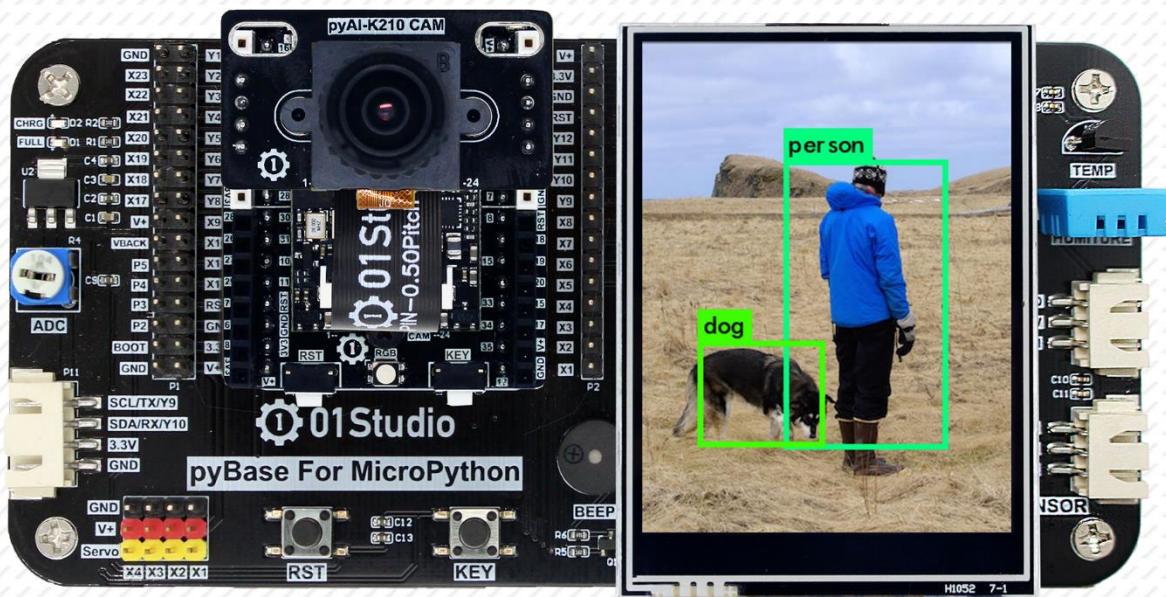


MicroPython 从0到1

用python做嵌入式编程

(基于K210平台)

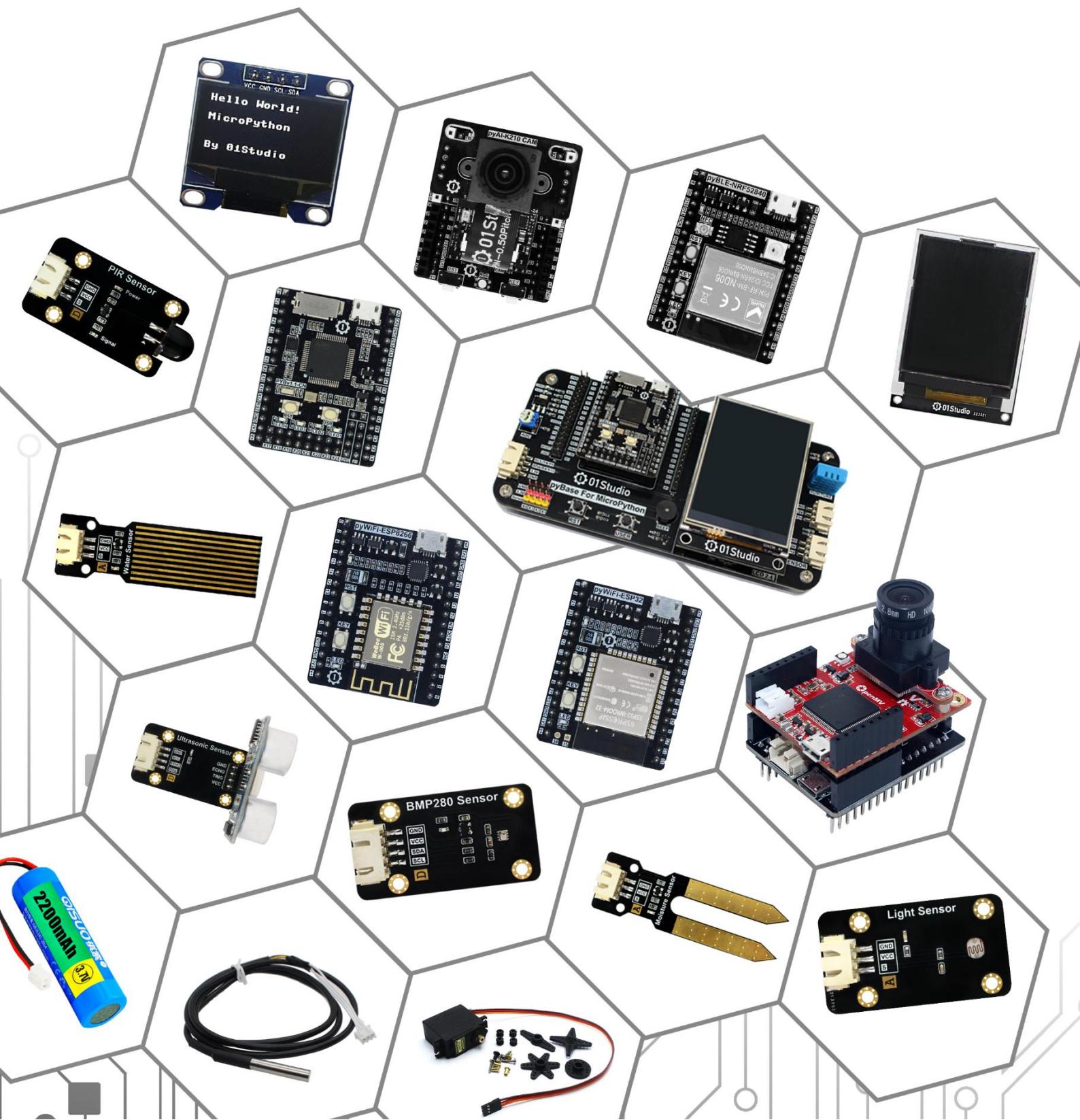
01Studio团队 编著



 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。而这一趋势并没有停止，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。

【官网：www.01Studio.cc】

【微信公众号：01Studio 社区】

版本说明

版本	日期	作者	更新内容
v1.0	2021-2-5	杰克船长	1. 第1版（重构）
v1.1	2022-4-13	杰克船长	1. 优化目录
v1.2	2022-8-31	杰克船长	1. 更新库文档链接

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 1 章 MicroPython 简介	7
1.1 MicroPython 是什么	7
1.2 MicroPython 支持的微控制器平台	8
1.3 MicroPython 相关学习资料	10
1.3.1 O1Studio 社区	10
1.3.2 MicroPython 官方网站	11
1.3.3 MaixPy 库文档	11
1.4 pyAI-K210 开发套件介绍	12
1.4.1 K210 平台	13
1.4.2 pyBase	16
1.4.3 IOT/通讯模块	18
1.4.4 拓展配件	21
第 2 章 Python 基础知识	26
2.1 原始数据类型和运算符	26
2.2 变量和集合	31
2.3 流程控制和迭代器	37
2.4 函数	41
2.5 类	44
2.6 模块	46
2.7 高级用法	47
第 3 章 开发环境快速搭建	49
3.1 基于 Windows	50
3.1.1 摄像头接线	50
3.1.2 LCD 接线	51
3.1.3 安装开发软件 MaixPy IDE	53
3.1.4 驱动安装	54
3.1.5 例程测试	58
3.1.6 REPL 串口调试	63
3.1.7 文件系统	68
3.1.8 固件更新	70
3.2 基于 Mac OS	73
3.2.1 安装开发软件 MaixPy IDE	73
3.3 基于 Linux	74
3.3.1 安装 MaixPy IDE	74
第 4 章 基础实验	75
4.1 点亮第一个 LED	76
4.2 流水灯	82
4.3 按键	88
4.4 外部中断	93
4.5 定时器	99
4.6 PWM	104
4.7 I2C 总线（OLED 显示屏）	111

4.8 UART (串口通信)	118
4.9 thread (线程)	125
第 5 章 机器视觉	128
5.1 LCD	128
5.2 摄像头应用	137
5.3 画图	145
5.4 颜色识别	151
5.5 二维码识别	156
5.6 人脸检测	161
5.7 物体识别	169
5.8 在线训练模型	175
5.9 图片拍摄	176
5.10 视频录制	181
第 6 章 机器听觉	187
6.1 声音频率识别 (FFT)	187
6.2 声源定位	195
第 7 章 拓展模块	200
7.1 电阻触摸屏	200
7.2 舵机	206
7.3 RGB 灯带	213
7.4 WiFi 模块	219
7.4.1 连接无线路由器	220
7.4.2 Socket 通信	225
7.4.3 MQTT 通信	238
7.5 摄像头镜头	255
7.5.1 广角镜头	256
7.5.2 长焦镜头	258
7.5.3 无畸变镜头	260
7.5.4 手动调焦镜头	262
第 8 章 项目应用	264
8.1 照相机	264
8.2 视频播放器	270
8.3 NES 游戏机	276

第1章 MicroPython 简介

1.1 MicroPython 是什么

第一次接触 MicroPython 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。

MicroPython 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，Micropython 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。

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 MicroPython 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。本书主要是围绕 K210 平台来进行编写。

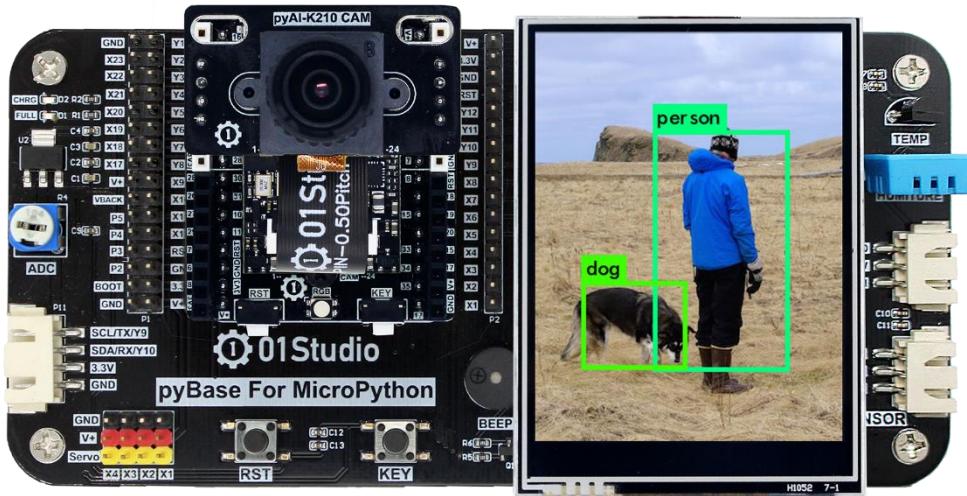


图 1-1 pyAI-K210+摄像头模块

以下是 01Studio 其它 micropython 开发平台：

01Studio MicroPython全系列开发板			
pyBoard (STM32F405)	Micro-Python开发板 	pyboard v1.1-CN 	pyBoard Mini 
哥伦布 (STM32F407)	哥伦布STM32F407开发板 	pyBoard Pro 	pyBoard Plus 
麦哲伦 (STM32H743)	麦哲伦STM32H743开发板 	STM32H743核心板 	
达芬奇 (TKM32F499)	达芬奇TKM32F499开发板 		
pyPico (RP2040)	pyPico开发套件 		
pyWiFi-ESP8266	PYTHON-ESP8266开发套件 	PYTHON-ESP8266 	
pyWiFi-ESP32-C3	PYTHON-ESP32-C3开发套件 	PYTHON-ESP32-C3 	
pyWiFi-ESP32	PYTHON-ESP32开发套件 	PYTHON-ESP32 	PYTHON-ESP32D 
pyWiFi-ESP32-S2	PYTHON-ESP32-S2开发套件 		
pyWiFi-ESP32-S3	PYTHON-ESP32-S3开发套件 		
pyBLE-NRF52840	PYTHON-NRF52840开发套件 	PYTHON-NRF52840 	
py4G-EC600	PYTHON-EC600开发套件 		
pyAI-OpenMV4	PYTHON-OpenMV4开发套件 	PYTHON-OpenMV4 Plus 	
pyAI-K210	PYTHON-K210开发套件 	PYTHON-K210核心板 	

1.3 MicroPython 相关学习资料

1.3.1 01Studio 社区

【论坛网址：bbs.01Studio.cc】

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。

The screenshot shows the homepage of the 01Studio forum. On the left, there's a sidebar with categories like 'MicroPython开发板' (11 posts) and 'Linux Python开发板' (3 posts). The main area has a search bar at the top. Below it, a post by user 'oxxxx' from August 26, 2021, with 11 replies, asks about a counter module. Another post by 'Jackey' from August 25, 2021, with 39 replies, discusses ESP32 temperature and humidity sensor data collection. To the right, a sidebar lists recommended posts, and at the bottom, there's a footer with copyright information.

图 1-2 01Studio 社区

1.3.2 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。

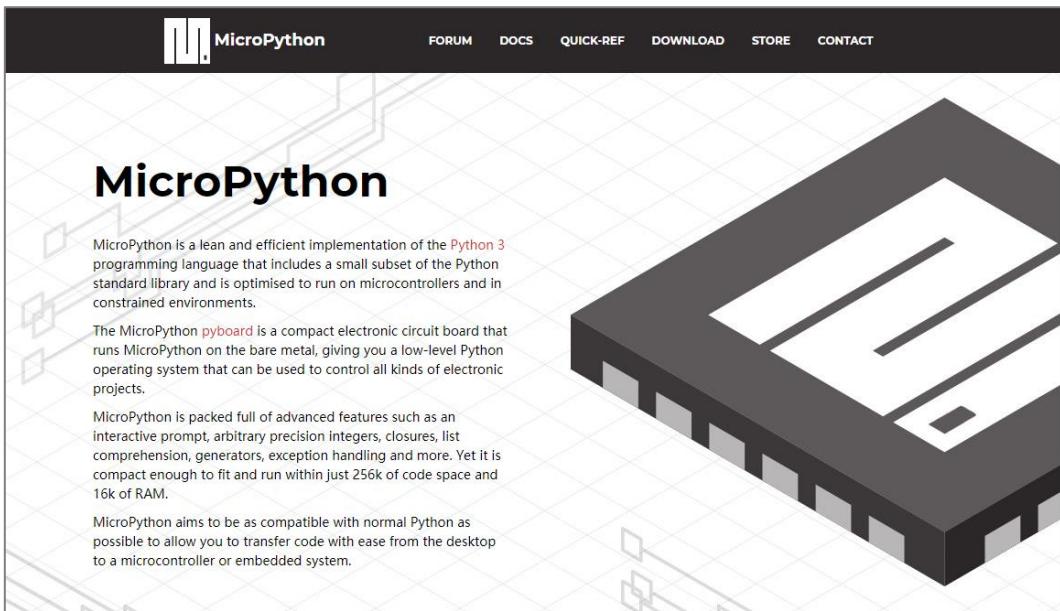


图 1-3 MicroPython 官网

1.3.3 MaixPy 库文档

本教程 K210 开发板基于 MaixPy 开源项目。教程中所用到的 Python 库都可以在文档中找到。

文档链接：<https://wiki.sipeed.com/soft/maixpy/zh/index.html>

1.4 pyAI-K210 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 MicroPython 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 K210 平台

1.4.1.1 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。

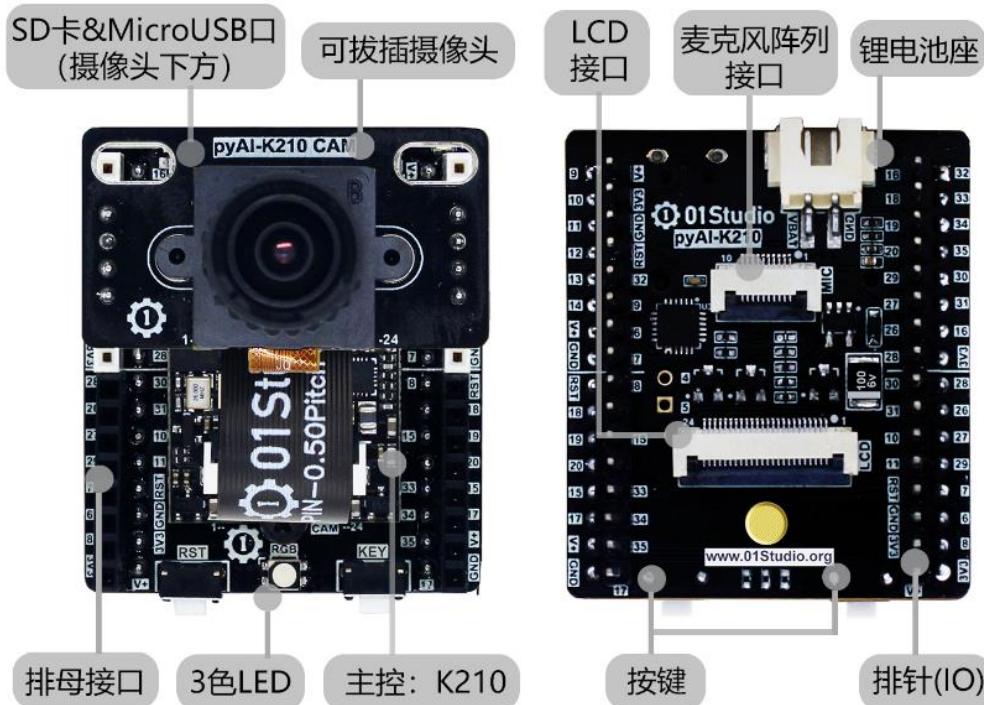


图 1-4 pyAI-K210 (正反面)

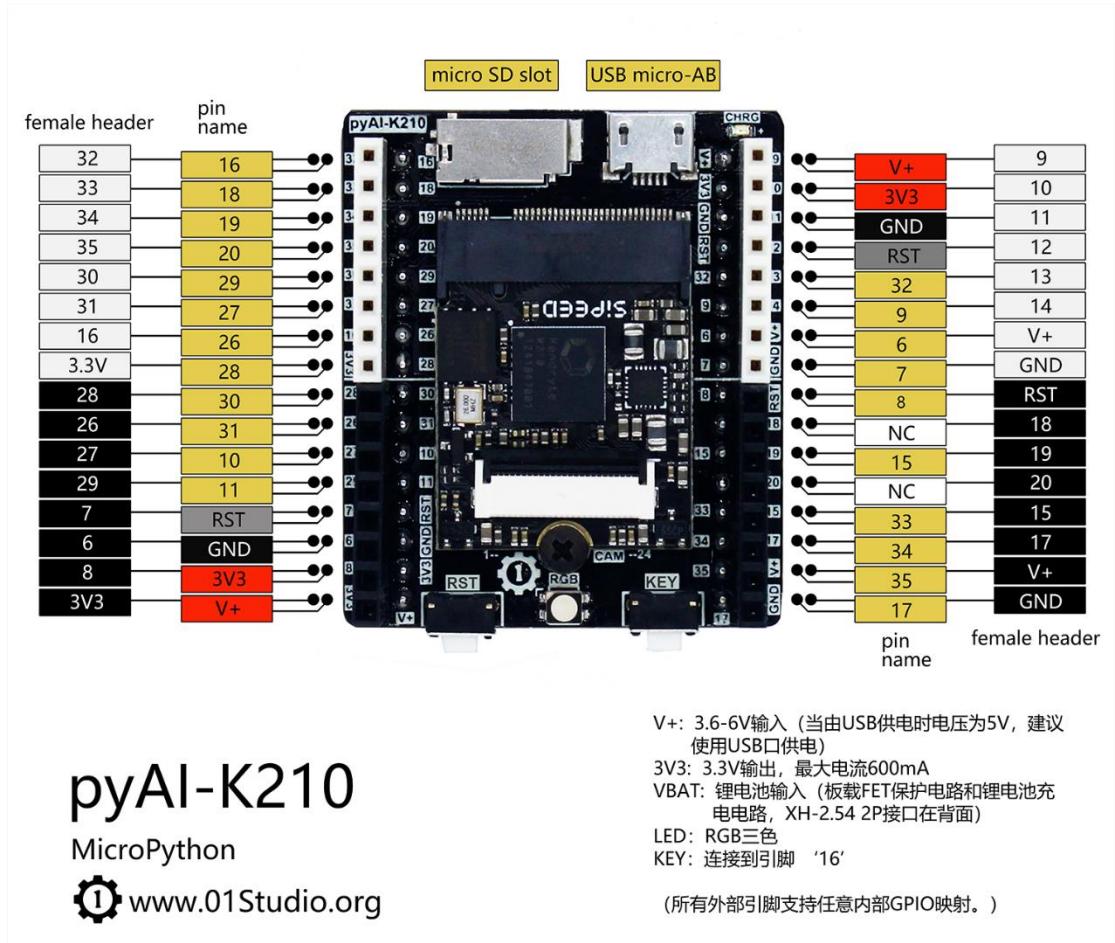


图 1-5 pyAI-K210 引脚图

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 1-1 pyAI-K210 功能参数表

1.4.1.2 pyAI-K210 开发套件

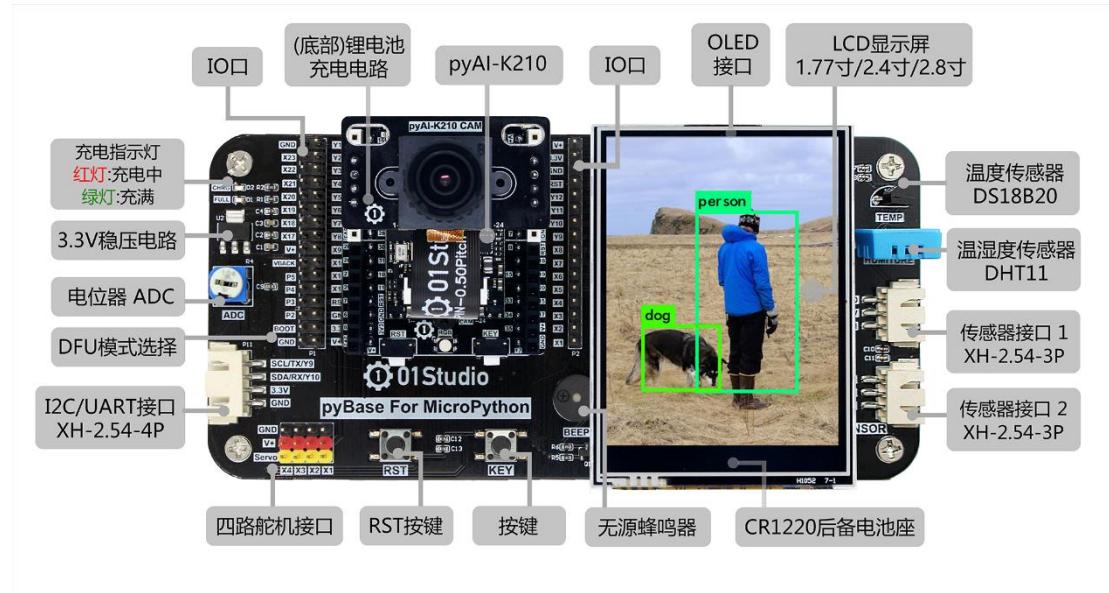


图 1-6 pyAI-K210 开发套件功能描述

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 1-2

1.4.2 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：

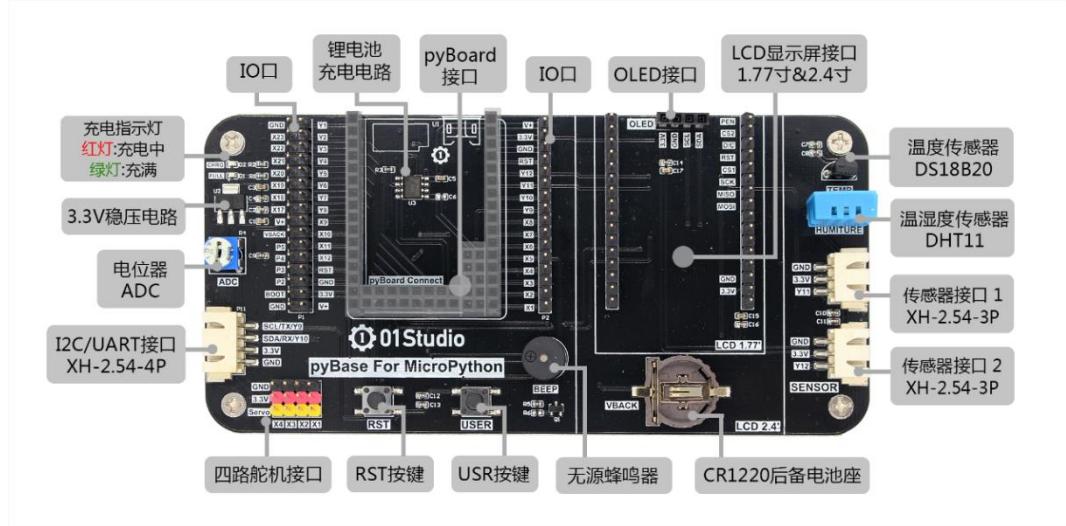


图 1-7 pyBase 功能说明

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 1-3

1.4.3 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。

1.4.3.1 pyIOT-BLE TLS01

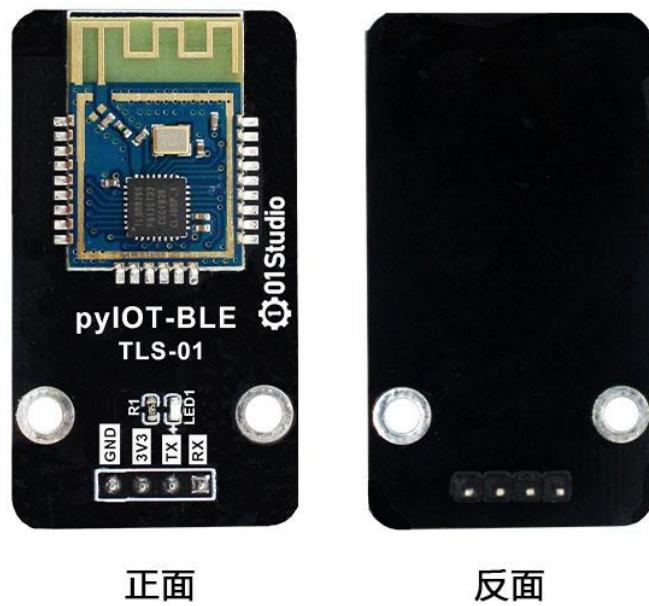


图 1-8

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 1-4

1.4.3.2 pyIOT-LORA E22

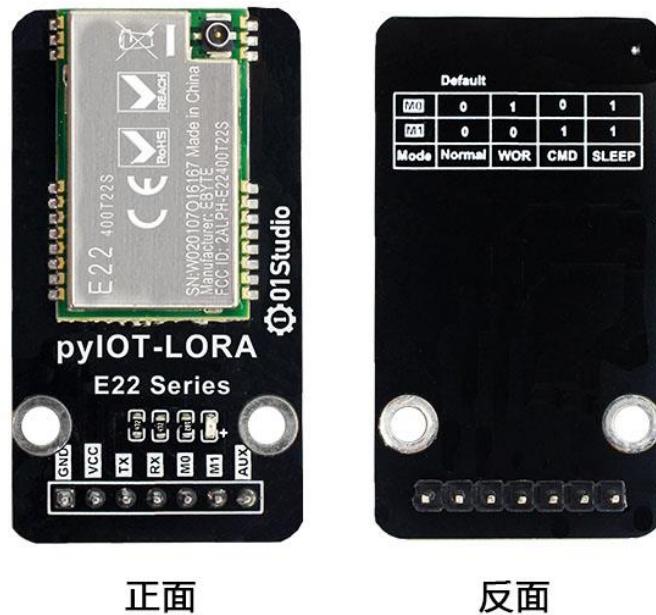


图 1-9

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 °C - 85 °C
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 1-5

1.4.3.3 pyIOT-GPS

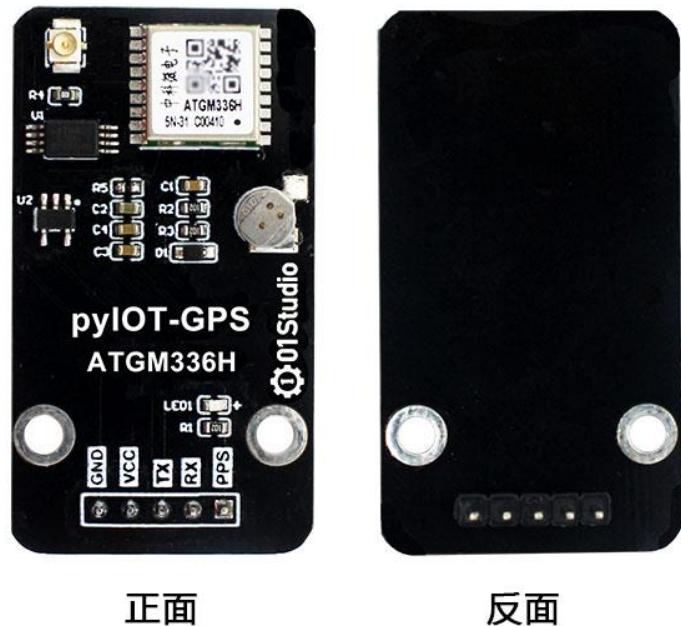


图 1-10

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 1-6

1.4.4 拓展配件

1.4.4.1 2.8 寸 LCD 显示屏（电阻触摸）

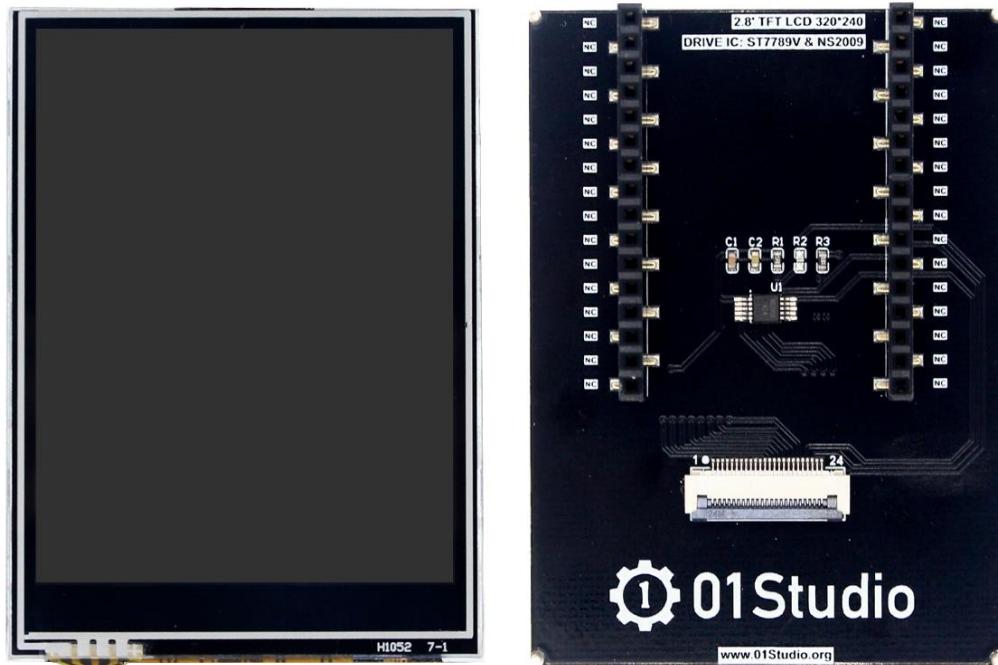


图 1-11 2.4 寸 LCD 显示屏（电阻触摸）

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 1-7

1.4.4.2 OLED 显示屏



图 1-12 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 1-8

1.4.4.3 舵机



图 1-13 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°C-60 °C
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 1-9

1.4.4.4 RGB 灯带

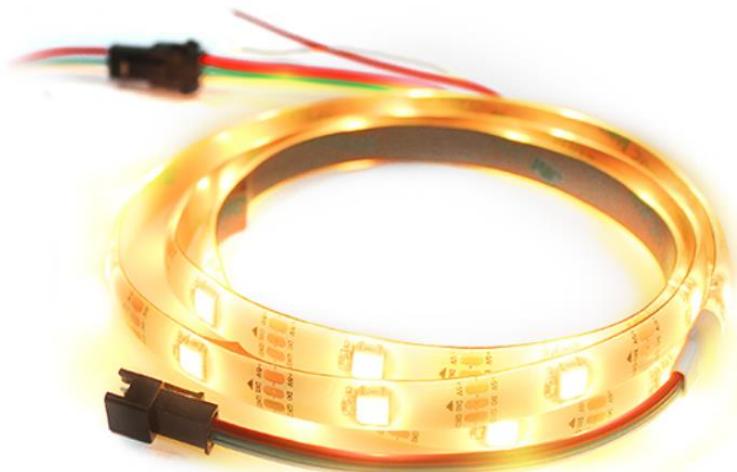


图 1-14 RGB 灯带

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 1-10

1.4.4.5 USB 转串口 TTL

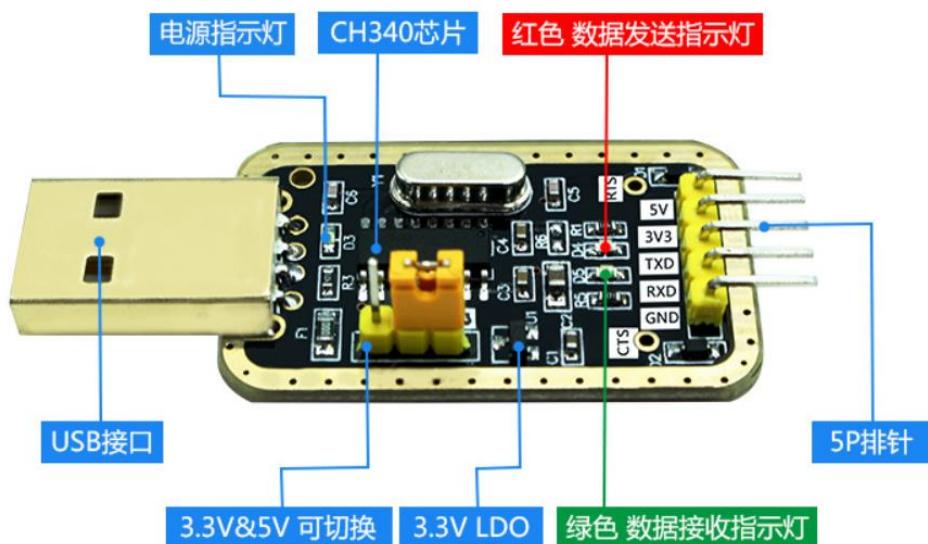


图 1-15 USB 转串口 TTL

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 1-11

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 python 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 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

# 可以把元组合成列表解包，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

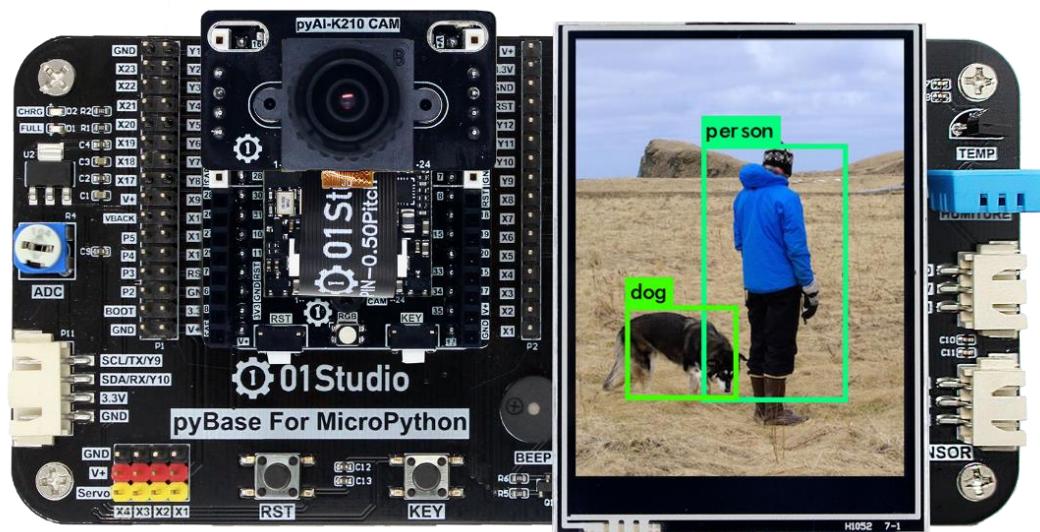
print(say()) # Can you buy me a beer?
print(say(say_please=True)) # Can you buy me a beer? Please! I am
poor :(
```

第3章 开发环境快速搭建

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 (RISC-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

3.1 基于 Windows

3.1.1 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）**

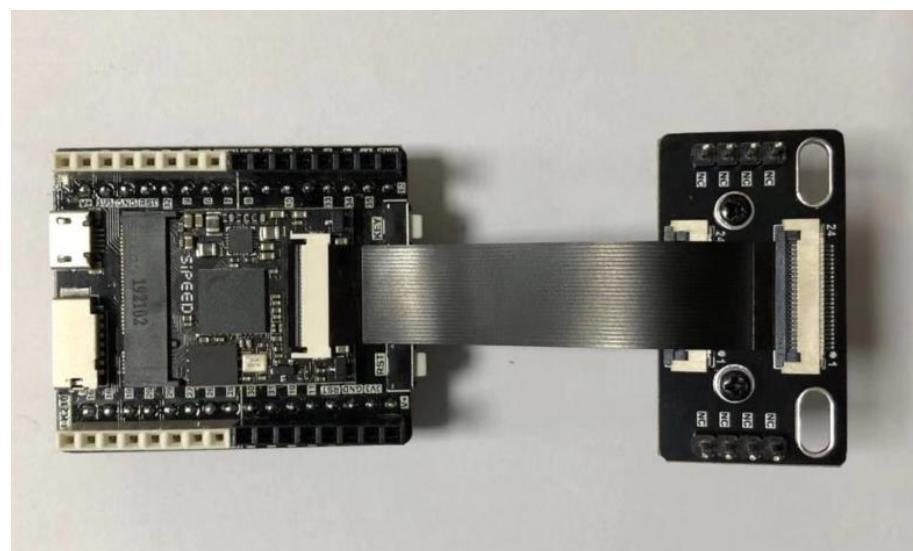


图 3-1 摄像头接线方式

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。

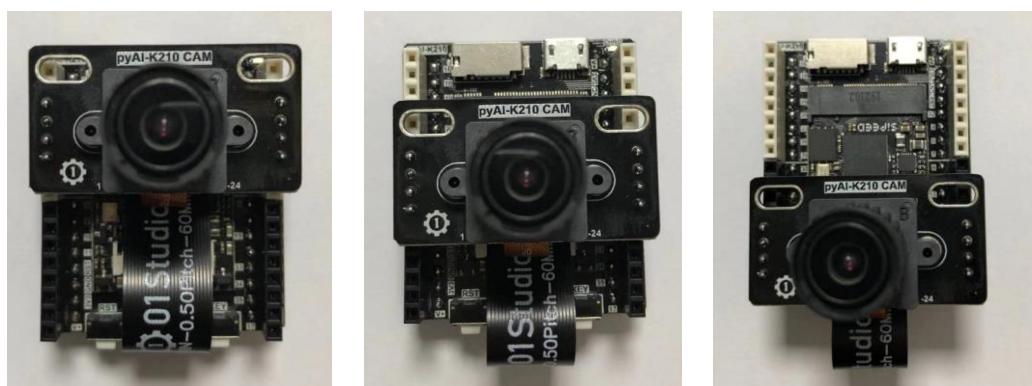


图 3-2 摄像头位置任意摆放

3.1.2 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。

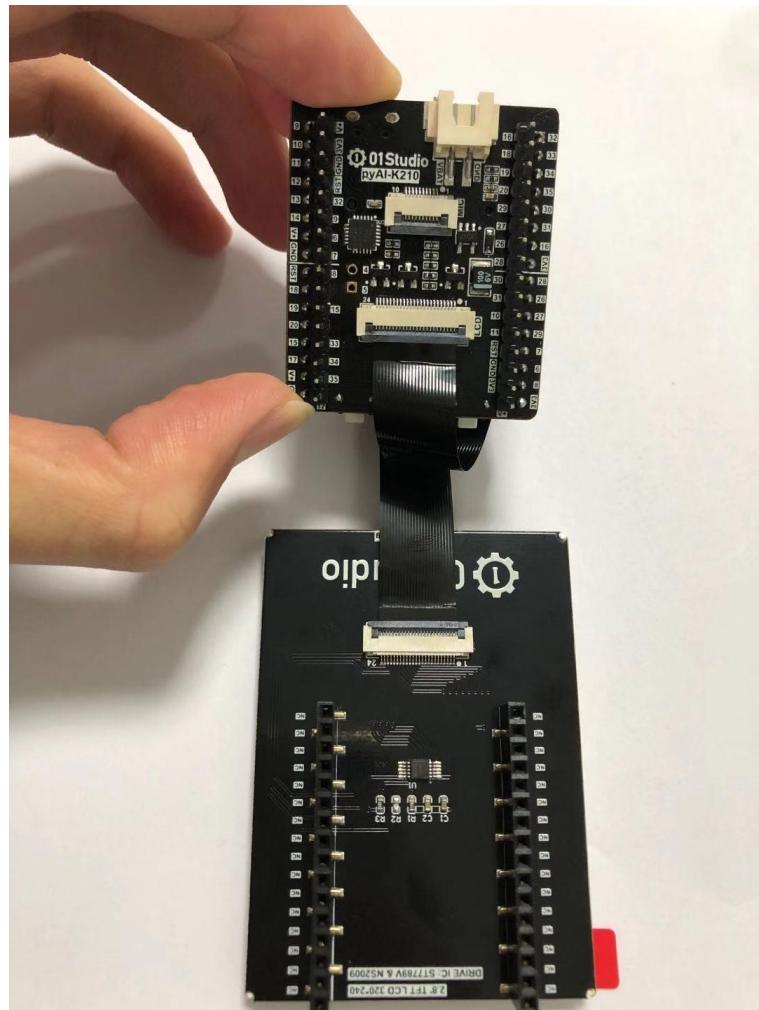


图 3-3 pyAI-K210 与 2.8 寸 LCD 接线方式

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：

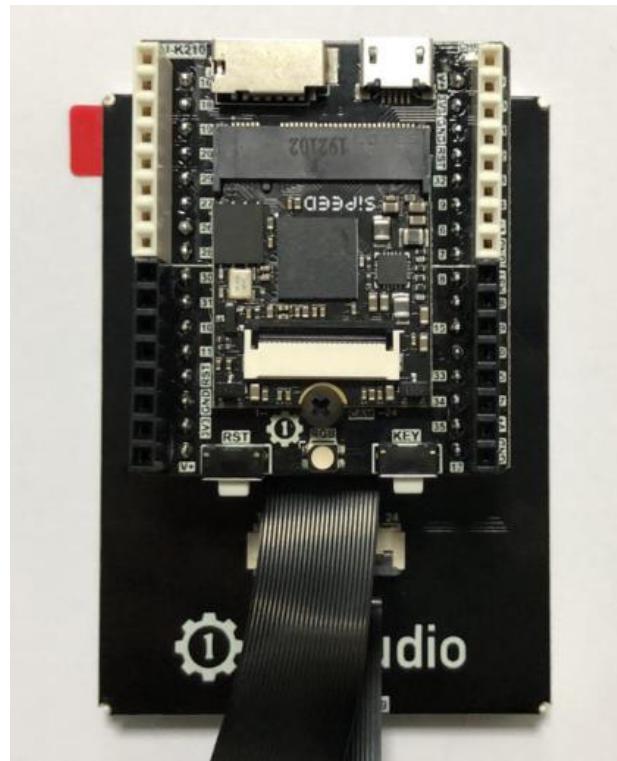


图 3-4 LCD 与 pyAI-K210

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 3-5 LCD 与 pyBase

3.1.3 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 v0.2.5，官网下载地址：（如有更新请下载最新版）

http://cn.dl.sipeed.com/MAIX/MaixPy/ide/_/v0.2.5，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 3-6 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 3-7 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。

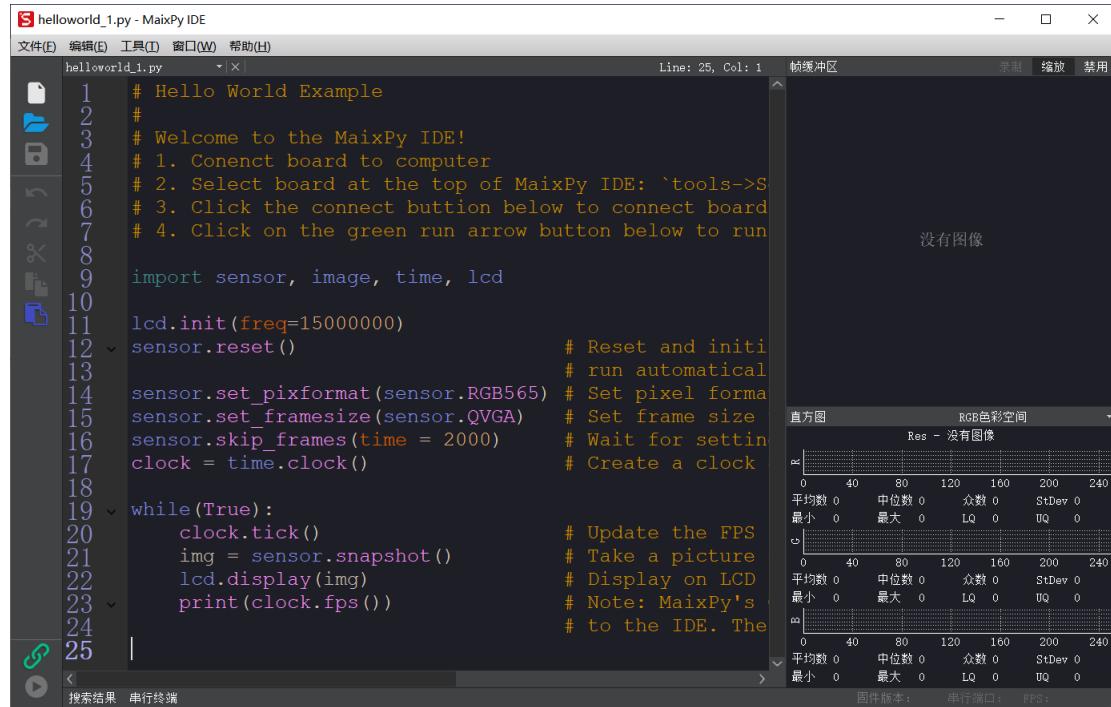


图 3-8 MaixPy IDE

3.1.4 驱动安装

pyAI-K210 通过串口烧写程序和通信，因此主要是安装 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：

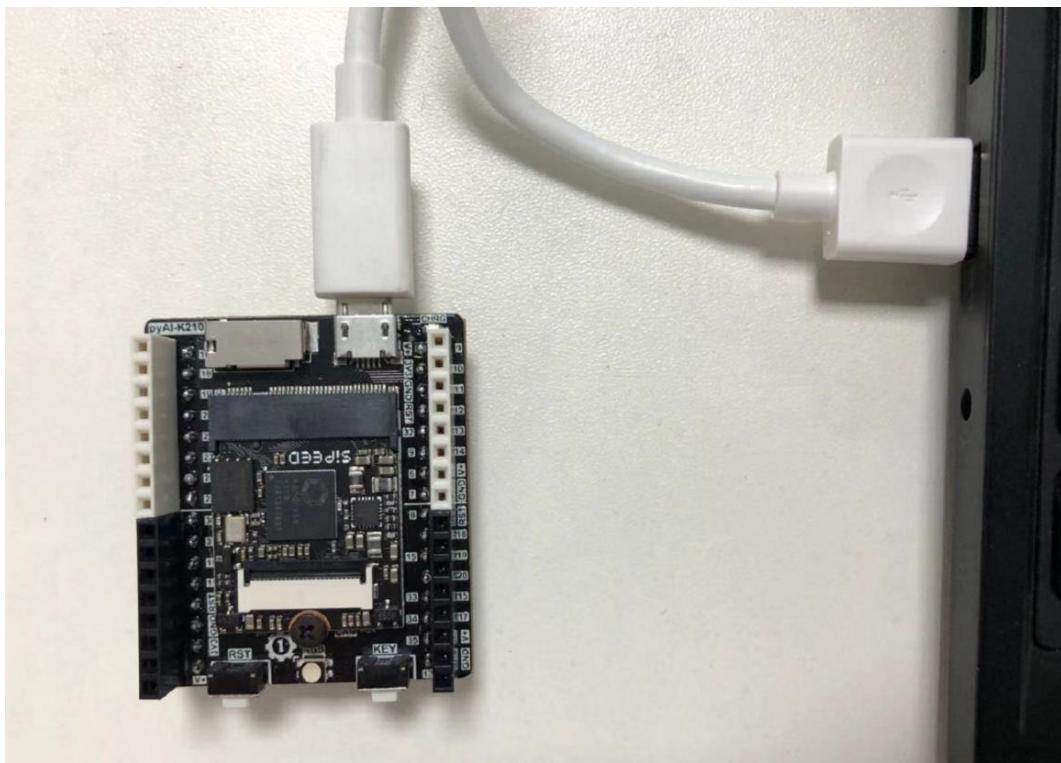


图 3-9 通过 MicroUSB 线连接到电脑

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。

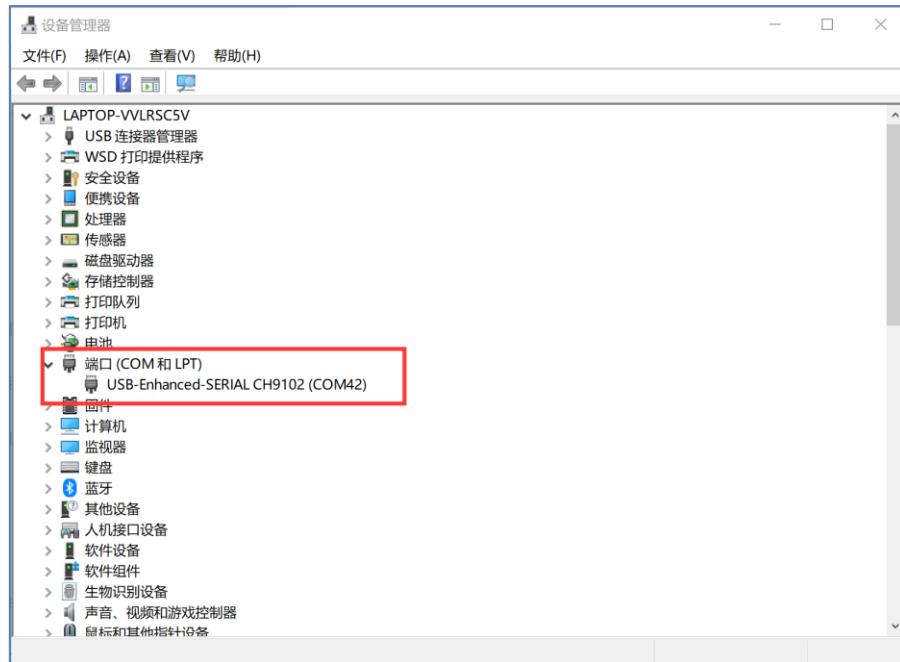


图 3-10 串口驱动成功安装

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”：

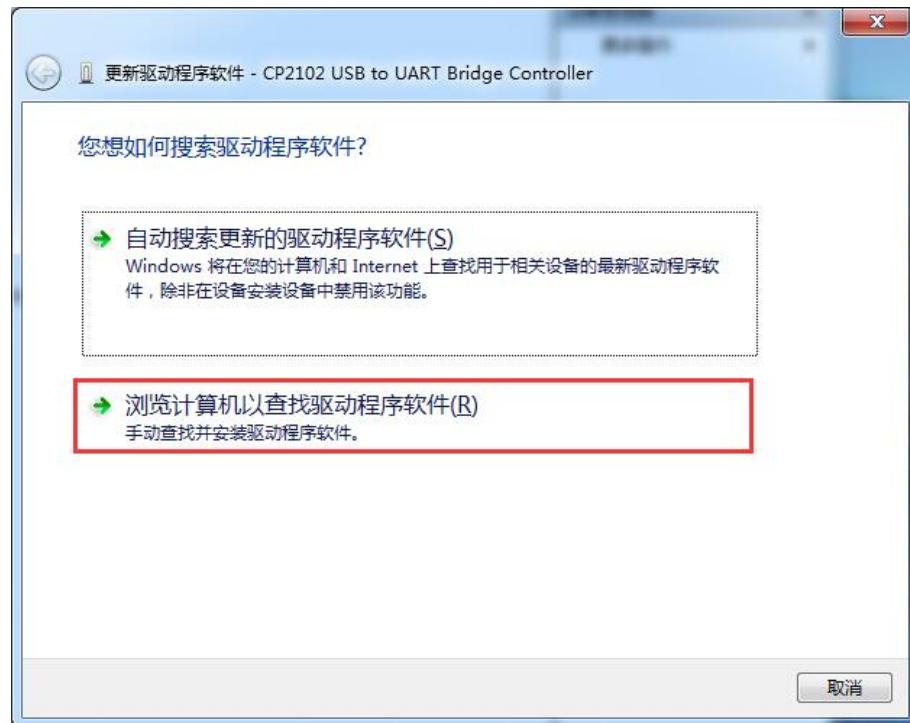


图 3-11

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\CH9102x 驱动，点击确认后即可自动安装：

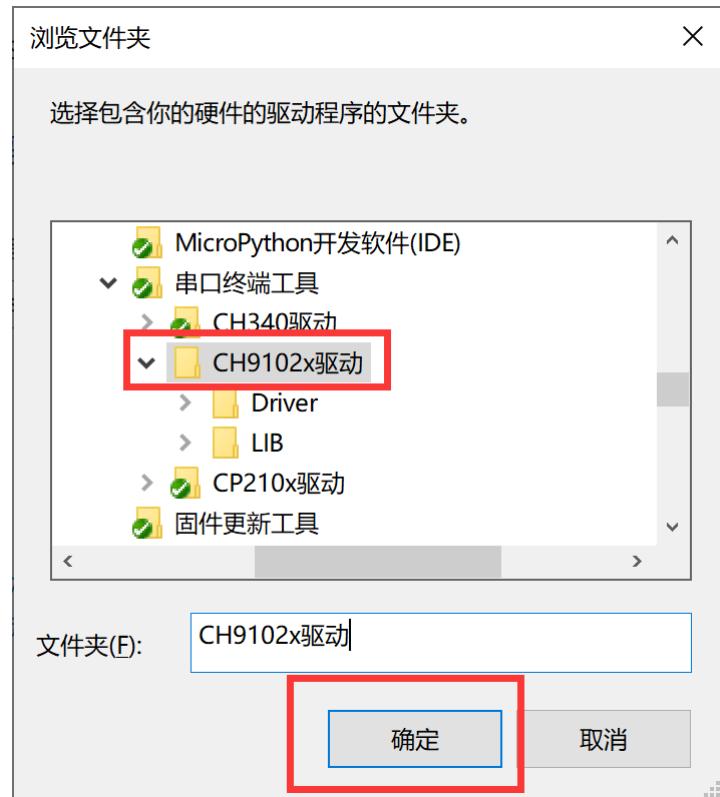


图 3-12

安装成功后如下图：

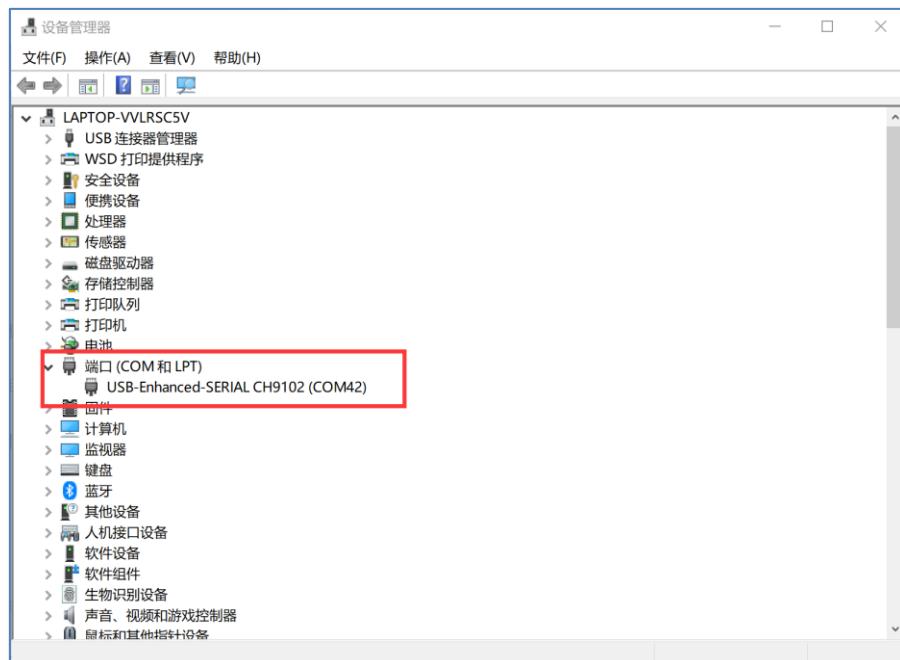


图 3-13 串口驱动安装成功

3.1.5 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：



The screenshot shows the MaixPy IDE interface with the file 'LED.py' open. The code is as follows:

```
1  ...
2  实验名称: 点亮LED_B蓝灯
3  版本: v1.0
4  日期: 2019.12
5  作者: 01Studio
6  实验目的: 学习led点亮。
7  ...
8  from Maix import GPIO
9  from fpioa_manager import fm
10 #将蓝灯引脚IO12配置到GPIO00, K210引脚支持任意配置
11 fm.register(12, fm.fpioa.GPIO00)
12
13 led_b = GPIO(GPIO.GPIO00, GPIO.OUT) #构建LED对象
14 led_b.value(0) #点亮LED
```

图 3-14

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。

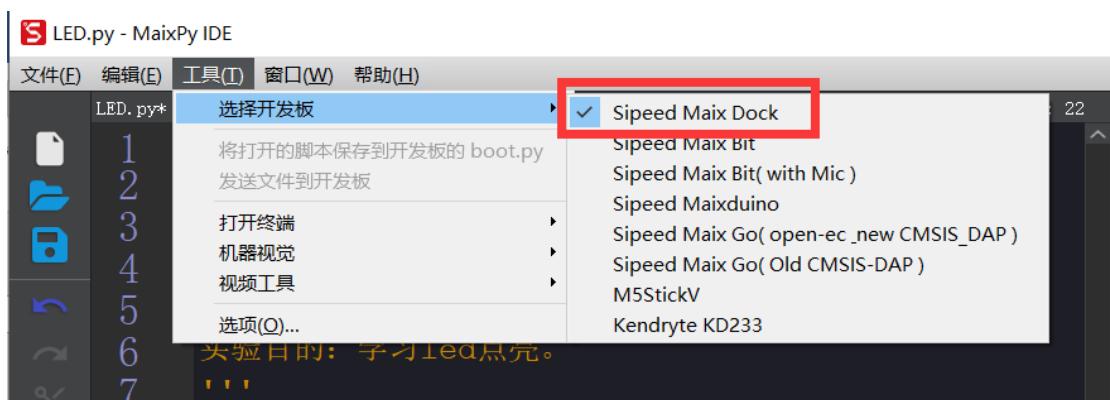


图 3-15

接下来点击左下角连接按钮：

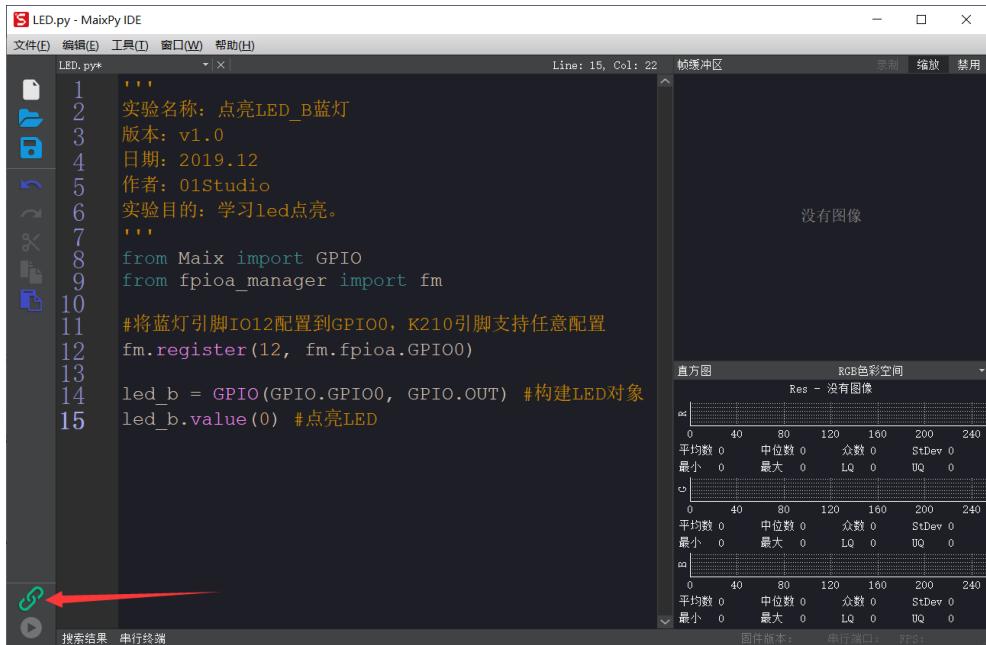


图 3-16 点击连接按钮

连接成功后，运行按钮变成绿色。



图 3-17 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：

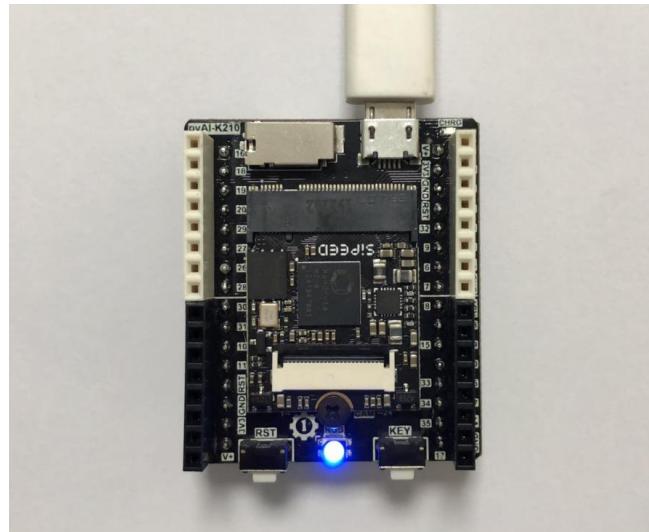


图 3-18 第一个实验

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。

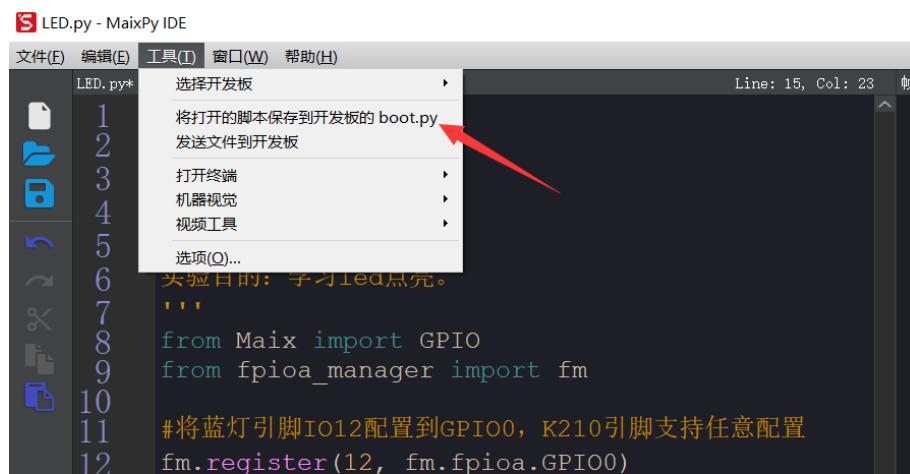


图 3-19 复制文件到 pyAI-K210 文件系统

点击后出现以下提示，点 yes 确认。

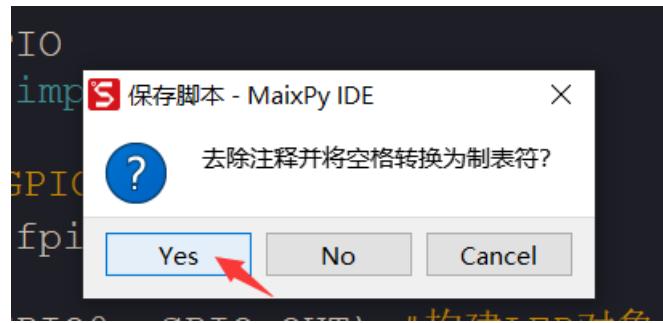


图 3-20 点击 YES

当出现保存脚本成功，说明程序烧写成功。



图 3-21

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。

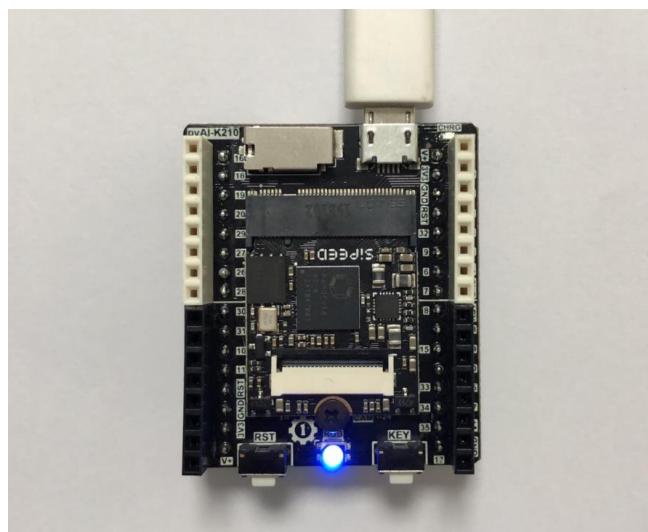


图 3-22 复位后程序运行

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。

The screenshot shows the MaixPy IDE interface with the following details:

- Title Bar:** LED.py - MaixPy IDE
- Menu Bar:** 文件(F) 编辑(E) 工具(I) 窗口(W) 帮助(H)
- Code Editor:** LED.py*
1 '''
2 实验名称: 点亮LED_B蓝灯
3 版本: v1.0
4 日期: 2019.12
5 作者: 01Studio
6 实验目的: 学习led点亮。
7 '''
8 from Maix import GPIO
9 from fpioa_manager import fm
10 #将蓝灯引脚IO12配置到GPIO00, K210引脚支持任意配置
11 fm.register(12, fm.fpioa.GPIO00)
12
13 串行终端 | 串行端口
14 >>>
15 >>> 1
16 MicroPython v0.5.0 on 2019-11-29; Sipeed_M1 with kendryte-k210
17 Type "help()" for more information.
18 >>>
19 >>> MicroPython v0.5.0 on 2019-11-29; Sipeed_M1 with kendryte-k210
20 Type "help()" for more information.
21 >>>
- Right Panel:** 直方图 (Histogram), RGB色彩空间 (RGB Color Space), Res - 没有图像 (Res - No Image). The histogram shows three empty 240x240 grids.
- Bottom Status Bar:** 搜索结果 (Search Results), 串行终端 (Serial Terminal) (highlighted with a red arrow), 串行端口: COM13, FPS: 0, 固件版本: 0.5.0

图 3-23 串口终端信息显示

3.1.6 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 3-24 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：

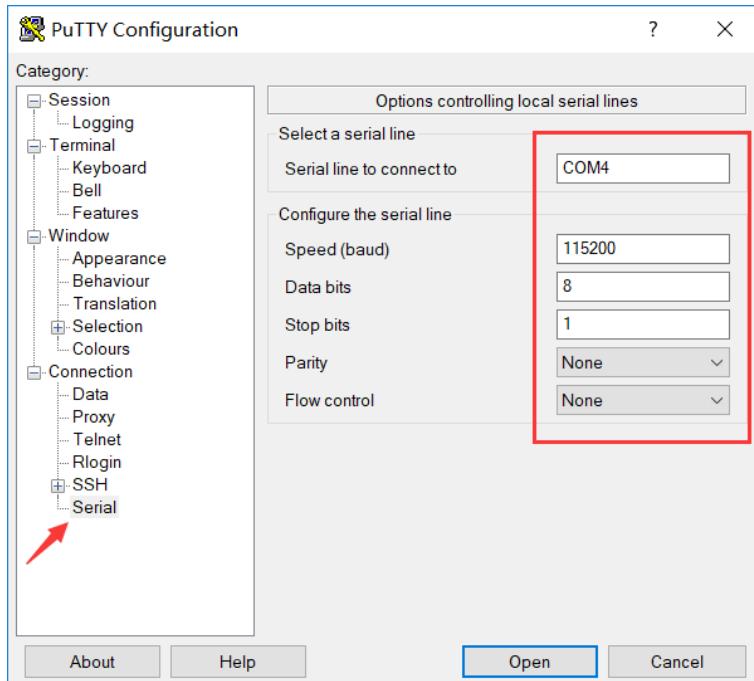


图 3-25 配置串行设备参数

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。

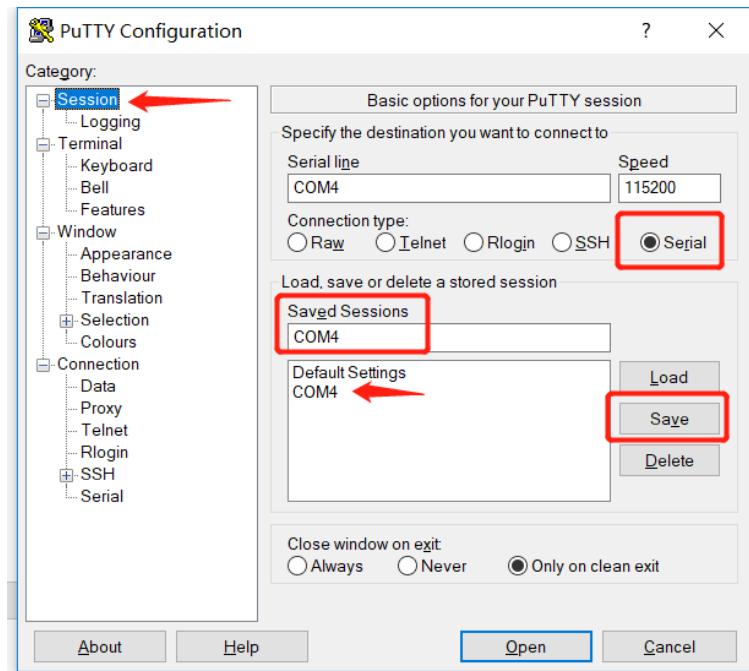
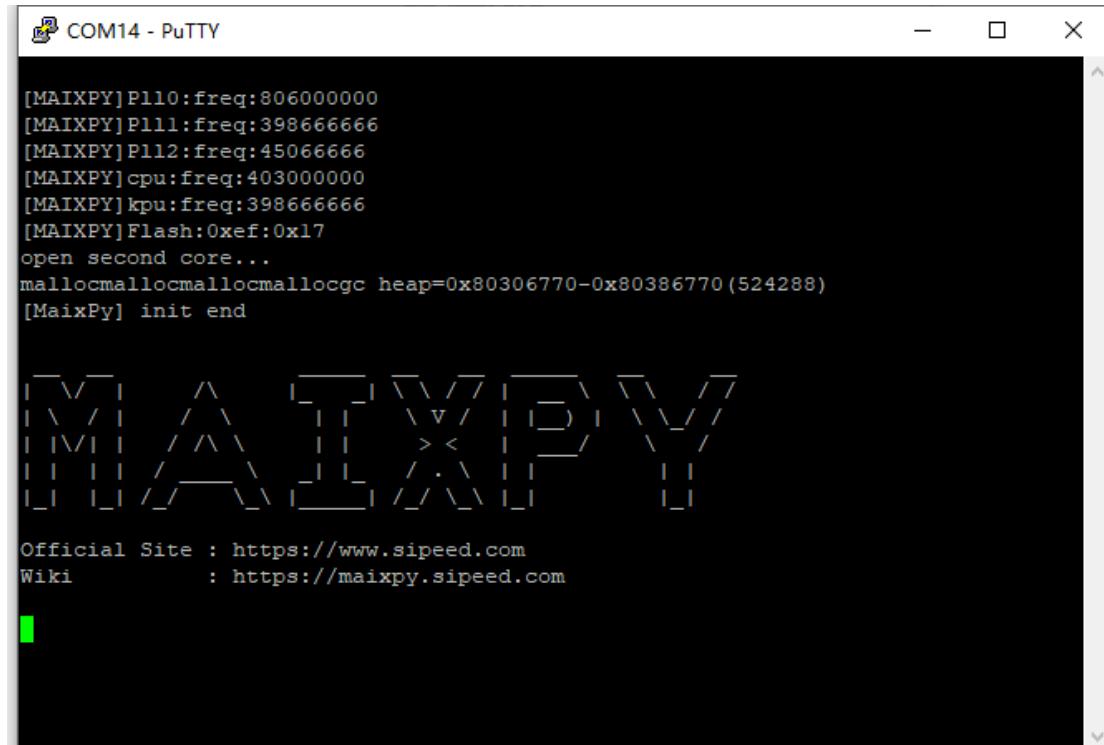


图 3-26

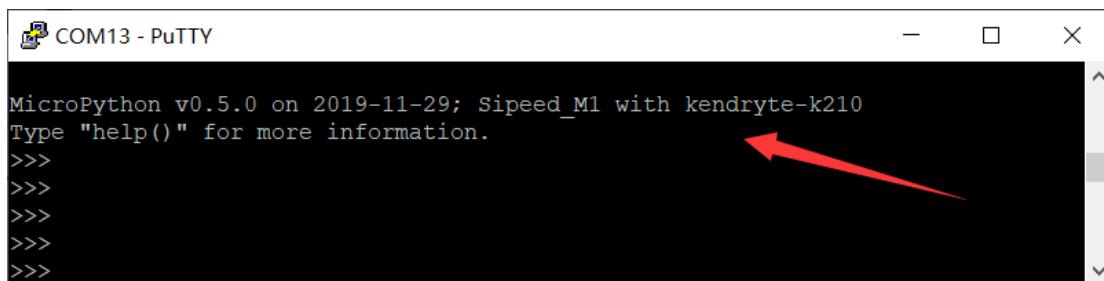
连接如果出现下面情况，说明开发板里面有代码在运行，阻塞了 REPL，这时候只需要按下键盘的 **Ctrl + C** 键即可打断在运行的程序。



```
[MAIXPY] P110:freq:806000000  
[MAIXPY] P111:freq:398666666  
[MAIXPY] P112:freq:450666666  
[MAIXPY] cpu:freq:403000000  
[MAIXPY] kpu:freq:398666666  
[MAIXPY] Flash:0xef:0x17  
open second core...  
mallocmallocmallocmallocgc heap=0x80306770-0x80386770 (524288)  
[MaixPy] init end  
  
[   \V ] [   ^ ] [   \V ] [ \V / ] [ \V ] [ \V / ]  
[ \V / ] [ / \V ] [   \V ] [ > < ] [ \V ] [ \V / ]  
[ \V / ] [ / \V ]  
[ \V / ] [ / \V ]  
  
Official Site : https://www.sipeed.com  
Wiki : https://maixpy.sipeed.com  
  
█
```

图 3-27 开发板有代码阻塞

Ctrl+C 打断后会出现下面可交互的对话框，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。

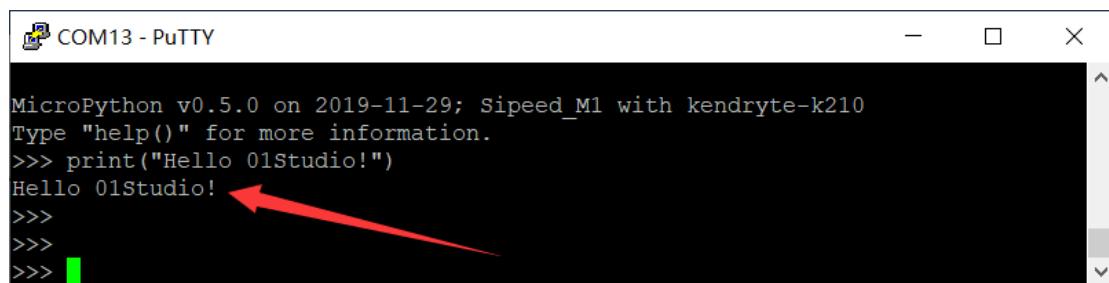


```
MicroPython v0.5.0 on 2019-11-29; Sipeed_M1 with kendryte-k210  
Type "help()" for more information.  
=>  
=>  
=>  
=>  
=>
```

图 3-28

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。

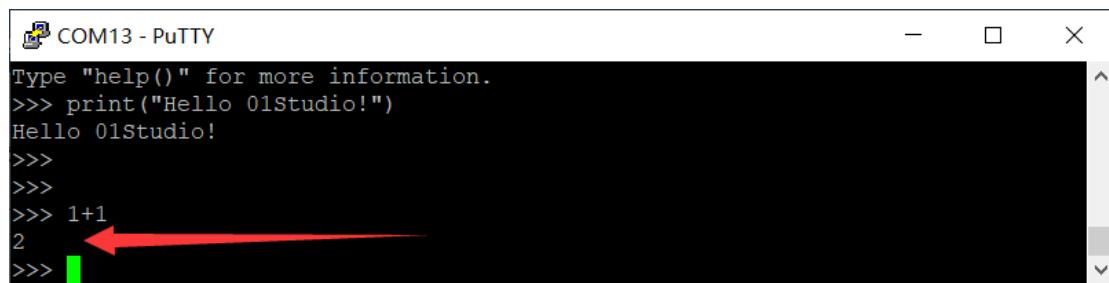
```
print("Hello 01Studio!")
```



```
MicroPython v0.5.0 on 2019-11-29; Sipeed_M1 with kendryte-k210
Type "help()" for more information.
>>> print("Hello 01Studio!")
Hello 01Studio!
```

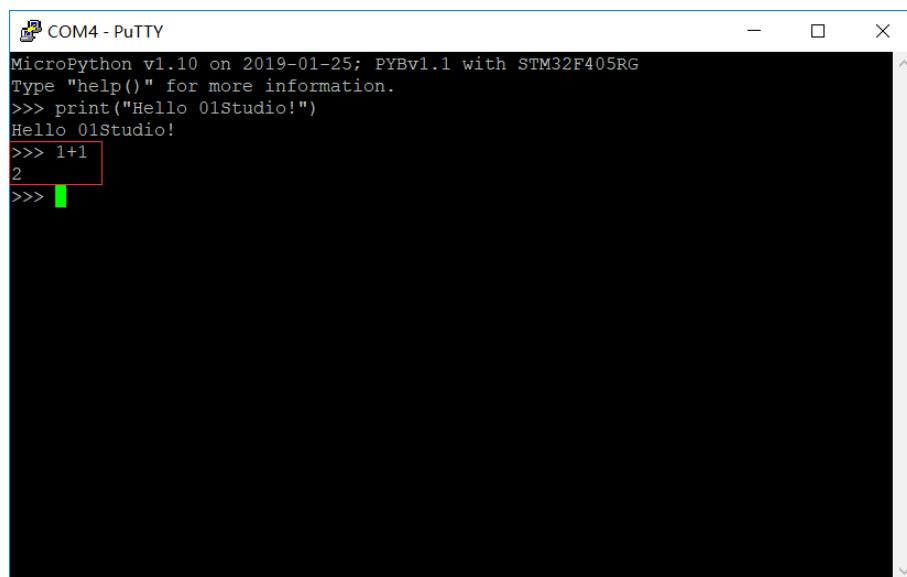
图 3-29

我们再输入 $1+1$ 按回车，得到了计算结果 2。



```
Type "help()" for more information.
>>> print("Hello 01Studio!")
Hello 01Studio!
>>>
>>>
>>> 1+1
2
>>>
```

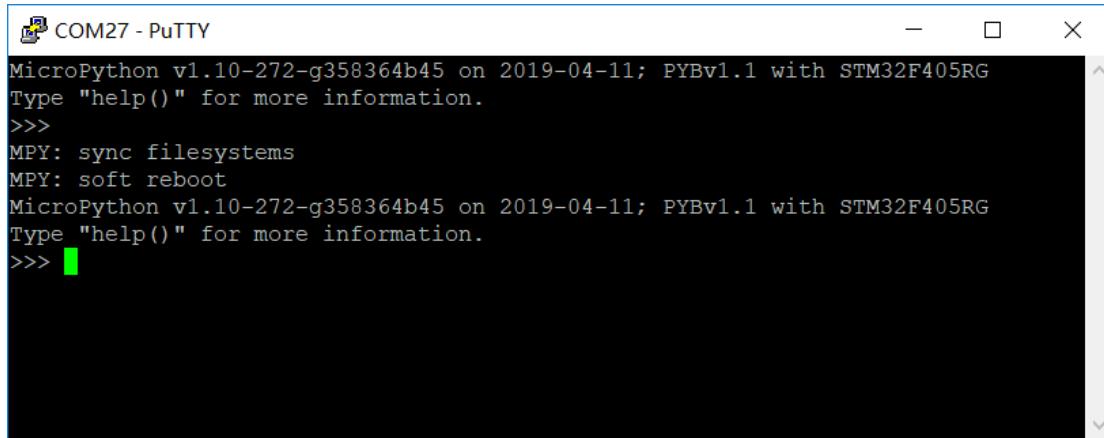
图 3-30



```
MicroPython v1.10 on 2019-01-25; PYBv1.1 with STM32F405RG
Type "help()" for more information.
>>> print("Hello 01Studio!")
Hello 01Studio!
>>> 1+1
2
>>>
```

图 3-31

溫馨提示: 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：



```
COM27 - PuTTY
MicroPython v1.10-272-g358364b45 on 2019-04-11; PYBv1.1 with STM32F405RG
Type "help()" for more information.
>>>
MPY: sync filesystems
MPY: soft reboot
MicroPython v1.10-272-g358364b45 on 2019-04-11; PYBv1.1 with STM32F405RG
Type "help()" for more information.
>>> █
```

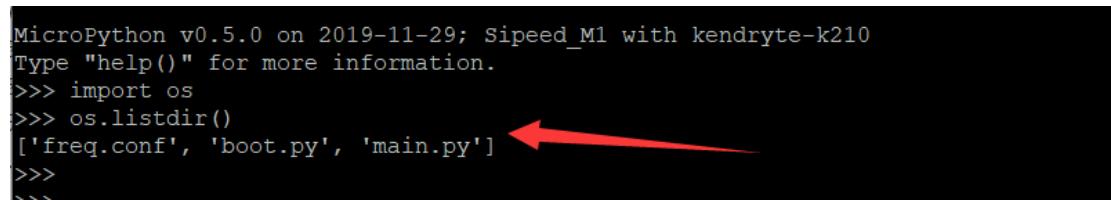
图 3-32 软件复位 (soft reboot)

3.1.7 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()
```



```
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', 'main.py'] ← Red arrow here
>>>
>>>
```

图 3-33 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：

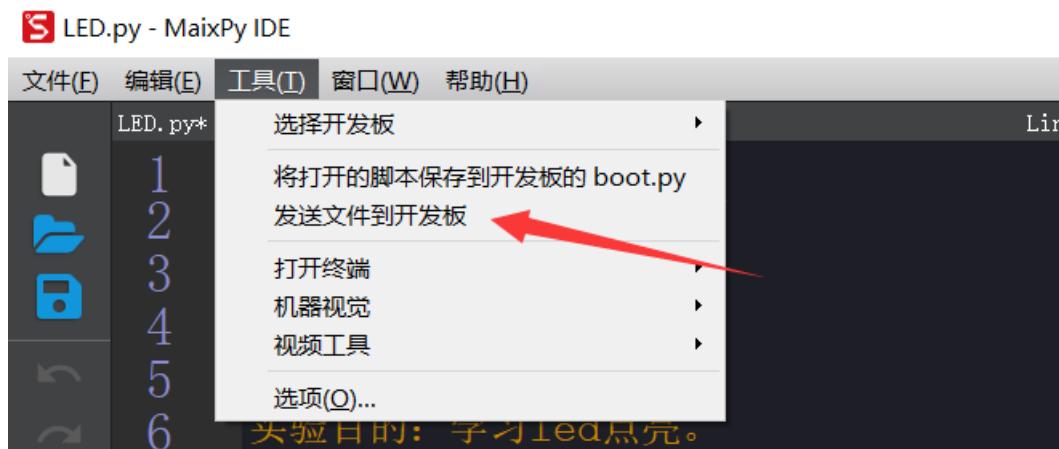


图 3-34

我们可以选择基础实验中的 LED.py 文件，保存到开发板。

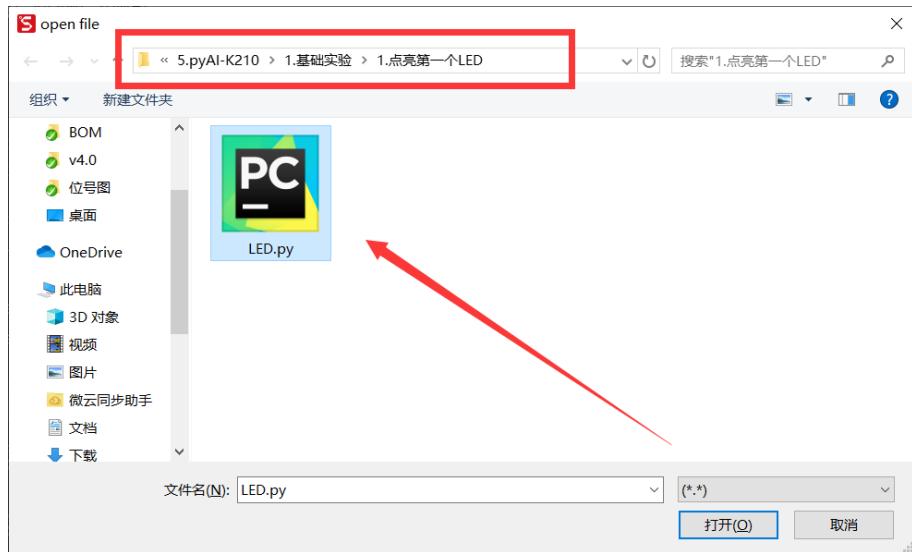


图 3-35

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 3-36 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！)

3.1.8 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。

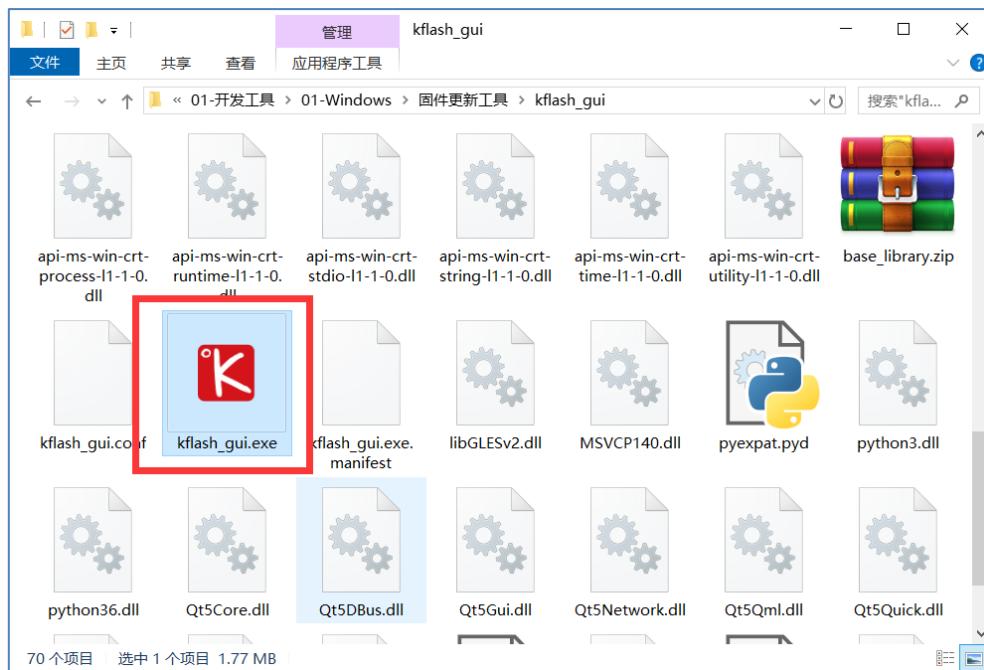


图 3-37 K210 固件烧录软件

打开软件后点击 open file:

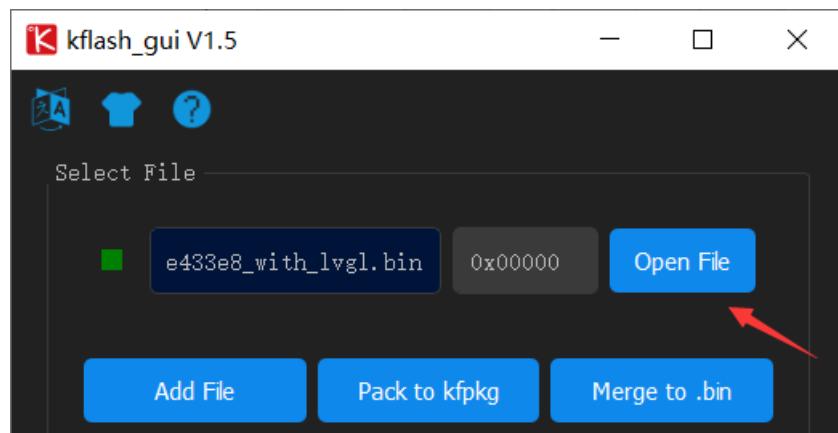


图 3-38

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：

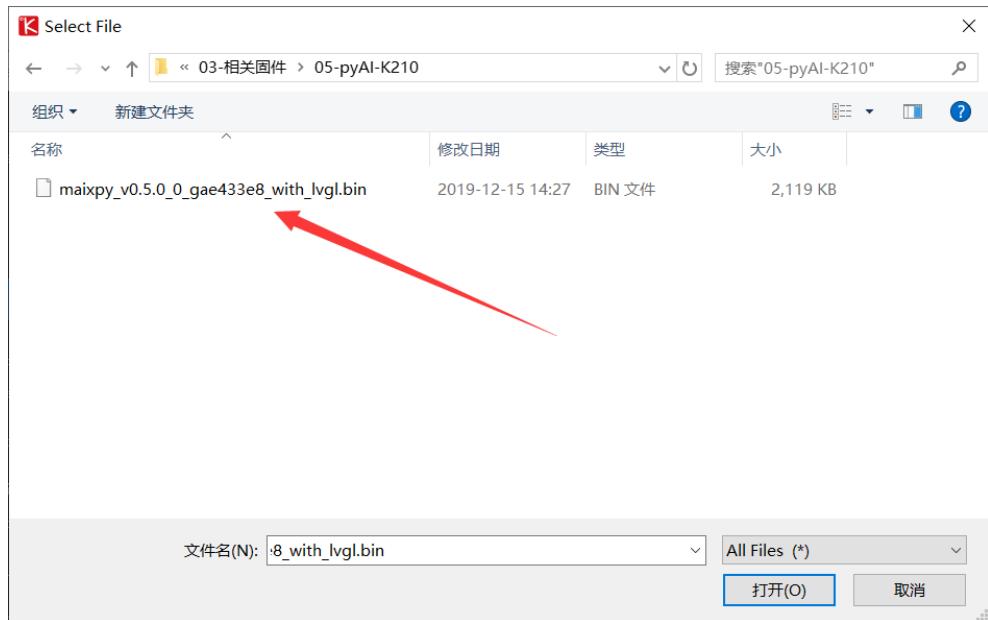


图 3-39 选择要更新的固件

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。

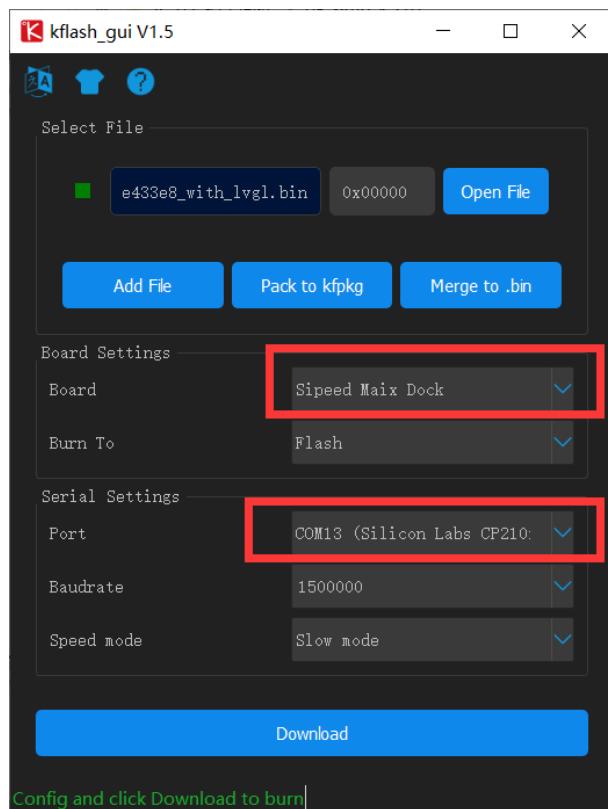


图 3-40 选择开发板和串口 COM

点击 Download 下载。(如出现一直等待情况说明无法自动下载，这时候按一下开发板的 RST 复位键即可。)

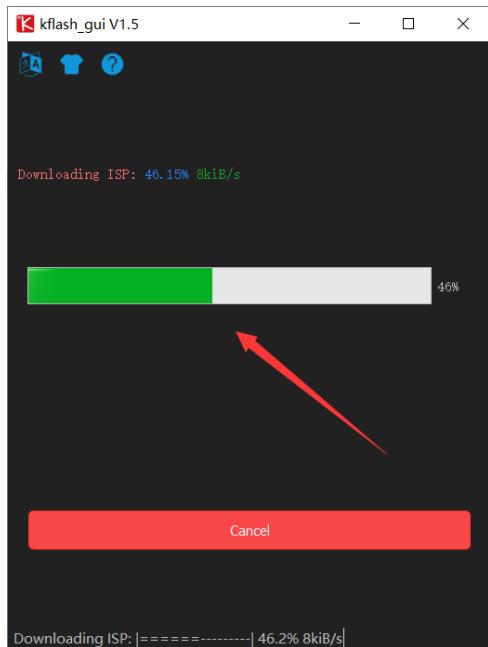


图 3-41 正在下载

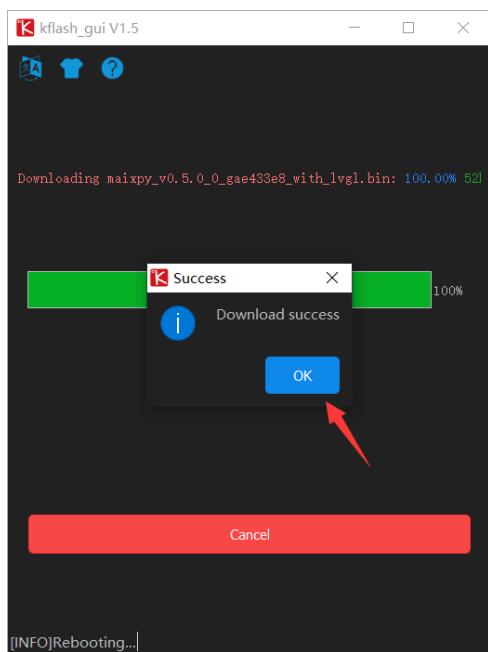


图 3-42 下载成功后弹出 success 对话框

MaixPy 针对不同的应用场景提供不同大小的固件，而且不断更新，详见固件下载链接（不同固件区别见里面的 `readme.txt`）：

<https://dl.sipeed.com/shareURL/MAIX/MaixPy/release/master>

3.2 基于 Mac OS

3.2.1 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。

The screenshot shows a download interface for the MaixPy IDE. At the top, there's a breadcrumb navigation: Home > MAIX > MaixPy > ide > v0.2.4. To the right of the navigation is a 'Hi' button. Below the navigation is a table listing files:

#	Name	Size
1	maixpy-ide-mac-0.2.4.dmg	110.6 MB
2	maixpy-ide-linux-x86_64-0.2.4-installer-archive.7z	99.2 MB
3	maixpy-ide-windows-0.2.4-installer-archive.7z	68.0 MB
4	sha256sum.txt	515 Bytes
5	maixpy-ide-linux-x86_64-0.2.4.run	126.4 MB
6	maixpy-ide-windows-0.2.4.exe	87.8 MB

At the bottom of the table, it says "0 folders and 6 files, 491.9 MB in total". The first file, "maixpy-ide-mac-0.2.4.dmg", is highlighted with a red box.

图 3-43 MaixPy IDE 下载界面

3.3 基于 Linux

3.3.1 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。

The screenshot shows a file download interface for the MaixPy IDE. At the top, there's a breadcrumb navigation: Home > MAIX > MaixPy > ide > _ > v0.2.4. Below the navigation, there's a 'Hide' button. The main area is a table with columns for number (#), name, and size. The table lists six files:

#	Name	Size
1	maixpy-ide-mac-0.2.4.dmg	110.6 MB
2	maixpy-ide-linux-x86_64-0.2.4-installer-archive.7z	99.2 MB
3	maixpy-ide-windows-0.2.4-installer-archive.7z	68.0 MB
4	sha256sum.txt	515 Bytes
5	maixpy-ide-linux-x86_64-0.2.4.run	126.4 MB
6	maixpy-ide-windows-0.2.4.exe	87.8 MB

At the bottom of the table, it says "0 folders and 6 files, 491.9 MB in total".

图 3-44 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，力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

- **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。

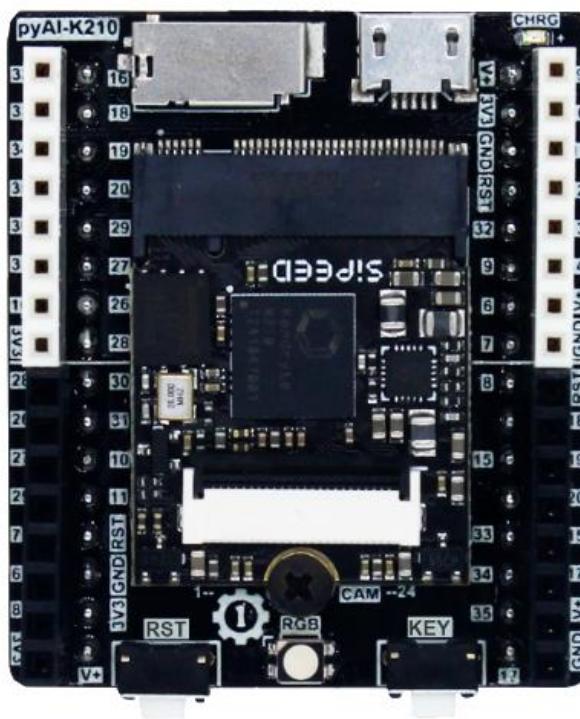


图 4-1 pyAI-K210 核心板

- **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)。

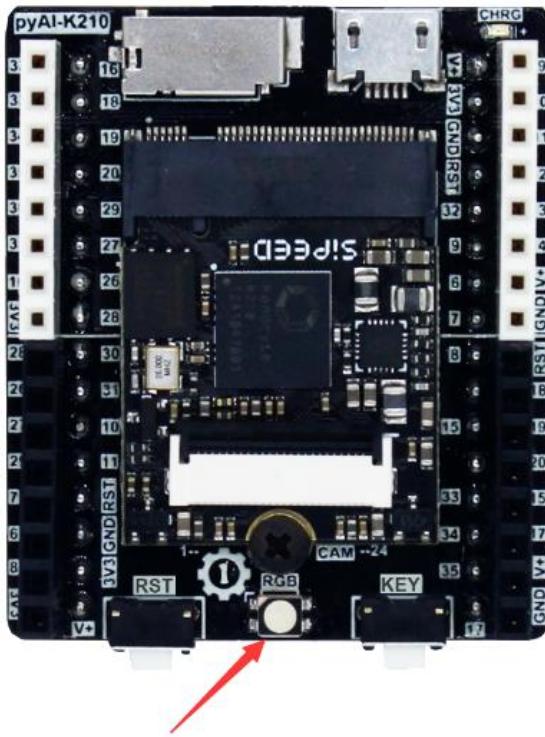


图 4-2 RGB 灯

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。

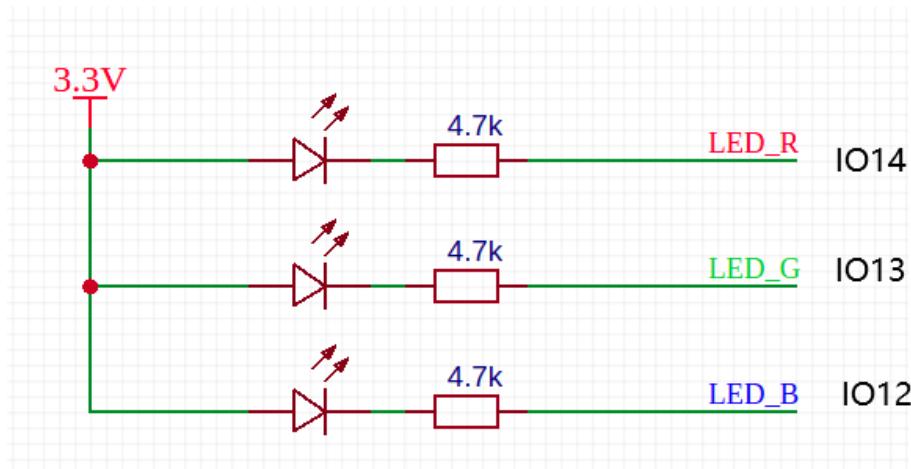


图 4-3 RGB LED 电路引脚图

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
<code>fm.register(pin,function,force=False)</code>
GPIO 注册函数；
使用方法
<code>fm.register(pin,function,force=False)</code>
【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： <code>fm.register(12, fm.fpioa.GPIO0,force=True)</code>
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://wiki.sipeed.com/soft/maixpy/zh/api_reference/Maix/gpio.html

表 4-1 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
<code>GPIO(ID,MODE,PULL,VALUE)</code>
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 4-4 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：

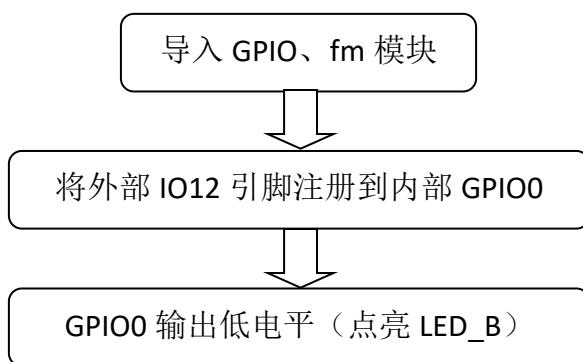


图 4-5 代码编写流程

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。

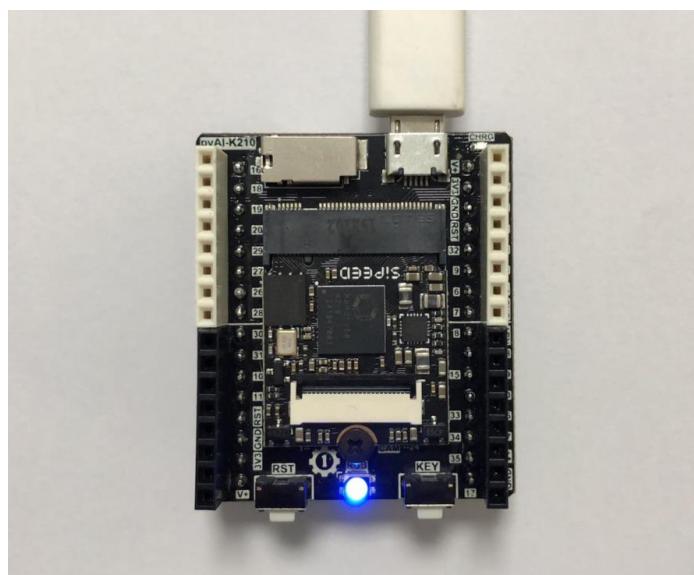


图 4-6 LED 蓝灯被点亮

- **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。

4.2 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。

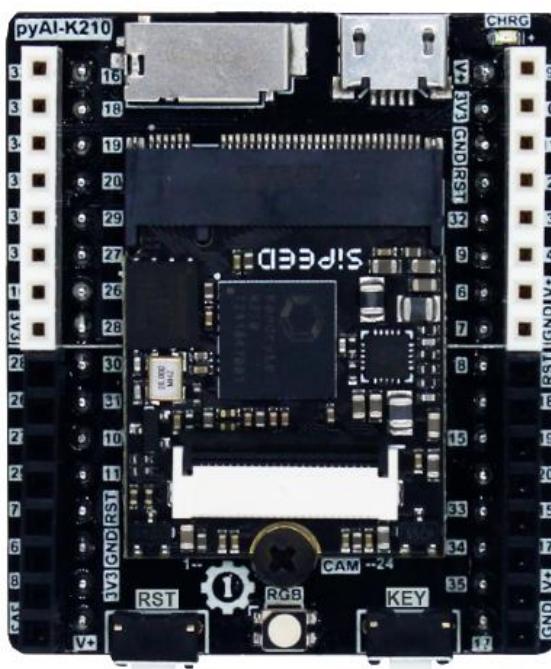


图 4-7

● 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
<code>utime()</code>
时间模块，直接使用。
使用方法
<code>utime.sleep(seconds)</code>
秒级颜色。 <code>seconds</code> : 延时秒数
<code>utime.sleep_ms(ms)</code>
毫秒级延时。 <code>ms</code> : 延时毫秒数。
<code>utime.sleep_us(us)</code>
微秒级延时。 <code>us</code> : 延时微秒数。
*更多用法请阅读 MaixPy 官方文档： https://wiki.sipeed.com/soft/maixpy/zh/api_reference/standard/utime.html

表 4-2 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：

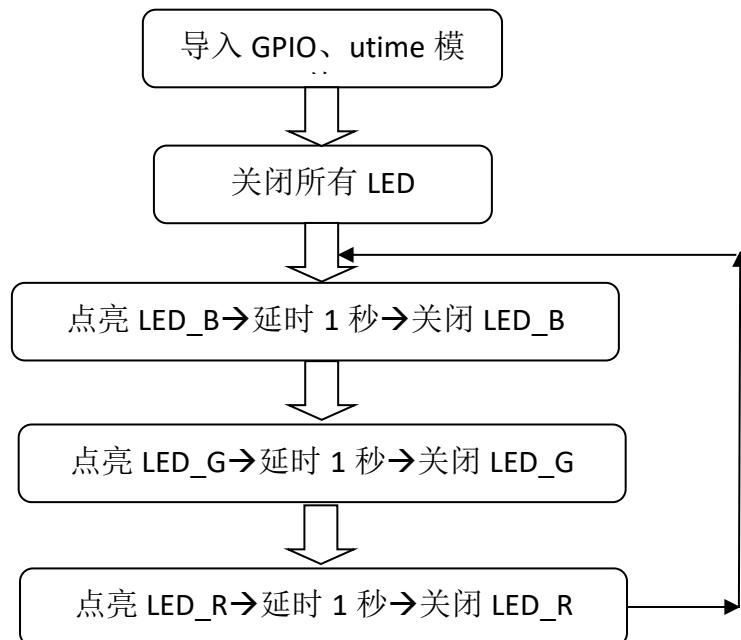


图 4-8 代码编写流程

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。

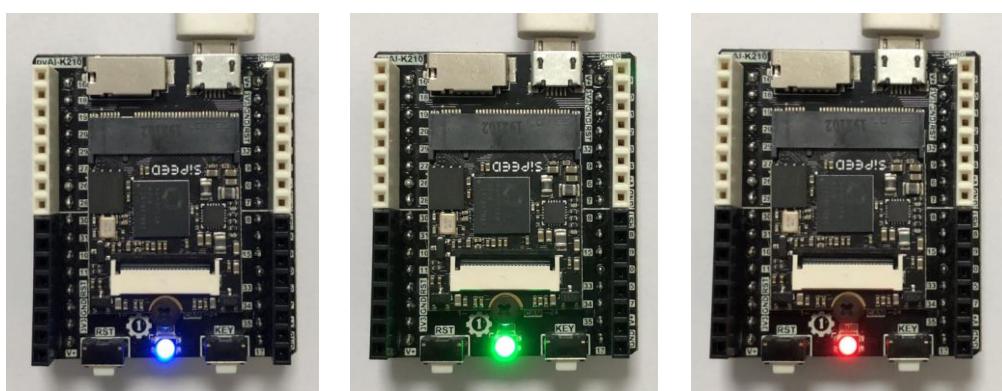


图 4-9 RGB 灯循环闪烁

- **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。

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：**

pyAI-K210。

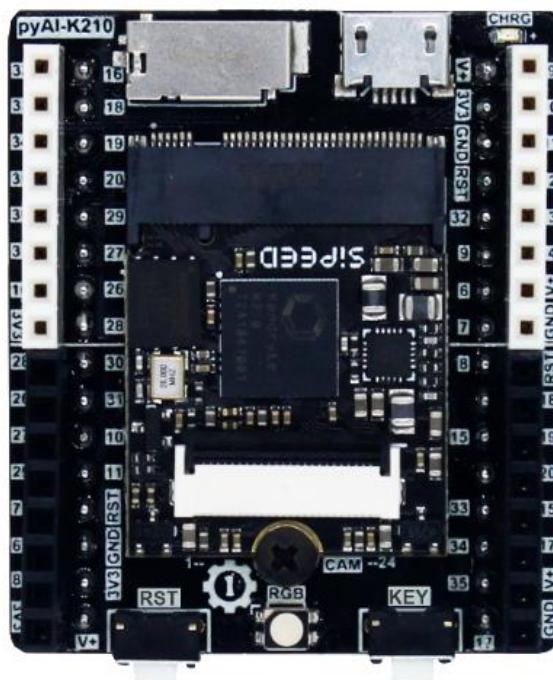


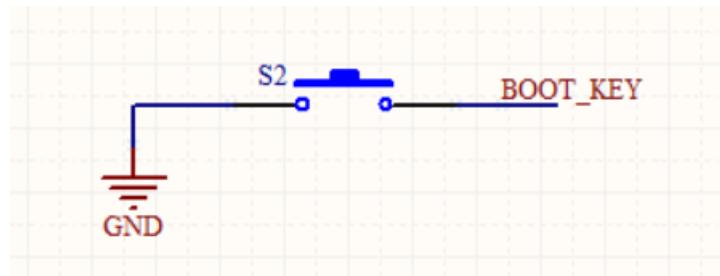
图 4-10

- **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。



IO21	49	IO18	MIC_BCK
IO18	51	IO19	MIC_WS
IO19	53	IO16	BOOT_KEY
BOOT	55	IO17	
IO17	57	IO14	LED_R
IO14	59	IO15	
IO15	61	IO12	LED_B
IO12	63	IO13	LED_G
IO13	65	IO10	

图 4-11 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
<code>GPIO.value([value])</code>
【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 4-3 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：

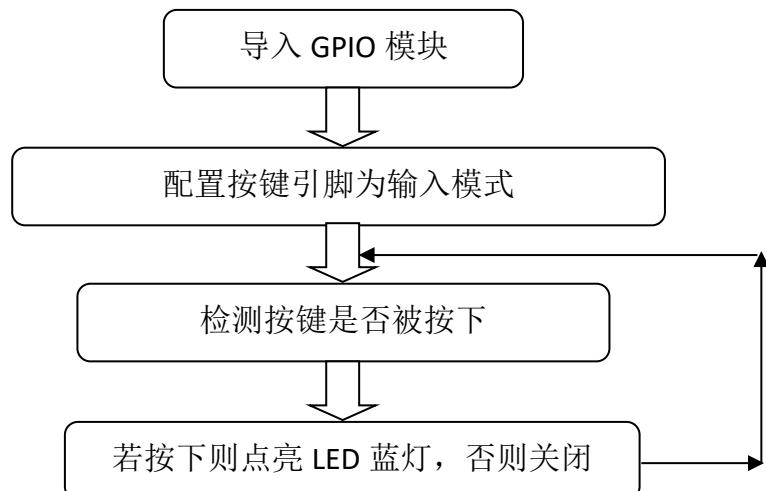


图 4-12 代码编写流程

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。

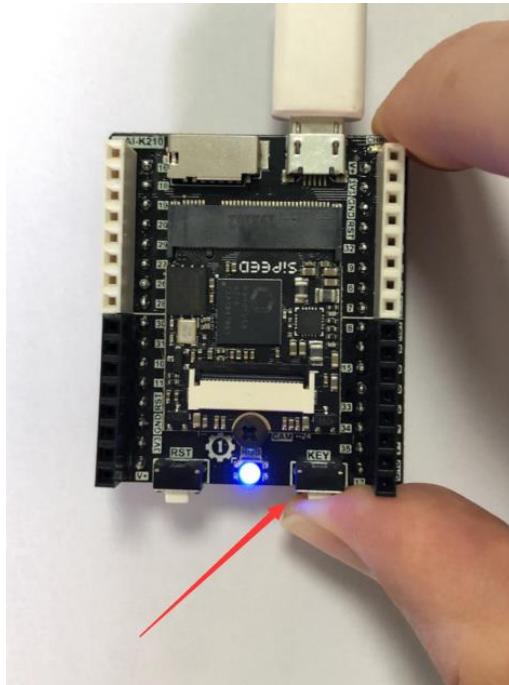


图 4-13 按键实验

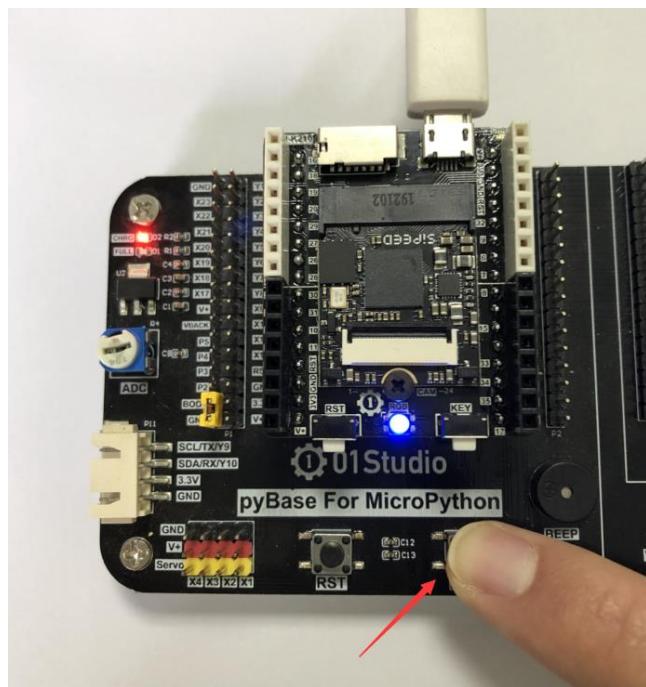


图 4-14 pyBase 上的按键

● 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。

4.4 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。

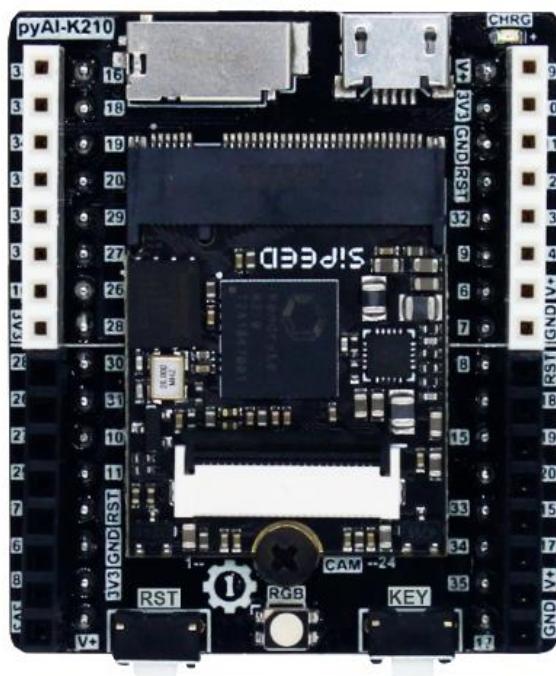


图 4-15

- 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
<code>GPIO(ID,MODE,PULL,VALUE)</code>
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
<code>GPIO.irq(CALLBACK_FUNC,TRIGGER_CONDITION)</code>
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: 下降沿触发 GPIO.IRQ_BOTH: 都触发
<code>GPIO.disirq()</code>
关闭中断。

表 4-4 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：

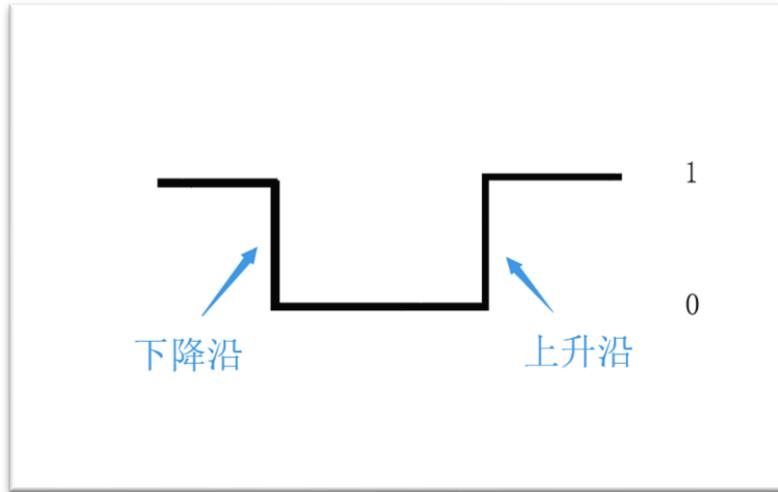


图 4-16 上升沿和下降沿

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：

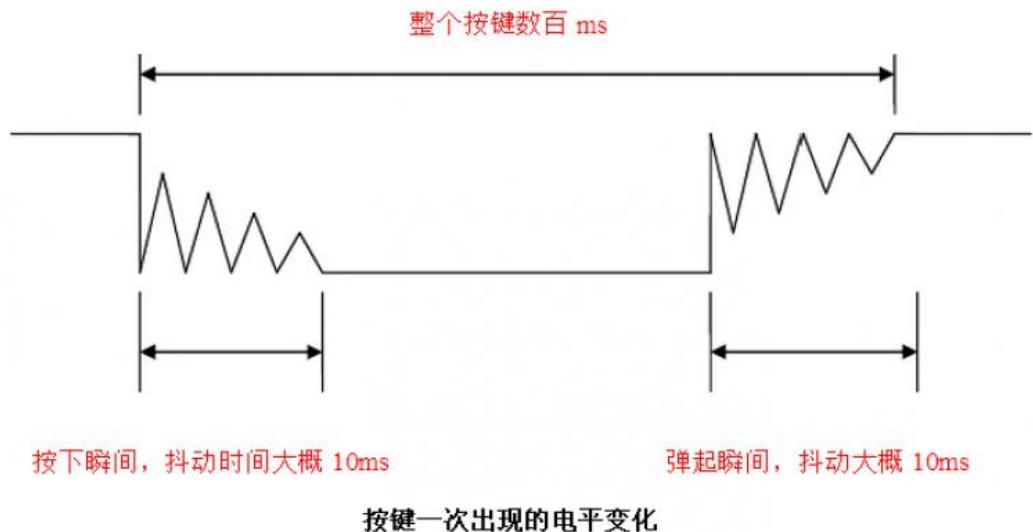


图 4-17 按键按下过程

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 4-5 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：

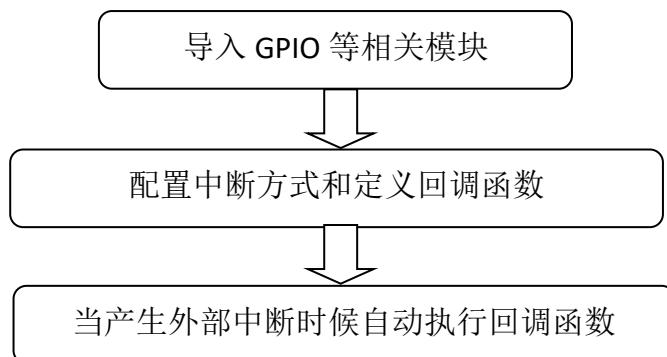


图 4-18 代码编写流程

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。按键可以用 pyAI-K210 底部的按键或者 pyBase 开发底板上的 KEY 按键。

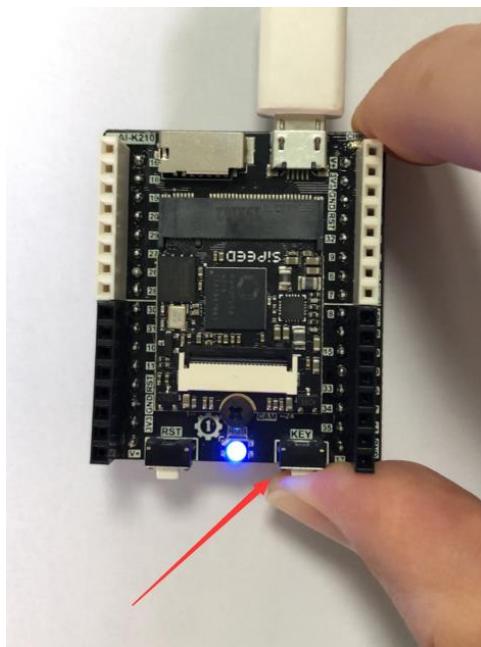


图 4-19 外部中断实验

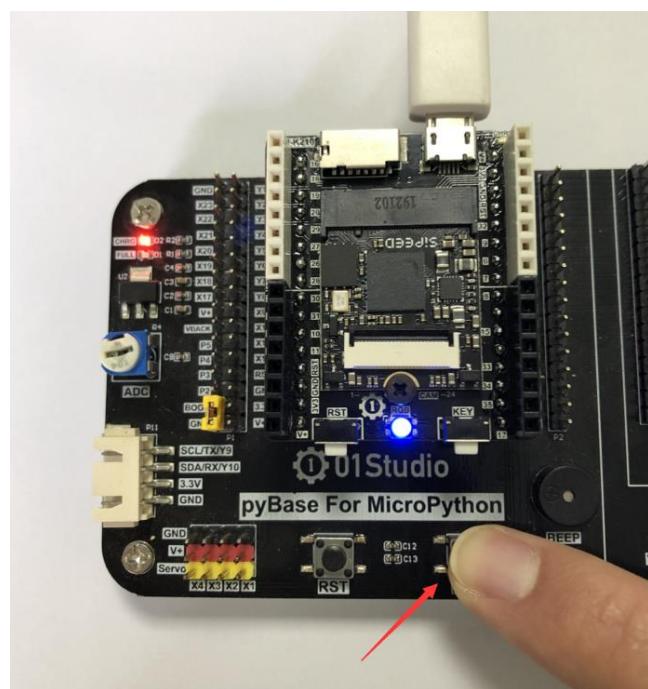


图 4-20 pyBase 上的按键 KEY

● 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。

4.5 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。

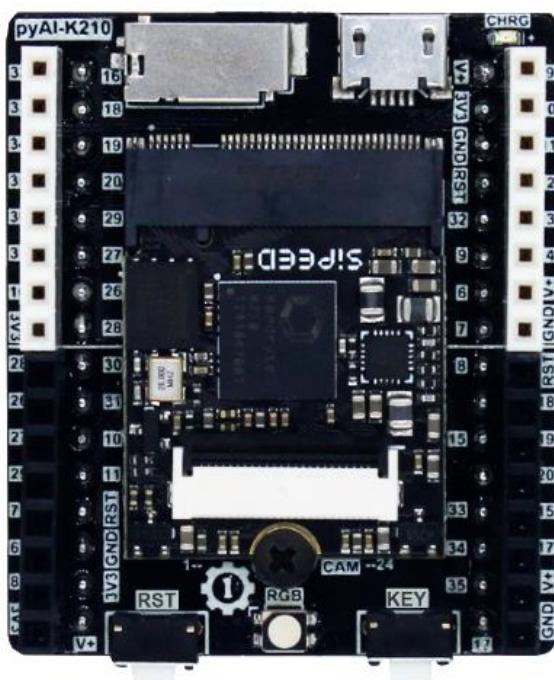


图 4-21

- **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: https://wiki.sipeed.com/soft/maixpy/zh/api_reference/machine/timer.html

表 4-6 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：

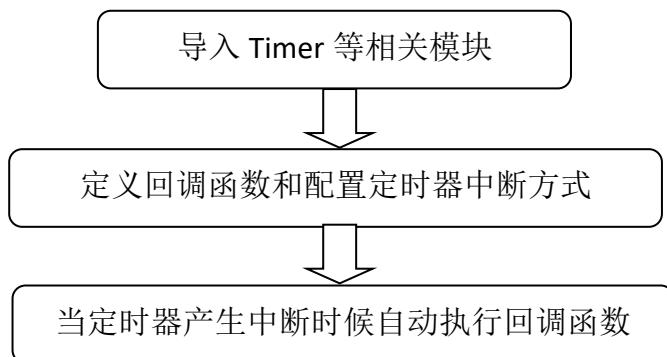


图 4-22 代码编写流程

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。

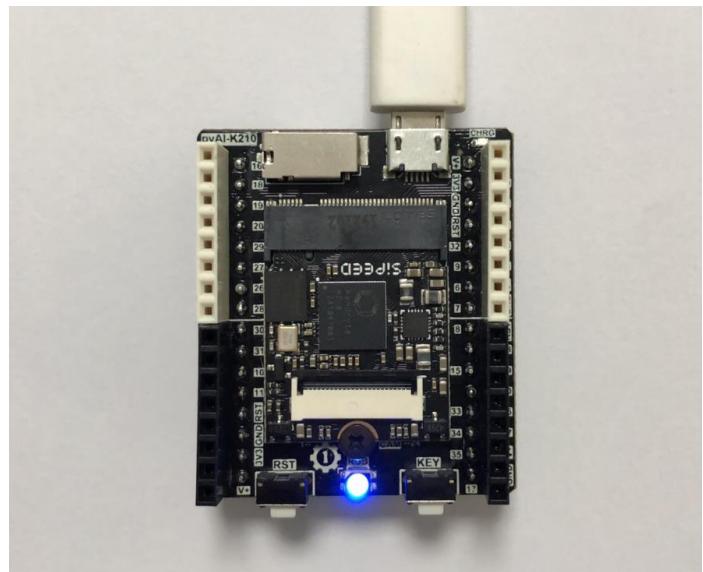


图 4-23 LED 定时闪烁

● 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。

4.6 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。

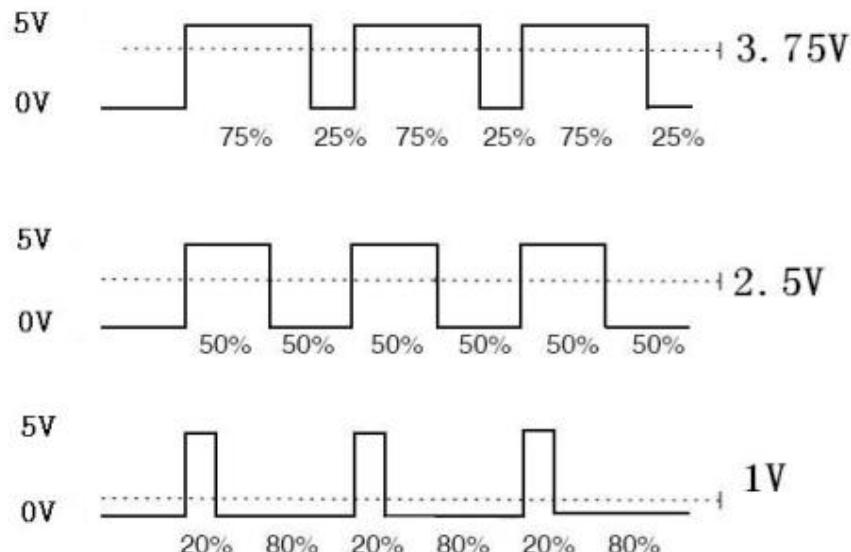


图 4-24 PWM 波形示例 (右边为等效电压)

● 实验平台：

pyAI-K210 开发套件。

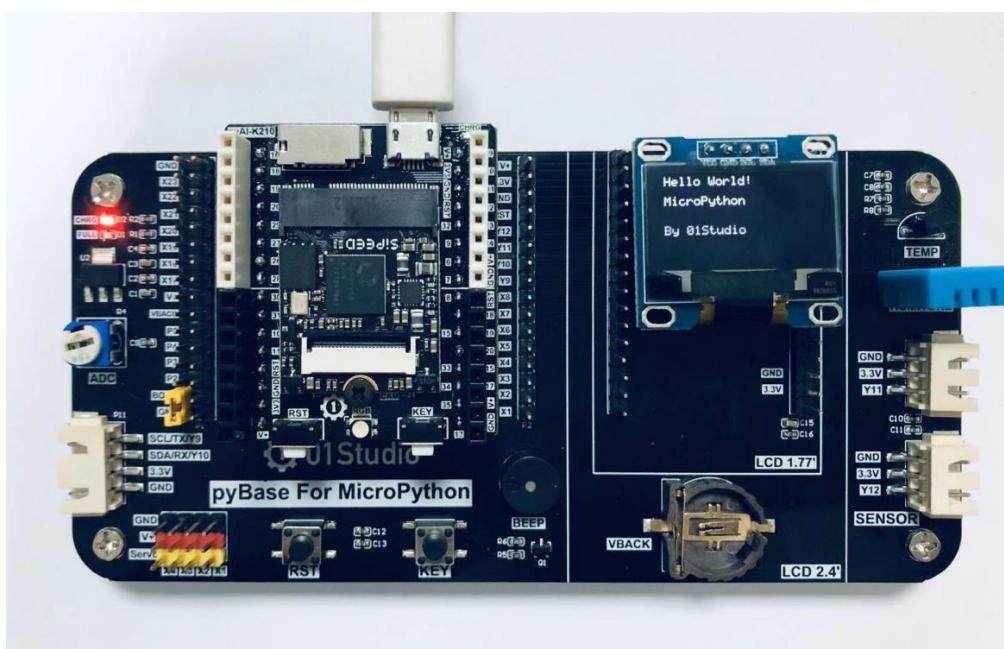


图 4-25

● 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：

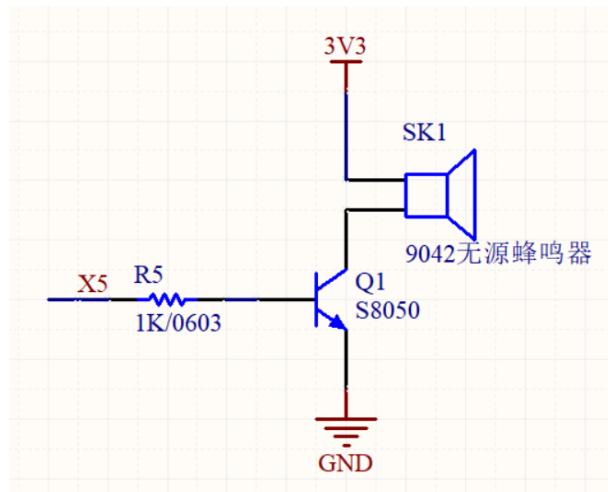


图 4-26 蜂鸣器原理图

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。

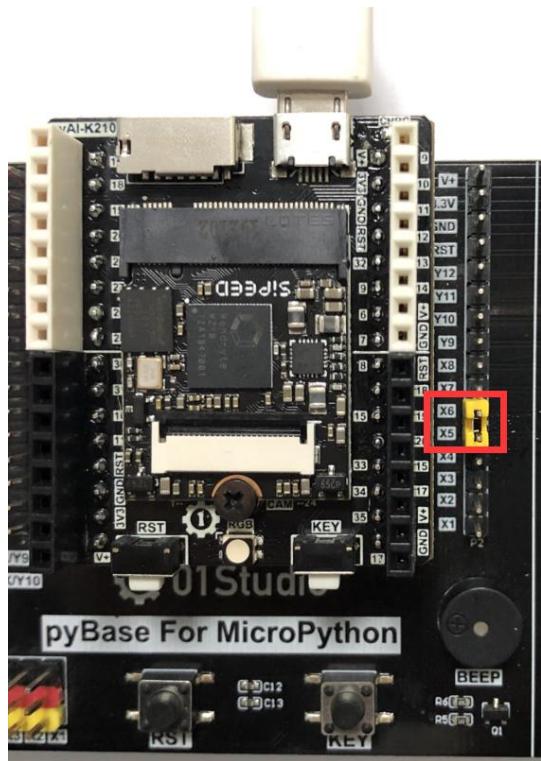


图 4-27 使用跳线帽将 X6 和 X5 短接

我们来看看 PWM 模块对象：

构造函数

```
machine.PWM(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

<code>PWM.enable()</code>
使能 PWM 输出。
<code>PWM.disable()</code>
暂停 PWM 输出。
<code>PWM.deinit()</code>
注销 PWM。

表 4-7 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。

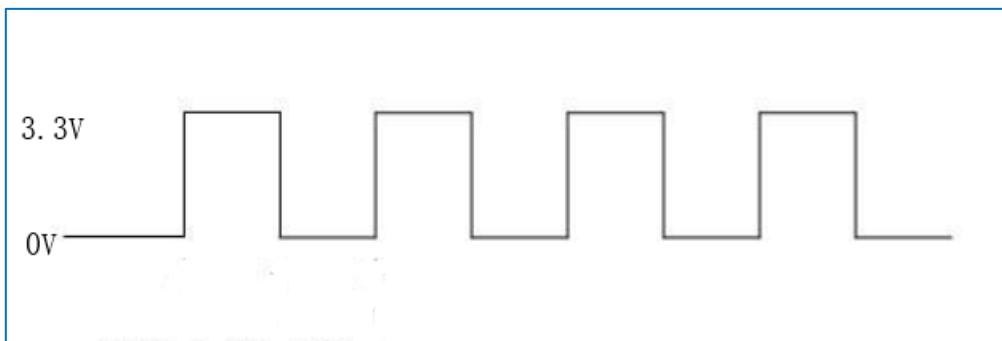


图 4-28 方波信号

结合上述讲解，总结出代码编写流程图如下：

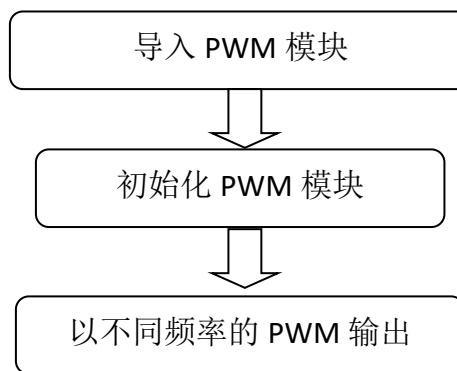


图 4-29 代码编写流程图

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：

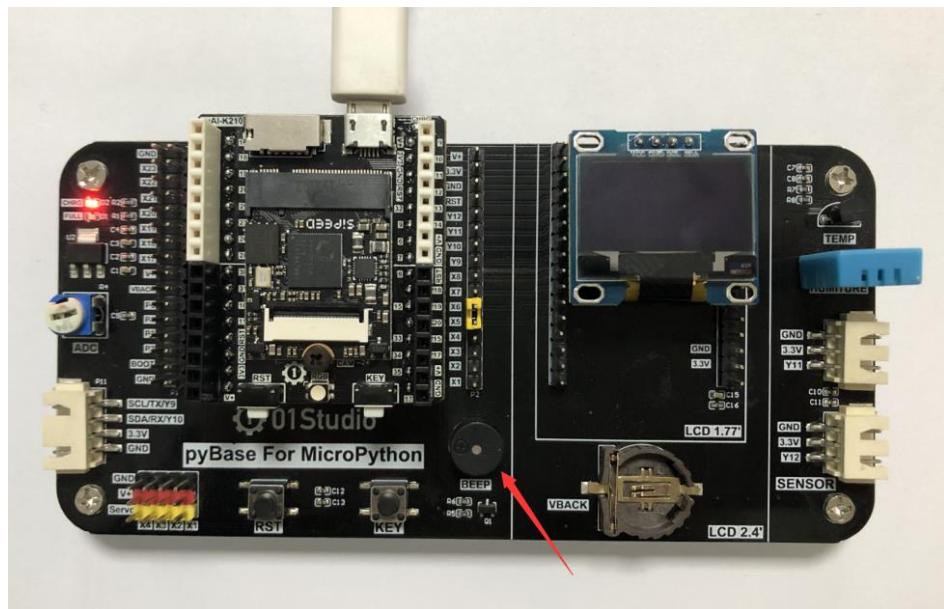


图 4-30

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：

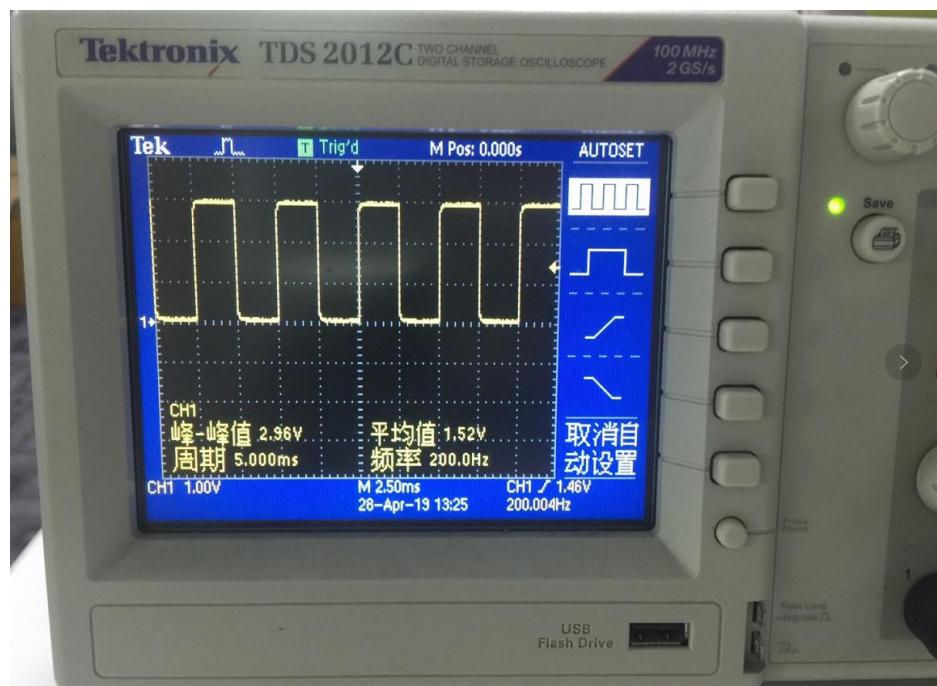


图 4-31 示波器显示波形

- **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。

4.7 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。

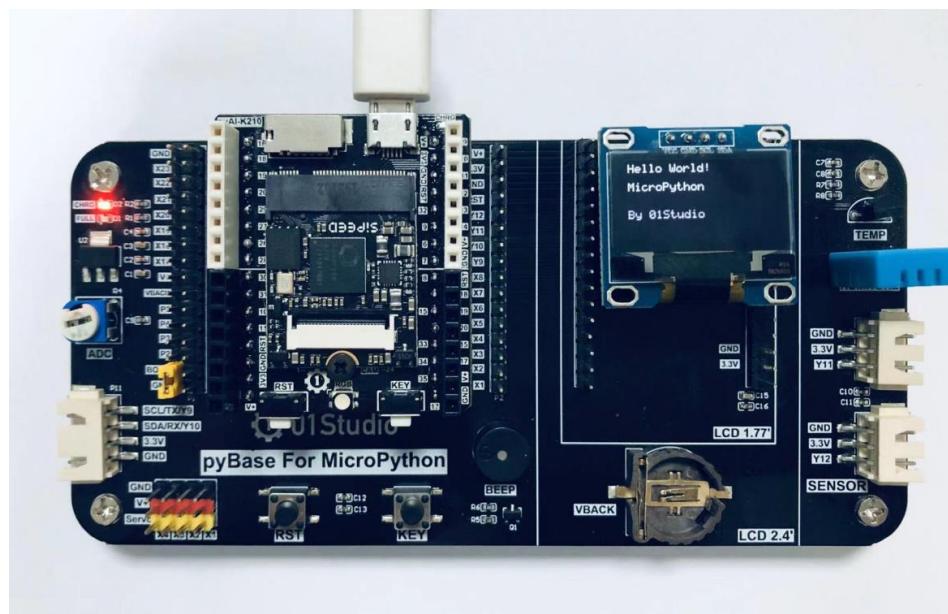


图 4-32

● 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 4-33 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。

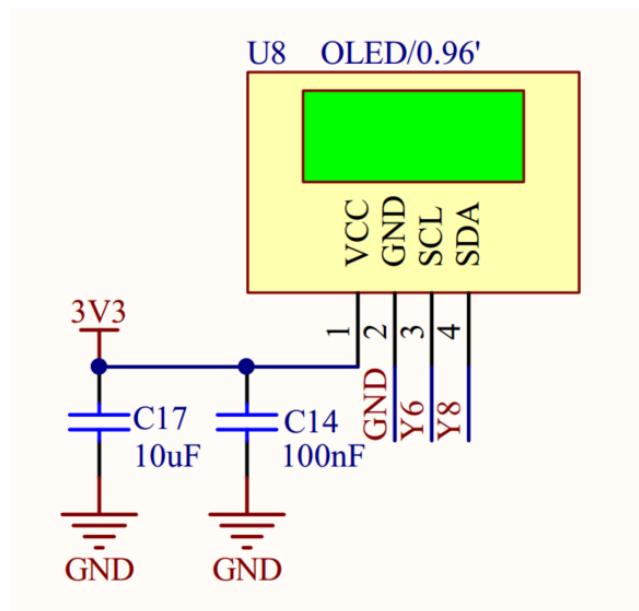
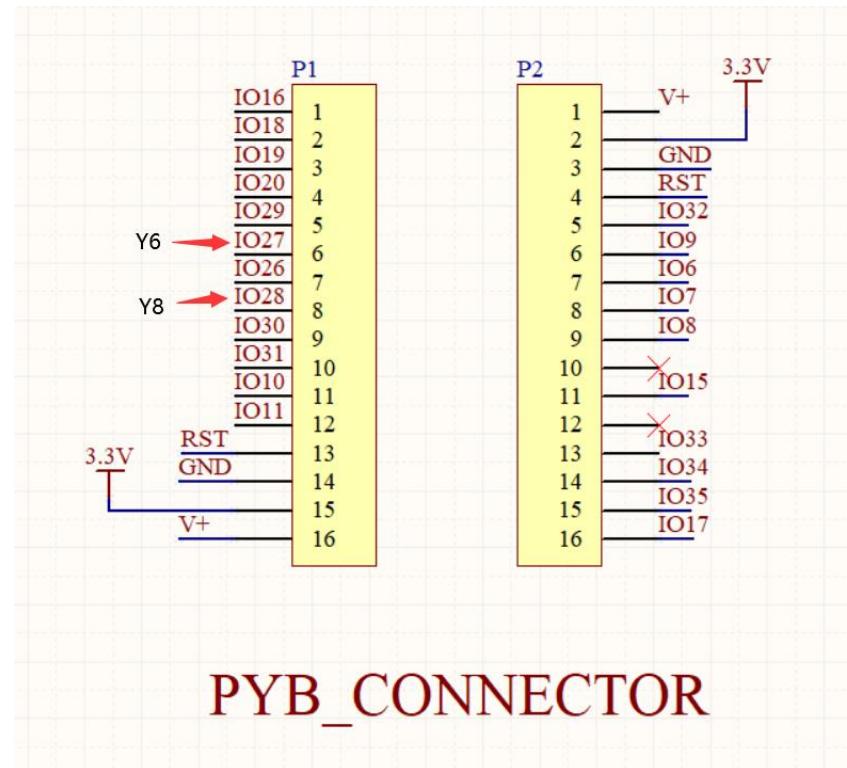


图 4-34 OLED 接口原理图

我们再来看看 pyAI-K210 转接板的原理图接口部分。



PYB_CONNECTOR

表 4-8 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 $Y6 \rightarrow SCL$ 和 $Y8 \rightarrow SDA$ 。对应 pyAI-K210 的关系是： $IO27 \rightarrow Y6 \rightarrow SCL$, $IO28 \rightarrow Y8 \rightarrow SDA$ 。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
<pre>machine.I2C(id, mode=I2C.MODE_MASTER, scl=None, sda=None, freq=400000, timeout=1000, addr=0, addr_size=7)</pre>

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： https://wiki.sipeed.com/soft/maixpy/zh/api_reference/machine/i2c.html

表 4-9 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 字符写在指定位置。

【string】: 字符;

【x】 横坐标; [0-127];

【y】 纵坐标。[0-7] 共 8 行。

表 4-10 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：

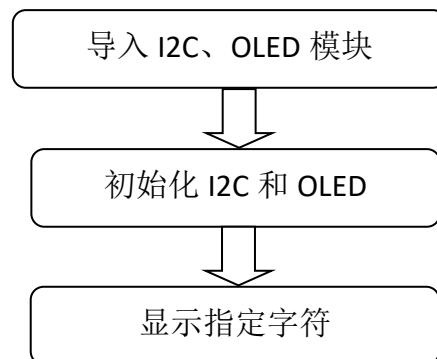


图 4-35 代码编写流程

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
```

```

#define 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。

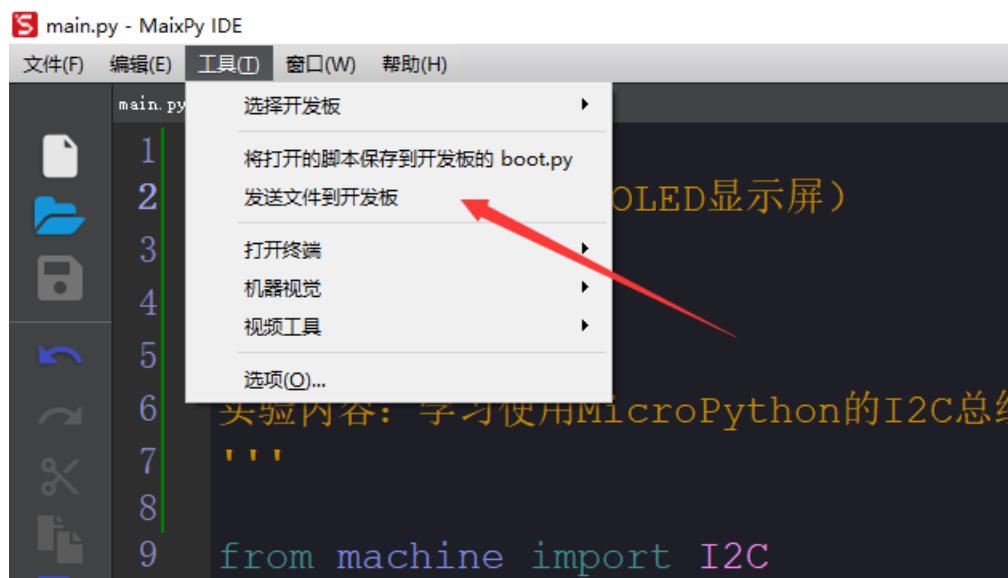


图 4-36 将 ssd1306k.py 拷贝到设备

● 实验结果：

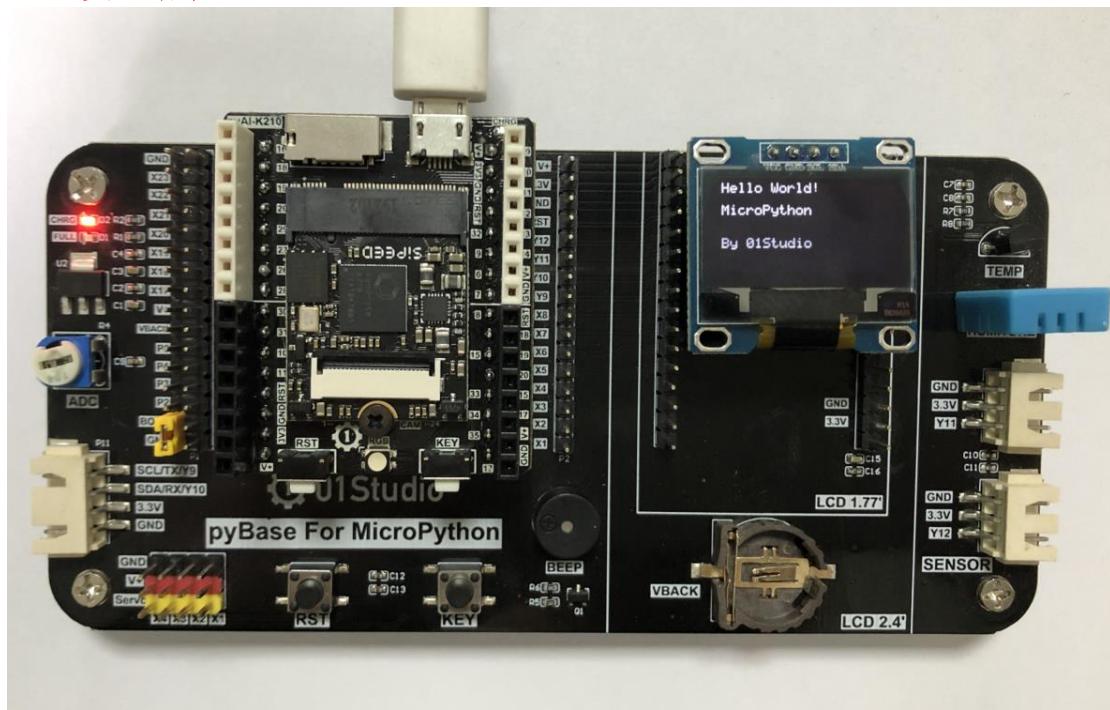


图 4-37 OLED 实验结果

● 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。

4.8 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。

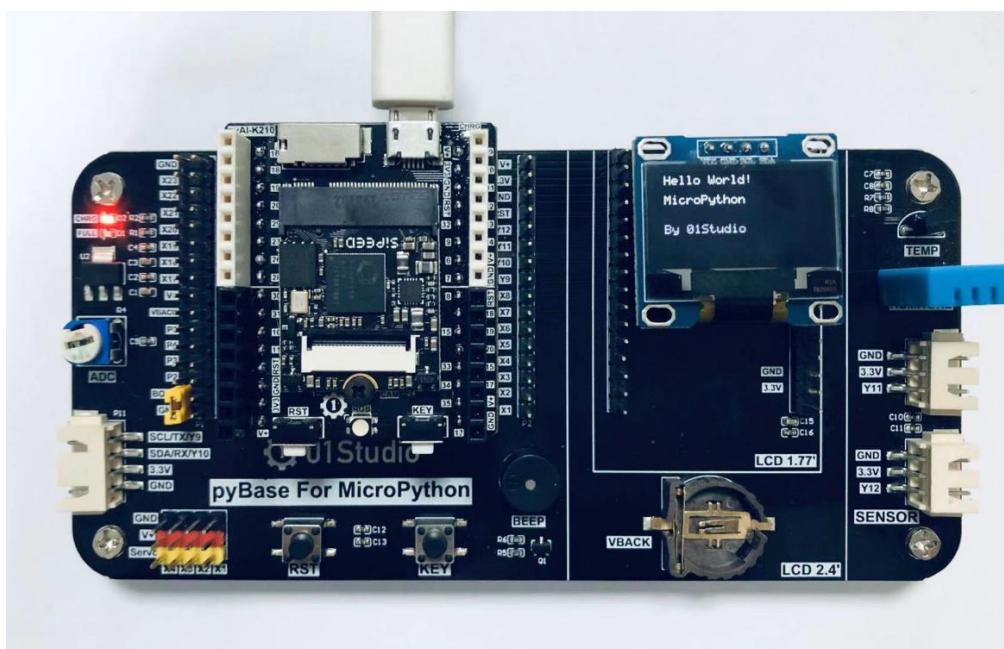


图 4-38 MicroPython 开发套件

- **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：

```
# I06→RX1, I07→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
<code>machine.UART(uart,baudrate,bits,parity,stop,timeout, read_buf_len)</code>
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
<code>UART.read(num)</code>
读取串口缓冲数据
【num】读取字节数
<code>UART.readline(num)</code>
读取串口缓冲数据的行
【num】行数
<code>UART.write(buf)</code>
串口发送数据
【buf】需要发送的数据
<code>UART.deinit()</code>
注销串口

表 4-11 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。

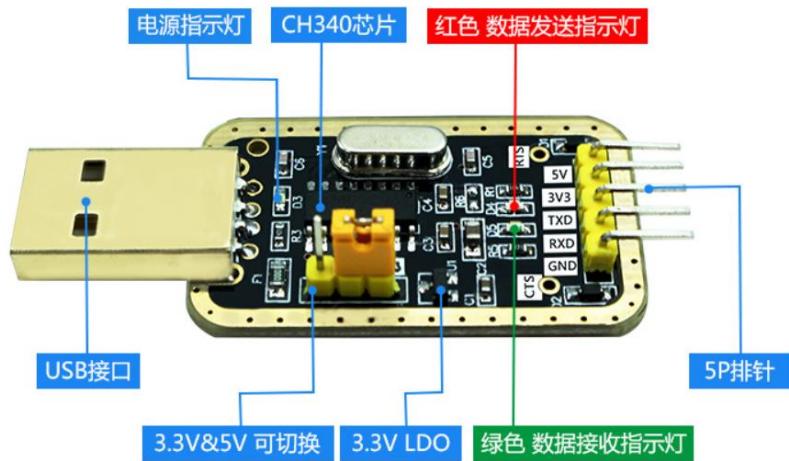


图 4-39 常用 USB 转串口工具

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：

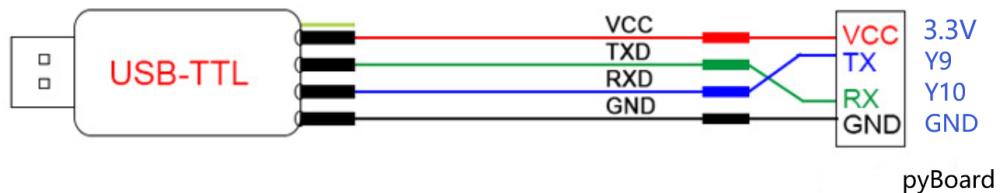


图 4-40 串口接线图

从 pyAI-K210 原理图可以看到外部 IO6→Y9→RX , IO7→Y10→TX。

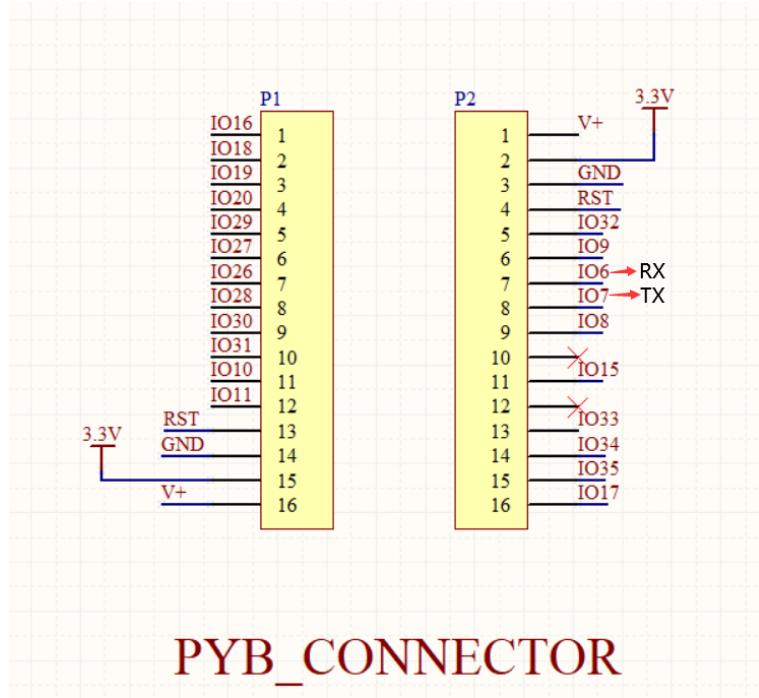


图 4-41 串口引脚

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：

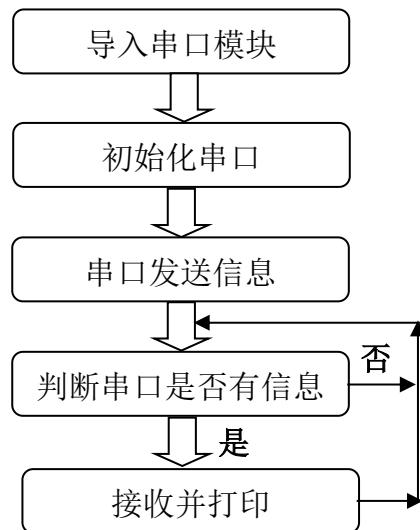


图 4-42 代码编程流程图

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。)

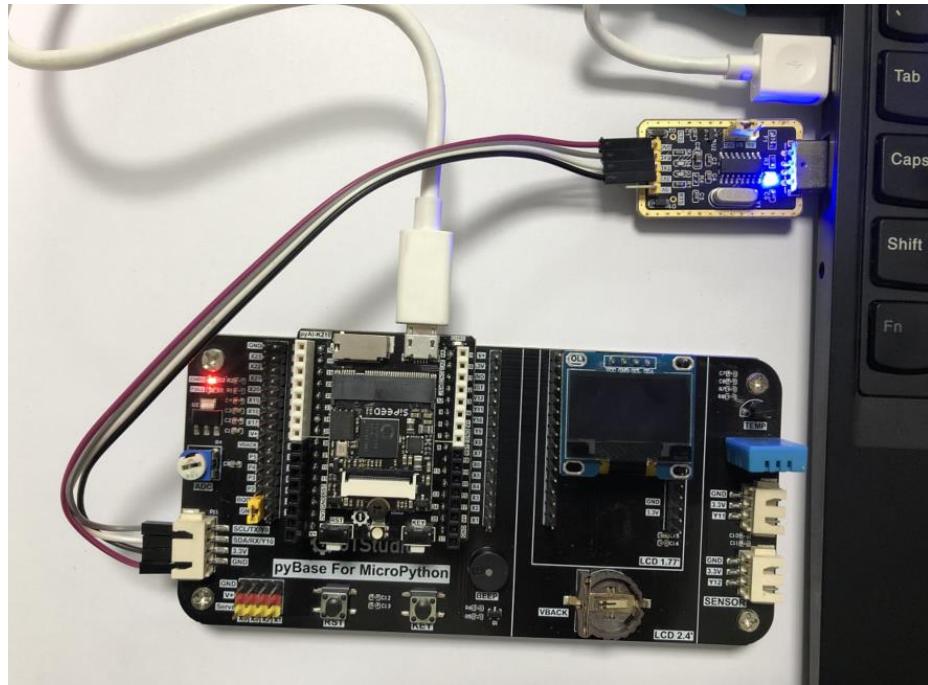


图 4-43 将 pyboard 和 usb ttl 工具同时接入电脑

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 4-44

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。

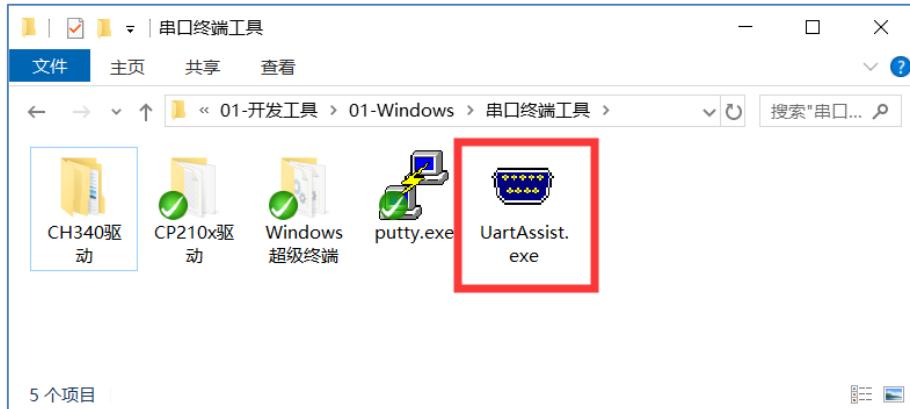


图 4-45

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：

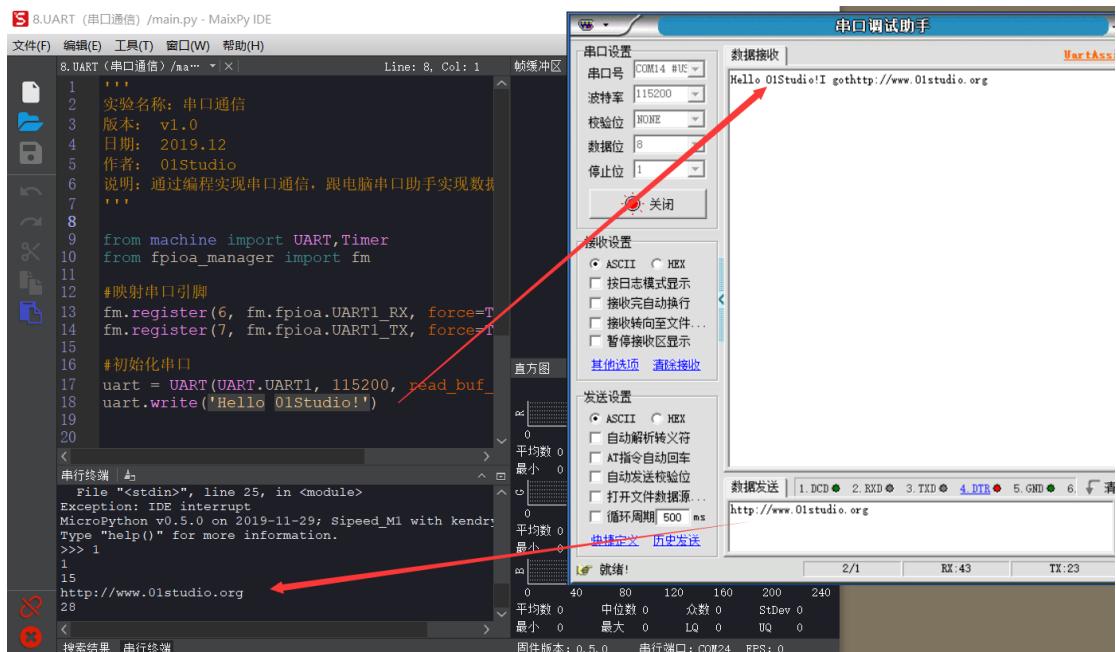


图 4-46 串口收发实验

● 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。

4.9 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。

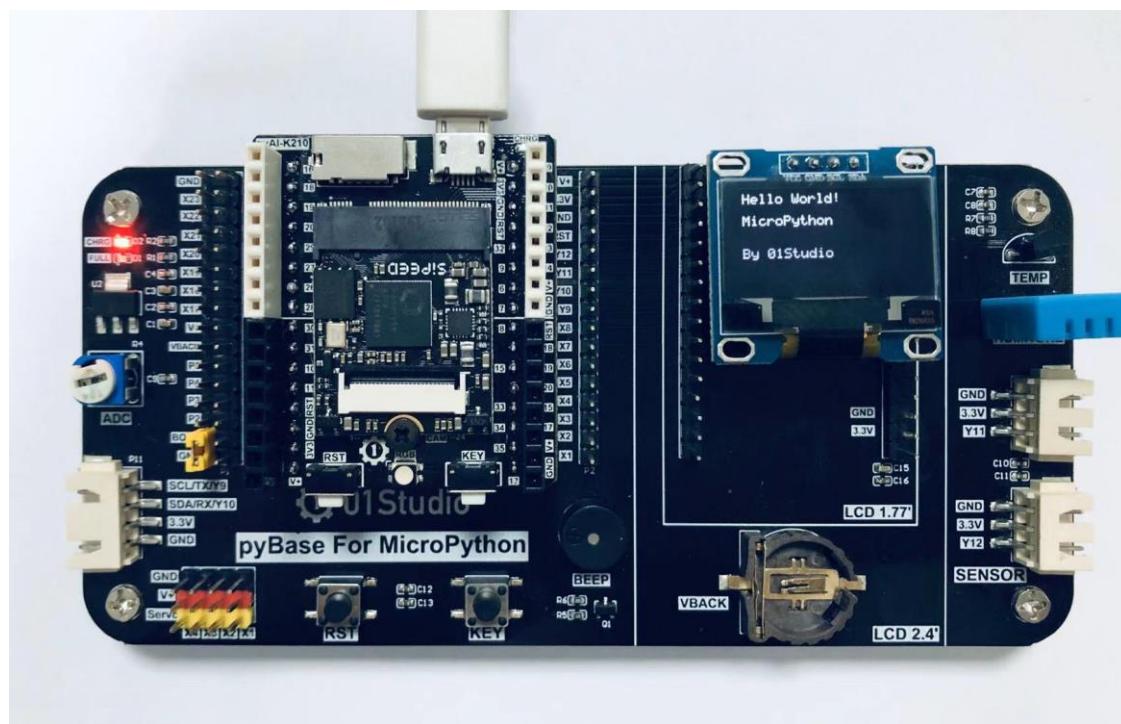


图 4-47

- 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：

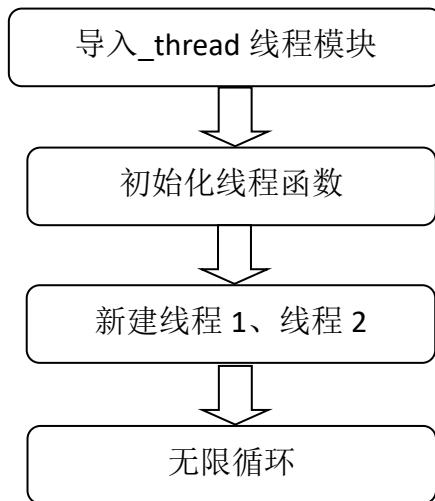


图 4-48 代码编写流程

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。

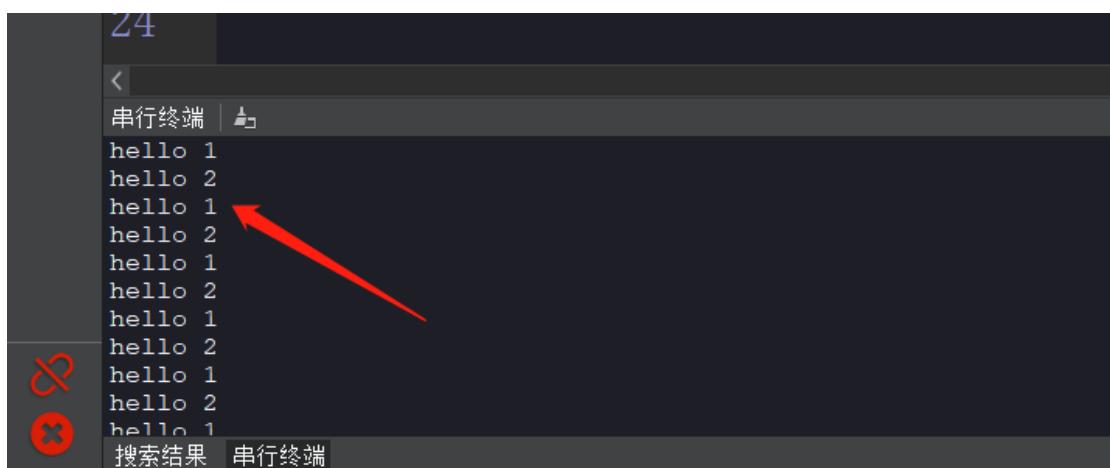


图 4-49 实验结果

● 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5章 机器视觉

5.1 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。

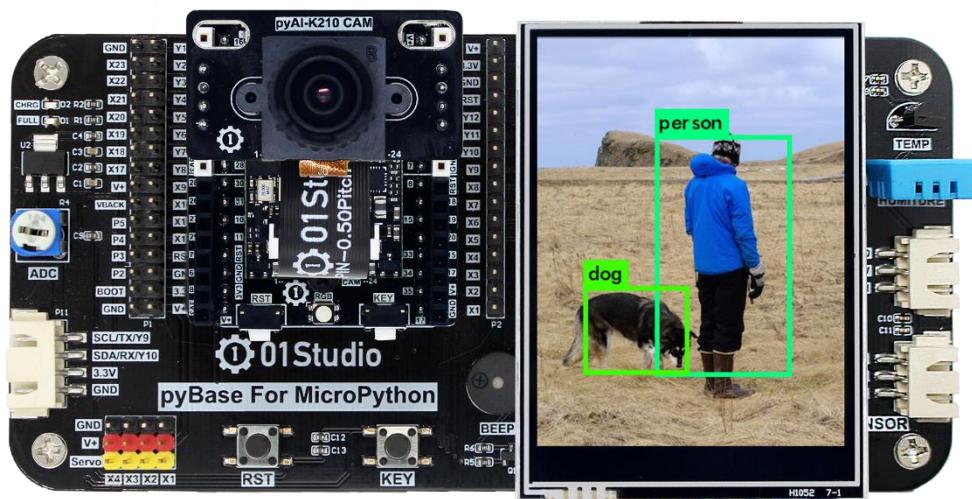


图 5-1 2.8 寸 LCD (电阻触摸)

- **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：

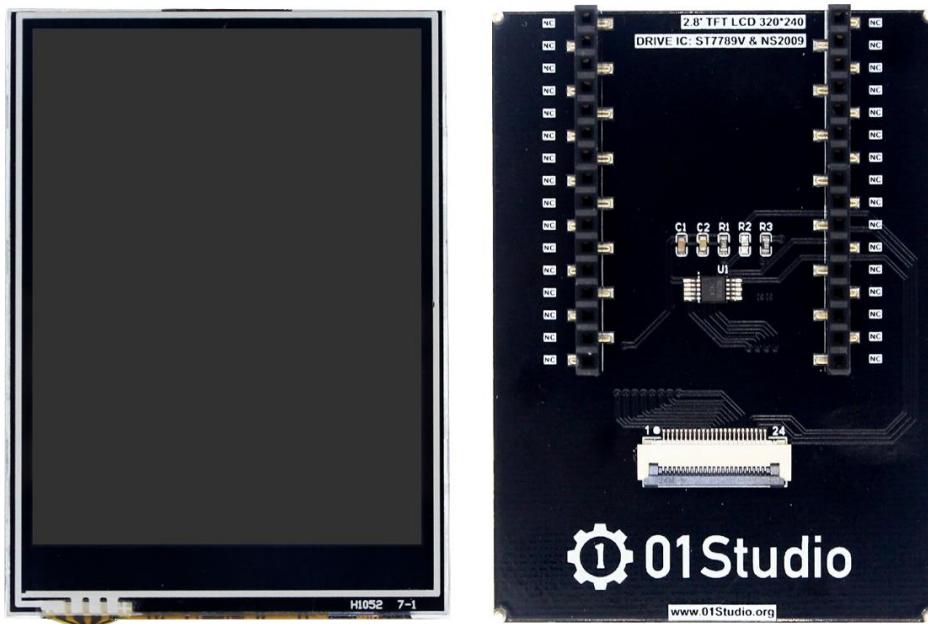


图 5-2 2.8 寸 LCD 显示屏 (带触摸)

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 5-1 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。

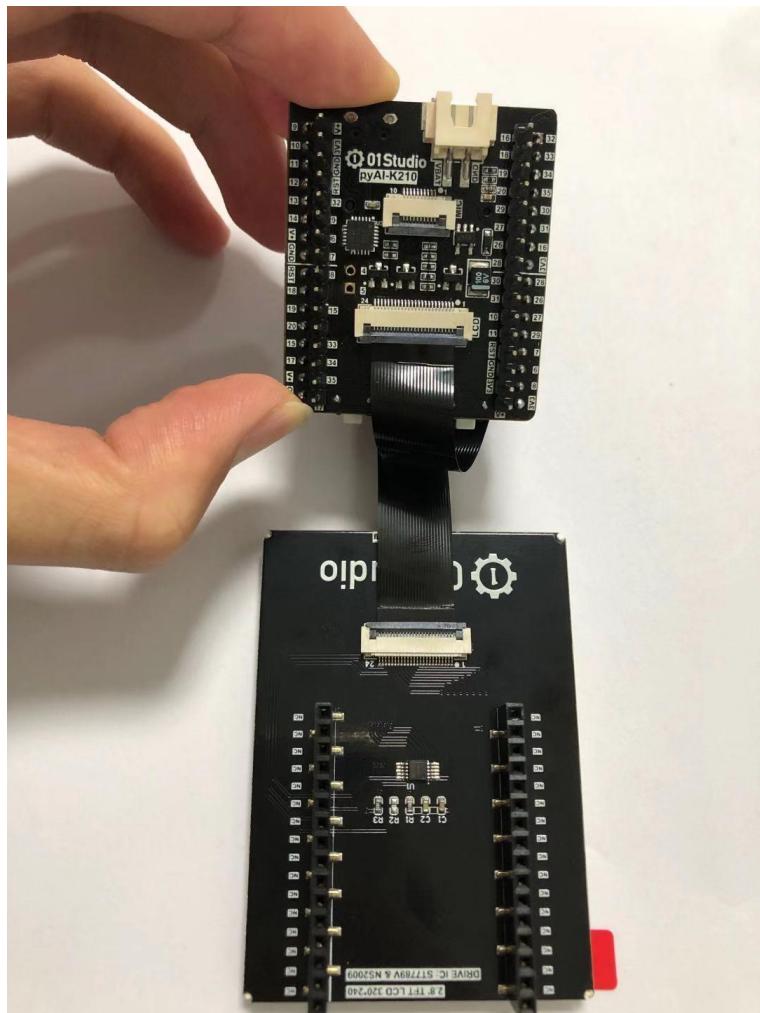


图 5-3 pyAI-K210 与 2.8 寸 LCD 接线方式

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：

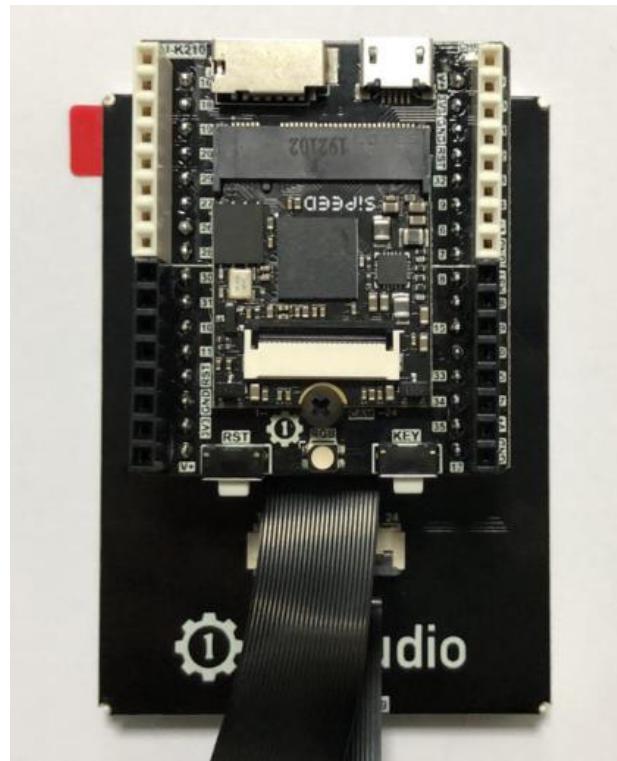


图 5-4 LCD 与 pyAI-K210

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 5-5 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
<code>lcd.init(type=1, freq=15000000, color= lcd.BLACK)</code>
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
<code>lcd.deinit()</code>
注销 LCD 驱动，释放 IO 引脚。
<code>lcd.clear(color)</code>
填充指定颜色。默认是黑色
<code>lcd.draw_string(x,y,str,color,bg_color)</code>
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
<code>lcd.display(image,roi=Auto)</code>

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://wiki.sipeed.com/soft/maixpy/zh/api_reference/machine_vision/lcd.html

表 5-2 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：

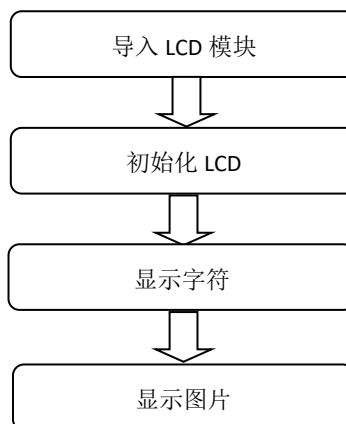


图 5-6 编程流程图

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
```

```
lcd.rotation(1) #由于图像默认是 240*320, 因此顺时钟旋转 90° 。
```

```
#显示图像, 必须先将 01Studio.bmp 文件发送到开发板才能正常运行
```

```
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。

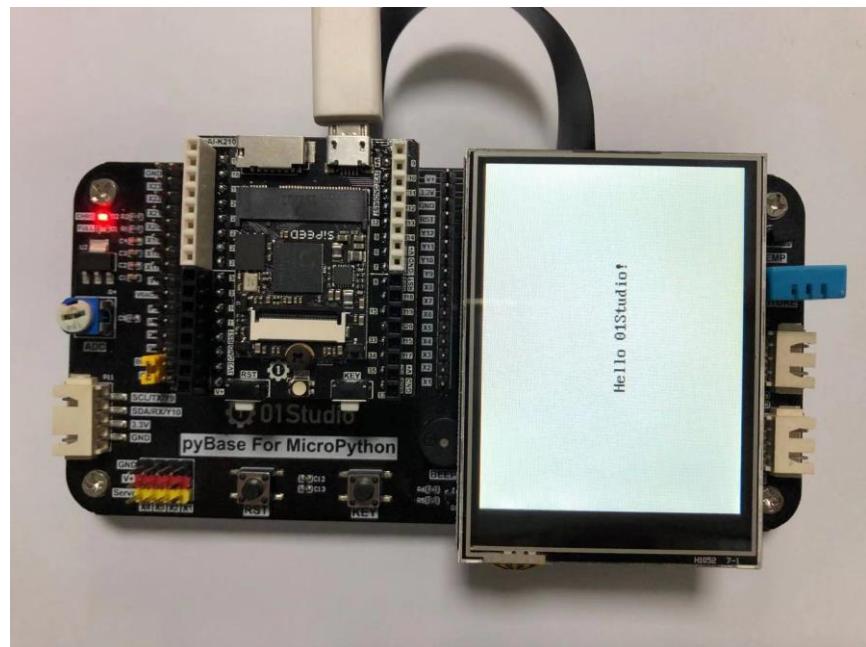


图 5-7 显示字符

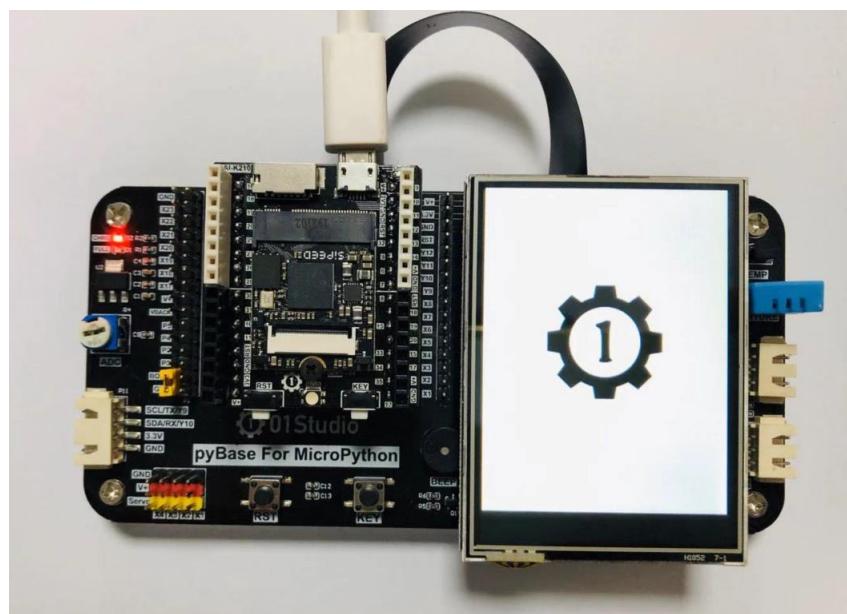


图 5-8 显示 LOGO 图片

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。）

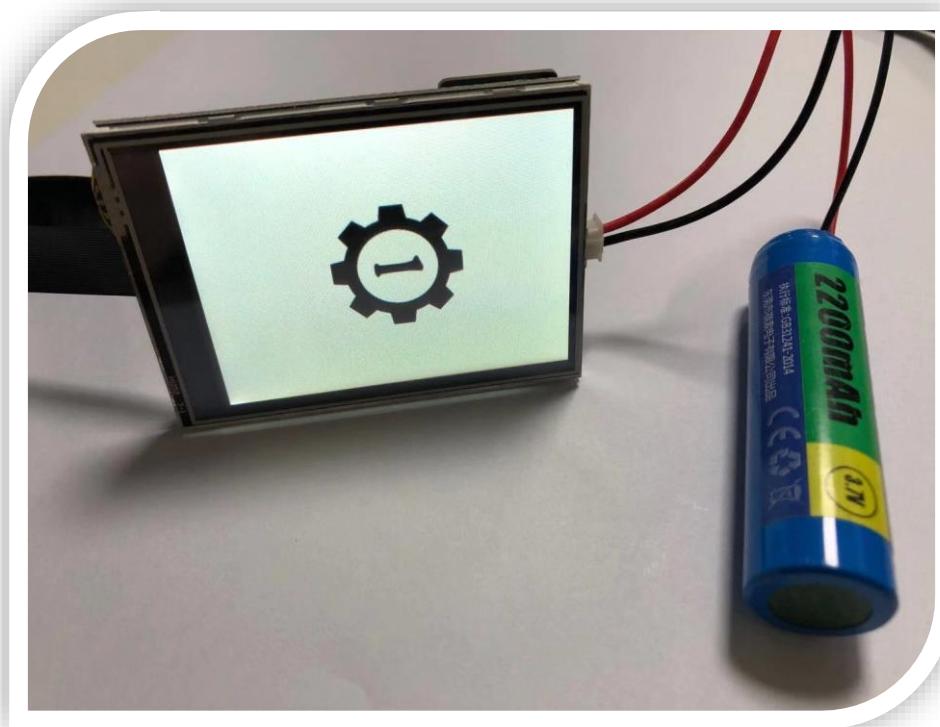


图 5-9 LCD 直接连接 pyAI-K210

● 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。

5.2 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。

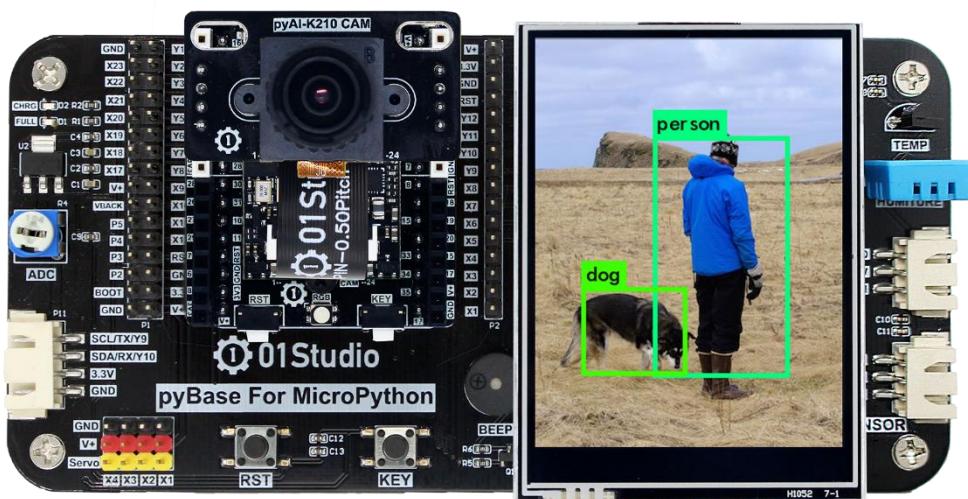


图 5-10 pyAI-K210+摄像头模块

- 实验目的：

学习官方自带的 `hello world` 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：

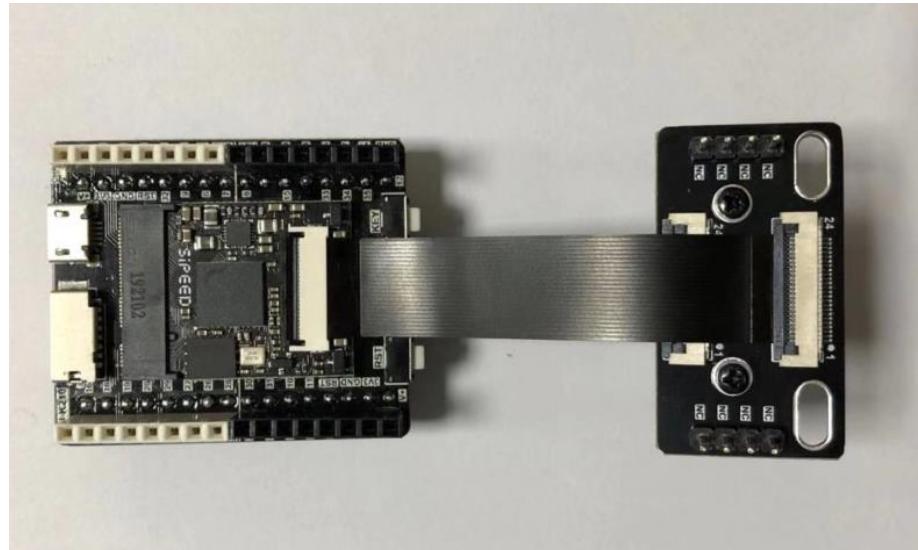


图 5-11 摄像头接线方式

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。

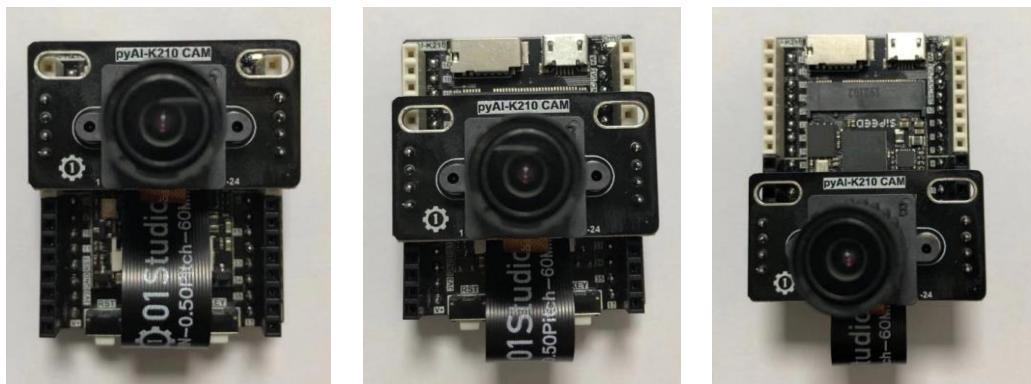


图 5-12 摄像头位置任意摆放

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。

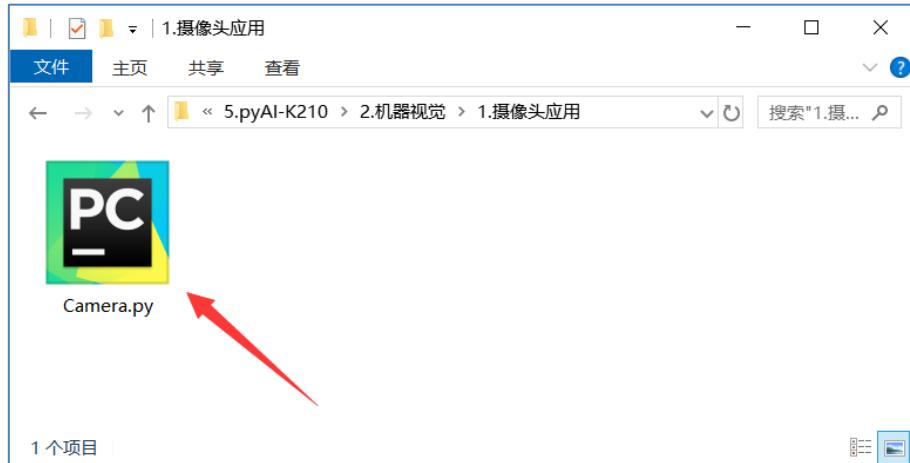


图 5-13 摄像头例程位置

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。)

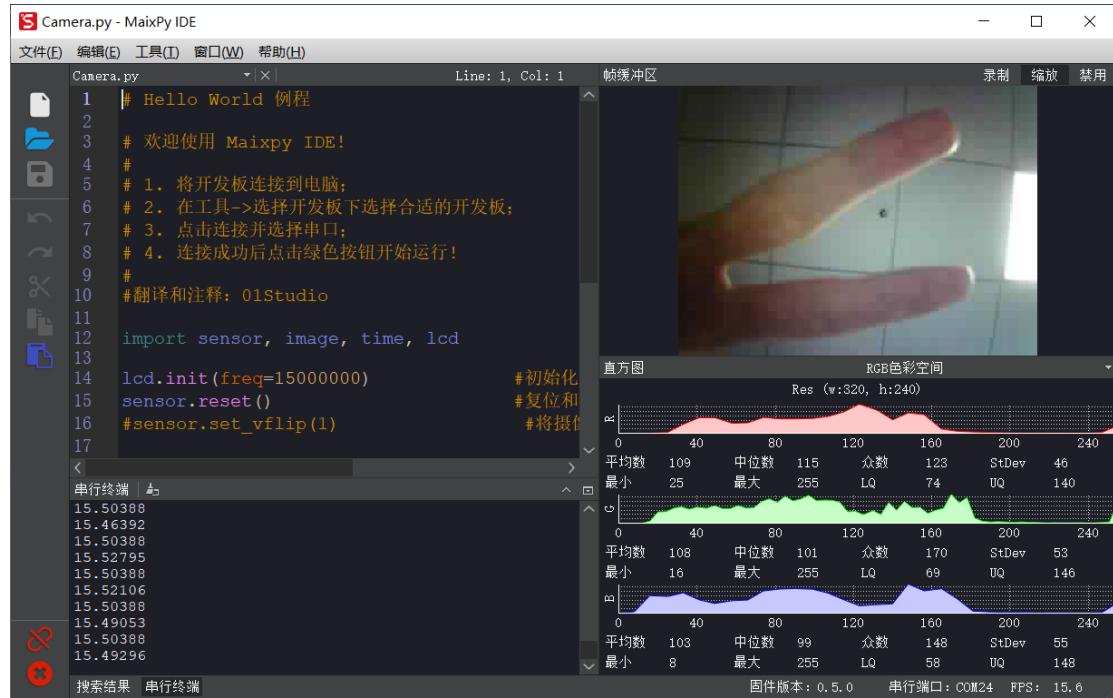


图 5-14

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 `sensor` 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
<code>sensor</code>
摄像头对象，通过 <code>import</code> 直接调用
使用方法
<code>sensor.reset()</code>
初始化摄像头
<code>sensor.set_pixformat(<i>pixformat</i>)</code>
设置像素格式。 <code>pixformat</code> 有 3 个参数。 <code>sensor.GRAYSCAL</code> : 灰度图像，每像素 8 位 (1 字节)，处理速度快； <code>sensor.RGB565</code> : 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。
<code>sensor.set_framesize(<i>framesize</i>)</code>
设置每帧大小 (即图像尺寸)。常用的 <code>framesize</code> 参数有下面这些： <code>sensor.QQVGA: 160*120;</code> <code>sensor.QVGA: 320*240;</code> <code>sensor.VGA: 640*480;</code>
<code>sensor.skip_frames([<i>n, time</i>])</code>
摄像头配置后跳过 <code>n</code> 帧或者等待时间 <code>time</code> 让其变稳定。 <code>n</code> : 跳过帧数； <code>time</code> : 等待时间，单位 ms。 (如果 <code>n</code> 和 <code>time</code> 均没指定，则默认跳过 300 毫秒的帧。)
<code>sensor.snapshot()</code>
使用相机拍摄一张照片，并返回 <code>image</code> 对象。
*其它更多用法请阅读 MaixPy 官方文档： https://wiki.sipeed.com/soft/maixpy/zh/api_reference/machine_vision/sensor.html

表 5-3 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
<code>clock=time.clock()</code>
创建一个时钟。
使用方法
<code>clock.tick()</code>
开始追踪运行时间。
<code>clock.fps ()</code>
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 <code>tick</code> 。
*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 5-15 `clock` 对象

我们来看看 `helloworld` 代码的编写流程图：

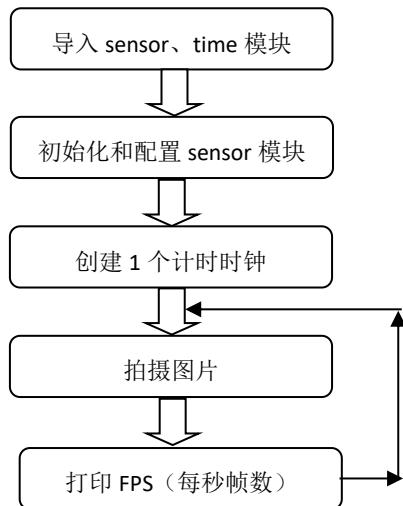


图 5-16 `helloworld` 代码编写流程

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 RGB 颜色直方图。

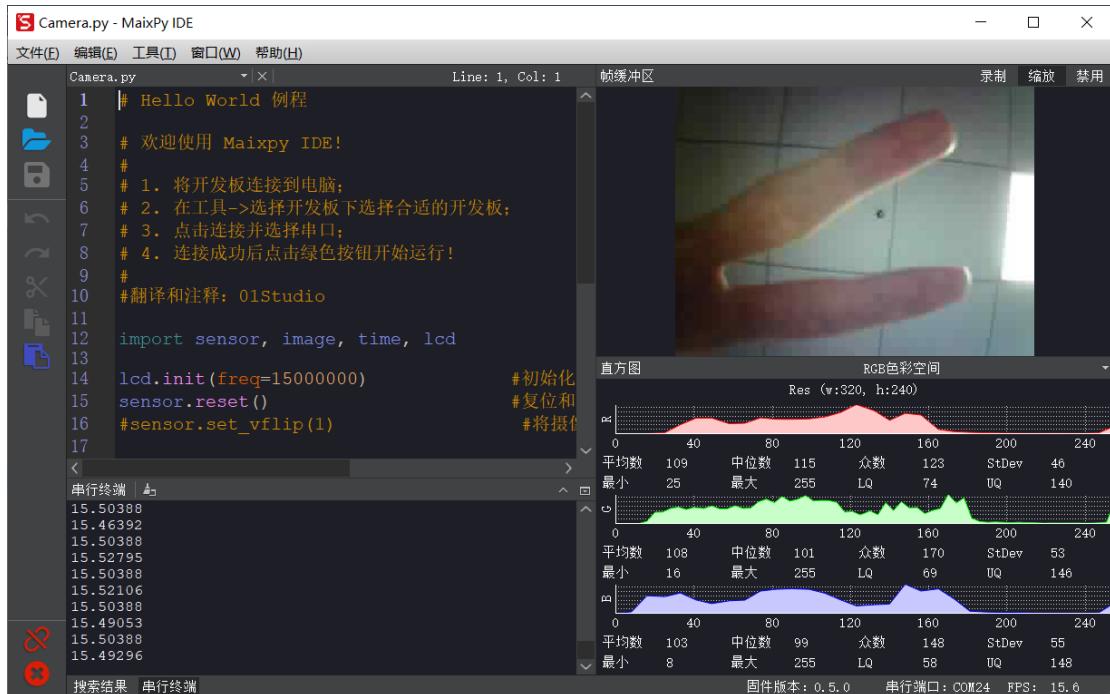


图 5-17 实验现象

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(每秒帧数)值约为 15 帧。

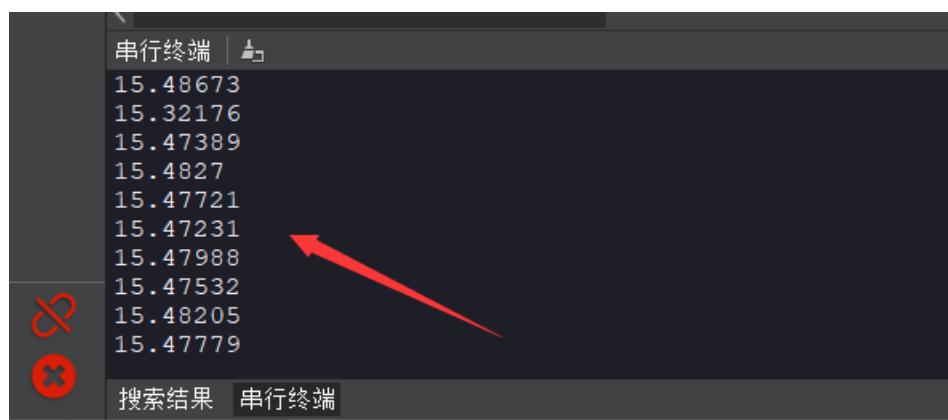


图 5-18 串口终端显示 FPS

同样可以看到 LCD 实时显示摄像头采集的图像。

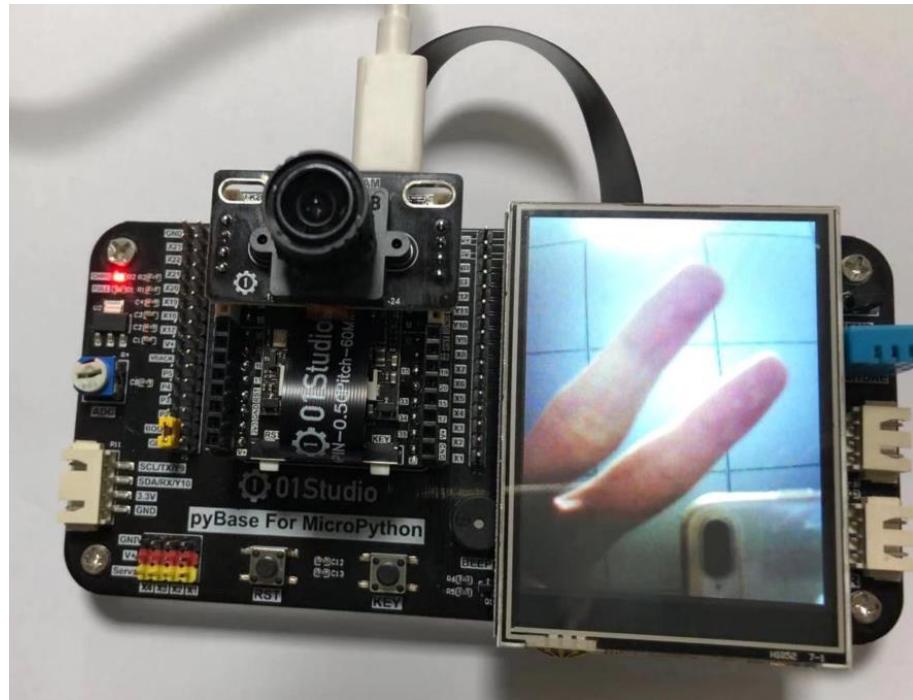


图 5-19 LCD 显示摄像头图像

● 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。

5.3 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。

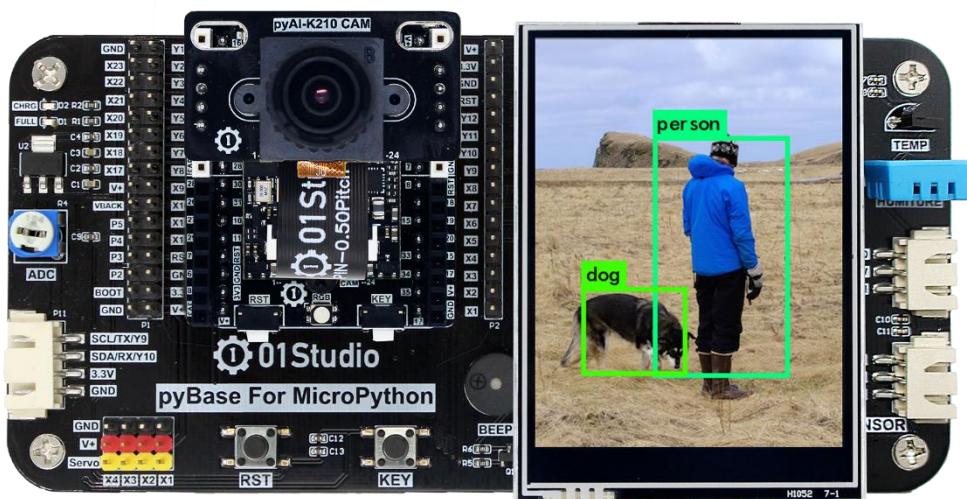


图 5-20 pyAI-K210 开发套件

- **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
<code>img=sensor.snapshot() 或 img=image.Image(path[, copy_to_fb=False])</code>
创建图像，通过拍摄或者读取文件路径获取。 <code>copy_to_fb=True</code> : 可以加载大图片; <code>copy_to_fb=False</code> : 不可以加载大图片。
示例: <code>img = image.Image("01Studio.bmp", copy_to_fb=True)</code> , 表示加载根目录下的 <code>01Studio.bmp</code> 图片。
使用方法
<code>image.draw_line(x0, y0, x1, y1[, color[, thickness=1]])</code>
画线段。 <code>(x0,y0)</code> :起始坐标; <code>(x1,y1)</code> :终点坐标; <code>color</code> :颜色, 如 <code>(255,0,0)</code> 表示红色; <code>thickness</code> : 粗细。
<code>image.draw_rectangle(x, y, w, h[, color[, thickness=1[, fill=False]]])</code>
画矩形。 <code>(x,y)</code> :起始坐标; <code>w</code> :宽度; <code>h</code> :长度; <code>color</code> : 颜色; <code>thickness</code> : 边框粗细; <code>fill</code> :是否填充。
<code>image.draw_circle(x, y, radius[, color[, thickness=1[, fill=False]]])</code>
画圆。 <code>(x,y)</code> :圆心; <code>radius</code> :半径; <code>color</code> : 颜色; <code>thickness</code> :线条粗细; <code>fill</code> : 是否填充。
<code>image.draw_arrow(x0, y0, x1, y1[, color[, size,[thickness=1]]])</code>
画箭头。 <code>(x0,y0)</code> :起始坐标; <code>(x1,y1)</code> :终点坐标; <code>color</code> :颜色; <code>size</code> :箭头位置大小。 <code>thickness</code> : 线粗细。
<code>image.draw_cross(x, y[, color[, size=5[, thickness=1]]])</code>
画十字交叉。 <code>(x,y)</code> :交叉坐标; <code>color</code> :颜色; <code>size</code> :尺寸; <code>thickness</code> : 线粗细。
<code>image.draw_string(x, y, text[, color[, scale=1[,mono_space=True...]]])</code>
写字符。 <code>(x,y)</code> : 起始坐标; <code>text</code> :字符内容; <code>color</code> : 颜色; <code>scale</code> : 字体大小; <code>mono_space</code> :强制间距。
*其它更多用法请阅读 MaixPy 官方文档: https://wiki.sipeed.com/soft/maixpy/zh/api_reference/machine_vision/image/image.html

表 5-4 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：

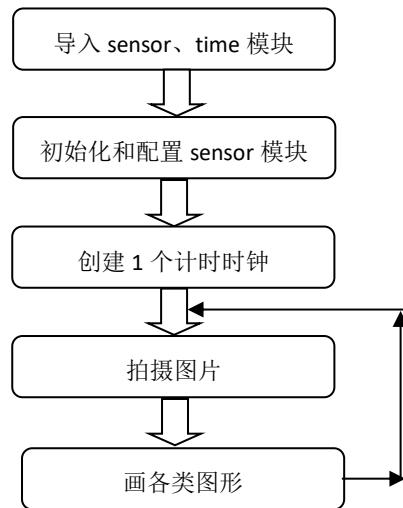


图 5-21 代码编写流程

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。

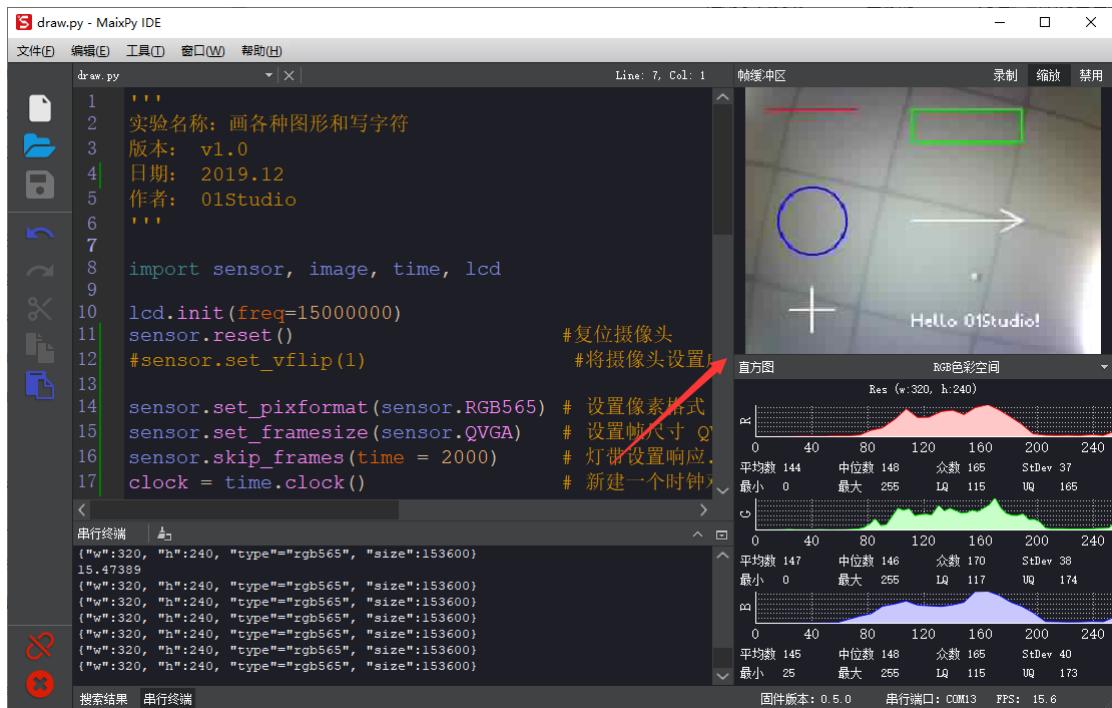


图 5-22 画图形

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。

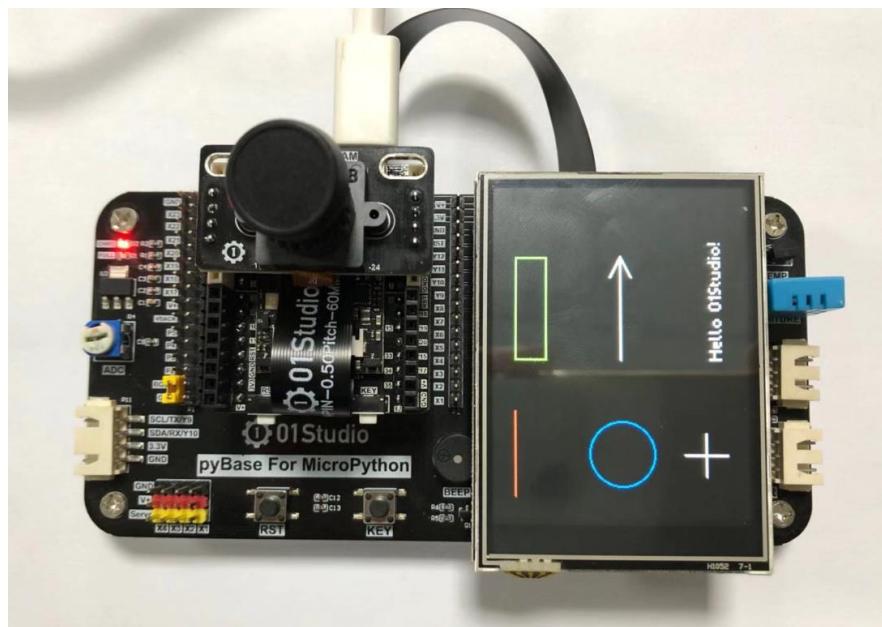


图 5-23 LCD 上显示结果

● 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。

5.4 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。

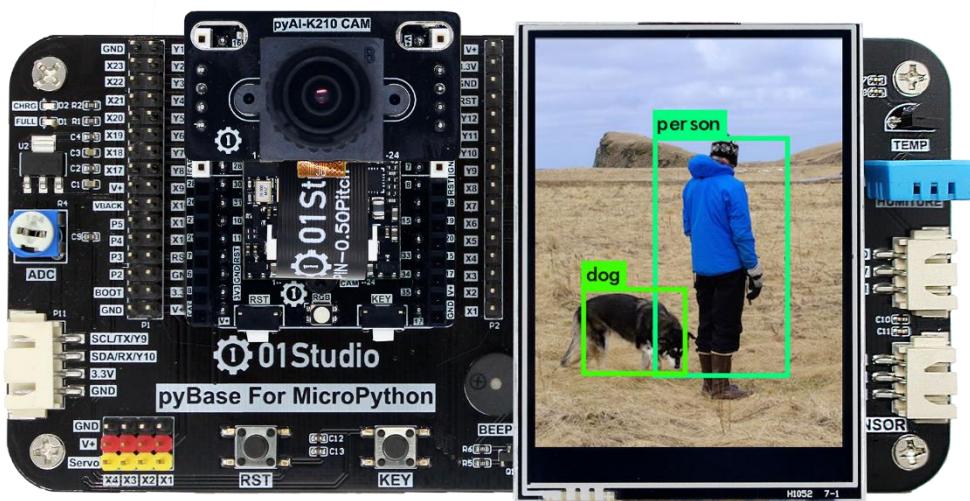


图 5-24 pyAI-K210 开发套件

- **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.blob` 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。

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://wiki.sipeed.com/soft/maixpy/zh/api_reference/machine_vision/image/image.html

表 5-5 `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：

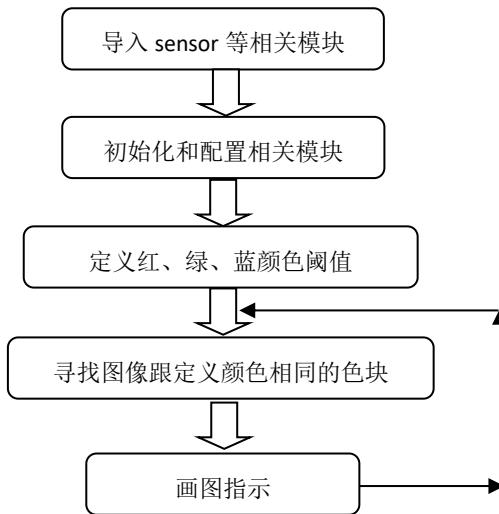


图 5-25 代码编写流程图

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:

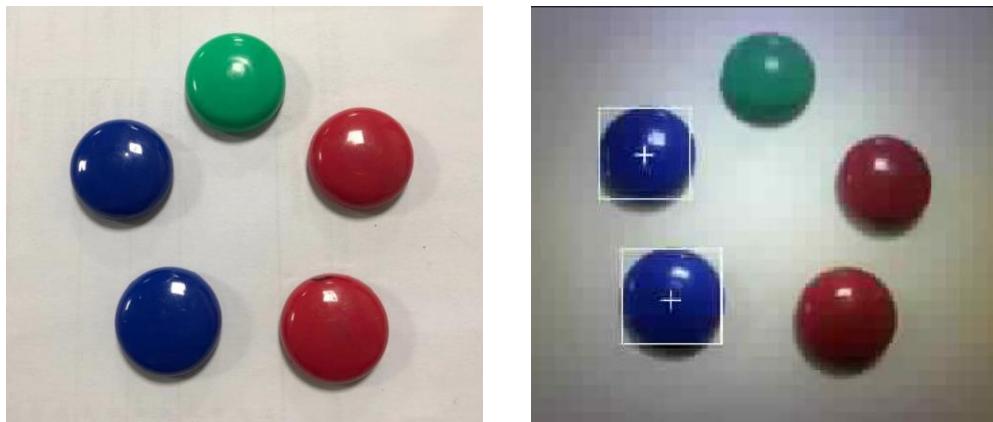


图 5-26 蓝色识别原图 (左) 和实验图片 (右)

● 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。

5.5 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。

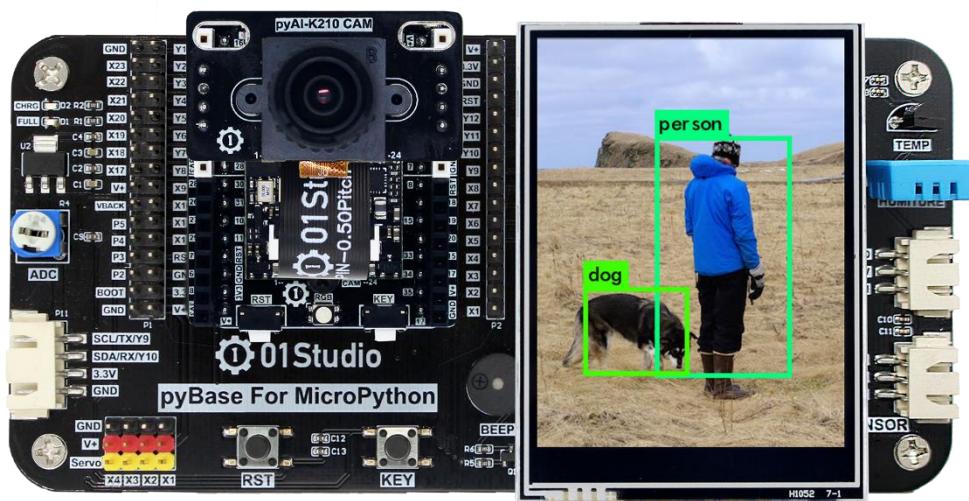


图 5-27 pyAI-K210 开发套件

- **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
<code>image.find_qrcodes([roi])</code>
查找 <code>roi</code> 区域内的所有二维码并返回一个 <code>image.qrcode</code> 的对象列表。
使用方法
以上函数返回 <code>image.qrcode</code> 对象列表。
<code>qrcode.rect()</code>
返回一个矩形元组 <code>(x,y,w,h)</code> ；
<code>qrcode.payload()</code>
返回二维码字符串信息。可以通过索引 [4] 来获得这个值。
<code>qrcode.verison()</code>
返回二维码版本号。
*更多使用说明请阅读官方文档： https://wiki.sipeed.com/soft/maixpy/zh/api_reference/machine_vision/image/image.html#image.find_qrcodes%28%5Broi%5D%29

表 5-6 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：

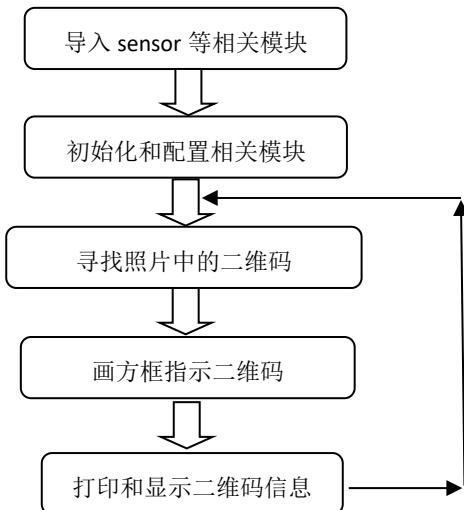


图 5-28 代码编写流程图

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 5-29 成功识别二维码

```
串行终端 | ↗
{
  "w": 320, "h": 240, "type": "rgb565", "size": 153600
}
{
  "w": 320, "h": 240, "type": "rgb565", "size": 153600
}
http://weixin.qq.com/r/nR3j_4nEOdRGrc2Y90iP
4.891304
{
  "w": 320, "h": 240, "type": "rgb565", "size": 153600
}
{
  "w": 320, "h": 240, "type": "rgb565", "size": 153600
}
http://weixin.qq.com/r/nR3j_4nEOdRGrc2Y90iP
4.840271
{
  "w": 320, "h": 240, "type": "rgb565", "size": 153600
}
搜索结果 串行终端
```

图 5-30 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。

5.6 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。

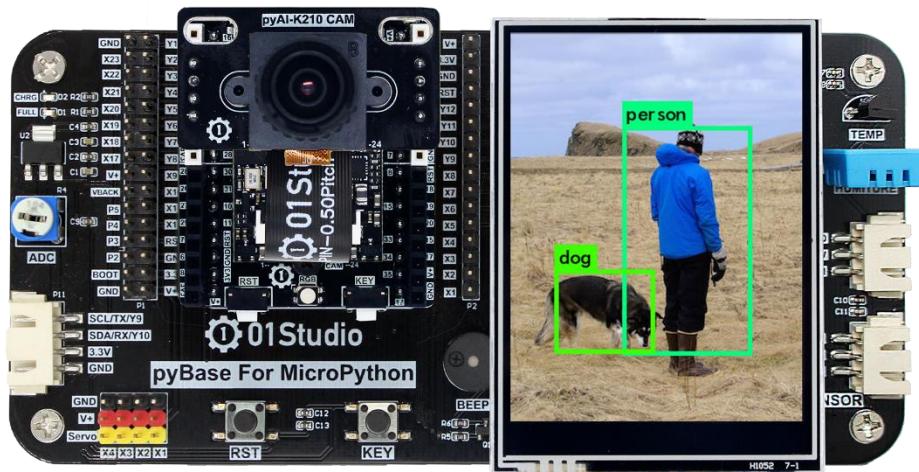


图 5-31 pyAI-K210 开发套件

● 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
<code>import KPU as kpu</code>
常用的 KPU 模块导入方法。
使用方法
<code>kpu.load(offset or file_path)</code>
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”
<code>kpu.init_yolo2(kpu_net,threshold,nms_value,anchor_num,anchor)</code>
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。
<code>kpu.run_yolo2(kpu_net,image)</code>
运行 yolo2 网络； 【kpu_net】从 <code>kpu_load()</code> 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 5-7 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：

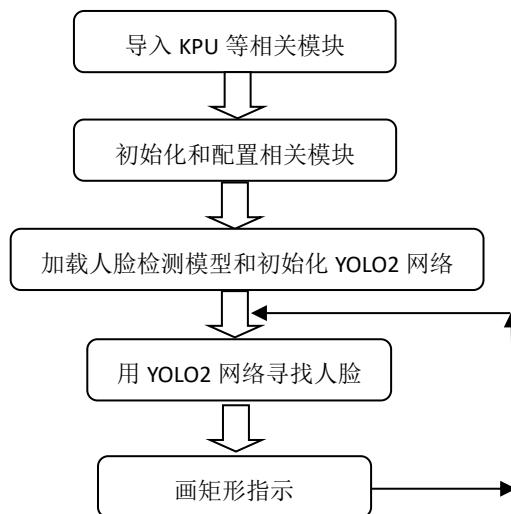


图 5-32 代码编写流程图

参考代码如下：

```
#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 5-33 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。

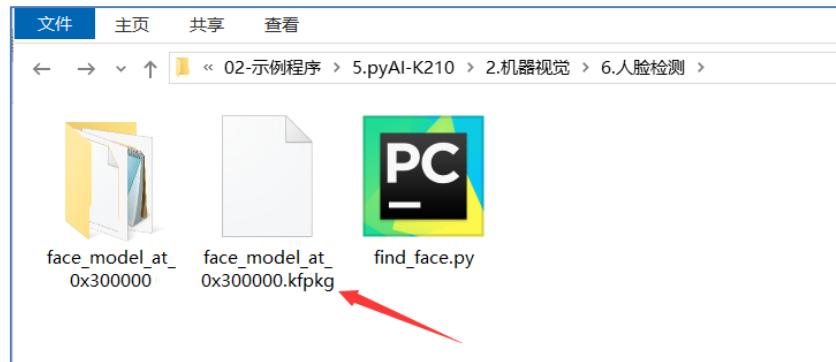


图 5-34

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。。

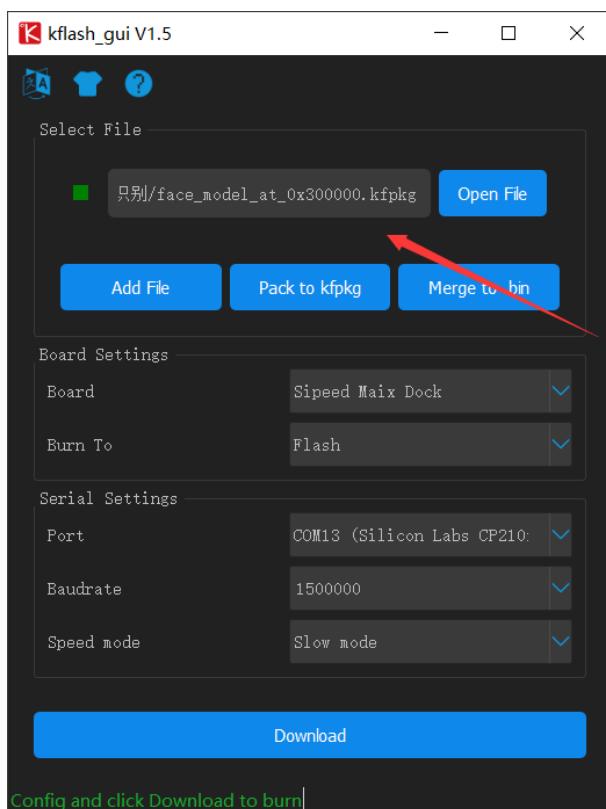


图 5-35

● 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。注意要将开发板竖起来, 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：

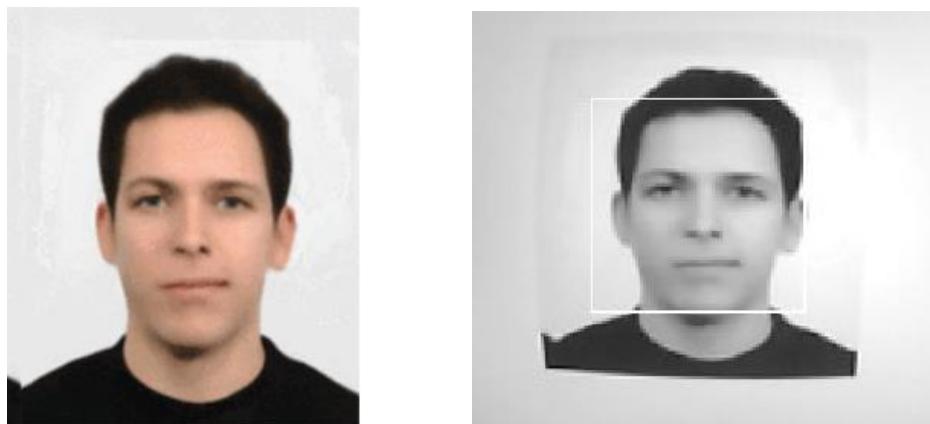


图 5-36 单个人脸检测 原图（左）和实验图片（右）

多个人脸识别：



图 5-37 多个人脸检测原图

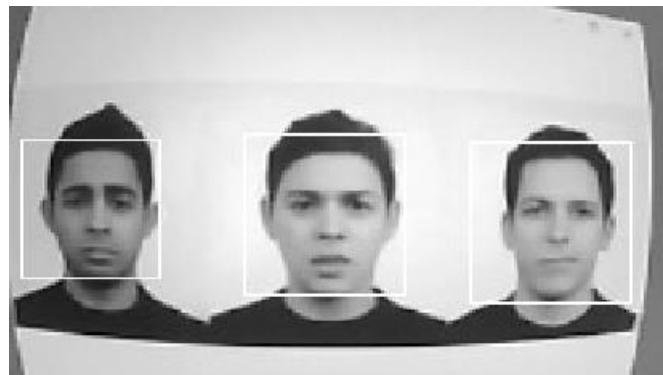


图 5-38 多个人脸检测实验图片

● 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。

5.7 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。

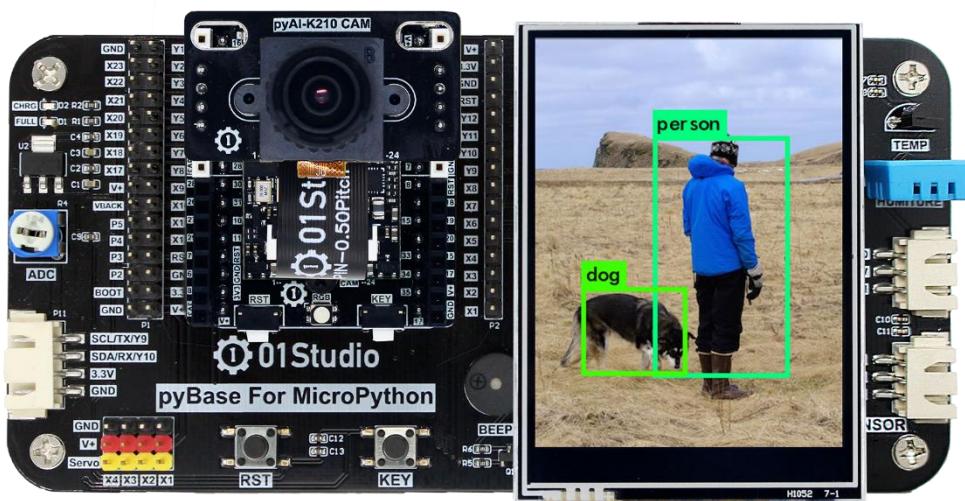


图 5-39 pyAI-K210 开发套件

● 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
<code>import KPU as kpu</code>
常用的 KPU 模块导入方法。
使用方法
<code>kpu.load(offset or file_path)</code>
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”
<code>kpu.init_yolo2(kpu_net,threshold,nms_value,anchor_num,anchor)</code>
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。
<code>kpu.run_yolo2(kpu_net,image)</code>
运行 yolo2 网络： 【kpu_net】 从 <code>kpu_load()</code> 中返回的网络对象； 【image】 从 sensor 中采集到的图像
<code>kpu.deinit(kpu_net)</code>
反初始化。 【kpu_net】 kpu 网络对象；

表 5-8 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：

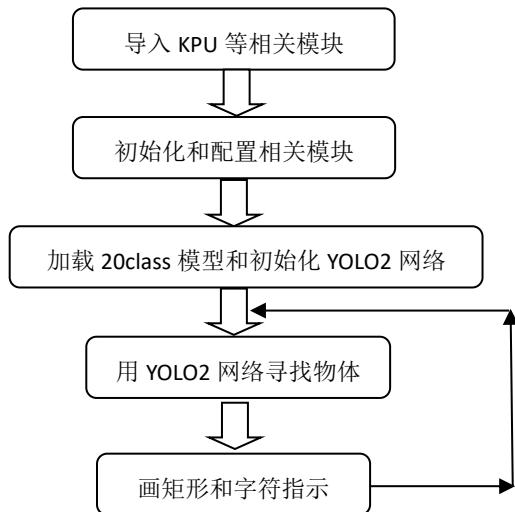


图 5-40 代码编写流程图

参考代码如下：

```

#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 5-41 被识别物体自行车

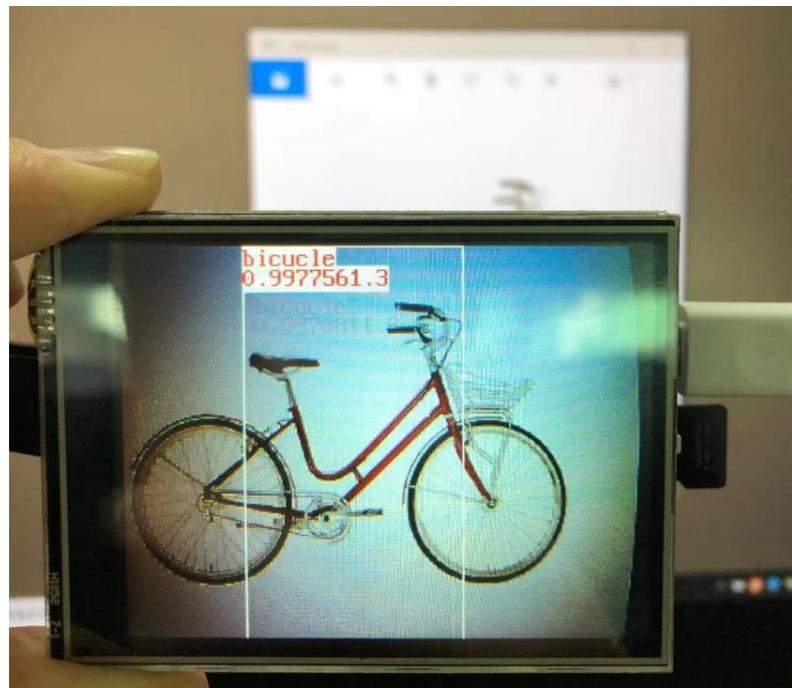


图 5-42 识别结果

● 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。

5.8 在线训练模型

前 2 节学习的人脸识别和物体识别都使用了模型。当我们想自己学习识别自己的东西(比如键盘和鼠标的区分),就可以通过在线训练平台训练自己的模型。在线训练上有完整的教程,这里不再重复。

MaixHub 在线训练链接: <https://maixhub.com/>

当然上面也有很多现成别人训练好的模型可以直接使用,在模型库中选择 nncase 即可以看到适合 K210 使用的模型。

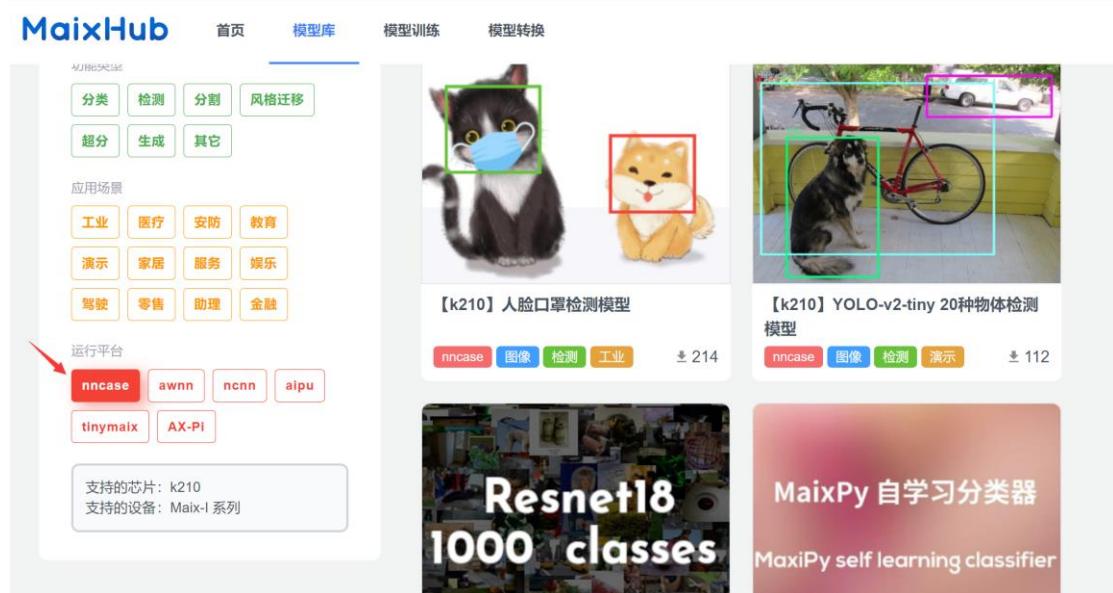


图 5-43

5.9 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。

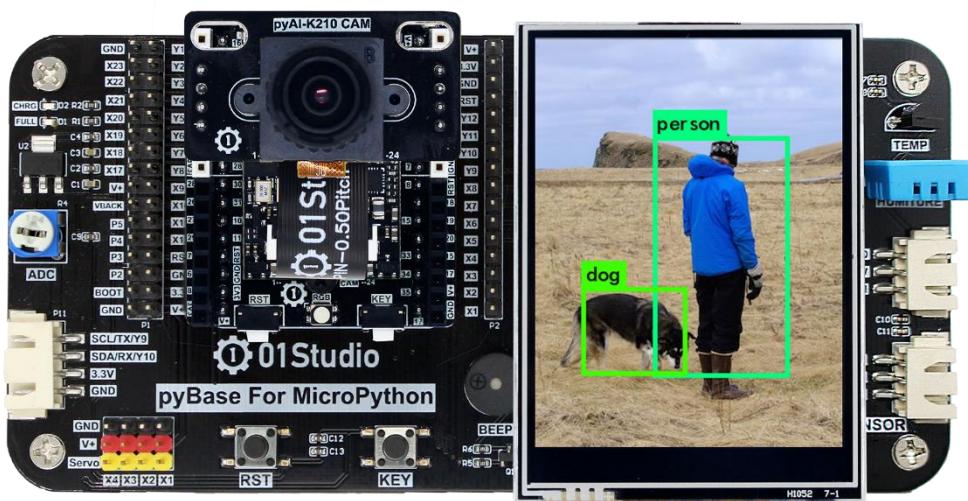


图 5-44 pyAI-K210 开发套件

- **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
<code>img=sensor.snapshot()</code>
通过拍摄创建图像 img
使用方法
<code>image.save(path[, roi[, quality=50]])</code>
保存图片。 <code>path</code> : 保存路径; <code>roi</code> : 指定保存区域(<code>x, y, w, h</code>), 默认全图保存; <code>quality</code> : 仅针对 JPEG 格式的质量控制, 有效值为 0-100。

表 5-9 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:

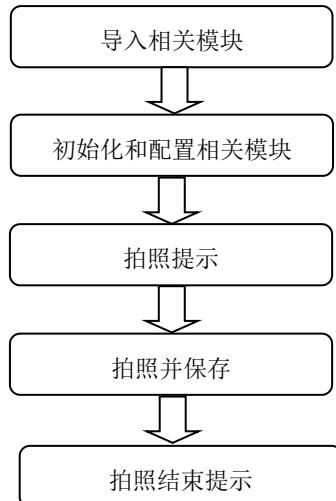


图 5-45 代码编写流程

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) #1 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 5-46 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。

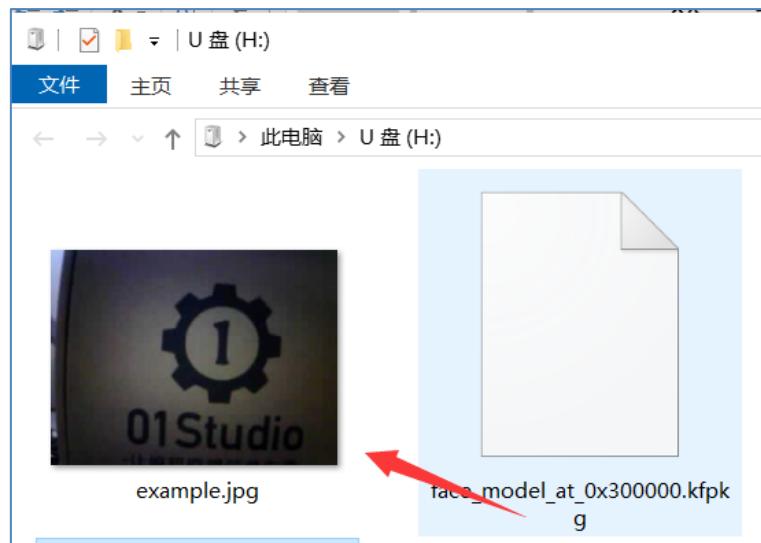


图 5-47 刚刚拍摄到的图片文件。

● 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。

5.10 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。

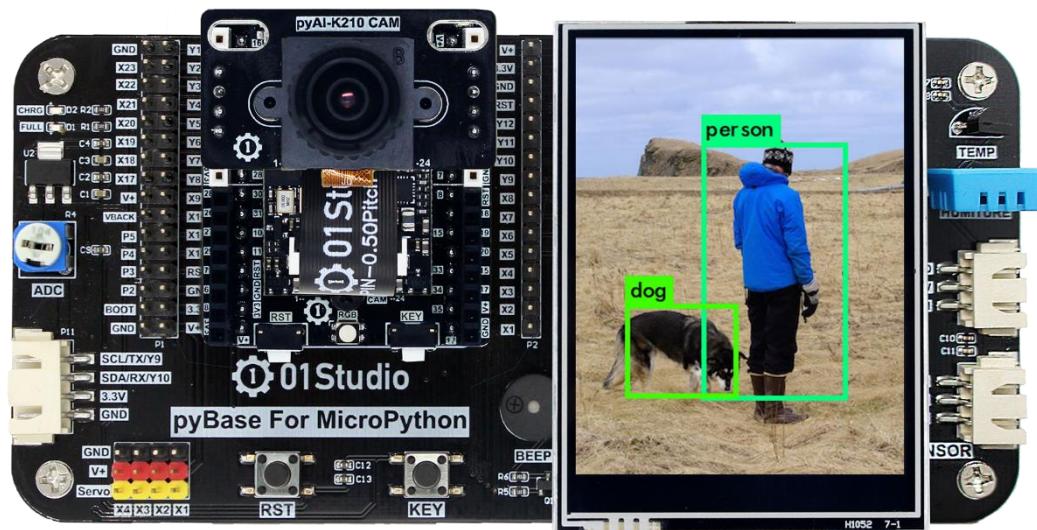


图 5-48 pyAI-K210 开发套件

- **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 5-10 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：

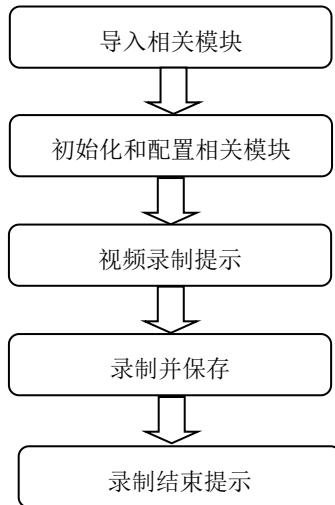


图 5-49 代码编写流程

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 5-50 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 5-51 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6章 机器听觉

6.1 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。

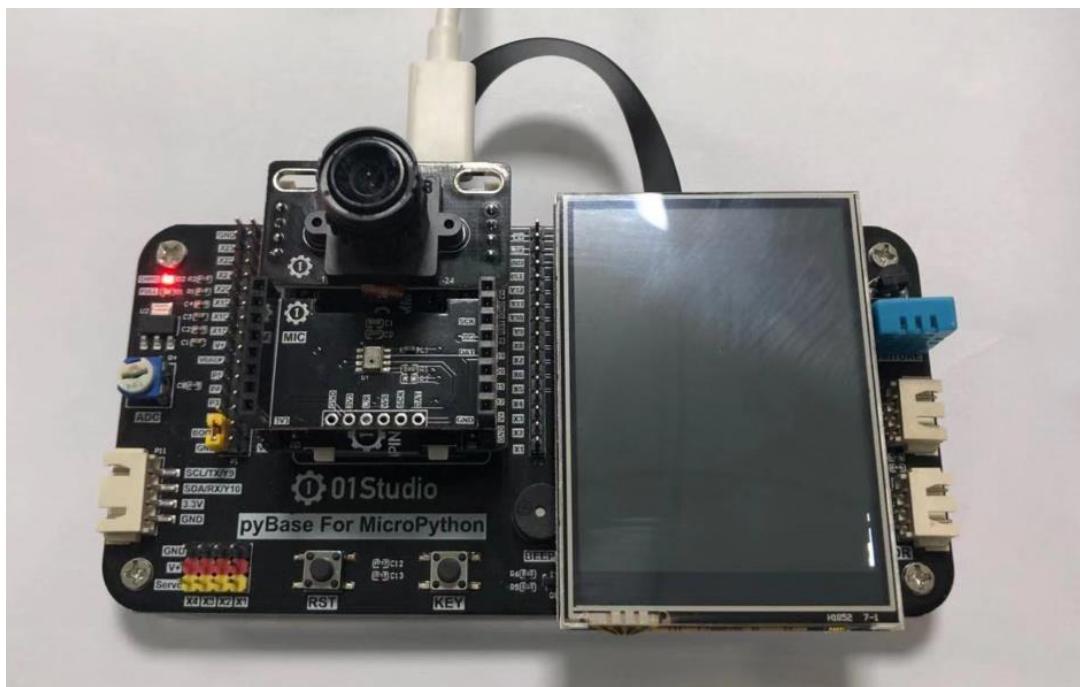


图 6-1 pyAI-K210 开发套件

- **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叠加，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。）

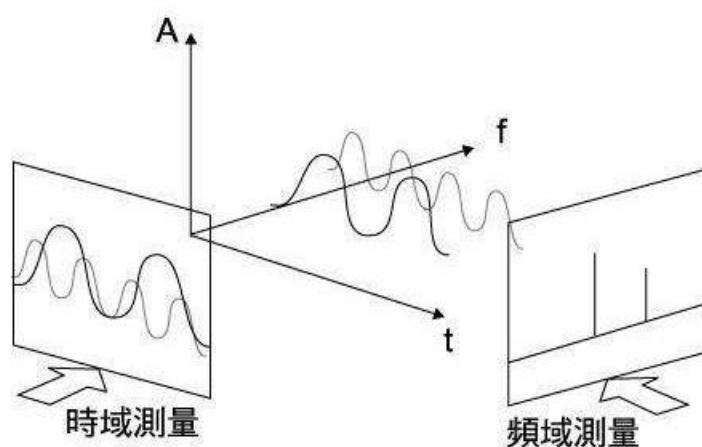


图 6-2 时域与频域关系

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：

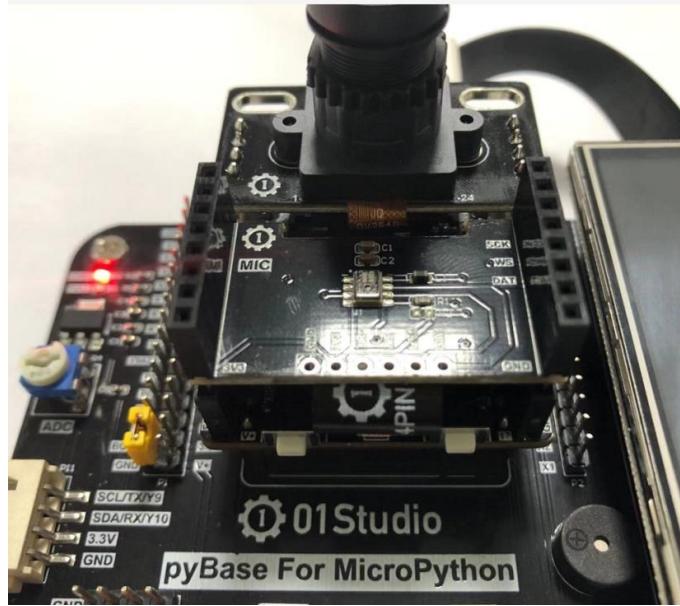


图 6-3 数字麦克风模块连接方式

```
#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 6-1 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， <code>bytearray</code> 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 <code>FFT_res</code> ：返回计算后的频域数据，以 <code>list</code> 类型呈现，该列表有 <code>points</code> 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 <code>res</code> ：返回一个列表，该列表存放的进行运算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 6-2 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：

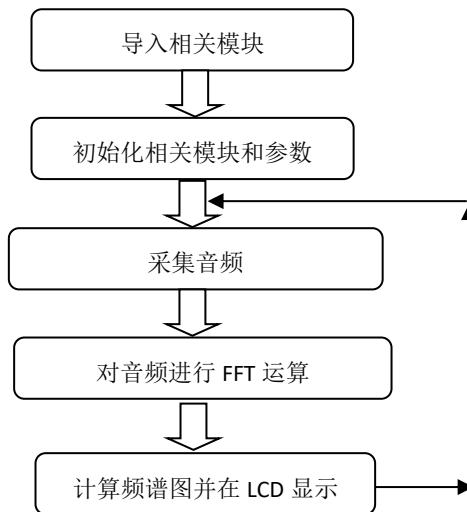


图 6-4 代码编写流程图

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。

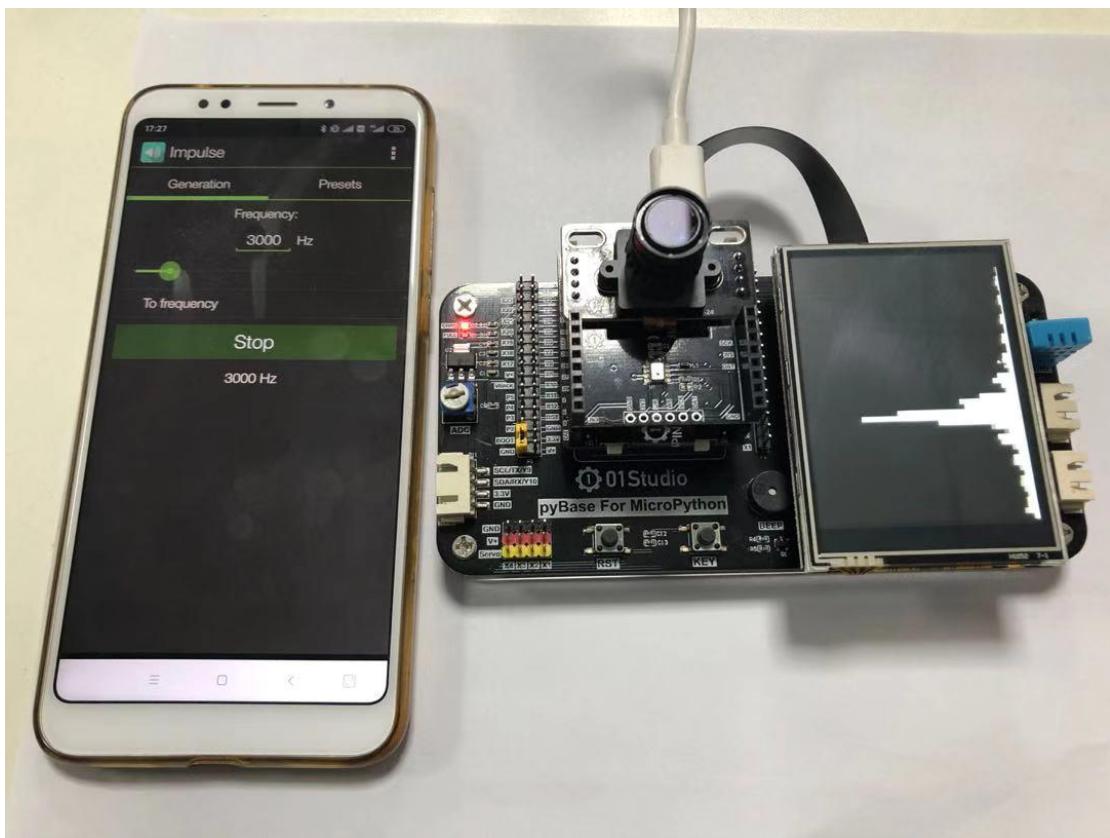


图 6-5 FFT 频谱实验

● 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。

6.2 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。

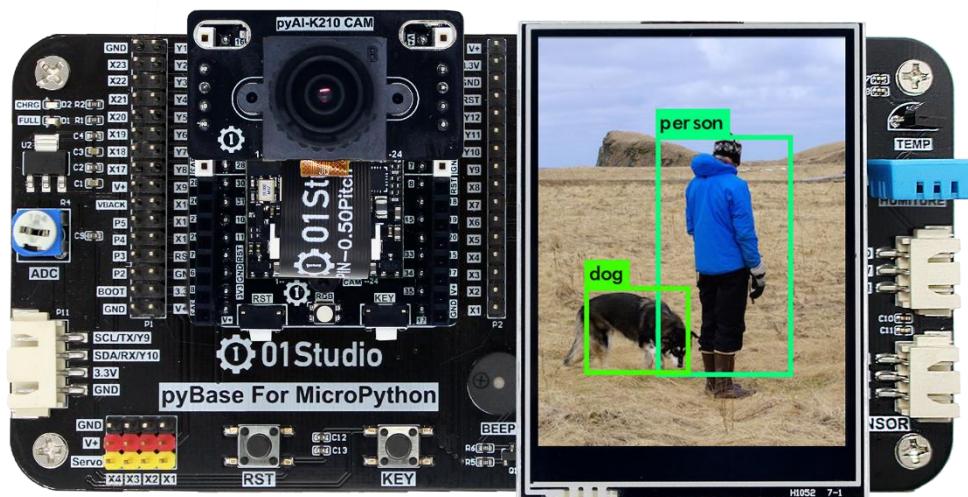


图 6-6 pyAI-K210 开发套件

- **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。

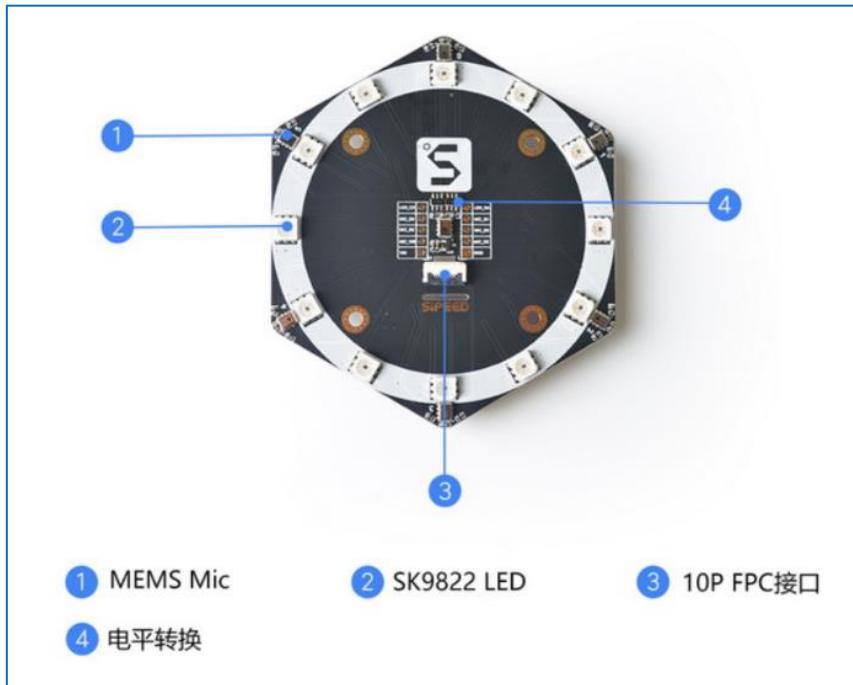


图 6-7 麦克风阵列模块

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）。

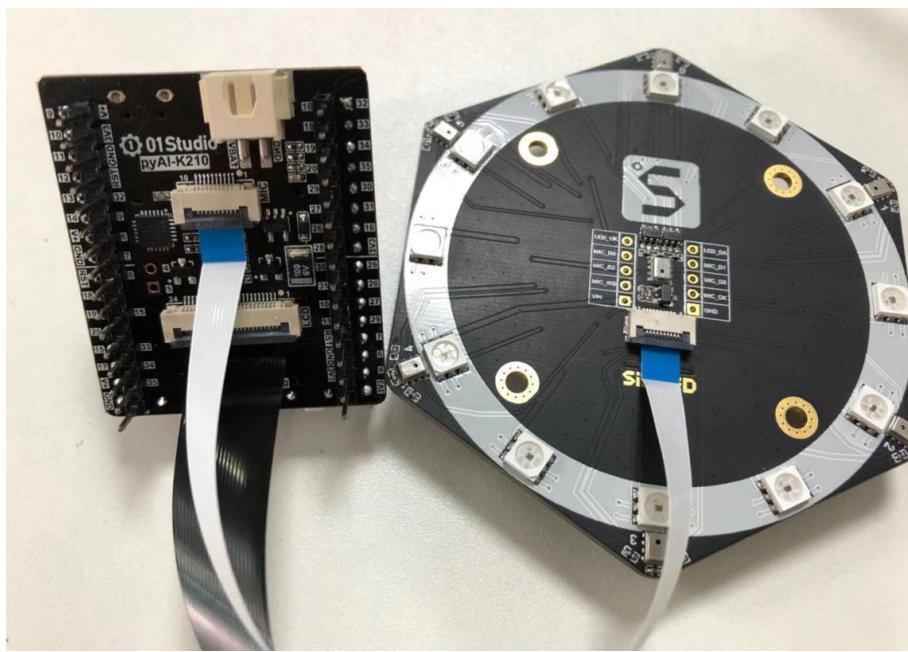


图 6-8 连接方式

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
<pre>from Maix import MIC_ARRAY as mic</pre>
导入 MIC_ARRAY 模块；
使用方法
<pre>mic.getmap()</pre>
返回声源黑白位图；尺寸 16*16。
<pre>mic.get_dir(img)</pre>
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。
<pre>mic.set_led(direction,color)</pre>
从计算的声源方向设置点亮对应的 LED。
<pre>mic.deinit ()</pre>
反初始化 mic。

表 6-3 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：

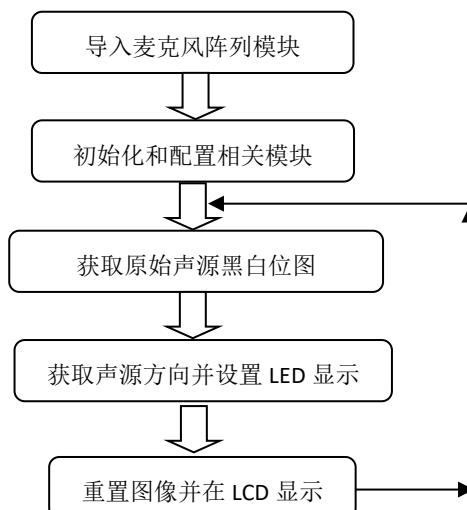


图 6-9 代码编写流程图

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。

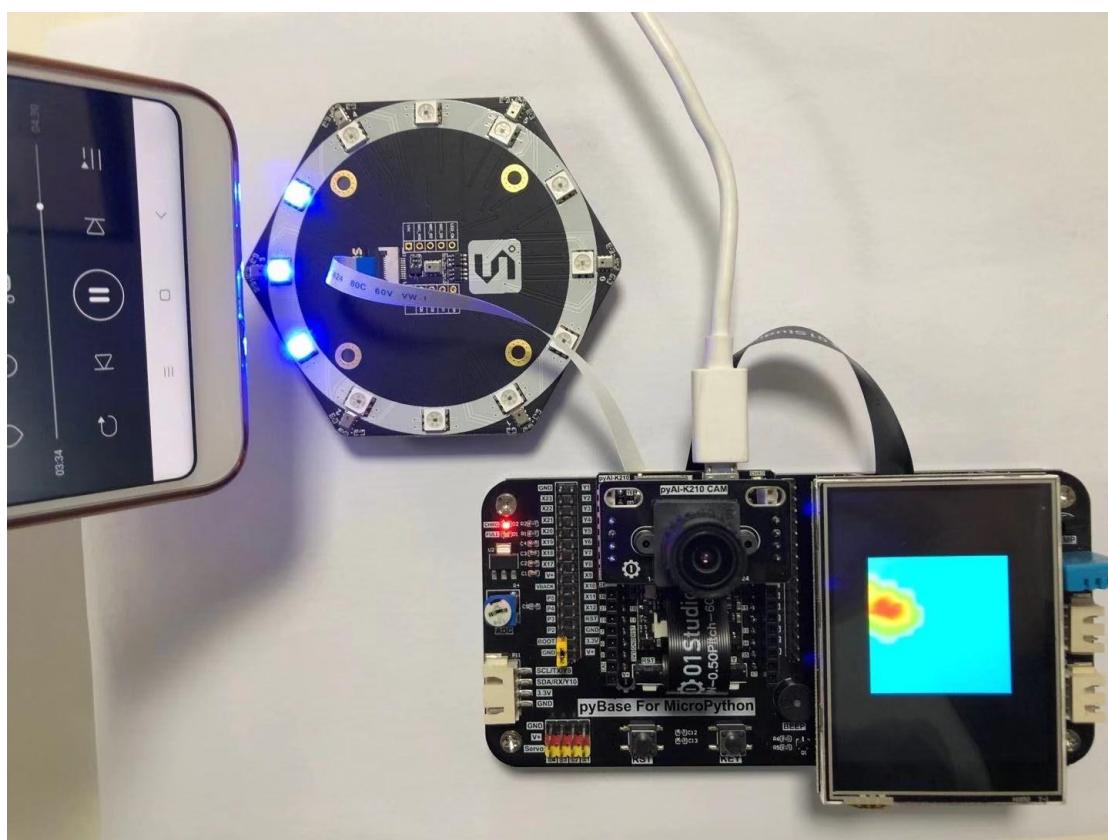


图 6-10 声源定位实验

● 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7章 拓展模块

7.1 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。

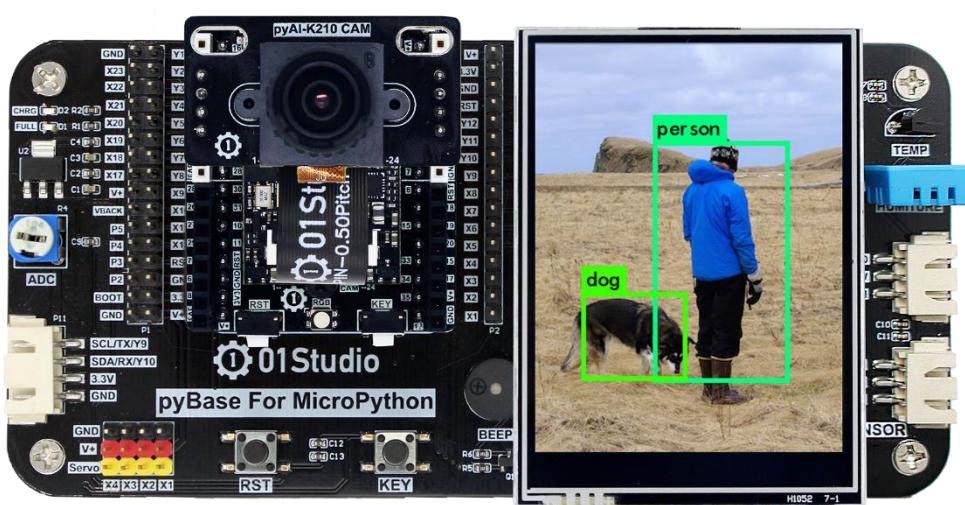


图 7-1 pyAI-K210 开发套件

● 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
<pre>import touchscreen as ts</pre>
导入 touchscreen 模块；
使用方法
<pre>ts.init(i2c=None,cal=None)</pre>
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。
<pre>ts.calibrate()</pre>
触摸校准。返回一个 7 个整型值的元组。
<pre>ts.read()</pre>
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 7-1 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：

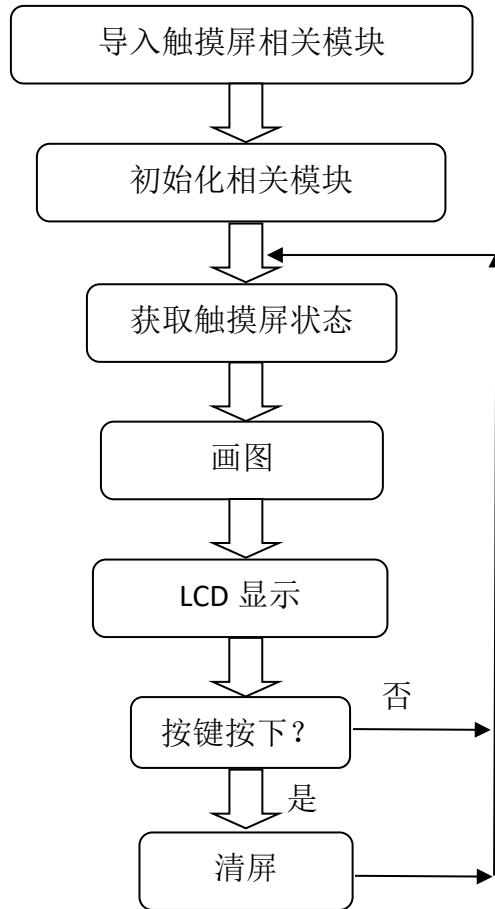


图 7-2 代码编程流程

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。

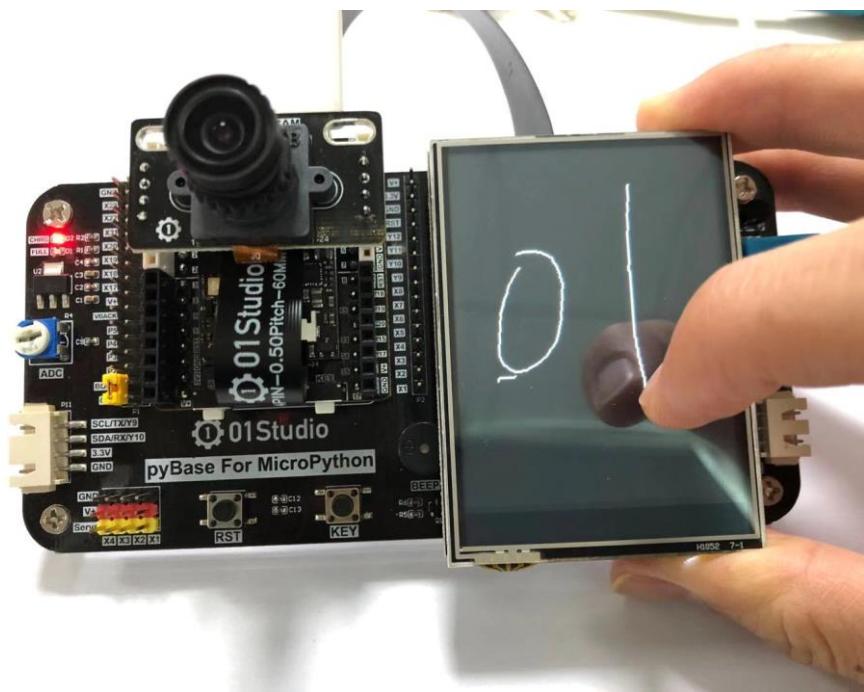


图 7-3 触摸屏实验

● 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。

7.2 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。

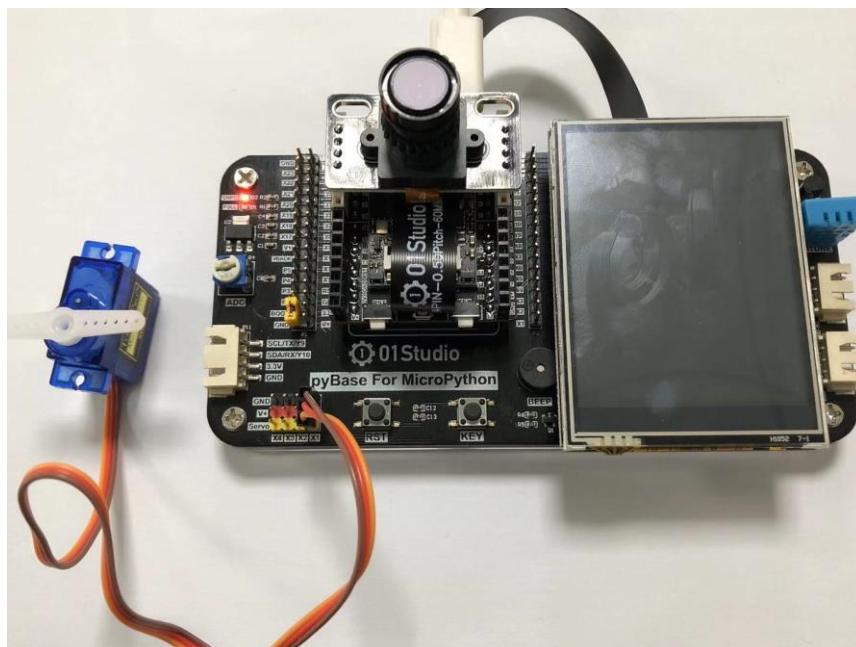


图 7-4 舵机接线

通常情况下：黑色表示 GND，红色表示 VCC，橙色表示信号线。

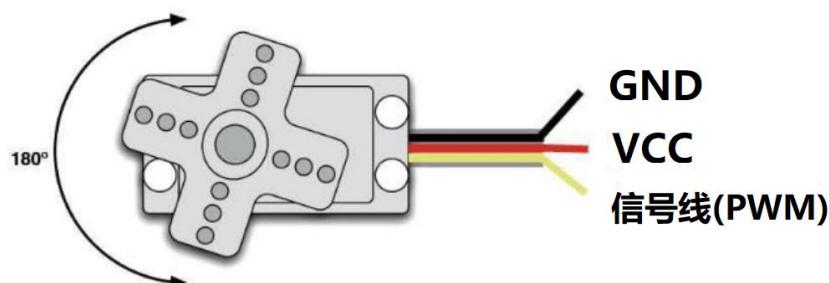


图 7-5 舵机示意图

- **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。

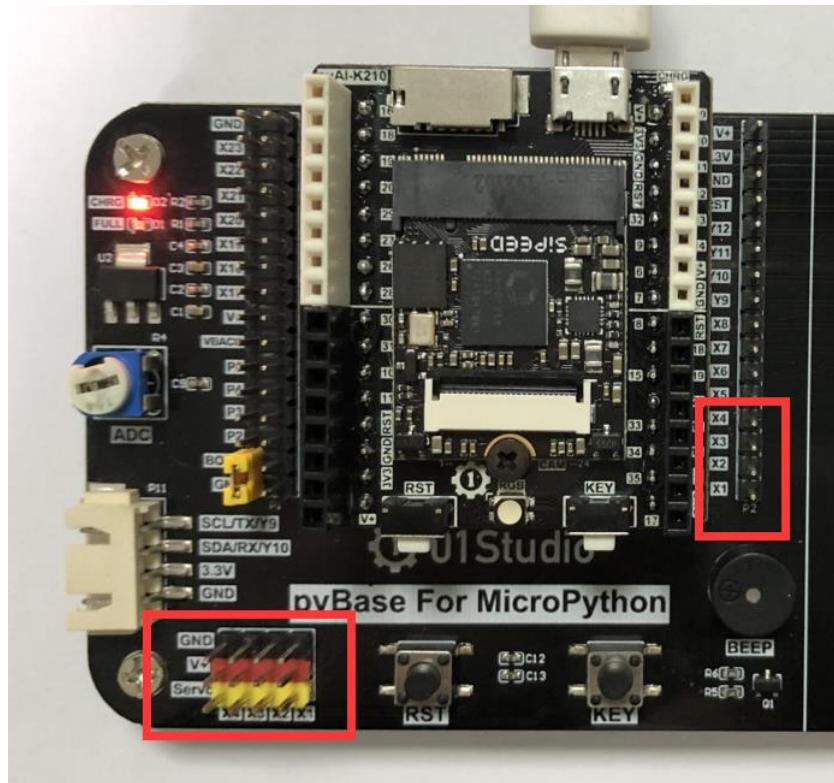


图 7-6 pyBase 的 4 路舵机接口

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° , 示意图如下:

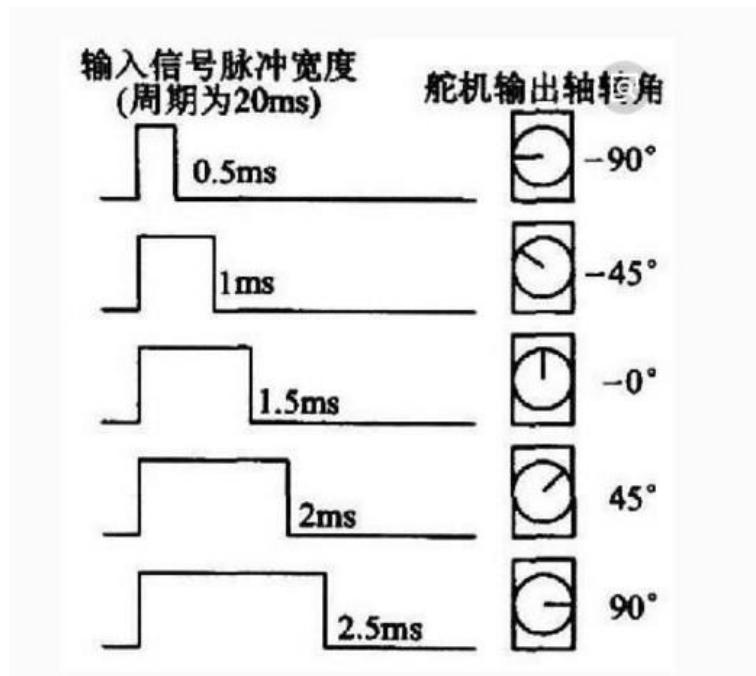


图 7-7 PWM 与角度关系

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：

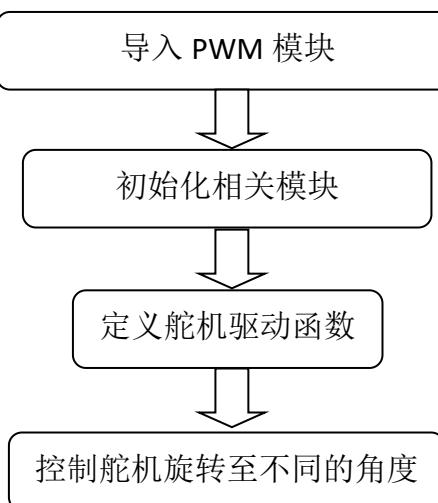


图 7-8 编程流程图

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。

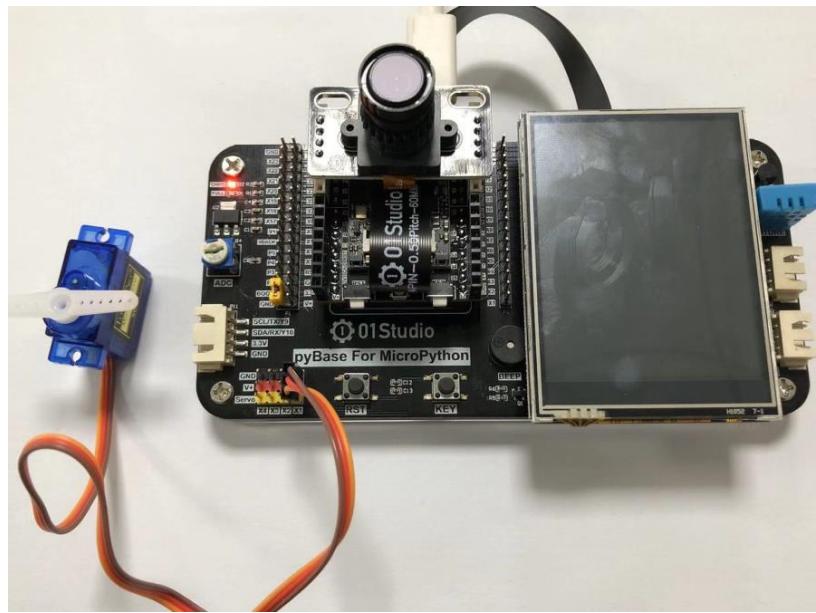


图 7-9 舵机角度旋转至不同角度

● 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。

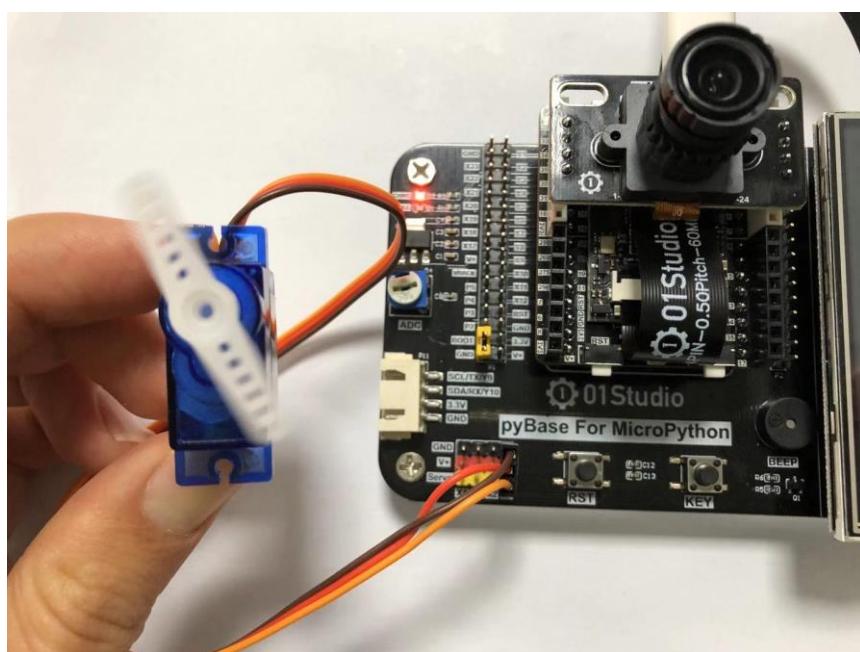


图 7-10 360° 连续旋转舵机

● 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。

7.3 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。

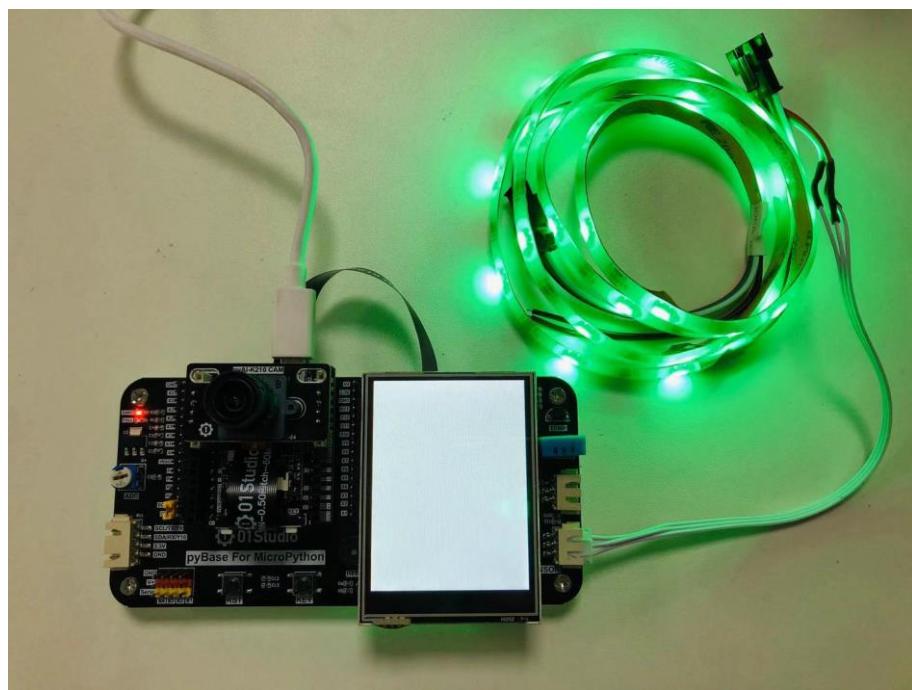


图 7-11 RGB 灯带接线方式

- **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。

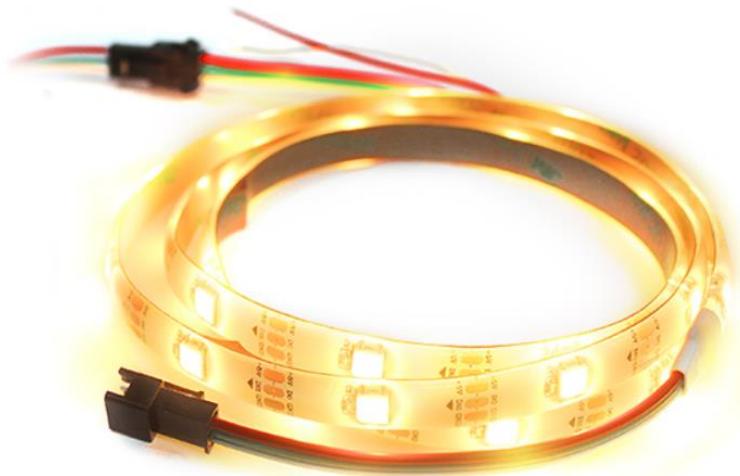


图 7-12 RGB 灯带

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。

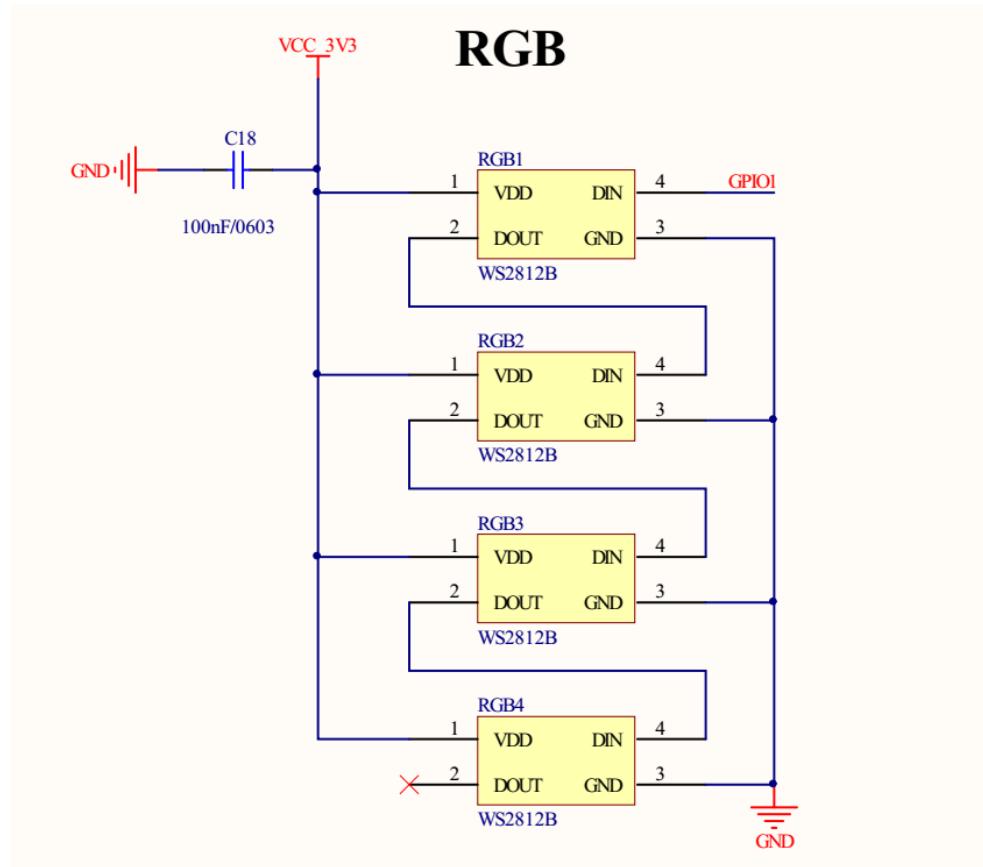


图 7-13 灯珠串联接线方式

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
<code>np=ws2812(pin, num)</code>
构建灯带对象。 【pin】灯带控制引脚； 【num】:灯珠数量；
使用方法
<code>np.set_led=(i,color)</code>
设置第 i 个灯珠参数。 【i】第 i 个灯珠； 【color】灯珠颜色，RGB 表示。
<code>np.display()</code>
往灯带写入设置的数据。

表 7-2 ws2812-RGB 灯带对象

代码编程流程图如下：

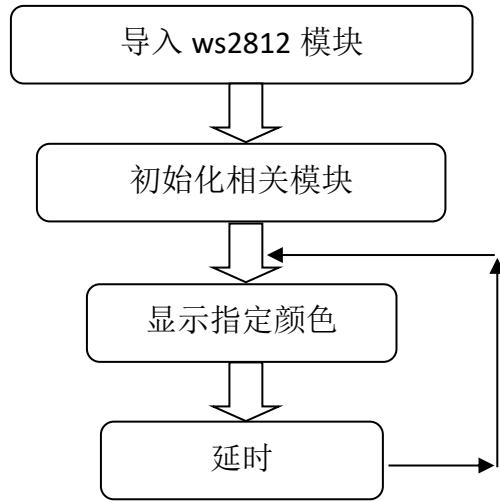


图 7-14 编程流程图

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

#define 红、绿、蓝三种颜色
RED=(255,0,0)
GREEN=(0,255,0)
BLUE=(0,0,255)

#define 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。

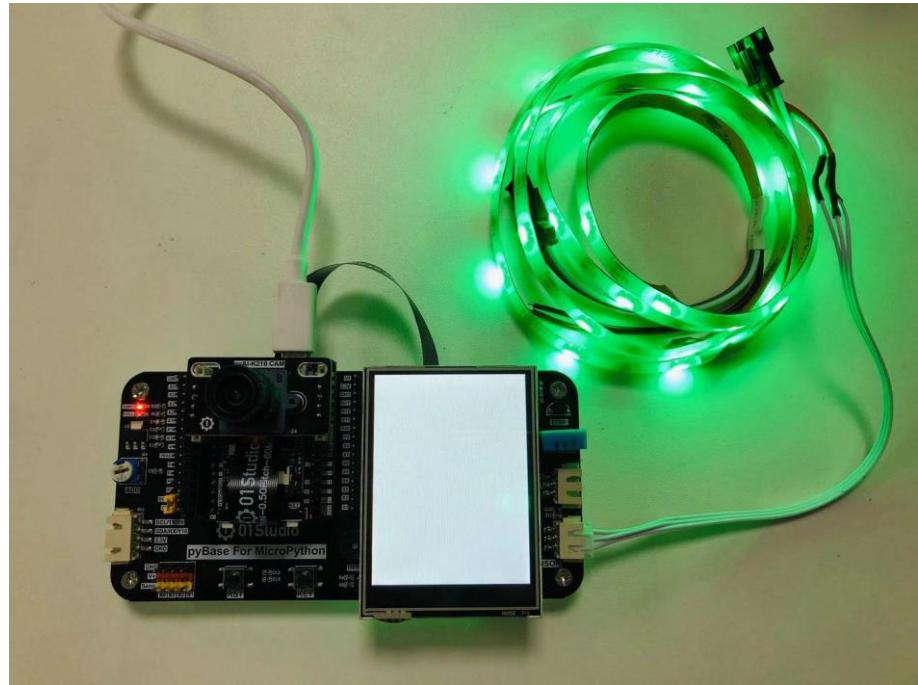


图 7-15 实验结果

● 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。

7.4 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：

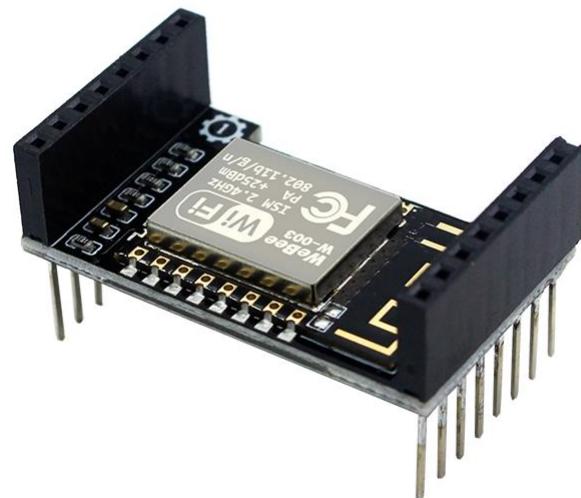


图 7-16 K210 WiFi 模块

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 7-3 K210 WiFi 模块功能参数

7.4.1 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。

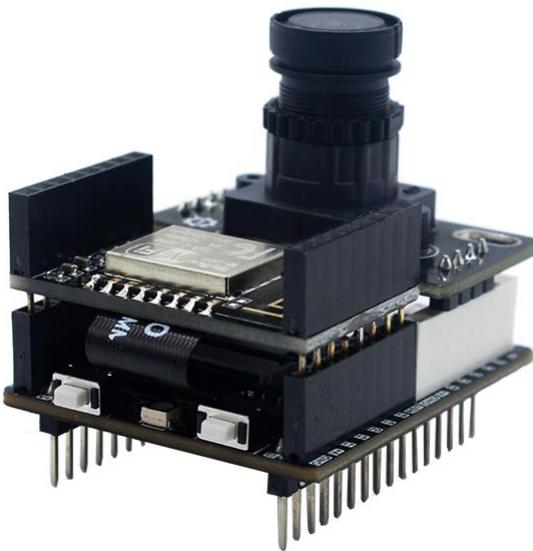


图 7-17 pyAI-K210+WiFi 模块

- **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.ESP8266(uart)
构建 WIFI 模块连接对象。ESP8266 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 True: 已连接; False: 未连接。
wlan.connect(ssid, password)
WIFI 连接。 【ssid】账号; 【password】密码;
wlan.ifconfig()
查看 wifi 连接信息。
wlan.disconnect()
断开连接。
wlan.enable_ap(ssid, key, ch1=5, ecn=3)
AP 热点使能; 【ssid】热点名称; 【key】密码; 【ch1】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 7-4 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：

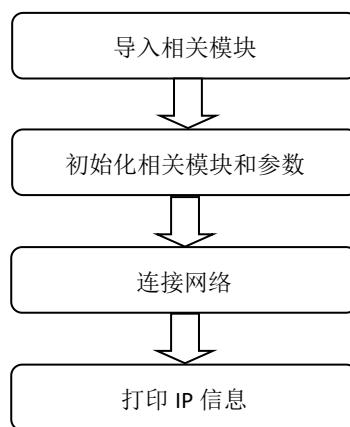


图 7-18 代码编写流程图

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 7-19

● **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。

7.4.2 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 7-20 pyAI-K210 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：

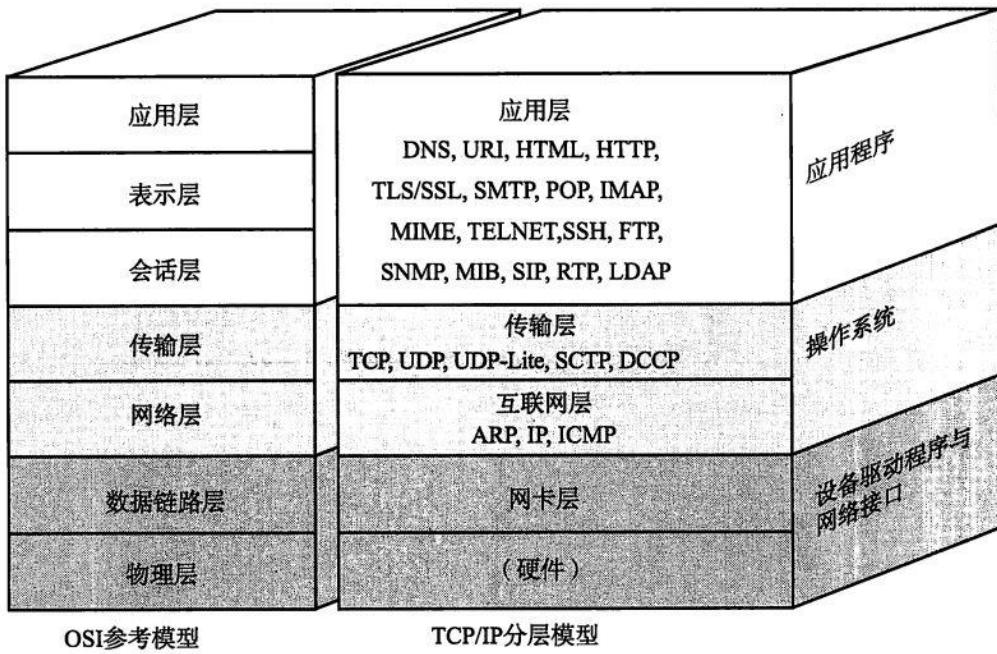


图 7-21 网络层级模型

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。

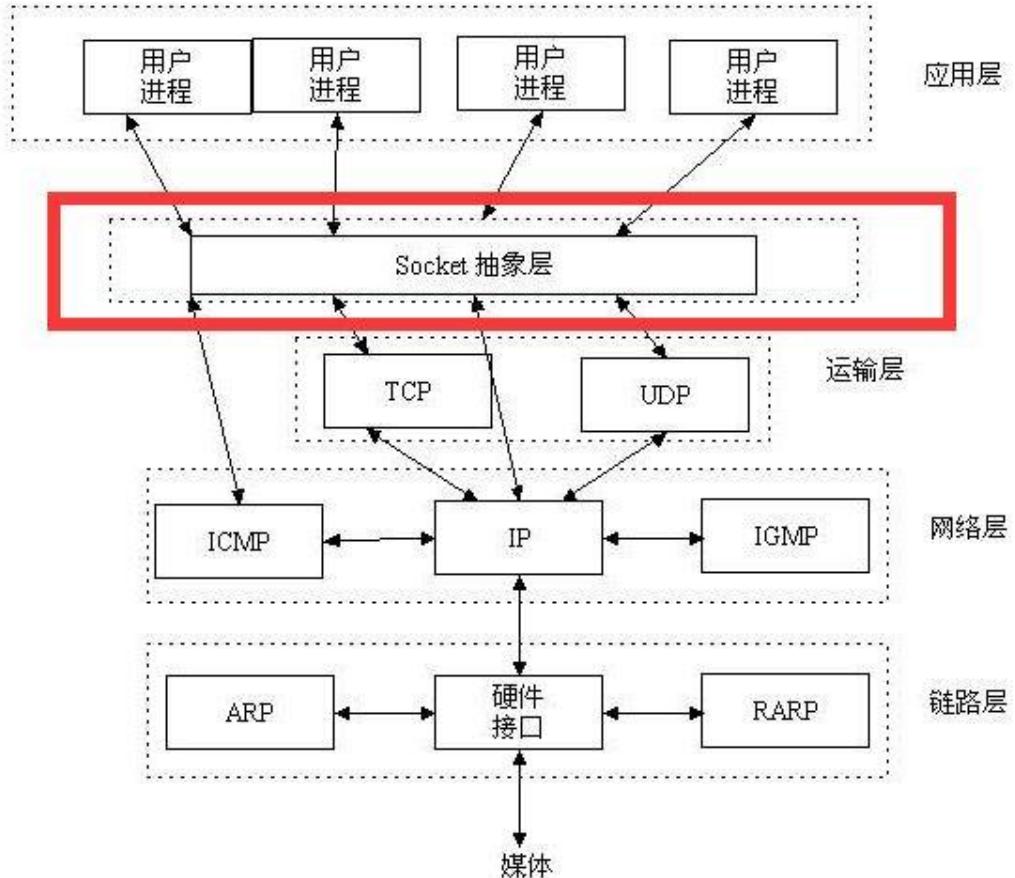


图 7-22 Socket 抽象层

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：

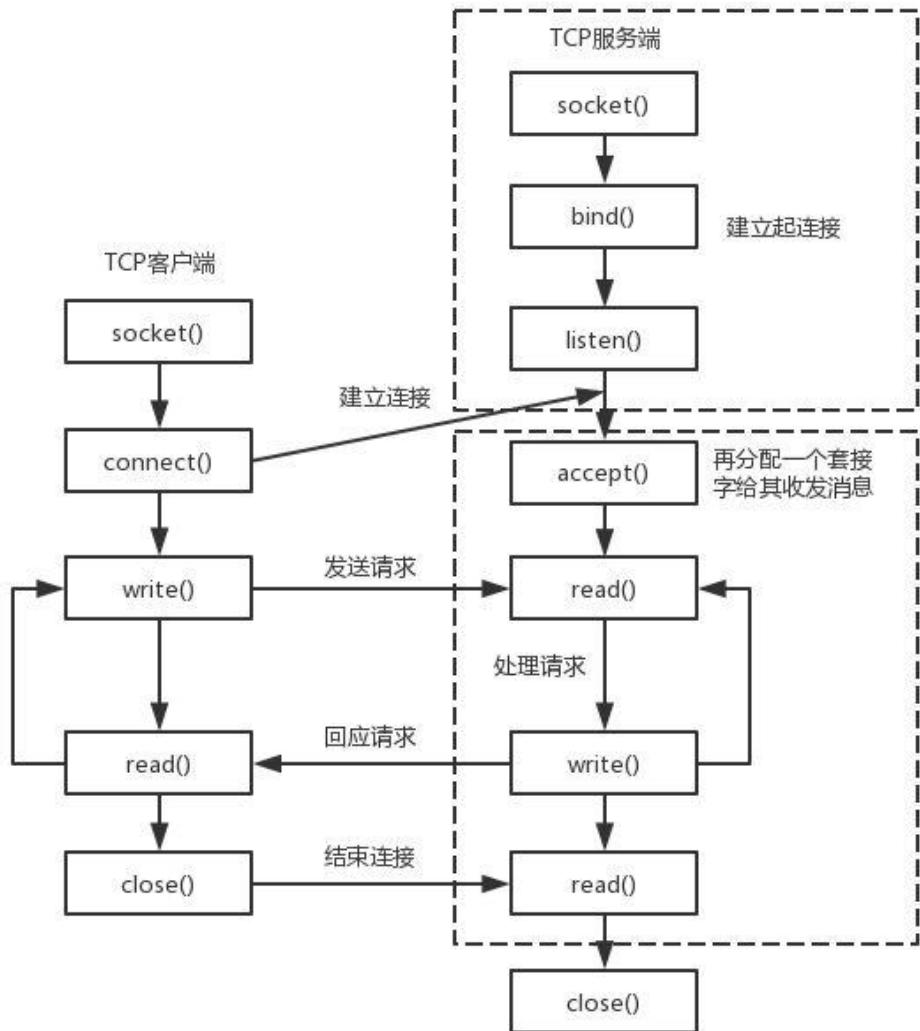


图 7-23 Socket 通信过程

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 和端口号即可建立连接。
 (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
<pre>s=usocket.socket(af=AF_INET, type=SOCK_STREAM,proto=IPPROTO_TCP)</pre>
构建 usocket 对象。 af: AF_INET→IPV4, AF_INET6 → IPV6; type: SOCK_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
<pre>addr=usocket.getaddrinfo('www.01studio.org', 80)[0][-1]</pre>
获取 Socket 通信格式地址。返回: ('47.91.208.161', 80)
<pre>s.connect(address)</pre>
创建连接。address: 地址格式为 IP+端口。例: ('192.168.1.115', 10000)
<pre>s.send(bytes)</pre>
发送。bytes: 发送内容格式为字节
<pre>s.recv(bufsize)</pre>
接收数据。bufsize: 单次最大接收字节个数。
<pre>s.bind(address)</pre>
绑定, 用于服务器角色
<pre>s.listen([backlog])</pre>
监听, 用于服务器角色。backlog: 允许连接个数, 必须大于 0。
<pre>s.accept()</pre>
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 7-5 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：

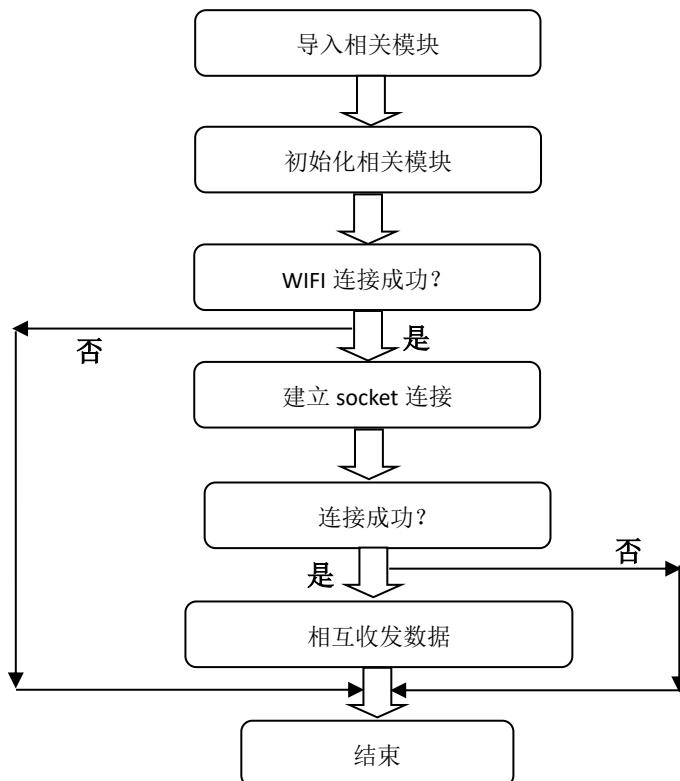


图 7-24 代码编写流程图

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.ESP8265(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("recv:", 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！

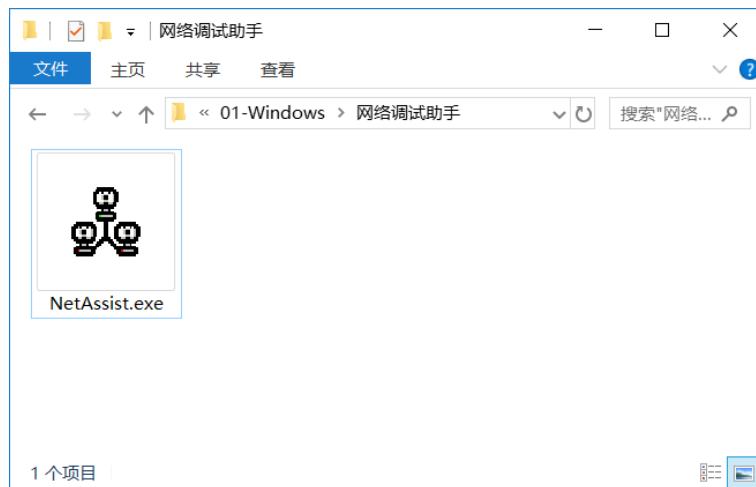


图 7-25 网络调试助手

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 7-26 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：

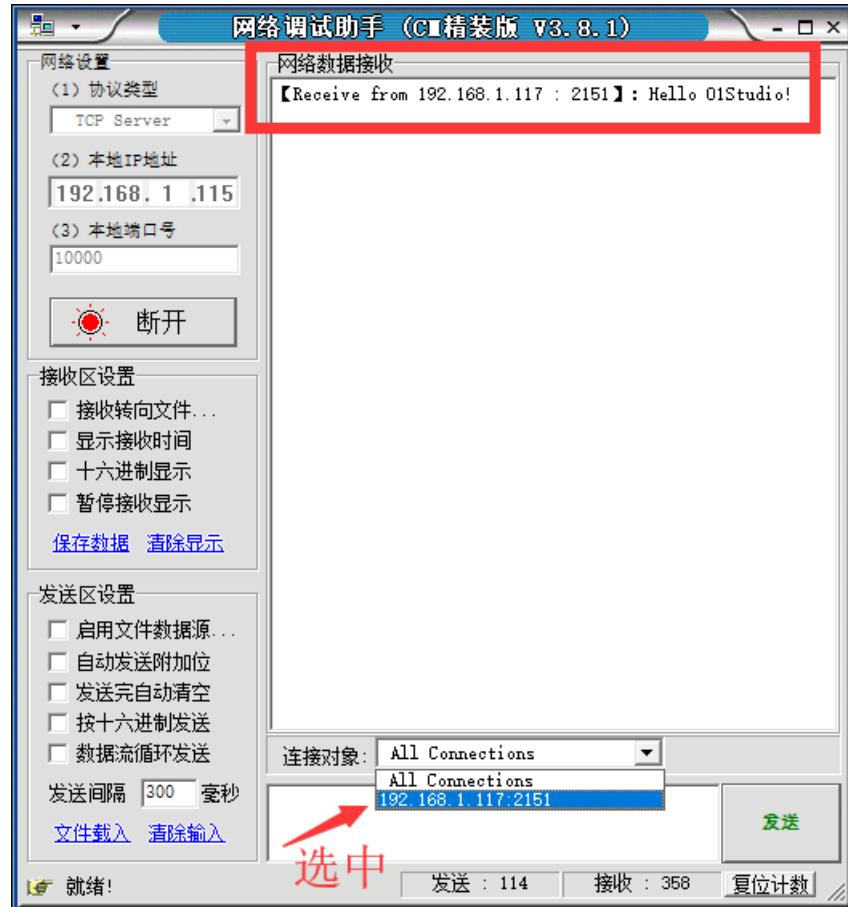


图 7-27 选中连接对象

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。

```
串行终端 | < >
Trying to connect... (may take a while)...
>>> 1
1
1
b'AT+UART_CUR=921600,8,1,0,0\r\n\r\nOK\r\n'
b'AT\r\n\r\nOK\r\n'
Trying to connect... (may take a while)...
('192.168.1.117', '255.255.255.0', '192.168.1.1', '0', '0', '28:6c:07:15
recv: 23 b'<a href="http://www.01studio.org">http://www.01studio.org'
```

图 7-28

- **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。

7.4.3 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 7-29 `pyAI-K210`

- **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

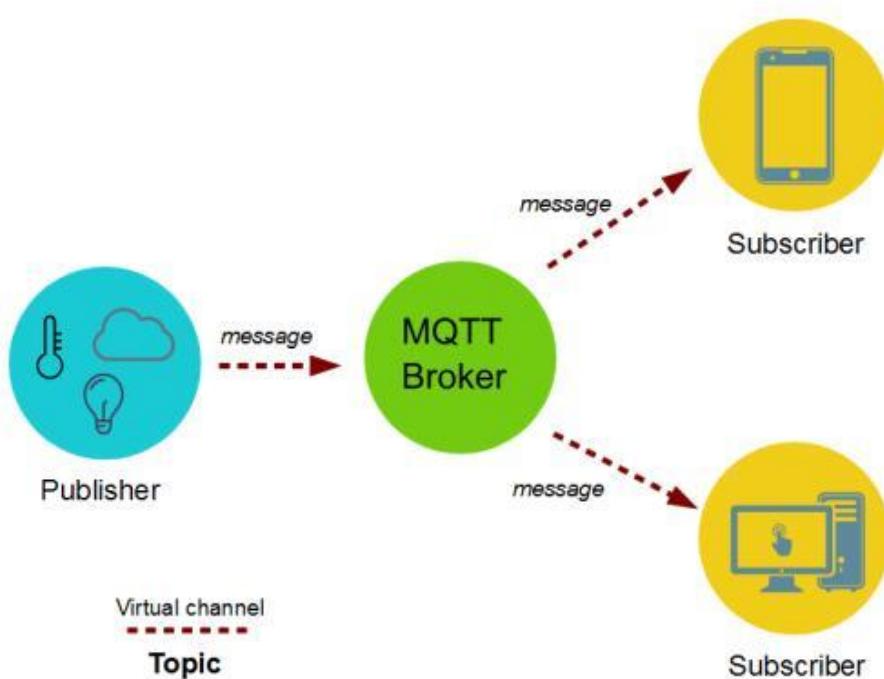


图 7-30 MQTT 通信流程

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：

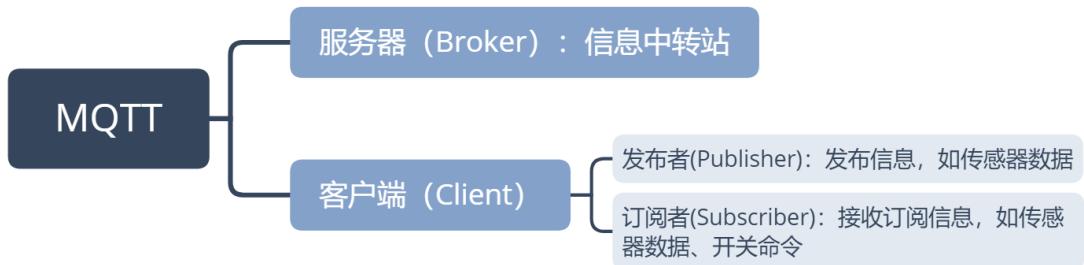


图 7-31 MQTT 角色说明

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 7-32 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
<code>client=mqtt.MQTTClient(client_id, server, port)</code>
构建 MQTT 客户端对象。 <code>client_id</code> : 客户端 ID，具有唯一性； <code>server</code> : 服务器地址，可以是 IP 或者网址； <code>port</code> : 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
<code>client.connect()</code>

连接到服务器。
<code>client.publish(TOPIC,message)</code>
发布。TOPIC: 主题编号; message: 信息内容, 例: 'Hello 01Studio!'
<code>client.subscribe(TOPIC)</code>
订阅。TOPIC: 主题编号。
<code>client.set_callback(callback)</code>
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。
<code>client.check_msg()</code>
检查订阅信息。如收到信息就执行设置过的回调函数 callback。

表 7-6 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:

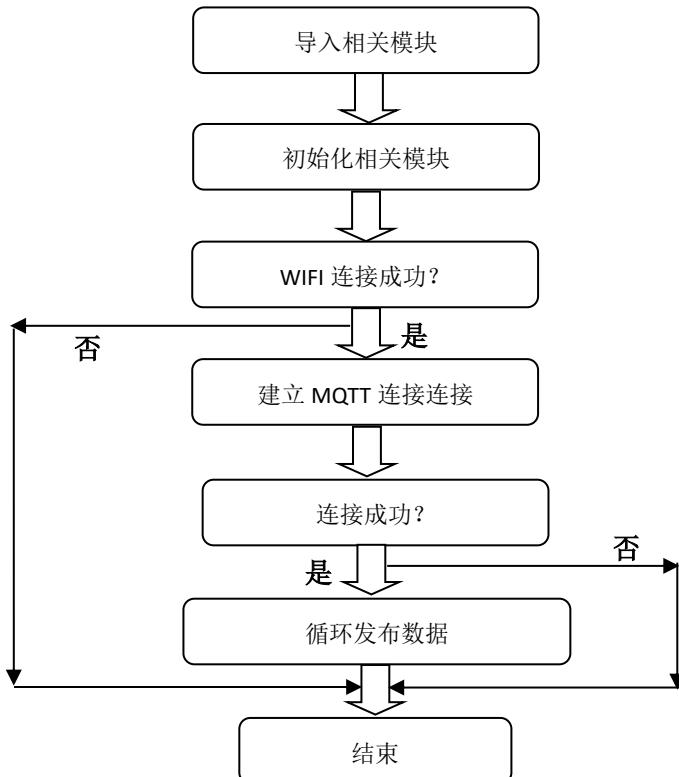


图 7-33 发布者代码编写流程

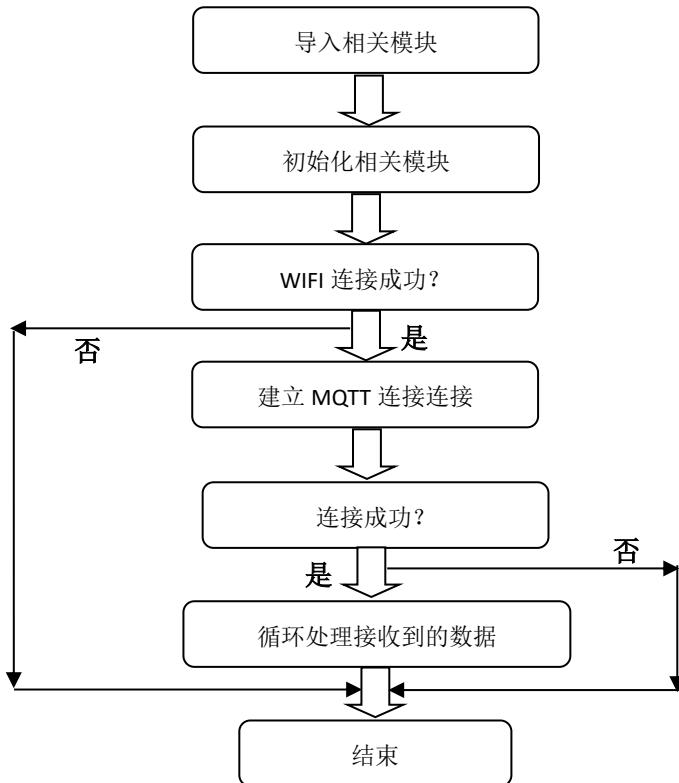


图 7-34 订阅者代码编写流程

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.ESP8265(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：

```

...
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())

# important! baudrate too low or read_buf_len too small will loose data
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.ESP825(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())

#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):
    try:
        client.check_msg()
    except OSError:
        pass

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

#定时器 0 初始化, 周期 300ms, 执行 MQTT 通信接收任务
tim = Timer(Timer.TIMER0, Timer.CHANNEL0, mode=Timer.MODE_PERIODIC,
            period=300, callback=MQTT_Rev)

while True:
    pass

```

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。

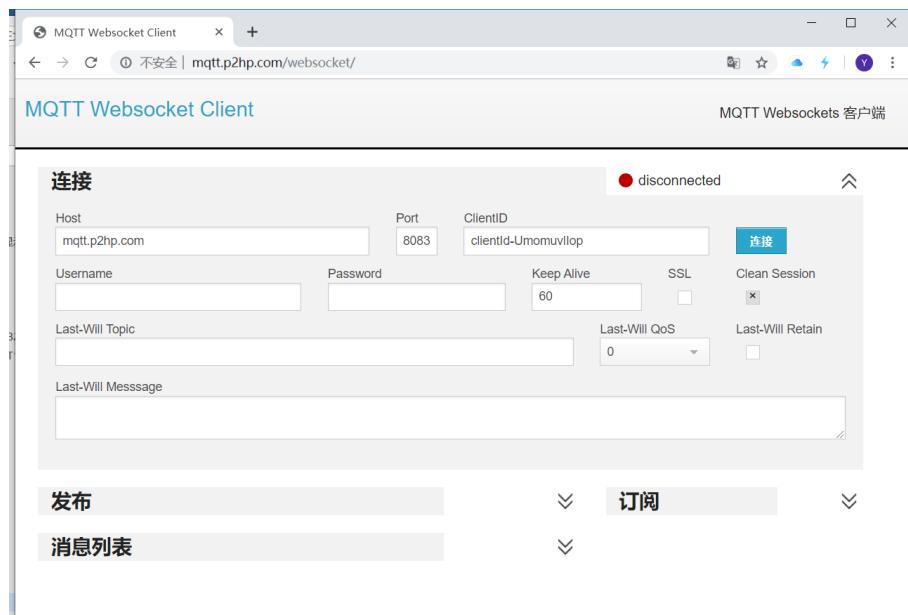


图 7-35 MQTT 助手

连接成功可以看到显示 Connected。

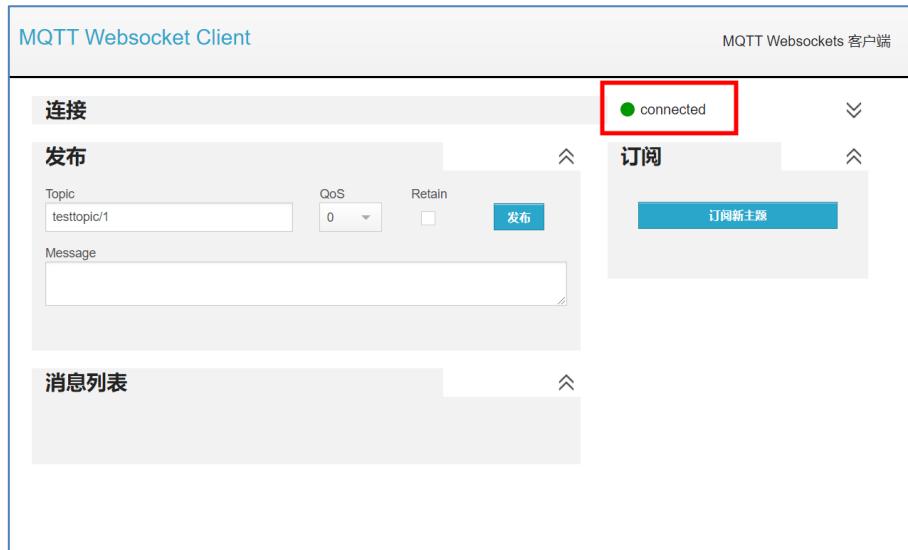


图 7-36 MQTT 助手连接成功

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。

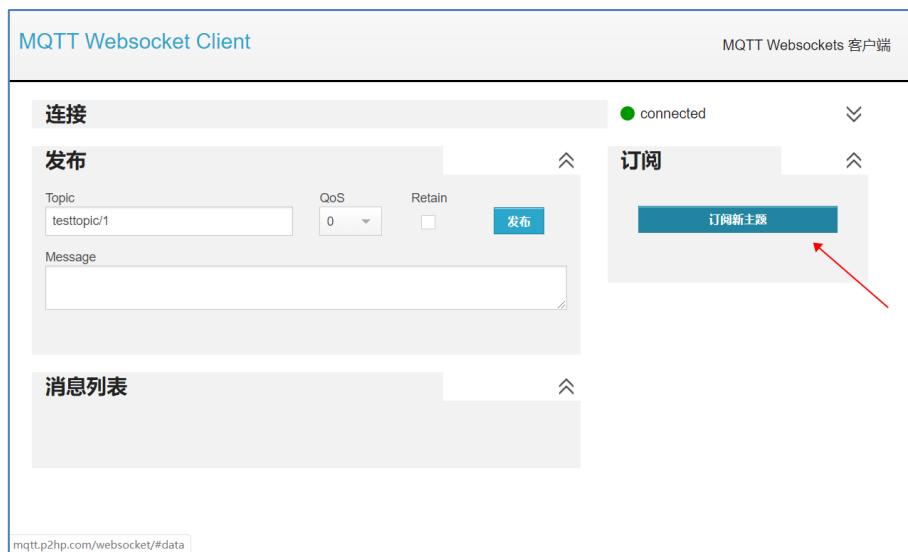


图 7-37 订阅主题

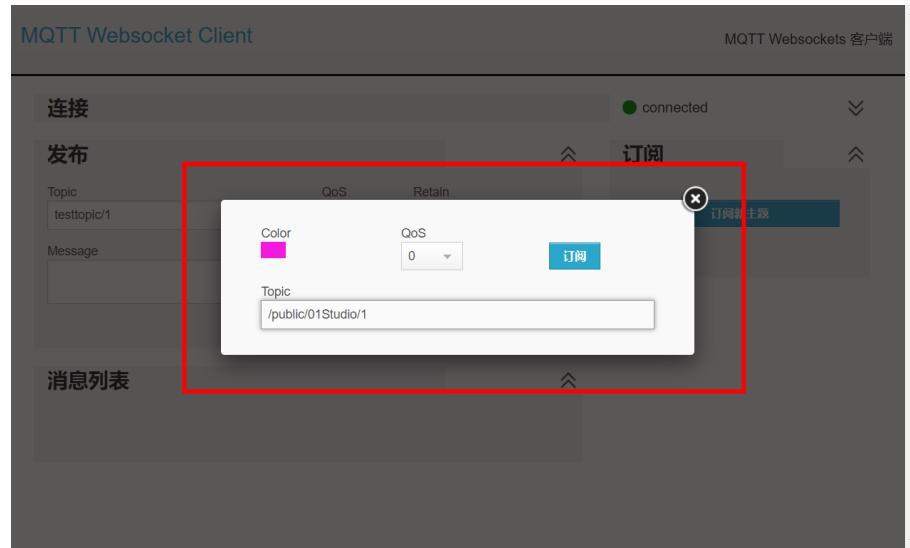


图 7-38

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。

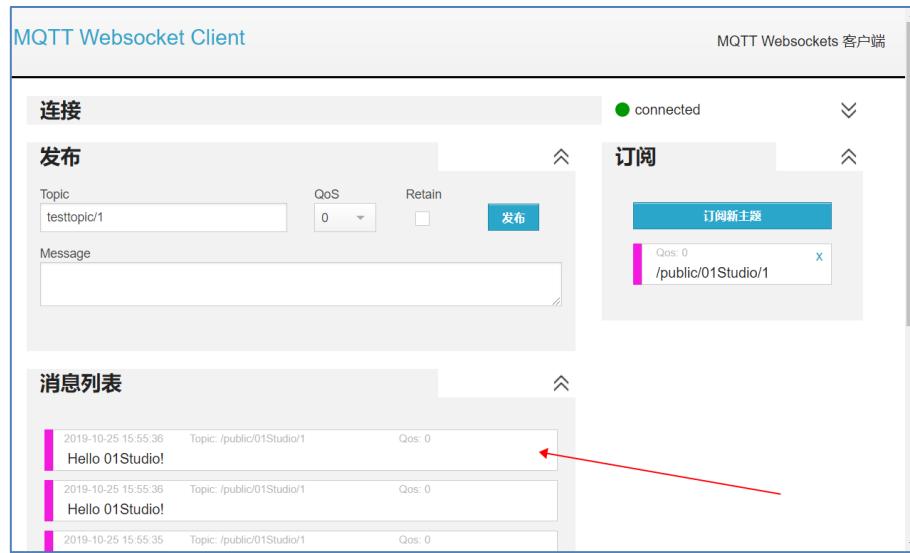


图 7-39 MQTT 助手收到开发板发布的信息

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!”。

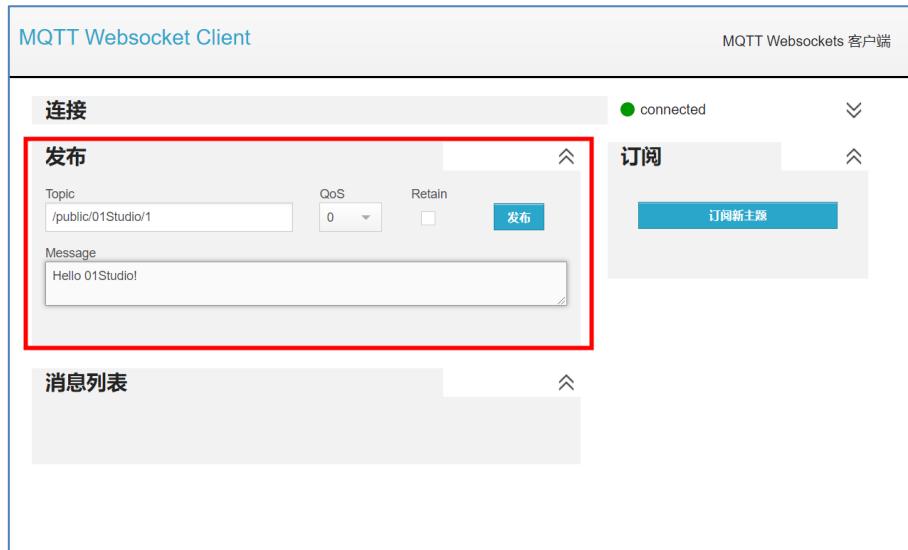


图 7-40 输入即将发布的主题和信息

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）。

```
串行终端 | ↻ ⟲
topic: b'/public/01Studio/1'
msg: b'Hello 01Studio!'
topic: b'/public/01Studio/1'
msg: b'Hello 01Studio!'
topic: b'/public/01Studio/1'
msg: b'Hello 01Studio!'

搜索结果 串行终端
```

The screenshot shows a serial terminal window with the title "串行终端". It displays three identical message pairs, each consisting of a topic and a message. The topic is "b'/public/01Studio/1'" and the message is "b'Hello 01Studio!'". The terminal has two tabs at the bottom: "搜索结果" (Search Results) and "串行终端" (Serial Terminal).

图 7-41 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。

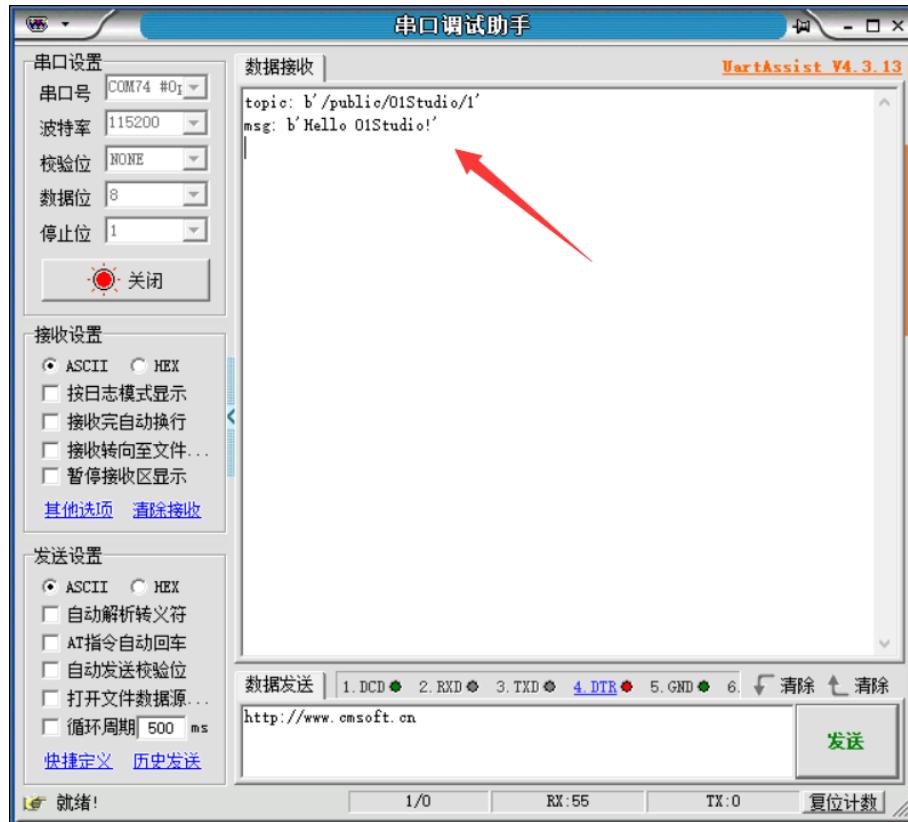


图 7-42 串口调试助手

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：

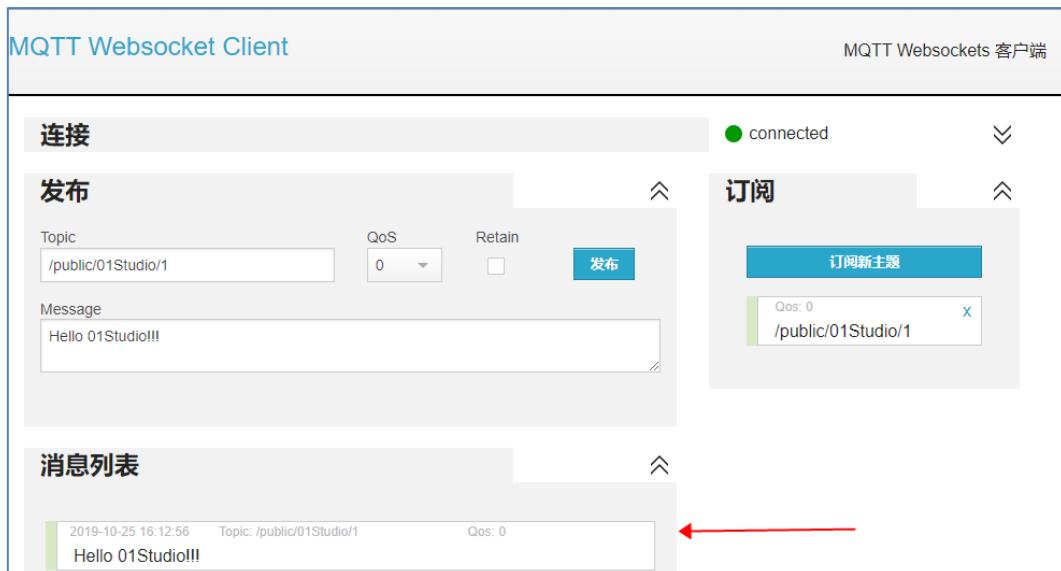


图 7-43 客户端同时发布和订阅信息

- **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。

7.5 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）。

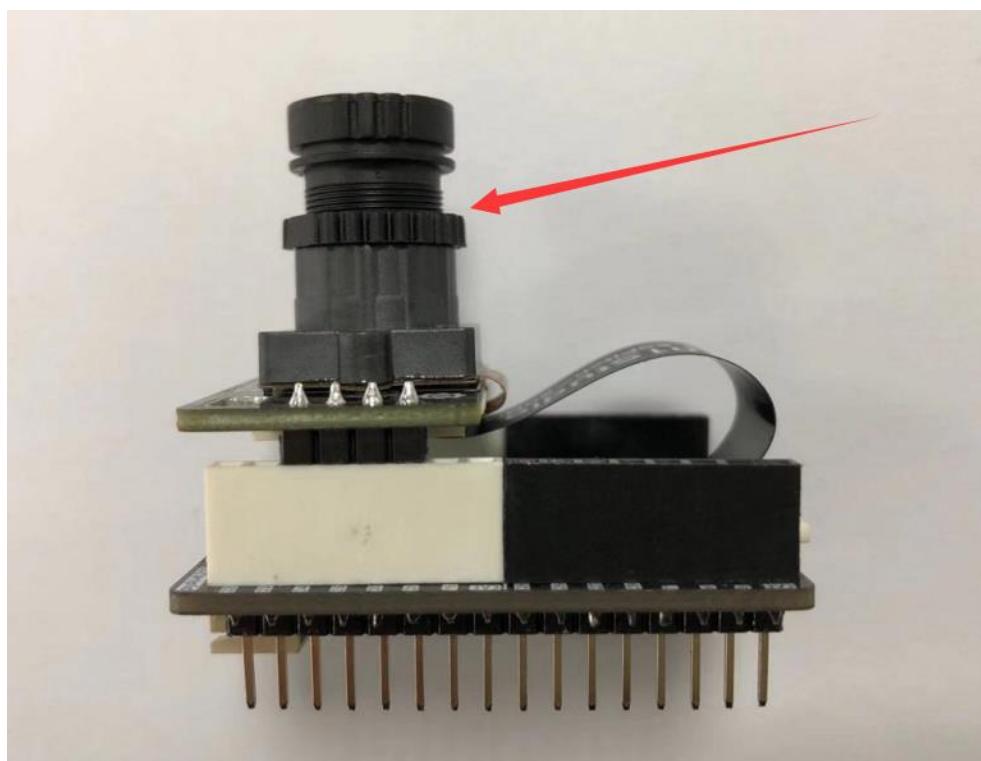


图 7-44

7.5.1 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 7-45 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 7-7

以下是标准镜头和广角镜头拍摄对比。



图 7-46 标准镜头



图 7-47 广角镜头

7.5.2 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 7-48 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 7-8

以下是标准镜头和长焦镜头拍摄对比。



图 7-49 标准镜头



图 7-50 长焦镜头

7.5.3 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 7-51 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 7-9

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 7-52 标准镜头

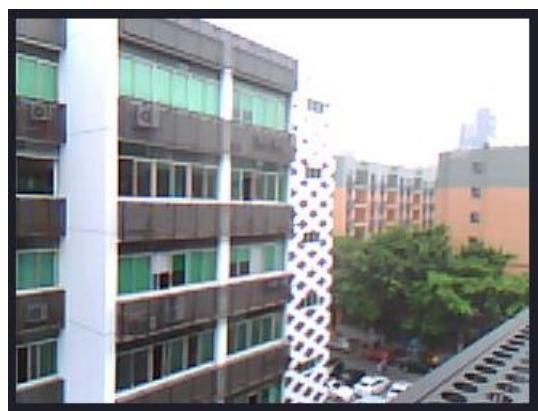


图 7-53 广角镜头

7.5.4 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 7-54 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 7-10

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 7-55 标准镜头



图 7-56 手动调节镜头

第8章 项目应用

8.1 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。

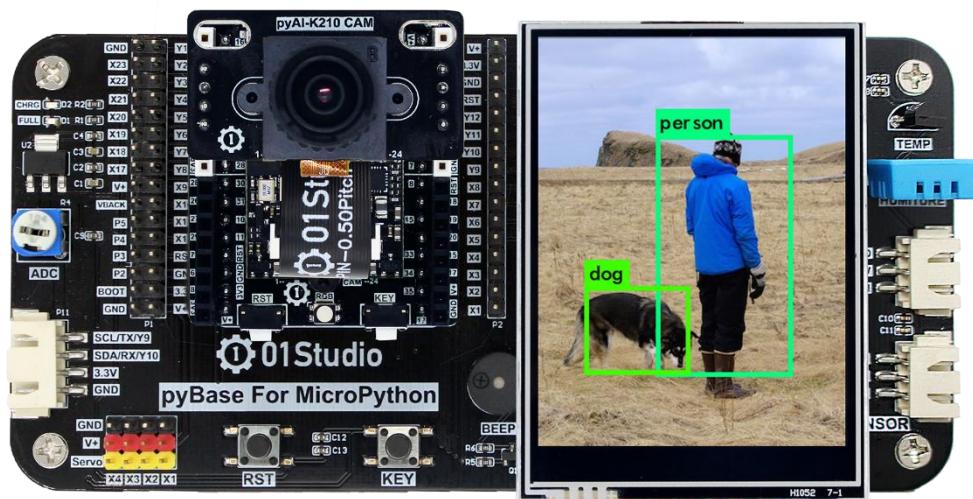


图 8-1 pyAI-K210 开发套件

● 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 [4.4 外部中断](#) 章节内容；

拍摄照片实验：请参阅 [5.9 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：

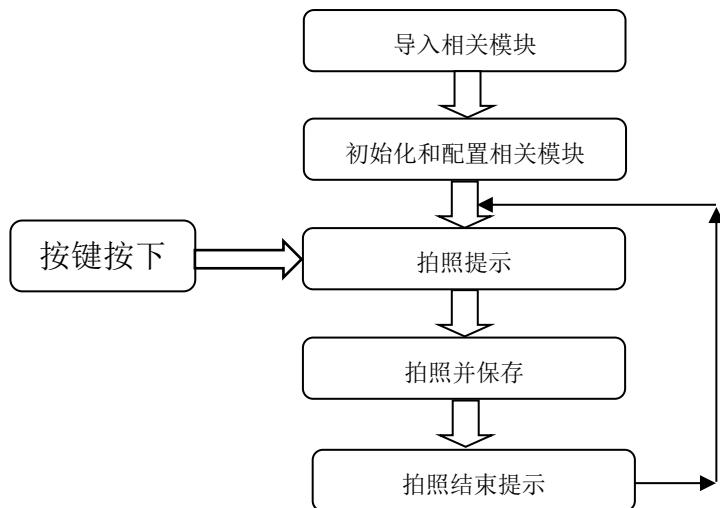


图 8-2 代码编写流程

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 K210 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！

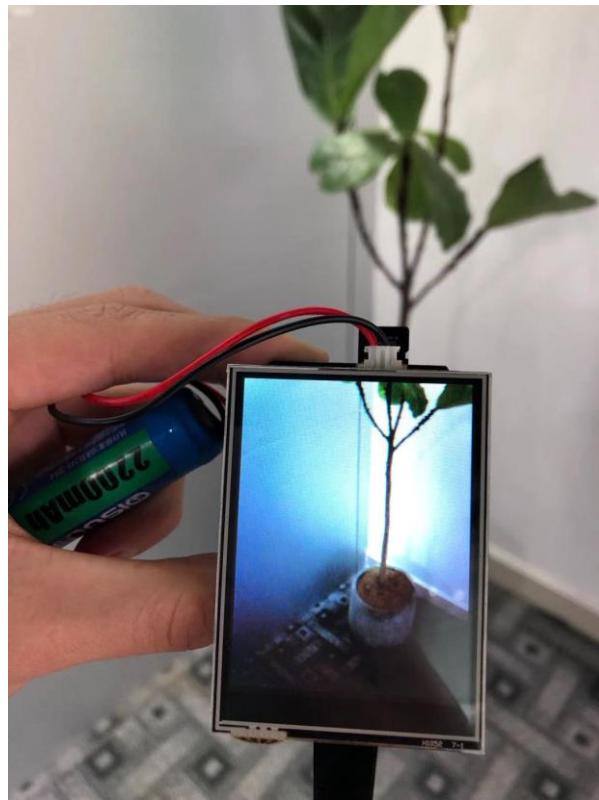


图 8-3 拍摄图片 (摄像头后置模式)

当然你也可以结合 pyBase 底板来自拍：

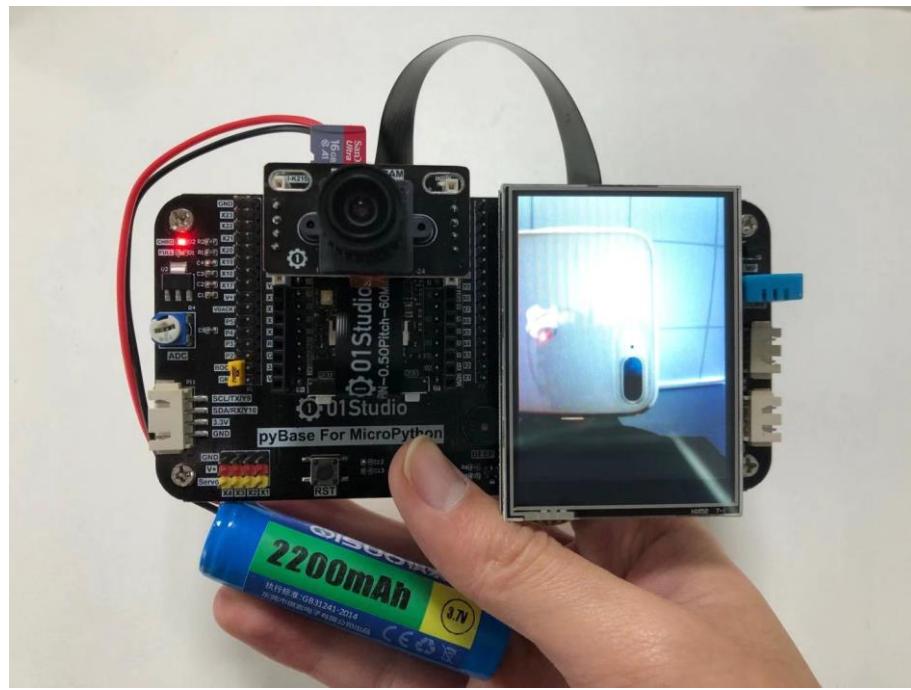


图 8-4 结合开发底板自拍 (摄像头前置模块式)

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 8-5 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。

8.2 视频播放器

- **前言：**

由于 pyAI-K210 开发套件有 LCD 和音视频功能，那么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。

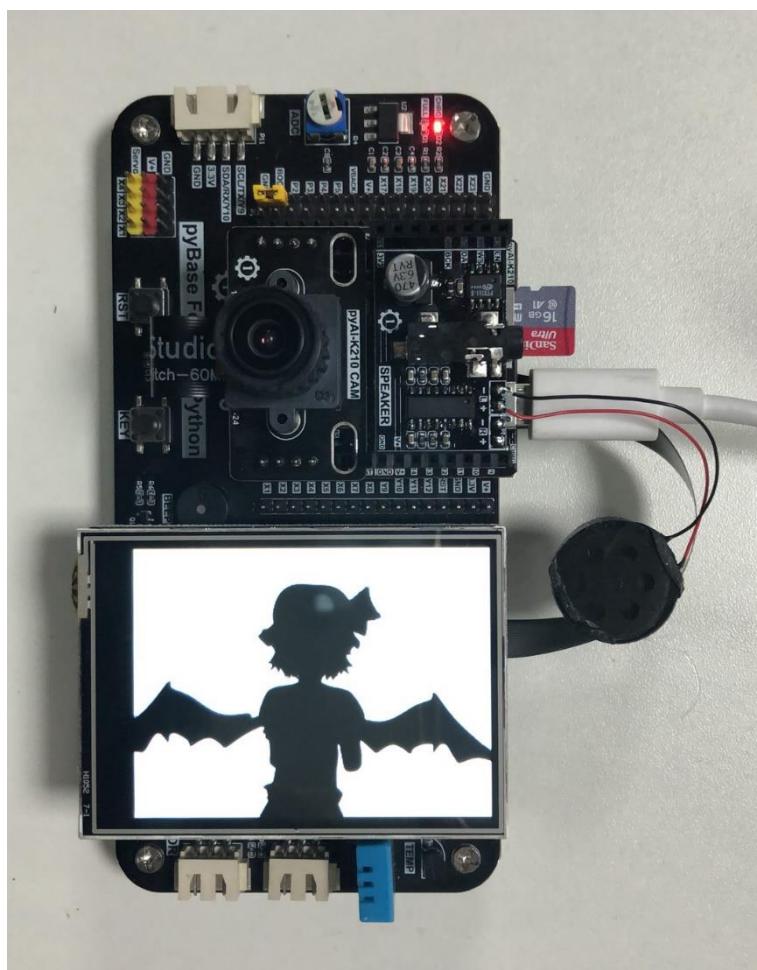


图 8-6 pyAI-K210 开发套件

- **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 video

v=video.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.record ()

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 8-1 video 对象

代码编写流程如下：

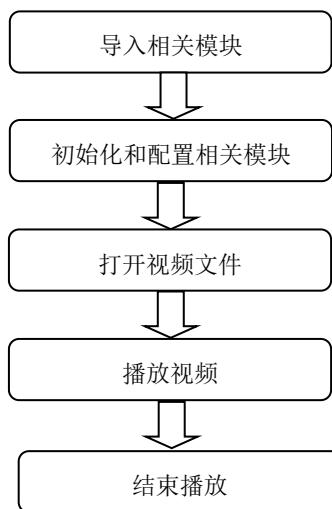


图 8-7 代码编写流程

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。

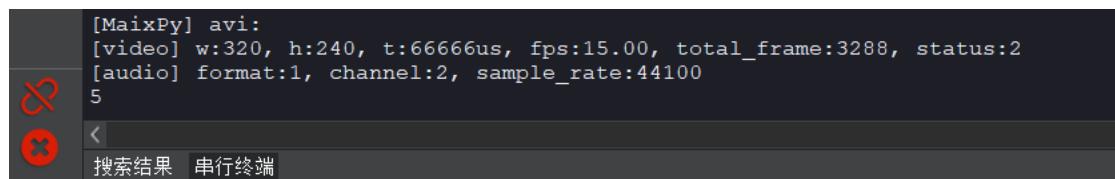


图 8-8

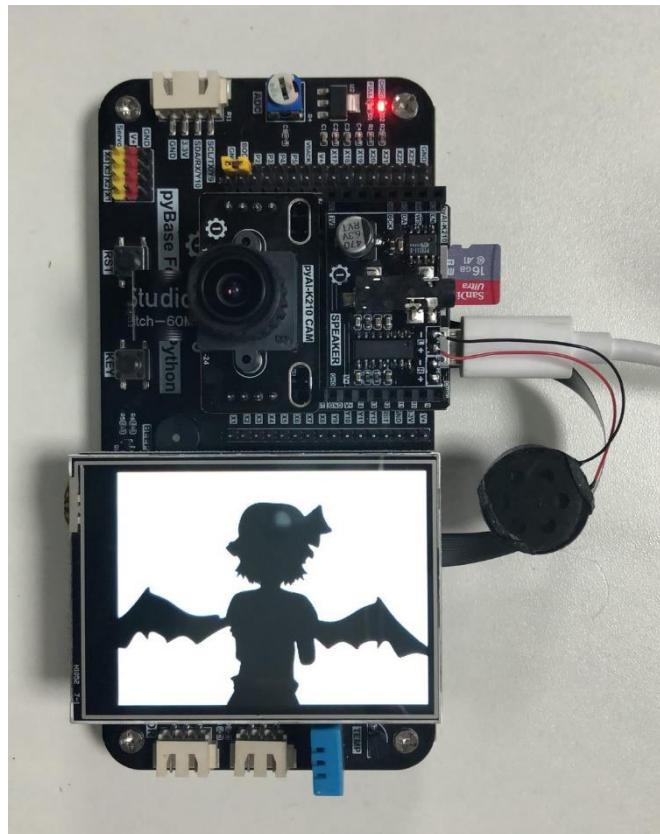


图 8-9 视频播放实验

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。

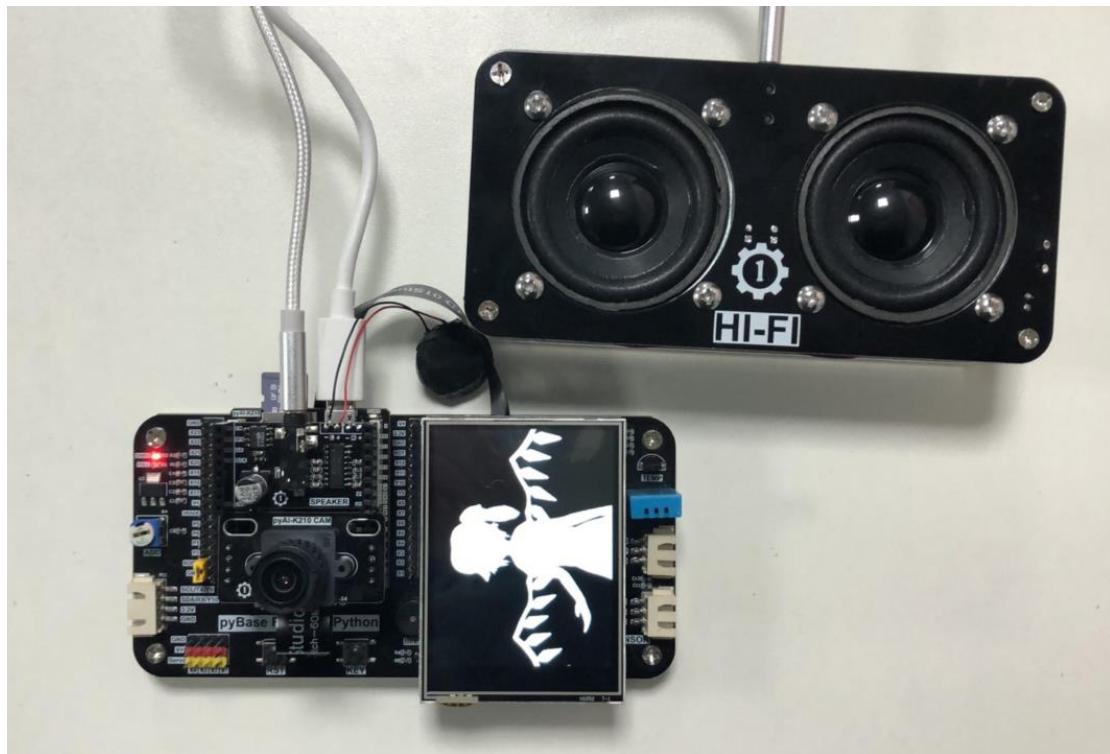


图 8-10 外接扬声器播放

● 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。

8.3 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 pyAI-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。

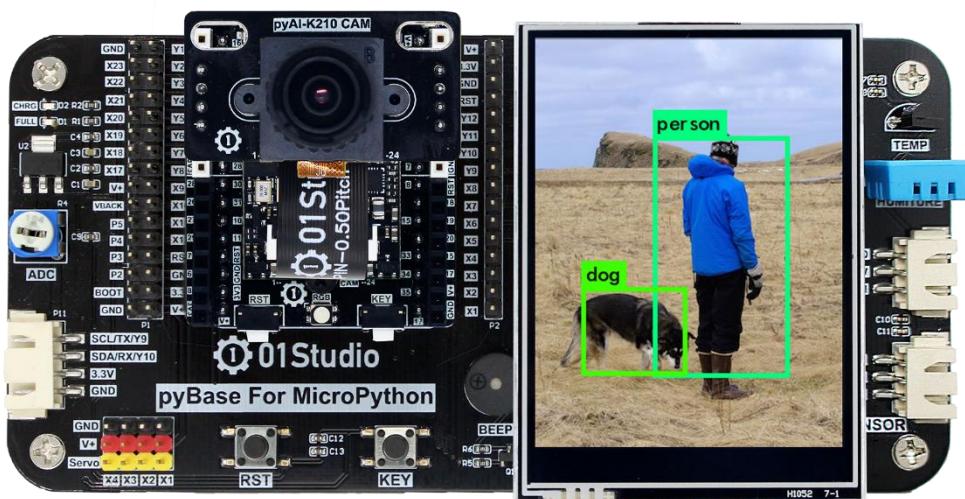


图 8-11 pyAI-K210 开发套件

- **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
<code>import nes</code>
导入 nes 模块;
使用方法
<code>nes.init(rc_type=nes.KEYBOARD, cs, mosi, miso, clk, repeat=16, vol=5)</code>
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
<code>nes.run(xx.nes)</code>
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 8-2 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】: □

【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 8-3 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：

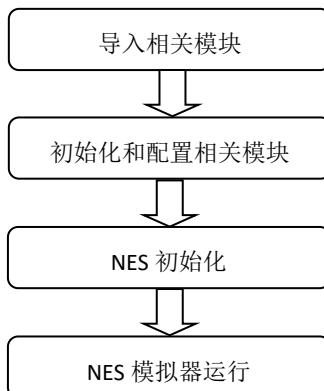


图 8-12 代码编写流程

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 8-13 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。

MicroBit 从0到1

用方块开始你的编程之旅
01Studio团队 编著



01Studio
-让编程变得简单有趣-

MicroPython 从0到1

用python做嵌入式编程

(基于pyBoard STM32F405平台)

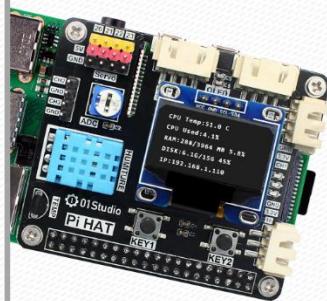
01Studio团队 编著



01Studio
-让编程变得简单有趣-

树莓派从0到1

(基于树莓派4B平台)
01Studio团队 编著



01Studio
-让编程变得简单有趣-

基于pyBoard STM32F405平台

MicroPython
从0到1
用python做嵌入式编程
(基于pyBoard STM32F405平台)
01Studio 编著



01Studio
-让编程变得简单有趣-

基于ESP8266平台

基于ESP32平台

基于NRF62840平台

基于OpenMV4平台

基于K210平台

基于哥伦布 STM2F407平台



关注公众号，免费下载