

MicroPython (命令行) 教程

日期	作者	内容	备注
2019.4.29	小董	完成文档框 架构建	
2019.4.30	小董	完善常用命 令	

什么是 MicroPython?

MicroPython^[1] , 是 Python3 编程语言的一个完整软件实现, 用 C 语言编写, 被优化于运行在微控制器之上。MicroPython 是运行在微控制器硬件之上的完全的 Python 编译器和运行时系统。提供给用户一个交互式提示符 (REPL) 来立即执行所支持的命令。除了包括选定的核心 Python 库, MicroPython 还包括了给予编程者访问低层硬件的模块。

MicroPython 努力与普通的 Python (称为 CPython) 尽可能兼容, 这样如果你了解 Python 就已经知道了 MicroPython。另一方面, 您对 MicroPython 的了解越多, 您在 Python 中的表现就越好。

除了实现一系列核心 Python 库之外, MicroPython 还包括用于访问低级硬件的“机器”等模块。

MicroPython 的优势?

1.编译环境

C 语言、汇编语言在编写程序之前, 需要下载编译环境, 然而 Micropython 不需要任何编译环境, 只需将开发板 U 口插入电脑, 便可在电脑上出现一个盘符, 跟插入 U 盘似得, 然后打开 U 盘, 直接记事本编辑 main.py 就行, 它的舒服在于 MicroPython 不需要任何工具和环境, 任何文本工具+1 块开发板即可开发编译, 可以给它起个外号 --口袋编程计算机。。。国内做的比较好的就是 TPYBoard v102 开发

板,有兴趣的可以去学习了解一下,它资料齐全,有视频学习资料,还是不错的,我就是这么入坑的哦。



2.操作难度

汇编语言基本操作简单,但是要实现复杂的项目目标相对较难,代码冗长,调试查错困难 ;C 语言有了语句和大量的库函数,相对汇编来说简单了 ;MicroPython 有了比 C 更多的库函数,代码格式不仅变得简洁,在库的支持下很多函数和方法都不用自己再去写,直接 `import xx` 就可以,把类库导入就 ok。以前点一个灯需要五六行代码,现在只需要 1 行代码就可以点亮 LED, So Easy。

```
1 # main.py -- put your code here!
2 import pyb
3 pyb.LED(1).on()
```

3.程序结构

汇编语言总体使用跳转结构,不管是子程序的调用还是循环或者散转理论上都是用跳转的方式,中断操作或强行出程序段的操作都需要对压栈有有精确的控制,要求非常严格 ;C 语言总体使用循环结构或顺序结构,不再需要到处跳来执行程序,中断操作会自动进行压栈不需要人为干预,在时序要求高时可嵌入汇编提高效率 ;MicroPython 可以使用 C 的程序结构或者使用线程结构,线程结构要求的是严格且合理的分配好线程工作时间不能出现冲突,对资源要有很好的把控,不能空挂线程浪费资源,理论上多线程可以更高效的运行,虽然单核同时只能运行单线程但是从宏观角度来说同步的,在时间要求不严格的情况下可以有效的降低程序的编写难度。

4.编写方式

汇编在不使用宏的情况下,通常都是指令直接操作单元,需要记忆大量的数据单元用途,且由于程序行数较多(可能会到一两千行)需要来回翻看程序代码,整个编写流程比较繁琐。而且由于都是 8 位的单元复杂的数据运算需要嵌套 C 语言来完成 ;C 语言使用任何变量都需要先定义,相比不用特意去记忆变量名称,整体结构清晰多数编程软件都有跳转子程序功能,查找程序块非常方便。有大量的成品头文件包含各种常用函数,相对少了很多复杂的程序代码编写 ; MicroPython 继承了 C 的编写优点,且不需要预定义,不需要任何结

束符号，只需要换行即可继续编写，但是要注意一些特殊情况下 Tab 键的使用。

5. 易读性

汇编本身难读因为标记少且直接操作单元，视觉结构上不仅就两列代码而且需要不停的跳转所以本身在读取上就很难；C 语言简单的操作代码比较清晰，但是复杂的就会方法中找方法，很多复杂的代码非本人去找，就让人感到非常头疼；MicroPython 相比 C 有更好的易读性，且由于取消了很多特殊符号整体感觉也比较清晰，很多方法都是单独写成单独的库，直接调用即可，每个功能都有单独的类库，看起来极其方便，一目了然。

通过这几个维度的对比，大家应该很轻松就知道我为什么说 MicroPython 用起来让人舒服。目前 AI 是当前社会发展的主流，MicroPython 的出现是一个时机，让很多像我这样的初学者，也可以自己做智能小车，学习硬件，它比 C 语言学习更简单，更适合加入教育，人人都可以学编程，写程序，由此可以看出 MicroPython 未来市场前景是非常大的，是比 Arduino 更强大、更容易编程的开发板。

我们可以用 MicroPython 干什么？

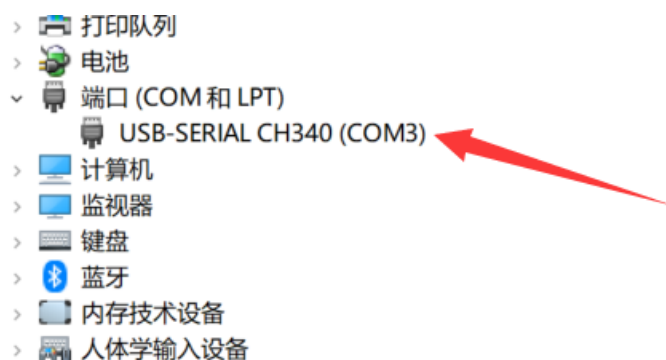
当然是**点灯**啦 !!!!

废话不多说，下面让我们开始一起点灯吧！

第一步：通过 USB 连接开发板

不像 STM32 版本的 pyboard, PineconePi ONE 所搭载的 SWM320VET7 本身没有 USB, 只能通过 TTL 串口访问 SWM320VET7。我们先介绍通过串口方式进行连接。

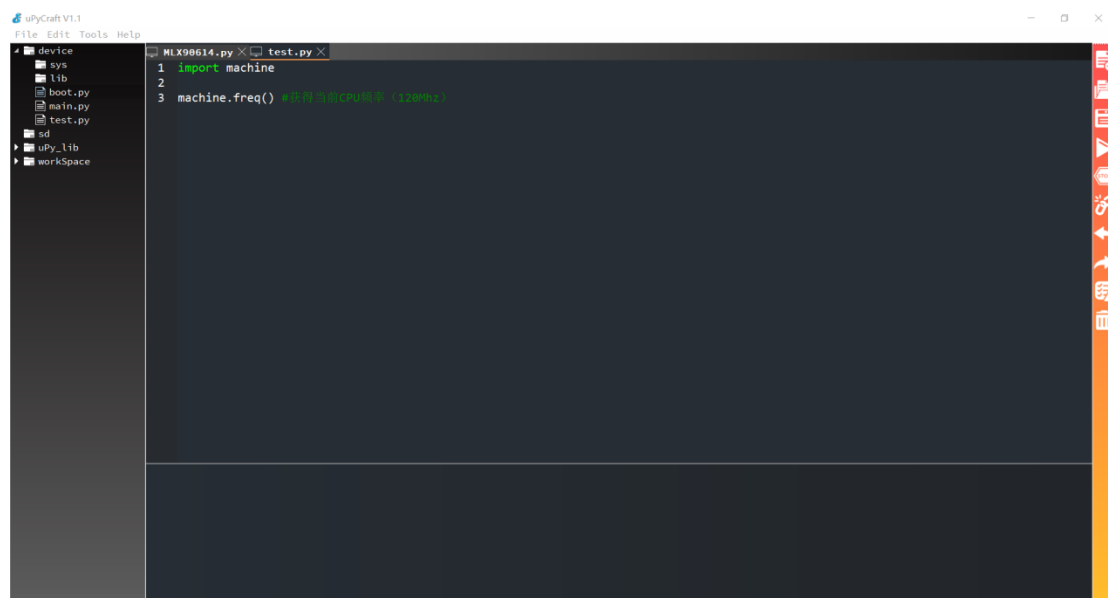
在 PineconePi ONE 开发板上，带有 Micro USB 接口，以及 USB 转串口芯片 CH330N，它可以方便实现计算机与 PineconePi ONE 的连接。使用前需要先安装 CH330 的驱动(与 CH340 驱动通用)，这样当 PineconePi ONE 连接到计算机，就会出现一个串口设备。下面是 windows 上显示的虚拟串口，Linux 下通常是/dev/ttyUSB0。



第二步：使用终端软件连接开发板

可使用 uPyCraft IDE(V1.1)来连接开发板,使用文档及下载链接见：

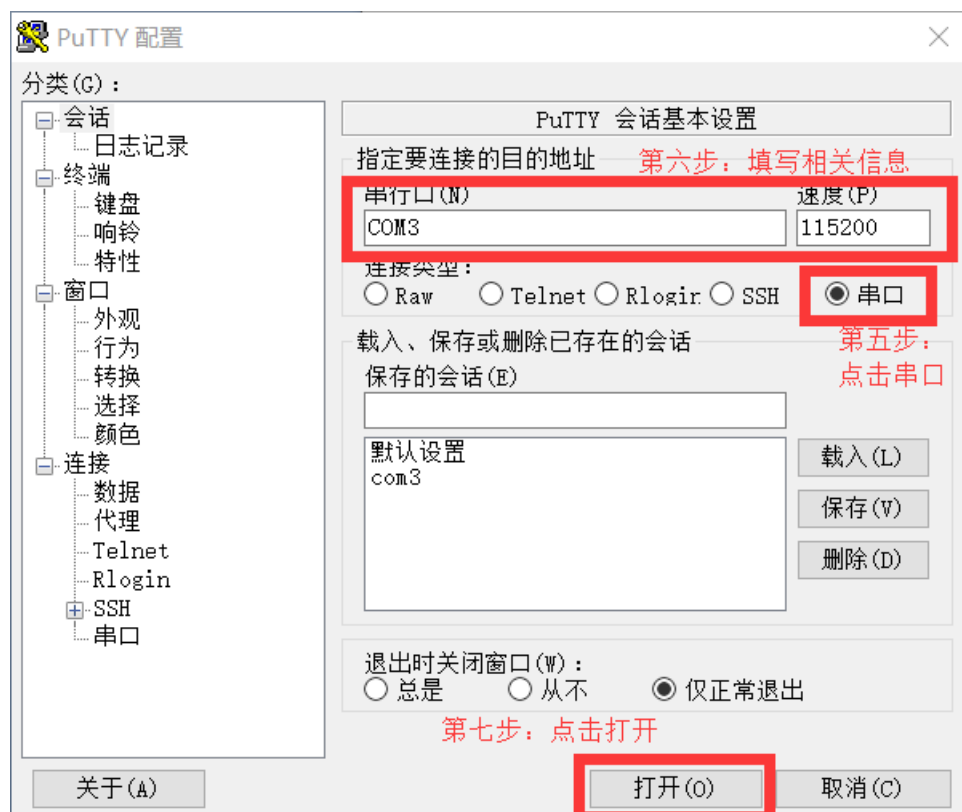
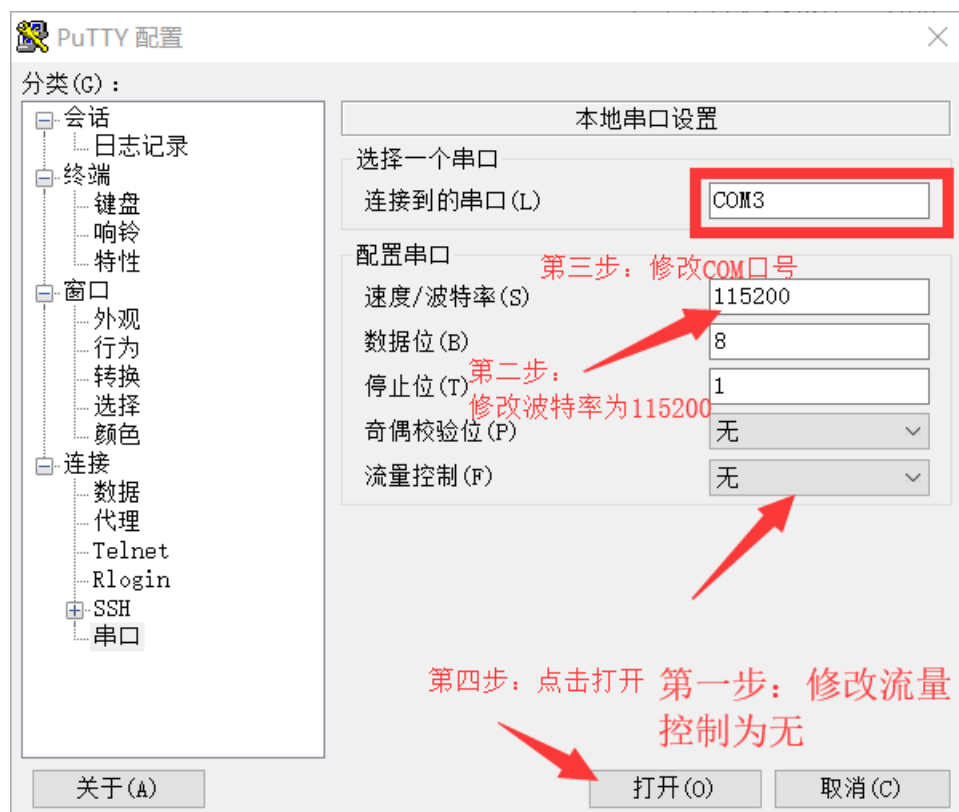
<http://docs.dfrobot.com.cn/upycraft/>



为了使用 MicroPython, 我们需要运行一个终端软件, 下面以 putty 为例, 其他软件用法也类似。先要设置串口, 选择 CH340 的串口

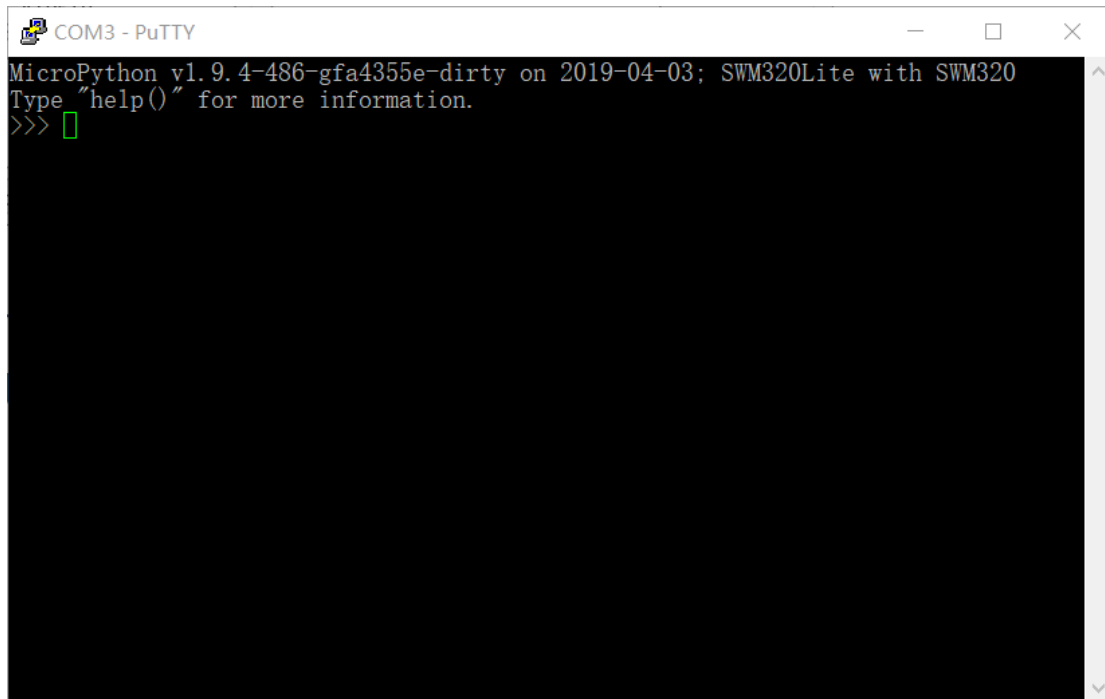
(Windows 上在设备管理器中查看串口, Linux 下在/dev/中查看), 并设置波特率为 115200。有些软件还要设置更多参数, 一般设置 8 位数据, 无校验, 1 位停止位, 无流量控制等。

PS :Putty 软件见 PineconePi ONE 仓库下/MicroPython/putty/



最终, 我们按下开发板上 S1 按钮进行硬件复位, 便可以在 Putty 上看

到如下画面：



```
COM3 - PuTTY
MicroPython v1.9.4-486-gfa4355e-dirty on 2019-04-03; SWM320Lite with SWM320
Type "help()" for more information.
>>> █
```

下面来介绍常规操作：

在终端下，灵活使用快捷键可以帮助我们。常用的快捷键有：

- CTRL-A – on a blank line, enter raw REPL mode (这个快捷键不是为了输入程序，一般不要使用)
- CTRL-B – 在空命令行下，回到正常 REPL 交互模式
- CTRL-C – 中断正在运行的程序
- CTRL-D – 软复位
- CTRL-E – 粘贴模式 (按下鼠标右键进行粘贴)
- 上下方向键 – 调出以前输入命令

运行程序时, 如果出现问题可以随时用 **Ctrl-C** 中止运行, 或者在空命令行下用 **Ctrl-D** 软复位。如果还不能解决问题, 就直接按复位键进行硬复位。

遇到有疑问的地方, 可以输入 **help()** 查看帮助, 甚至可以查看一个函数或者库的帮助, 如 **help(machine)**。

还可以用 **dir()** 查看已经载入的模块、函数、变量, 也可以用 **dir** 查看一个库里面包含的内容, 如 **dir(machine)**

下面总算可以点灯了！！！！

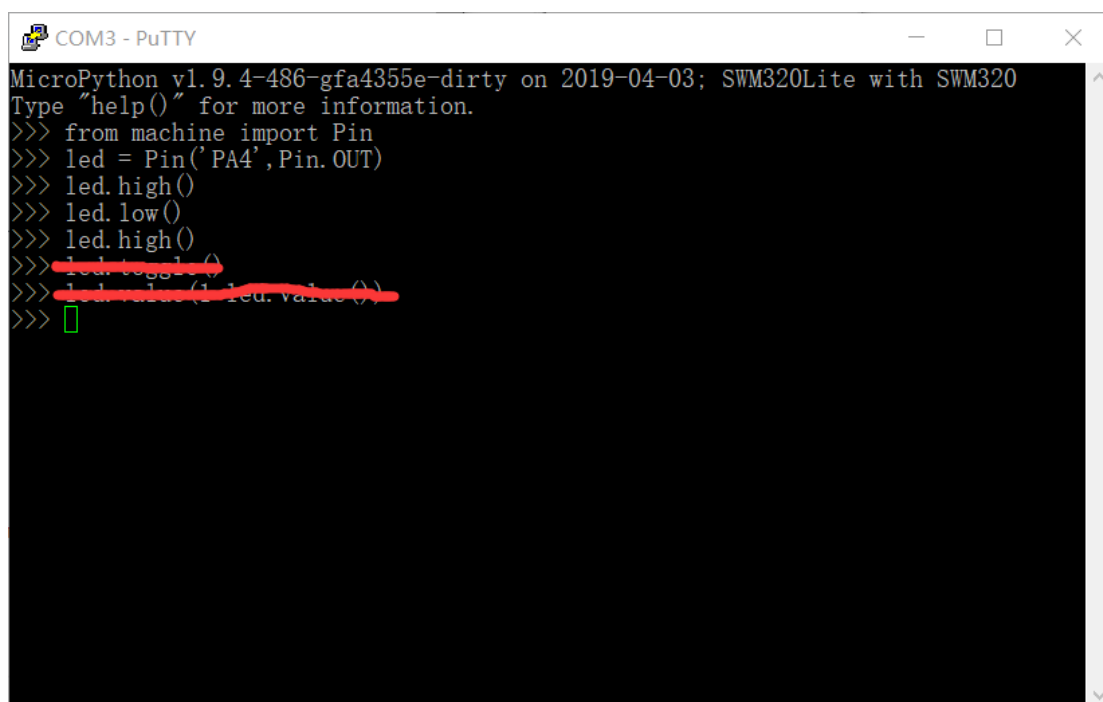
让我们输入代码 (当然, 可以 **CTRL+E** 进入粘贴模式后, 直接点击鼠标右键进行粘贴)

```
from machine import Pin

led = Pin('PA5', Pin.OUT)

led.low()

led.high()
```



```
COM3 - PuTTY
MicroPython v1.9.4-486-gfa4355e-dirty on 2019-04-03; SWM320Lite with SWM320
Type "help()" for more information.
>>> from machine import Pin
>>> led = Pin('PA4', Pin.OUT)
>>> led.high()
>>> led.low()
>>> led.high()
>>> led.toggle()
>>> led.value(1-led.value())
>>> █
```

我们便点亮了第一盏 LED 灯！

附录

(常用命令指南)

MicroPython port to SWM320 (Cortex-M4 MCU)

上电后默认运行 **boot.py** 和 **main.py**

系统

CPU 主频

```
import machine
```

```
machine.freq() #获得当前 CPU 频率 (120Mhz)
```

文件

```
import os
```

```
os.listdir() #查看系统当前文件列表
```

```
os.listdir('lib') #查看目录 lib
```

```
os.mkdir("PineconePi") #创建文件夹 PineconePi
```

```
os.chdir("PineconePi") #进入子目录 PineconePi
```

```
os.getcwd() # 获取当前目录
```

```
os.rmdir("hx") #删除文件夹 (只能是空的文件夹)
```

```
os.rename("hm.txt","11") #重命名文件

os.remove("11") #删除文件


f = open('main.py', 'r') #以读取打开 main.py
f.read() #读取 main.py 内容
f.close() #关闭文件


f = open('main.py', 'W') #以写入打开 main.py
f.write("#PineconePi ONE\n") #对 main.py 写入内容 #PineconePi
ONE 并换行
f.close() #关闭文件
```

Pin

输出

```
from machine import Pin

led = Pin('PA5', Pin.OUT)

led.low()

led.high()

led.toggle()

led.value(1-led.value())
```

输入

```
key = Pin('PA4', Pin.IN)

key = Pin('PA4', Pin.IN, mode=Pin.PULL_UP) # 可开启内部上拉、
下拉

key.value()
```

中断

```
led = Pin('PA5', Pin.OUT)

key = Pin('PA4', Pin.IN, irq=Pin.FALL_EDGE, callback=lambda key:
led.toggle())
```

Timer

定时器

```
from machine import Pin, Timer
```

```
led = Pin('PA5', Pin.OUT)
```

```
tmr = Timer(0, 10000, callback=lambda tmr: led.toggle())
```

```
tmr.start()
```

第二个参数'10000'是定时周期时长, 单位为 0.1ms

计数器

```
from machine import Pin, Timer
```

```
led = Pin('PA5', Pin.OUT)
```

```
ctr = Timer(2, 3, mode=Timer.MODE_COUNTER, pin='PA4',
```

```
callback=lambda ctr: led.toggle())
```

```
ctr.start()
```

Timer4、Timer5 没有 Counter 功能

UART

```
from machine import UART

ser = UART(1, 57600, txd='PA9', rxd='PA10') # 等效于下面一行
ser = UART(1, 57600, bits=UART.DATA_8BIT,
parity=UART.PARITY_None, stop=UART.STOP_1BIT, txd='PA9',
rxd='PA10')

ser.write('Hi from Synwit\n')
if ser.any(): s = ser.read()
```

UART0 是 micropython 命令界面, 用户程序中不要使用

SPI

```
from machine import Pin, SPI

spi = SPI(0, 100000, mosi='PC4', miso='PC5', sclk='PC7') # 等
效于下面一行
spi = SPI(0, 100000, polarity=0, phase=0, bits=8, mosi='PC4',
miso='PC5', sclk='PC7')
```



```
spi_cs = Pin('PC2', Pin.OUT)
spi_cs.high()

spi_cs.low()
spi.write(b'\x33\x78\xAF')
spi_cs.high()
```

I2C

```
from machine import I2C

i2c = I2C(0, 100000, scl='PA10', sda='PA9')

i2c.writeto(0x6C, '\x33\x78\xAF')
```

端口 A、C 上的引脚有内部上拉，外部可不接上拉电阻；端口 B 上的引脚必须外接上拉电阻

ADC

```
from machine import ADC
```

```
adc = ADC(0, chns=ADC.ADC_CH4)
for i in range(5): print(adc.read())

adc.chn_select(ADC.ADC_CH4|ADC.ADC_CH5|ADC.ADC_CH6|ADC.
ADC_CH7)

for i in range(5): print(adc.read())
```

PWM

独立输出

```
import time
from machine import PWM
pwm = PWM(0, 15000, 5000, pin='PA10')
pwm.start()
time.sleep_ms(50)
pwm.period(30000)
pwm.duty(15000)
```

PWM 的计数时钟为 15MHz, 计数周期 15000 可得方波频率为 1KHz

互补输出

```
import time

from machine import PWM

pwm = PWM(1, 1500, 500, mode=PWM.MODE_COMPL, pinA='PA9',
pinB='PA11', deadzone=50)

pwm.start()


time.sleep_ms(50)

pwm.period(3000)

pwm.duty(1500)

死区最大取值 63
```

notice

- 1、PA12 引脚没有外设功能映射功能, 只能用作 Pin