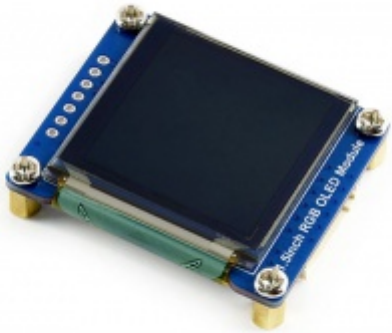
1.5inch RGB OLED Module

来自Waveshare Wiki
跳转至： 导航、 搜索



功能简介	
1.5英寸	
128×128	
OLED	SPI

产品概述

本产品提供树莓派、STM32、arduino例程

产品特性

- 工作电压 : 3.3V / 5V **（请保证供电电压和逻辑电压一致，否则会导致无法正常工作）**
- 支持接口 : 4-wire SPI、3-wire SPI
- 驱动芯片 : SSD1351
- 分辨率 : 128(H)RGB x128(V)
- 显示尺寸： 26.855（H）x 26.855（V）mm
- 像素大小： 0.045（H）x 0.194（V）mm
- 显示颜色 : 65K彩色

管脚配置

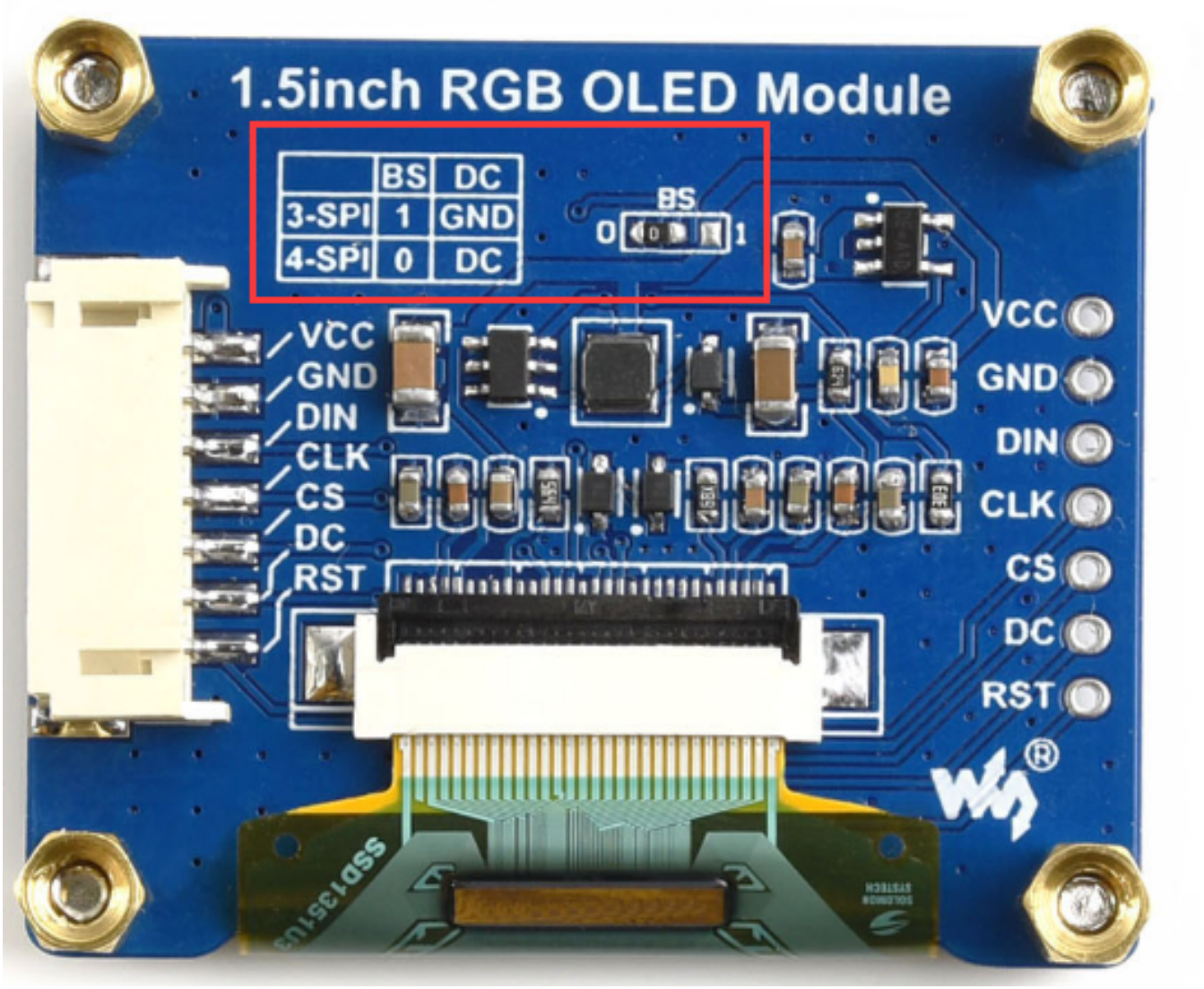
功能引脚	描述
VCC	电源正
GND	电源地
DIN	数据输入
CLK	时钟输入
CS	片选，低电平有效
DC	数据/命令信号选择，低电平表示命令，高电平表示数据
RST	复位信号，低电平有效

PS：本模块只有SPI接口，使用时请引起注意

硬件配置

- 本OLED模块提供两种通信方式：4-wire SPI和3wire-SPI
- 在模块的背面有一个可选择焊接的电阻，通过该电阻来选择通信方式。

如下图：



模块出厂默认使用4线SPI通信，即BS0默认接0

注：下表为接口连接。

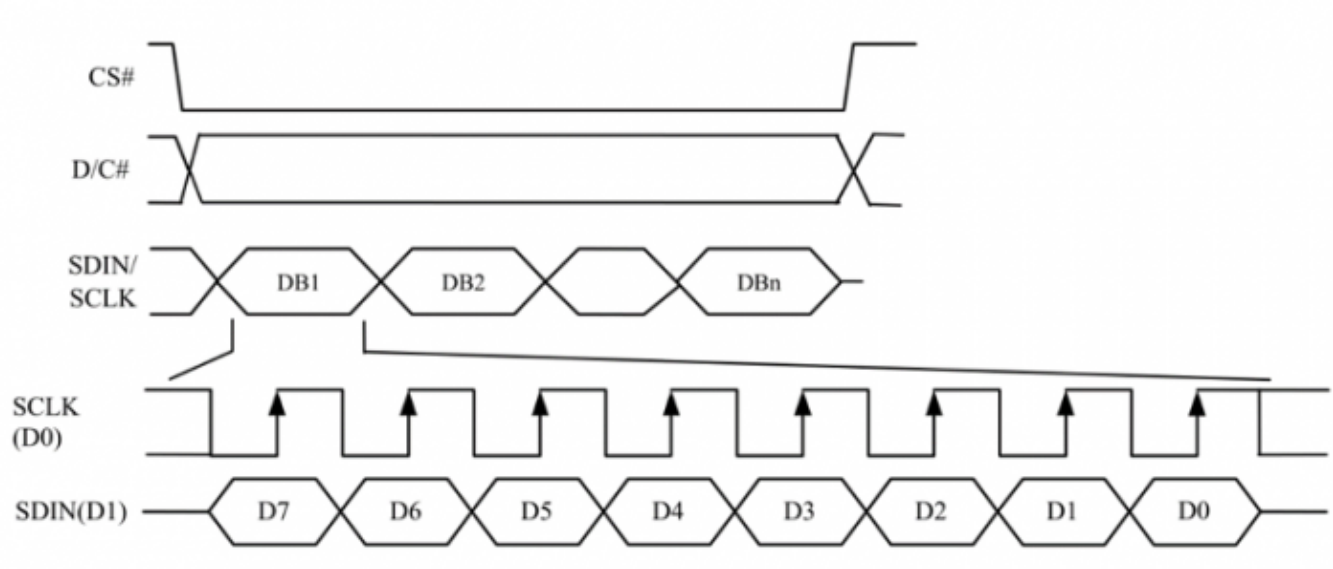
通信协议	BS	CS	DC	DIN	CLK
4Wire SPI	0	片选	DC	MOSI	SCK
3Wire SPI	1	片选	GND	MOSI	SCK

OLED 及其控制器

本款OLED使用的内置驱动器为SSD1351,其是一款128RGB * 128 Dot Matrix OLED/PLED 控制器，内部有一个128*128*18bit的SRAM作为显示缓存区，支持262k和65k两种颜色深度。并支持8bit 8080并行、8bit 6800并行、3wire-SPI和4wire-SPI等通信方式。

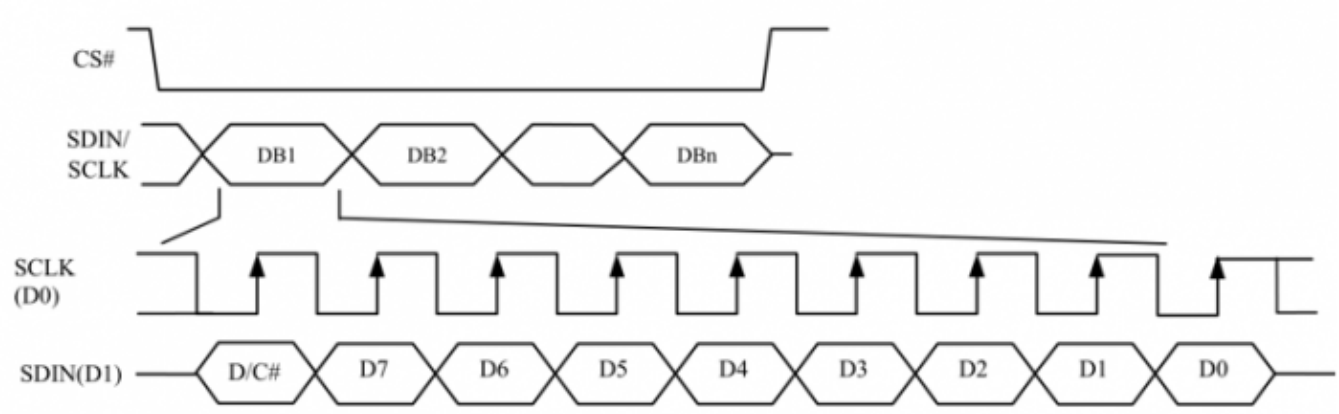
本模块选择了4wire-SPI和3wire-SPI作为通信方式，减小模块的面积，同时节省了控制器的IO资源。

4WIRE-SPI通信协议



- 在4wire-SPI通信中，先将DC置1或置0，再发送一个或多个字节的数据。
- 当DC置0时，发送的字节将作为控制OLED的命令，发送命令时，一般一次只发送一个字节。
- 当DC置1时，发送的字节将作为数据存入SSD1351的指定的寄存器或者SRAM。在发送数据时，可以连续发送多个字节。
- 详见SSD1351 Datasheet Figure 8-5。

3WIRE-SPI通信协议



- 3wire-SPI和4wire-SPI唯一的区别在于，它去掉了控制发送命令和数据的DC引脚
- 在每次SPI传输的字节前加一个位来识别该字节是命令还是数据。
- 故在3wire-SPI中，DC引脚需要接地，此外，每次传输的数据不是8bit，而是9bit。

取模设置

参考设置如图所示：水平扫描，16位真彩色，高位在前

树莓派软件说明

提供BCM2835、WiringPi、文件IO、RPI（Python）库例程

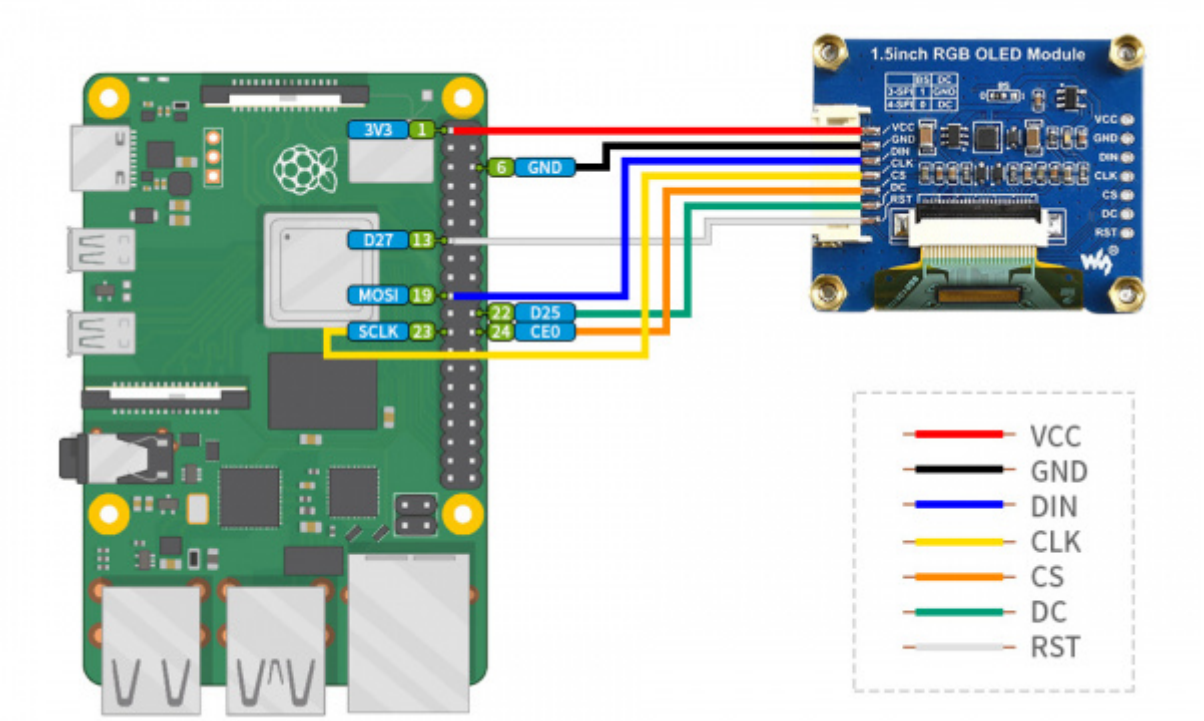
硬件连接

连接树莓派的时候，选择用7PIN排线连接，请参考下方的引脚对应表格

树莓派连接引脚对应关系

OLED	Raspberry Pi	
	BCM2835编码	Board物理引脚序号
VCC	3.3V	3.3V
GND	GND	GND
DIN	SPI:D10(MOSI) / I2C:D2	SPI:MOSI / I2C:SDA.1
CLK	SPI:D11(SCLK) / I2C:D3	SPI:SCLK / I2C:SCL.1
CS	D8(CE0)	CE0
DC	D25	GPIO.6
RST	D27	GPIO.2

- 四线SPI接线图



开启SPI和I2C接口

- 打开树莓派终端，输入以下指令进入配置界面

```
sudo raspi-config
选择Interfacing Options -> SPI -> Yes 开启SPI接口
```

```
1 Change User Password Change password for the current user
2 Network Options      Configure network settings
3 Boot Options         Configure options for start-up
4 Localisation Options Set up language and regional settings to match your location
5 Interfacing Options  Configure connections to peripherals
6 Overclock            Configure overclocking for your Pi
7 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool
```

```
P1 Camera      Enable/Disable connection to the Raspberry Pi Camera
P2 SSH         Enable/Disable remote command line access to your Pi using SSH
P3 VNC         Enable/Disable graphical remote access to your Pi using RealVNC
P4 SPI         Enable/Disable automatic loading of SPI kernel module
P5 I2C         Enable/Disable automatic loading of I2C kernel module
P6 Serial      Enable/Disable shell and kernel messages on the serial connection
P7 1-Wire      Enable/Disable one-wire interface
P8 Remote GPIO Enable/Disable remote access to GPIO pins
```

```
Would you like the SPI interface to be enabled?

                    <Yes>                    <No>
```

然后重启树莓派：

```
sudo reboot
```

I2C同理，进入配置界面选择Interfaceing Options -> I2C -> Yes 开启IIC接口，然后重启

安装库

BCM2835

```
#打开树莓派终端，并运行以下指令
wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.71.tar.gz
tar zxvf bcm2835-1.71.tar.gz
cd bcm2835-1.71/
sudo ./configure && sudo make && sudo make check && sudo make install
# 更多的可以参考官网：http://www.airspayce.com/mikem/bcm2835/
```

wiringPi

```
#打开树莓派终端，并运行以下指令
sudo apt-get install wiringpi
#对于树莓派2019年5月之后的系统（早于之前的可不用执行），可能需要进行升级：
wget https://project-downloads.drogon.net/wiringpi-latest.deb
sudo dpkg -i wiringpi-latest.deb
gpio -v
# 运行gpio -v会出现2.52版本，如果没有出现说明安装出错

#Bullseye分支系统使用如下命令：
git clone https://github.com/WiringPi/WiringPi
cd WiringPi
./build
gpio -v
# 运行gpio -v会出现2.60版本，如果没有出现说明安装出错
```

- 安装Python函数库

```
#python2
sudo apt-get update
sudo apt-get install python-pip
sudo apt-get install python-pil
sudo apt-get install python-numpy
sudo pip install RPi.GPIO
sudo pip install spidev
#python3
sudo apt-get update
sudo apt-get install python3-pip
sudo apt-get install python3-pil
sudo apt-get install python3-numpy
sudo pip3 install RPi.GPIO
sudo pip3 install spidev
```

下载测试程序

打开树莓派终端，执行：

```
sudo apt-get install p7zip-full
sudo wget  https://www.waveshare.net/w/upload/2/2c/OLED_Module_Code.7z
7z x OLED_Module_Code.7z -O./OLED_Module_Code
cd OLED_Module_Code/RaspberryPi
```

运行测试程序

以下命令请在RaspberryPi目录下执行，否则索引不到目录；

C语言

- 重新编译，编译过程可能需要几秒

```
cd c
sudo make clean
sudo make -j 8
```

所有屏幕的测试程序，可以直接通过输入对应的尺寸进行调用：

```
sudo ./main 屏幕尺寸
```

根据不同LCD，应当输入以下某一条指令：

```
#0.91inch OLED Module
sudo ./main 0.91
#0.95inch RGB OLED (A)/(B)
sudo ./main 0.95rgb
#0.96inch OLED (A)/(B)
sudo ./main 0.96
#1.3inch OLED (A)/(B)
sudo ./main 1.3
#1.3inch OLED Module (C)
sudo ./main 1.3c
#1.5inch OLED Module
sudo ./main 1.5
#1.5inch RGB OLED Module
sudo ./main 1.5rgb
```

Python

- 进入python程序目录

```
cd python/example
```

- **运行对应型号OLED的例程**，程序支持python2/3

假如你购买了1.3inch OLED Module （C），请输入：

```
# python2
sudo python OLED_1in3_c_test.py
# python3
sudo python3 OLED_1in3_c_test.py
```

假如你购买了1.5inch RGB OLED Module，请输入：

```
# python2
sudo python OLED_1in5_rgb_test.py
# python3
sudo python3 OLED_1in5_rgb_test.py
```

- 型号指令对应表

```
#0.91inch OLED Module
sudo python OLED_0in91_test.py
#0.95inch RGB OLED (A)/(B)
sudo python OLED_0in95_rgb_test.py
#0.96inch OLED (A)/(B)
sudo python OLED_0in96_test.py
#1.3inch OLED (A)/(B)
sudo python OLED_1in3_c_test.py
#1.3inch OLED Module (C)
sudo python OLED_1in3_test.py
#1.5inch OLED Module
sudo python OLED_1in5_test.py
#1.5inch RGB OLED Module
sudo python OLED_1in5_rgb_test.py
```

- 请确保SPI没有被其他的设备占用，你可以在/boot/config.txt中间检查

C语言部分

底层接口简析

C语言使用了3种方式进行驱动：分别是BCM2835库、WiringPi库和Dev库
默认使用Dev库进行操作，如果你需要使用BCM2835或者WiringPi来驱动的话，可以打开RaspberryPi\c\Makefile，修改13-15行，如下：

```
#USELIB = USE_BCM2835_LIB
#USELIB = USE_WIRINGPI_LIB
USELIB = USE_DEV_LIB
DEBUG = -D $(USELIB)
ifeq ($(USELIB), USE_BCM2835_LIB)
    LIB = -lbcm2835 -lm
else ifeq ($(USELIB), USE_WIRINGPI_LIB)
    LIB = -lwiringPi -lm
else ifeq ($(USELIB), USE_DEV_LIB)
    LIB = -lpthread -lm
endif
```

我们进行了底层的封装，由于硬件平台不一样，内部的实现是不一样的，如果需要了解内部实现可以去对应的目录中查看

在DEV_Config.c(.h)可以看到很多定义，在目录：RaspberryPi\c\lib\Config

- 接口选择：

```
#define USE_SPI_4W  1
#define USE_IIC     0
注意：切换SPI/I2C直接修改这里
```

- 数据类型：

```
#define UBYTE      uint8_t
#define UWORD      uint16_t
#define UDOUBLE    uint32_t
```

- 模块初始化与退出的处理：

```
void DEV_Module_Init(void);
void DEV_Module_Exit(void);
注意：
1.这里是处理使用液晶屏前与使用完之后一些GPIO的处理。
```

- 写GPIO

```
void DEV_Digital_Write(UWORD Pin, UBYTE Value)
参数：
    UWORD Pin：GPIO引脚号
    UBYTE Value：要输出的电平，0或1
```

- 读GPIO

```
UBYTE DEV_Digital_Read(UWORD Pin)
参数：
    UWORD Pin：GPIO引脚号
返回值：
    GPIO的的电平，0或1
```

- GPIO模式设置

```
void DEV_GPIO_Mode(UWORD Pin, UWORD Mode)
参数：
    UWORD Pin：GPIO引脚号
    UWORD Mode：模式，0：输入，1：输出
```

GUI简析

对于屏幕而言，如果需要进行画图、显示中英文字符、显示图片等怎么办，这些都是上层应用做的。这有很多小伙伴有问到一些图形的处理，我们这里提供了一些基本的功能 在目录：RaspberryPi\c\lib\GUI\GUI_Paint.c(.h)中可以找到GUI

名称	修改日期	类型	大小
GUI_BMP.c	2020/6/8 14:59	C 文件	5 KB
GUI_BMP.h	2020/6/5 10:58	H 文件	3 KB
GUI_Paint.c	2020/6/16 17:18	C 文件	31 KB
GUI_Paint.h	2020/6/16 17:23	H 文件	6 KB

在目录：RaspberryPi\c\lib\Fonts下是GUI依赖的字符字体，

名称	修改日期	类型	大小
font8.c	2020/5/20 11:58	C 文件	18 KB
font12.c	2020/5/20 11:58	C 文件	27 KB
font12CN.c	2020/6/5 18:57	C 文件	6 KB
font16.c	2020/5/20 11:58	C 文件	49 KB
font20.c	2020/5/20 11:58	C 文件	65 KB
font24.c	2020/5/20 11:58	C 文件	97 KB
font24CN.c	2020/6/5 19:01	C 文件	28 KB
fonts.h	2020/5/20 11:58	H 文件	4 KB

- 新建图像属性:新建一个图像属性，这个属性包括图像缓存的名称、宽度、高度、翻转角度、颜色

```
void Paint_NewImage(UBYTE *image, UWORD Width, UWORD Height, UWORD Rotate, UWORD Co
lor)
参数：
        image ： 图像缓存的名称，实际上是一个指向图像缓存首地址的指针；
        Width ： 图像缓存的宽度；
        Height: 图像缓存的高度；
        Rotate：图像的翻转的角度
        Color ： 图像的初始颜色；
```

- 选择图像缓存:选择图像缓存，选择的目的是你可以创建多个图像属性，图像缓存可以存在多个，你可以选择你所创建的每一张图像

```
void Paint_SelectImage(UBYTE *image)
参数：
        image: 图像缓存的名称，实际上是一个指向图像缓存首地址的指针；
```

- 图像旋转:设置选择好的图像的旋转角度，最好使用在Paint_SelectImage()后，可以选择旋转0、90、180、270

```
void Paint_SetRotate(UWORD Rotate)
参数：
        Rotate: 图像选择角度，可以选择ROTATE_0、ROTATE_90、ROTATE_180、ROTATE_270分别对
应0、90、180、270度
```

- 设置像素点的尺寸

```
void Paint_SetScale(UBYTE scale)
参数：
        scale: 像素点的尺寸，2：每个像素点占一位；4：每个像素点占两位
```

- 图像镜像翻转:设置选择好的图像的镜像翻转，可以选择不镜像、关于水平镜像、关于垂直镜像、关于图像中心镜像。

```
void Paint_SetMirroring(UBYTE mirror)
参数：
        mirror: 图像的镜像方式，可以选择MIRROR_NONE、MIRROR_HORIZONTAL、MIRROR_VERTICA
L、MIRROR_ORIGIN分别对应不镜像、关于水平镜像、关于垂直镜像、关于图像中心镜像
```

- 设置点在缓存中显示位置和颜色：这里是GUI最核心的一个函数、处理点在缓存中显示位置和颜色；

```
void Paint_SetPixel(UWORD Xpoint, UWORD Ypoint, UWORD Color)
参数：
        Xpoint：点在图像缓存中X位置
        Ypoint：点在图像缓存中Y位置
        Color ：点显示的颜色
```

- 图像缓存填充颜色:把图像缓存填充为某颜色，一般作为屏幕刷白的作用

```
void Paint_Clear(UWORD Color)
参数：
        Color：填充的颜色
```

- 图像缓存部分窗口填充颜色：把图像缓存的某部分窗口填充为某颜色，一般作为窗口刷白的作用，常用于时间的显示，刷白上一秒

```
void Paint_ClearWindows(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD C
olor)
参数：
        Xstart：窗口的X起点坐标
        Ystart：窗口的Y起点坐标
        Xend：窗口的X终点坐标
        Yend：窗口的Y终点坐标
        Color：填充的颜色
```

- 画点:在图像缓存中，在（Xpoint, Ypoint）上画点，可以选择颜色，点的大小，点的风格

```
void Paint_DrawPoint(UWORD Xpoint, UWORD Ypoint, UWORD Color, DOT_PIXEL Dot_Pixel,
DOT_STYLE Dot_Style)
参数：
        Xpoint: 点的X坐标
        Ypoint: 点的Y坐标
        Color: 填充的颜色
        Dot_Pixel: 点的大小，提供默认的8种大小点
                 typedef enum {
                        DOT_PIXEL_1X1  = 1,    // 1 x 1
                        DOT_PIXEL_2X2  ,               // 2 X 2
                        DOT_PIXEL_3X3  ,               // 3 X 3
                        DOT_PIXEL_4X4  ,               // 4 X 4
                        DOT_PIXEL_5X5  ,               // 5 X 5
                        DOT_PIXEL_6X6  ,               // 6 X 6
                        DOT_PIXEL_7X7  ,               // 7 X 7
                        DOT_PIXEL_8X8  ,               // 8 X 8
                } DOT_PIXEL;
        Dot_Style: 点的风格,大小扩充方式是以点为中心扩大还是以点为左下角往右上扩大
                typedef enum {
                   DOT_FILL_AROUND  = 1,
                   DOT_FILL_RIGHTUP,
                } DOT_STYLE;
```

- 画线：在图像缓存中，从 (Xstart, Ystart) 到 (Xend, Yend) 画线，可以选择颜色，线的宽度，线的风格

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Colo
r, LINE_STYLE Line_Style , LINE_STYLE Line_Style)
参数：
        Xstart: 线的X起点坐标
        Ystart: 线的Y起点坐标
        Xend: 线的X终点坐标
        Yend: 线的Y终点坐标
        Color: 填充的颜色
        Line_width: 线的宽度，提供默认的8种宽度
                typedef enum {
                        DOT_PIXEL_1X1  = 1,    // 1 x 1
                        DOT_PIXEL_2X2  ,               // 2 X 2
                        DOT_PIXEL_3X3  ,               // 3 X 3
                        DOT_PIXEL_4X4  ,               // 4 X 4
                        DOT_PIXEL_5X5  ,               // 5 X 5
                        DOT_PIXEL_6X6  ,               // 6 X 6
                        DOT_PIXEL_7X7  ,               // 7 X 7
                        DOT_PIXEL_8X8  ,               // 8 X 8
                } DOT_PIXEL;
         Line_Style: 线的风格,选择线是以直线连接还是以虚线的方式连接
                typedef enum {
                        LINE_STYLE_SOLID = 0,
                        LINE_STYLE_DOTTED,
                } LINE_STYLE;
```

- 画矩形:在图像缓存中，从 (Xstart, Ystart) 到 (Xend, Yend) 画一个矩形，可以选择颜色，线的宽度，是否填充矩形内部

```
void Paint_DrawRectangle(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD
Color, DOT_PIXEL Line_width, DRAW_FILL Draw_Fill)
参数:
        Xstart: 矩形的X起点坐标
        Ystart: 矩形的Y起点坐标
        Xend: 矩形的X终点坐标
        Yend: 矩形的Y终点坐标
        Color: 填充的颜色
        Line_width: 矩形四边的宽度，提供默认的8种宽度
                typedef enum {
                        DOT_PIXEL_1X1  = 1,    // 1 x 1
                        DOT_PIXEL_2X2  ,               // 2 X 2
                        DOT_PIXEL_3X3  ,               // 3 X 3
                        DOT_PIXEL_4X4  ,               // 4 X 4
                        DOT_PIXEL_5X5  ,               // 5 X 5
                        DOT_PIXEL_6X6  ,               // 6 X 6
                        DOT_PIXEL_7X7  ,               // 7 X 7
                        DOT_PIXEL_8X8  ,               // 8 X 8
                } DOT_PIXEL;
        Draw_Fill: 填充,是否填充矩形的内部
                typedef enum {
                        DRAW_FILL_EMPTY = 0,
                        DRAW_FILL_FULL,
                } DRAW_FILL;
```

- 画圆:在图像缓存中，以 (X_Center Y_Center) 为圆心，画一个半径为Radius的圆，可以选择颜色，线的宽度，是否填充圆内部

```
void Paint_DrawCircle(UWORD X_Center, UWORD Y_Center, UWORD Radius, UWORD Color, DO
T_PIXEL Line_width, DRAW_FILL Draw_Fill)
参数：
        X_Center: 圆心的X坐标
        Y_Center: 圆心的Y坐标
        Radius：圆的半径
        Color: 填充的颜色
        Line_width: 圆弧的宽度，提供默认的8种宽度
                typedef enum {
                        DOT_PIXEL_1X1  = 1,    // 1 x 1
                        DOT_PIXEL_2X2  ,               // 2 X 2
                        DOT_PIXEL_3X3  ,               // 3 X 3
                        DOT_PIXEL_4X4  ,               // 4 X 4
                        DOT_PIXEL_5X5  ,               // 5 X 5
                        DOT_PIXEL_6X6  ,               // 6 X 6
                        DOT_PIXEL_7X7  ,               // 7 X 7
                        DOT_PIXEL_8X8  ,               // 8 X 8
                } DOT_PIXEL;
        Draw_Fill: 填充,是否填充圆的内部
                typedef enum {
                        DRAW_FILL_EMPTY = 0,
                        DRAW_FILL_FULL,
                } DRAW_FILL;
```

- 写Ascii字符:在图像缓存中，在 (Xstart Ystart) 为左顶点，写一个Ascii字符，可以选择Ascii码可视字符字库、字体前景色、字体背景色

```
void Paint_DrawChar(UWORD Xstart, UWORD Ystart, const char Ascii_Char, sFONT* Font,
UWORD Color_Foreground, UWORD Color_Background)
参数：
        Xstart: 字符的左顶点X坐标
        Ystart: 字体的左顶点Y坐标
        Ascii_Char：Ascii字符
        Font: Ascii码可视字符字库，在Fonts文件夹中提供了以下字体：
                font8：5*8的字体
                font12：7*12的字体
                font16：11*16的字体
                font20：14*20的字体
                font24：17*24的字体
        Color_Foreground: 字体颜色
        Color_Background: 背景颜色
```

- 写英文字符串:在图像缓存中，在 (Xstart Ystart) 为左顶点，写一串英文字符，可以选择Ascii码可视字符字库、字体前景色、字体背景色

```
void Paint_DrawString_EN(UWORD Xstart, UWORD Ystart, const char * pString, sFONT* F
ont, UWORD Color_Foreground, UWORD Color_Background)
参数：
        Xstart：字符的左顶点X坐标
        Ystart：字体的左顶点Y坐标
        pString：字符串，字符串是一个指针
        Font：Ascii码可视字符字库，在Fonts文件夹中提供了以下字体：
                font8：5*8的字体
                font12：7*12的字体
                font16：11*16的字体
                font20：14*20的字体
                font24：17*24的字体
        Color_Foreground：字体颜色
        Color_Background：背景颜色
```

- 写中文字符串：在图像缓存中，在 (Xstart Ystart) 为左顶点，写一串中文字符，可以选择GB2312编码字符字库、字体前景色、字体背景色；

```
void Paint_DrawString_CN(UWORD Xstart, UWORD Ystart, const char * pString, cFONT* f
ont, UWORD Color_Foreground, UWORD Color_Background)
参数：
        Xstart：字符的左顶点X坐标
        Ystart：字体的左顶点Y坐标
        pString：字符串，字符串是一个指针
        Font：GB2312编码字符字库，在Fonts文件夹中提供了以下字体：
                font12CN：ascii字符字体11*21，中文字体16*21
                font24CN：ascii字符字体24*41，中文字体32*41
        Color_Foreground：字体颜色
        Color_Background：背景颜色
```

- 写数字:在图像缓存中，在 (Xstart Ystart) 为左顶点，写一串数字，可以选择Ascii码可视字符字库、字体前景色、字体背景色

```
void Paint_DrawNum(UWORD Xpoint, UWORD Ypoint, double Nummber, sFONT* Font, UWORD D
igit, UWORD Color_Foreground, UWORD Color_Background)
参数：
        Xpoint：字符的左顶点X坐标
        Ypoint：字体的左顶点Y坐标
        Nummber：显示的数字，可以是小数
        Digit：小数位数，不足补零
        Font：Ascii码可视字符字库，在Fonts文件夹中提供了以下字体：
                font8：5*8的字体
                font12：7*12的字体
                font16：11*16的字体
                font20：14*20的字体
                font24：17*24的字体
        Color_Foreground：字体颜色
        Color_Background：背景颜色
```

- 显示时间:在图像缓存中，在 (Xstart Ystart) 为左顶点，显示一段时间，可以选择Ascii码可视字符字库、字体前景色、字体背景色；

```
void Paint_DrawTime(UWORD Xstart, UWORD Ystart, PAINT_TIME *pTime, sFONT* Font, UWO
RD Color_Background, UWORD Color_Foreground)
参数：
        Xstart：字符的左顶点X坐标
        Ystart：字体的左顶点Y坐标
        pTime：显示的时间，这里定义好了一个时间的结构体，只要把时分秒各位数传给参数；
        Font：Ascii码可视字符字库，在Fonts文件夹中提供了以下字体：
                font8：5*8的字体
                font12：7*12的字体
                font16：11*16的字体
                font20：14*20的字体
                font24：17*24的字体
        Color_Foreground：字体颜色
        Color_Background：背景颜色
```

Python(适用于Raspberry Pi)

适用于python和python3

对于python而言他的调用没有C复杂

config.py

- 接口选择

```
Device_SPI = 1
Device_I2C = 0
注意：切换SPI/I2C修改这里
```

- 模块初始化与退出的处理：

```
def module_init()
def module_exit()
注意：
1.这里是处理使用液晶屏前与使用完之后一些GPIO的处理。
2.module_init()函数会在液晶屏的init()初始化程序自动调用，但module_exit()需要自行调用
```

- SPI写数据

```
def spi_writebyte(data)
```

- IIC写数据

```
i2c_writebyte(reg, value):
```

main.py

主函数,如果你的python版本是python2，在linux命令模式下重新执行如下：

```
sudo python main.py
```

如果你的python版本是python3，在linux命令模式下重新执行如下：

```
sudo python3 main.py
```

画图GUI

由于python有一个image库pil官方库链接，他十分的强大，不需要像C从逻辑层出发编写代码，可以直接引用image库进行图像处理，以下将以1.54inch LCD为例，对程序中使用了的进行简要说明

- 需要使用image库，需要安装库

```
sudo apt-get install python3-pil   安装库
```

然后导入库

```
from PIL import Image,ImageDraw,ImageFont
```

其中Image为基本库、ImageDraw为画图功能、ImageFont为文字

- 定义一个图像缓存，以方便在图片上进行画图、写字等功能

```
image1 = Image.new("1", (disp.width, disp.height), "WHITE")
```

第一个参数定义图片的颜色深度，定义为"1"说明是一位深度的位图，第二个参数是一个元组，定义好图片的宽度和高度，第三个参数是定义缓存的默认颜色，定义为“WHITE”。

- 创建一个基于image1的画图对象，所有的画图操作都在这个对象上

```
draw = ImageDraw.Draw(image1)
```

- 画线

```
    draw.line([(0,0),(127,0)], fill = 0)
```

第一个参数为一个4个元素的元组，以（0，0）为起始点，（127，0）为终止点，画一条直线，fill="0"表示线为白色。

- 画框

```
draw.rectangle([(20,10),(70,60)],fill = "WHITE",outline="BLACK")
```

第一个参数为一个4个元素的元组，（20，10）矩形左上角坐标值，（70，60）为矩形右下角坐标值，fill= "WHITE"表示内部填充黑色，outline="BLACK"表示外边框为黑色。

- 画圆

```
draw.arc((150,15,190,55),0, 360, fill =(0,255,0))
```

在正方形内画一个内切圆，第一个参数为一个4个元素的元组，以（150，15）为正方形的左上角顶点，（190，55）为正方形右下角顶点，规定矩形框的水平中位线为0度角，角度顺时针变大，第二个参数表示开始角度，第三个参数标识结束角度，fill = 0表示线为白色
如果不是正方形，画出来的就是椭圆，这个实际上是圆弧的绘制。

除了arc可以话圆之外，还有ellipse可以画实心圆

```
draw.ellipse((150,65,190,105), fill = 0)
```

实质是椭圆的绘制，第一个参数指定弦的圆外切矩形，fill = 0表示内部填充颜色为白色，如果椭圆的外切矩阵为正方形，椭圆就是圆了。

- 写字符

写字符往往需要写不同大小的字符，需要导入ImageFont模块，并实例化：

```
    Font1 = ImageFont.truetype("../Font/Font01.ttf",25)
    Font2 = ImageFont.truetype("../Font/Font01.ttf",35)
    Font3 = ImageFont.truetype("../Font/Font02.ttf",32)
```

为了有比较好的视觉体验，这里使用的是来自网络的免费字体，如果是其他的ttf结尾的字库文件也是支持的。
注：每字库包含的字符均有不同；如果某些字符不能显示，建议根据字库使用的编码集来使用
写英文字符直接使用即可，写中文，由于编码是GB2312所以需要在前面加个u：

```
draw.text((5, 68), 'Hello world', fill = 0, font=Font1)
text= u"微雪电子"
draw.text((5, 200), text, fill = 0, font=Font3)
```

第一个参数为一个2个元素的元组，以（5，68）为左顶点，字体为font1，点，fill为字体颜色， fill = 0，所以会显示字体颜色为白色，第二句显示微雪电子，字体颜色为白色。

- 读取本地图片

```
image = Image.open('../pic/pic.bmp')
```

参数为图片路径。

- 其他功能

python的image库十分强大，如果需要实现其他的更多功能，可以上官网学习 http://effbot.org/imagingbook pil，官方的是英文的，如果感觉对你不友好，当然我们国内也有很多的优秀的博客都有讲解。

STM32使用教程

提供基于STM32F103RBT6的例程

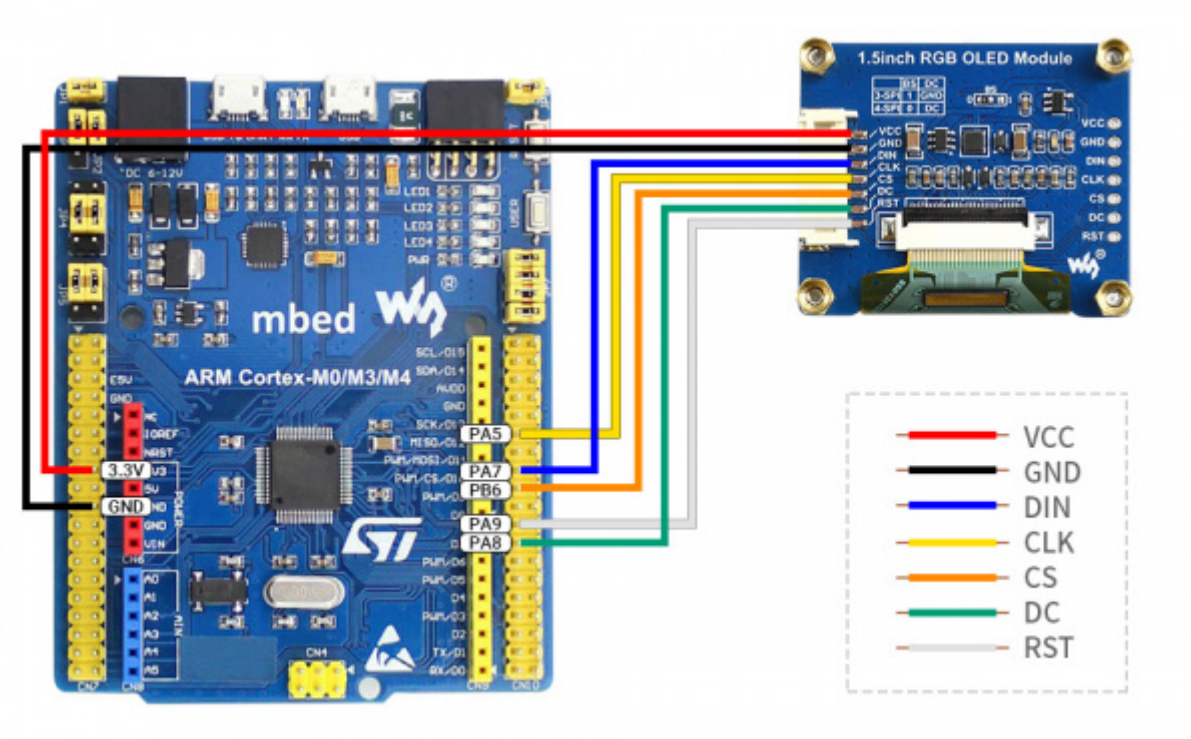
硬件连接

我们提供的例程是基于STM32F103RBT6的，提供的连接方式也是对应的STM32F103RBT6的引脚，如果需要移植程序，请按实际引脚连接

STM32F103RBT6连接引脚对应关系

OLED	STM32
VCC	3.3V
GND	GND
DIN	SPI:PA7 / I2C:PB9 / I2C_SOFT：PC8
CLK	SPI:PA5 / I2C:PB8 / I2C_SOFT：PC6
CS	PB6
DC	PA8
RST	PA9

- 四线SPI接线图



运行程序

- 下载程序，找到 STM32 程序文件目录，使用 Keil5 打开 \STM32\STM32-F103RBT6\MDK-ARM 目录下的 oled_demo.uvprojx
- 然后根据购买的屏幕型号修改 main.c 中对应的函数注释，最后重新编译下载即可。

```
97  //  OLED_0in91_test();  //  Only IIC !!!
98
99  //  OLED_0in95_rgb_test();  //  Only SPI !!!
100
101 //  OLED_0in96_test();    //  IIC must USE_IIC_SOFT
102
103 //  OLED_1in3_test();   //  IIC must USE_IIC_SOFT
104
105 //  OLED_1in3_c_test();
106
107 //  OLED_1in5_test();
108
109 //  OLED_1in5_rgb_test(); //  Only SPI !!!
```

- **假如**您购买了 1.3inch OLED Module （C）就将105行的注释取消掉（注：不能同时存在多条语句没注释；行号可能有改动，请根据实际情况修改）
- 型号指令对应表

屏幕型号	例程函数
0.91inch OLED Module	OLED_0in91_test();
0.95inch RGB OLED (A)/(B)	OLED_0in95_rgb_test();
0.96inch OLED (A)/(B)	OLED_0in96_test();
1.3inch OLED (A)/(B)	OLED_1in3_test();
1.3inch OLED Module (C)	OLED_1in3_c_test();
1.5inch OLED Module	OLED_1in5_test();
1.5inch RGB OLED Module	OLED_1in5_rgb_test();

软件说明

例程是基于HAL库进行开发的。 下载程序，找到STM32程序文件目录，打开STM32\STM32F103RBT6\MDK-ARM目录下的oled_demo.uvprojx，即可看到程序。

另外，在STM32\STM32-F103RBT6\User\目录下可以看到工程的文件目录，五个文件夹依次为底层驱动、示例程序、字库、GUI、OLED驱动

程序说明

底层硬件接口

我们进行了底层的封装，由于硬件平台不一样，内部的实现是不一样的，如果需要了解内部实现可以去对应的目录中查看 在DEV_Config.c(.h)可以看到很多定义

- 接口选择：

```
#define USE_SPI_4W      1
#define USE_IIC         0
#define USE_IIC_SOFT    0
注意：切换SPI/I2C直接修改这里
```

- 数据类型：

```
#define UBYTE   uint8_t
#define UWORD   uint16_t
#define UDOUBLE uint32_t
```

- 模块初始化与退出的处理：

```
UBYTE   System_Init(void);
void    System_Exit(void);
注意：
1.这里是处理使用液晶屏前与使用完之后一些GPIO的处理；
2.System_Exit(void)函数使用后，会关闭OLED显示屏；
```

- GPIO读写：

```
void    DEV_Digital_Write(UWORD Pin, UBYTE Value);
UBYTE   DEV_Digital_Read(UWORD Pin);
```

- SPI写数据:

```
UBYTE   SPI4W_Write_Byte(uint8_t value);
```

- IIC写数据:

```
void    I2C_Write_Byte(uint8_t value, uint8_t Cmd);
```

上层应用

对于屏幕而言，如果需要进行画图、显示中英文字符、显示图片等怎么办，这些都是上层应用做的。这有很多小伙伴有问到一些图形的处理，我们这里提供了一些基本的功能

在目录：STM32\STM32F103RB\User\GUI\GUI_Paint.c(.h)中可以找到GUI

在目录STM32\STM32F103RB\User\Fonts下是GUI依赖的字符字体

- 新建图像属性:新建一个图像属性，这个属性包括图像缓存的名称、宽度、高度、翻转角度、颜色

```
void Paint_NewImage(UWORD Width, UWORD Height, UWORD Rotate, UWORD Color);
参数：
        Width ： 图像缓存的宽度；
        Height: 图像缓存的高度；
        Rotate：图像的翻转的角度
        Color ： 图像的初始颜色；
```

- 设置清屏函数，通常直接调用OLED的clear函数；

```
void Paint_SetClearFuntion(void (*Clear)(UWORD));
参数：
        Clear ： 指向清屏函数的指针，用于快速将屏幕清空变成某颜色；
```

- 设置画像素点函数；

```
void Paint_SetDisplayFuntion(void (*Display)(UWORD,UWORD,UWORD));
参数：
        Display: 指向画像素点函数的指针，用于向OLED内部RAM指定位置写入数据；
```

- 选择图像缓存:选择图像缓存，选择的目的是你可以创建多个图像属性，图像缓存可以存在多个，你可以选择你所创建的每一张图像

```
void Paint_SelectImage(UBYTE *image)
参数：
        image: 图像缓存的名称，实际上是一个指向图像缓存首地址的指针；
```

- 图像旋转:设置选择好的图像的旋转角度，最好使用在Paint_SelectImage()后，可以选择旋转0、90、180、270

```
void Paint_SetRotate(UWORD Rotate)
参数：
        Rotate: 图像选择角度，可以选择ROTATE_0、ROTATE_90、ROTATE_180、ROTATE_270分别对应0、90、180、270度
```

- 图像镜像翻转:设置选择好的图像的镜像翻转，可以选择不镜像、关于水平镜像、关于垂直镜像、关于图像中心镜像。

```
void Paint_SetMirroring(UBYTE mirror)
参数：
        mirror: 图像的镜像方式，可以选择MIRROR_NONE、MIRROR_HORIZONTAL、MIRROR_VERTICAL、MIRROR_ORIGIN分别对应不镜像、关于水平镜像、关于垂直镜像、关于图像中心镜像
```

- 设置点在缓存中显示位置和颜色：这里是GUI最核心的一个函数、处理点在缓存中显示位置和颜色；

```
void Paint_SetPixel(UWORD Xpoint, UWORD Ypoint, UWORD Color)
参数：
        Xpoint: 点在图像缓存中X位置
        Ypoint: 点在图像缓存中Y位置
        Color : 点显示的颜色
```

- 图像缓存填充颜色:把图像缓存填充为某颜色，一般作为屏幕刷白的作用

```
void Paint_Clear(UWORD Color)
参数：
        Color: 填充的颜色
```

- 图像缓存部分窗口填充颜色：把图像缓存的某部分窗口填充为某颜色，一般作为窗口刷白的作用，常用于时间的显示，刷白上一秒

```
void Paint_ClearWindows(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD C
olor)
参数：
        Xstart: 窗口的X起点坐标
        Ystart: 窗口的Y起点坐标
        Xend: 窗口的X终点坐标
        Yend: 窗口的Y终点坐标
        Color: 填充的颜色
```

- 画点:在图像缓存中，在（Xpoint, Ypoint）上画点，可以选择颜色，点的大小，点的风格

```
void Paint_DrawPoint(UWORD Xpoint, UWORD Ypoint, UWORD Color, DOT_PIXEL Dot_Pixel,
DOT_STYLE Dot_Style)
参数：
        Xpoint: 点的X坐标
        Ypoint: 点的Y坐标
        Color: 填充的颜色
        Dot_Pixel: 点的大小，提供默认的8种大小点
                 typedef enum {
                        DOT_PIXEL_1X1  = 1,    // 1 x 1
                        DOT_PIXEL_2X2  ,               // 2 X 2
                        DOT_PIXEL_3X3  ,               // 3 X 3
                        DOT_PIXEL_4X4  ,               // 4 X 4
                        DOT_PIXEL_5X5  ,               // 5 X 5
                        DOT_PIXEL_6X6  ,               // 6 X 6
                        DOT_PIXEL_7X7  ,               // 7 X 7
                        DOT_PIXEL_8X8  ,               // 8 X 8
                } DOT_PIXEL;
        Dot_Style: 点的风格,大小扩充方式是以点为中心扩大还是以点为左下角往右上扩大
                typedef enum {
                   DOT_FILL_AROUND  = 1,
                   DOT_FILL_RIGHTUP,
                } DOT_STYLE;
```

- 画线：在图像缓存中，从 (Xstart, Ystart) 到 (Xend, Yend) 画线，可以选择颜色，线的宽度，线的风格

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Colo
r, LINE_STYLE Line_Style , LINE_STYLE Line_Style)
```
参数：

Xstart: 线的X起点坐标

Ystart: 线的Y起点坐标

Xend: 线的X终点坐标

Yend: 线的Y终点坐标

Color: 填充的颜色

Line_width: 线的宽度，提供默认的8种宽度

```
typedef enum {
    DOT_PIXEL_1X1  = 1,    // 1 x 1
    DOT_PIXEL_2X2  ,               // 2 X 2
    DOT_PIXEL_3X3  ,               // 3 X 3
    DOT_PIXEL_4X4  ,               // 4 X 4
    DOT_PIXEL_5X5  ,               // 5 X 5
    DOT_PIXEL_6X6  ,               // 6 X 6
    DOT_PIXEL_7X7  ,               // 7 X 7
    DOT_PIXEL_8X8  ,               // 8 X 8
} DOT_PIXEL;
```

Line_Style: 线的风格,选择线是以直线连接还是以虚线的方式连接

```
typedef enum {
    LINE_STYLE_SOLID = 0,
    LINE_STYLE_DOTTED,
} LINE_STYLE;
```

- 画矩形:在图像缓存中，从 (Xstart, Ystart) 到 (Xend, Yend) 画一个矩形，可以选择颜色，线的宽度，是否填充矩形内部

```
void Paint_DrawRectangle(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD
Color, DOT_PIXEL Line_width, DRAW_FILL Draw_Fill)
参数:
        Xstart: 矩形的X起点坐标
        Ystart: 矩形的Y起点坐标
        Xend: 矩形的X终点坐标
        Yend: 矩形的Y终点坐标
        Color: 填充的颜色
        Line_width: 矩形四边的宽度，提供默认的8种宽度
                typedef enum {
                        DOT_PIXEL_1X1  = 1,    // 1 x 1
                        DOT_PIXEL_2X2  ,               // 2 X 2
                        DOT_PIXEL_3X3  ,               // 3 X 3
                        DOT_PIXEL_4X4  ,               // 4 X 4
                        DOT_PIXEL_5X5  ,               // 5 X 5
                        DOT_PIXEL_6X6  ,               // 6 X 6
                        DOT_PIXEL_7X7  ,               // 7 X 7
                        DOT_PIXEL_8X8  ,               // 8 X 8
                } DOT_PIXEL;
        Draw_Fill: 填充,是否填充矩形的内部
                typedef enum {
                        DRAW_FILL_EMPTY = 0,
                        DRAW_FILL_FULL,
                } DRAW_FILL;
```

- 画圆:在图像缓存中，以 (X_Center Y_Center) 为圆心，画一个半径为Radius的圆，可以选择颜色，线的宽度，是否填充圆内部

```
void Paint_DrawCircle(UWORD X_Center, UWORD Y_Center, UWORD Radius, UWORD Color, DO
T_PIXEL Line_width, DRAW_FILL Draw_Fill)
参数：
        X_Center: 圆心的X坐标
        Y_Center: 圆心的Y坐标
        Radius：圆的半径
        Color: 填充的颜色
        Line_width: 圆弧的宽度，提供默认的8种宽度
                typedef enum {
                        DOT_PIXEL_1X1  = 1,    // 1 x 1
                        DOT_PIXEL_2X2  ,               // 2 X 2
                        DOT_PIXEL_3X3  ,               // 3 X 3
                        DOT_PIXEL_4X4  ,               // 4 X 4
                        DOT_PIXEL_5X5  ,               // 5 X 5
                        DOT_PIXEL_6X6  ,               // 6 X 6
                        DOT_PIXEL_7X7  ,               // 7 X 7
                        DOT_PIXEL_8X8  ,               // 8 X 8
                } DOT_PIXEL;
        Draw_Fill: 填充,是否填充圆的内部
                typedef enum {
                        DRAW_FILL_EMPTY = 0,
                        DRAW_FILL_FULL,
                } DRAW_FILL;
```

- 写Ascii字符:在图像缓存中，在 (Xstart Ystart) 为左顶点，写一个Ascii字符，可以选择Ascii码可视字符字库、字体前景色、字体背景色

```
void Paint_DrawChar(UWORD Xstart, UWORD Ystart, const char Ascii_Char, sFONT* Font,
UWORD Color_Foreground, UWORD Color_Background)
参数：
        Xstart: 字符的左顶点X坐标
        Ystart: 字体的左顶点Y坐标
        Ascii_Char：Ascii字符
        Font: Ascii码可视字符字库，在Fonts文件夹中提供了以下字体：
                font8：5*8的字体
                font12：7*12的字体
                font16：11*16的字体
                font20：14*20的字体
                font24：17*24的字体
        Color_Foreground: 字体颜色
        Color_Background: 背景颜色
```

- 写英文字符串:在图像缓存中，在 (Xstart Ystart) 为左顶点，写一串英文字符，可以选择Ascii码可视字符字库、字体前景色、字体背景色

```
void Paint_DrawString_EN(UWORD Xstart, UWORD Ystart, const char * pString, sFONT* F
ont, UWORD Color_Foreground, UWORD Color_Background)
参数：
        Xstart：字符的左顶点X坐标
        Ystart：字体的左顶点Y坐标
        pString：字符串，字符串是一个指针
        Font：Ascii码可视字符字库，在Fonts文件夹中提供了以下字体：
                font8：5*8的字体
                font12：7*12的字体
                font16：11*16的字体
                font20：14*20的字体
                font24：17*24的字体
        Color_Foreground：字体颜色
        Color_Background：背景颜色
```

- 写中文字符串：在图像缓存中，在 (Xstart Ystart) 为左顶点，写一串中文字符，可以选择GB2312编码字符字库、字体前景色、字体背景色；

```
void Paint_DrawString_CN(UWORD Xstart, UWORD Ystart, const char * pString, cFONT* f
ont, UWORD Color_Foreground, UWORD Color_Background)
参数：
        Xstart：字符的左顶点X坐标
        Ystart：字体的左顶点Y坐标
        pString：字符串，字符串是一个指针
        Font：GB2312编码字符字库，在Fonts文件夹中提供了以下字体：
                font12CN：ascii字符字体11*21，中文字体16*21
                font24CN：ascii字符字体24*41，中文字体32*41
        Color_Foreground：字体颜色
        Color_Background：背景颜色
```

- 写数字:在图像缓存中，在 (Xstart Ystart) 为左顶点，写一串数字，可以选择Ascii码可视字符字库、字体前景色、字体背景色

```
void Paint_DrawNum(UWORD Xpoint, UWORD Ypoint, double Nummber, sFONT* Font, UWORD D
igit, UWORD Color_Foreground, UWORD Color_Background)
参数：
        Xpoint：字符的左顶点X坐标
        Ypoint：字体的左顶点Y坐标
        Nummber：显示的数字，可以是小数
        Digit：小数位数，不足补零
        Font：Ascii码可视字符字库，在Fonts文件夹中提供了以下字体：
                font8：5*8的字体
                font12：7*12的字体
                font16：11*16的字体
                font20：14*20的字体
                font24：17*24的字体
        Color_Foreground：字体颜色
        Color_Background：背景颜色
```

- 显示时间:在图像缓存中，在 (Xstart Ystart) 为左顶点，显示一段时间，可以选择Ascii码可视字符字库、字体前景色、字体背景色；

```
void Paint_DrawTime(UWORD Xstart, UWORD Ystart, PAINT_TIME *pTime, sFONT* Font, UWO
RD Color_Background, UWORD Color_Foreground)
参数：
        Xstart：字符的左顶点X坐标
        Ystart：字体的左顶点Y坐标
        pTime：显示的时间，这里定义好了一个时间的结构体，只要把时分秒各位数传给参数；
        Font：Ascii码可视字符字库，在Fonts文件夹中提供了以下字体：
                font8：5*8的字体
                font12：7*12的字体
                font16：11*16的字体
                font20：14*20的字体
                font24：17*24的字体
        Color_Foreground：字体颜色
        Color_Background：背景颜色
```

Arduino使用教程

提供基于UNO PLUS的例程

硬件连接

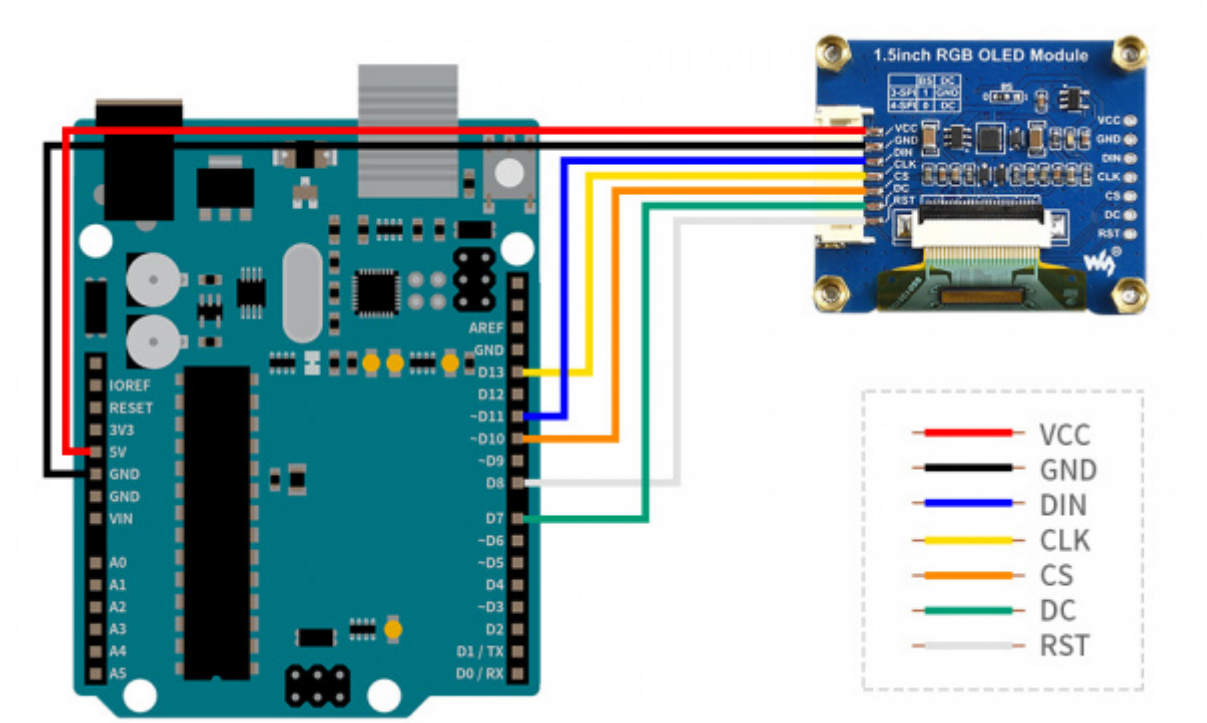
我们提供的例程是基于UNO PLUS的，提供的连接方式也是对应的UNO PLUS的引脚，如果需要移植程序，请按实际引脚连接

Arduino UNO连接引脚对应关系

OLED	UNO

VCC	5V
GND	GND
DIN	SPI:D11 / I2C:SDA
CLK	SPI:D13 / I2C:SCL
CS	D10
DC	D7
RST	D8

四线SPI接线图:



arduino IDE 安装教程

arduino IDE 安装教程

运行程序

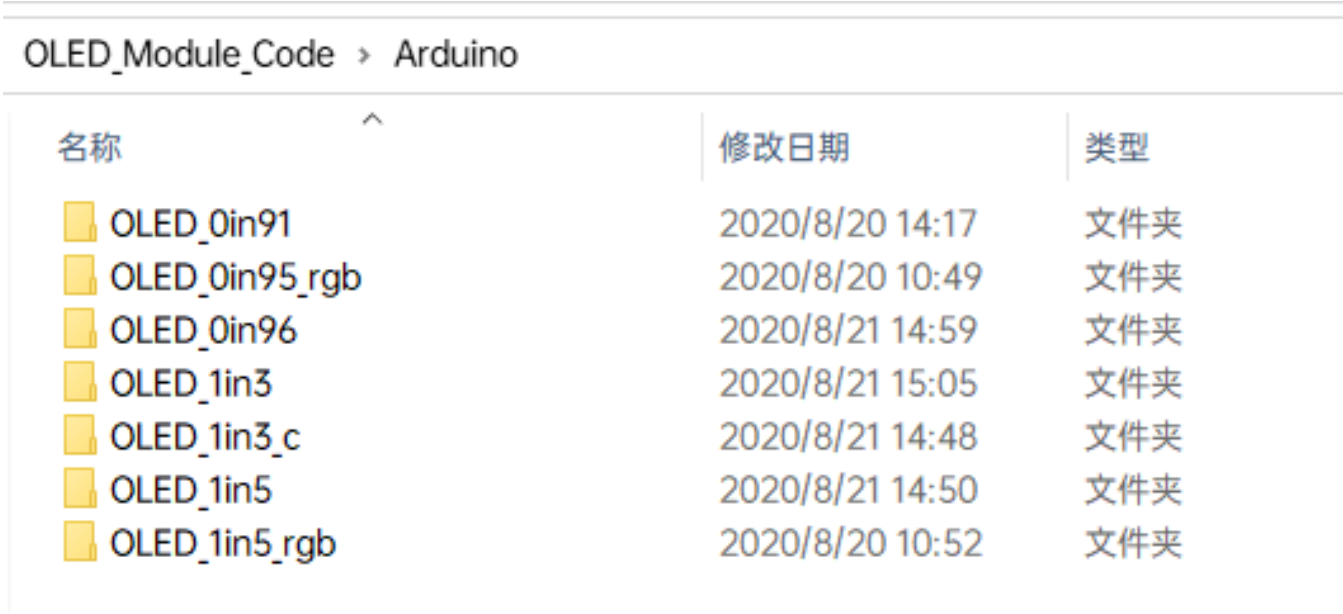
下载程序，找到Arduino程序文件目录。

使用Arduino IDE打开对应型号的工程文件夹下的 .ino 文件，重新编译下载即可。

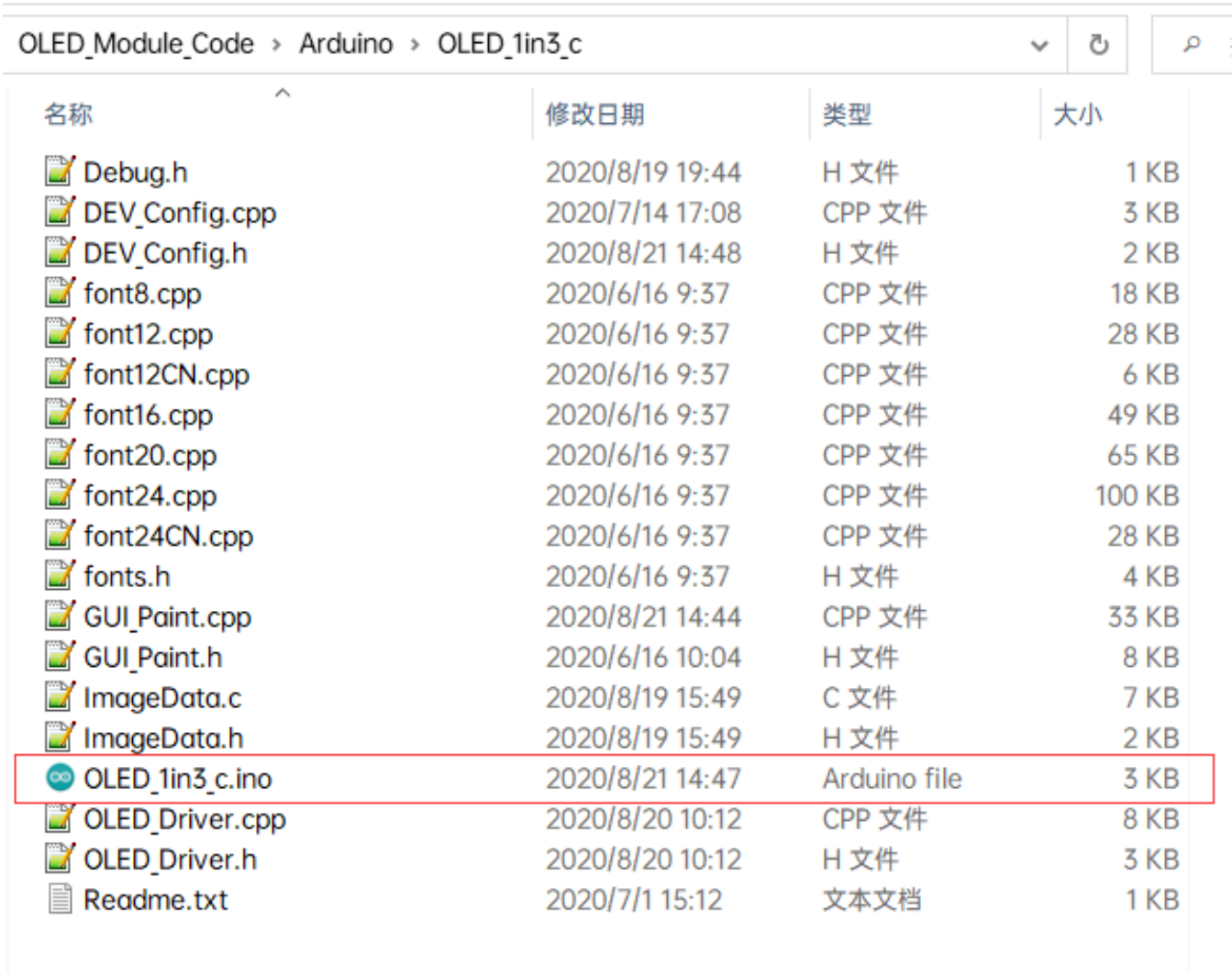
例如您购买的是 1.3inch OLED Module (C) 就打开 \Arduino\OLED_1in3_c 目录下的 OLED_1in3_c.ino

软件说明

- 下载程序，打开Arduino程序文件目录，即可看到不同型号OLED的Arduino程序，具体对应关系可以看下方 ‘型号程序对应表’



- 根据你购买的尺寸和类型选择打开的文件夹，并打开 xxx.ino 文件，我们以1.3inch OLED Module （C）**为例**：打开OLED_1in3_c，然后双击 OLED_1in3_c.ino 打开Arduino的工程。



- 型号程序对应表

屏幕型号	程序文件夹
0.91inch OLED Module	OLED_0in91
0.95inch RGB OLED (A)/(B)	OLED_0in95_rgb
0.96inch OLED (A)/(B)	OLED_0in96
1.3inch OLED (A)/(B)	OLED_1in3
1.3inch OLED Module (C)	OLED_1in3_c
1.5inch OLED Module	OLED_1in5
1.5inch RGB OLED Module	OLED_1in5_rgb

程序说明

底层硬件接口

我们进行了底层的封装，由于硬件平台不一样，内部的实现是不一样的，如果需要了解内部实现可以去对应的目录中查看 在DEV_Config.c(.h)可以看到很多定义

- 接口选择：

```
#define USE_SPI_4W  1
#define USE_IIC     0
注意：切换SPI/I2C直接修改这里
```

- 数据类型：

```
#define UBYTE   uint8_t
#define UWORD   uint16_t
#define UDOUBLE uint32_t
```

- 模块初始化与退出的处理：

```
UBYTE   System_Init(void);
void    System_Exit(void);
注意：
1.这里是处理使用液晶屏前与使用完之后一些GPIO的处理；
2.System_Exit函数使用后，会关闭OLED显示屏；
```

- GPIO读写：

```
void    DEV_Digital_Write(UWORD Pin, UBYTE Value);
UBYTE   DEV_Digital_Read(UWORD Pin);
```

- SPI写数据

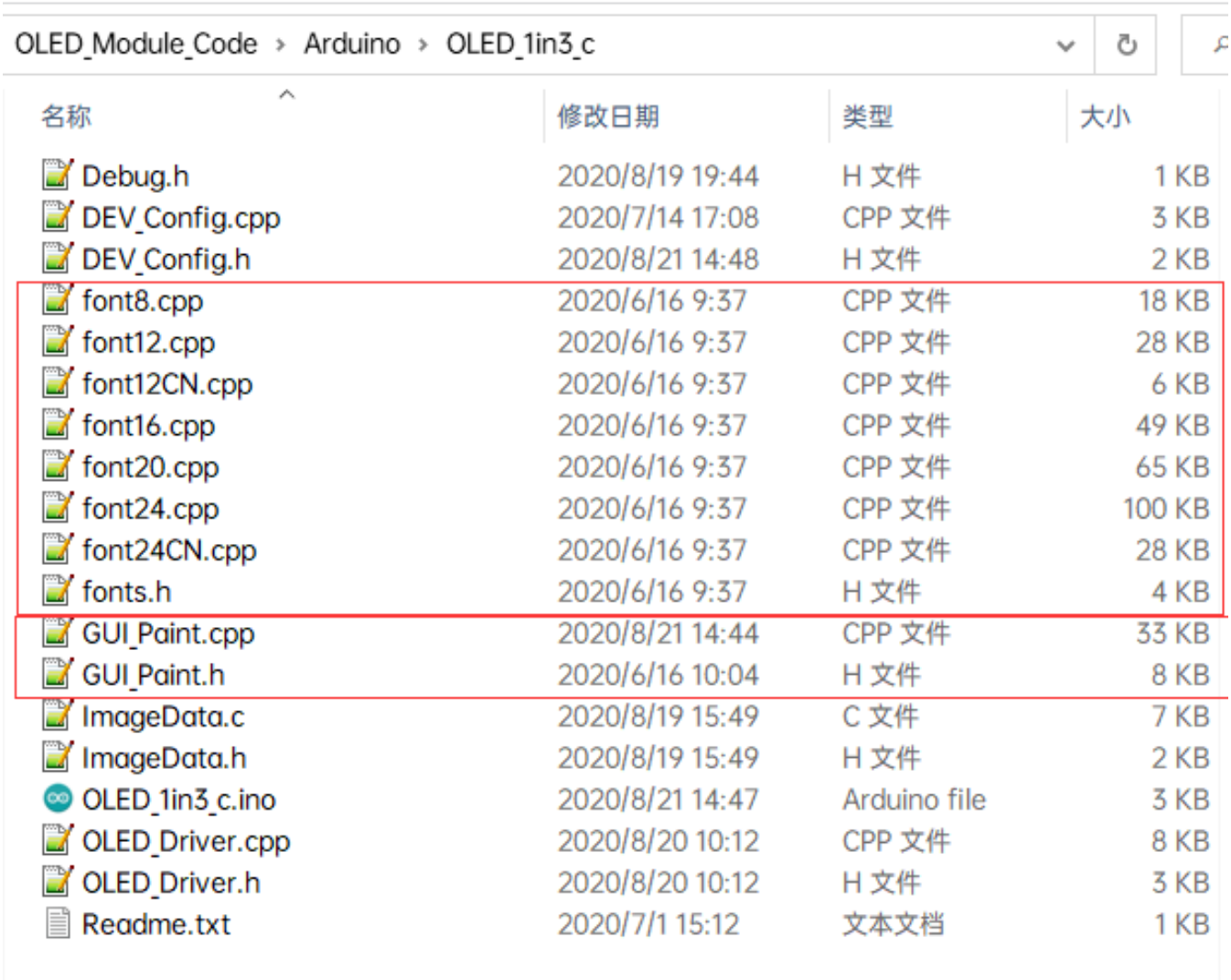
```
UBYTE SPI4W_Write_Byte(uint8_t value);
```

- IIC写数据：

```
void I2C_Write_Byte(uint8_t value, uint8_t Cmd);
```

上层应用

对于屏幕而言，如果需要进行画图、显示中英文字符、显示图片等怎么办，这些都是上层应用做的。这有很多小伙伴有问到一些图形的处理，我们这里提供了一些基本的功能 在如下的目录中可以找到GUI和其自带的字库，在目录：Arduino\OLED_xxx\GUI_Paint.c(.h)



- 新建图像属性:新建一个图像属性，这个属性包括图像缓存的名称、宽度、高度、翻转角度、颜色

```
void Paint_NewImage(UWORD Width, UWORD Height, UWORD Rotate, UWORD Color);
参数：
        Width ： 图像缓存的宽度；
        Height: 图像缓存的高度；
        Rotate：图像的翻转的角度
        Color ： 图像的初始颜色；
```

- 设置清屏函数，通常直接调用OLED的clear函数；

```
void Paint_SetClearFuntion(void (*Clear)(UWORD));
参数：
        Clear ： 指向清屏函数的指针，用于快速将屏幕清空变成某颜色；
```

- 设置画像素点函数；

```
void Paint_SetDisplayFuntion(void (*Display)(UWORD,UWORD,UWORD));
参数：
        Display: 指向画像素点函数的指针，用于向OLED内部RAM指定位置写入数据；
```

- 选择图像缓存:选择图像缓存，选择的目的是你可以创建多个图像属性，图像缓存可以存在多个，你可以选择你所创建的每一张图像

```
void Paint_SelectImage(UBYTE *image)
参数：
        image: 图像缓存的名称，实际上是一个指向图像缓存首地址的指针；
```

- 图像旋转:设置选择好的图像的旋转角度，最好使用在Paint_SelectImage()后，可以选择旋转0、90、180、270

```
void Paint_SetRotate(UWORD Rotate)
参数：
        Rotate: 图像选择角度，可以选择ROTATE_0、ROTATE_90、ROTATE_180、ROTATE_270分别对应0、90、180、270度
```

- 图像镜像翻转:设置选择好的图像的镜像翻转，可以选择不镜像、关于水平镜像、关于垂直镜像、关于图像中心镜像。

```
void Paint_SetMirroring(UBYTE mirror)
参数：
        mirror: 图像的镜像方式，可以选择MIRROR_NONE、MIRROR_HORIZONTAL、MIRROR_VERTICAL、MIRROR_ORIGIN分别对应不镜像、关于水平镜像、关于垂直镜像、关于图像中心镜像
```

- 设置点在缓存中显示位置和颜色：这里是GUI最核心的一个函数、处理点在缓存中显示位置和颜色；

```
void Paint_SetPixel(UWORD Xpoint, UWORD Ypoint, UWORD Color)
参数：
        Xpoint: 点在图像缓存中X位置
        Ypoint: 点在图像缓存中Y位置
        Color : 点显示的颜色
```

- 图像缓存填充颜色:把图像缓存填充为某颜色，一般作为屏幕刷白的作用

```
void Paint_Clear(UWORD Color)
参数：
        Color: 填充的颜色
```

- 图像缓存部分窗口填充颜色：把图像缓存的某部分窗口填充为某颜色，一般作为窗口刷白的作用，常用于时间的显示，刷白上一秒

```
void Paint_ClearWindows(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD C
olor)
参数：
        Xstart: 窗口的X起点坐标
        Ystart: 窗口的Y起点坐标
        Xend: 窗口的X终点坐标
        Yend: 窗口的Y终点坐标
        Color: 填充的颜色
```

- 画点:在图像缓存中，在（Xpoint, Ypoint）上画点，可以选择颜色，点的大小，点的风格

```
void Paint_DrawPoint(UWORD Xpoint, UWORD Ypoint, UWORD Color, DOT_PIXEL Dot_Pixel,
DOT_STYLE Dot_Style)
参数：
        Xpoint: 点的X坐标
        Ypoint: 点的Y坐标
        Color: 填充的颜色
        Dot_Pixel: 点的大小，提供默认的8种大小点
                 typedef enum {
                        DOT_PIXEL_1X1  = 1,    // 1 x 1
                        DOT_PIXEL_2X2  ,               // 2 X 2
                        DOT_PIXEL_3X3  ,               // 3 X 3
                        DOT_PIXEL_4X4  ,               // 4 X 4
                        DOT_PIXEL_5X5  ,               // 5 X 5
                        DOT_PIXEL_6X6  ,               // 6 X 6
                        DOT_PIXEL_7X7  ,               // 7 X 7
                        DOT_PIXEL_8X8  ,               // 8 X 8
                } DOT_PIXEL;
        Dot_Style: 点的风格,大小扩充方式是以点为中心扩大还是以点为左下角往右上扩大
                typedef enum {
                   DOT_FILL_AROUND  = 1,
                   DOT_FILL_RIGHTUP,
                } DOT_STYLE;
```

- 画线：在图像缓存中，从 (Xstart, Ystart) 到 (Xend, Yend) 画线，可以选择颜色，线的宽度，线的风格

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Colo
r, LINE_STYLE Line_Style , LINE_STYLE Line_Style)
参数：
        Xstart: 线的X起点坐标
        Ystart: 线的Y起点坐标
        Xend: 线的X终点坐标
        Yend: 线的Y终点坐标
        Color: 填充的颜色
        Line_width: 线的宽度，提供默认的8种宽度
                typedef enum {
                        DOT_PIXEL_1X1  = 1,    // 1 x 1
                        DOT_PIXEL_2X2  ,               // 2 X 2
                        DOT_PIXEL_3X3  ,               // 3 X 3
                        DOT_PIXEL_4X4  ,               // 4 X 4
                        DOT_PIXEL_5X5  ,               // 5 X 5
                        DOT_PIXEL_6X6  ,               // 6 X 6
                        DOT_PIXEL_7X7  ,               // 7 X 7
                        DOT_PIXEL_8X8  ,               // 8 X 8
                } DOT_PIXEL;
         Line_Style: 线的风格,选择线是以直线连接还是以虚线的方式连接
                typedef enum {
                        LINE_STYLE_SOLID = 0,
                        LINE_STYLE_DOTTED,
                } LINE_STYLE;
```

- 画矩形:在图像缓存中，从 (Xstart, Ystart) 到 (Xend, Yend) 画一个矩形，可以选择颜色，线的宽度，是否填充矩形内部

```
void Paint_DrawRectangle(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD
Color, DOT_PIXEL Line_width, DRAW_FILL Draw_Fill)
参数:
        Xstart: 矩形的X起点坐标
        Ystart: 矩形的Y起点坐标
        Xend: 矩形的X终点坐标
        Yend: 矩形的Y终点坐标
        Color: 填充的颜色
        Line_width: 矩形四边的宽度，提供默认的8种宽度
                typedef enum {
                        DOT_PIXEL_1X1  = 1,    // 1 x 1
                        DOT_PIXEL_2X2  ,               // 2 X 2
                        DOT_PIXEL_3X3  ,               // 3 X 3
                        DOT_PIXEL_4X4  ,               // 4 X 4
                        DOT_PIXEL_5X5  ,               // 5 X 5
                        DOT_PIXEL_6X6  ,               // 6 X 6
                        DOT_PIXEL_7X7  ,               // 7 X 7
                        DOT_PIXEL_8X8  ,               // 8 X 8
                } DOT_PIXEL;
        Draw_Fill: 填充,是否填充矩形的内部
                typedef enum {
                        DRAW_FILL_EMPTY = 0,
                        DRAW_FILL_FULL,
                } DRAW_FILL;
```

- 画圆:在图像缓存中，以 (X_Center Y_Center) 为圆心，画一个半径为Radius的圆，可以选择颜色，线的宽度，是否填充圆内部

```
void Paint_DrawCircle(UWORD X_Center, UWORD Y_Center, UWORD Radius, UWORD Color, DO
T_PIXEL Line_width, DRAW_FILL Draw_Fill)
参数：
        X_Center: 圆心的X坐标
        Y_Center: 圆心的Y坐标
        Radius：圆的半径
        Color: 填充的颜色
        Line_width: 圆弧的宽度，提供默认的8种宽度
                typedef enum {
                        DOT_PIXEL_1X1  = 1,    // 1 x 1
                        DOT_PIXEL_2X2  ,               // 2 X 2
                        DOT_PIXEL_3X3  ,               // 3 X 3
                        DOT_PIXEL_4X4  ,               // 4 X 4
                        DOT_PIXEL_5X5  ,               // 5 X 5
                        DOT_PIXEL_6X6  ,               // 6 X 6
                        DOT_PIXEL_7X7  ,               // 7 X 7
                        DOT_PIXEL_8X8  ,               // 8 X 8
                } DOT_PIXEL;
        Draw_Fill: 填充,是否填充圆的内部
                typedef enum {
                        DRAW_FILL_EMPTY = 0,
                        DRAW_FILL_FULL,
                } DRAW_FILL;
```

- 写Ascii字符:在图像缓存中，在 (Xstart Ystart) 为左顶点，写一个Ascii字符，可以选择Ascii码可视字符字库、字体前景色、字体背景色

```
void Paint_DrawChar(UWORD Xstart, UWORD Ystart, const char Ascii_Char, sFONT* Font,
UWORD Color_Foreground, UWORD Color_Background)
参数：
        Xstart: 字符的左顶点X坐标
        Ystart: 字体的左顶点Y坐标
        Ascii_Char：Ascii字符
        Font: Ascii码可视字符字库，在Fonts文件夹中提供了以下字体：
                font8：5*8的字体
                font12：7*12的字体
                font16：11*16的字体
                font20：14*20的字体
                font24：17*24的字体
        Color_Foreground: 字体颜色
        Color_Background: 背景颜色
```

- 写英文字符串:在图像缓存中，在 (Xstart Ystart) 为左顶点，写一串英文字符，可以选择Ascii码可视字符字库、字体前景色、字体背景色

```
void Paint_DrawString_EN(UWORD Xstart, UWORD Ystart, const char * pString, sFONT* F
ont, UWORD Color_Foreground, UWORD Color_Background)
参数：
        Xstart：字符的左顶点X坐标
        Ystart：字体的左顶点Y坐标
        pString：字符串，字符串是一个指针
        Font：Ascii码可视字符字库，在Fonts文件夹中提供了以下字体：
                font8：5*8的字体
                font12：7*12的字体
                font16：11*16的字体
                font20：14*20的字体
                font24：17*24的字体
        Color_Foreground：字体颜色
        Color_Background：背景颜色
```

- 写中文字符串：在图像缓存中，在 (Xstart Ystart) 为左顶点，写一串中文字符，可以选择GB2312编码字符字库、字体前景色、字体背景色；

```
void Paint_DrawString_CN(UWORD Xstart, UWORD Ystart, const char * pString, cFONT* f
ont, UWORD Color_Foreground, UWORD Color_Background)
参数：
        Xstart：字符的左顶点X坐标
        Ystart：字体的左顶点Y坐标
        pString：字符串，字符串是一个指针
        Font：GB2312编码字符字库，在Fonts文件夹中提供了以下字体：
                font12CN：ascii字符字体11*21，中文字体16*21
                font24CN：ascii字符字体24*41，中文字体32*41
        Color_Foreground：字体颜色
        Color_Background：背景颜色
```

- 写数字:在图像缓存中，在 (Xstart Ystart) 为左顶点，写一串数字，可以选择Ascii码可视字符字库、字体前景色、字体背景色

```
void Paint_DrawNum(UWORD Xpoint, UWORD Ypoint, double Nummber, sFONT* Font, UWORD D
igit, UWORD Color_Foreground, UWORD Color_Background)
参数：
        Xpoint：字符的左顶点X坐标
        Ypoint：字体的左顶点Y坐标
        Nummber：显示的数字，可以是小数
        Digit：小数位数，不足补零
        Font：Ascii码可视字符字库，在Fonts文件夹中提供了以下字体：
                font8：5*8的字体
                font12：7*12的字体
                font16：11*16的字体
                font20：14*20的字体
                font24：17*24的字体
        Color_Foreground：字体颜色
        Color_Background：背景颜色
```

- 显示时间:在图像缓存中，在 (Xstart Ystart) 为左顶点，显示一段时间，可以选择Ascii码可视字符字库、字体前景色、字体背景色；

```
void Paint_DrawTime(UWORD Xstart, UWORD Ystart, PAINT_TIME *pTime, sFONT* Font, UWO
RD Color_Background, UWORD Color_Foreground)
参数：
        Xstart：字符的左顶点X坐标
        Ystart：字体的左顶点Y坐标
        pTime：显示的时间，这里定义好了一个时间的结构体，只要把时分秒各位数传给参数；
        Font：Ascii码可视字符字库，在Fonts文件夹中提供了以下字体：
                font8：5*8的字体
                font12：7*12的字体
                font16：11*16的字体
                font20：14*20的字体
                font24：17*24的字体
        Color_Foreground：字体颜色
        Color_Background：背景颜色
```

资料

提供文档、程序、数据手册等全套资料

文档

- 原理图

程序

- 新版示例程序

3D图纸

- 1.5inch RGB OLED Module 3D Drawing

软件

- 汉字取模软件
- Image2Lcd 图片取模软件

数据手册

- SDD1351数据手册
- 1.5inch RGB OLED数据手册

应用笔记

- 显示图片

其他

- 旧版示例程序及其教程

批量下载教程——请戳☛ 

FAQ

问题：OLED模块的工作电流是多少?

在3.3V工作电压下:全白约为60mA，全黑约为4mA。

问题：OLED模块接上电源为什么不亮?

OLED是没有背光的，显示属于自发光方式。只接VCC和GND，OLED是不会亮的。 必须用程序控制才能亮点OLED。