

OpenART mini 说明书

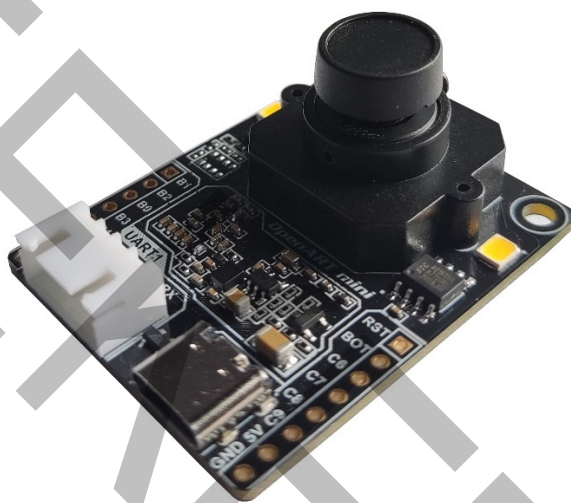
目录

目录.....	1
一、产品介绍.....	3
二、产品特性.....	3
三、技术参数.....	4
四、硬件说明.....	5
1、引脚说明.....	5
2、指示灯说明.....	5
五、使用说明.....	6
1、基本说明.....	6
1.1、供电.....	6
1.2、SD 卡及脱机运行.....	7
2、例程解析.....	8
2.1、LED 的控制.....	8
2.2、GPIO 的控制.....	10
2.3、PWM 的控制.....	12
2.4、UART 的使用.....	13
2.5、DEBUG_UART 的使用.....	17
2.6、SPI 的使用.....	19
3、OpenART mini 运行模型.....	24
六、附录-常见问题及注意事项.....	25
文档版本.....	26

SEEKFREE

一、产品介绍

OpenART mini 是我们在 NXP 的 OpenART 套件的基础上，去除非视觉部分而制作出来的迷你版。虽说只是迷你版，但“麻雀虽小，五脏俱全”。OpenART mini 不仅可以很轻松的完成机器视觉 (machine vision) 应用，还可以完成 OpenMV 不能完成的神经网络模型的部署和训练。对于有人工智能教育，非多媒体数据上的机器学习、机器视觉等需求的人士来说，OpenART mini 实在是一大利器！

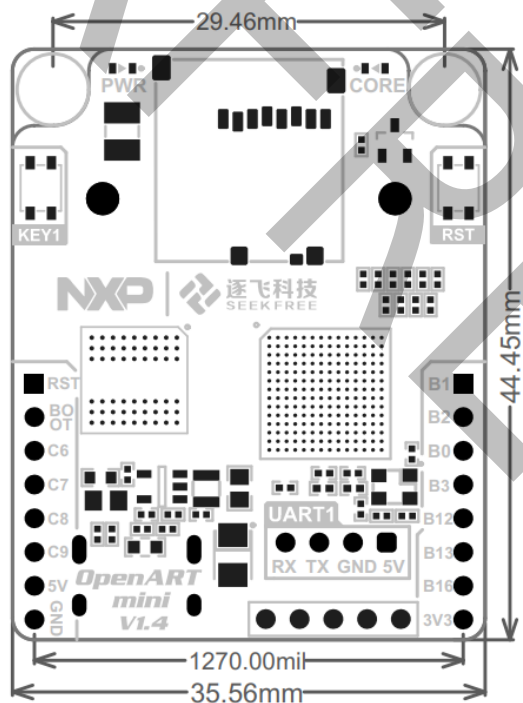


二、产品特性

- 1、 高性能处理器：采用 MIMXRT1064 芯片，其主频达到 600MHz,同时拥有 1M 片内 SRAM、4M 片内 FLASH 以及 32M 外置 SDRAM;
- 2、 Type-C 接口及 SD 卡槽：插上 SD 卡，OpenART mini 连接到电脑后会出现 COM 端口和一个 U 盘；
- 3、 一个高速的 SPI 总线，两个串口总线 (TX/RX);
- 4、 RT-Thread 内核、驱动、软件组件及开发环境；
- 5、 Micropython 环境，可用于二次开发及 AI 教学；
- 6、 OpenMV 机器视觉库，可运行 OpenMV IDE，自带视觉处理脚本；

三、技术参数

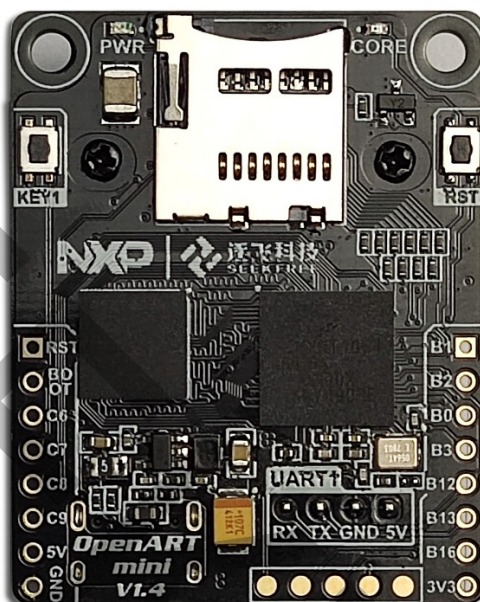
- 1、 输入电压：5V
- 2、 输出电压：3.3V
- 3、 支持的图像格式：Grayscale、RGB565
- 4、 最大支持的像素：QVGA
- 5、 重量：
 - 130°镜头：13.0 ± 0.2 g
 - 110°镜头：12.5 ± 0.2 g
 - 90° 镜头：16.0 ± 0.2 g
- 6、 长度：44.45 mm
- 7、 宽度：35.56 mm
- 8、 功率：0.85 ± 0.03 W



规格丝印图

四、硬件说明

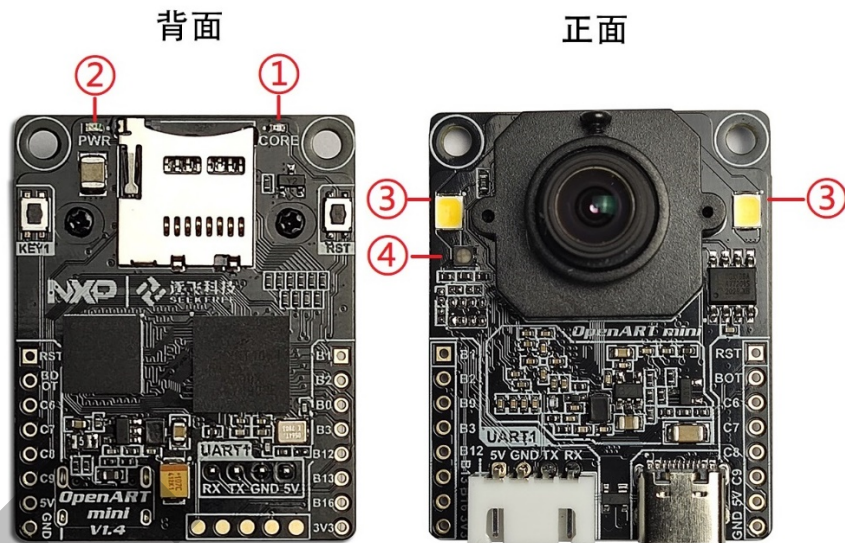
1、引脚说明



引脚名称	引脚定义	引脚名称	引脚定义
RST	Reset 引脚	B1	SPI3_MOSI / GPIO
BOOT/BOT	BOOT 引脚	B2	SPI3_MISO / GPIO
C6	PWM2_M0_A / GPIO	B0	SPI3_SCK / GPIO
C7	PWM2_M0_B / GPIO	B3	SPI3_CS0 / GPIO
C8	PWM2_M1_A / DEBUG_UART_TX / GPIO	B12	UART1_TXD / GPIO
C9	PWM2_M1_B / DEBUG_UART_RX / GPIO	B13	UART1_RXD / GPIO
5V	5 V 电源 (输入)	B16	GPIO
GND	电源地	3V3	3.3 V 电源 (输出)

2、指示灯说明

OpenART mini 上有五个指示灯，如下图所示：



- (1) ①为内核灯，此灯未亮，OpenART mini 将无法正常工作，请检查是否先给其引脚供上电了，如果先给引脚供上电了，请断开引脚供电后，再接上 Type-C 使用，若还有问题请及时与客服联系；
- (2) ②为电源指示灯，此灯未亮，OpenART mini 可能未正确供电或者损坏；
- (3) ③为照明灯，对应 LED4，在本文使用方式章节会给出控制方式的例程说明；
- (4) ④为三色指示灯，三种颜色分别为红色、绿色、蓝色，又分别对应 LED1、LED2、LED3；
- (5) 当 OpenART mini 上电的瞬间，绿色灯会亮一下而后马上熄灭，同时内核灯和电源指示灯会常亮，这即说明 OpenART mini 正常启动了；
- (6) 在上电前，按住 OpenART mini 背面的 KEY1 按键不放，接上电源后松手即进入固件升级模式（此功能为预留功能，用户无需使用），此时绿色灯会常亮；

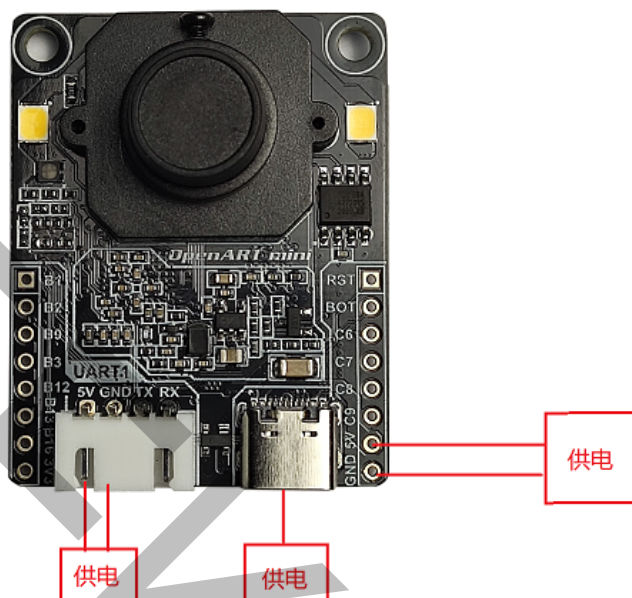
五、使用说明

1、基本说明

1.1、供电

OpenART mini 有两类电源输入端：Type -C 供电和 5V 引脚供电；

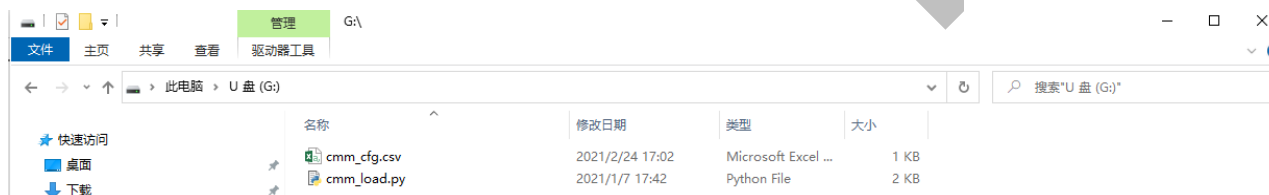
其中 5V 引脚供电，OpenART mini 上用 XH2.54 接口座子引出，配备我们送的 XH2.54 端子线，可以很方便的和单片机连接使用，当然你也可以直接从 5V 引脚处直接供电；当你要使用 OpenMV IDE 就只能使用 Type-C 供电使用了，如图所示：



特别说明：3V3 引脚是一个电源输出端，可以输出 3.3V，用于给其他传感器供电。但不能给它接上 3.3V 电源，没有保护很容易将 OpenART mini 烧毁。

1.2、SD 卡及脱机运行

当上电的时候，如果插入 SD 卡，OpenART mini 会从 SD 卡读取相应的配置文件，所以在使用的时候，务必将我们提供的资料里 SD 卡必备文件中的 `cmm_cfg.csv` 和 `cmm_load.py` 两个文件复制到 SD 卡根目录下，如下图所示：



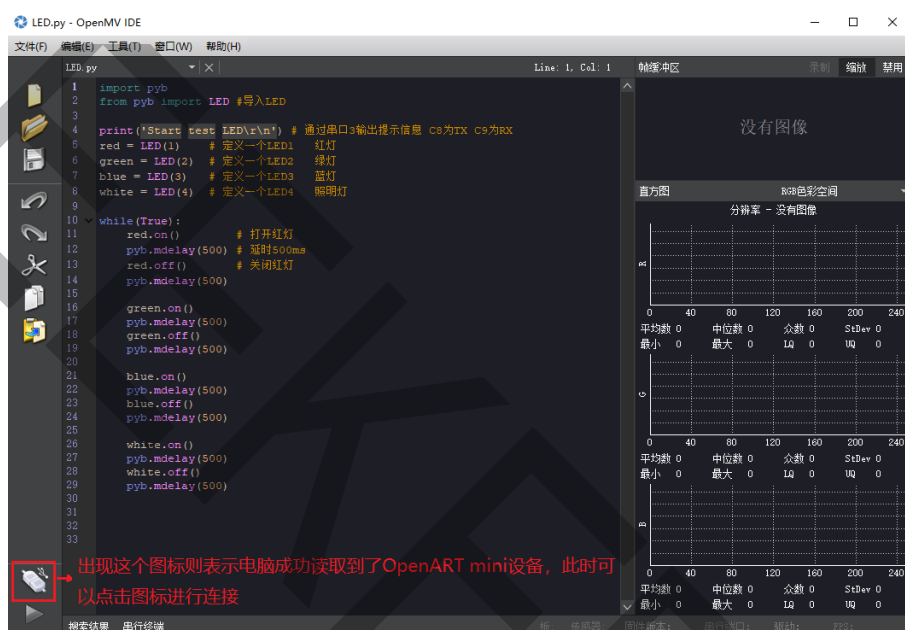
另外，你想脱机运行的话，可以将你的程序名称改成 `main.py` 复制放在根目录下，这样一来，你只要给 OpenART mini 供电，它就会自动运行你的代码了。**SD 卡最大支持 32G 的容量。**

2、例程解析

2.1、LED 的控制

首先，将 SD 卡插入 OpenART mini,然后连接上电脑，待电脑识别出 U 盘后，确定 SD 卡中是否有 1.2 章节中提到的必备文件，如果有则可进行下一步操作；

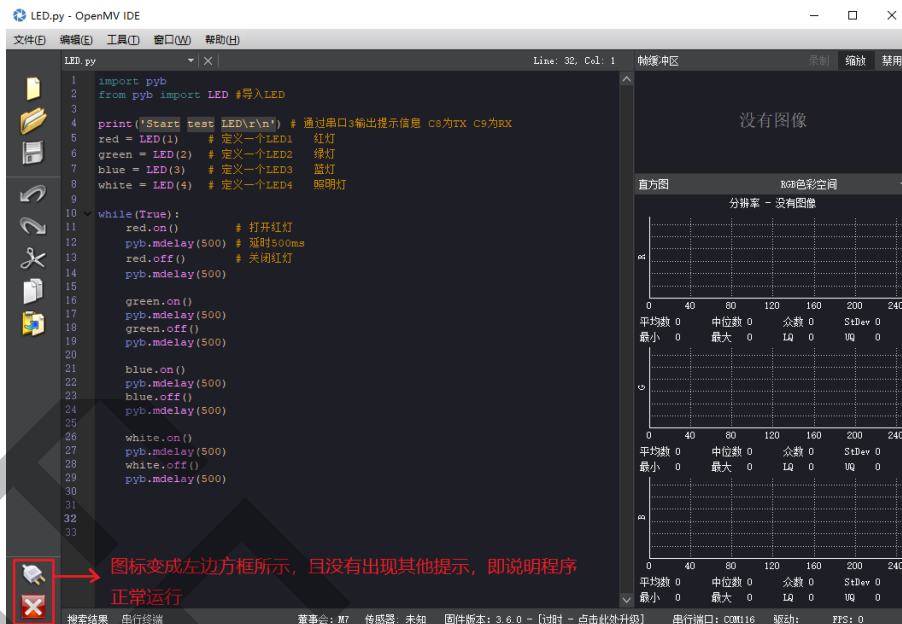
①用 OpenMV IDE 打开 LED.py，如下图所示：



②点击连接，如下图所示：



③运行 LED.py,如下图所示:



完成上述操作后,你将会看到 OpenART mini 上的指示灯会像代码中写一样红、绿、蓝、白依次闪烁起来。

下面也演示下如何脱机运行 LED.py 这个代码:

①备份原代码,如下图所示:

名称	修改日期	类型	大小
LED.py	2021/4/12 18:19	Python File	1 KB
LED - 副本.py	2021/4/12 18:19	Python File	1 KB

②将副本名称改为 main.py,复制到 SD 卡根目录下,如图所示:

名称	修改日期	类型	大小
LED.py	2021/4/12 18:19	Python File	1 KB
main.py	2021/4/12 18:19	Python File	1 KB

U盘(G:) 可用 29.7G

- cmm_cfg.csv
- cmm_load.py
- main.py

③按下 OpenART mini 的 RST 按键进行复位或者拔下 Type-C 重新上电,你同样会看到红、绿、蓝、白灯依次闪烁起来。

要点总结:

A、务必把 cmm_cfg.csv 和 cmm_load.py 两个文件复制到 SD 卡根目录;

B、要控制 LED, 必须加上以下两句话:

```
import pyb
```

```
from pyb import LED
```

这两句话就好比 C 语言中 `#include<xxx.h>` 包含头文件一样的作用, 此处就是引入 LED 控制所依赖的模块;

C、`red = LED(1)`, 这句话等号左边的名称可以任意取 (但要基于 python 的命名规则), `LED()`

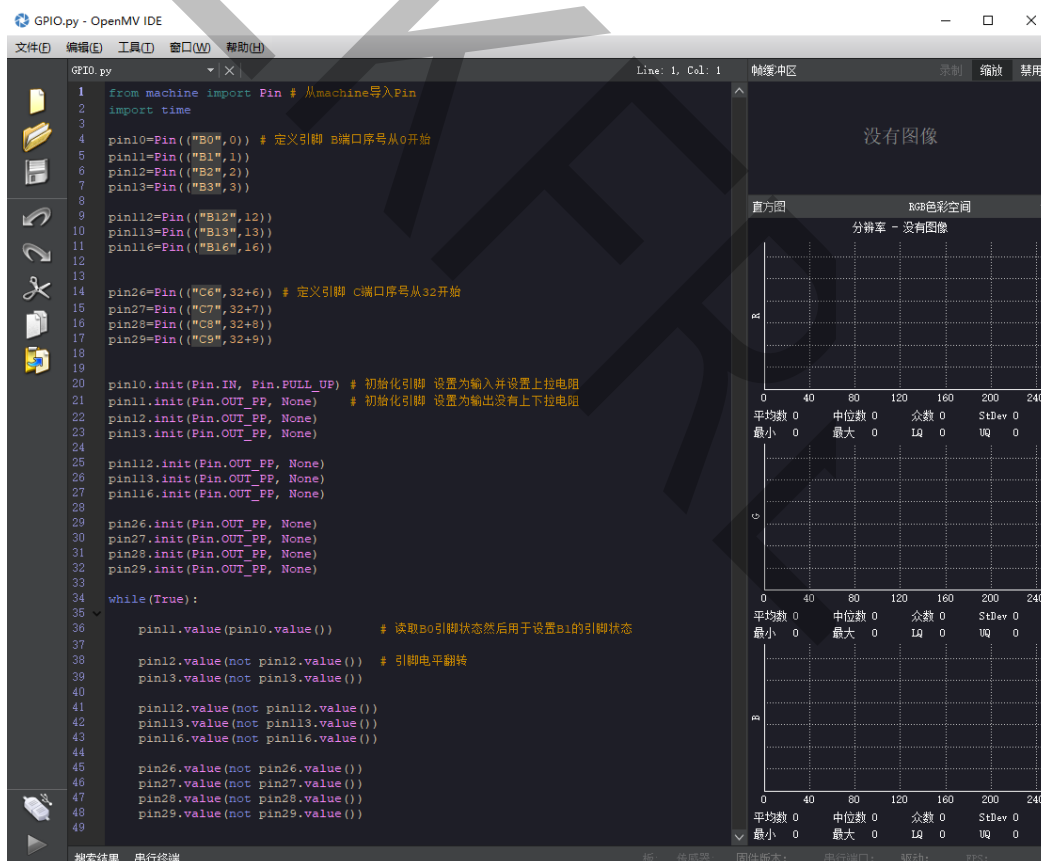
是用来创建 LED 对象的, 括号中可填的参数为 1、2、3、4, 分别对应 OpenART mini 的

红、绿、蓝、白灯;

D、`red.on()` #点亮红灯

`red.off()` #关闭红灯

2.2、GPIO 的控制



前面的步骤和 LED 控制操作一样, 用 OpenMV IDE 打开 GPIO.py 后, 就可以看到上图中的内容。这个程序运行后, 不能直观的看到现象。目的只是为了给出每个引脚的初始化定义以

及对引脚操作的示例。例如，我需要设置 OpenART mini 的 C7 引脚设置为输入并上拉，我们就可以根据上图第 20 行代码那样进行设置 `pin27.init(Pin.IN, Pin.PULL_UP)`。下面，我们把 LED 控制和 GPIO 控制结合起来，写一个示例。

```
1 import pyb
2 from machine import Pin # 从machine导入Pin
3 from pyb import LED #导入LED
4
5 PB0 = Pin("B0",0) # 定义引脚 B端口序号从0开始
6 PB0.init(Pin.OUT_PP, None)
7
8 # 函数功能：根据引脚电平，点亮或者关闭LED(4)
9 def led(pin):
10     if pin.value():
11         LED(4).on()
12     if not pin.value():
13         LED(4).off()
14
15 while(True):
16     PB0.value(1) # 设置PB0引脚为高电平
17     led(PB0)
18     pyb.mdelay(500) # 延时500ms
19
20     PB0.value(0) # 设置PB0引脚为低电平
21     led(PB0)
22     pyb.mdelay(1500) # 延时1500ms
```

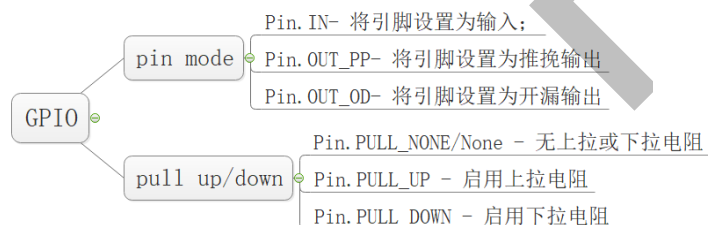
如上图所示，相信你看到注释后，应该可以理解这段程序所做的事情。按照上述代码写好后，点击运行，你就会看到 OpenART mini 上白色灯亮 0.5S，灭 1.5S，如此循环进行下去。

要点总结：

A、要控制 GPIO，必须加上下面句话：

```
from machine import Pin
```

B、引脚配置可选参数



C、设置引脚的高低电平

```
pin.value(1) # 设置引脚为高电平
```

```
pin.value(0) # 设置引脚为低电平
```

2.3、PWM 的控制

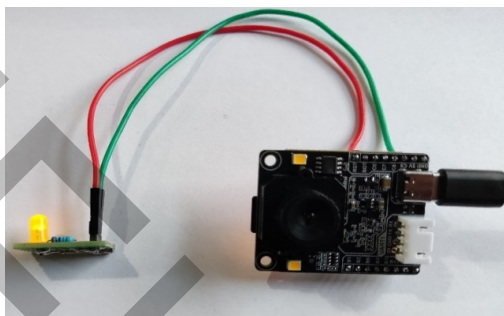


前面的步骤和上述几个例程操作一样，用 OpenMV IDE 打开 PWM.py 后，就可以看到上图中的内容。运行上述例程后，我们也无法直观看出什么现象，不过你身边如果有万用表和示波器，你可以测量对应的引脚（具体的引脚可以查看引脚说明章节，也可以看上图中的注释）电压或者波形，你会看到电压或者波形的变化。下面我们还是将例程稍作修改，写一个呼吸灯的程序，这样一来，初学者可以更方便地看到现象。

```
1 import pyb
2 from machine import PWM # 导入PWM
3
4
5 pwm1=PWM(2,1,2000,50) # 初始化pwm 使用pwm2 通道1 频率2khz 占空比数值设置为50(设置范围是0-255) PWM通过C6输出
6 pwm2=PWM(2,2,2000,100) # 使用pwm2通道2与使用pwm2通道1使用的是同一个定时器，因此两个通道频率必须一致，占空比可以不同
7
8 pwm3=PWM(2,3,3000,150) # 初始化pwm 使用pwm2 通道3 频率3khz 占空比数值设置为150(设置范围是0-255) PWM通过C8输出
9 pwm4=PWM(2,4,3000,200) # 使用pwm2通道3与使用pwm2通道4使用的是同一个定时器，因此两个通道频率必须一致，占空比可以不同
10
11 n = 255
12 while(n>0):
13     pwm1.duty(n) # 修改占空比 占空比设置范围0-255
14     pwm2.duty(n) # 修改占空比 占空比设置范围0-255
15     pwm3.duty(n)
16     pwm4.duty(n)
17     n = n-1 # 重新计算占空比 下次设置
18     pyb.mdelay(10) # 延时2ms
19     if n == 0:
20         while(n < 255):
21             pwm1.duty(n) # 修改占空比 占空比设置范围0-255
22             pwm2.duty(n) # 修改占空比 占空比设置范围0-255
23             pwm3.duty(n)
24             pwm4.duty(n)
25             n = n+1
26             pyb.mdelay(10) # 延时2ms
27
28 pwm1.deinit() # PWM2通道1反初始化 释放资源
29 pwm2.deinit()
30 pwm3.deinit()
31 pwm4.deinit()
32
```

根据上图，我们来进行简单的分析，由于 pwm1 和 pwm2 使用的是同一个时钟，所以要

求这两个通道频率一致，这一点也可以由引脚说明章节查看引脚定义得到印证，pwm1 对应的是 C6 (PWM2_M0_A) ,pwm2 对应的是 C7 (PWM2_M0_B) ,图中设置的是 2000hz。pwm3 和 pwm4 同理。后面的内容就比较简单了，就是设置四个通道的占空比。这个时候就可以用万用表或者示波器测量四个引脚的电压或者波形，又或者身边有 LED 的，就可以自己测试下效果。灯的正极接四个输出 PWM 的任意一个引脚，灯的负极接地，然后你就可以看到呼吸灯的效果了。



要点总结:

A、要控制 PWM，必须加上下面句话：

```
from machine import PWM
```

B、C6 和 C7 频率必须相同，占空比可以不同，C8 和 C9 频率必须相同，占空比可以不同；

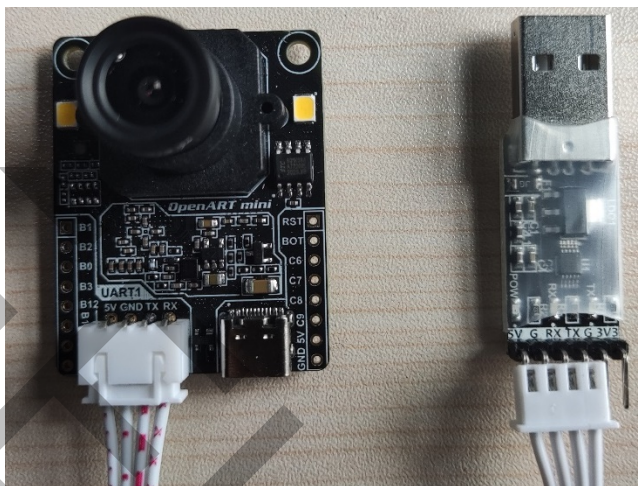
C、pwm.duty(n) 设置占空比为 n (n 的范围为 0-255)；

D、pwm.deinit() #释放资源

2.4、UART 的使用



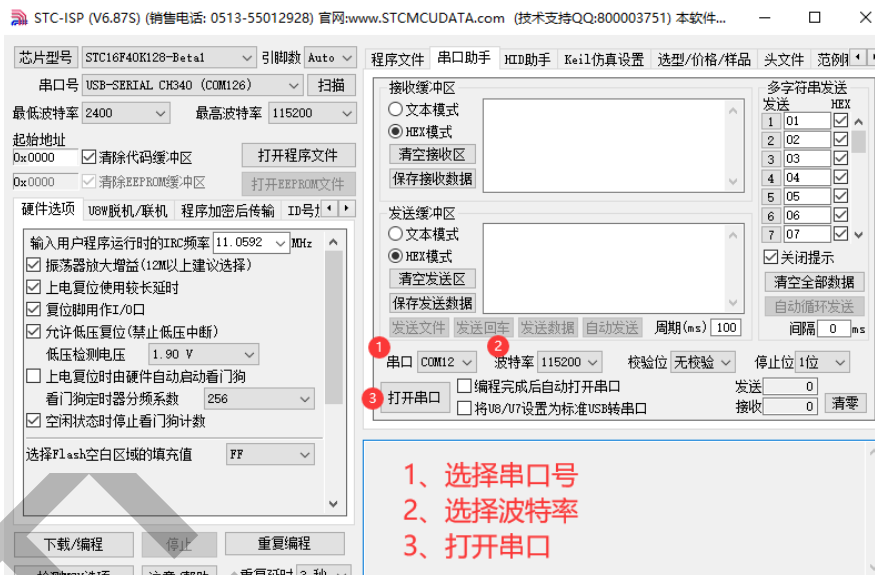
前面的步骤和上述几个例程操作一样，用 OpenMV IDE 打开 UART.py 后，就可以看到上图中的内容。由于是串口通信，这里我们就借助串口助手来进行调试。从代码我们可以看出来用的是 UART1,其对应的引脚是 B12、B13。OpenART mini 上特地为 UART1 引出了一个 XH2.54 接口座子，配备我们送的 XH2.54 端子线可以很方便的用来连接其他外设。



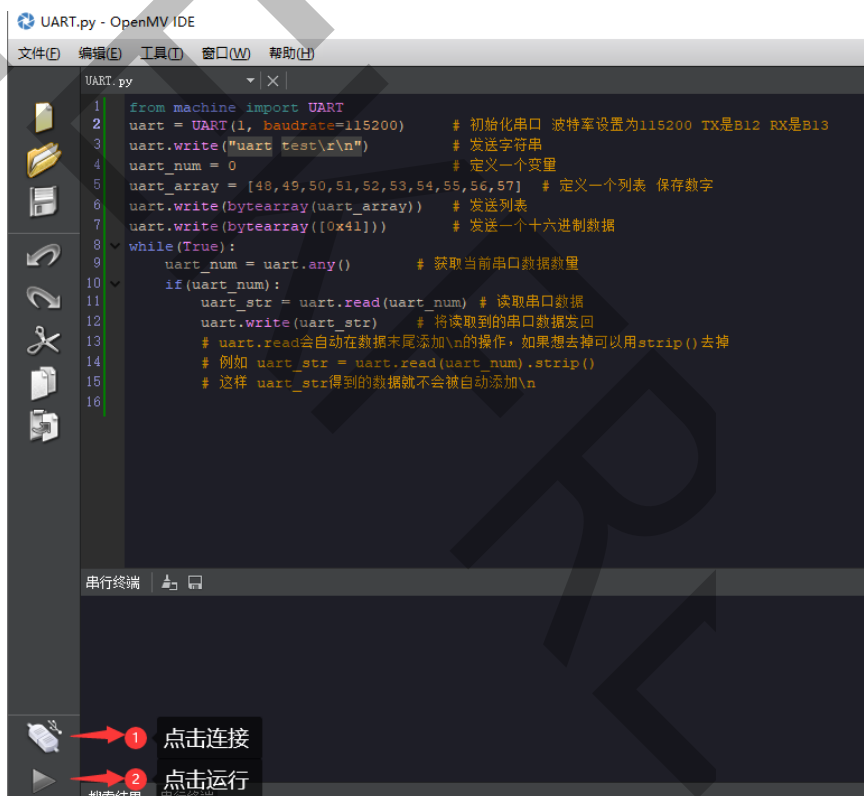
上图是 OpenART mini 接 USB 转 TTL 的接线示意图，这个示意图最主要的作用也就是为了说明串口的接线方式。OpenART mini 的 TX 必须接 USB 转 TTL 的 RX, OpenART mini 的 RX 必须接 USB 转 TTL 的 TX。如果 OpenART mini 接其他 MCU 进行串口通信的话，也需要注意这点。

接好线后，由于 XH2.54 接口座子这里也有 5V 电源的接口，USB 转 TTL 接上电源后也会对 OpenART mini 进行供电。但有可能因为 USB 转 TTL 供电能力不足等问题，导致 OpenART mini 未能正常初始化启动，进而导致串口通信不成功。所以我们推荐先插上 Type-C 接口给 OpenART mini 供电，然后再把 USB 转 TTL 与电脑连接。

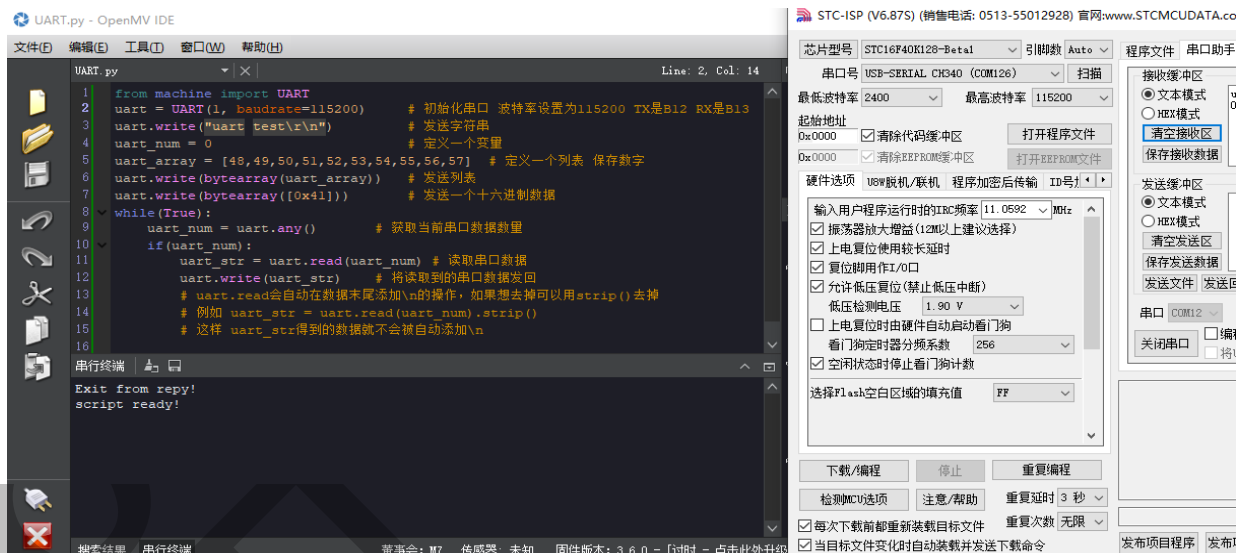
打开任意一个串口助手，点击如下图所示：



连接 OpenART mini, 点击运行, 如下图所示:



点击运行后, 串口助手会接收到数据, 数据内容就是代码第 3 行的“uart test”, 以及第 6 行的列表, 第 7 行的十六进制数, 如下图所示:



从上图可以看到，串口助手的接收缓冲区里收到了“uart test”、“0123456789”、“A”，这分别对应 3、6、7 行代码所发送的内容。

我们再来测试 OpenART mini 的接收，我们在串口助手发送缓冲区里输入要发送的内容，然后点击发送数据，如下图所示：



从上图可以看到，缓冲接收区接收到了我发送的内容。让我们看看代码中 while 循环里做了什么事。首先获取串口，然后读取串口数据，最后将读取到的数据又发送出去，这也是为什么我发送了“hello OpenART mini”接收缓冲区会接收到同样内容的原因。

要点总结：

A、要使用串口通信，必须加上下面这句话：

```
from machine import UART
```

B、串口通信要注意收发两方的波特率保持一致；

2.5、DEBUG_UART 的使用

DEBUG_UART 实际上是 OpenART mini 提供的 REPL（交互式解释器）环境，其相应的引脚是 C8、C9，具体的引脚定义可以去查看引脚说明章节，这里不再赘述。下面我们就来讲讲如何使用这个 REPL 环境。

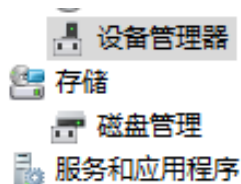
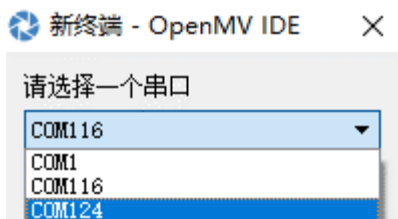
首先，先将 C8 接到 USB 转 TTL 的 RX，C9 接到 USB 转 TTL 的 TX，然后我们把 OpenMV IDE 打开，点击连接，再打开工具栏选择打开终端，如下图所示：



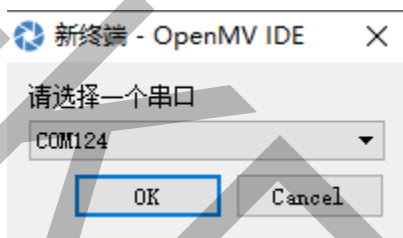
如下图所示，点击 OK：



选择串口，这里要注意下，COM116 是 OpenART mini 接到我电脑的端口号，COM124 才是 USB 转 TTL 的，也就是 DEBUG_UART 的端口号。具体使用的时候，你们可以通过设备管理器来查看自己设备的端口号，如下图所示：



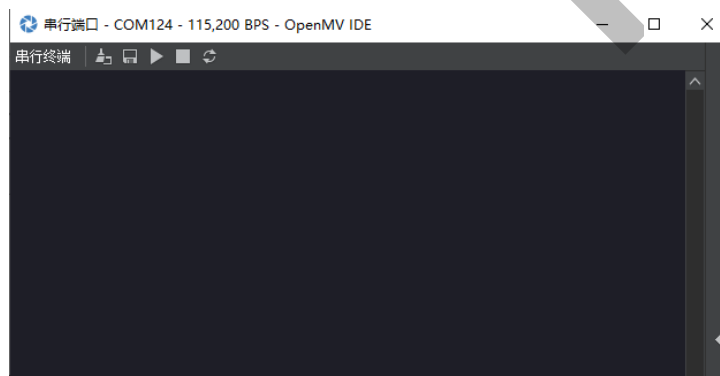
选择 COM124 后，点击 OK:



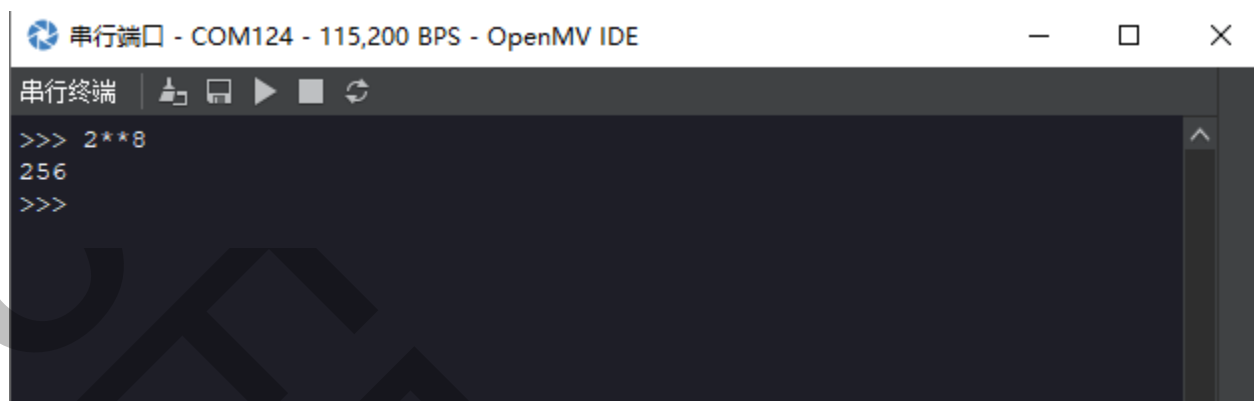
选择波特率，点击 OK:



然后会出现我们新建的串行端口窗口，如下图所示:



这个时候，我们就可以使用 REPL 环境了，先在串行端口敲个回车，你会看到窗口显示“>>>”，这个时候我们就可以开始执行我们需要的操作了。例如，我们想计算 2^8 ，先输入 $2^{**}8$ ，再敲下回车，你就会发现它会把结果计算出来，如下图所示：



```

串行端口 - COM124 - 115,200 BPS - OpenMV IDE
串行终端
>>> 2**8
256
>>>

```

再比如，我们来点亮红色 LED。和前面 LED 控制讲的一样，首先要加载必要的依赖模块，所以先输入 `import pyb` 敲下回车，再输入 `from pyb import LED` 敲下回车，最后再输入 `LED(1).on()`，敲下回车，你就会看到 OpenART mini 上亮起了红灯。如下图所示：



```

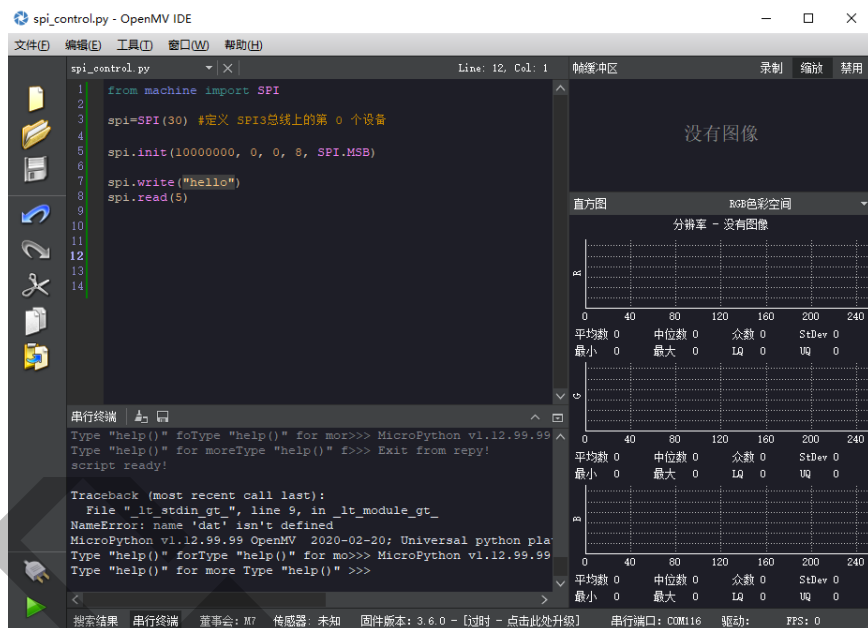
串行端口 - COM124 - 115,200 BPS - OpenMV IDE
串行终端
>>> 2**8
256
>>> import pyb
>>> from pyb import LED
>>> LED(1).on()
>>>

```

再输入 `LED(1).off()` 敲下回车，你又会看到红灯被关闭。相信到这里你已经对这个 REPL 环境有所了解了，后面就请结合自己的需求来使用吧！

2.6、SPI 的使用

先给出个简单 SPI 的示例，围绕这个示例，我们来讲解相关的内容，示例如下图所示：



1、先导入 SPI 所依赖的模块，如上图第 1 行所示。

2、创建对象 `spi = SPI(30)`。此处用的是硬件 SPI，对照引脚定义我们可以知道 B0~B3 为 SPI3 相关的引脚，其中“30”即是表示 SPI3 总线上的第 0 个设备。

3、初始化 SPI 总线,对应第 5 行代码：

`spi.init(baudrate, polarity, phase, bits, firstbit)`

第一个参数是波特率，代码中设置的是 100000000；

第二个参数是极性，可以是 0 或 1，指的是时钟空闲时所处的电平；

第三个参数是相位，可以是 0 或 1，用于指定在第一个或者第二个时钟边缘采集数据；

第四个参数是每次传输的数据长度，一般是 8 位；

第五个参数是用于指定传输数据是从高位开始还是从低位开始，可以是 SPI.MSB(高位传输) 或者 SPI.LSB（低位传输）；

4、`spi.write("hello")` #spi 写入“hello”；

5、`spi.read(5)` #读取 5 字节,返回读出的字节对象；

有了这些基础，我们来写一个 0.96 寸的 oled 屏幕的驱动程序,如下图所示：

```

1  from machine import SPI,Pin
2  import time
3
4  cs = Pin("B3", 3))    #引脚定义 OLED CS引脚接B3
5  rst = Pin("B12",12))  #引脚定义 OLED RES引脚接B12
6  dc = Pin("B13",13))  #引脚定义 OLED DC引脚接B13
7
8  dc.init(Pin.OUT_PP, Pin.PULL_NONE) #引脚初始化, 方向: 输出 无上拉
9  rst.init(Pin.OUT_PP, Pin.PULL_NONE) #引脚初始化, 方向: 输出 无上拉
10 cs.init(Pin.OUT_PP, Pin.PULL_NONE) #引脚初始化, 方向: 输出 无上拉
11
12 #字库6*8
13 asc = { ... }
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78 spi = SPI(30) #创建对象 SPI3总线上的第0个设备
79 spi.init(100000,0,0,8,SPI.MSB)#初始化 波特率100000, 极性0, 相位0, 传输数据长度8位, 从高位开始传输数据
80
81 #写命令
82 def write_command_byte(c):
83     c = c.to_bytes(1,'little')
84     cs.value(0)
85     dc.value(0)
86     spi.write(c)
87     cs.value(1)
88
89 #写数据
90 def write_data_byte(c):
91     c = c.to_bytes(1,'little')
92     cs.value(0)
93     dc.value(1)
94     spi.write(c)
95     cs.value(1)
96
97 #写命令
98 def write_command(c, *data):
99     write_command_byte(c)
100     if data:
101         for d in data: write_data_byte(d)
102
103 #设置坐标
104 def OLED_Set_Pos(x,y):
105     write_command(0xb0+y);
106     write_command(((x&0xf0)>>4)|0x10)
107     write_command((x&0x0f)|0x01)
108
109 #清屏
110 def Clear():
111     i = 0
112     while(i<8):
113         write_command(0xb0+i)
114         i = i + 1
115         write_command(0x00)
116         write_command(0x10)
117         n = 128
118         while(n>0):
119             write_data_byte(0)
120             n = n -1
121
122 #显示字符
123 def OLED_ShowChar(x,y,s):
124     OLED_Set_Pos(x,y+1)
125     for d in asc[s]:
126         write_data_byte(d)
127
128

```

```

127 #OLED初始化
128 def init():
129     rst.value(0)
130     time.sleep(100)
131     rst.value(1)
132     time.sleep(100)
133     write_command(0xAE)
134     write_command(0x00)
135     write_command(0x10)
136     write_command(0x40)
137     write_command(0x81)
138     write_command(0xCF)
139     write_command(0xA1)
140     write_command(0xC8)
141     write_command(0xA6)
142     write_command(0xA8)
143     write_command(0x3f)
144     write_command(0xD3)
145     write_command(0x00)
146     write_command(0xd5)
147     write_command(0x80)
148     write_command(0xD9)
149     write_command(0xF1)
150     write_command(0xDA)
151     write_command(0x12)
152     write_command(0xDB)
153     write_command(0x40)
154     write_command(0x20)
155     write_command(0x02)
156     write_command(0x8D)
157     write_command(0x14)
158     write_command(0xA4)
159     write_command(0xA6)
160     write_command(0xAF)
161
162     write_command(0xAF)
163     Clear()
164
165
166 init()
167 while(True):
168     OLED_ShowChar(0,0,"O")
169     OLED_ShowChar(8,0,"p")
170     OLED_ShowChar(16,0,"e")
171     OLED_ShowChar(24,0,"n")
172     OLED_ShowChar(32,0,"A")
173     OLED_ShowChar(40,0,"R")
174     OLED_ShowChar(48,0,"T")
175     OLED_ShowChar(64,0,"m")
176     OLED_ShowChar(72,0,"i")
177     OLED_ShowChar(80,0,"n")
178     OLED_ShowChar(88,0,"i")
179     OLED_ShowChar(104,0,"6")
180     OLED_ShowChar(112,0,"6")
181     OLED_ShowChar(120,0,"6")
182

```

部分字库截图：

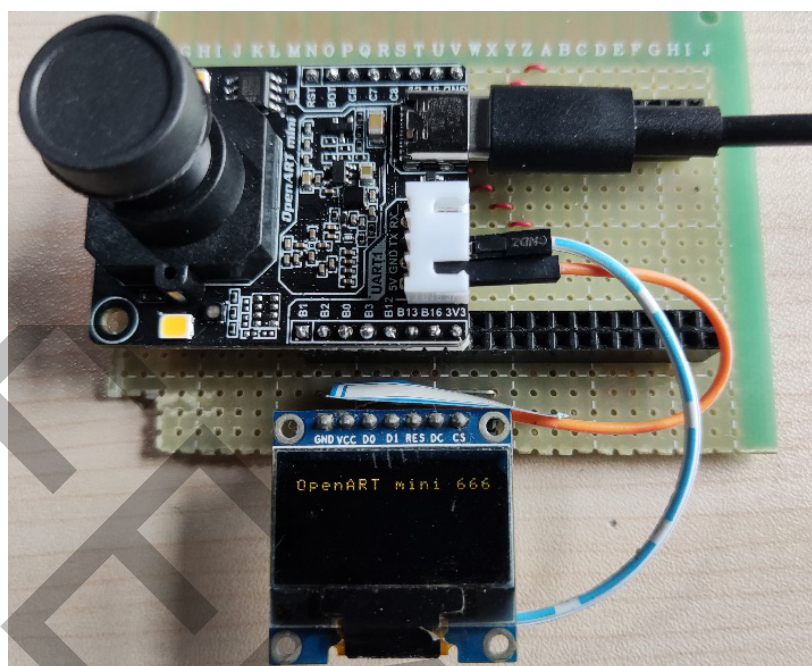
```

13 asc = {
14     "0": [0x00, 0x3E, 0x51, 0x49, 0x45, 0x3E],
15     "1": [0x00, 0x00, 0x42, 0x7F, 0x40, 0x00],
16     "2": [0x00, 0x42, 0x61, 0x51, 0x49, 0x46],
17     "3": [0x00, 0x21, 0x41, 0x45, 0x4B, 0x31],
18     "4": [0x00, 0x18, 0x14, 0x12, 0x7F, 0x10],
19     "5": [0x00, 0x27, 0x45, 0x45, 0x45, 0x39],
20     "6": [0x00, 0x3C, 0x4A, 0x49, 0x49, 0x30],
21     "7": [0x00, 0x01, 0x71, 0x09, 0x05, 0x03],
22     "8": [0x00, 0x36, 0x49, 0x49, 0x49, 0x36],
23     "9": [0x00, 0x06, 0x49, 0x49, 0x29, 0x1E],

```


上面几张图片就是用 SPI 驱动 OLED 的程序，具体的时序可以自己参照 OLED 的资料。

下图就是最终的显示效果：



3、OpenART mini 运行模型

模型的训练以及运行模型测试这部分内容可以查看《AI 视觉组新手入门教程》，里面详细介绍了相关环境的安装部署以及模型运行的操作步骤，所以这里就不再赘述这些内容。资料可以关注逐飞科技微信公众号，在公众号里找到相关推文《恩智浦 AI 视觉组浅析》、《恩智浦 AI 视觉组入门教程发布》，或者在 [OpenART mini 淘宝详情页](#)里找到资料。



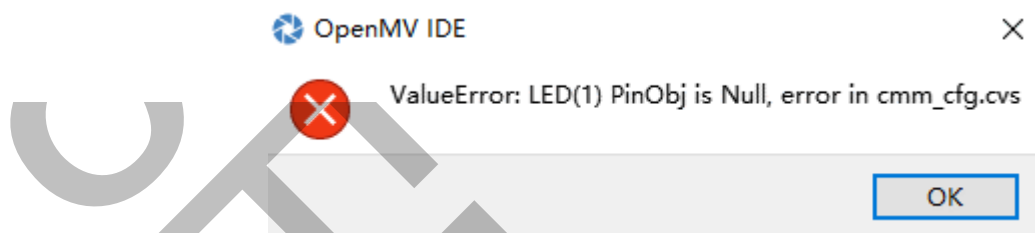
微信公众号二维码



淘宝店铺二维码

六、附录-常见问题及注意事项

1、出现类似这种错误，就是没有找到引脚定义的配置文件，请查看是否插入了 SD 卡或者插入的 SD 卡里是否放置了 SD 卡的必备文件，如果没有放入，请参考第五章节的 1.2 小节的说明。



2、模型加载错误地放入了循环体内，如下图所示：

```

40
41 while(True):
42     net = nncu.load(net_path, load_to_fb=True)           # 加载模型
43     clock.tick() # Track elapsed milliseconds between snapshots().
44     img = sensor.snapshot() # Capture snapshot.
45

```

这样使用会导致程序异常。

3、OpenART mini 用 IDE 运行代码的时候，尽量 SD 卡里面不要放 main.py 文件；

4、放入文件之后，最好先重启 OpenART mini，再使用 OpenMV IDE 连接模块运行程序；

文档版本

版本号	日期	内容变更
V1.0	2021-04-15	初始版本