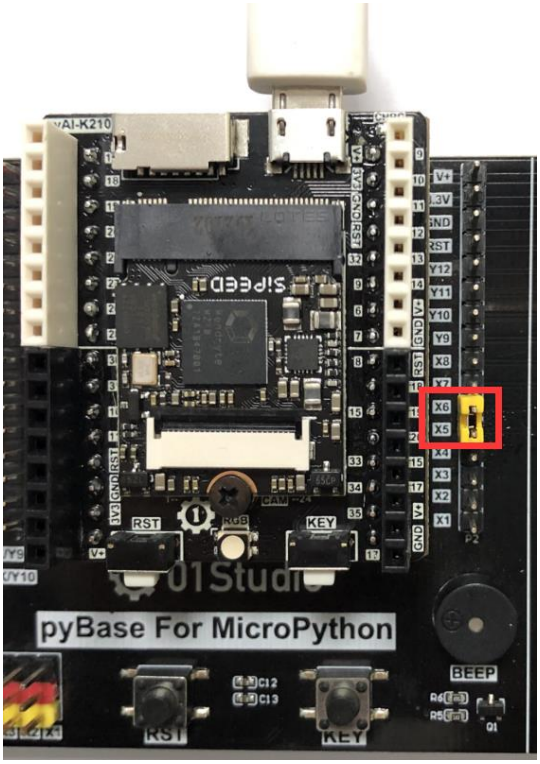


图 29-27 使用跳线帽将 X6 和 X5 短接

我们来看看 PWM 模块对象：

构造函数
machine.PWM(tim, freq, duty, pin, enable=True)
PWM 对象在 machine 模块下。 【tim】K210 的 PWM 依赖于定时器来产生波形 【freq】PWM 频率 【duty】PWM 占空比 【pin】PWM 输出引脚 【enable】是否在构建对象后立即产生波形，默认 True。
使用方法
PWM.freq(freq)
设置频率。不传参数返回当前频率值。
PWM.duty(duty)
设置占空比。不传参数返回当前占空比值。[0-100]表示占空比百分比

PWM.enable()
使能 PWM 输出。
PWM.disable()
暂停 PWM 输出。
PWM.deinit()
注销 PWM。

表 29-7　PWM 对象

无源蜂鸣器我们可以用特定频率的方波来驱动，方波的原理很简单，就是一定频率的高低电平转换，可以简单理解成占空比为 50%的 PWM 输出。

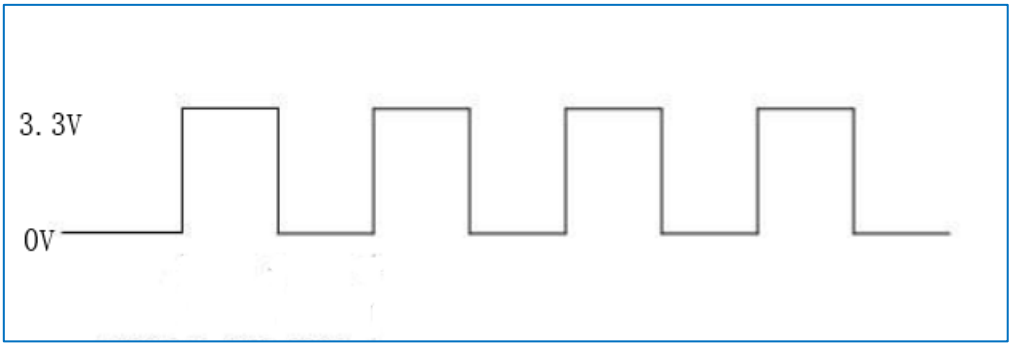


图 29-28　方波信号

结合上述讲解，总结出代码编写流程图如下：

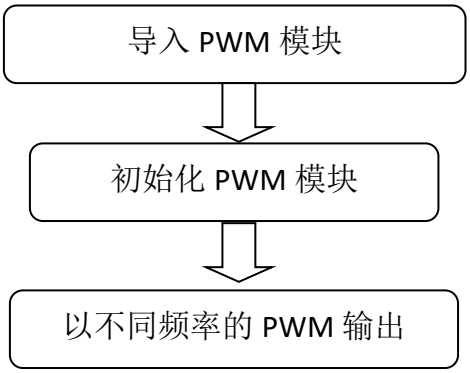


图 29-29 代码编写流程图

实验参考代码：

```
'''
实验名称： PWM
版本： v1.0
日期： 2019.12
作者： 01Studio
说明：通过不同频率的 PWM 信号输出，驱动无源蜂鸣器发出不同频率的声音。
'''
from machine import Timer,PWM
import time

#PWM 通过定时器配置，接到 IO15 引脚
tim = Timer(Timer.TIMER0, Timer.CHANNEL0, mode=Timer.MODE_PWM)
beep = PWM(tim, freq=1, duty=50, pin=15)

#循环发出不同频率响声。
while True:
    beep.freq(200)
    time.sleep(1)

    beep.freq(400)
    time.sleep(1)

    beep.freq(600)
    time.sleep(1)

    beep.freq(800)
    time.sleep(1)

    beep.freq(1000)
    time.sleep(1)
```

- **实验结果：**

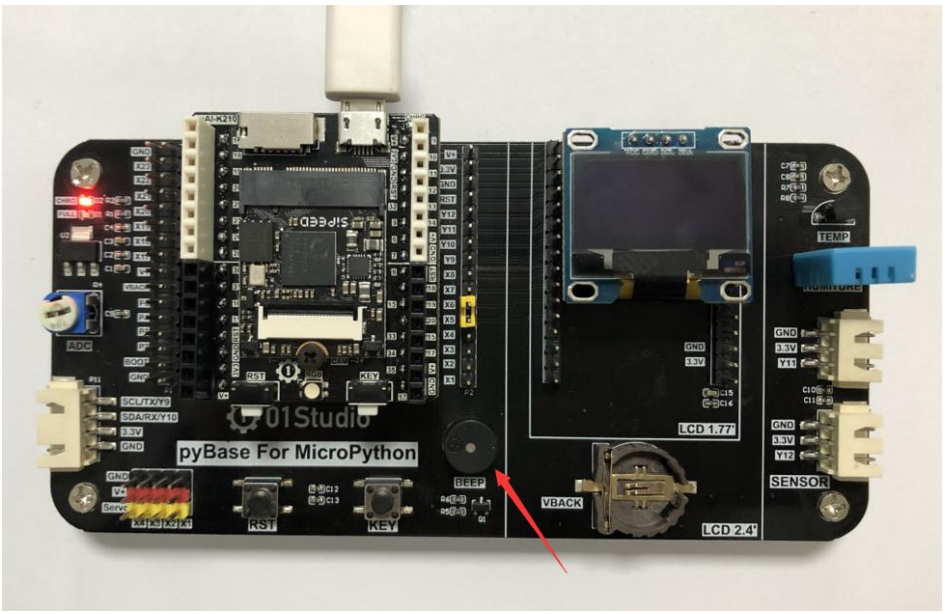


图 29-30

有条件的朋友可以使用示波器测量 pyAI-K210 的 IO15 引脚或 pyBase 的 X5 引脚，观察信号波形的变化：

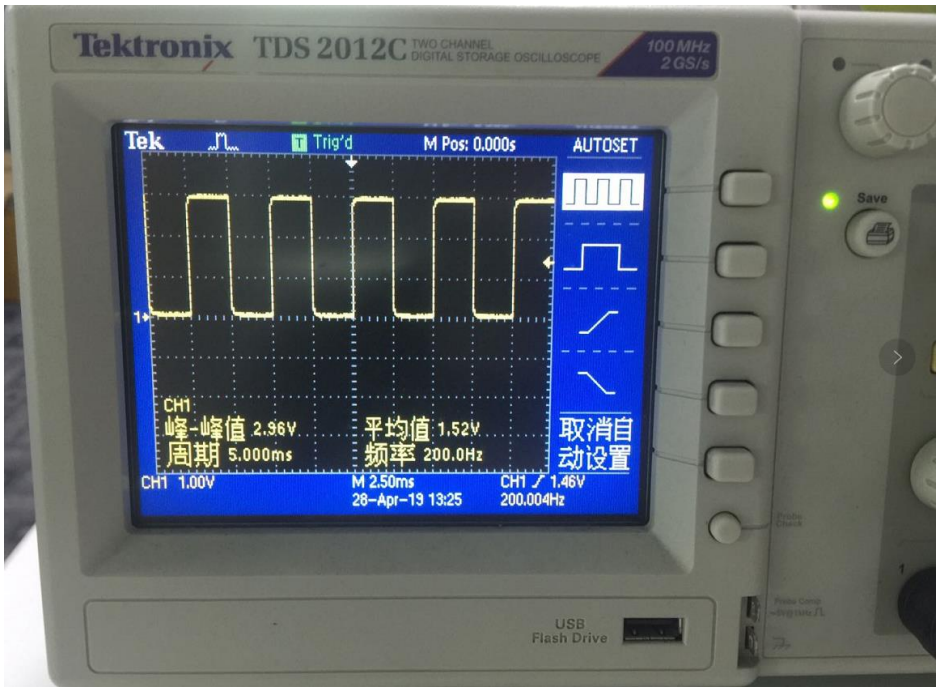


图 29-31　示波器显示波形

- **总结：**

到了这一节，我们发现实验中对象函数使用方法非常简单，这是好事，让我们可以将更多精力放在应用上，做出更多好玩的创意。而不需要过多的关注复杂的底层代码开发。而随着要实现功能的复杂化让编程的代码数量变多，逻辑也将略显复杂。

29.7 I2C 总线（OLED 显示屏）

- **前言：**

前面学习了按键输入设备后，这一节我们来学习输出设备 OLED 显示屏，其实之前的 LED 灯也算是输出设备，因为它们确切地告诉了我们硬件的状态。只是相对于只有亮灭的 LED 而言，显示屏可以显示更多的信息，体验更好。

本章节的 OLED 显示屏学习，实际上是在使用 I2C 的总线接口，pyAI-K210 是通过 I2C 总线与 OLED 显示屏通讯的。这章稍微复杂一点，但我们把它放在了前面来学习，是因为学会了显示屏的使用，那么在后面的实验中可玩性就更强了。

- **实验平台：**

pyAI-K210 开发套件。

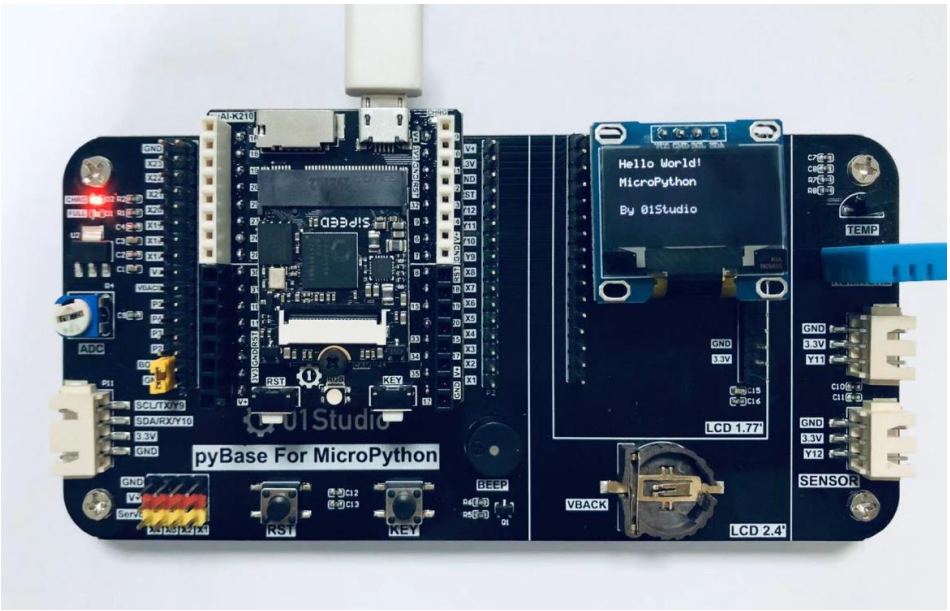


图 29-32

- **实验目的：**

学习使用 MicroPython 的 I2C 总线通讯编程和 OLED 显示屏的使用。

- **实验讲解：**

什么是 I2C？

I2C 是用于设备之间通信的双线协议，在物理层面，它由 2 条线组成：SCL 和 SDA，分别是时钟线和数据线。也就是说不通设备间通过这两根线就可以进行通信。

什么是 OLED 显示屏？

OLED 的特性是自己发光，不像 TFT LCD 需要背光，因此可视度和亮度均高，其次是电压需求低且省电效率高，加上反应快、重量轻、厚度薄，构造简单，成本低等特点。简单来说跟传统液晶的区别就是里面像素的材料是由一个个发光二极管组成，因为密度不高导致像素分辨率低，所以早期一般用作户外 LED 广告牌。随着技术的成熟，使得集成度越来越高。小屏也可以制作出较高的分辨率。



图 29-33 I2C 接口的 OLED

在了解完 I2C 和 OLED 显示屏后，我们先来看看开发板的原理图，也就是 MicroPython 上的 OLED 接口是如何连线的。下图是 pyBase 开发底板的原理图。

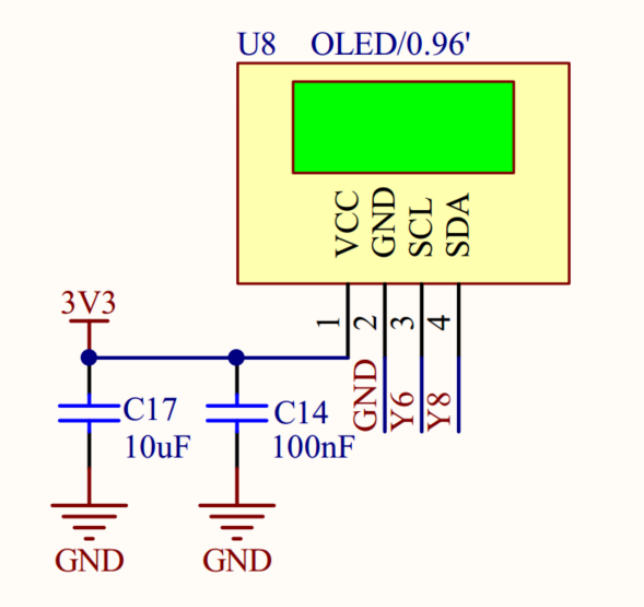


图 29-34 OLED 接口原理图

我们再来看看 pyAI-K210 转接板的原理图接口部分。

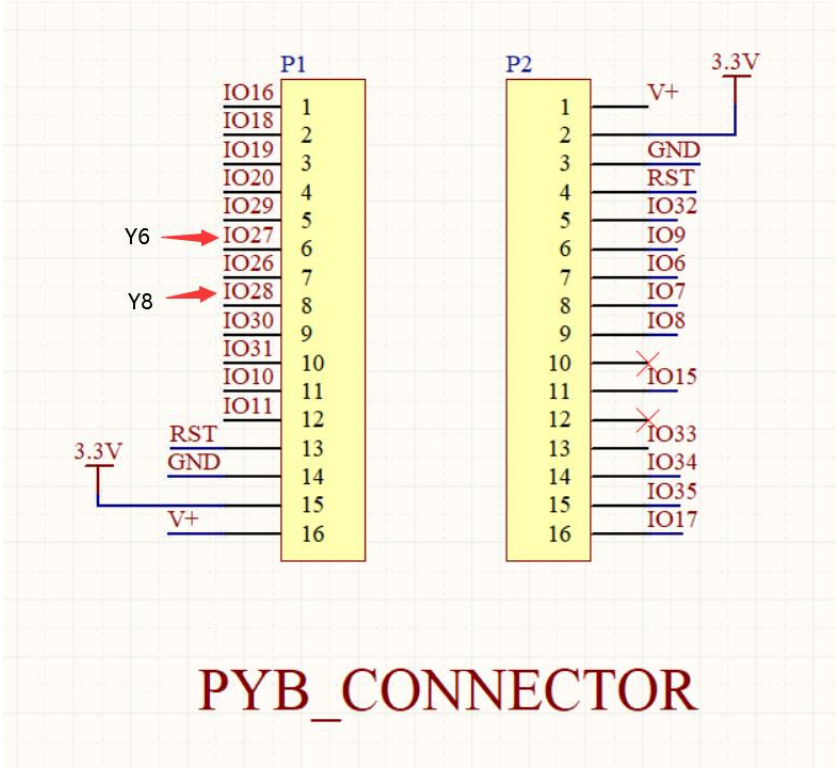


表 29-8　pyAI-k210 PYB 接口图

结合以上可以得知 pyBase 底板连接到 OLED 的对应关系是 Y6→SCL 和 Y8→SDA。对应 pyAI-K210 的关系是：IO27→Y6→SCL，IO28→Y8→SDA。本例程将使用 MicroPython 的 Machine 模块的 I2C 来定义 Pin 口和 I2C 初始化。具体如下：

构造函数
machine.I2C(id, mode=I2C.MODE_MASTER, scl=None, sda=None, freq=400000, timeout=1000, addr=0, addr_size=7)
构建 I2C 对象。 【id】I2C ID,[I2C.I2C0~I2C.I2C2] 【scl】时钟引脚；直接传引脚编号； 【sda】数据引脚；直接传引脚编号； 【freq】通信频率，即速度； 【timeout】参数保留，设置无效；

【addr】从机地址； 【addr_size】地址长度， 支持 7 位寻址和 10 位寻址， 取值 7 或者 10。
使用方法
i2c.scan()
扫描 I2C 总线的设备。返回地址，如：0x3c；
i2c.readfrom(addr,len)
从指定地址读数据。addr:指定设备地址；len:读取字节数；
i2c.writeto(addr,buf)
写数据。addr:从机地址；buf:数据内容；
i2c.deinit()
注销 I2C。
*其它更多用法请阅读官方文档： 文档链接： https://maixpy.sipeed.com/zh/libs/machine/i2c.html

表 29-9 I2C 对象

定义好 I2C 后，还需要驱动一下 OLED。这里我们已经写好了 OLED 的库函数，在 ssd1306k.py 文件里面。开发者只需要将改 py 文件拷贝到 pyAI-K210 文件系统里面，然后在 main.py 里面调用函数即可。人生苦短，我们学会调用函数即可，也就是注重顶层的应用，想深入的小伙伴也可以自行研究 ssd1306k.py 文件代码。OLED 显示屏对象介绍如下：

构造函数
oled = SSD1306_I2C(i2c, addr)
构 OLED 显示屏对象。默认分辨率 128*64； 【i2c】定义好的 I2C 对象; 【addr】显示屏设备地址。
使用方法
oled.fill([value])
清屏； value=0x00 (黑屏)；value=0xFF(白屏)

oled.text(string,x,y)
将 string 字符写在指定为位置。 【string】：字符； 【x】横坐标；[0-127]； 【y】纵坐标。[0-7] 共 8 行。

表 29-10 OLED 对象

学习了 I2C、OLED 对象用法后我们通过编程流程图来理顺一下思路：

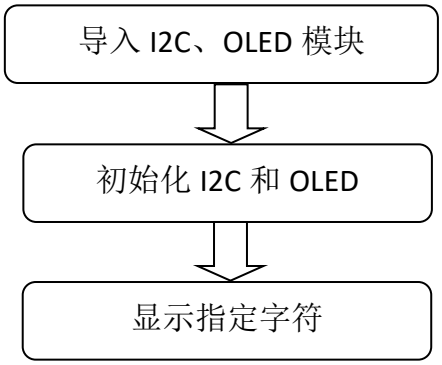


图 29-35 代码编写流程

参考代码如下：

```
'''
实验名称：I2C 总线（OLED 显示屏）
版本：v1.0
日期：2019.12
作者：01Studio
实验内容：学习使用 MicroPython 的 I2C 总线通讯编程和 OLED 显示屏的使用。
'''

from machine import I2C
from ssd1306k import SSD1306

#定义 I2C 接口和 OLED 对象
```

```
i2c = I2C(I2C.I2C0, mode=I2C.MODE_MASTER,scl=27, sda=28)
oled = SSD1306(i2c, addr=0x3c)

#清屏,0x00(白屏)，0xff(黑屏)
oled.fill(0)



#显示字符。参数格式为（str,x,y）,其中 x 范围是 0-127，y 范围是 0-7（共 8 行）
oled.text("Hello World!", 0, 0) #写入第 0 行内容
oled.text("MicroPython", 0, 2) #写入第 2 行内容
oled.text("By 01Studio", 0, 5) #写入第 5 行内容
```

上述代码中 OLED 的 I2C 地址是 0x3C,不同厂家的产品地址可能预设不一样，具体参考厂家的说明书。或者也可以通过 I2C.scan()来获取设备地址。

另外记得将我们提供的示例代码中的 ssd1306k.py 驱动文件拷贝到 pyAI-K210 的文件系统下。通过发送文件到开发板方式。

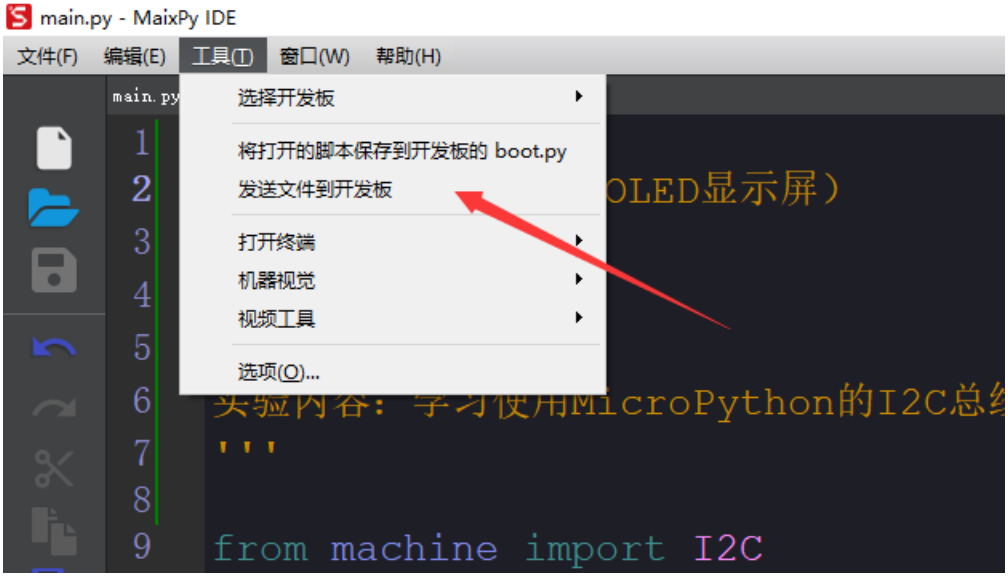


图 29-36 将 ssd1306k.py 拷贝到设备

- **实验结果：**

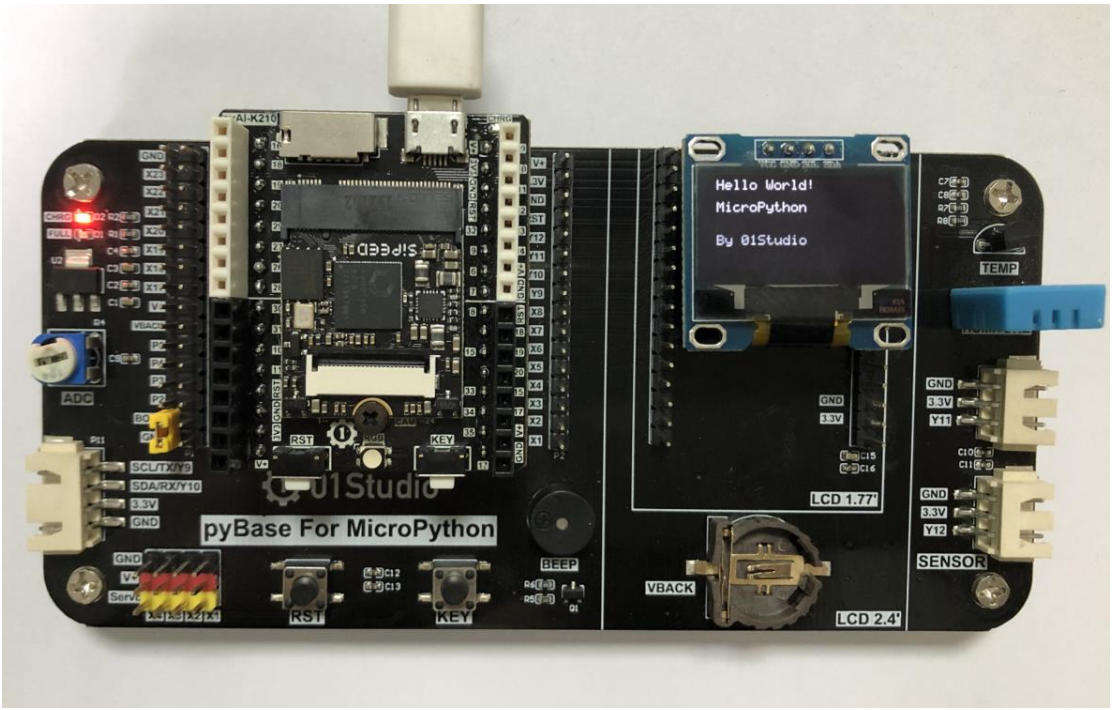


图 29-37　OLED 实验结果

- **总结：**

这一节我们学会了驱动 OLED 显示屏，换着以往如果从使用单片机从 0 开发的话你需要了解 I2C 总线原理，了解 OLED 显示屏的使用手册，编程 I2C 代码，有经验的嵌入式工程师搞不好也要弄个几天。现在基本半个小时解决问题。当然前提是别人已经给你搭好桥了，有了强大的底层驱动代码支持，我们只做好应用就好。

这一节学习的意义不仅在完成实验。在学习完 OLED 显示屏实验后，接下来我们的实验都可以使用这个 OLED 来跟用户交互了，这大大提高了实验的可观性。

29.8 UART（串口通信）

- **前言：**

串口是非常常用的通信接口，有很多工控产品、无线透传模块都是使用串口来收发指令和传输数据，这样用户就可以在无须考虑底层实现原理的前提下将各类串口功能模块灵活应用起来。

- **实验平台：**

pyAI-K210 开发套件。

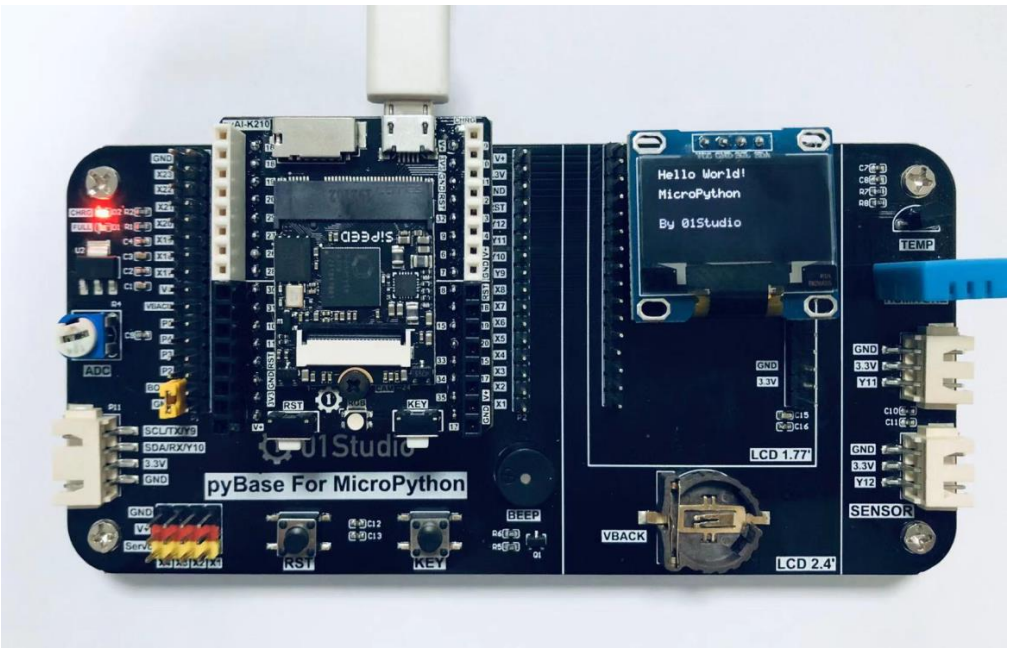


图 29-38 MicroPython 开发套件

- **实验目的：**

编程实现串口收发数据。

- **实验讲解：**

K210 一共有 3 个串口，每个串口可以自由映射引脚。 例：

```
# IO6→RX1，IO7→TX1
fm.register(6, fm.fpioa.UART1_RX, force=True)
fm.register(7, fm.fpioa.UART1_TX, force=True)
```

我们来了解一下 MaixPy 串口对象的构造函数和使用方法：

构造函数
machine.UART(uart,baudrate,bits,parity,stop,timeout, read_buf_len)
创建 UART 对象。 【uart】串口编号。[UART.UART1~UART3] 【baudrate】波特率，常用 115200、9600 【bits】数据位,默认 8 【parity】校验；默认 None, 0(偶校验)，1(奇校验) 【stop】停止位，默认 1 【timeout】串口接收超时时间 【read_buf_len】串口接收缓冲大小。
使用方法
UART.read(num)
读取串口缓冲数据 【num】读取字节数
UART.readline(num)
读取串口缓冲数据的行 【num】行数
UART.write(buf)
串口发送数据 【buf】需要发送的数据
UART.deinit()
注销串口

表 29-11　PYB 的 UART 对象

我们可以用一个 USB 转 TTL 工具，配合电脑上位机串口助手来跟 MicroPython 开发板模拟通信。

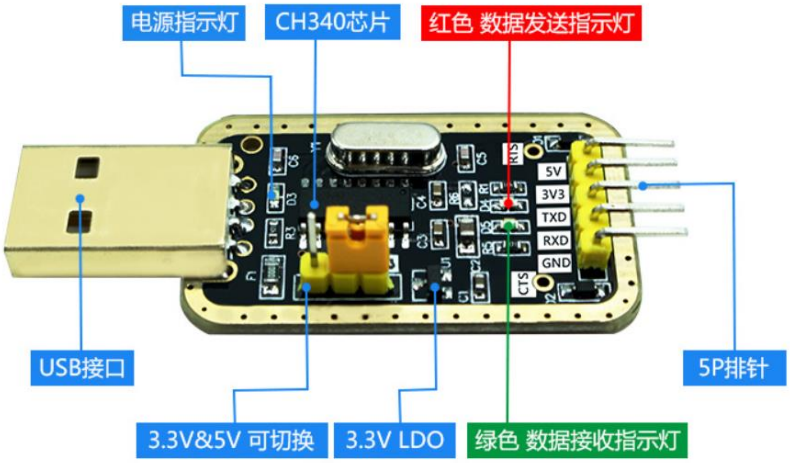


图 29-39 常用 USB 转串口工具

注意要使用 3.3V 电平的 USB 转串口 TTL 工具，本实验我们使用 pyBase 的外接串口引脚，也就是 Y9（TX）和 Y10（RX），接线示意图如下：



图 29-40 串口接线图

从 pyAI-K210 原理图可以看到外部 IO6→Y9→RX ， IO7→Y10→TX。

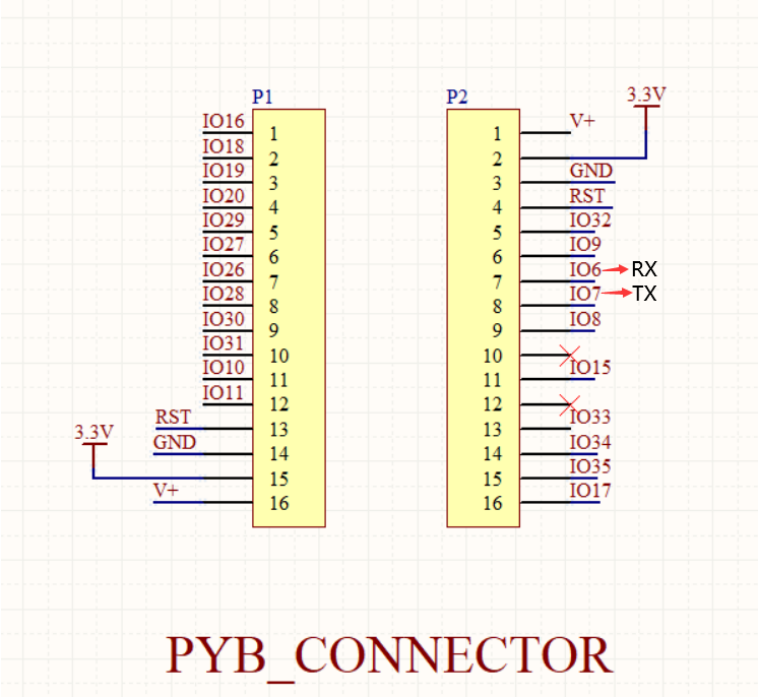


图 29-41 串口引脚

在本实验中我们可以先初始化串口，然后给串口发去一条信息，这样 PC 机的串口助手就会在接收区显示出来，然后进入循环，当检测到有数据可以接收时候就将数据接收并打印，并通过 REPL 打印显示。代码编写流程图如下：

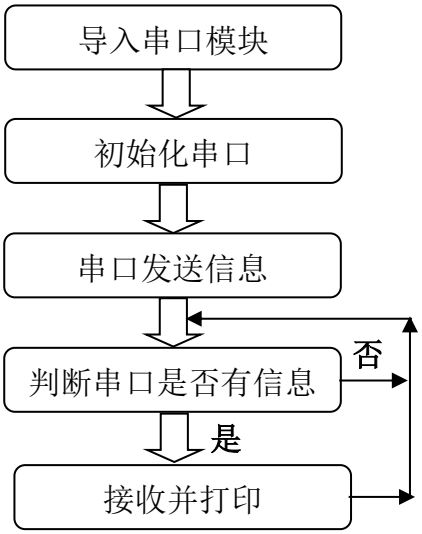


图 29-42 代码编程流程图

参考代码：

```
'''
实验名称：串口通信
版本： v1.0
日期： 2019.12
作者： 01Studio
说明：通过编程实现串口通信，跟电脑串口助手实现数据收发。
'''

from machine import UART,Timer
from fpioa_manager import fm

#映射串口引脚
fm.register(6, fm.fpioa.UART1_RX, force=True)
fm.register(7, fm.fpioa.UART1_TX, force=True)

#初始化串口
uart = UART(UART.UART1, 115200, read_buf_len=4096)
uart.write('Hello 01Studio!')

while True:

    text=uart.read() #读取数据

    if text: #如果读取到了数据
        print(text.decode('utf-8')) #REPL 打印
        uart.write('I got'+text.decode('utf-8')) #数据回传
```

- **实验结果：**

我们按照上述方式将 USB 转 TTL 的 TX 接到 Y10(IO6)，RX 接到 Y9(IO7)。GND 接一起，3.3V 可以选择接或不接。（可以直接接到 pyBase 左侧的 4P 防呆接口。）

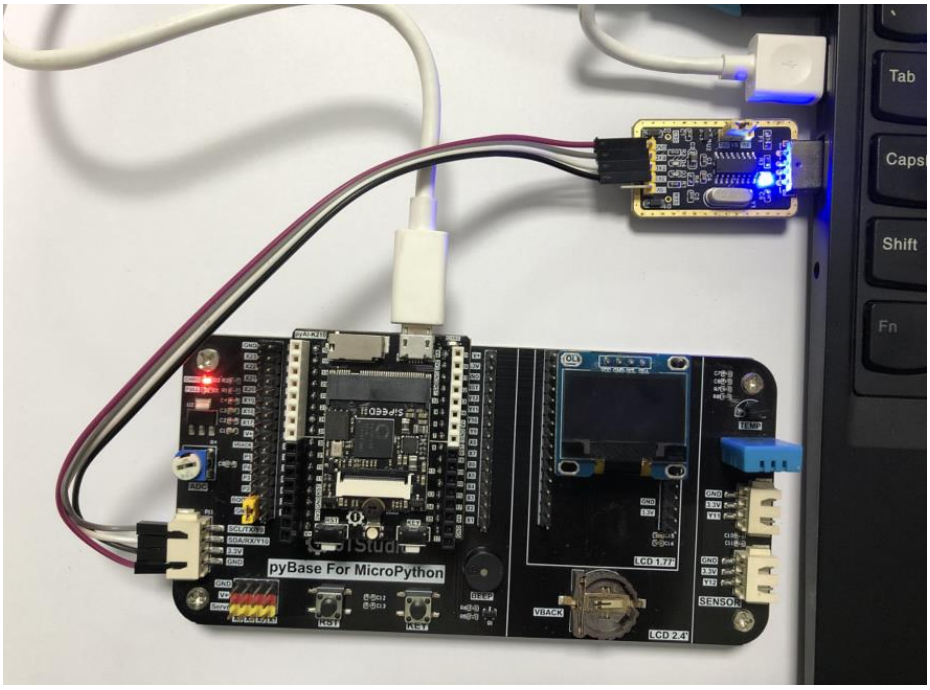


图 29-43 将 pyboard 和 usb ttl 工具同时接入电脑

这时候打开电脑的设备管理器，能看到 2 个 COM。写着 CH340 的是串口工具，另外一个则是 pyAI-K210 的 REPL。如果 CH340 驱动没安装，则需要手动安装，驱动在：配套资料包→开发工具→windows→串口终端→CH340 文件夹下。



图 29-44

本实验要用到串口助手，打开配套资料包→开发工具→windows→串口终端工具下的【UartAssist.exe】软件。

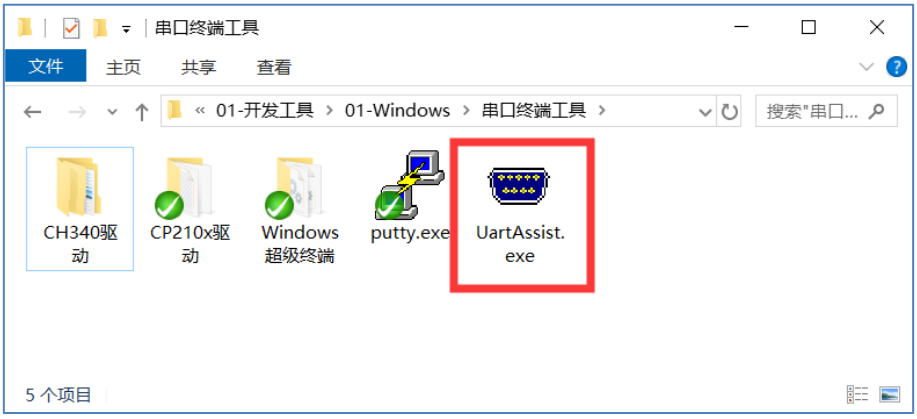


图 29-45

将串口工具配置成 COM14（根据自己的串口号调整）。波特率 115200。运行程序，可以看到一开始串口助手收到 pyAI-K210 上电发来的信息“Hello 01Studio!”。我们在串口助手的发送端输入“http://www.01studio.org”， 点击发送，可以看到 pyAI-K210 在接收到该信息后在 REPL 里面打印了出来。如下图所示：

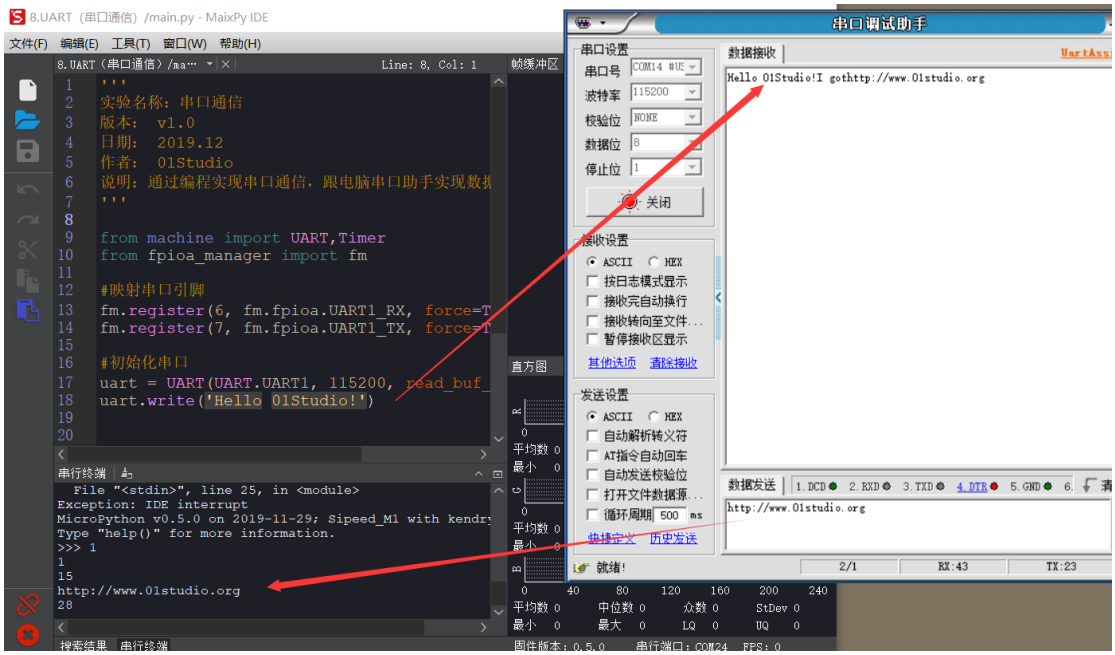


图 29-46 串口收发实验

- **总结：**

通过本节我们学会了串口收发应用，pyAI-K210 拥有 3 个串口，因此可以接多个串口外设。从而实现更多的功能。

29.9 thread（线程）

- **前言：**

我们看到前面的编程都是一个循环来完成，但当我们需要分时完成不同任务时候，线程编程就派上用场了。这有点像 RTOS(实时操作系统)，今天我们就来学习一下如何通过 MicroPython 编程实现多线程。

- **实验平台：**

pyAI-K210 开发套件。

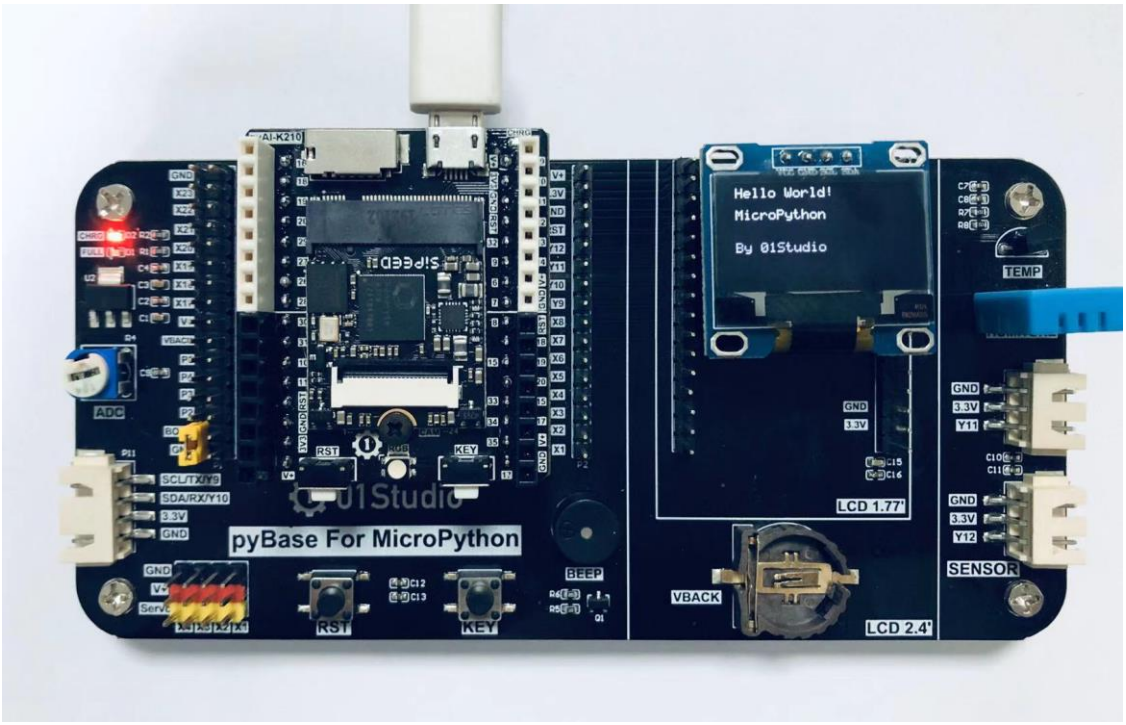


图 29-47

- **实验目的：**

编程实现多线程同时运行任务。

- **实验讲解：**

pyAI-K210 的 MicroPython 固件已经集成了_thread 线程模块。我们直接调用即可。该模块衍生于 python3，属于低级线程，详情可以看官网介绍：https://docs.python.org/3.5/library/_thread.html#module-thread

编程流程如下：

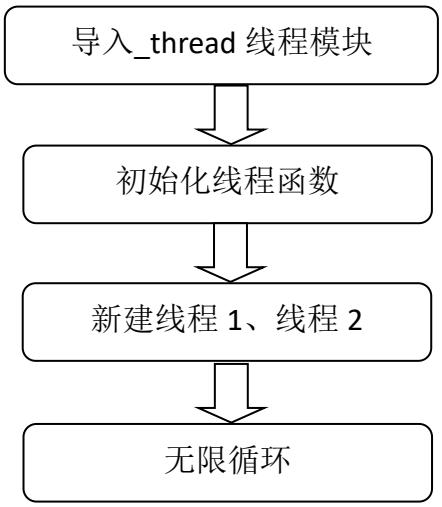


图 29-48 代码编写流程

参考代码如下：

```
'''
实验名称：线程
版本： v1.0
日期： 2019.12
翻译和注释： 01Studio
说明：通过编程实现多线程。
'''

import _thread #导入线程模块
import time

#线程函数
def func(name):
    while True:
        print("hello {}".format(name))
        time.sleep(1)
```

```
_thread.start_new_thread(func,("1",)) #开启线程 1，参数必须是元组
_thread.start_new_thread(func,("2",)) #开启线程 2，参数必须是元组


while True:
    pass
```

- **实验结果：**

运行代码，可以看到串口终端重复执行 2 个线程。

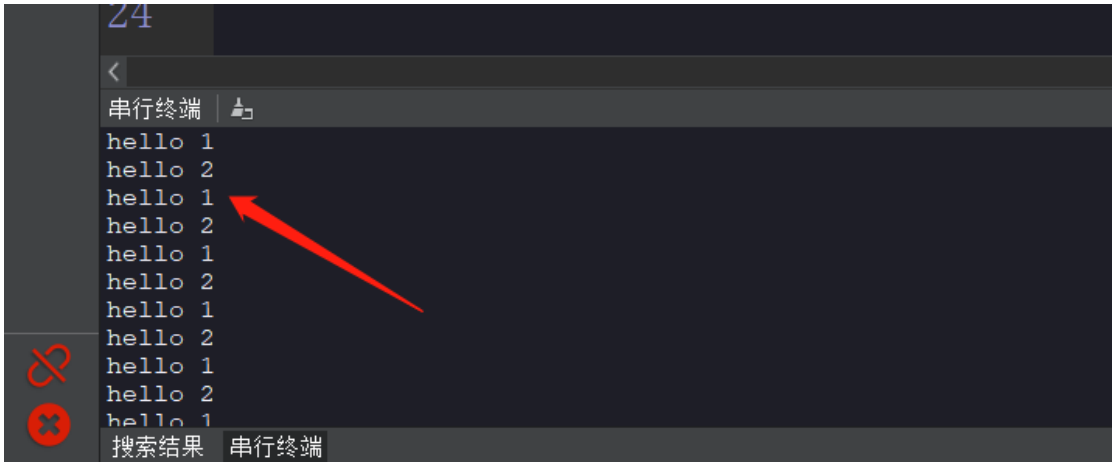


图 29-49 实验结果

- **总结：**

本章我们学习了基于 MicroPython 的线程编程，实际是 python3 的低级线程编程。线程的引入让我们在处理多任务时候变得简单，大大增加了程序编写的灵活性。

第30章　机器视觉

30.1　LCD

- **前言：**

LCD 液晶显示屏是非常常见的一个外接显示设备，跟前面的 OLED 显示屏相比，LCD 会更常用一些，我们看到的手持设备、小型电器，很多都用到 LCD，部分配合触摸屏应用，能实现非常多的功能。

除此之外，LCD 还是 pyAI-K210 机器视觉应用中显示的重要工具。

- **实验平台：**

pyAI-K210 开发套件。

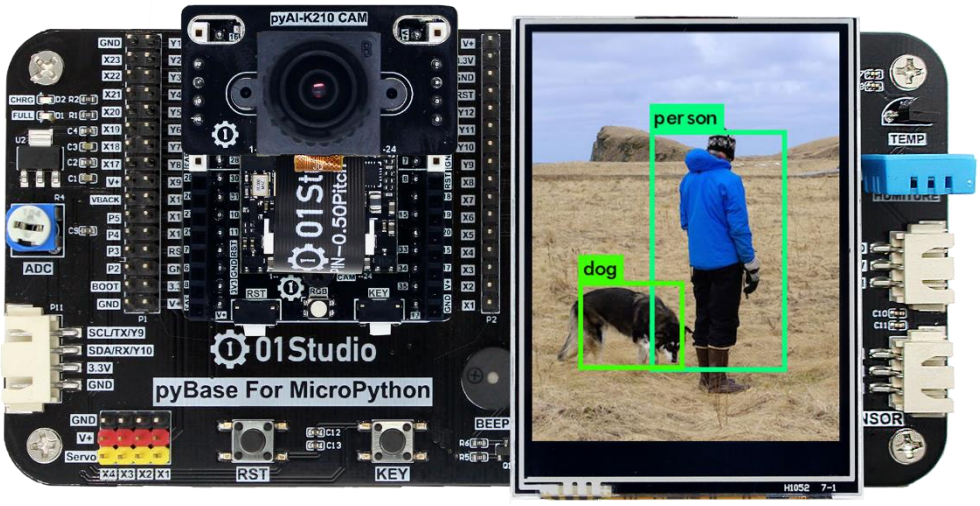


图 30-1　2.8 寸 LCD（电阻触摸）

- **实验目的：**

通过 MicorPython 编程了解 LCD 的使用方法。

- **实验讲解：**

我们先来看看实验所使用 LCD 的参数介绍：

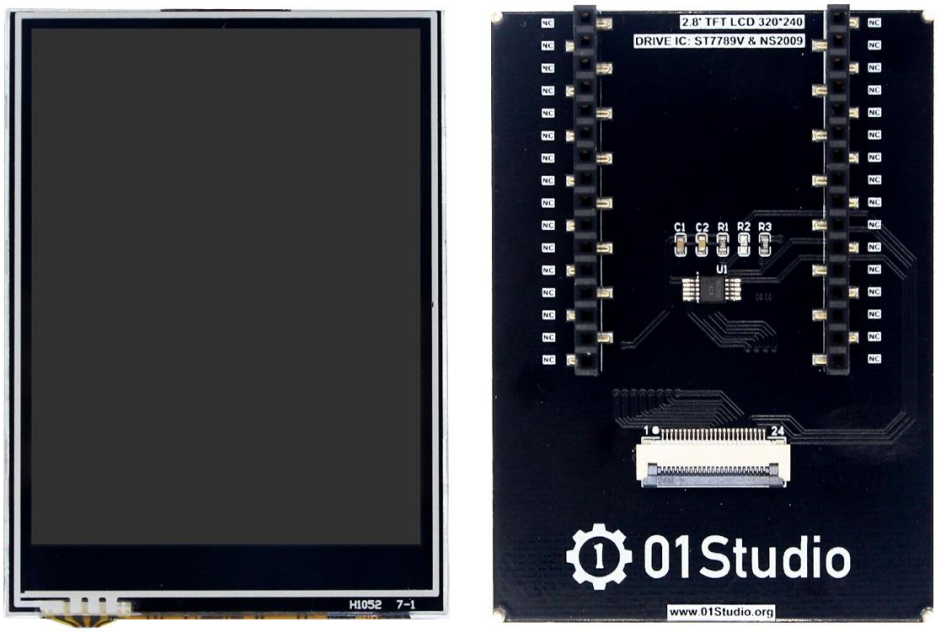


图 30-2　2.8 寸 LCD 显示屏（带触摸）

功能参数	
供电电压	3.3V
屏幕尺寸	2.8 寸
颜色参数	TFT 彩色
驱动芯片	ST7789V + NS2009
触摸方式	电阻屏
通讯方式	8 位并口总线
接口定义	24Pin-0.5mm-FPC 座
整体尺寸	7*5 cm

表 30-1　LCD 功能参数

接线方法：

2.8 寸 LCD 跟 pyAI-K210 通过底部的 24P 排线连接，注意排线均为下接（排线金手指朝下）。将排线塞进座子，扣下即可。

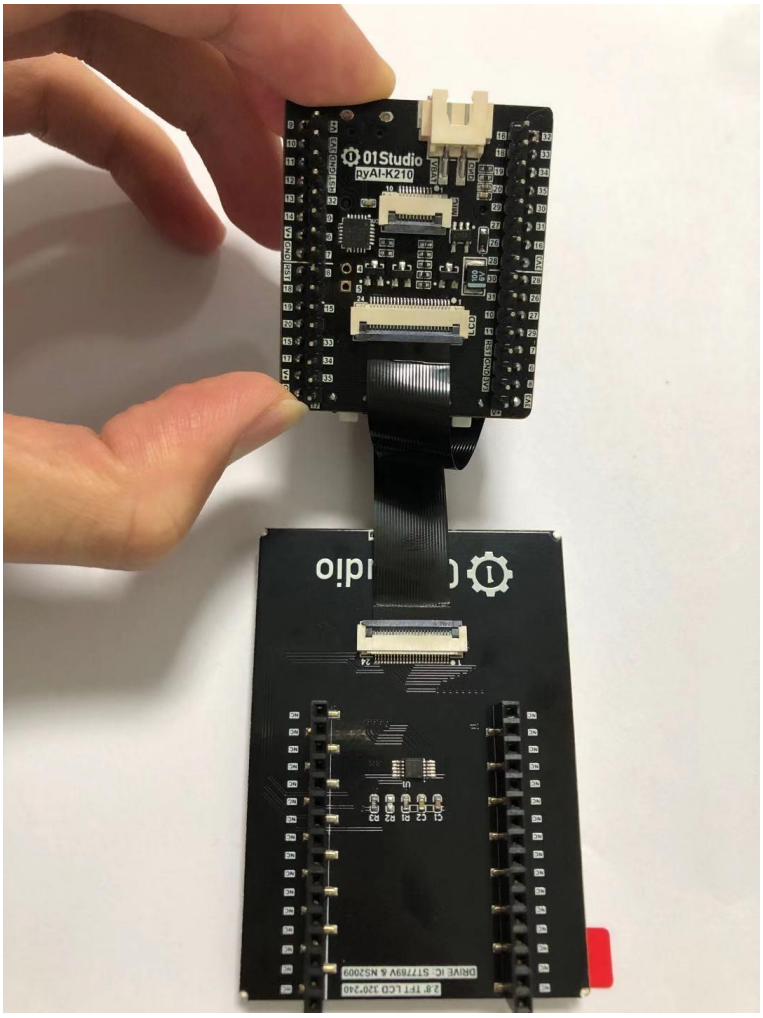


图 30-3 pyAI-K210 与 2.8 寸 LCD 接线方式

2.8 寸 LCD 背面的排母是没有电气连接的，仅仅是为了方便用户安装，可以接到 pyAI-K210 背面或者 pyBase 上，具体如下：

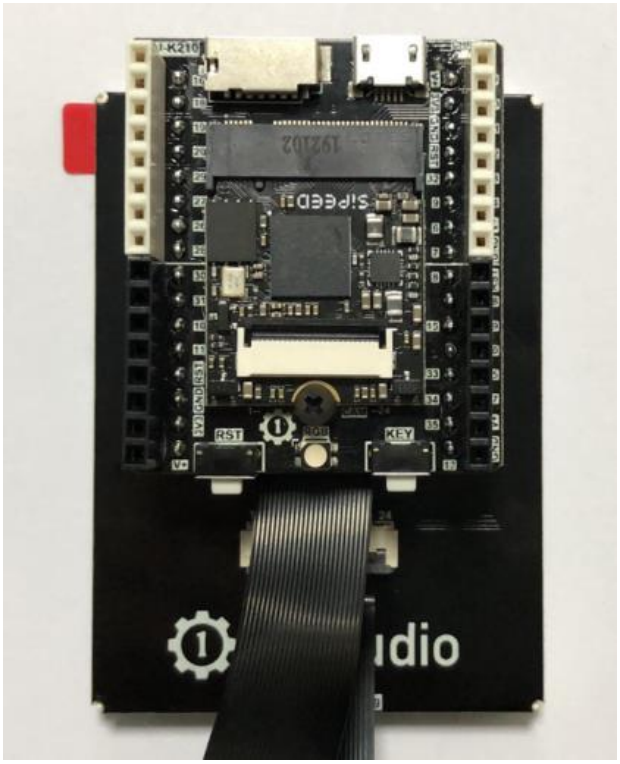


图 30-4　LCD 与 pyAI-K210

pyAI-K210 与 LCD 连接后，可以插到 pyBase 上，建议排线藏到后方，首次插进注意排线不要压着即可。组装完成如下图所示。

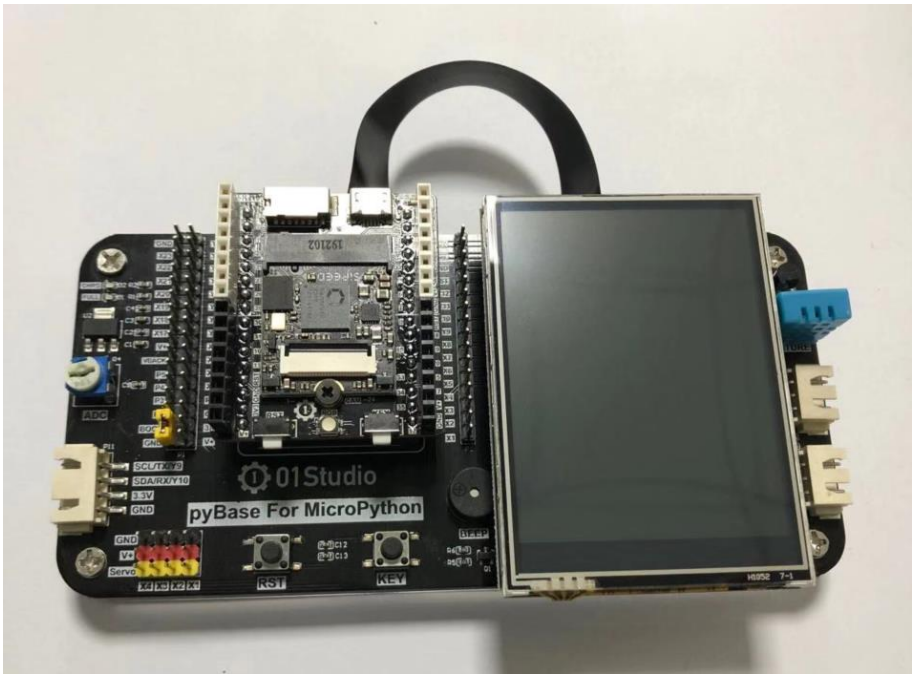


图 30-5 LCD 与 pyBase

本实验用的 LCD 是 2.8 寸，驱动是常见的 ST7789V，使用 8 位接口跟 pyAI-K210 通信，按以往嵌入式 C 语言开发，我们需要对 ST7789 进行编程实现驱动，然后再建立各种字符显示及显示图片等函数。

使用 MicroPython 其实也需要做以上工作，但由于可读性和移植性强的特点，我们只需要搞清各个对象函数使如何使用即可。总的来说和之前一样，有构造函数和功能函数。构造函数解决的是初始化问题，告诉 pyAI-K210 外设是怎么接线，是什么样的；而功能函数解决的则是使用问题，我们基于自己的需求直接调用相关功能函数，实现自己的功能即可！

我们管这些函数的集合叫驱动，MaixPy 已经将这 LCD.py 驱动写好了，我们学会如何使用即可。其构造函数和使用方法如下：

构造函数
`lcd.init(type=1,freq=15000000,color=lcd.BLACK)`
初始化 LCD。 【type】LCD 类型； 【freq】通信频率； 【color】LCD 初始化的颜色。
使用方法
`lcd.deinit()`
注销 LCD 驱动，释放 IO 引脚。
`lcd.clear(color)`
填充指定颜色。默认是黑色
`lcd.draw_string(x,y,str,color,bg_color)`
写字符 【x,y】起始坐标； 【str】字符内容 【color】字体颜色 【bg_color】字体背景颜色
`lcd.display(image,roi=Auto)`

显示图片。 【image】RGB565 或 GRAYSCALE 图片。 【ROI】显示的感兴趣区域，未指定则为图像大小。
lcd.rotation(dir)
LCD 屏幕方向设定。 【dir】取值范围[0-3]，从 0 到 3 依顺时钟旋转。
lcd.mirror(invert)
镜面显示。 【invert】=True 则为镜面显示；=False 则否。
更多 LCD 模块说明请看 MaxiPy 官方文档： https://maixpy.sipeed.com/zh/libs/machine_vision/lcd.html

表 30-2 LCD 对象

有了上面的对象构造函数和使用说明，编程可以说是信手拈来了，我们来跑一下其主要功能显示字符和图像，代码编写流程如下：

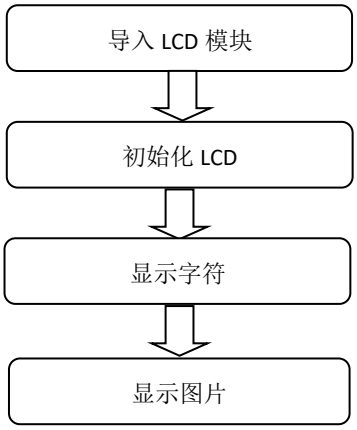


图 30-6 编程流程图

参考函数代码如下：

```
'''
实验名称：LCD
```

```
版本： v1.0
日期： 2019.12
作者： 01Studio
说明：编程实现 LCD 显示信息。将 01Studio.bmp 文件发送到开发板。
'''

import lcd,image,utime

lcd.init() #初始化 LCD
lcd.clear(lcd.WHITE) #清屏白色

#显示字符
lcd.draw_string(110, 120, "Hello 01Studio!",lcd.BLACK, lcd.WHITE) #显示
字符

utime.sleep(2) #延时 2 秒

#显示图像
lcd.rotation(1) #由于图像默认是 240*320，因此顺时钟旋转 90°。
lcd.display(image.Image("01Studio.bmp"))
```

- **实验结果：**

将示例程序文件夹中的“01Studio.bmp”文件发送到开发板。运行程序，可以看到 pyAI-K210 开发套件上的 LCD 先显示指定字符，再显示图片。

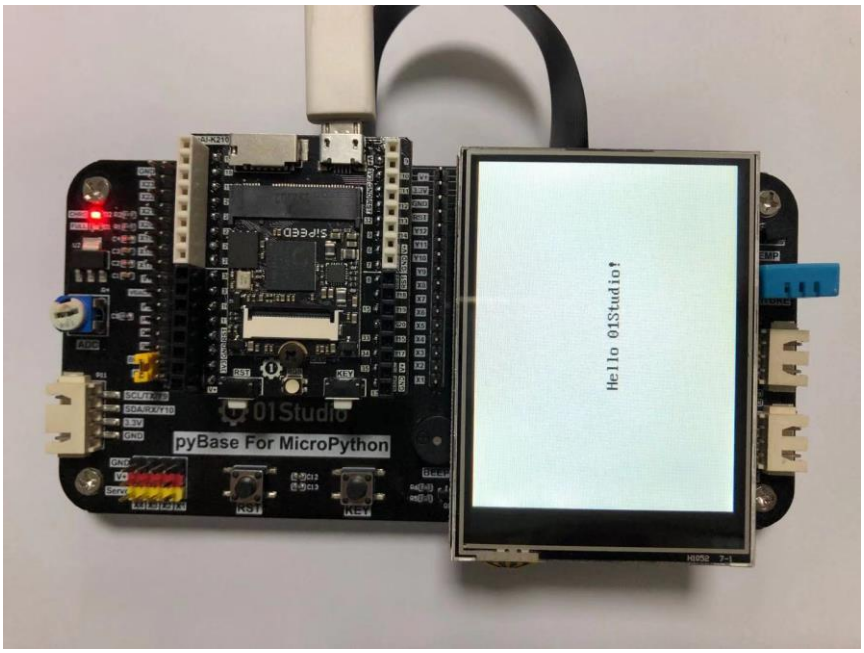


图 30-7 显示字符

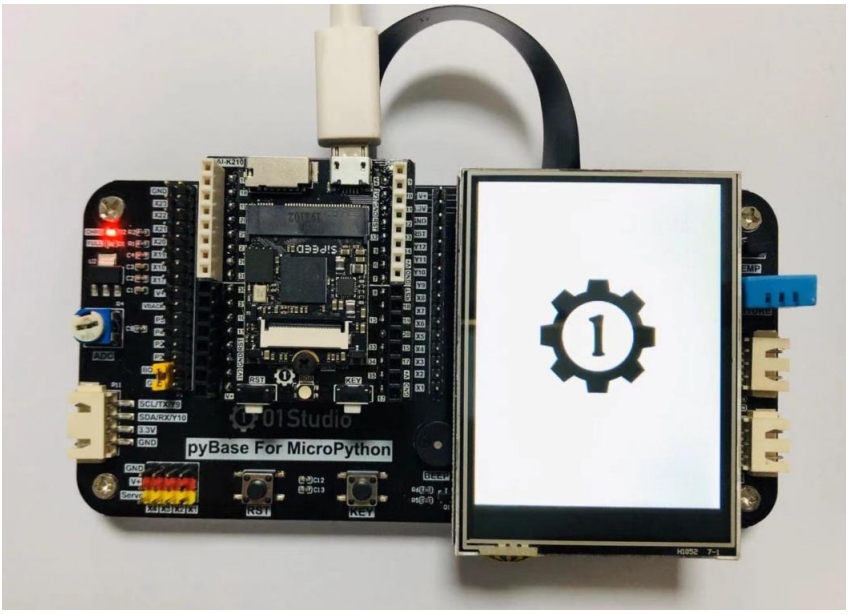


图 30-8 显示 LOGO 图片

由于我们 LCD 背面增加了 pyboard 标准排母接口（电路特性实际是全部引脚悬空），因此也可以直接将 pyAI-K210 插到 LCD 背面，使用锂电池供电。如下图所示：（注意要先将 LCD.py 改成 main.py，发送到开发板。）

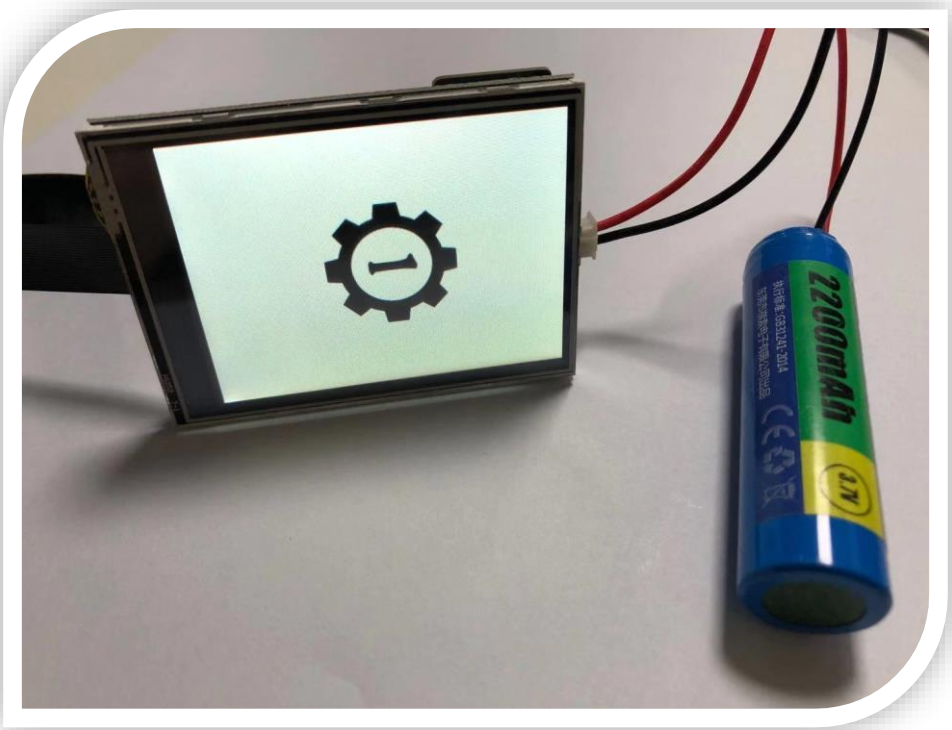


图 30-9　LCD 直接连接 pyAI-K210

- **总结：**

本实验用到的 LCD 和其驱动芯片 ST7789V 是常见的产品，可以看到 MicroPython 编程只关注屏幕的驱动和分辨率，对于不用尺寸和型号的 LCD 来说，移植性非常强，有兴趣的朋友可以自行驱动其它系列的 LCD。

30.2 摄像头应用

- **前言：**

从前面的基础实验我们熟悉了 K210 基于 MicroPython 的编程方法，但那可以说是只发挥了 K210 冰山一角的性能应用，摄像头是整个机器视觉应用的基础。今天我们就通过示例代码来看看 pyAI-K210 是如何使用摄像头的。

- **实验平台：**

pyAI-K210 开发套件。

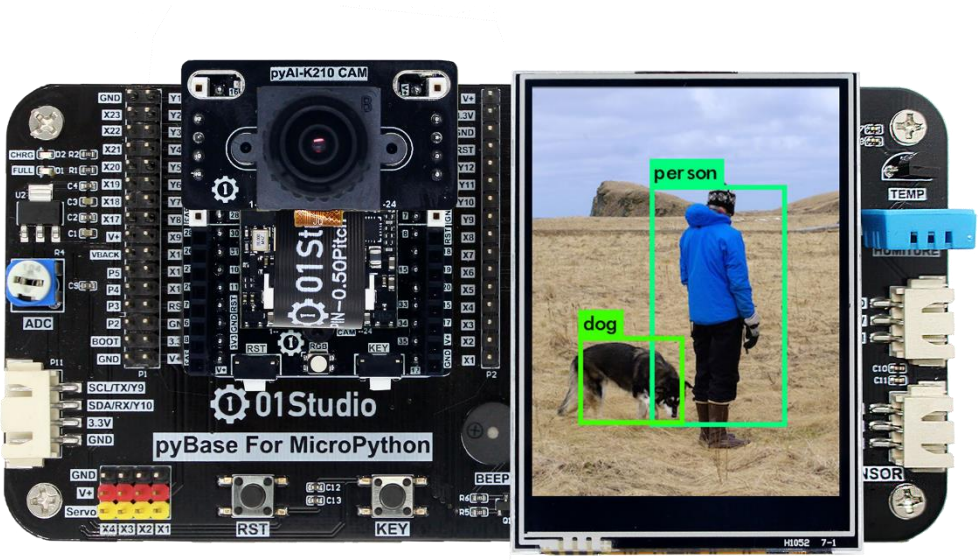


图 30-10 pyAI-K210+摄像头模块

- **实验目的：**

学习官方自带的 hello wrold 例程，理解 K210 摄像头基本编程和配置原理。

- **实验讲解：**

摄像头接线方式：

pyAI-K210 集成了 24P 摄像头接口，可以直接连接标准 OV2640 等 24P 摄像头模块，排线接线方式均为下接（排线金手指朝下），具体如下：

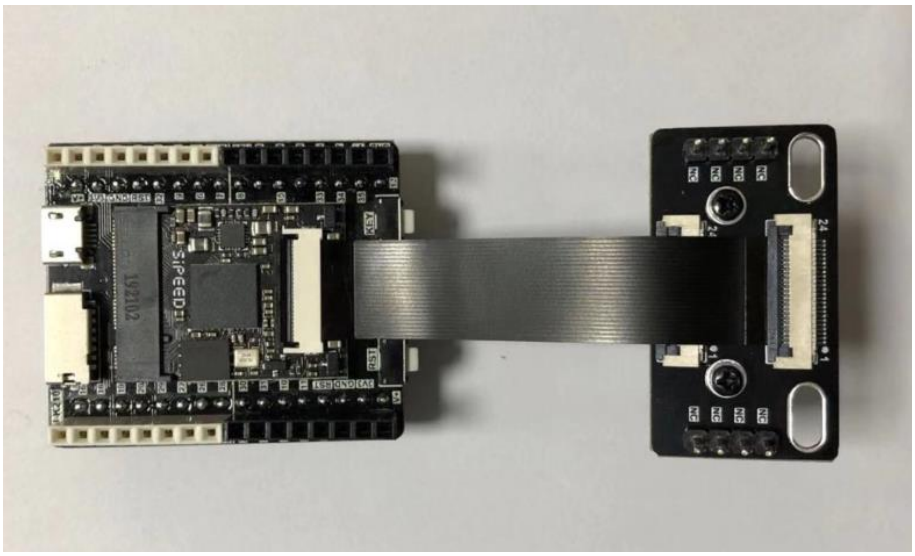


图 30-11 摄像头接线方式

接完后可以直接将 K210 摄像头模块插到 pyAI-K210 的拓展排母上，由于摄像头的排针均没有电气连接，所以可以根据自己需要任意摆放位置。

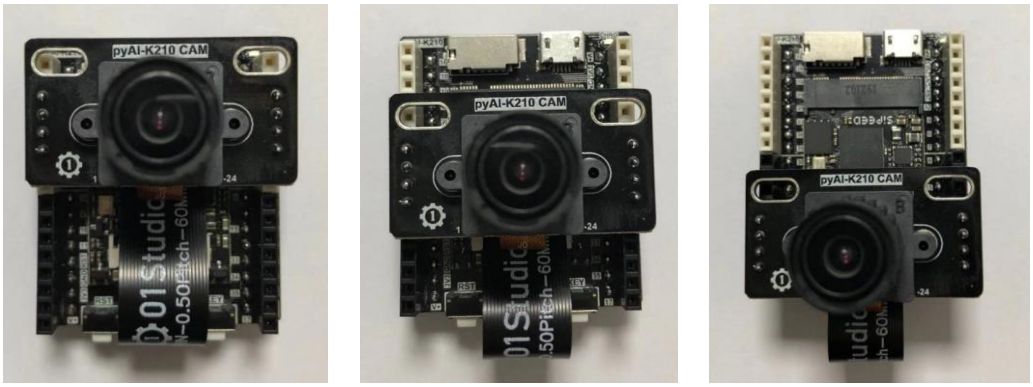


图 30-12 摄像头位置任意摆放

在 IDE 中打开 **零一科技（01Studio）MicroPython 开发套件配套资料\02-示例程序\5.pyAI-K210\2.机器视觉\1.摄像头应用** 目录下的 Camera.py 文件。

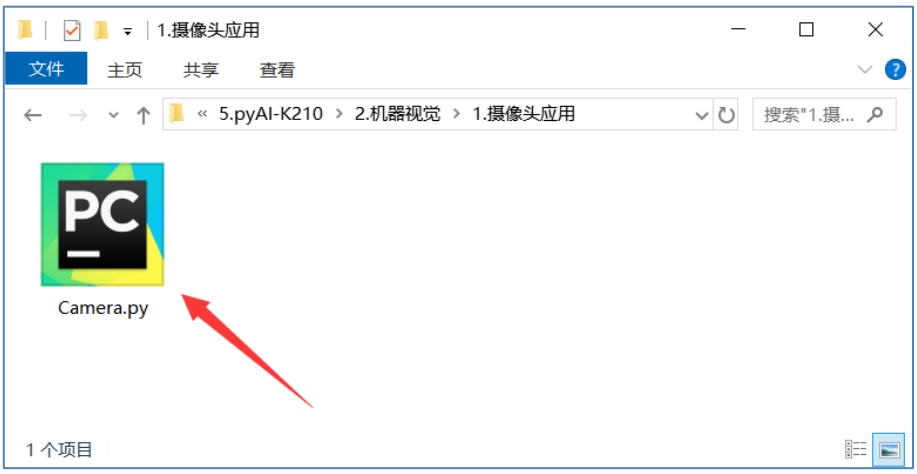


图 30-13 摄像头例程位置

打开后发现编辑框出现了相关代码，我们可以先直接跑一下代码看看实验现象，连接 pyAI-K210，点击运行，可以发右图上方出现了摄像头实时采集的图像。(由于 K210 通过串口方式跟 IDE 交互，官方降低 IDE 缓冲区显示图像质量，官方推荐通过 LCD 显示。)

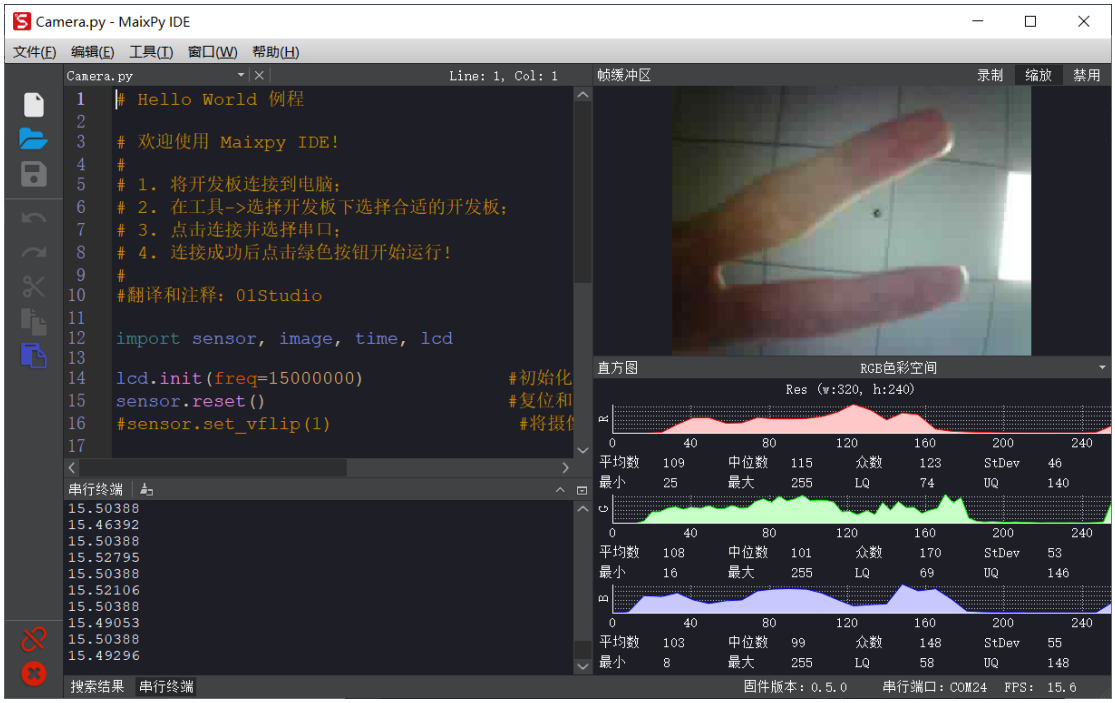


图 30-14

MaixPy 机器视觉库代码大部分都是参考 OpenMV 移植过来，其已经将所有的摄像头功能封装到 sersor 模块中，用户可以通过调用轻松使用。这也是使用

MicroPython 编程的魅力所在。

构造函数
sensor
摄像头对象，通过 import 直接调用
使用方法
`sensor.reset()`
初始化摄像头
`sensor.set_pixformat(`*`pixformat`*`)`
设置像素格式。`pixformat` 有 3 个参数。 `sensor.GRAYSCAL`：灰度图像，每像素 8 位（1 字节），处理速度快； `sensor.RGB565`：每像素为 16 位（2 字节），5 位用于红色，6 位用于绿色，5 位用于蓝色，处理速度比灰度图像要慢。
sensor.set_framesize(*framesize*)

设置每帧大小（即图像尺寸）。常用的 *framesize* 参数有下面这些：
`sensor.QQVGA: 160*120;`
`sensor.QVGA: 320*240;`
`sensor.VGA: 640*480;`

sensor.skip_frames([*n*, *time*])
摄像头配置后跳过 n 帧或者等待时间 `time` 让其变稳定。`n`:跳过帧数；`time`：等待时间,单位 `ms`。 （如果 n 和 `time` 均没指定，则默认跳过 `300` 毫秒的帧。）
sensor.snapshot()
使用相机拍摄一张照片，并返回 `image` 对象。
*其它更多用法请阅读 `MaixPy` 官方文档： 文档链接：https://maixpy.sipeed.com/zh/libs/machine_vision/sensor.html

表 30-3　sensor 摄像头对象

我们再来看看本例程用于计算 FPS（每秒帧数）的 clock 模块。

构造函数
clock=time.clock()
创建一个时钟。
使用方法
clock.tick()
开始追踪运行时间。
clock.fps ()
停止追踪运行时间，并返回当前 FPS（每秒帧数）。 在调用该函数前始终首先调用 tick 。
*其它更多用法请阅读 Maixpy 官方文档： 文档链接：http://docs.openmv.io/library/omv.time.html

图 30-15 clock 对象

我们来看看 helloword 代码的编写流程图：

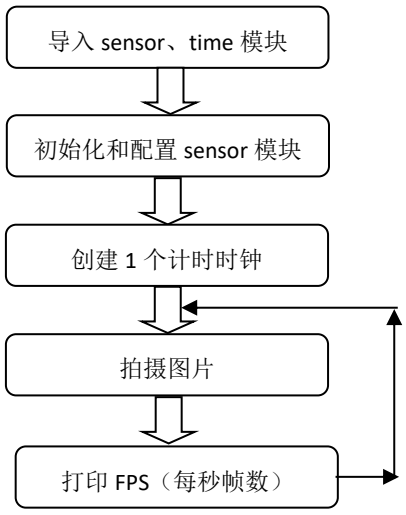


图 30-16 helloword 代码编写流程

这个实验运行的就是编辑框里面的 helloworld 代码：

```
# Hello World 例程
```

```
# 欢迎使用 Maixpy IDE!
#
# 1. 将开发板连接到电脑；
# 2. 在工具->选择开发板下选择合适的开发板；
# 3. 点击连接并选择串口；
# 4. 连接成功后点击绿色按钮开始运行!
#
#翻译和注释：01Studio

import sensor, image, time, lcd

lcd.init(freq=15000000)      #初始化 LCD
sensor.reset()               #复位和初始化摄像头，执行 sensor.run(0)停止。
#sensor.set_vflip(1)           #将摄像头设置成后置方式（所见即所得）

sensor.set_pixformat(sensor.RGB565) # 设置像素格式为彩色 RGB565 (或灰色)
sensor.set_framesize(sensor.QVGA)   # 设置帧大小为 QVGA (320x240)
sensor.skip_frames(time = 2000)     # 等待设置生效.
clock = time.clock()                # 创建一个时钟来追踪 FPS（每秒拍摄帧数）

while(True):
    clock.tick()                # 更新 FPS 时钟.
    img = sensor.snapshot()     # 拍摄一个图片并保存.
    lcd.display(img)            # 在 LCD 上显示
    print(clock.fps())          # 注意: 当 K210 连接到 IDE 时候，运行速度减
                                #半，因此当断开 IDE 时 FPS 会提升。
```

- **实验结果：**

点击运行，可以看到在右边显示摄像头实时拍摄情况，下方则显示 RGC 颜色直方图。

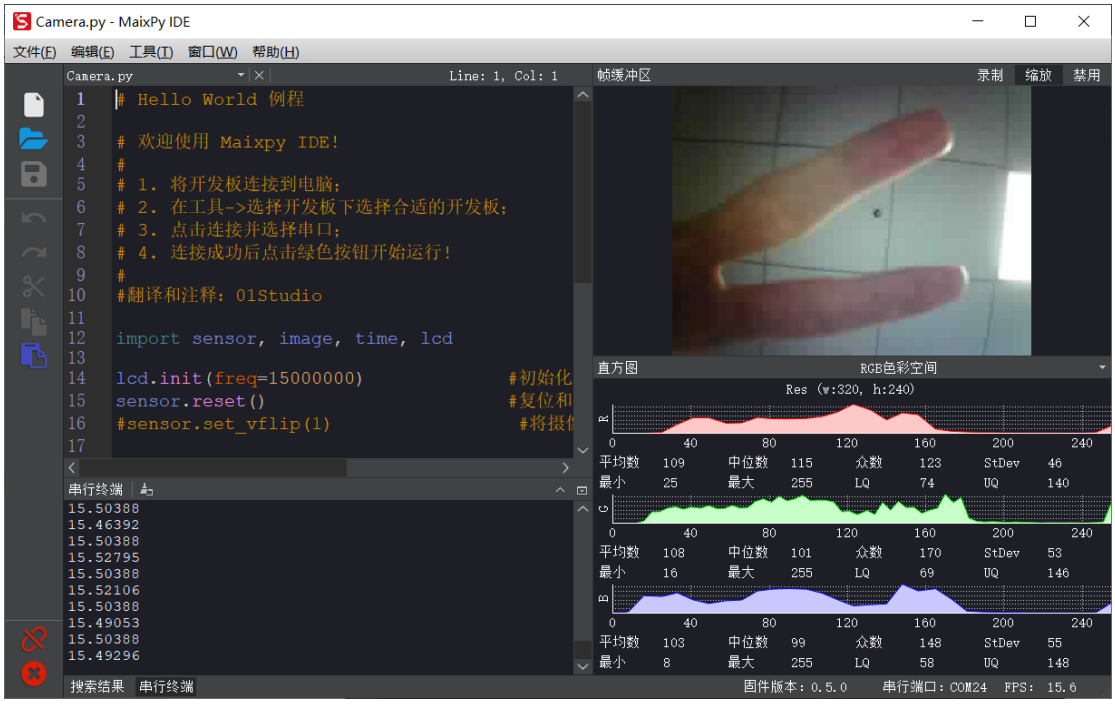


图 30-17 实验现象

点击左下角串口终端，可以看到软件弹出串口打印串口，实时显示当前的 FPS(没秒帧数)值约为 15 帧。



图 30-18 串口终端显示 FPS

通用可以看到 LCD 实时显示摄像头采集的图像.

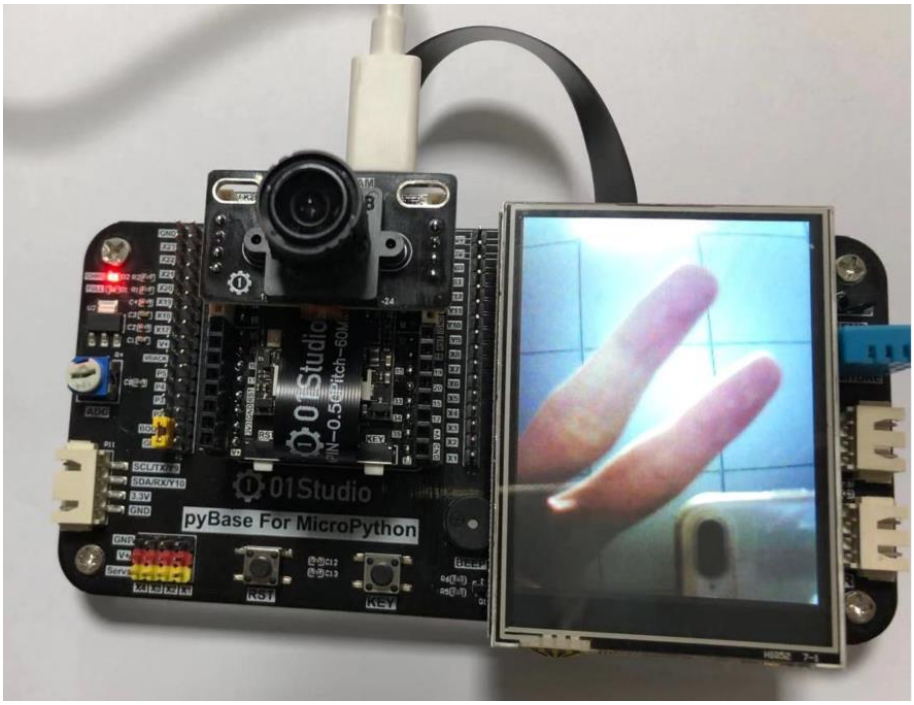


图 30-19 LCD 显示摄像头图像

- **总结：**

通过本实验，我们了解了摄像头 sensor 模块以及时间 time 模块的原理和应用，可以看到用于 pyAI-K210 的 MaixPy 将摄像头功能封装成 sensor 模块，用户不必关注底层代码编可以轻松使用。

30.3 画图

- **前言：**

通过摄像头采集到照片后，我们会进行一些处理，而这时候往往需要一些图形来指示，比如在图片某个位置标记箭头、人脸识别后用矩形框提示等。本节就是学习在图形上画图的使用功能。

- **实验平台：**

pyAI-K210 开发套件。

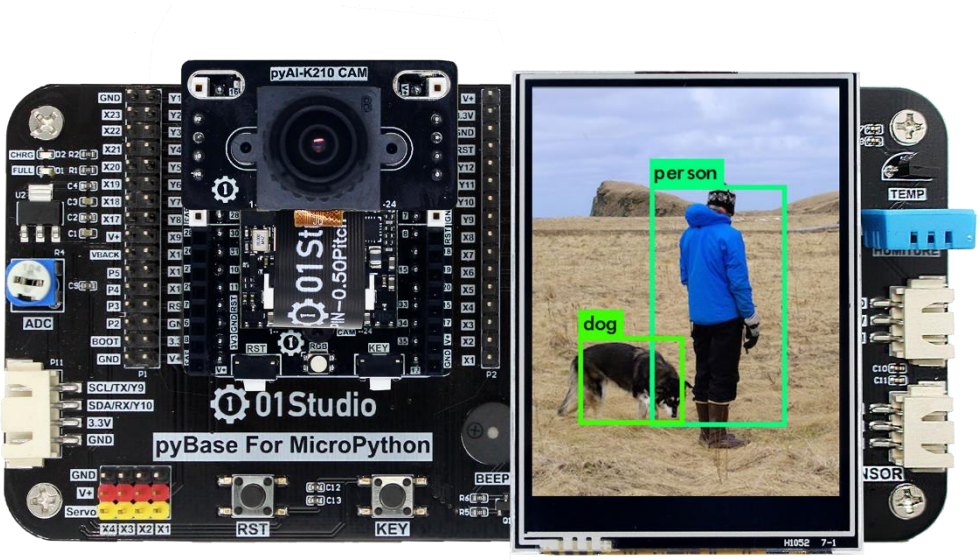


图 30-20 pyAI-K210 开发套件

- **实验目的：**

在拍摄的图片上画各种图形。

- **实验讲解：**

上一节我们学习了摄像头 sensor 模块应用，通过摄像头实时采集到的是图片 image，没错，本节实验就是建立在非常重要的 image 模块上面。MaxiPy 已经将图片处理（包含画图）封装成各类模块，我们只需要熟悉其构造函数和使用方法即可，具体如下：

构造函数
img=sensor.snapshot() 或 img=image.Image(*path*[, *copy_to_fb=False*])
创建图像，通过拍摄或者读取文件路径获取。 *copy_to_fb=True：可以加载大图片；* copy_to_fb=False：不可以加载大图片。 示例：img = image.Image("01Studio.bmp", copy_to_fb=True),表示加载根目录下的 01Studio.bmp 图片。
使用方法
image.draw_line(*x0*, *y0*, *x1*, *y1*[, *color*[, *thickness=1*]])
画线段。（x0,y0）:起始坐标；（x1,y1）:终点坐标；color:颜色，如（255,0,0）表示红色；thickness：粗细。
image.draw_rectangle(*x*, *y*, *w*, *h*[, *color*[, thickness=*1*[, *fill=False*]]])
画矩形。（x,y）:起始坐标；w:宽度；h:长度；color：颜色；thickness：边框粗细；fill:是否填充。
image.draw_circle(*x*, *y*, *radius*[, *color*[, thickness=*1*[, *fill=False*]]])
画圆。（x,y）:圆心； radius:半径； color：颜色；thickness:线条粗细；fill：是否填充。
image.draw_arrow(*x0*, *y0*, *x1*, *y1*[, *color*[, size,[thickness=*1]*]])
画箭头。（x0,y0）:起始坐标；（x1,y1）:终点坐标；color:颜色；size:箭头位置大小。thickness：线粗细。
image.draw_cross(*x*, *y*[, *color*[, *size=5*[, *thickness=1*]]])
画十字交叉。（x,y）: 交叉坐标；color:颜色；size:尺寸；thickness：线粗细。
image.draw_string(*x*, *y*, *text*[, *color*[, *scale*=1[,*mono_space=True*…]]]])
写字符。(x,y): 起始坐标；text:字符内容；color：颜色；scale：字体大小；mono_space:强制间距。
*其它更多用法请阅读 MaixPy 官方文档： 文档链接：https://maixpy.sipeed.com/zh/libs/machine_vision/image.html

表 30-4 image 对象画图功能

熟悉了 image 对象的画图功能后，我们尝试在摄像头采集到的画面依次画出线段、矩形、圆形、箭头、十字交叉和字符。具体编程思路如下：

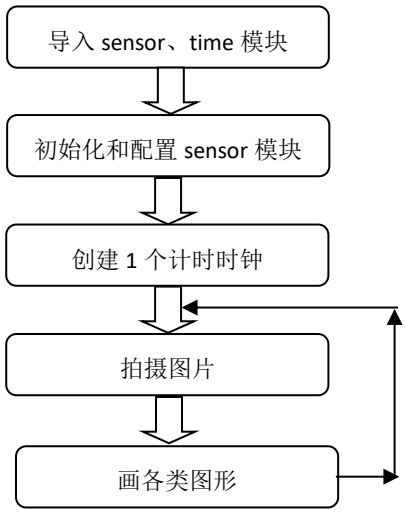


图 30-21　代码编写流程

参考代码：

```
'''
实验名称：画各种图形和写字符
版本： v1.0
日期： 2019.11
作者： 01Studio
'''

import sensor, image, time, lcd

lcd.init(freq=15000000)
sensor.reset()                      #复位摄像头
#sensor.set_vflip(1)                 #将摄像头设置成后置方式（所见即所得）

sensor.set_pixformat(sensor.RGB565) # 设置像素格式 RGB565 (or GRAYSCALE)
sensor.set_framesize(sensor.QVGA)   # 设置帧尺寸 QVGA (320x240)
sensor.skip_frames(time = 2000)     # 灯带设置响应.
```

```
clock = time.clock()                # 新建一个时钟对象计算 FPS.

while(True):
    clock.tick()
    img = sensor.snapshot()

    # 画线段：从 x0, y0 到 x1, y1 坐标的线段，颜色红色，线宽度 2。
    img.draw_line(20, 20, 100, 20, color = (255, 0, 0), thickness = 2)

    #画矩形：绿色不填充。
    img.draw_rectangle(150, 20, 100, 30, color = (0, 255, 0),
                       thickness = 2, fill = False)

    #画圆：蓝色不填充。
    img.draw_circle(60, 120, 30, color = (0, 0, 255), thickness = 2,
                    fill = False)

    #画箭头：白色。
    img.draw_arrow(150, 120, 250, 120, color = (255, 255, 255), size =
                    20, thickness = 2)

    #画十字交叉。
    img.draw_cross(60, 200, color = (255, 255, 255), size = 20,
                    thickness = 2)

    #写字符。
    img.draw_string(150, 200, "Hello 01Studio!", color = (255, 255,
                    255), scale = 2,mono_space = False)

    lcd.display(img)                # Display on LCD
```

```
    print(clock.fps())              # Note: MaixPy's Cam runs about half
as fast when connected
                                  # to the IDE. The FPS should increase
once disconnected.
```

- **实验结果：**

在 MaixPy IDE 中打开例程文件，点击运行。可以看到在图像缓冲区中画上了各种图形。

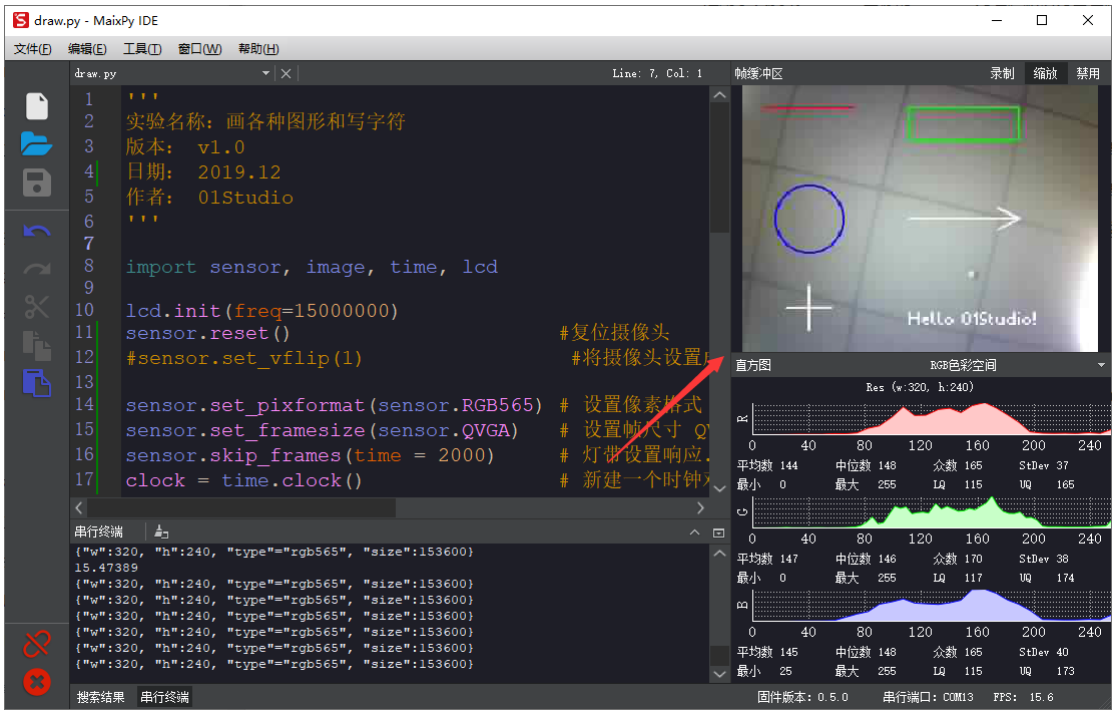


图 30-22 画图形

在 LCD 上同样能看到实验结果。由于 IDE 缓冲区图片经过压缩，所以 LCD 上的效果会更好一点。

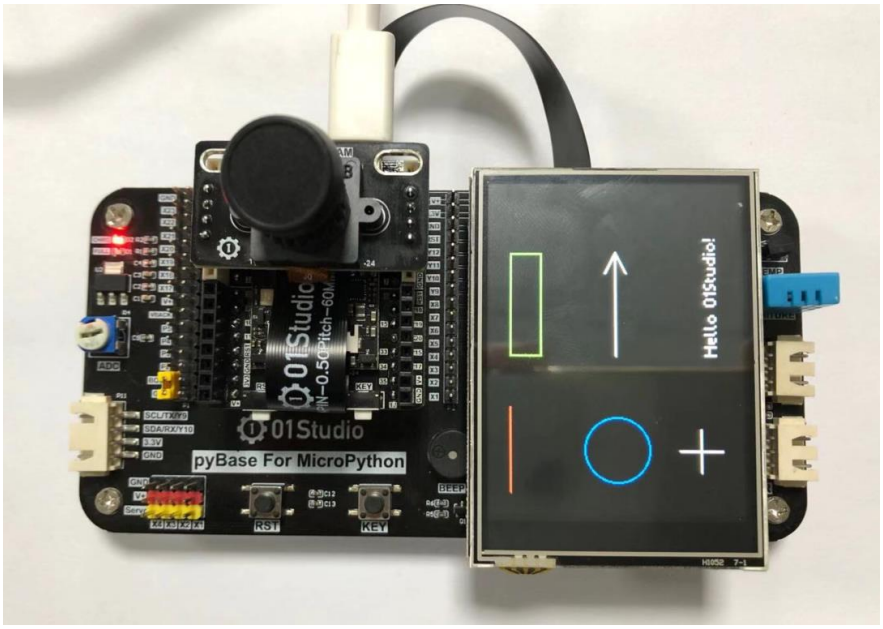


图 30-23 LCD 上显示结果

- **总结：**

画图形是很基础的功能，但在以后的实验中特别是指示识别内容时候会经常用到。

30.4 颜色识别

- **前言：**

我们活在一个色彩斑斓的世界里。本节我们来学习机器视觉中的颜色识别。我们会预先设定颜色阈值，如红、绿、蓝。这样 K210 摄像头采集图像后就能自动识别了。

- **实验平台：**

pyAI-K210 开发套件。

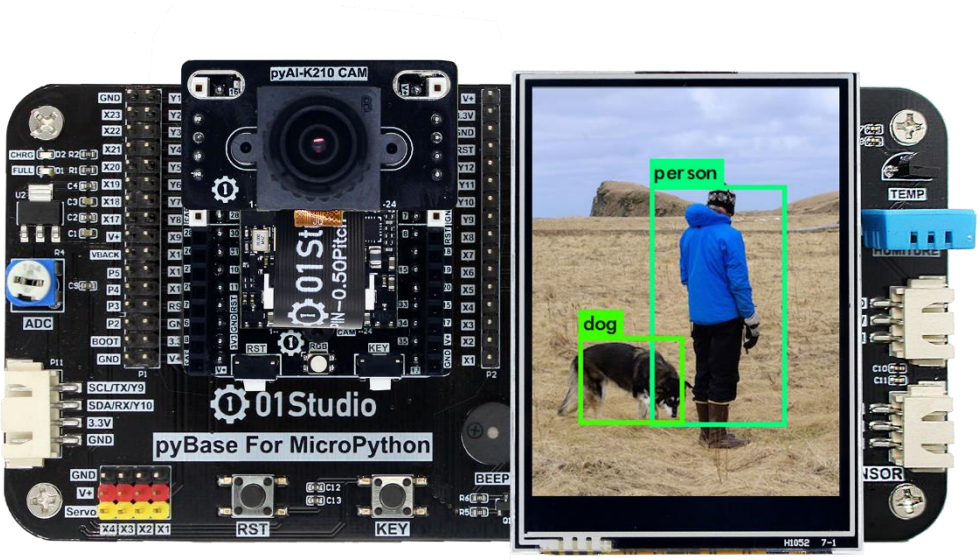


图 30-24 pyAI-K210 开发套件

- **实验目的：**

通过编程实现 pyAI-K210 识别程序预先设定的颜色色块，分别是红、绿、蓝三种颜色。

- **实验讲解：**

MaixPy 集成了 RGB565 颜色块识别 find_blobs 函数，主要是基于 LAB 颜色模型（每个颜色都是用一组 LAB 阈值表示，有兴趣的用户可以自行查阅相关模型资料）。其位于 image 模块下，因此我们直接将拍摄到的图片进行处理即可，那么我们像以往一样像看一下本实验相关对象和函数说明，具体如下：

构造函数
*image.find_blobs(*thresholds*[,* invert=False*[,* roi*[,* x_stride=2*[,* y_stride=1*[,* area_threshold=10*[,* pixels_threshold=10*[,* merge=False*[,* margin=0*[,* threshold_cb=None*[,* merge_cb=None*]]]]]]]]]]*)*
查找图像中指定的色块。返回 image.blog 对象列表； 【thresholds】 必须是元组列表。 [(lo, hi), (lo, hi), ..., (lo, hi)] 定义你想追踪的颜色范围。 对于灰度图像，每个元组需要包含两个值 - 最小灰度值和最大灰度值。 仅考虑落在这些阈值之间的像素区域。 对于 RGB565 图像，每个元组需要有六个值(l_lo，l_hi，a_lo，a_hi，b_lo，b_hi) - 分别是 LAB L，A 和 B 通道的最小值和最大值。 【area_threshold】若色块的边界框区域小于此参数值，则会被过滤掉； 【pixels_threshold】若色块的像素数量小于此参数值，则会被过滤掉； 【merge】若为 True,则合并所有没有被过滤的色块； 【margin】调整合并色块的边缘。 *更多参数说明请阅读官方文档： 文档链接：https://maixpy.sipeed.com/zh/libs/machine_vision/image.html
使用方法
以上函数返回 image.blob。
blob.rect()
返回一个矩形元组（x,y,w,h）,如色块边界。可以通过索引[0-3]来获得这些值。
blob.cx()
返回色块(int)的中心 x 位置。可以通过索引[5]来获得这个值。
blob.cy()
返回色块(int)的中心 y 位置。可以通过索引[6]来获得这个值。
*更多使用说明请阅读官方文档： https://maixpy.sipeed.com/zh/libs/machine_vision/image.html#blobrect

表 30-5 find_blobs 函数说明

了解了找色块函数应用方法后，我们可以理清一下编程思路，代码编写流程如下：

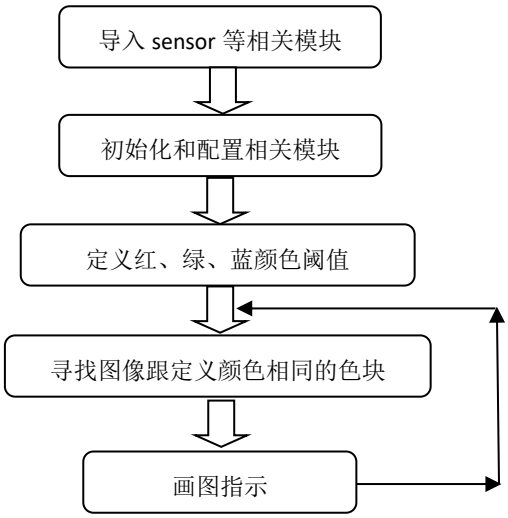


图 30-25 代码编写流程图

参考代码如下：

```
'''
实验名称：颜色识别
版本： v1.0
日期： 2019.12
作者： 01Studio
实验目的：单个颜色识别
'''

import sensor,lcd,time

#摄像头初始化
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.set_vflip(1) #后置模式，所见即所得
```

```
#lcd 初始化
lcd.init()

clock=time.clock()

# 颜色识别阈值 (L Min, L Max, A Min, A Max, B Min, B Max) LAB 模型
# 下面的阈值元组是用来识别 红、绿、蓝三种颜色，当然你也可以调整让识别变得更好。
thresholds = [(30, 100, 15, 127, 15, 127), # 红色阈值
              (30, 100, -64, -8, -32, 32), # 绿色阈值
              (0, 30, 0, 64, -128, -20)] # 蓝色阈值

while True:

    clock.tick()

    img=sensor.snapshot()

    blobs = img.find_blobs([thresholds[2]]) # 0,1,2 分别表示红，绿，蓝色。
    if blobs:
        for b in blobs:
            tmp=img.draw_rectangle(b[0:4])
            tmp=img.draw_cross(b[5], b[6])

    lcd.display(img)     #LCD 显示图片
    print(clock.fps())   #打印 FPS
```

- **实验结果：**

在 IDE 中运行代码，代码默认检测的是蓝色，用户可以自行修改 find_blobs() 参数的阈值数组编号来切换识别颜色，如下：

蓝色识别：

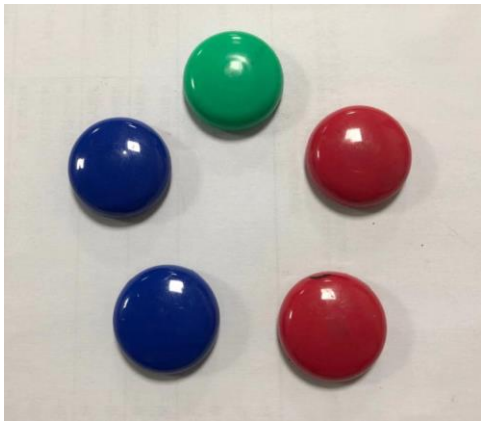




图 30-26 蓝色识别原图（左）和实验图片（右）

- 总结：

本节学习了通过 MicroPython 编程在 pyAI-K210 上实现单种颜色识别。本实验主要是基于 LAB 颜色模型来判断。有兴趣的小伙伴可以自行查阅 LAB 模块相关资料，再结合打印 threshold 查看其元组数据来深入学习，还有就是可以找多种不规则形状的颜色物体对比学习。

30.5 二维码识别

- **前言：**

相信大家都知道二维码了，特别是在扫描支付越来越流行的今天，二维码的应用非常广泛。今天我们就来学习如何使用 pyAI-K210 开发套件实现二维码信息识别

- **实验平台：**

pyAI-K210 开发套件。

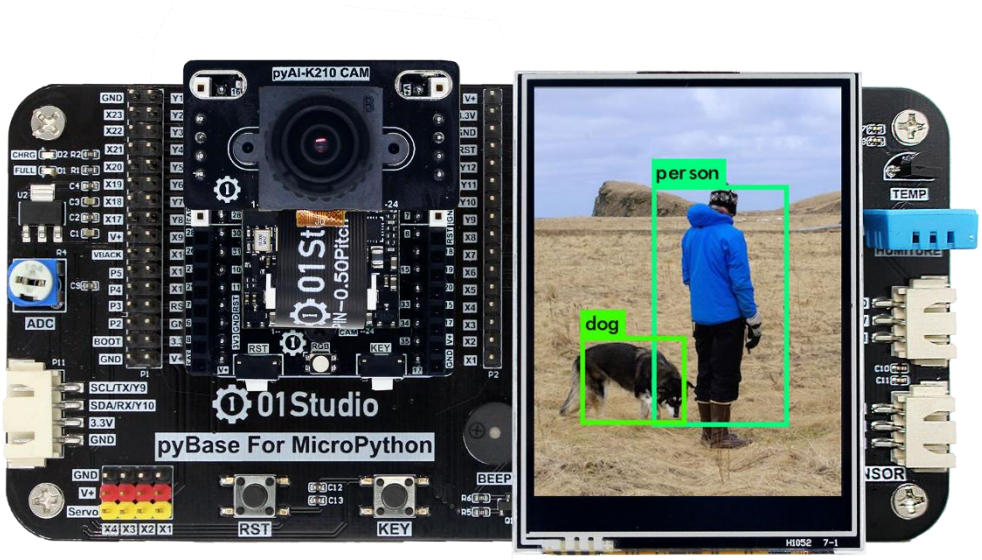


图 30-27 pyAI-K210 开发套件

- **实验目的：**

编程实现二维码识别，并将识别到的信息打印出来。

- **实验讲解：**

二维码又称二维条码，常见的二维码为 QR Code，QR 全称 Quick Response，是一个近几年来移动设备上超流行的一种编码方式，它比传统的 Bar Code 条形码能存更多的信息，也能表示更多的数据类型。

二维条码/二维码（2-dimensional bar code）是用某种特定的几何图形按一定规律在平面（二维方向上）分布的、黑白相间的、记录数据符号信息的图形；在

代码编制上巧妙地利用构成计算机内部逻辑基础的“0”、“1”比特流的概念，使用若干个与二进制相对应的几何形体来表示文字数值信息，通过图象输入设备或光电扫描设备自动识读以实现信息自动处理：它具有条码技术的一些共性：每种码制有其特定的字符集；每个字符占有一定的宽度；具有一定的校验功能等。同时还具有对不同行的信息自动识别功能、及处理图形旋转变化点。

而对于 pyAI-K210 而言，直接使用 MicroPython 中的 find_qrcodes()即可获取摄像头采集图像中二维码的相关信息。具体说明如下：

构造函数
`image.find_qrcodes([roi])`
查找 roi 区域内的所有二维码并返回一个 image.qrcode 的对象列表。 *更多参数说明请阅读官方文档： https://maixpy.sipeed.com/zh/libs/machine_vision/image.html?h=find_qrcode
使用方法
以上函数返回 `image.qrcode` 对象列表。
`qrcode.rect()`
返回一个矩形元组（`x,y,w,h`）；
`qrcode.payload()`
返回二维码字符串信息。可以通过索引`[4]`来获得这个值。
`qrcode.verison()`
返回二维码版本号。
*更多使用说明请阅读官方文档： https://maixpy.sipeed.com/zh/libs/machine_vision/image.html#qrcodecorners

表 30-6 二维码对象

从上表可以看到，使用 MicroPython 编程我们只需要简单地调用 find_qrcodes()函数，对得到的结果再进行处理即可，非常方便。代码编写流程如下图所示：

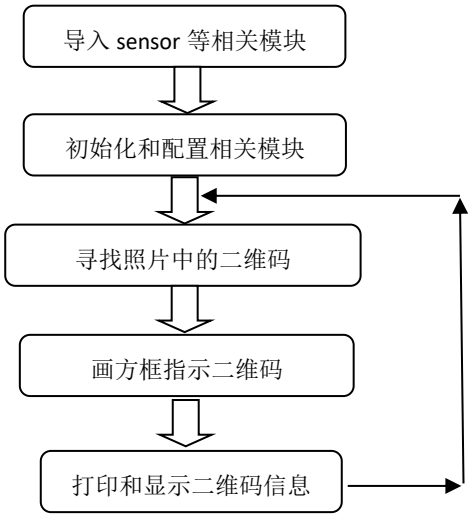


图 30-28 代码编写流程图

参考代码如下：

```
#实验名称：二维码识别
#版本：v1.0
#日期：2019.12
#翻译和注释：01Studio

import sensor,lcd,time

#摄像头模块初始化
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.set_vflip(1)     #后置模式
sensor.skip_frames(30)

#lcd 初始化
```

```
lcd.init()

clock = time.clock()

while True:

    clock.tick()

    img = sensor.snapshot()
    res = img.find_qrcodes() #寻找二维码

    if len(res) > 0: #在图片和终端显示二维码信息
        img.draw_rectangle(res[0].rect())
        img.draw_string(2,2, res[0].payload(), color=(0,128,0), scale=2)
        print(res[0].payload())

    lcd.display(img)
    print(clock.fps())
```

为了更好地识别，图像上二维码需比较平展。另外也可以使用无畸变镜头，降低因镜头因素导致的图片畸变问题，提高二维码识别的准确率。

- **实验结果：**

运行程序，打开一个二维码图片（暂时不支持太奇葩的二维码）。将 pyAI-K210 摄像头设置为后置模式，摄像头正对二维码，识别成功后可以看到图片出现方框已经在串口终端打印出二维码信息。



图 30-29 成功识别二维码

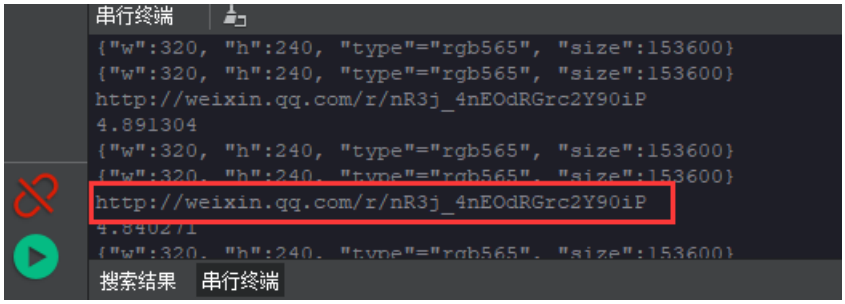


图 30-30 串口终端打印二维码信息

- **总结：**

二维码是日常生活应用非常广泛的东西，有了本节实验技能，我们就可以轻松打造一个属于自己的二维码扫描仪了。

30.6 人脸检测

- **前言：**

人脸检测，简单来说就是告诉你将一幅图片中哪些是人脸。今天我们来学习一下如何通过 MicroPython 编程快速实现人脸检测。

- **实验平台：**

pyAI-K210 开发套件，配 SD 卡放模型文件。

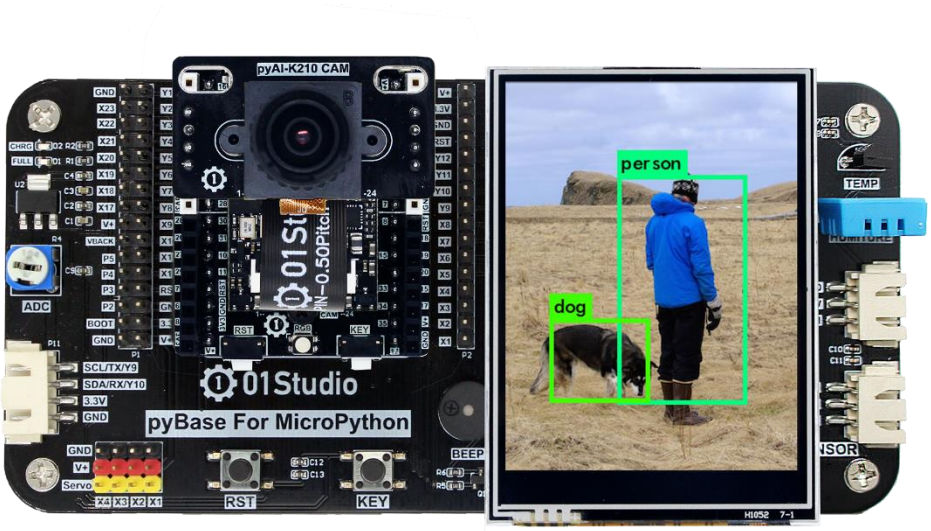


图 30-31 pyAI-K210 开发套件

- **实验目的：**

将摄像头拍摄到的画面中的人脸用矩形框表示出来。

- **实验讲解：**

我们来简单介绍一下 K210 的 KPU。KPU 是 K210 内部一个神经网络处理器，它可以在低功耗的情况下实现卷积神经网络计算，实时获取被检测目标的大小、坐标和种类，对人脸或者物体进行检测和分类。

KPU 具备以下几个特点：

- 支持主流训练框架按照特定限制规则训练出来的定点化模型
- 对网络层数无直接限制，支持每层卷积神经网络参数单独配置，包括输入输出通道数目、输入输 出行宽列高

- 支持两种卷积内核 1x1 和 3x3
- 支持任意形式的激活函数
- 实时工作时最大支持神经网络参数大小为 5.5MiB 到 5.9MiB
- 非实时工作时最大支持网络参数大小为（Flash 容量-软件体积）

简单来说就是 KPU 能加载和运行各种现成的 AI 算法模型，实现各种机器视觉等功能。

MaixPy 中人脸识别本质是目标检测，主要通过在 K210 的 KPU 上跑 YOLO（You Only Look Once）目标检测算法来实现。我们来看一下 KPU 在 MaixPy 下的用法。

构造函数
`import KPU as kpu`
常用的 KPU 模块导入方法。
使用方法
`kpu.load(offset or file_path)`
加载模型。 【offset】模型存放在 flash 的偏移量，如 0x300000; 【file_path】模型在文件系统为文件名，如“xxx.kmodel”
`kpu.init_yolo2(kpu_net,threshold,nms_value,anchor_num,anchor)`
初始化 yolo2 网络； 【kpu_net】kpu 网络对象； 【threshold】概率阈值； 【nms_value】box_iou 门限； 【anchor_num】描点数； 【anchor】描点参数与模型参数一致。
`kpu.run_yolo2(kpu_net,image)`
运行 yolo2 网络； 【kpu_net】从 kpu_load()中返回的网络对象；

【image】从 sensor 中采集到的图像
kpu.deinit(kpu_net)
反初始化。 【kpu_net】kpu 网络对象；

表 30-7　kpu 运行 yolo2 算法

从上表可以看到通过 KPU 模块直接加载 YOLO2 网络，再结合人脸检测模型来实现人脸识别。具体编程思路如下：

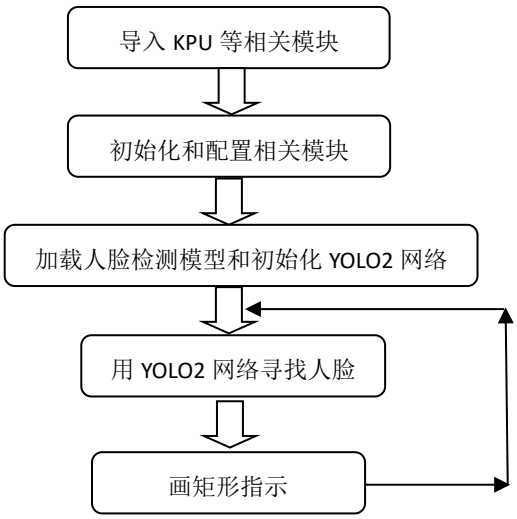


图 30-32 代码编写流程图

代码编写流程如下：

```
#实验名称：人脸检测
#翻译和注释：01Studio

import sensor,lcd,time
import KPU as kpu

#设置摄像头
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
```

```
#sensor.set_vflip(1)    #设置摄像头后置

lcd.init() #LCD 初始化

clock = time.clock()

#需要将模型（face.kfpkg）烧写到 flash 的 0x300000 位置
#task = kpu.load(0x300000)
#将模型放在 SD 卡中。
task = kpu.load("/sd/facedetect.kmodel") #模型 SD 卡上

#模型描参数
anchor = (1.889, 2.5245, 2.9465, 3.94056, 3.99987, 5.3658, 5.155437,
          6.92275, 6.718375, 9.01025)

#初始化 yolo2 网络
a = kpu.init_yolo2(task, 0.5, 0.3, 5, anchor)

while(True):
    clock.tick()
    img = sensor.snapshot()
    code = kpu.run_yolo2(task, img) #运行 yolo2 网络

    #识别到人脸就画矩形表示
    if code:
        for i in code:
            print(i)
            b = img.draw_rectangle(i.rect())

    #LCD 显示
```

```
    lcd.display(img)


    print(clock.fps())   #打印 FPS
```

有了代码后我们还需要将模型放在文件系统中。这里介绍 2 个方法：

方法一：将模型放在 SD 卡中。

在本节示例程序路径中可以看到有 1 个件夹 **face_model_at_0x300000**，将里面的 **facedetect.kmodel** 文件移动到 SD 卡，运行上述代码即可。



图 30-33 模型原文文件

方法二：将模型烧录到 K210 的 Flash 中。

打开本节示例程序路径的件夹 **face_model_at_0x300000** 里面的 **flash-list.json** 文件，内容如下（告诉烧录软件烧写地址和文件名）：

```
{
  "version": "0.1.0",
  "files": [
    {
      "address": 0x00300000,
      "bin": "facedetect.kmodel",
      "sha256Prefix": false
    }
  ]
}
```

接下来直接将 kmodel 和 json 这两个文件用 zip 方式压缩(不要用文件夹)，然后将 zip 后缀名改成 kfpkg，得到一个可以用 K210 固件烧录工具烧录的文件。

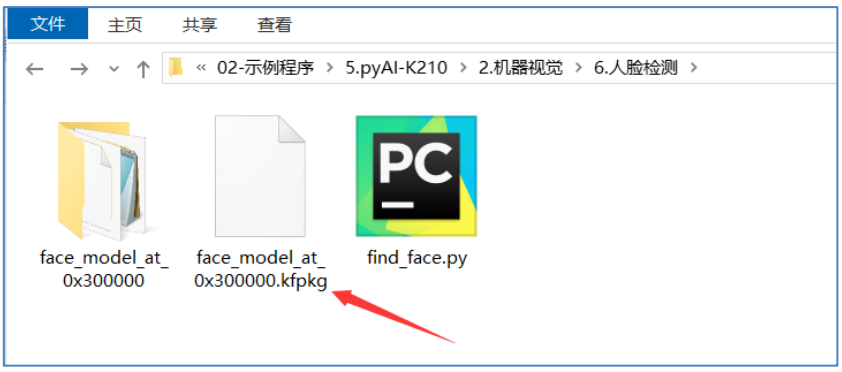


图 30-34

再使用 K210 固件烧录工具烧录直接烧录该文件即可，烧录软件会根据上述的 json 文件自动调整烧录地址，无需再次填写。。

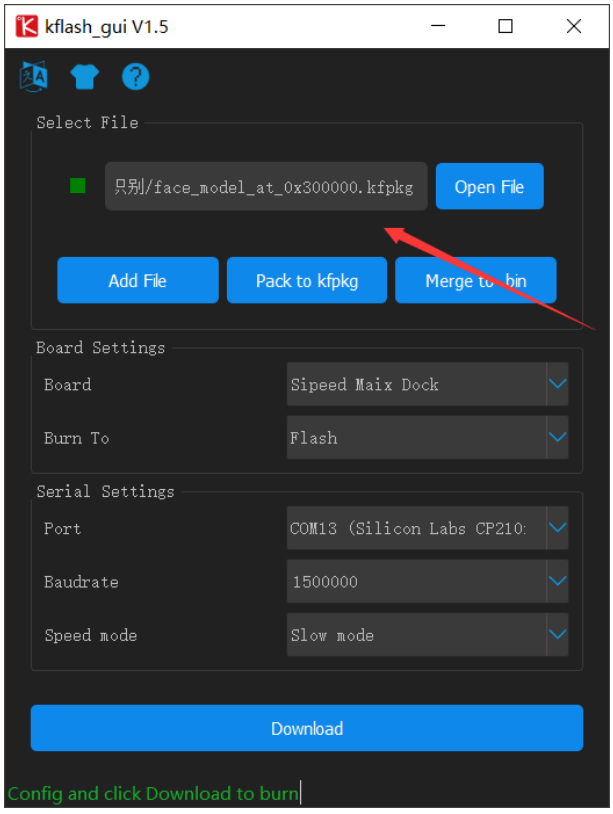


图 30-35

- **实验结果：**

运行代码，将摄像头正对自己，可以看到将自己的脸检测出来。系统默认摄像头是前置。注要将开发板竖起来，即 LCD 横屏显示。

当我们去识别图片时候，可以将摄像头设置成后置，sensor 初始化时增加以下代码：(LCD 装在 pyAI-K210 核心板背面，横屏测试。)

```
sensor.set_vflip(1)     #设置摄像头后置
```

我们用图片演示一下单个人脸和多个人脸检测的场景。

单个人脸检测：

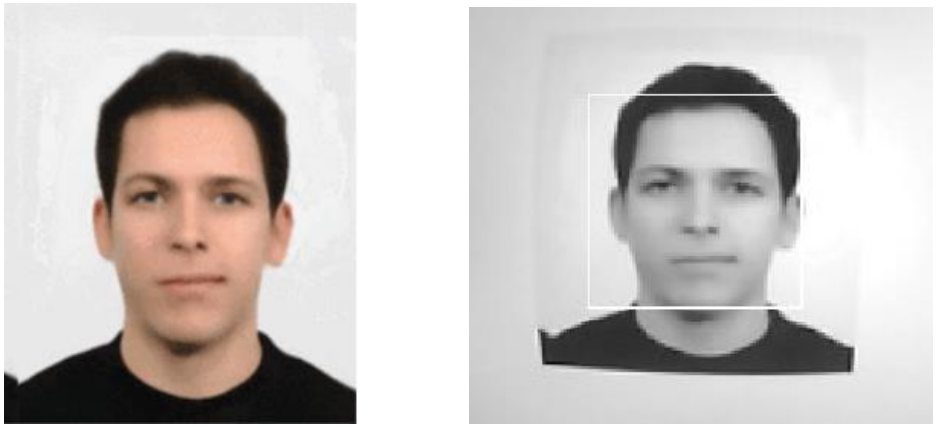


图 30-36 单个人脸检测 原图（左）和实验图片（右）

多个人脸识别：



图 30-37 多个人脸检测原图

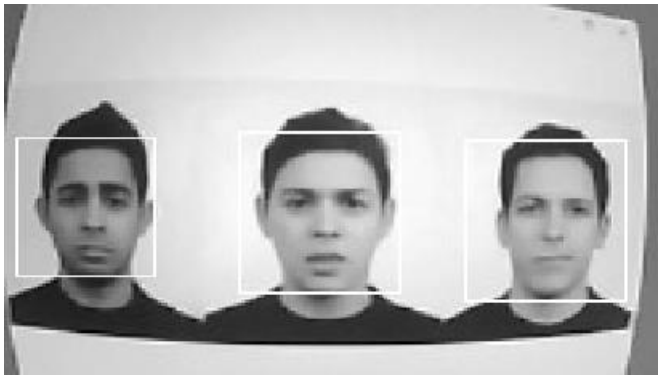


图 30-38 多个人脸检测实验图片

- **总结：**

本节学习了人脸检测，可以看到 pyAI-K210 通过 KPU+YOLO2+face 模型轻松实现人脸检测，而且检测的准确率非常高，也就是结合 MicroPython 编程我们轻松完成了实验。而且本实验支持单个和多个人脸同时检测，是机器视觉中非常有代表性的实验。

30.7 物体识别

- **前言：**

物体识别，是机器视觉里面非常典型的应用。要实现的就是将一幅图片里面的各种物体检测出来，然后跟已知模型做比较从而判断物体是什么。今天我们来学习一下如何通过 MicroPython 编程快速实现物体识别。

- **实验平台：**

pyAI-K210 开发套件，配 SD 卡放模型文件。

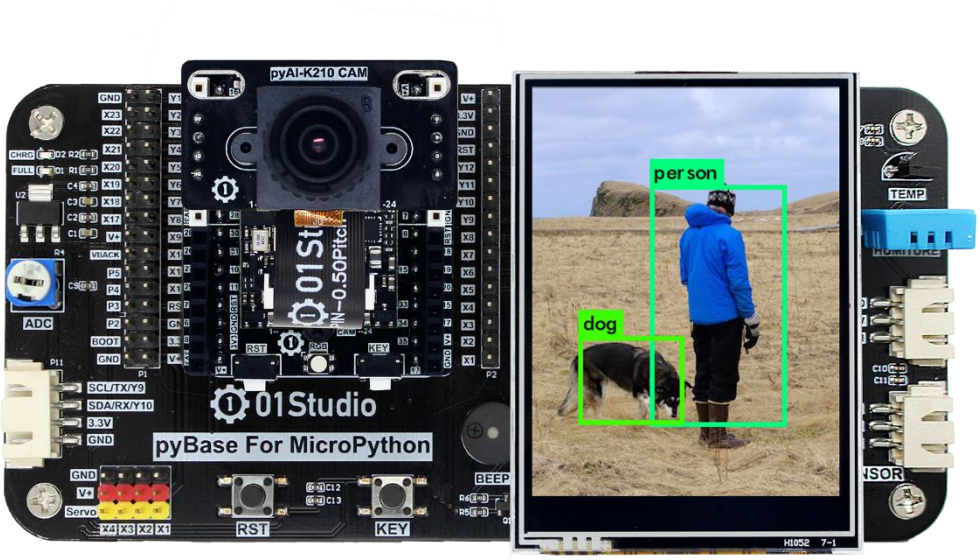


图 30-39 pyAI-K210 开发套件

- **实验目的：**

物体识别并显示名称和可信度（概率）。

- **实验讲解：**

在上一节人脸检测章节我们已经介绍过 KPU 的用法，这里不再重复。本实验还是使用到 YOLO2 网络，结合 20class 模型（20 种物体分类模型）来识别图像中的物体。下面重温一下 KPU 的用法：

构造函数
import KPU as kpu
常用的 KPU 模块导入方法。
使用方法
kpu.load(offset or file_path)
加载模型。 【offset】模型存放在 flash 的偏移量，如 0x300000; 【file_path】模型在文件系统为文件名，如“xxx.kmodel”
kpu.init_yolo2(kpu_net,threshold,nms_value,anchor_num,anchor)
初始化 yolo2 网络； 【kpu_net】kpu 网络对象； 【threshold】概率阈值； 【nms_value】box_iou 门限； 【anchor_num】描点数； 【anchor】描点参数与模型参数一致。
kpu.run_yolo2(kpu_net,image)
运行 yolo2 网络； 【kpu_net】从 kpu_load()中返回的网络对象； 【image】从 sensor 中采集到的图像
kpu.deinit(kpu_net)
反初始化。 【kpu_net】kpu 网络对象；

表 30-8　kpu 运行 yolo2 算法

从上表可以看到通过 KPU 模块直接加载 YOLO2 网络，再结合 20class 物体识别模型来实现物体识别。具体编程思路如下：

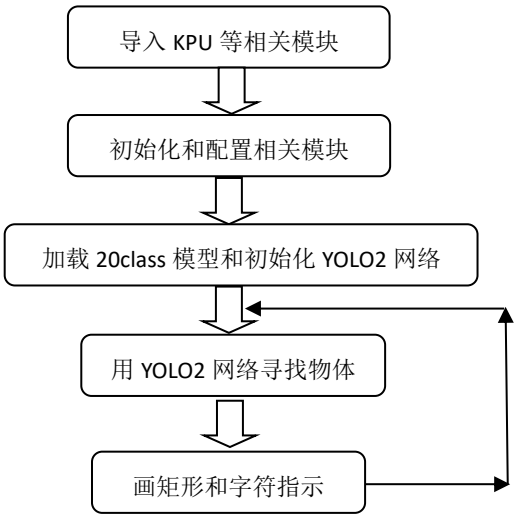


图 30-40 代码编写流程图

代码编写流程如下：

```
#实验名称：物品检测
#翻译和注释：01Studio
#实验目的：使用 20class 模型识别 20 种物体

import sensor,image,lcd,time
import KPU as kpu

#摄像头初始化
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.set_vflip(1)  #摄像头后置方式

lcd.init() #LCD 初始化

clock = time.clock()

#模型分类，按照 20class 顺序
```

```
classes = ['aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus',
           'car', 'cat', 'chair', 'cow', 'diningtable', 'dog', 'horse',
           'motorbike', 'person', 'pottedplant', 'sheep', 'sofa',
           'train', 'tvmonitor']

#下面语句需要将模型（20class.kfpkg）烧写到 flash 的 0x500000 位置
#task = kpu.load(0x500000)

#将模型放在 SD 卡中。
task = kpu.load("/sd/20class.kmodel") #模型 SD 卡上

#网络参数
anchor = (1.889, 2.5245, 2.9465, 3.94056, 3.99987, 5.3658, 5.155437,
            6.92275, 6.718375, 9.01025)

#初始化 yolo2 网络，识别可信概率为 0.7（70%）
a = kpu.init_yolo2(task, 0.7, 0.3, 5, anchor)

while(True):

    clock.tick()

    img = sensor.snapshot()
    code = kpu.run_yolo2(task, img) #运行 yolo2 网络

    if code:
        for i in code:
            a=img.draw_rectangle(i.rect())
            a = lcd.display(img)
```

```
            lcd.draw_string(i.x(), i.y(), classes[i.classid()], lcd.RED,
                            lcd.WHITE)
            lcd.draw_string(i.x(), i.y()+12, '%f1.3'%i.value(), lcd.RED,
                           lcd.WHITE)
    else:
        a = lcd.display(img)

    print(clock.fps()) #打印 FPS
```

我们将示例程序中的 20class.kmodel 模型文件拷贝到 SD 卡中。（**上一节人脸检测章节已经介绍过如何将模型拷贝到 SD 卡或者 K210 的 Flash 里面，这里不再重复。**）

- **实验结果：**

运行程序，准备 20class 里面相关的物体图片，可以看到 pyAI-K210 可以轻易地将相关物体识别出来。



图 30-41 被识别物体自行车

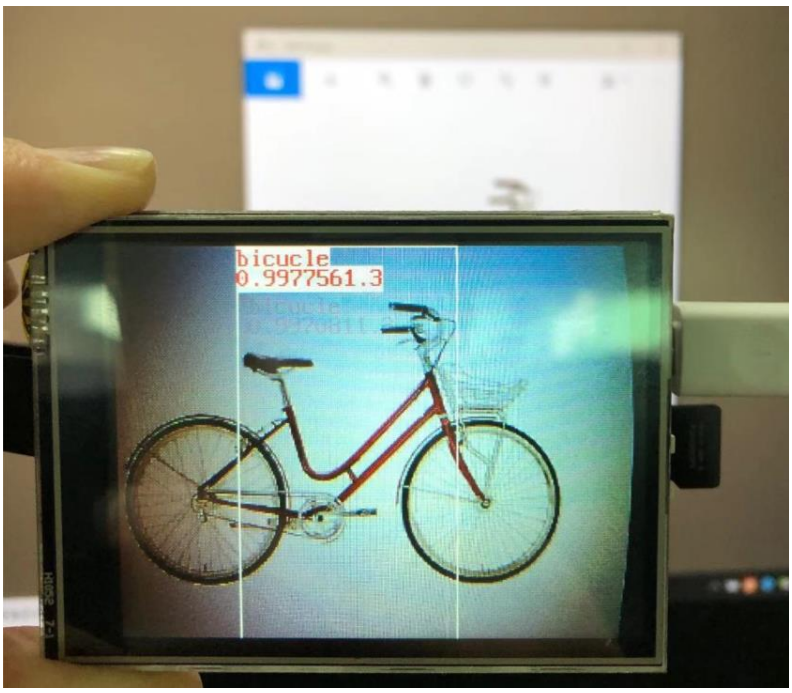


图 30-42 识别结果

- **总结：**

本节学习了物体识别，可以看到 pyAI-K210 通过 KPU+YOLO2+20class 模型轻松实现特定物体识别，而且检测的准确率非常高，也就是结合 MicroPython 编程我们轻松完成了实验。

30.8 图片拍摄

- **前言：**

普通图片拍摄是最基础的功能，pyAI-K210 图像算法处理的前提就是实时连续采集多帧图片，然后进行分析。这个功能可以将 pyAI-K210 变成一个随身照相机。

- **实验平台：**

pyAI-K210 开发套件。另外需要配备 SD 卡进行图片储存。

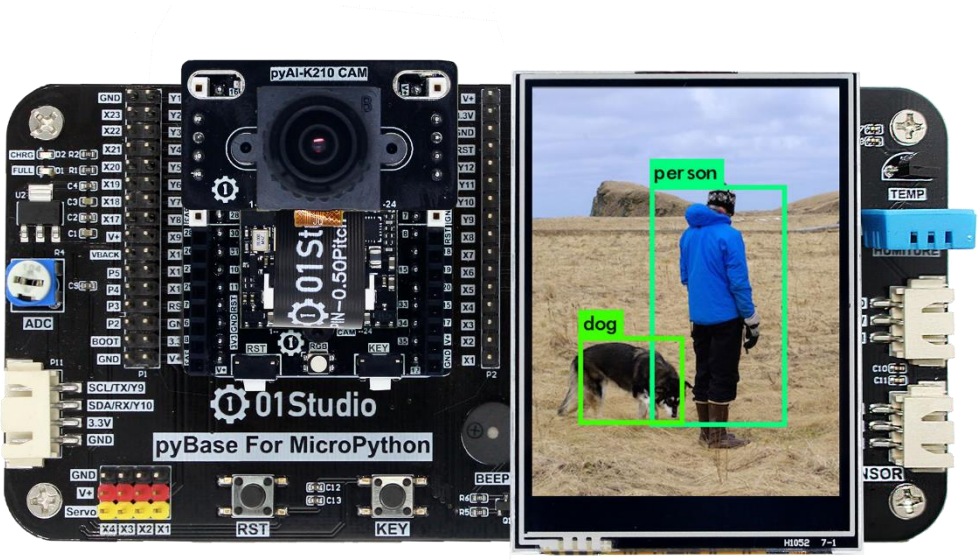


图 30-43 pyAI-K210 开发套件

- **实验目的：**

通过编程实现拍照并保存。

- **实验讲解：**

我们在前面摄像头应用章节已经学习过拍摄是使用 image=sensor.snapshot()函数模块，那么我们只需要学会将图片保存即可。保存也是可以直接使用 image 下的 save 模块，具体如下：

构造函数
img=sensor.snapshot()
通过拍摄创建图像 img
使用方法
image.save(*path*[, *roi*[, *quality=50*]])
保存图片。 path：保存路径； roi:指定保存区域(x, y, w, h)，默认全图保存； quality:仅针对 JPEG 格式的质量控制，有效值为 0-100。

表 30-9 摄像头对象 snapshot

掌握了拍照和保存功能，我们就可以编程实现了，例程编程代码流程图如下：

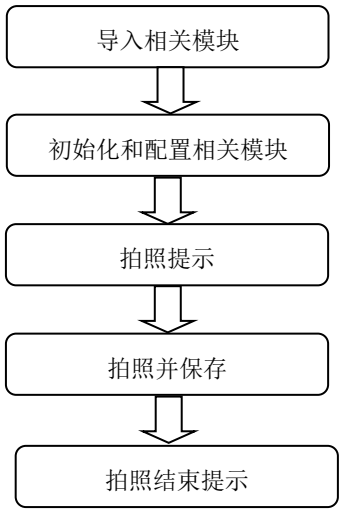


图 30-44 代码编写流程

参考代码如下：

```
# 普通拍照例程

# 提示：你需要插入 SD 卡来运行这个例程.

#作者：01Studio
```

```
    import sensor, lcd, image
    from Maix import GPIO
    from fpioa_manager import fm

#配置 LED 蓝、红引脚
fm.register(12, fm.fpioa.GPIO0,force=True)
fm.register(14, fm.fpioa.GPIO1,force=True)

LED_B = GPIO(GPIO.GPIO0, GPIO.OUT,value=1) #构建 LED 对象
LED_R = GPIO(GPIO.GPIO1, GPIO.OUT,value=1) #构建 LED 对象

#摄像头初始化
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(40)

lcd.init()  #LCD 初始化

#红灯亮提示拍照开始
LED_R.value(0)
sensor.skip_frames(time = 2000) # 给 2 秒时间用户准备.
LED_R.value(1)

#蓝灯亮提示正在拍照
LED_B.value(0)
print("You're on camera!")

# 拍摄并保存相关文件，也可以用"example.bmp"或其它文件方式。
sensor.snapshot().save("/sd/example.jpg")
```

```
LED_B.value(1) #l 蓝灯灭提示拍照完成

    lcd.display(image.Image("/sd/example.jpg")) #LCD 显示照片
```

- **实验结果：**

运行上述代码。可以看到串口提示完成了一次完整的拍照过程。最终并将图片显示在 LCD 上。



图 30-45 实验图片

将 SD 卡通过读卡器读取，可以看到 SD 卡里面、出现了刚刚拍照保存的 example.jpg 文件。

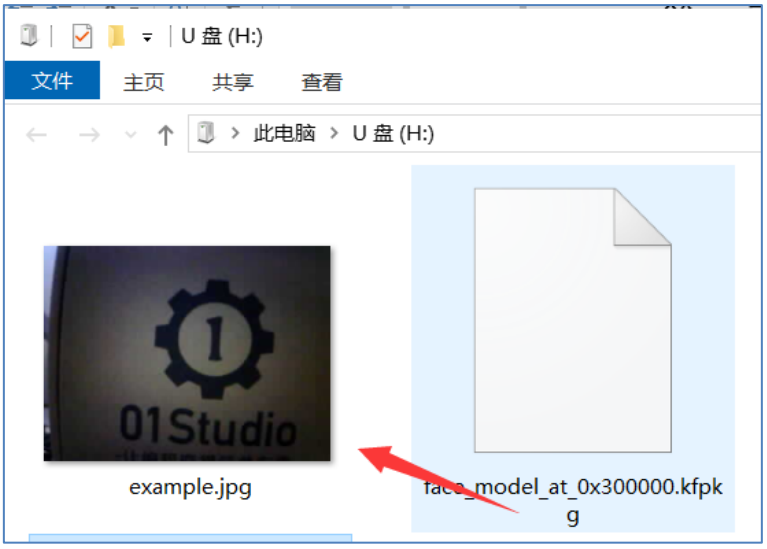


图 30-46 刚刚拍摄到的图片文件。

- **总结：**

可以看到 pyAI-K210 通过 micropython 编程可以非常容易使用起来。本节结合基础例程的按键可以打造按键照相机，有兴趣的用户可以自行编程实现。

30.9 视频录制

- **前言：**

AVI 是非常常见的视频格式，今天我们来学习一下使用 pyAI-K210 开发板来录制 avi 格式视频。

- **实验平台：**

pyAI-K210 开发套件。另外需要配备 SD 卡进行视频储存。

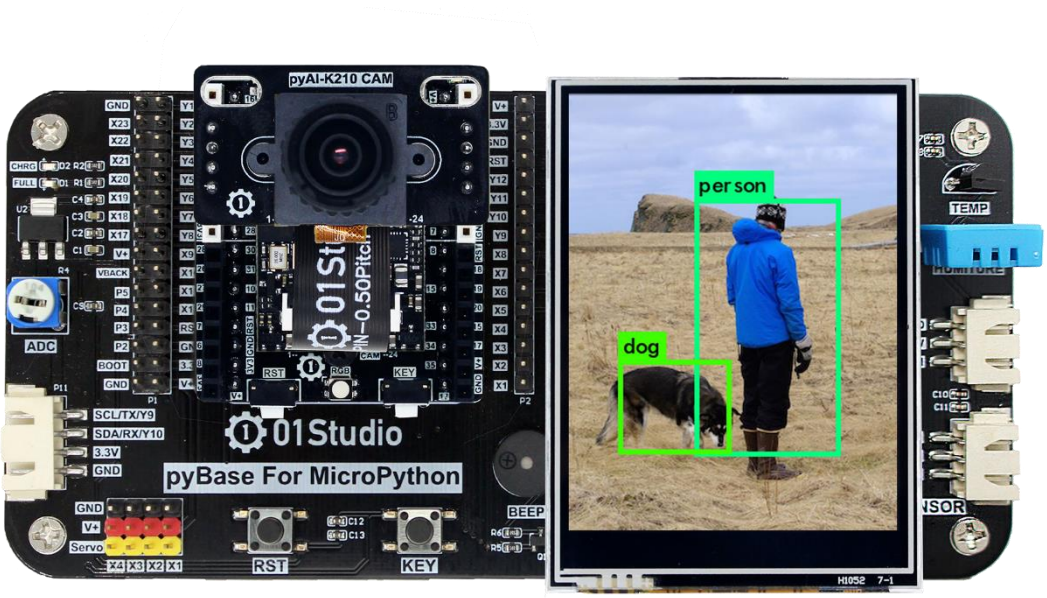


图 30-47 pyAI-K210 开发套件

- **实验目的：**

录制 AVI 视频。

- **实验讲解：**

pyAI-K210 使用的 MaixPy 集成了 vedio 视频模块，也就是通过 MicroPython 编程可以轻松实现录制视频功能，我们来看看 vedio 对象：

构造函数
import vedio v=vedio.open((path, record=False, interval=100000, quality=50, width=320, height=240, audio=False, sample_rate=44100, channels=1)))
播放或录制视频文件。 【path】文件路径，比如：/sd/badapple.avi； 【record】=True 表示视频录制，=False 表示视频播放； 【interval】录制帧间隔，单位是微妙；FPS=1000000/interval，默认值是 100000，即 FPS 默认是 10（每秒 10 帧）； 【quality】jpeg 压缩质量（%），默认 50； 【width】录制屏幕宽度，默认 320； 【height】录制屏幕高度，默认 240； 【audio】是否录制音频，默认 False; 【sample_rate】录制音频采样率，默认 44100（44.1k）； 【channels】录制音频声道数，默认 1，即单声道。
使用方法
v.play()
播放视频；
v.volume([value])
设置音量值。 【value】0-100;
v.revord ()
录制音视频；
v.revord_finish ()
停止录制；
*更多使用说明请阅读官方文档： https://maixpy.sipeed.com/zh/libs/machine_vision/video.html

表 30-10　vedio 对象

学习了 vedio 的相关用法后，我们整理思路，代码编写流程如下：

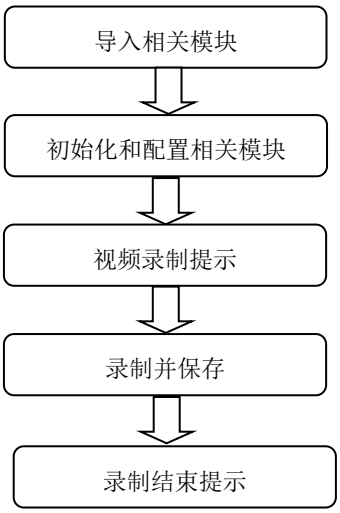


图 30-48 代码编写流程

参考代码如下：

```
#视频录制示例
#
# 提示：你需要插入 SD 卡来运行此程序.
#
#翻译和注释：01Studio

import video, sensor, image, lcd, time

#摄像头初始化
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.set_vflip(1)   #后置拍摄模式
sensor.skip_frames(30)

#LCD 初始化
lcd.init()
```

```
#指定录制文件路径和文件名
v = video.open("/sd/example.avi", record=1)

i = 0  #计算录制帧数

while True:

    tim = time.ticks_ms()
    img = sensor.snapshot()

    lcd.display(img)
    img_len = v.record(img) #img_len 为返回的录制帧长度。

    print("record",time.ticks_ms() - tim) #打印录制的每帧间隔

    #录制 100 帧,每帧默认 100ms，即 10 秒视频。
    i += 1
    if i > 100:
        break

v.record_finish() #停止录制
print("finish") #录制完成提示
```

- **实验结果：**

  运行代码，将摄像头对准要拍摄的物体，点击运行。

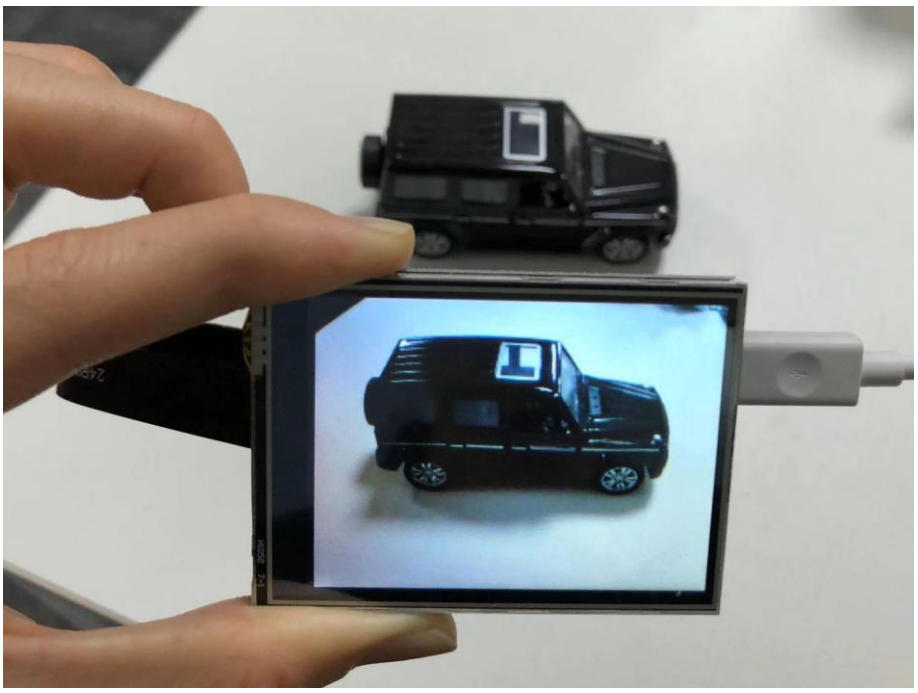


图 30-49 对准拍摄物体

拍摄完成后，取下 SD 卡，用读卡器打开。看到文件名为：“example.avi”。点击打开就可以看到录制的视频。



图 30-50 avi 视频文件

（注意：有些时候拍摄视频后电脑无法播放文件，这里有个 Bug 待官方修复，目前可以在同一次操作中重复运行代码 2 次，让后面的文件覆盖前面的 avi 文件可解决此问题！）

- **总结：**

本节学习了视频录制，结合基础例程的按键可以打造按键录像机。有兴趣的用户可以自行编程实现。

第31章　机器听觉

31.1　声音频率识别（FFT）

- **前言：**

生活中我们经常会遇到以下场景，当一群人在说话的时候我们能听到多个声音，甚至嘈杂，但我们大脑却可以快速分辨出不同声音，如男声、女声或者是某人的声音，这些声音是同时发生的，因此我们大脑对声音进行分析，区别出声音不同的特点。一个重要的识别特征就是频率，比如男声通常频率低，女声频率高。

今天我们就通过进行频率识别实验。

- **开发平台：**

pyAI-K210 开发套件+麦克风模块。

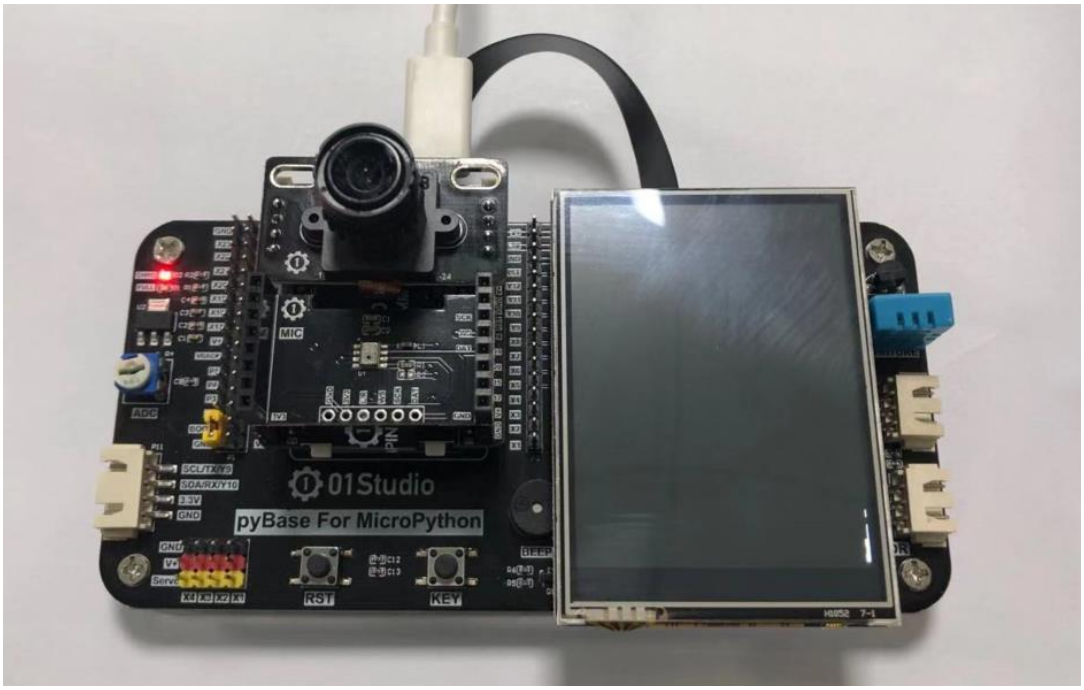


图 31-1 pyAI-K210 开发套件

- **实验目的：**

通过麦克风模块实现声音获取，计算其频率，并在 LCD 上显示。

- **实验讲解：**

本节用到的一个知识是数学中常用的公式，快速傅里叶变换（FFT），傅里叶

的一个基础理论就是认为每一个信号都可以分解成 N 个不同的周期信号叠加而成。

一般情况下我们在一段时间内听到一段声音，声音中夹杂着不同频率，我们可以建立坐标，横坐标为时间 t，纵坐标为不同频率的分解，这便是时域。

而以频率 f 为横坐标，幅值为纵坐标建立的坐标，便展示了频域。具体如下图所示：（有关 FFT 更详细内容，可以自行科普。）

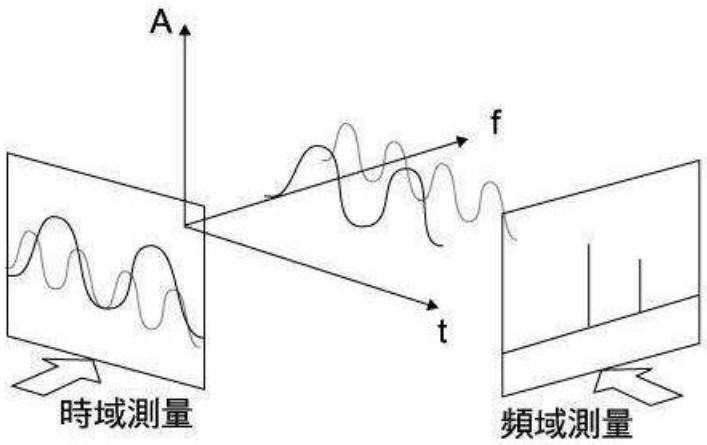


图 31-2 时域与频域关系

MaixPy 集成了 FFT 运算的 MicroPython 库，对用户来说我们使用 pyAI-K210 开发套件和数字麦克风模块即可编程实现该功能。

先来看一下本实验使用的是 01Studio 配套的数字麦克风模块，数字麦克风模块使用 I2S 总线通信，I2S(Inter—IC Sound)总线, 又称集成电路内置音频总线，是飞利浦公司为数字音频设备之间的音频数据传输而制定的一种总线标准，该总线专门用于音频设备之间的数据传输，广泛应用于各种多媒体系统。它采用了沿独立的导线传输时钟与数据信号的设计，通过将数据和时钟信号分离，避免了因时差诱发的失真，为用户节省了购买抵抗音频抖动的专业设备的费用。

模块直接插到 pyAI-k210 黑色排母接口即可，其与 K210 引脚连接关系如下：

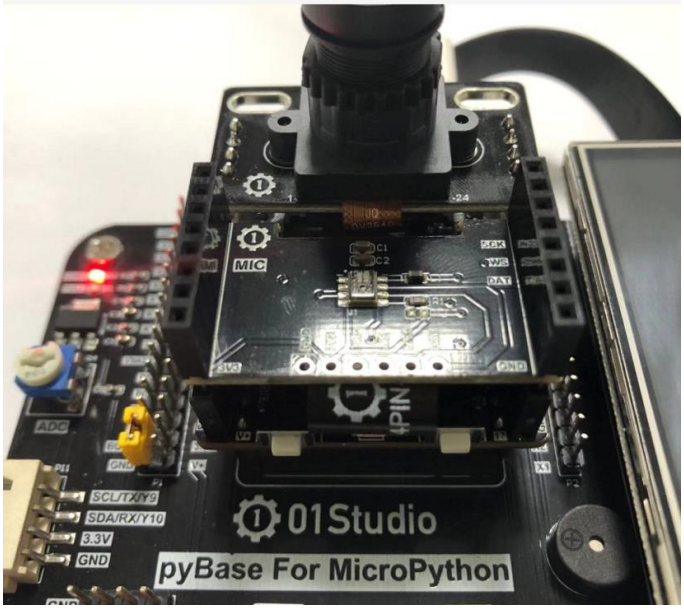


图 31-3 数字麦克风模块连接方式

```
#K210 引脚连接说明
SCK -- > IO18
WS -- > IO19
DAT -- > IO20
```

I2S 模块主要用于驱动 I2S 设备，k210 一共有 3 个 I2S 设备，每个设备一共有 4 个通道，在使用前需要对引脚进行映射管理,模块说明如下：

构造函数
from Maix import I2S i2s_dev = I2S(device_num)
导入 I2S 模块; 【device_num】共 3 个设备：DEVICE_0, DEVICE_1, DEVICE_2;
使用方法
i2s_dev.channel_config(channel,mode,resolution,cycles,align_mode)
通道配置; 【channel】共 4 个：CHANNEL_0-CHANNEL_3; 【mode】通道传输模式，有接收（RECEIVER）和发送(TRANSMITTER)模式，录音为接收，播放为发送;

【resolution】通道分辨率； 【cycles】单个数据时钟数； 【align_mode】通道对齐模式
i2s_dev.set_sample_rate(sample_rate)
设置采样率。
audio = i2s_dev.record(points)
采集音频； 【points】一次采集音频点数；
i2s_dev.play(audio)
播放音频； 【audio】播放音频

表 31-1 I2S 对象

FFT 运行模块说明如下：

构造函数
import FFT FFT_res = FFT.run(data, points, shift)
FFT 运算模块； 【data】输入的时域数据，bytearray 类型; 【points】FFT 运算点数，仅支持 64,128,256 和 512 点； 【shift】偏移，默认为 0。 返回值 FFT_res: 返回计算后的频域数据，以 list 类型呈现，该列表有 points 个元组，每个元组有 2 个元素，第一个元素为实部，第二个为虚部.
使用方法
res = FFT.freq(points, sample_rate)
频率计算； 【points】计算点数； 【sample_rate】采样率； 返回值 res ：返回一个列表，该列表存放的进行运算后后所有频率点的频率值；

amp = FFT.amplitude(FFT_res)
计算各个频率点的幅值； 返回值 amp：返回一个列表，该列表存放了各个频率点的幅值

表 31-2 FFT 对象

从上表可以看到，利用现成的库模块可以轻松地通过 MicroPython 编程实现麦克风音频采集和 FFT 运算，代码编写流程如下：

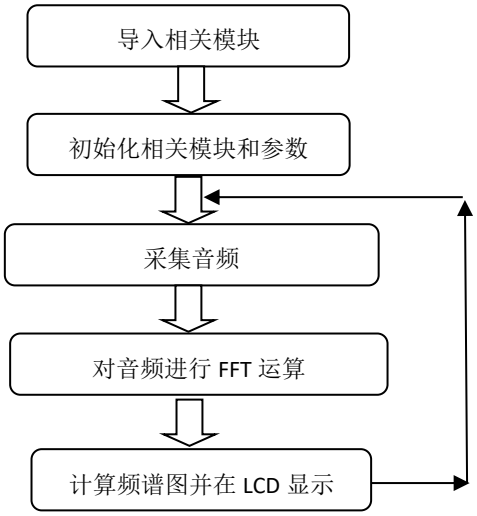


图 31-4 代码编写流程图

参考代码如下：

```
'''
实验名称：FFT（快速傅里叶）运算
版本： v1.0
日期： 2019.12
实验说明：通过 FFT 运算，将输入的音频从时域信号转换成频域信号。通过 LCD 条形柱显示。
翻译和注释： 01Studio
'''
from Maix import GPIO, I2S, FFT
import image, lcd, math
from fpioa_manager import fm
```

```
#FFT 参数配置
sample_rate = 38640  #采样率
sample_points = 1024 #音频采样点数
fft_points = 512  #FFT 运算点数
hist_x_num = 50 #条形柱数量
x_shift = 0    #频率

#lcd 初始化
lcd.init()

#麦克风初始化
fm.register(20,fm.fpioa.I2S0_IN_D0, force=True)
fm.register(19,fm.fpioa.I2S0_WS, force=True)
fm.register(18,fm.fpioa.I2S0_SCLK, force=True)

rx = I2S(I2S.DEVICE_0)
rx.channel_config(rx.CHANNEL_0, rx.RECEIVER, align_mode =
                    I2S.STANDARD_MODE)
rx.set_sample_rate(sample_rate)

#设置 LCD 条形柱显示宽度
if hist_x_num > 320:
    hist_x_num = 320
hist_width = int(320 / hist_x_num) #changeable

#新建一张图片
img = image.Image()

while True:
```

```
audio = rx.record(sample_points)   #采集音频
fft_res = FFT.run(audio.to_bytes(),fft_points) #FFT 运算
fft_amp = FFT.amplitude(fft_res) #计算频谱幅值
img = img.clear()
x_shift = 0

#计算幅值，最大为 240（LCD 高为 240 像素）
for i in range(hist_x_num):
    if fft_amp[i] > 240:
        hist_height = 240
    else:
        hist_height = fft_amp[i]

    #计算要显示的图像，矩形实心显示。
    img = img.draw_rectangle((x_shift,240-hist_height,hist_width,
                              hist_height),[255,255,255],2,True)
    x_shift = x_shift + hist_width

lcd.display(img) #LCD 显示
fft_amp.clear()  #幅度值清 0
```

- **实验结果：**

手机安装一个音频发生器（频率发生器）APP 用于播放不同频率的声源，在开发板上运行上述程序，可以看到 LCD 实时显示当前频谱图。

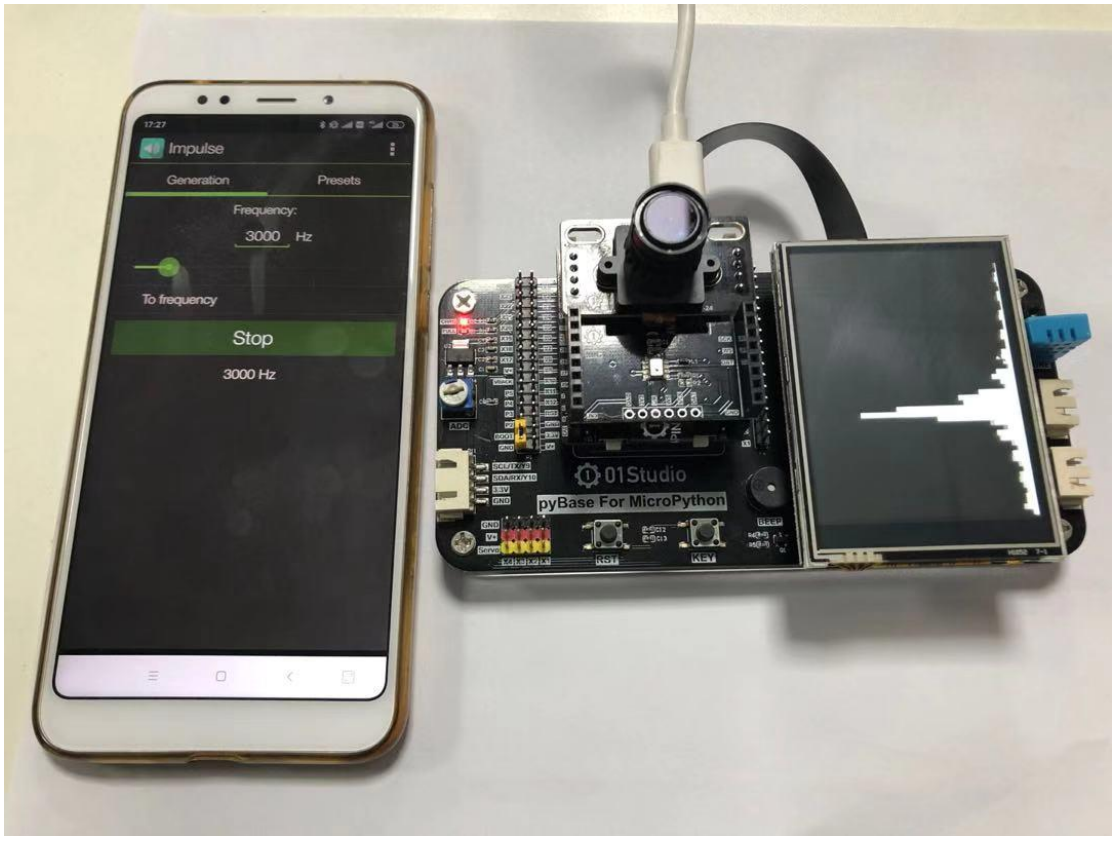


图 31-5 FFT 频谱实验

- **总结：**

通过本实验学习我们了解了 FFT 原理，同时学会了 I2S 数字麦克风音频设备和 FFT 运算通过使用 micropython 的编程方法。该实验支持多个音频输入，有兴趣用户可以使用多个手机（声源）进行测试。

31.2 声源定位

- **前言：**

从亚马逊的 Alexa、google 智能音箱到国内的天猫精灵、小爱同学等智能音箱都有一个共同的功能，那就是声源定位，其原理是利用多个分布在不同方向的麦克风阵列采集声音，进而定位声源的位置。今天我们就来编程实现声源定位。

- **开发平台：**

pyAI-K210 开发套件+麦克风阵列模块。

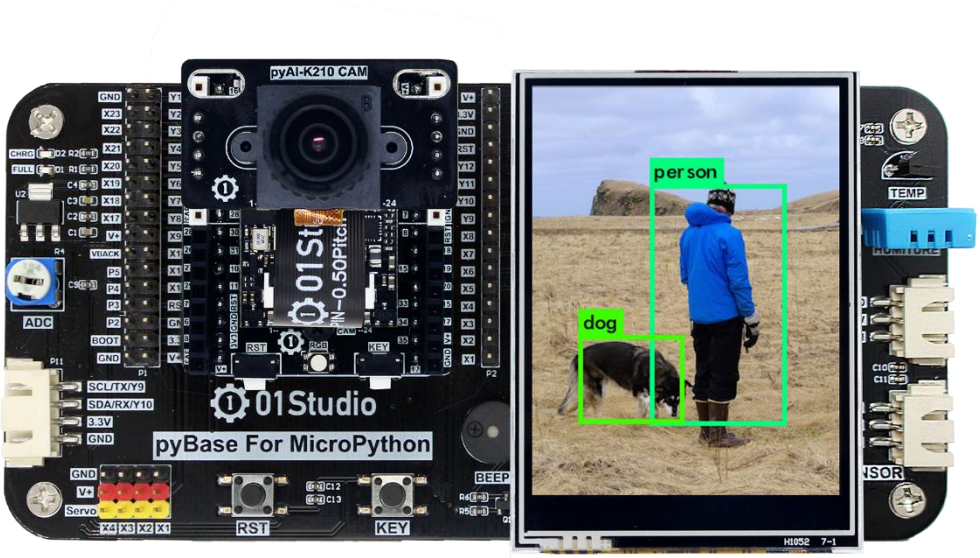


图 31-6 pyAI-K210 开发套件

- **实验目的：**

通过麦克风阵列编程实现声源定位，并在 LCD 上显示。

- **实验讲解：**

我们先来看一下开发板与麦克风阵列模块的连接方式。

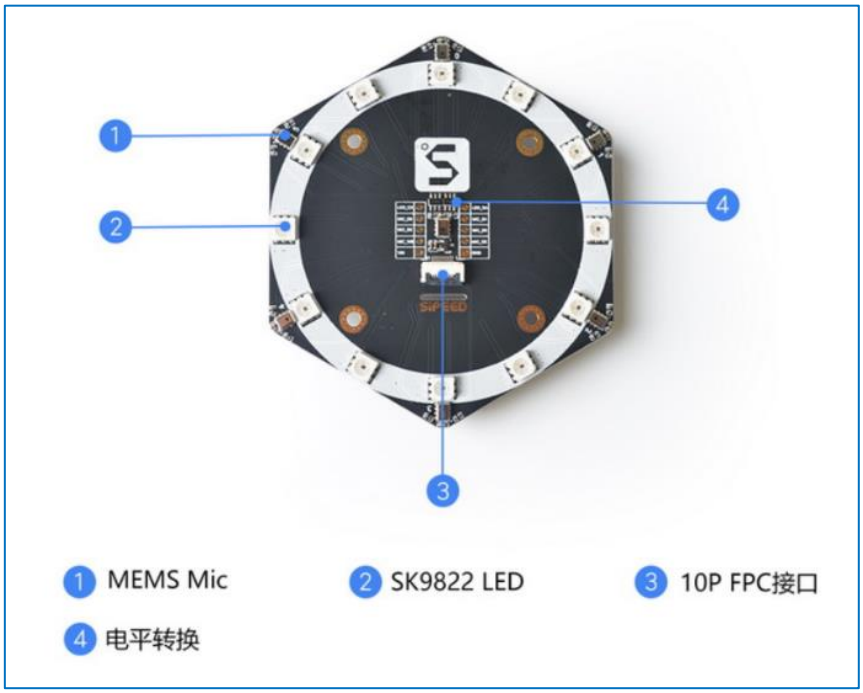


图 31-7 麦克风阵列模块

将麦克风阵列模块与 pyAI-K210 使用 FPC 10P 排线进行连接，接口在开发板背部（排线金手指下接）。

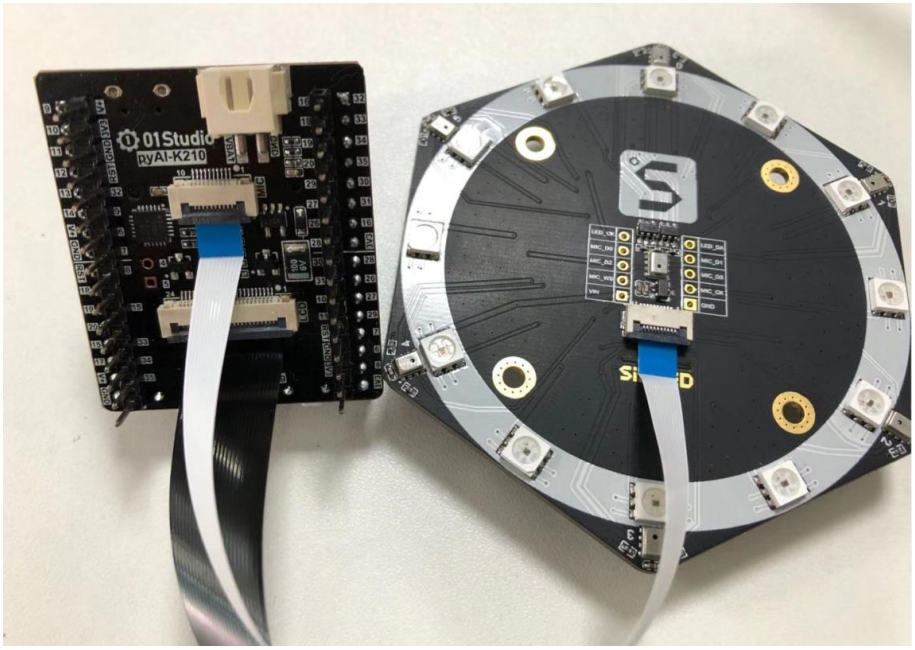


图 31-8 连接方式

麦克风阵列扩展模块由 7 个数字麦克风组成，其中 6 个分布在四周不同方向，1 个在模块正中央，便于识别各个方向声源。另外 12 个 LED 指示灯，方便指示声源位置。

本实验目的是编程实现麦克风阵列声源定位，在 LCD 上显示声源位置。而 MaixPy 内置了 MIC_MARRY 模块,位于 Maix 大模块下。使用该模块可以轻松实现声源定位。模块说明如下：

构造函数
from Maix import MIC_ARRAY as mic
导入 MIC_ARRAY 模块;
使用方法
`mic.getmap()`
返回声源黑白位图；尺寸 16*16。
`mic.get_dir(img)`
从声源位图计算声源方向。返回 12 个强度值。对应 12 个 LED 指示灯。
`mic.set_led(direction,color)`
从计算的声源方向设置点亮对应的 LED。
`mic.deinit ()`
反初始化 mic。

表 31-3 麦克风阵列对象

从上表可以看到，利用现成的库模块可以轻松地通过 MicroPython 编程实现麦克风阵列编程，从而实现声源定位。代码编写流程如下：

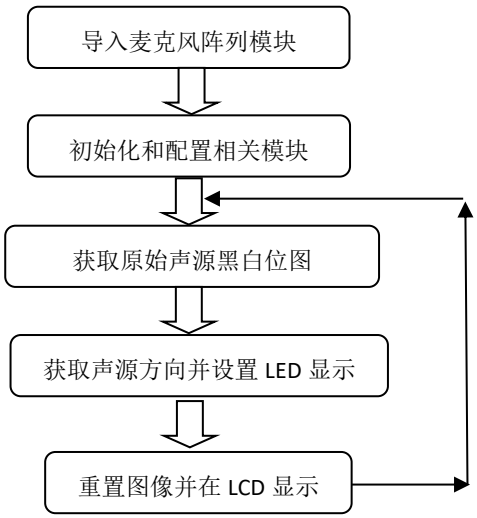


图 31-9 代码编写流程图

参考代码如下：

```
'''
实验名称：声源定位
版本： v1.0
日期： 2019.12
翻译和注释： 01Studio
实验目的：通过麦克风阵列编程实现声源定位，并在 LCD 上显示。
'''

#导入 MIC_ARRAY 和 LCD 模块
from Maix import MIC_ARRAY as mic
import lcd

#初始化模块
lcd.init()
mic.init()

while True:

    #获取原始的声源黑白位图，尺寸 16*16
    imga = mic.get_map()

    #获取声源方向并设置 LED 显示
    b = mic.get_dir(imga)
    a = mic.set_led(b,(0,0,255))

    #将声源地图重置成正方形，彩虹色
    imgb = imga.resize(160,160)
    imgc = imgb.to_rainbow(1)
```

```
    #显示声源图
    lcd.display(imgc)


mic.deinit()
```

- **实验结果：**

运行代码，打开手机播放音乐，将手机扬声器放置不同的位置，可以看 LED 和 LCD 声源图均指示声源方向。

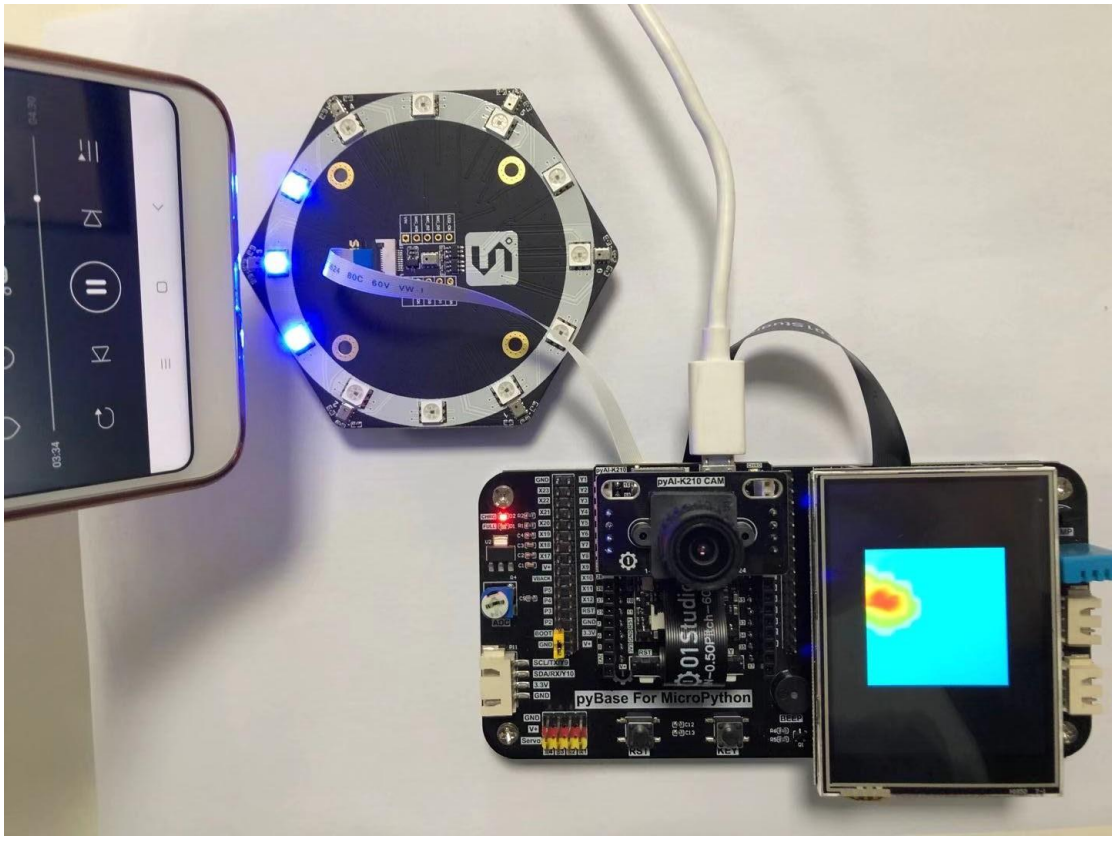


图 31-10 声源定位实验

- **总结：**

声源定位是机器听觉中非常实用的功能，特别是应用在智能音箱和一些需要定位声源的应用。通过 MicroPython 编程快速实现了声源定位，再次体验了 python 嵌入式编程的强大。

第32章　拓展模块

32.1　电阻触摸屏

- **前言：**

触摸屏是非常棒的人机交互方式。看早期的按键手机到现在已经全面被触摸屏取代。前面我们学习过来 LCD 的应用，但这是缺乏直接交互的，触摸屏很好地解决了人跟屏直接交互的问题。今天我们就来学习一下触摸屏的应用。

- **实验平台：**

pyAI-K210 开发套件。

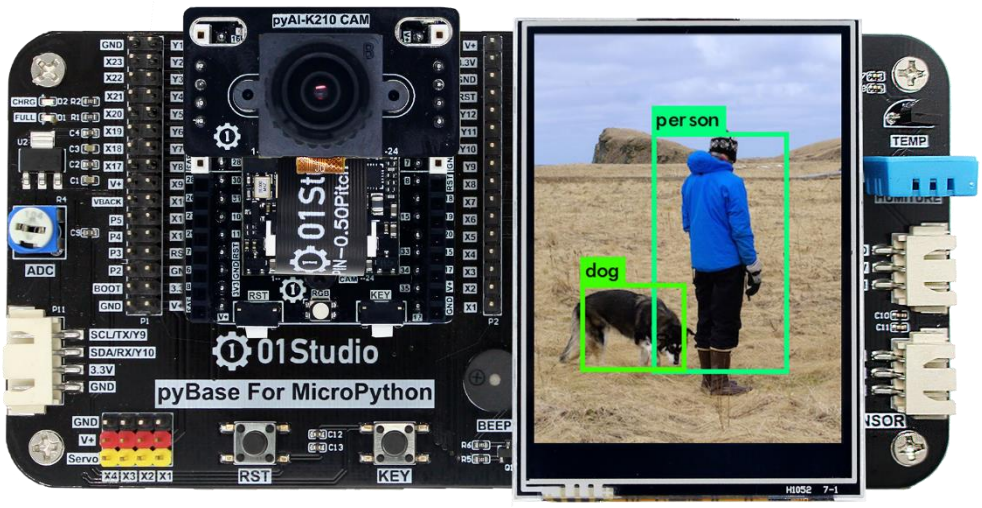


图 32-1 pyAI-K210 开发套件

- **实验目的：**

编程实现触摸屏应用。画图并通过 LCD 显示。

- **实验讲解：**

配套的 LCD 触摸屏是电阻触摸屏，与电容屏不一样，电阻触摸屏是不支持多点触控的。还有部分场合需要校准。但由于其高性价比和使用简单，还是不少嵌入式系统选择使用电电阻屏。

本实验 LCD 触摸屏上使用 NS2009 芯片，将电阻触摸屏信号转化为 I2C 信号跟 K210 通信，而 MaixPy 已经集成了触摸屏应用的相关函数模块，具体介绍如下：

构造函数
`import touchscreen as ts`
导入 touchscreen 模块;
使用方法
`ts.init(i2c=None,cal=None)`
初始化触摸屏。 【i2c】I2C 总线； 【cal】一个 7 个整型值的元组，触摸校准数据。
`ts.calibrate()`
触摸校准。返回一个 7 个整型值的元组。
`ts.read()`
读取屏幕状态和坐标信息。返回（status,x,y） 【status】：触摸状态，取值有如下 touchscreen.STATUS_RELEASE,值为 1，触摸屏没动作； touchscreen.STATUS_PRESS,值为 2，触摸屏被按下； touchscreen.STATUS_MOVE,值为 3，触摸屏在滑动； 【x】x 轴坐标 【y】y 轴坐标

表 32-1　touchscreen 对象

从上表可以看到，通过 MicroPython 封装后的电阻触摸屏使用变得非常容易，另外加入一个按键 KEY 作为清屏功能，本节代码编程流程如下：

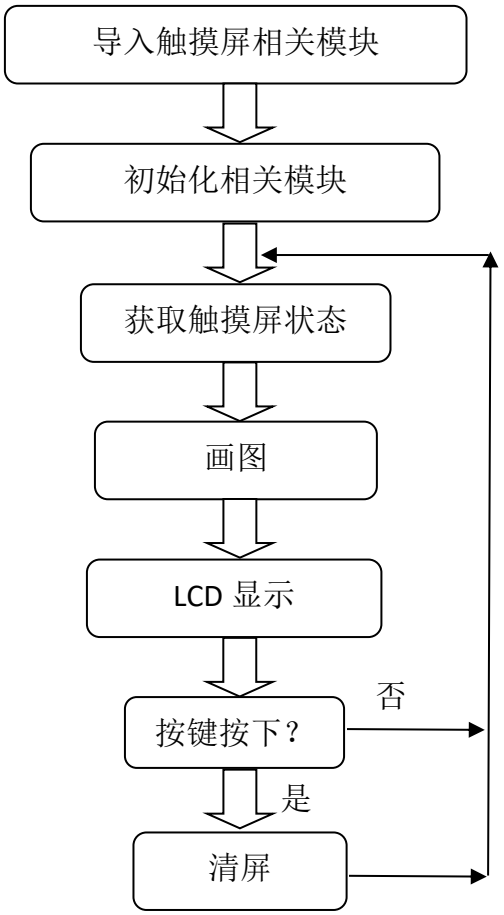


图 32-2 代码编程流程

参考代码如下：

```
'''
实验名称：触摸屏应用
版本： v1.0
日期： 2019.12
翻译和注释： 01Studio
'''

#导入相关模块
from machine import I2C
from fpioa_manager import fm
from Maix import GPIO
```

```
import lcd, image
import touchscreen as ts

#按键 KEY 用于清屏
fm.register(16, fm.fpioa.GPIO1, force=True)
btn_clear = GPIO(GPIO.GPIO1, GPIO.IN)

#触摸使用 I2C 控制（NS2009）
i2c = I2C(I2C.I2C0, freq=400000, scl=30, sda=31)

#触摸屏初始化
ts.init(i2c)
#ts.calibrate() #触摸校准

#LCD 初始化
lcd.init()
lcd.clear()

#新建图像和触摸屏相关参数变量
img = image.Image()
status_last = ts.STATUS_IDLE
x_last = 0
y_last = 0
draw = False

while True:

    #获取触摸屏状态
    (status,x,y) = ts.read()
    print(status, x, y)
```

```
#画图
if draw:
    img.draw_line((x_last, y_last, x, y))

#更新最后坐标
x_last = x
y_last = y

#根据触摸屏状态判断是否继续执行画图功能
if status_last!=status:
    if (status==ts.STATUS_PRESS or status == ts.STATUS_MOVE):
        draw = True
    else: #松开
        draw = False
    status_last = status

#LCD 显示
lcd.display(img)

#按键 KEY 按下清屏
if btn_clear.value() == 0:
    img.clear()
```

- **实验结果：**

运行程序，在触摸屏上滑动，可以看到 LCD 画线显示。按下按键 KEY 后可以实现清屏。

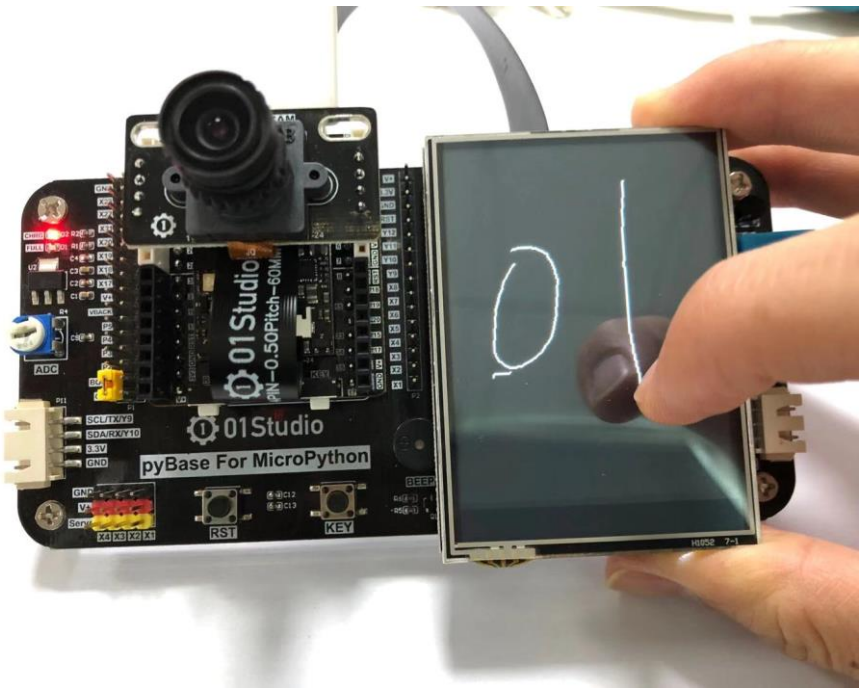


图 32-3 触摸屏实验

- **总结：**

有了触摸屏的应用，让开发板的人机交互变得更简单而有趣。

32.2 舵机

- **前言：**

舵机又叫伺服电机，是一个可以旋转特定角度的电机，可转动角度通常是 90°、180° 和 360°（360° 可以连续旋转）。我们看到的机器人身上就有非常多的舵机，它们抬手或者摇头的动作往往是通过舵机完成，因此机器人身上的舵机越多，意味着动作越灵活。

- **实验平台：**

pyAI-k210 开发套件和舵机。

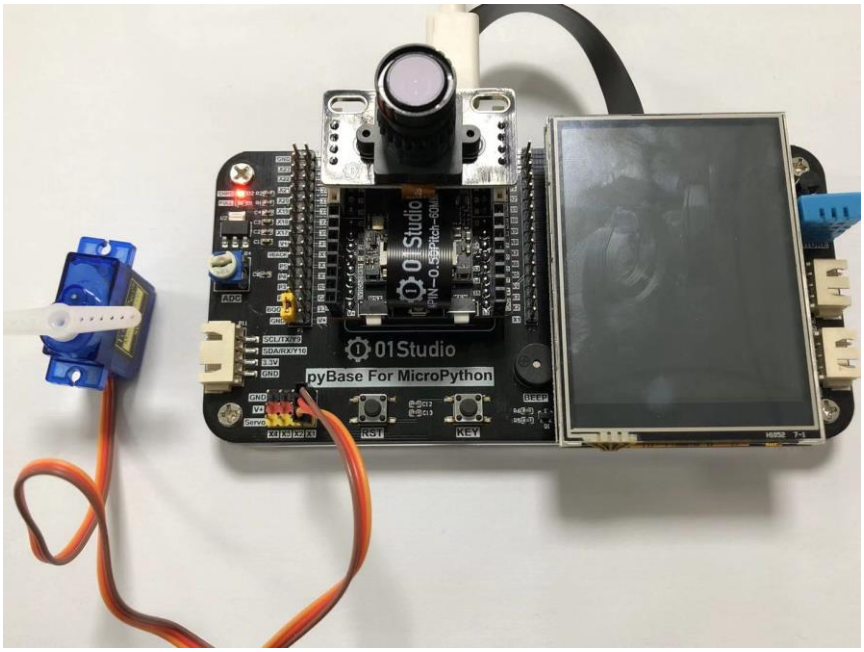


图 32-4 舵机接线

通常情况下：**黑色表示 GND，红色表示 VCC，橙色表示信号线。**

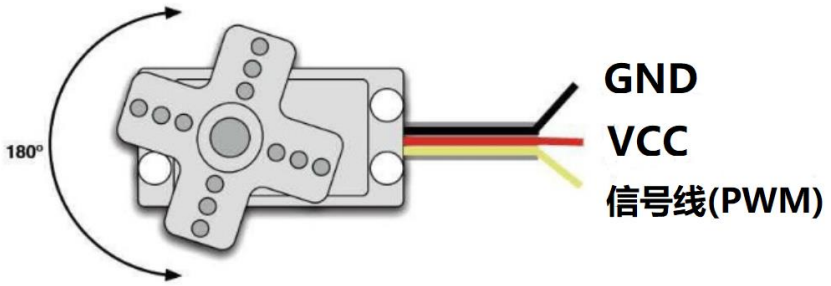


图 32-5 舵机示意图

- **实验目的：**

通过编程实现对舵机的控制。

- **实验讲解：**

伺服电机对象通过 3 线（信号，电源，地）控制，pyBase 上有 4 个位置可以插这些电机，分别是 X1-X4 引脚。对应 pyAI-K210 的 17、35、34、33 脚。

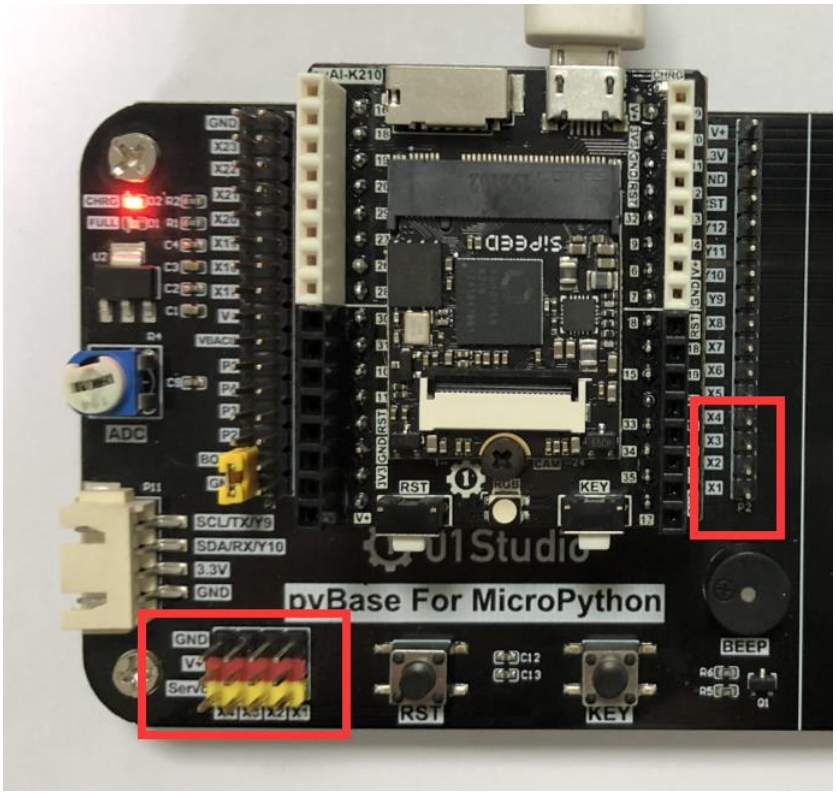


图 32-6　pyBase 的 4 路舵机接口

180°舵机的控制一般需要一个 20ms 左右的时基脉冲，该脉冲的高电平部分一般为 0.5ms-2.5ms 范围内的角度控制脉冲部分，总间隔为 2ms。以 180 度角度伺服为例，在 MicroPython 编程对应的控制关系是从-90°至 90°，示例图如下：

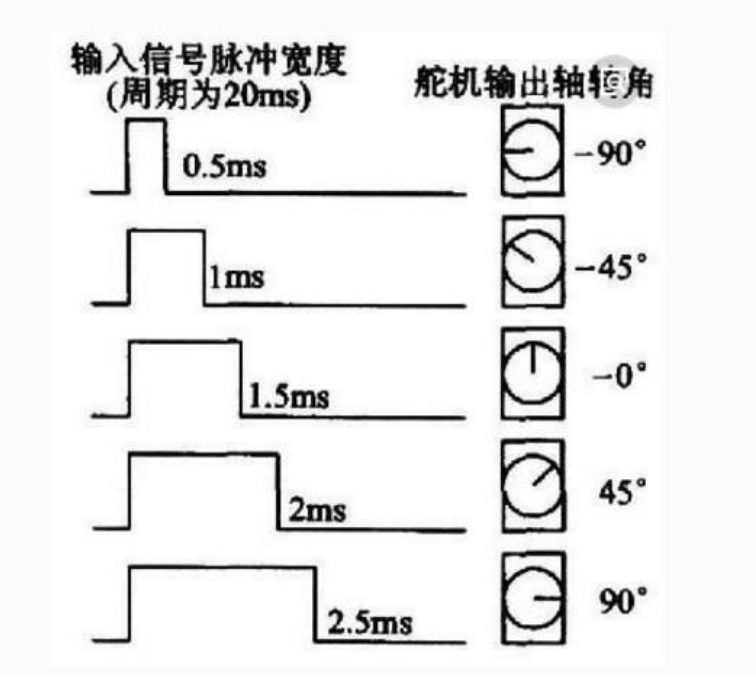


图 32-7　PWM 与角度关系

而对于 360°连续旋转舵机，上面的脉冲表则对应从正向最大速度旋转到反向最大速度旋转的过程。

如果过你学习过前面基于 STM32 平台的舵机实验，那就知道在 STM32 平台集成了舵机模块，使用起来非常方便。当前的 ESP32 平台并没有集成 Servo 模块，但从上面可以看到上面是通过 PWM 来控制的，我们可以直接写 PWM 函数驱动即可。代码编程流程图如下：

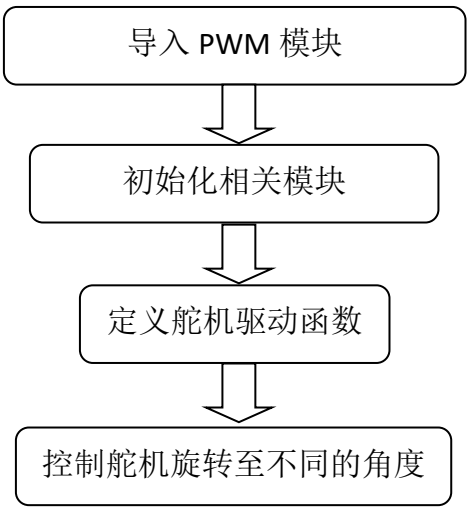


图 32-8 编程流程图

参考代码如下：

```
'''
实验名称：舵机控制
版本：v1.0
日期：2019.12
作者：01Studio 【www.01Studio.org】
说明：通过编程控制舵机旋转到不同角度
'''

from machine import Timer,PWM
import time

#PWM 通过定时器配置，接到 IO17 引脚
tim = Timer(Timer.TIMER0, Timer.CHANNEL0, mode=Timer.MODE_PWM)
S1 = PWM(tim, freq=50, duty=0, pin=17)

'''
说明：舵机控制函数
功能：180 度舵机：angle:-90 至 90 表示相应的角度
     360 连续旋转度舵机：angle:-90 至 90 旋转方向和速度值。
    【duty】占空比值：0-100
'''

def Servo(servo,angle):
    S1.duty((angle+90)/180*10+2.5)



while True:
    #-90 度
    Servo(S1,-90)
```

```
time.sleep(2)


#-45 度
Servo(S1,-45)
time.sleep(2)


#0 度
Servo(S1,0)
time.sleep(2)


#45 度
Servo(S1,45)
time.sleep(2)


#90 度
Servo(S1,90)
time.sleep(2)
```

- **实验结果：**

将 180° 舵机插到 pyBase 的 X1 的三线接口，运行程序。可以看到舵机依次旋转至不同角度。

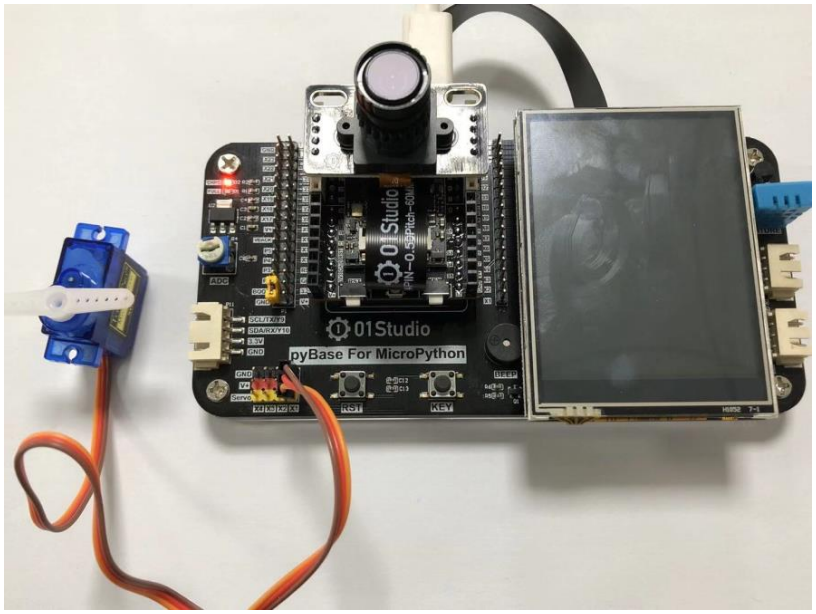


图 32-9 舵机角度旋转至不同角度

- **实验拓展：**

我们刚刚实现了 180° 舵机的角度控制，现在来做一下 360° 连续旋转舵机的实验，360° 连续旋转舵机可以实现直流减速电机功能，用在小车或者航模上。

实验的代码不变，参数【-90 至 90】代表旋转方向和速度值大小。插上 360° 连续旋转舵机。可以看到舵机的旋转速度和方向逐渐变变化。

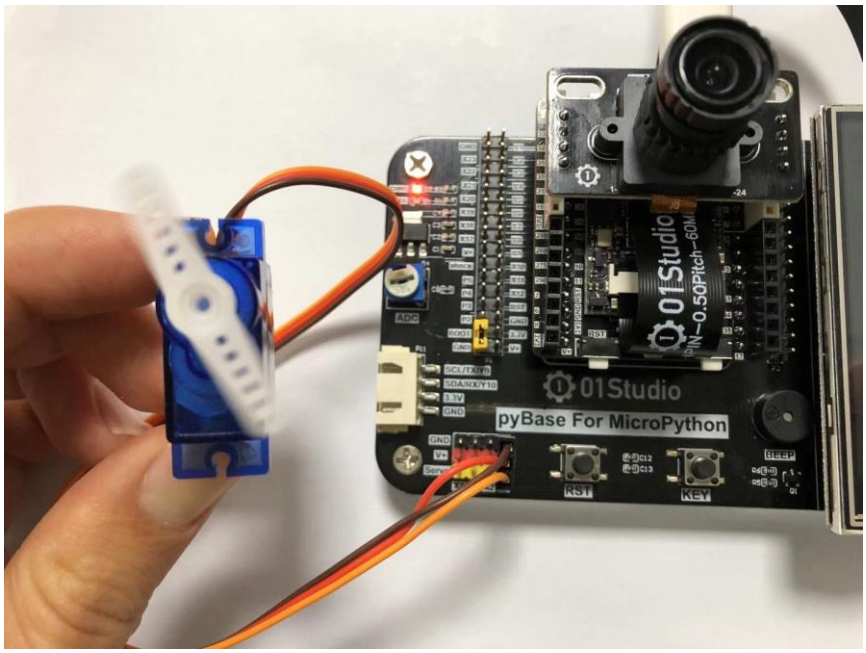


图 32-10 360°连续旋转舵机

- **总结：**

通过本节我们学会了使用不同类型的舵机，通过多路电机的组合使用，可以实现模型、航模、小车、机器人的实验。

32.3 RGB 灯带

- **前言：**

RGB 灯带是彩灯的一种，我们看到长长的灯带以及 LED 彩色显示屏，都是采用一个个灯珠组合而成。在本章节我们将通过编程实现 RGB 灯带的控制，可以应用到家居布置、节日气氛的场景中去。

- **实验平台：**

pyAI-K210 开发套件和 RGB 灯带。连接 pyBase 到“Y12”接口。

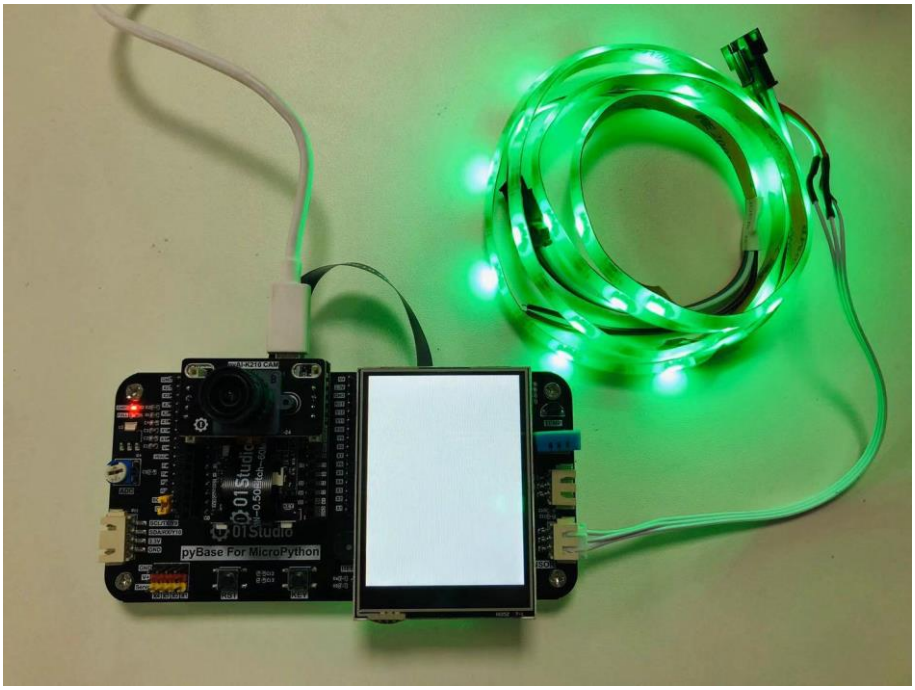


图 32-11 RGB 灯带接线方式

- **实验目的：**

通过编程实现灯带循环显示红色（RED）、绿色（GREEN）和蓝色（BLUE）。

- **实验讲解：**

先来介绍一下本实验用到的 RGB 灯带。

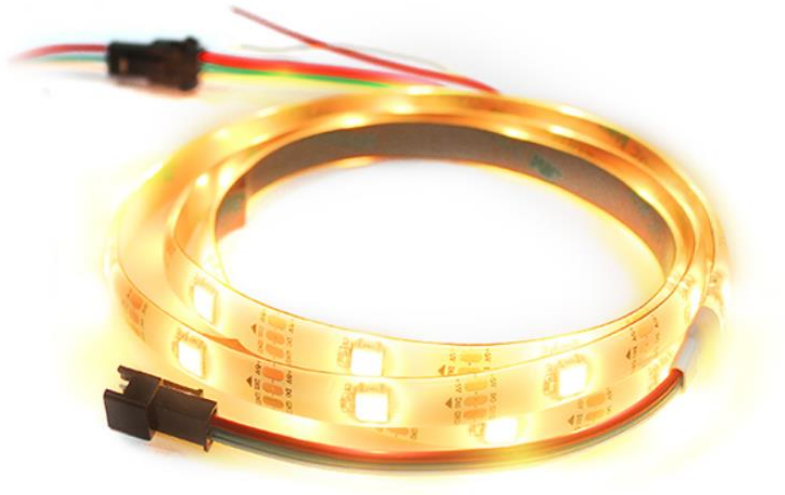


图 32-12 RGB 灯带

该灯带为 3 线接口，长 1 米，有 30 颗灯珠，每个灯珠里面都有一个驱动 IC，型号是 WS2812B，单总线驱动。每个灯珠首尾相连。灯带末端预留了接口可以串联更多的灯带。也就是说只需要通过 1 个 GPIO 口就可以控制了数十上百的灯珠了。

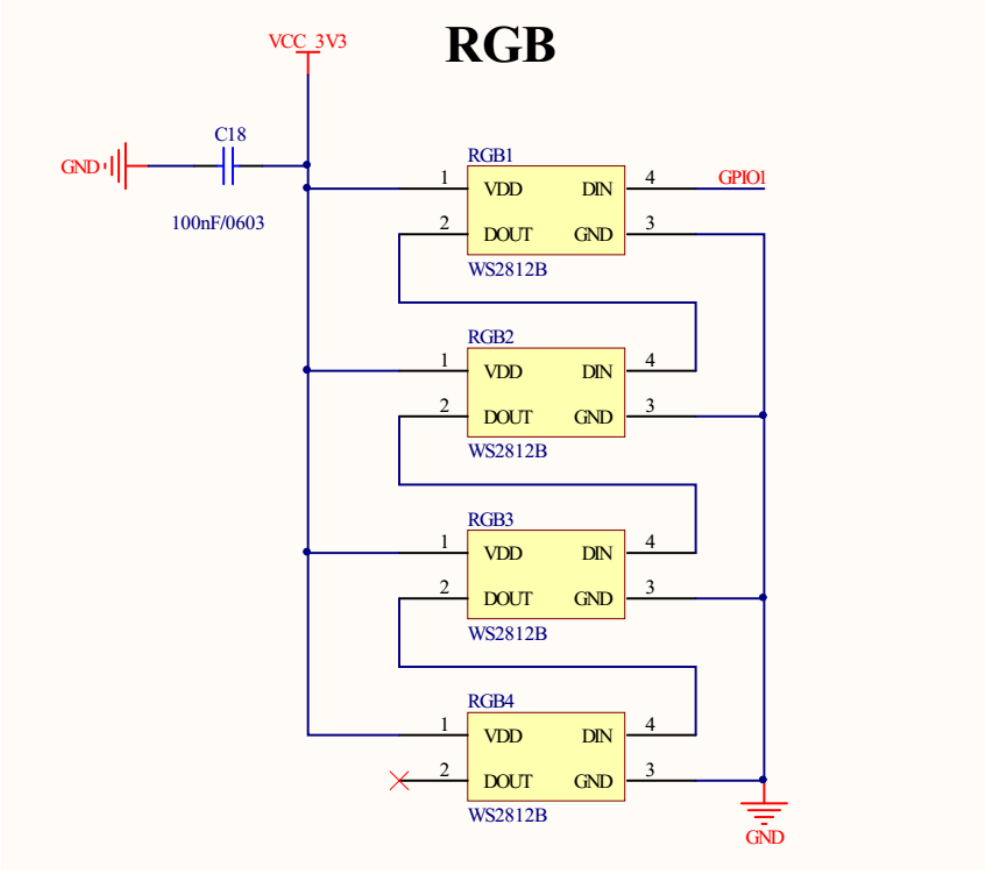


图 32-13 灯珠串联接线方式

在了解了接线方式后，我们知道颜色是由最基本的三种颜色的不同亮度混合出新颜色。这 3 个最基本的颜色顺序分别是红，绿，蓝（RGB）。这里每个颜色的亮度级别从 0-255,0 表示没有，255 表示最亮。如（255,0,0）则表示红色最亮。GPIO 口就是将这些数据逐一发送给灯带。

K210 的 MicroPython 固件集成了彩灯驱动模块，适用于 WS2812B 驱动的灯珠。因此我们可以直接使用。说明如下：

构造函数
np=ws2812(pin, num)
构建灯带对象。 【pin】灯带控制引脚; 【num】:灯珠数量;
使用方法
np.set_led=(i,color)
设置第 i 个灯珠参数。 【i】第 i 个灯珠; 【color】灯珠颜色，RGB 表示。
np.display()
往灯带写入设置的数据。

表 32-2　ws2812-RGB 灯带对象

代码编程流程图如下：

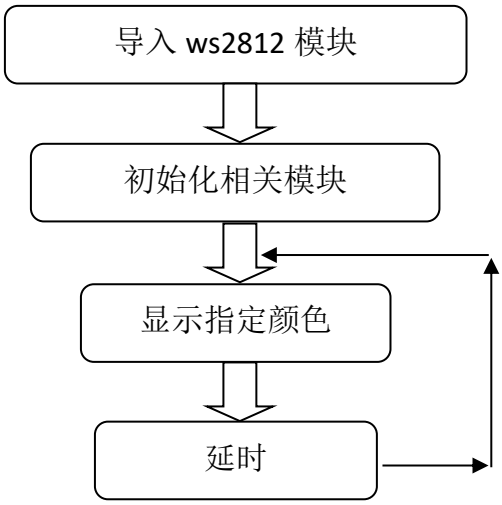


图 32-14 编程流程图

参考程序代码如下：

```
'''
实验名称：RGB 彩灯控制
版本：v1.0
日期：2019.12
作者：01Studio 【www.01Studio.org】
说明：通过编程实现灯带红、绿、蓝不同颜色的变化。
'''

from modules import ws2812
import utime

#定义红、绿、蓝三种颜色
RED=(255,0,0)
GREEN=(0,255,0)
BLUE=(0,0,255)

#定义 RGB 灯带对象,引脚 32 连接，灯珠数量：30。
```

```
np = ws2812(32,30)


while True:


    #显示红色
    for i in range(30):
        np.set_led(i,RED)
    np.display()
    utime.sleep(1)


    #显示绿色
    for i in range(30):
        np.set_led(i,GREEN)
    np.display()
    utime.sleep(1)


    #显示蓝色
    for i in range(30):
        np.set_led(i,BLUE)
    np.display()
    utime.sleep(1)
```

- **实验结果：**

  将程序下载到开发套件，可以看到灯带的颜色在红、绿、蓝三色中循环变化。

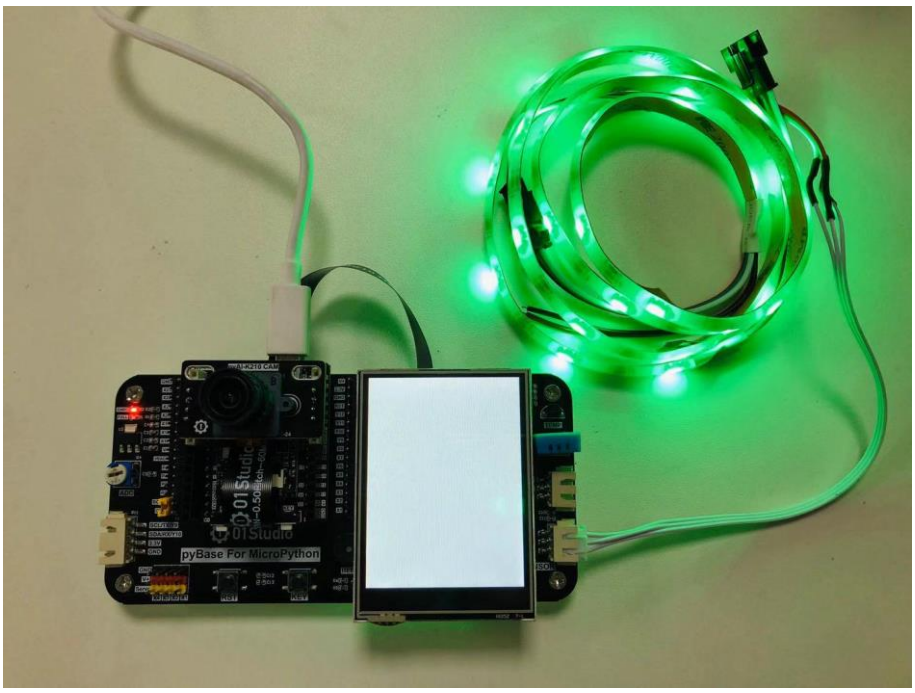


图 32-15 实验结果

- **总结：**

RGB 灯带的单总线特点使得我们可以轻易的增加和减少灯带的数量而无需过多的修改程序。但需要注意的是如果灯带数量过多，那么需要外接电源供电，否则会因为电流不足而影响使用。

32.4 WiFi 模块

pyAI-K210 可以通过 WiFi 扩展模块连接无线路由器，从而实现 Socket、MQTT、等相关无线和物联网应用。模块的接口和功能参数介绍如下：

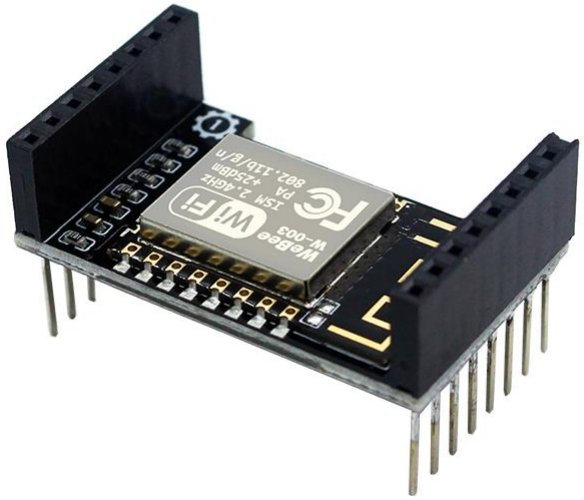


图 32-16 K210 WiFi 模块

功能参数	
WiFi 主控	ESP8266
控制方式	UART（串口）
WiFi 协议	802.11 b/g/n
加密类型	WPA/WPA2
工作电流	80mA (3.3V)
模块尺寸	3.0*2.5cm
适用平台	pyAI-K210

表 32-3 OpenMV WiFi 模块功能参数

32.4.1 连接无线路由器

- **前言：**

WIFI 是物联网中非常重要的角色，现在基本上家家户户都有 WIFI 网络了，通过 WIFI 接入到互联网，成了智能家居产品普遍的选择。而要想上网，首先需要连接上无线路由器。这一节我们就来学习如何通过 WiFi 扩展模块让 K210 通过 MicroPython 编程实现连上路由器。

- **实验平台：**

pyAI-K210 和 K210 WiFi 模块。



图 32-17 pyAI-K210+WiFi 模块

- **实验目的：**

编程实现连接路由器，将 IP 地址等相关信息通过串口终端打印（只支持 2.4G 网络）。

- **实验讲解：**

连接路由器上网是我们每天都做的事情，日常生活中我们只需要知道路由器的账号和密码，就能使用电脑或者手机连接到无线路由器，然后上网冲浪。

MaixPy 已经集成了 network 模块，开发者使用内置的 network 模块函数可以非常方便地连接上路由器。

我们先来看看 network 基于 WiFi（WLAN 模块）的构造函数和使用方法。

构造函数
wlan = network.ESP8285(uart)
构建 WIFI 模块连接对象。ESP8285 表示模块型号。 【uart】串口对象
使用方法
wlan.scan ()
扫描允许访问的 SSID。
wlan.isconnected()
检查设备是否已经连接上。返回 Ture:已连接；False：未连接。
wlan.connect(ssid,passwork)
WIFI 连接。 【ssid】账号； 【passwork】密码；
wlan.ifconfig()
查看 wifi 连接信息。
wlan.disconnected()
断开连接。
wlan.enable_ap(ssid,key,chl=5,ecn=3)
AP 热点使能； 【ssid】热点名称； 【key】密码； 【chl】wifi 信号的信道； 【ecn】加密方法
wlan.disable_ap()
关闭热点。
*更多参数说明请阅读 MaixPy 官方文档： 文档链接：https://maixpy.sipeed.com/zh/libs/machine/network.html

表 32-4 K210 network 对象

从上表可以看到 MicroPython 通过模块封装，让 WIFI 联网变得非常简单。模块包含热点 AP 模块和客户端 STA 模式，热点 AP 是指电脑端直接连接 K210 发出的热点实现连接，但这样你的电脑就不能上网了，因此我们一般情况下都是使用 STA 模式。也就是电脑和设备同时连接到相同网段的路由器上。

代码编写流程如下：

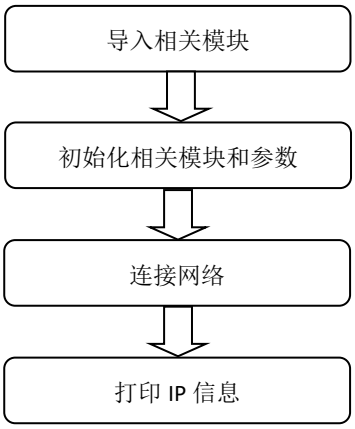


图 32-18 代码编写流程图

实验参考代码如下：

```
# 实验名称：WiFi 无线连接实验（串口 WiFi 模块）
#
# 说明：pyAI-K210 通过 WiFi 拓展模块连接无线路由器
#
# 作者：01Studio

import network, time
from machine import UART
from Maix import GPIO
from fpioa_manager import fm

SSID='01Studio' # WiFi 账号
KEY='88888888'  # WiFi 密码
```

```
###### WiFi 模块初始化 ######
#使能引脚初始化
fm.register(8, fm.fpioa.GPIOHS0, force=True)
wifi_en=GPIO(GPIO.GPIOHS0, GPIO.OUT)

#串口初始化
fm.register(7, fm.fpioa.UART2_TX, force=True)
fm.register(6, fm.fpioa.UART2_RX, force=True)
uart = UART(UART.UART2,115200, read_buf_len=4096)

#使能函数
def wifi_enable(en):
    global wifi_en
    wifi_en.value(en)

#使能 wifi 模块
wifi_enable(1)
time.sleep(1)

#构建 WiFi 对象
wlan = network.ESP8285(uart)

#正在连接印提示
print("Trying to connect... (may take a while)...")

#连接网络
wlan.connect(SSID,KEY)

#打印 IP 相关信息
```

```
print(wlan.ifconfig())
```

上面代码在 main.py 文件中，我们也可以新建一个 WIFI.py 文件，然后通过模块引用方式来供 main.py 调用，以提高程序的灵活性。

- **实验结果：**

运行程序，可以观察到连接成功后串口终端打印 IP 等信息。

```
>>> 1
1
1
Trying to connect... (may take a while)...
('192.168.1.117', '255.255.255.0', '192.168.1.1', '0', '0', '28:6c:07:
MicroPython v0.5.0 on 2019-11-29; Sipeed_M1 with kendryte-k210
Type "help()" for more information.
>>>
```

图 32-19

- **总结：**

本节是 WIFI 应用的基础，成功连接到无线路由器的实验后，后面就可以做 socket 和 MQTT 等相关网络通信的应用了。

32.4.2 Socket 通信

- **前言：**

上一节我们学习了如何通过 MicroPython 编程实现 pyAI-K210 结合 WiFi 拓展模块连接到无线路由器。这一节我们则来学习一下 Socket 通信实验。Socket 几乎是整个互联网通信的基础。

- **实验平台：**

pyAI-K210 和 K210 WiFi 模块。



图 32-20 pyAI-OpenMV4 开发套件

- **实验目的：**

通过 Socket 编程实现 pyAI-K210 与电脑服务器助手建立连接，相互收发数据。

- **实验讲解：**

Socket 我们听得非常多了，但由于网络工程是一门系统工程，涉及的知识非常广，概念也很多，任何一个知识点都能找出一堆厚厚的的书，因此我们经常会混淆。在这里，我们尝试以最容易理解的方式来讲述 Socket，如果需要全面了解，可以自行查阅相关资料学习。

我们先来看看网络层级模型图，这是构成网络通信的基础：

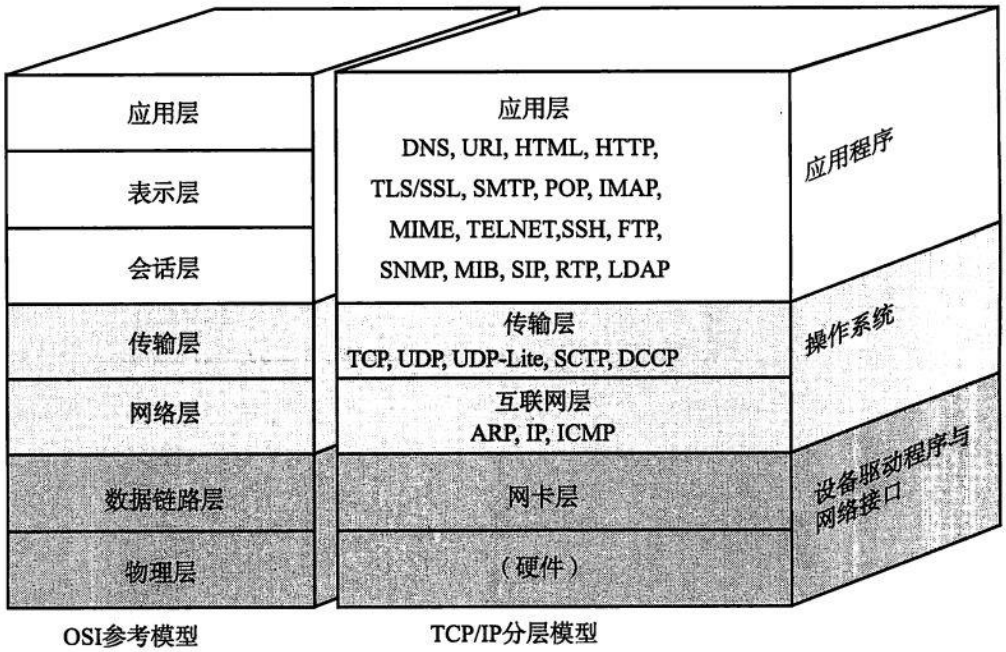


图 32-21 网络层级模型

我们看看 TCP/IP 模型的传输层和应用层，传输层比较熟悉的概念是 TCP 和 UDP，UPD 协议基本就没有对 IP 层的数据进行任何的处理了。而 TCP 协议还加入了更加复杂的传输控制，比如滑动的数据发送窗口（Slice Window），以及接收确认和重发机制，以达到数据的可靠传送。应用层中网页常用的则是 HTTP。那么我们先来解析一下这 TCP 和 HTTP 两者的关系。

我们知道网络通信是最基础是依赖于 IP 和端口的，HTTP 一般情况下默认使用端口 80。举个简单的例子：我们逛淘宝，浏览器会向淘宝网的网址（本质是 IP）和端口发起请求，而淘宝网收到请求后响应，向我们手机返回相关网页数据信息，实现了网页交互的过程。而这里就会引出一个多人连接的问题，很多人访问淘宝网，实际上接收到网页信息后就断开连接，否则淘宝网的服务器是无法支撑这么多人长时间的连接的，哪怕能支持，也非常占资源。

也就是应用层的 HTTP 通过传输层进行数据通信时，TCP 会遇到同时为多个应用程序进程提供并发服务的问题。多个 TCP 连接或多个应用程序进程可能需要通过同一个 TCP 协议端口传输数据。为了区别不同的应用程序进程和连接，许多计算机操作系统为应用程序与 TCP / IP 协议交互提供了套接字(Socket)接口。应用层可以和传输层通过 Socket 接口，区分来自不同应用程序进程或网络连接的通信，实

现数据传输的并发服务。

简单来说，Socket 抽象层介于传输层和应用层之间，跟 TCP/IP 并没有必然的联系。Socket 编程接口在设计的时候，就希望也能适应其他的网络协议。

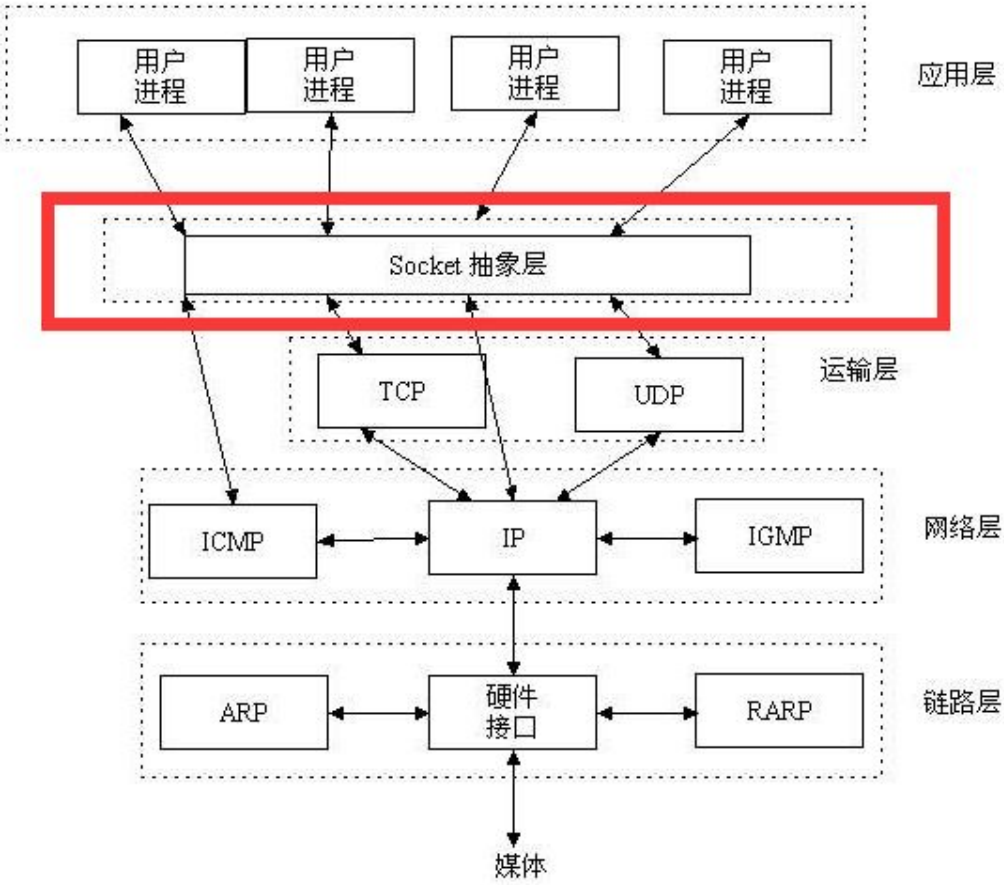


图 32-22　Socket 抽象层

套接字（socket）是通信的基石，是支持 TCP/IP 协议的网络通信的基本操作单元。它是网络通信过程中端点的抽象表示，包含进行网络通信必须的五种信息：**连接使用的协议（通常是 TCP 或 UDP），本地主机的 IP 地址，本地进程的协议端口，远地主机的 IP 地址，远地进程的协议端口。**

所以，socket 的出现只是可以更方便的使用 TCP/IP 协议栈而已，简单理解就是其对 TCP/IP 进行了抽象，形成了几个最基本的函数接口。比如 create，listen，accept，connect，read 和 write 等等。以下是通讯流程：

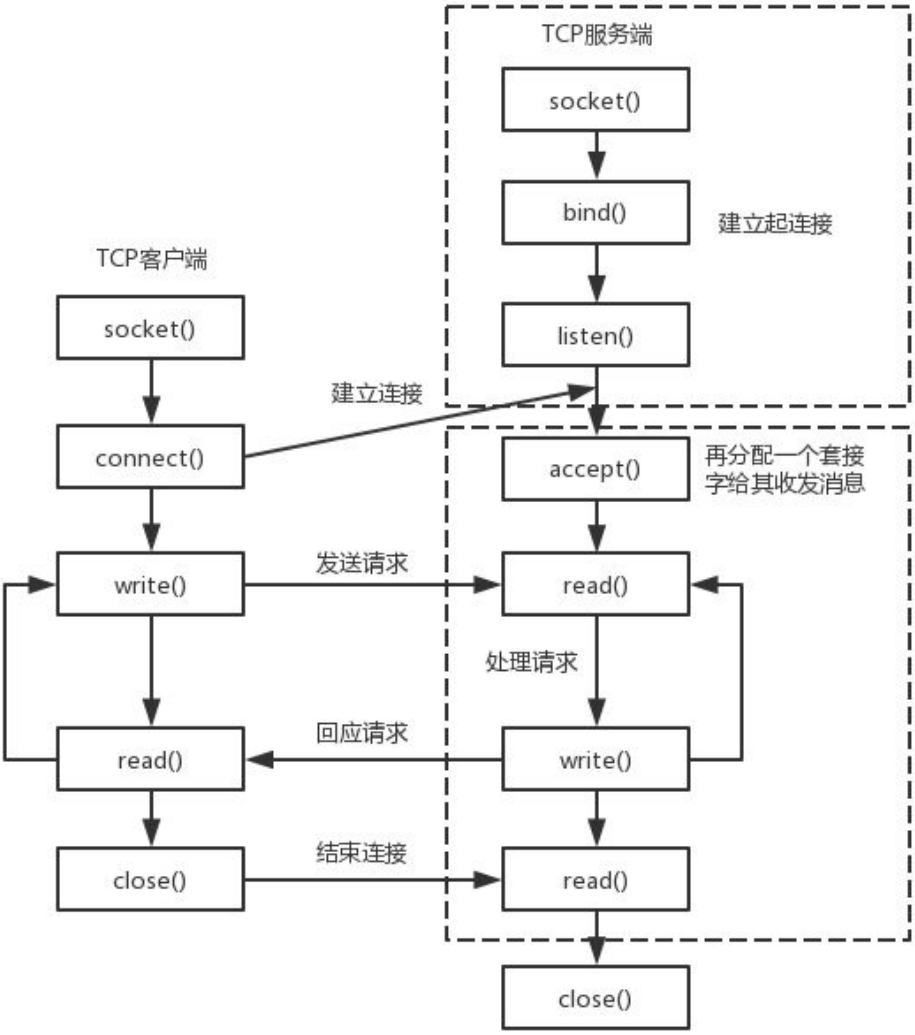


图 32-23　Socket 通信过程

从上图可以看到，建了 Socket 通信需要一个服务器端和一个客户端，以本实验为例，pyAI-K210 作为客户端，电脑使用网络调试助手作为服务器端，双方使用 TCP 协议传输。对于客户端，则需要知道电脑端的 IP 和端口即可建立连接。（端口可以自定义，范围在 0~65535，注意不占用常用的 80 等端口即可。）

以上的内容，简单来说就是如果用户面向应用来说，那么 K210 只需要知道**通讯协议是 TCP 或 UDP、服务器的 IP 和端口号**这 3 个信息，即可向服务器发起连接和发送信息。就这么简单。

MicroPython 已经封装好相关模块 usocket,跟传统的 socket 大部分兼容，两者均可使用，本实验使用 usocket，对象如下介绍：

构造函数
s=usocket.socekt(af=AF_INET, type=SOCK_STREAM,proto=IPPROTO_TCP)
构建 usocket 对象。 af: AF_INET→IPV4，AF_INET6 → IPV6； type: SCOK_STREAM→TCP，SOCK_DGRAM→UDP； proto: IPPROTO_TCP→TCP 协议，IPPROTO_UDP→UDP 协议。 （如果要构建 TCP 连接，可以使用默认参数配置，即不输入任何参数。）
使用方法
addr=usocket.getaddrinfo('www.01studio.org', 80)[0][-1]
获取 Socket 通信格式地址。返回：('47.91.208.161',80)
s.connect(address)
创建连接。address:地址格式为 IP+端口。例：('192.168.1.115',10000)
s.send(bytes)
发送。bytes：发送内容格式为字节
s.recv(bufsize)
接收数据。bufsize：单次最大接收字节个数。
s.bind(address)
绑定，用于服务器角色
s.listen([backlog])
监听，用于服务器角色。backlog:允许连接个数，必须大于 0。
s.accept()
接受连接，用于服务器角色。
*其它更多用法请阅读 MicroPython 文档：(搜索:usocket) 文档链接：http://docs.openmv.io/library/usocket.html

表 32-5　Socket 对象

本实验中 pyAI-K210 属于客户端，因此只用到客户端的函数即可。实验代码

编写流程如下：

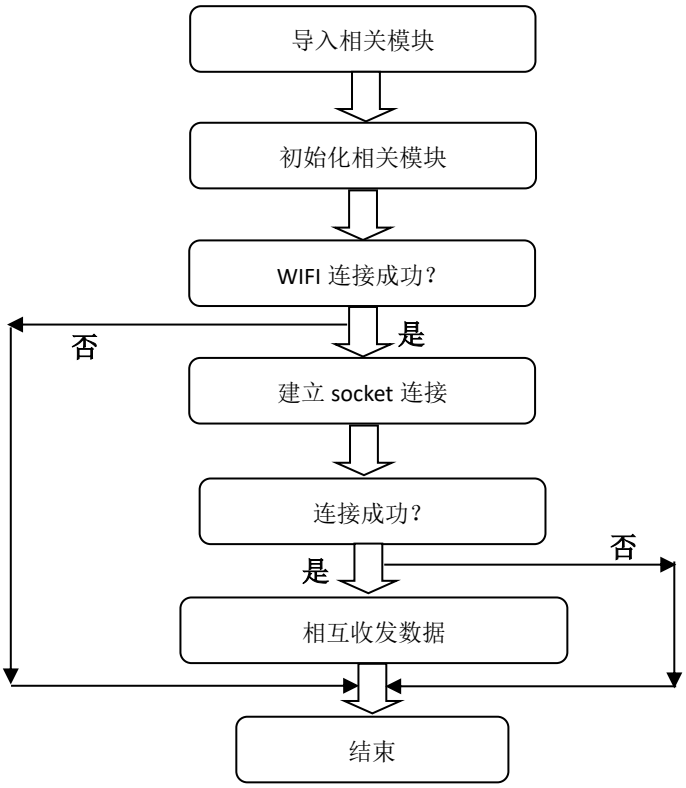


图 32-24 代码编写流程图

实验参考代码：

```
# TCP 客户端（Socket 通信） 实验
#
# 通过 WiFi 模块编程实现 K210 的 Socket 通信，数据收发.
#
#作者：01Studio


import network,usocket,time
from machine import UART,Timer
from Maix import GPIO
from fpioa_manager import fm


SSID='01Studio' # WiFi 账号
```

```
KEY='88888888'  # WiFi 密码

#socket 数据接收中断标志位
socket_node = 0

###### WiFi 模块初始化 ######
#使能引脚初始化
fm.register(8, fm.fpioa.GPIOHS0, force=True)
wifi_en=GPIO(GPIO.GPIOHS0, GPIO.OUT)

#串口初始化
fm.register(7, fm.fpioa.UART2_TX, force=True)
fm.register(6, fm.fpioa.UART2_RX, force=True)
uart = UART(UART.UART2,115200,timeout=1000,read_buf_len=4096)

#WiFi 使能函数
def wifi_enable(en):
    global wifi_en
    wifi_en.value(en)

#WiFi 对象初始化，波特率需要修改
def wifi_init():
    global uart
    wifi_enable(0)
    time.sleep_ms(200)
    wifi_enable(1)
    time.sleep(2)
    uart = UART(UART.UART2,115200,timeout=1000, read_buf_len=4096)
    tmp = uart.read()
    uart.write("AT+UART_CUR=921600,8,1,0,0\r\n")
```

```
    print(uart.read())
    #实测模块波特率太低或者缓存长度太短会导致数据丢失。
    uart = UART(UART.UART2,921600,timeout=1000, read_buf_len=10240)
    uart.write("AT\r\n")
    tmp = uart.read()
    print(tmp)
    if not tmp.endswith("OK\r\n"):
        print("reset fail")
        return None
    try:
        nic = network.ESP8285(uart)
    except Exception:
        return None
    return nic

############################
#########  主程序  ##########
############################

#构建 WiFi 对象并使能
wlan = wifi_init()

#正在连接印提示
print("Trying to connect... (may take a while)...")

#连接网络
wlan.connect(SSID,KEY)

#打印 IP 相关信息
print(wlan.ifconfig())
```

```
#创建 socket 连接，连接成功后发送“Hello 01Studio！”给服务器。
client=usocket.socket()

addr=('192.168.1.103',10000) #服务器 IP 和端口
client.connect(addr)
client.send('Hello 01Studio!')
client.settimeout(0.1)

#定时器回调函数
def fun(tim):
    global socket_node
    socket_node = 1 #改变 socket 标志位

#定时器 0 初始化，周期 100ms
tim = Timer(Timer.TIMER0, Timer.CHANNEL0, mode=Timer.MODE_PERIODIC,
           period=100, callback=fun)

while True:
    if socket_node:

        try:
            data = client.recv(256)
        except OSError:
            data = None

        if data:
            print("rcv:", len(data),data)

        socket_node = 0
```

```
    else: pass
```

WIFI 连接代码在上一节已经讲解，这里不再重复，程序在连接成功后建了 Socket 连接，连接成功发送‘Hello 01Studio!’信息到服务器。另外 K210 定时器设定了了每 100ms 处理从服务器接收到的数据。将接收到数据通过串口打印和重新发送给服务器。

- **实验结果：**

先在电脑端打开网络调试助手并建立服务器，软件在 零一科技（01Studio）MicroPython 开发套件配套资料\01-开发工具\01-Windows\网络调试助手 下的 NetAssist.exe ，直接双击打开即可！

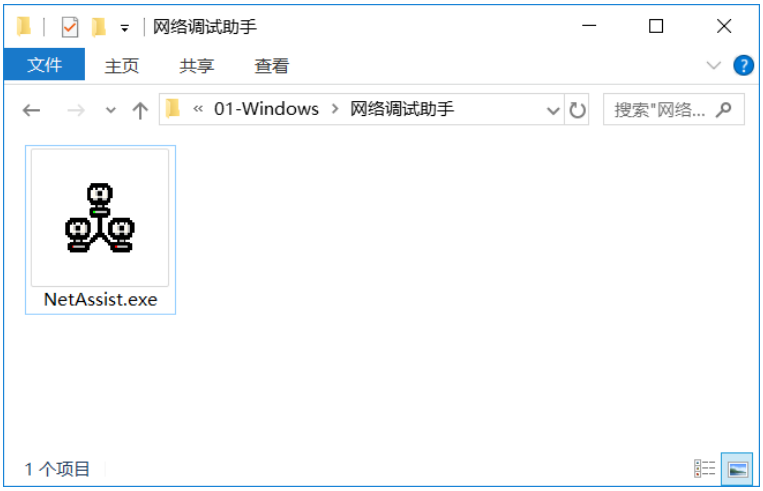


图 32-25 网络调试助手

以下是新建服务器的方法，打开网络调试助手后在左上角协议类型选择 TCP Server；中间的本地 IP 地址是自动识别的，不要修改，这个就是服务器的 IP 地址。然后端口写 10000（0-65535 都可以。），点击连接，成功后红点亮。如下图：



图 32-26 TCP 服务器配置

在时候服务器已经在监听状态！用户需要根据自己的实际情况自己输入 WIFI 信息和服务器 IP 地址+端口。即修改上面的代码以下部分内容。（服务器 IP 和端口可以在网络调试助手找到。）

```
# WiFi 信息

SSID='01Studio' # Network SSID

KEY='88888888'  # Network key
```

```
addr=('192.168.1.115',10000) #服务器 IP 和端口
```

运行程序，开发板成功连接 WIFI 后，发起了 socket 连接，连接成功可以可以看到网络调试助手收到了开发板发来的信息。在下方列表多了一个连接对象，点击选中：

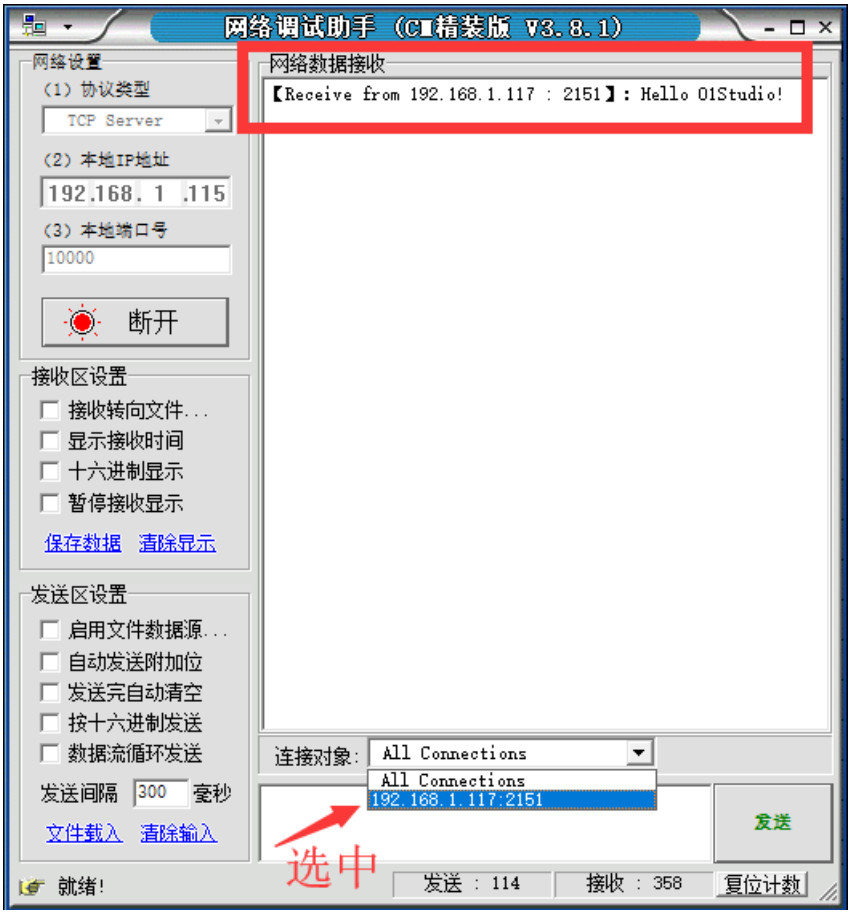


图 32-27 选中连接对象

选中后我们在发送框输入信息“http://www.01studio.org”，点击发送，可以看到开发板的串行终端打印出来该信息，类型为字节数据。

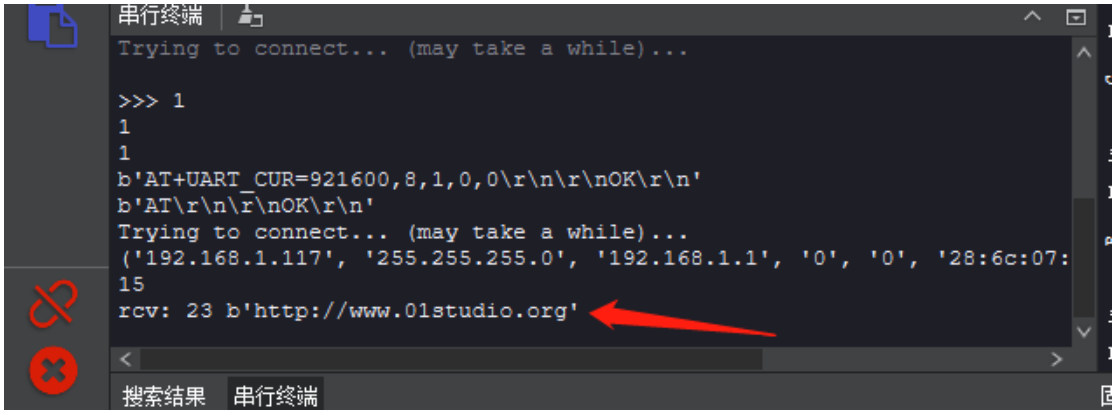


图 32-28

- **总结：**

通过本节学习，我们了解了 socket 通信原理以及使用 MicroPython 进行 socket 编程并且通信的实验。得益于优秀的封装，让我们可以直接面向 socket 对象编程就可以快速实现 socket 通信，从而开发更多的网络应用，例如将前面采集到的传感器数据发送到服务器。

32.4.3 MQTT 通信

- **前言：**

上一节，我们学习了 Socket 通信，当服务器和客户端建立起连接时，就可以相互通信了。在互联网应用大多使用 WebSocket 接口来传输数据。而在物联网应用中，常常出现这样的情况：海量的传感器，需要时刻保持在线，传输数据量非常低，有着大量用户使用。如果仍然使用 socket 作为通信，那么服务器的压力和通讯框架的设计随着数量的上升将变得异常复杂！

那么有无一个框架协议来解决这个问题呢，答案是有的。那就是 MQTT(消息队列遥测传输)。

- **实验平台：**

pyAI-K210 和 K210 WiFi 模块。



图 32-29 pyAI-OpenMV4

- **实验目的：**

通过编程让 pyAI-K210 实现 MQTT 协议信息的发布和订阅（接收）。

- **实验讲解：**

MQTT 是 IBM 于 1999 年提出的，和 HTTP 一样属于应用层，它工作在 TCP/IP

协议族上，通常还会调用 socket 接口。是一个基于客户端-服务器的消息发布/订阅传输协议。其特点是协议是轻量、简单、开放和易于实现的，这些特点使它适用范围非常广泛。在很多情况下，包括受限的环境中，如：机器与机器（M2M）通信和物联网（IoT）。其在，通过卫星链路通信传感器、偶尔拨号的医疗设备、智能家居、及一些小型化设备中已广泛使用。

总结下来 MQTT 有如下特性/优势：

- **异步消息协议**
- **面向长连接**
- **双向数据传输**
- **协议轻量级**
- **被动数据获取**

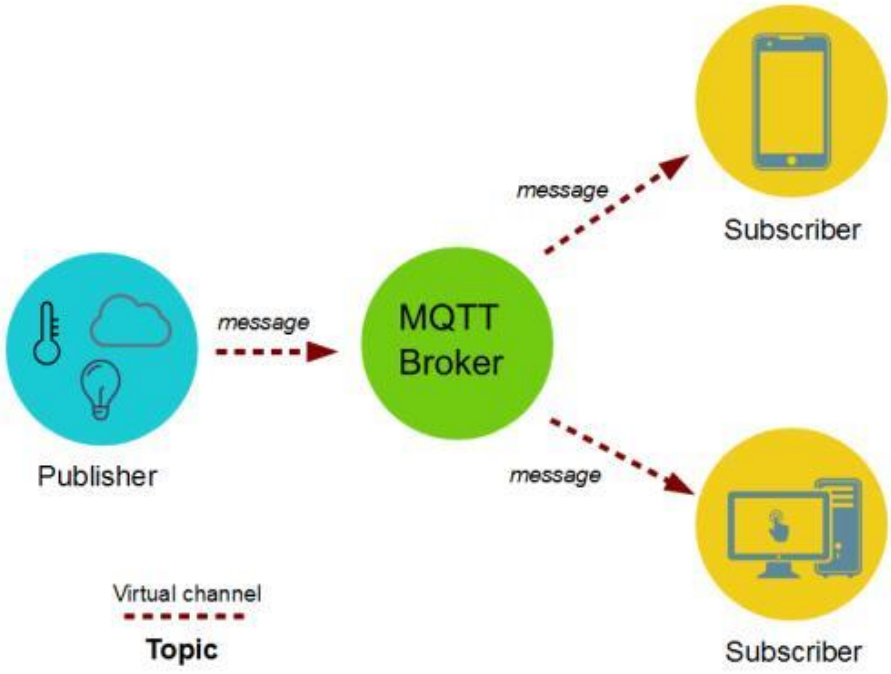


图 32-30 MQTT 通信流程

从上图可以看到，MQTT 通信的角色有两个，分别是服务器和客户端。服务器只负责中转数据，不做存储；客户端可以是信息发送者或订阅者，也可以同时是两者。具体如下图：

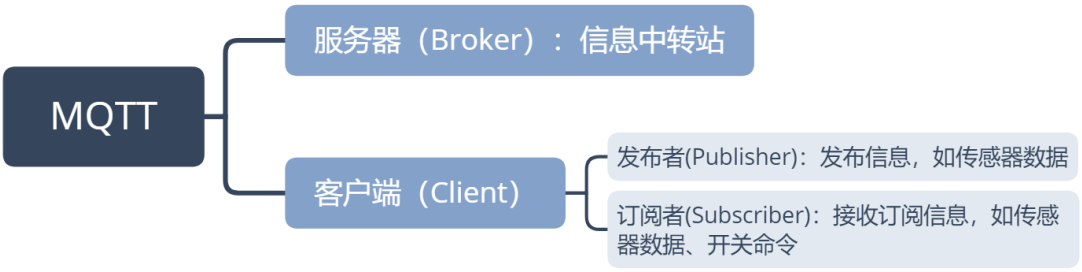


图 32-31　MQTT 角色说明

确定了角色后是如何传输数据呢？下表示 MQTT 最基本的数据帧格式，例如温度传感器发布主题“Temperature”编号,消息是“25”（表示温度）。那么所有订阅了这个主题编号的客户端（手机应用）就会收到相关信息，从而实现通信。如下表所示：

MQTT 数据帧格式	
Topic ID (主题编号)	Message (消息)
Temperature	25

图 32-32 MQTT 数据格式

由于特殊的发布/订阅机制，服务器不需要存储数据（当然也可以在服务器的设备上建立一个客户端来订阅保存信息），因此非常适合海量设备的传输。

人生苦短，而 MicroPython 已经封装好了 MQTT 客户端的库文件。让我们的应用变得简单美妙。K210 的 MQTT 模块文件为例程文件夹里面的 simple.py 文件。使用方法如下：

构造函数
client=mqtt. MQTTClient (client_id, server, port)
构建 MQTT 客户端对象。 client_id: 客户端 ID，具有唯一性； server: 服务器地址，可以是 IP 或者网址； port:服务器端口。（默认是 1883，服务器通常采用的端口，也可以自定义。）
使用方法
`client.connect()`

连接到服务器。
client.publish(TOPIC,message)
发布。TOPIC：主题编号；message：信息内容，例：'Hello 01Studio!'
client.subscribe(TOPIC)
订阅。TOPIC：主题编号。
client.set_callback(callback)
设置回调函数。callback：订阅后如果接收到信息，就执行相名称的回调函数。
client.check_msg()
检查订阅信息。如收到信息就执行设置过的回调函数 callback。

表 32-6 MQTT 客户端对象

由于客户端分为发布者和订阅者角色，因此为了方便大家更好理解，本实验分开两个案例来编程，分别为发布者和订阅者。再结合 MQTT 网络调试助手来测试。代表编写流程图如下：

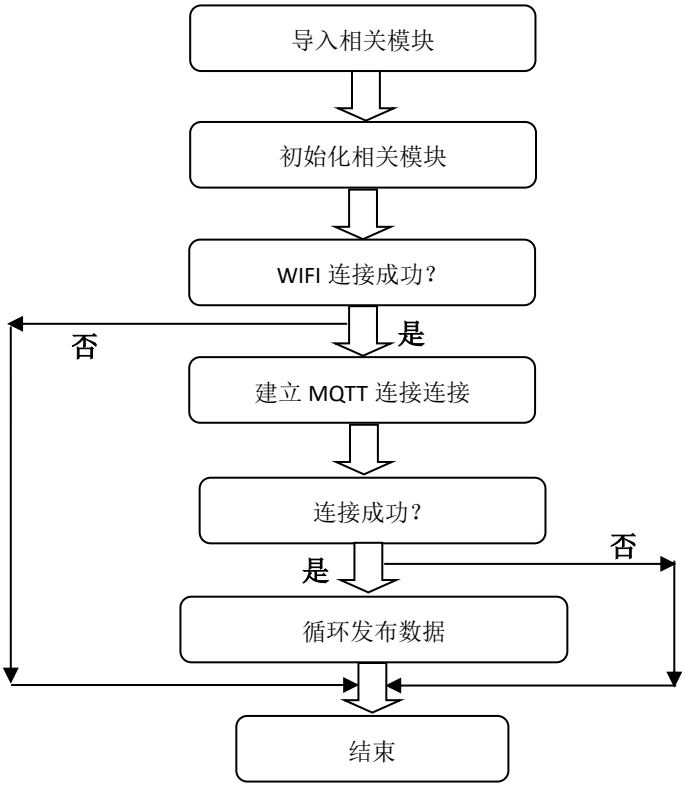


图 32-33 发布者代码编写流程

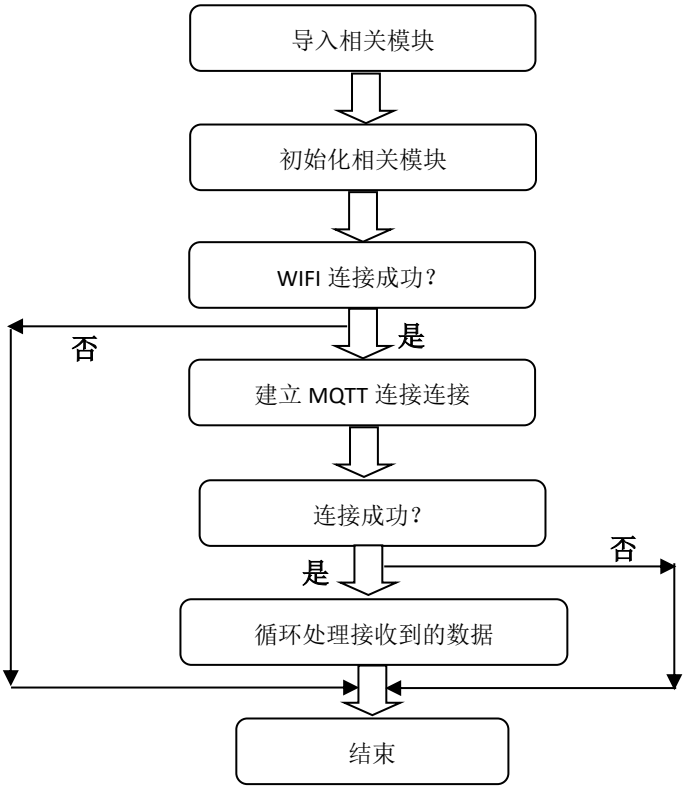


图 32-34 订阅者代码编写流程

发布者（publish）参考代码：

```
'''
实验名称：MQTT 通信
版本：v1.0
日期：2019.8
作者：01Studio
说明：编程实现 MQTT 通信，实现发布数据。
'''

import network,time
from machine import UART,Timer
from Maix import GPIO
from fpioa_manager import fm
from simple import MQTTClient
```

```
SSID='01Studio' # WiFi 账号
KEY='88888888'  # WiFi 密码

###### WiFi 模块初始化 ######
#使能引脚初始化
fm.register(8, fm.fpioa.GPIOHS0, force=True)
wifi_en=GPIO(GPIO.GPIOHS0, GPIO.OUT)

#串口初始化
fm.register(7, fm.fpioa.UART2_TX, force=True)
fm.register(6, fm.fpioa.UART2_RX, force=True)
uart = UART(UART.UART2,115200,timeout=1000,read_buf_len=4096)

#WiFi 使能函数
def wifi_enable(en):
    global wifi_en
    wifi_en.value(en)

def wifi_init():
    global uart
    wifi_enable(0)
    time.sleep_ms(200)
    wifi_enable(1)
    time.sleep(2)
    uart = UART(UART.UART2,115200,timeout=1000, read_buf_len=4096)
    tmp = uart.read()
    uart.write("AT+UART_CUR=921600,8,1,0,0\r\n")
    print(uart.read())
```

```
    uart = UART(UART.UART2,921600,timeout=1000, read_buf_len=10240) #
important! baudrate too low or read_buf_len too small will loose data
    uart.write("AT\r\n")
    tmp = uart.read()
    print(tmp)
    if not tmp.endswith("OK\r\n"):
        print("reset fail")
        return None
    try:
        nic = network.ESP8285(uart)
    except Exception:
        return None
    return nic

##############################
#########  主程序  ##########
##############################

#构建 WiFi 对象并使能
wlan = wifi_init()

#正在连接印提示
print("Trying to connect... (may take a while)...")

#连接网络
wlan.connect(SSID,KEY)

#打印 IP 相关信息
print(wlan.ifconfig())
```

```
#发布数据任务
def MQTT_Send(tim):
    client.publish(TOPIC, 'Hello 01Studio!')

SERVER = 'mqtt.p2hp.com'
PORT = 1883
CLIENT_ID = '01Studio-K210' # 客户端 ID
TOPIC = '/public/01Studio/1' # TOPIC 名称
client = MQTTClient(CLIENT_ID, SERVER, PORT)
client.connect()

#定时器 0 初始化，周期 1 秒
tim = Timer(Timer.TIMER0, Timer.CHANNEL0, mode=Timer.MODE_PERIODIC,
            period=1000, callback=MQTT_Send)

while True:
    pass
```

订阅者（subscribe）参考代码：

```
# MQTT 订阅者（subscribe）例程.
#
#需要复制 mqtt.py 文件到 OpenMV 文件系统.
#
#翻译和注释：01Studio

import time, network
from mqtt import MQTTClient

SSID='WeBee_office_2.4G' # Network SSID
KEY='webee0123456789'  # Network key
```

```
# Init wlan module and connect to network
print("Trying to connect... (may take a while)...")

wlan = network.WINC()
wlan.connect(SSID, key=KEY, security=wlan.WPA_PSK)

# We should have a valid IP now via DHCP
print(wlan.ifconfig())

#设置 MQTT 回调函数,有信息时候执行
def MQTT_callback(topic, msg):
    print('topic: {}'.format(topic))
    print('msg: {}'.format(msg))

SERVER = 'mqtt.p2hp.com'
PORT = 1883
CLIENT_ID = '01Studio-OpenMV' # 客户端 ID
TOPIC = 'OpenMV/test' # TOPIC 名称



client = MQTTClient(CLIENT_ID, SERVER, PORT)
client.set_callback(MQTT_callback)  #配置回调函数
client.connect()
client.subscribe(TOPIC) #订阅主题

while (True):
    client.check_msg() #检测是否收到信息，收到则执行打印。
```

从以上代码可以看到发布者和订阅者的编程方式相近，另外本实验需要一个 MQTT 服务器（Broker），这里使用的是跟我们将用到的 MQTT 在线网络助手同一个服务器和端口。

```
SERVER = 'mqtt.p2hp.com'
PORT = 1883
```

- **实验结果：**

我们将 MQTT 示例程序中的 simple.py 文件拷贝到 K210 文件系统，然后在 MaixPy IDE 分别运行上述代码。

为了方便测试，我们可以使用 MQTT 网络助手进行调试。这里推荐一个在线 MQTT 网络调试助手：http://mqtt.p2hp.com/websocket/

打开上面网址，即可看到 MQTT 在线调试助手。可以配置基本信息，这里默认即可，点击连接。

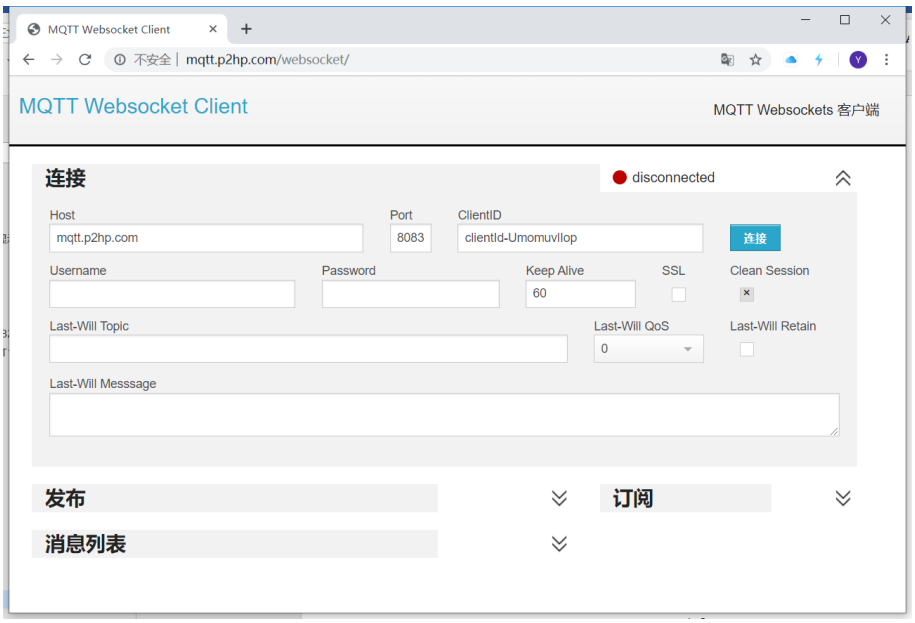


图 32-35 MQTT 助手

连接成功可以看到显示 Connected。

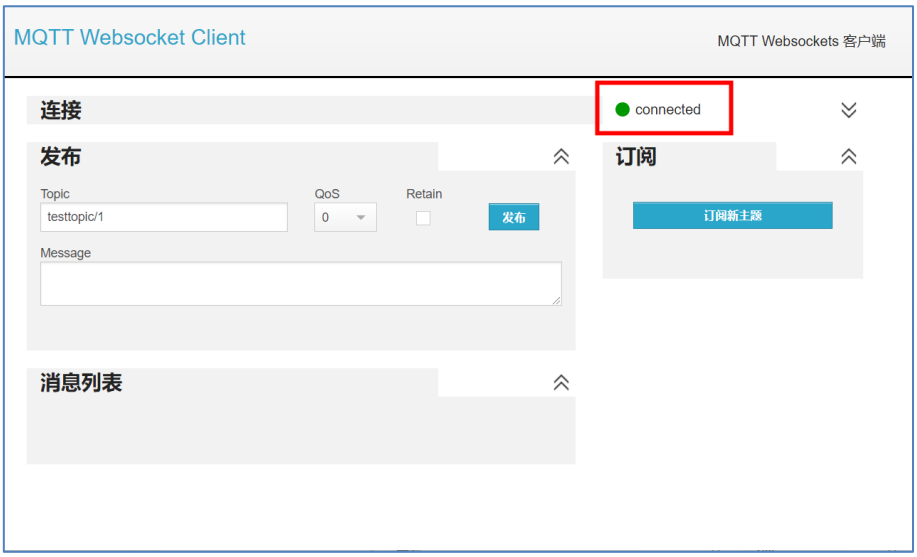


图 32-36 MQTT 助手连接成功

测试“发布者”代码：

我们先测试“发布者”代码。在 MQTT 助手中订阅主题修改为：'/public/01Studio/1'（跟代码发布的主题一致），QOS 选择 0 即可。然后点击订阅主题。

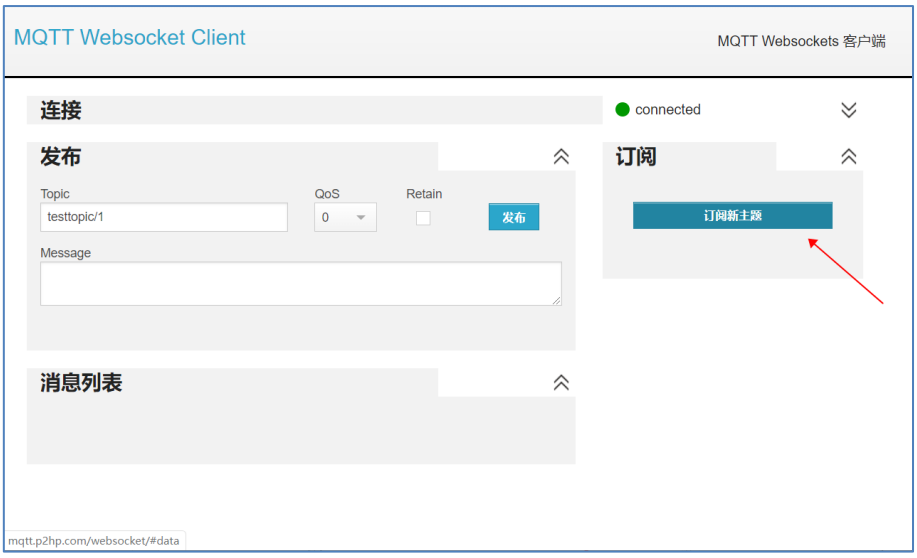


图 32-37 订阅主题

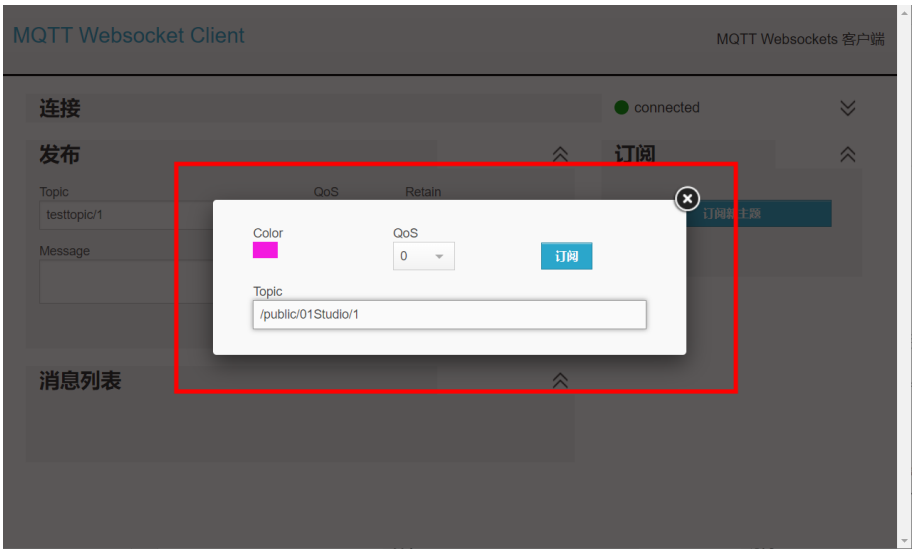


图 32-38

订阅后,运行程序，可以看到接收框收到来自开发板发布的信息。

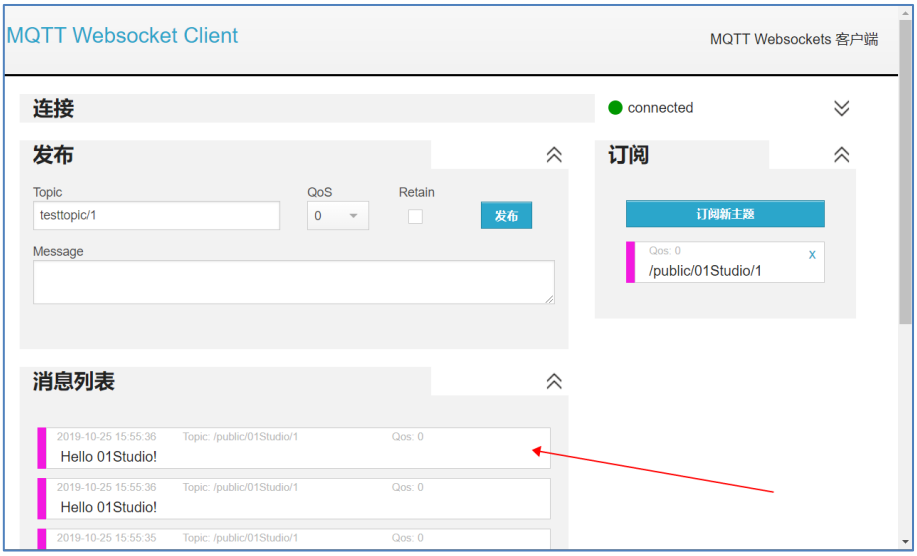


图 32-39 MQTT 助手收到开发板发布的信息

测试“订阅者”代码：

“订阅者”代码测试方法跟“发布者”相反。在电脑 MQTT 助手中发布主题修改为：'/public/01Studio/1'（跟代码发布的主题一致。）在下方空白框输入“Hello 01Studio!”。

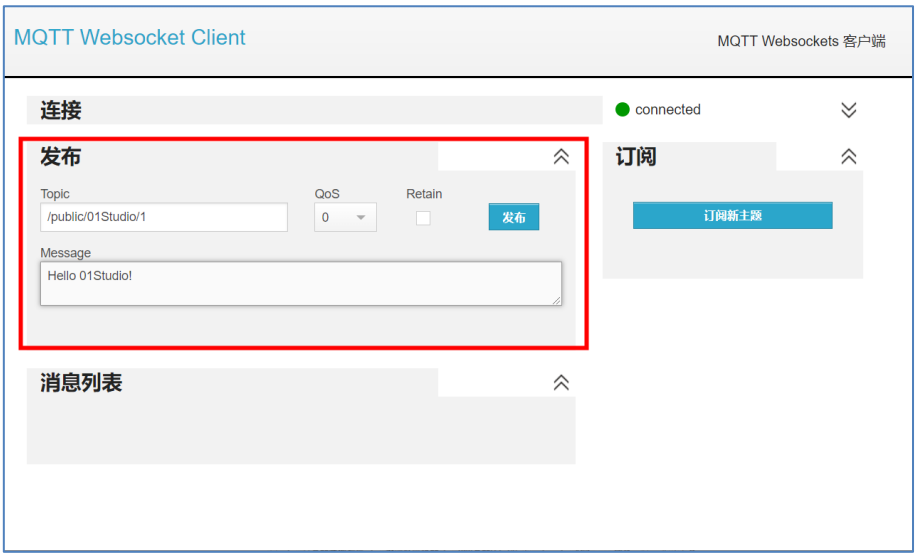


图 32-40 输入即将发布的主题和信息

开发板运行“订阅者”代码，成功连接后回到 MQTT 助手点击【发布】发布信息，可以在开发板的 REPL 看到接收到的订阅信息“Hello 01Studio!”被打印出来了（数据格式为字节数据）。

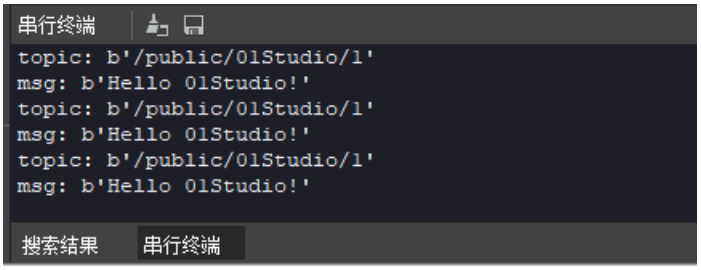


图 32-41 开发板接收到相关信息并打印

当使用 main.py 脱机运行时候，IDE 的串口终端和 putty 并不能打印信息。可以使用串口助手观察。波特率为 115200。

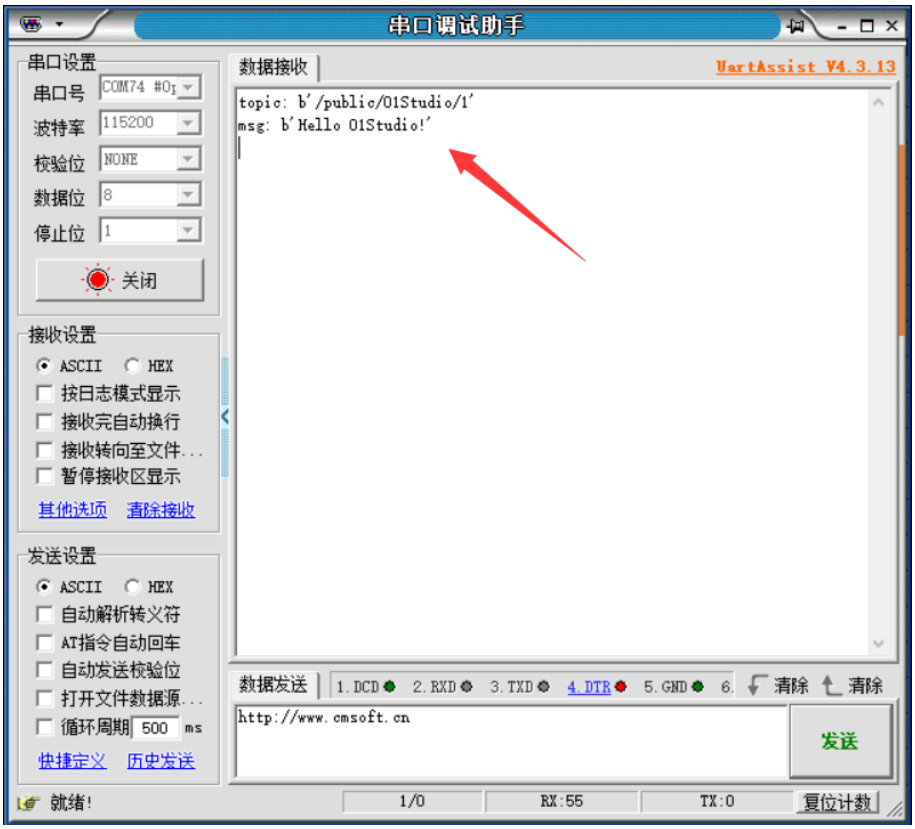


图 32-42 串口调试助手

当然你也可以在同一个 MQTT 在线助手下测试发布和订阅功能来做一些测试，只需要将主题设置一致即可，如下图所示：

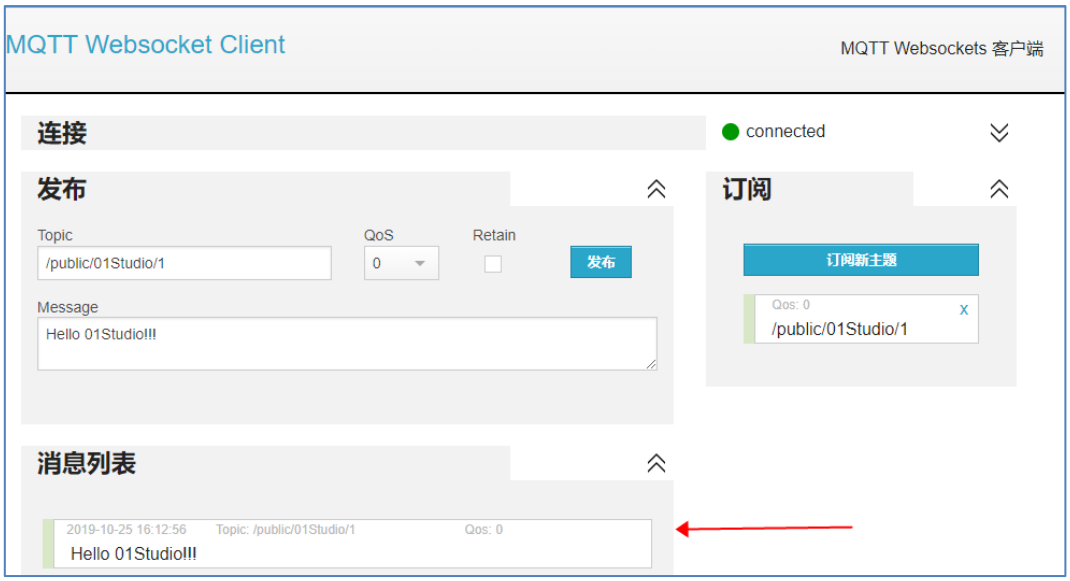


图 32-43 客户端同时发布和订阅信息

- **总结：**

通过本节我们了解了 MQTT 通信原理以及成功实现通信。目前市面上大部分物联网云平台支持 MQTT，原理大同小异。大家可以基于不同平台协议来开发，实现自己的物联网设备远程连接。

32.5 摄像头镜头

pyAI-K210 配套的摄像头模块镜头更换非常简单，只需要拧下原来镜头，将新的镜头拧上并调好焦距即可（黑色小圆环用于当调好焦距后固定摄像头位置）。

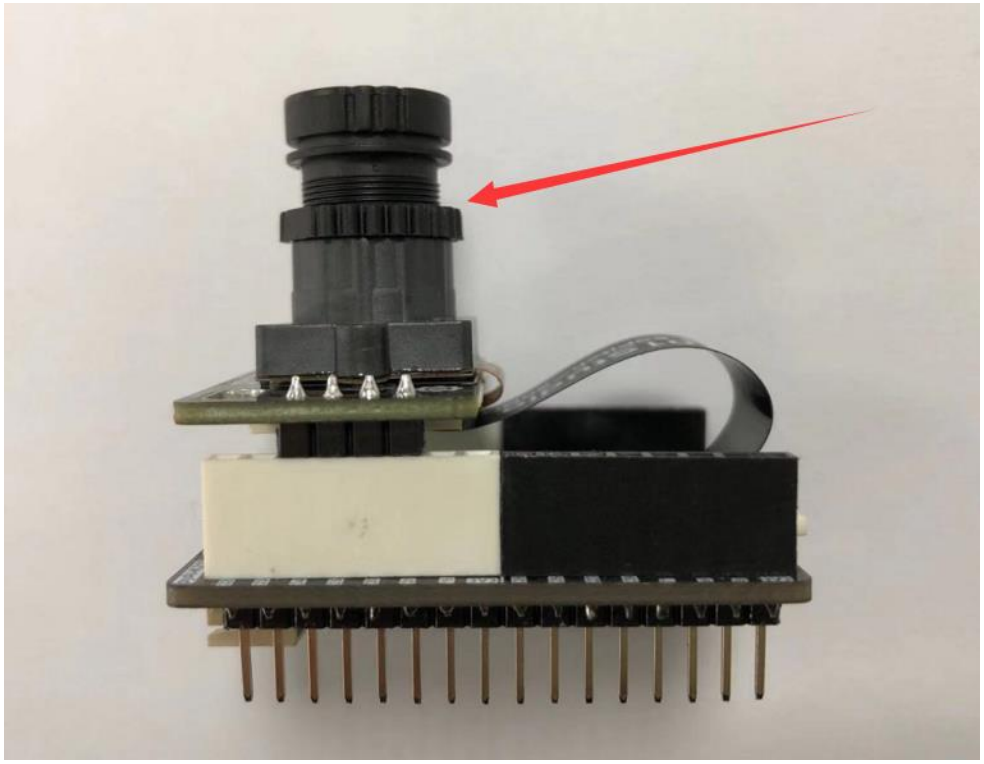


图 32-44

32.5.1 广角镜头

广角镜头是指可以让摄像头拍摄的范围变大，我们直接将广角镜头替换到 pyAI-K210 摄像头模块上，通过旋转调好焦距即可使用。



图 32-45 广角镜头

功能参数	
焦距	1.7mm
像素	5mp（500 万）
视场角	1/2.5’’：185°
接口	M12*0.5 mm
功能	广角拍摄
其它	含滤光片

表 32-7

以下是标准镜头和广角镜头拍摄对比。

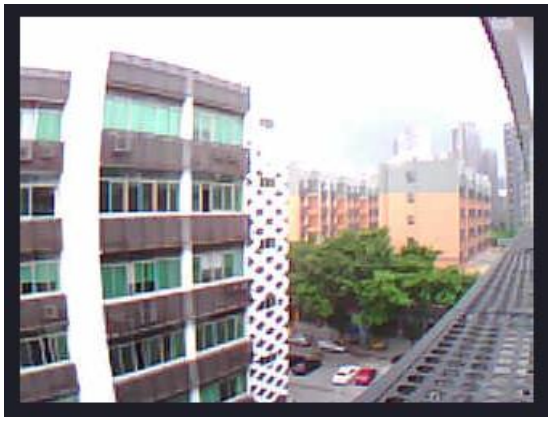


图 32-46 标准镜头



图 32-47 广角镜头

32.5.2 长焦镜头

长焦镜头可以理解成是望远镜，是指可以让摄像头拍摄远景，我们直接将长焦镜头替换到 pyAI-K210 摄像头模块上，通过旋转调好焦距即可使用。



图 32-48 长焦镜头

功能参数	
焦距	2.5mm
像素	300 万
视场角	D*H*V（1/3’’）：13*11*8
畸变率	1/3’’：-0.55%
功能	长焦（远景）拍摄
其它	含滤光片

表 32-8

以下是标准镜头和长焦镜头拍摄对比。

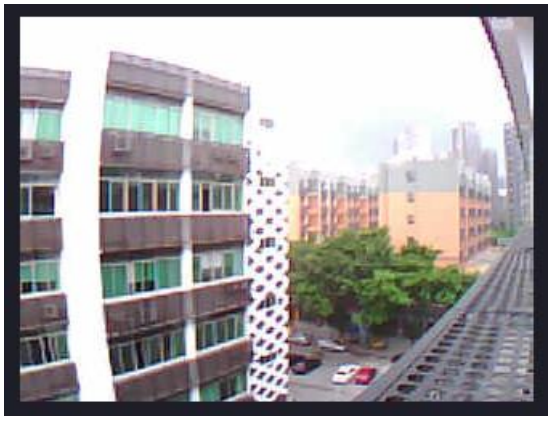


图 32-49 标准镜头



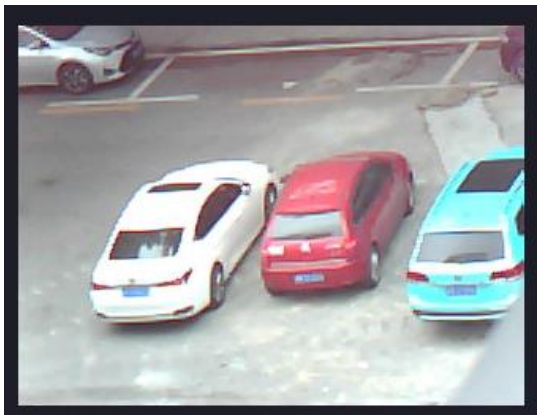


图 32-50 长焦镜头

32.5.3 无畸变镜头

标准镜头下的图片会出现一定的畸变（图像变形），而使用无畸变镜头可以解决这一问题，使拍摄出来的图片形状发生变形。我们直接将无畸变镜头替换到 pyAI-K210 摄像头模块上，通过旋转调好焦距即可使用。



图 32-51 无畸变镜头

功能参数	
焦距	3.6mm
像素	500 万
视场角	D*H*V（1/2.5’’）：96*81.8*65.8
畸变率	1/2.5’’：<0.7%（无畸变）
光圈	F2.8
功能	无畸变拍摄
其它	含滤光片

表 32-9

以下是标准镜头和无畸变镜头拍摄对比。明细看到无畸变摄像头拍摄出来的图像两侧毫无变形。



图 32-52 标准镜头

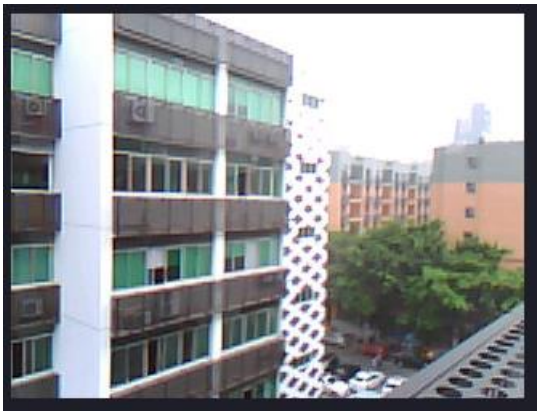


图 32-53 广角镜头

32.5.4 手动调焦镜头

手动调焦镜头可以让摄像头通过手动调节来获取最佳拍摄效果。我们直接将手动调焦镜头替换到 pyAI-K210 摄像头模块上，通过旋转调好焦距即可使用。



图 32-54 手动调焦镜头

功能参数	
焦距	2.8mm – 12mm（4 倍变焦）
像素	300 万
光圈类型	固定
功能	手动调焦拍摄
其它	含滤光片

表 32-10

以下是标准镜头和手动调焦镜头拍摄对比。下面对比图片看到差别不大，但手动调焦的优势是针对不同距离的拍摄都可以任意调焦到最清晰的位置，特别是近景。支持 2.8-12mm。

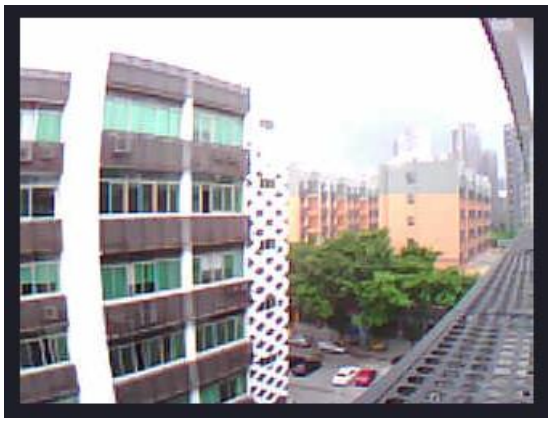


图 32-55 标准镜头

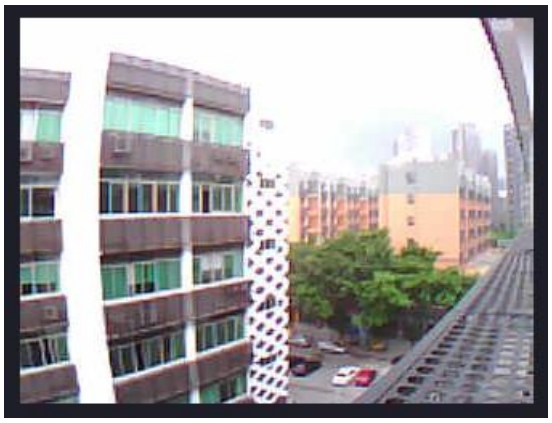


图 32-56 手动调节镜头

第33章　项目应用

33.1　照相机

- **前言：**

pyAI-K210 的外观非常酷，拥有按键功能，因此我们完全可以用来打造一台属于自己的照相机（尽管当前像素有点低）。现在无论数码相机还是手机都是带屏幕，主要是能实时显示方便拍摄，再配合锂电池，你就可以带上自制的照相机出去浪了！

- **实验平台：**

pyAI-K210 开发套件，需要使用 SD 卡和 LCD 显示屏。

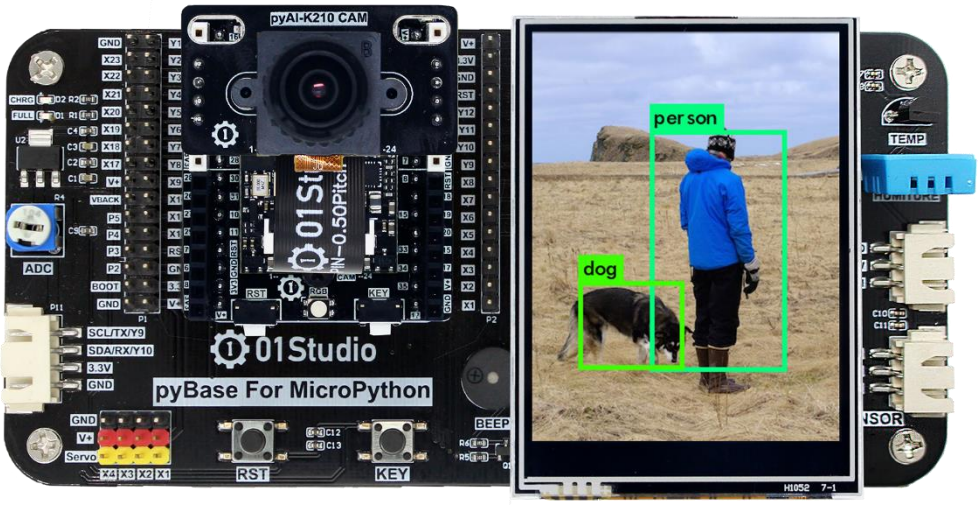


图 33-1 pyAI-OpenMV4 开发套件

- **实验目的：**

打造一台照相机，可以通过按键拍摄和 LCD 实时显示。

- **实验讲解：**

本项目主要是按键应用和拍照的相结合，这些内容可以在前面的实验找到，这里不再重复。

外部中断按键实验：请参阅 **29.4 外部中断** 章节内容；

拍摄照片实验：请参阅 **30.8 图片拍摄** 章节内容。

拍照后我们应该让图片停留一段时间，让用户观察照片的拍摄情况，然后再进行继续拍摄。代码编写流程如下：

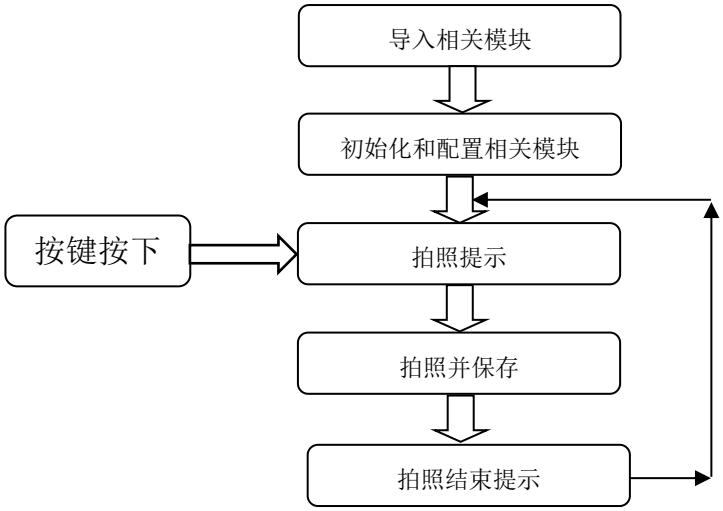


图 33-2 代码编写流程

参考代码如下：

```
'''
实验名称：照相机
版本：v1.0
日期：2019.12
作者：01Studio 【www.01Studio.org】
说明：通过按键拍照并在 LCD 上显示（本实验需要 SD 卡）。
'''

import sensor, lcd, utime
from Maix import GPIO
from fpioa_manager import fm
```

```
#注册 KEY 的外部 IO
fm.register(16, fm.fpioa.GPIOHS0, force=True)

#构建 KEY 对象
KEY=GPIO(GPIO.GPIOHS0, GPIO.IN, GPIO.PULL_UP)

#摄像头初始化
sensor.reset() # Initialize the camera sensor.
sensor.set_pixformat(sensor.RGB565) # or sensor.GRAYSCALE
sensor.set_framesize(sensor.QVGA) # or sensor.QVGA (or others)
sensor.skip_frames(30) # Let new settings take affect.
#sensor.set_vflip(1)    #摄像头后置模式

#LCD 初始化
lcd.init()

key_node = 0  #按键标志位
name_num = 0  #照片名字

################################################
#  按键和其回调函数
################################################
def fun(KEY):
    global key_node
    utime.sleep_ms(10) #消除抖动
    if KEY.value()==0: #确认按键被按下
        key_node = 1

#开启中断，下降沿触发
```

```
KEY.irq(fun, GPIO.IRQ_FALLING)

while True:

    lcd.display(sensor.snapshot()) # LCD 实时显示

    if key_node==1: #按键被按下
        key_node = 0 #清空按键标志位

        #拍照并保存，保存文件用时间来命名。
        lcd.display(sensor.snapshot().save(str(name_num)+".jpg"))
        name_num=name_num+1 #名字编码加 1

        print("Done! Reset the camera to see the saved image.")

        #延时 3 秒，观看拍摄图片
        utime.sleep_ms(3000)
```

- **实验结果：**

pyAI-K210 插入 SD 卡、接上 LCD 和锂电池。然后将以上代码编写好的 main.py 文件复制到 OpenMV4 文件系统中，按下复位即可运行。

然后对准要拍摄的东西，按下开发板的 KEY 按键，即可拍摄！

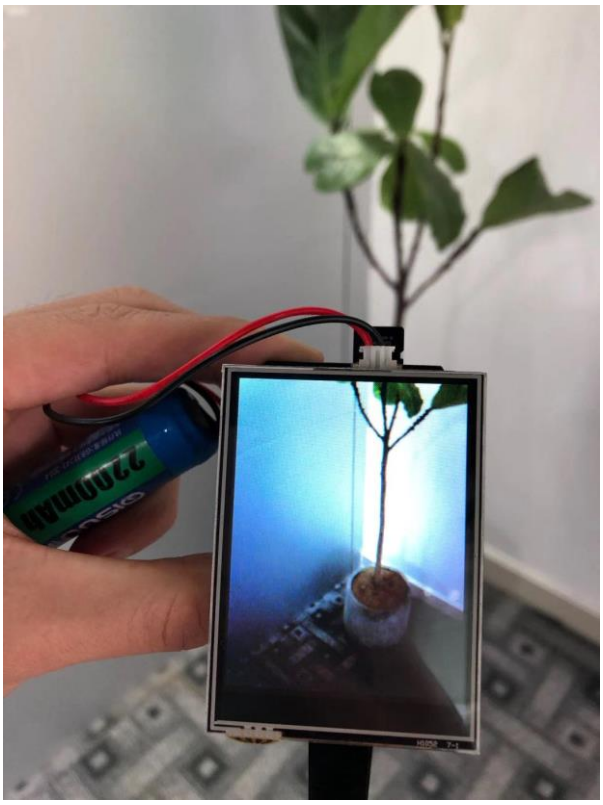


图 33-3 拍摄图片（摄像头后置模式）

当然你也可以结合 pyBase 底板来自拍：

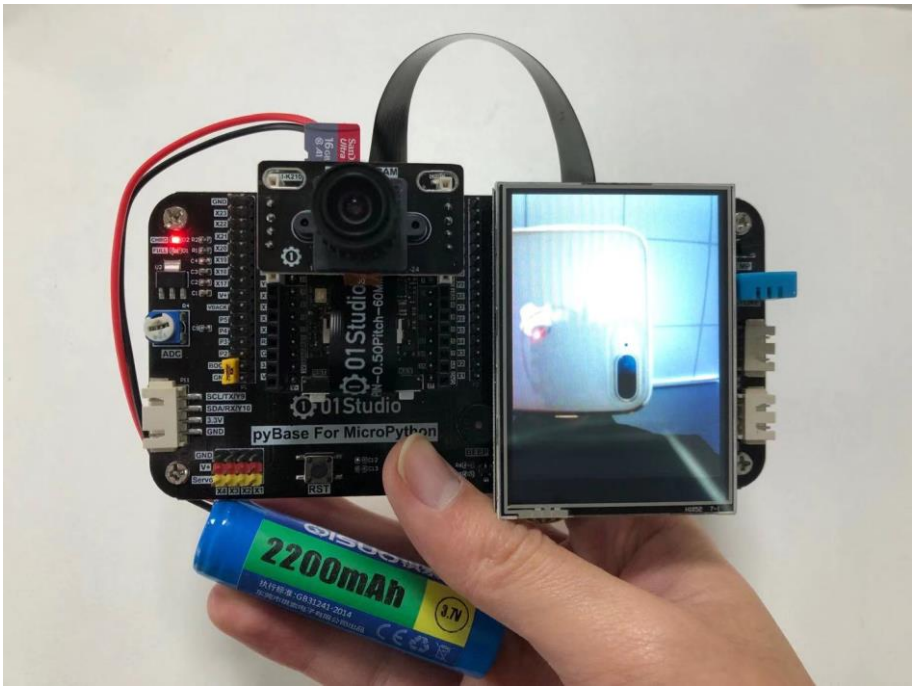


图 33-4 结合开发底板自拍（摄像头前置模块式）

拍摄完后取下 SD 卡，使用读卡器读取，可以看到被保存的拍摄 jpg 图片文件。



图 33-5 保存的图片文件

- **总结：**

本节学习了一个简单的项目应用，照相机。可以看到项目是前面实验的一个综合，只要把基础知识掌握牢固了，就可以轻松使用 pyAI-K210 实现更多好玩有趣的项目。同时 MaixPy 也是不断升级的产品，相信不久就能用上更高清的摄像头。

33.2 视频播放器

- **前言：**

由于 pyAI-K210 开发套件有 LCD 和音视频功能，那麼我们完全可以打造一个视频播放器了。可以理解成早期的 MP4。配合外置音箱，你甚至可以打造自己的多媒体播放器。

- **实验平台：**

pyAI-K210 开发套件+音频模块+sd 卡。

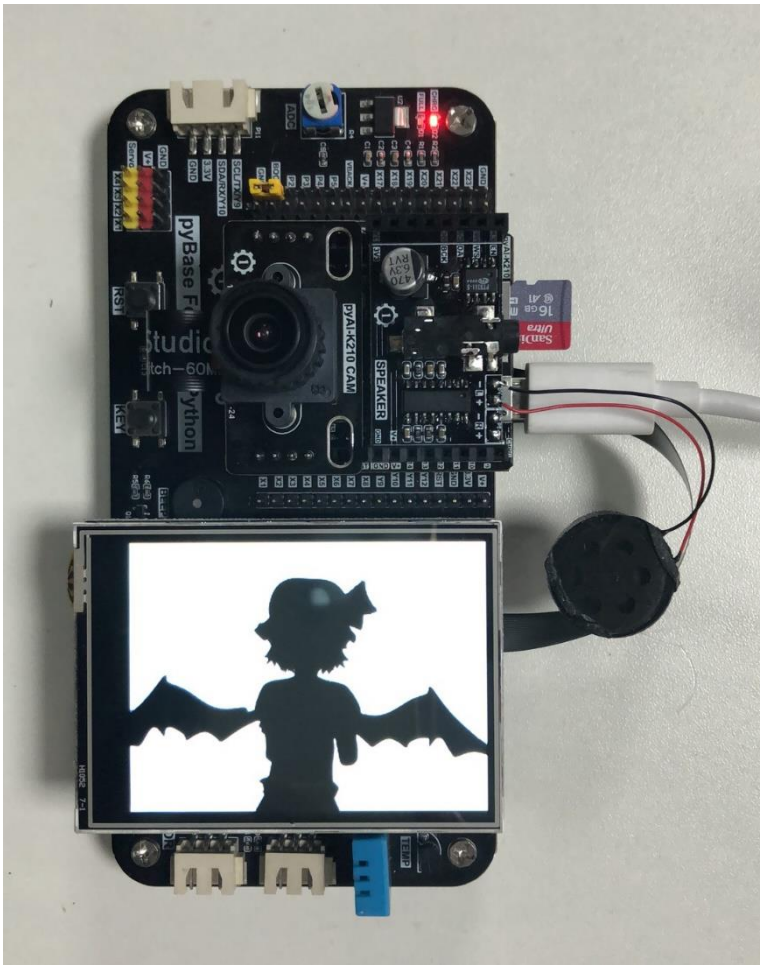


图 33-6 pyAI-K210 开发套件

- **实验目的：**

MicroPython 实现编程实现视频播放。

- **实验讲解：**

音视频解码是一个复杂的过程，但 K210 底层 MicroPython 库写好后，着重

应用来编程就变得非常简单了。和以往一样，我们只需要熟悉模块用法即可。

本实验实验 01Studio 音频模块，基于 PAM8403 的一款 D 类功放 IC，和麦克风一样使用 I2S 接口通信，这里不再重复 I2S 内容。

而视频播放被封装成 video 模块，在前面视频录制章节内容已经介绍过，这里重温一下，模块说明如下：

构造函数
import vedio v=vedio.open((path, record=False, interval=100000, quality=50, width=320, height=240, audio=False, sample_rate=44100, channels=1))
播放或录制视频文件。 【path】文件路径，比如：/sd/badapple.avi； 【record】=True 表示视频录制，=False 表示视频播放； 【interval】录制帧间隔，单位是微妙；FPS=1000000/interval，默认值是 100000，即 FPS 默认是 10（每秒 10 帧）； 【quality】jpeg 压缩质量（%），默认 50； 【width】录制屏幕宽度，默认 320； 【height】录制屏幕高度，默认 240； 【audio】是否录制音频，默认 False; 【sample_rate】录制音频采样率，默认 44100（44.1k）； 【channels】录制音频声道数，默认 1，即单声道。
使用方法
v.play()
播放视频；
v.volume([value])
设置音量值。 【value】0-100;
v.revord ()
录制音视频；

v.revord_finish ()
停止录制；
*更多使用说明请阅读官方文档： https://maixpy.sipeed.com/zh/libs/machine_vision/video.html

表 33-1 video 对象

代码编写流程如下：

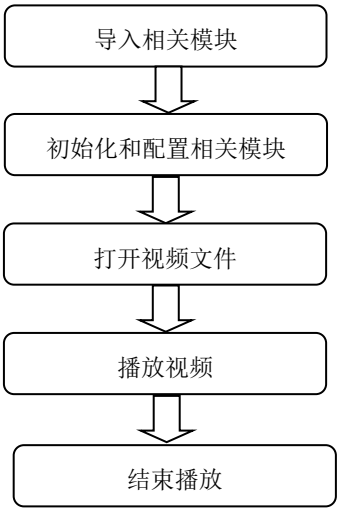


图 33-7 代码编写流程

参考代码如下：

```
'''
实验名称：视频播放器
版本：v1.0
日期：2019.12
翻译和注释：01Studio
说明：AVI 视频播放。
'''
import video,time
from Maix import GPIO
from board import board_info
from fpioa_manager import fm
```

```
import lcd


lcd.init()


# 音频使能 IO
AUDIO_PA_EN_PIN = 32


#注册音频使能 IO
if AUDIO_PA_EN_PIN:
    fm.register(AUDIO_PA_EN_PIN, fm.fpioa.GPIO1, force=True)
    wifi_en=GPIO(GPIO.GPIO1, GPIO.OUT)
    wifi_en.value(1)


#注册音频控制 IO
fm.register(34,  fm.fpioa.I2S0_OUT_D1, force=True)
fm.register(35,  fm.fpioa.I2S0_SCLK, force=True)
fm.register(33,  fm.fpioa.I2S0_WS, force=True)


#播放 avi 文件
v = video.open("/sd/badapple.avi")


#打印视频文件信息
print(v)


#音量调节
v.volume(5)


while True:
    if v.play() == 0: #播放完毕
        print("play end")
```

```
        break

v.__del__() #销毁对象，释放内存
```

- **实验结果：**

本实验播放的视频是 badapple.avi，文件在本例程文件夹中，先将该文件拷贝到 sd 卡。然后将 sd 卡插到 pyAI-K210。

接上 01Studio 音频模块，运行本实验程序代码，可以见到串口终端打印了 avi 视频信息后，开发板便开始播放视频。

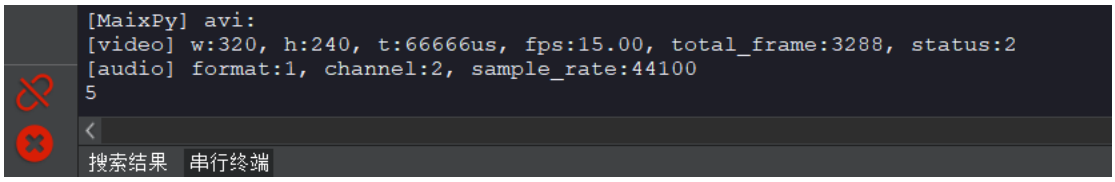


图 33-8

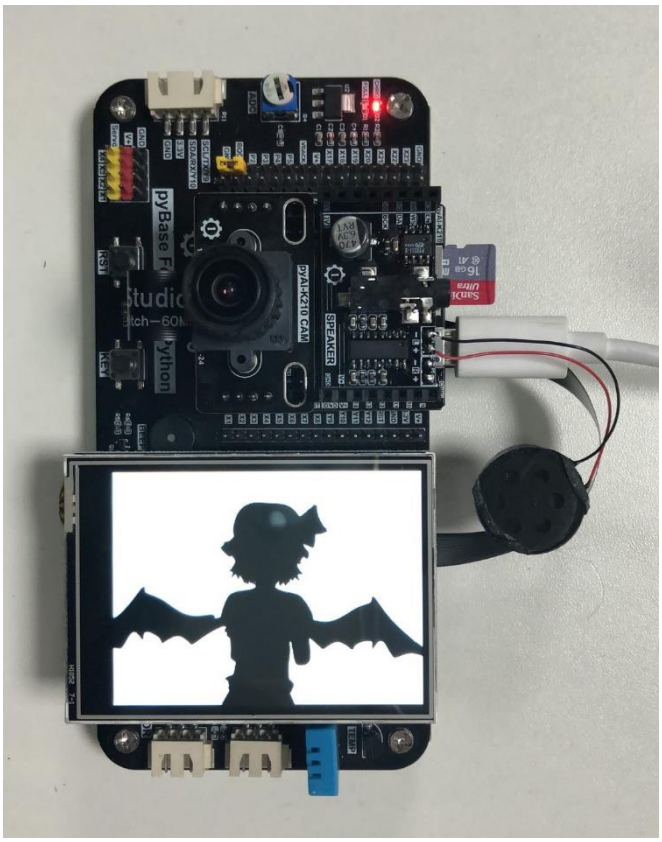


图 33-9 视频播放实验

01Studio 音频模块支持扬声器外接，标准 3.5mm 音频接口，我们也可以外接扬声器播放。

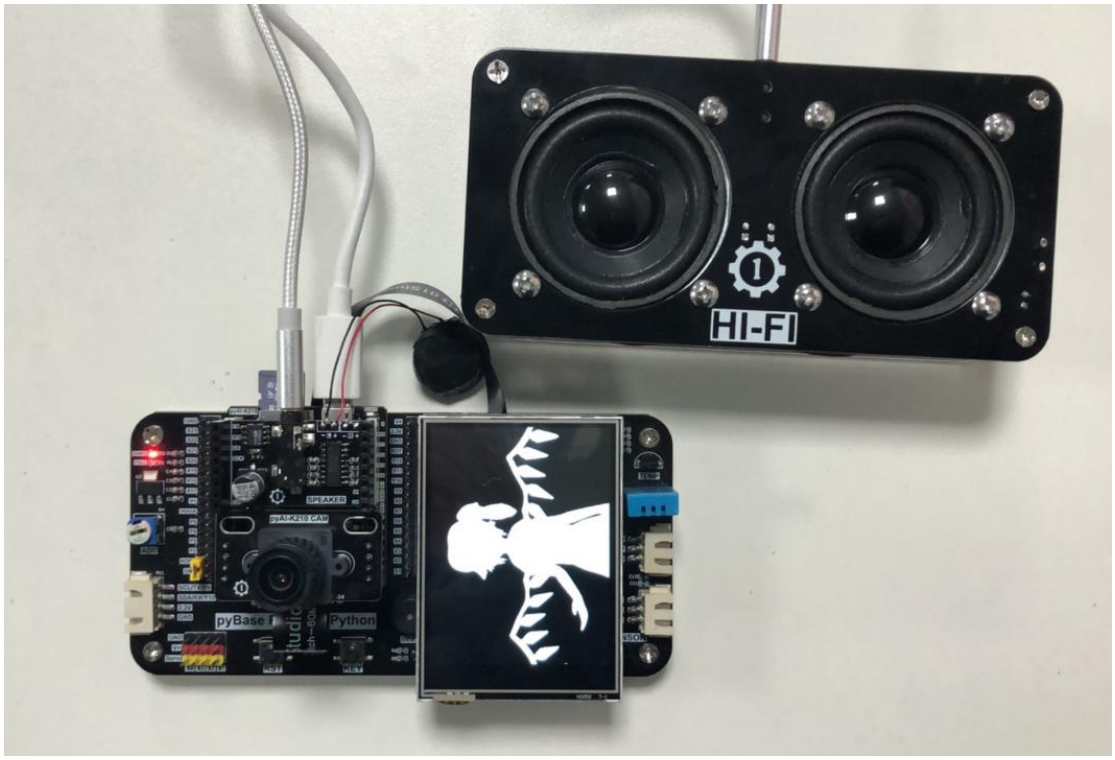


图 33-10 外接扬声器播放

- **总结：**

本节学习了视频播放应用，可以看到 MicroPython 编程让 K210 上的视频播放变得非常简单，而且流畅度也很棒，有了这个技能，我们可以打造自己的视频播放器。

33.3 NES 游戏机

- **前言：**

超级玛丽、魂斗罗等经典游戏是不少 80、90 后的童年，还记得过去满大街的小霸王游戏机，和同学交换游戏卡的时候吗，今天我们就来使用 paAI-K210 开发套件来实现 NES 游戏模拟，重温童年的乐趣。

- **实验平台：**

pyAI-K210 开发套件。需要使用 SD 卡和 LCD 显示屏。

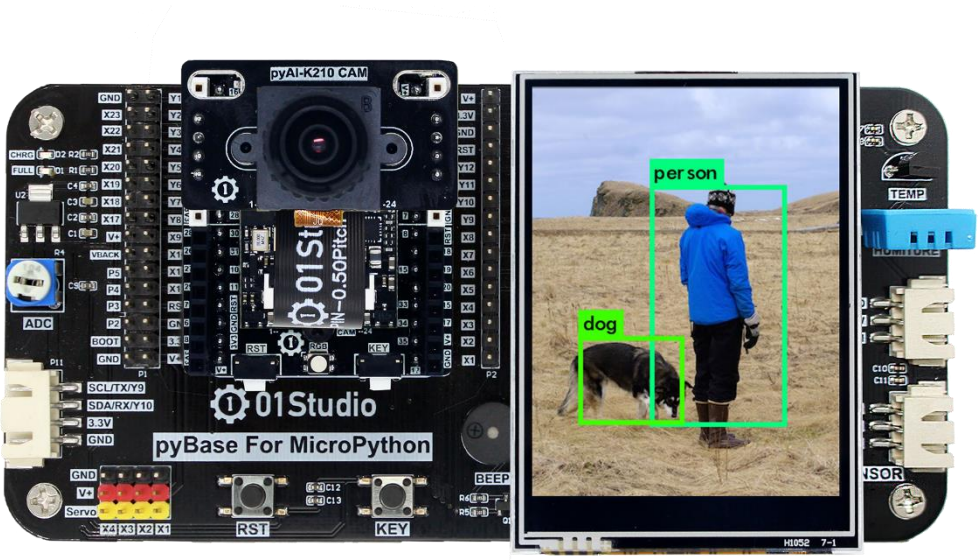


图 33-11 pyAI-OpenMV4 开发套件

- **实验目的：**

编程实验 NES 游戏模拟，加载 NES 游戏。

- **实验讲解：**

以往我们会觉得这类游戏很强大，通关永远通不完，但其实这类游戏的 nes 格式文件非常小，只有几十到几百 KB。

MaixPy 集成了 NES 的 MicroPython 模块,用户通过几行代码就可以实现游戏的加载，已经使用键盘或者标准游戏手柄来操控。NES 对象如下：

构造函数
import nes
导入 nes 模块；
使用方法
nes.init(rc_type=nes.KEYBOARD, cs, mosi, miso, clk, repeat=16, vol=5)
初始化 nes 游戏模拟器； 【rc_type】遥控类型。nes.KEYBOARD:REPL 中使用键盘；nes.JOYSTICK:PS2 手柄。 【cs,mosi,miso,clk】使用 PS2 手柄时的引脚配置； 【repeat】键盘按键重复率； 【vol】音量
nes.run(xx.nes)
运行 nes 文件。
*更多使用说明请阅读官方文档： https://maixpy.sipeed.com/zh/application/nes.html

表 33-2 NES 对象

键盘和手柄的快捷键如下：

键盘（串口）
【移动】：WSAD(上下左右) 【A】：J 【B】：K 【start】：M 或 Enter 【option】：N 或\ 【退出】：ESC 【音量-】：- 【音量+】：= 【运行速度-】：R 【运行速度+】：F

PS2 手柄
【移动】：方向键(上下左右) 【A】：口 【B】：X 【start】：START 【option】：SELECT 【退出】：暂无 【音量-】：R2 【音量+】：R1 【运行速度-】：L1 【运行速度+】：L2

表 33-3 游戏按键

从上表 NES 对象看到，只需要简单的初始化和运行语句，即可运行 NES 游戏模拟器，编程思路如下：

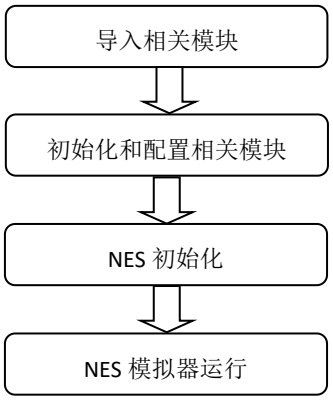


图 33-12 代码编写流程

参考代码如下：

```
'''
实验名称：NES 游戏模拟器
版本：v1.0
日期：2019.12
```

```
说明：NES 模拟器运行游戏。
翻译和注释：01Studio
'''

import nes, lcd
from Maix import GPIO
from fpioa_manager import fm

# 音频使能 IO
AUDIO_PA_EN_PIN = 32

#注册音频使能 IO
if AUDIO_PA_EN_PIN:
    fm.register(AUDIO_PA_EN_PIN, fm.fpioa.GPIO1, force=True)

#LCD 初始化
lcd.init()

#初始化 nes，配置为键盘控制
nes.init(nes.KEYBOARD)

#运行游戏
nes.run("/sd/Bomberman.nes")
```

- **实验结果：**

本实验使用的 NES 文件 Wie 炸弹人，在示例程序文件夹下，先将 Bomberman.nes 文件拷贝到 SD 卡根目录下，然后运行上面程序。可以看到开发板出现游戏界面：

接上 01Studio 音频模块，还能播放游戏声音。



图 33-13 NES 游戏实验

- **总结：**

从本实验可以看到，基于 MicroPython 的 NES 游戏模拟器编程非常简单，只需要简单的初始化和运行即可，背后是底层工程师做了大量的代码移植工作，让我们顶层用户直接应用。有兴趣的用户可以自行下载更多 nes 游戏文件，打造自己的游戏机。