1.28inch LCD Module

来自Waveshare Wiki
跳转至： 导航、 搜索

说明

产品简介

本产品提供树莓派、STM32、arduino例程

产品参数

- 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）**
- 通信接口： SPI
- 屏幕类型： IPS
- 控制芯片： GC9A01
- 分辨率： 240(H)RGB x 240(V)
- 显示尺寸： Φ32.4mm
- 像素大小： 0.135（H）x0.135（V）mm
- 产品尺寸: 40.4×37.5(mm) Φ37.5(mm)

(https://www.waveshare.net/shop/1.28inch-LCD-Module.htm)

功能简介

显示尺寸	1.28英寸
分辨率	240×240
接口	SPI (/wiki/%E5%88%86%E7%B1%BB:SPI%E6%8E%A5%E5%8F%A3)

功能引脚

树莓派硬件连接

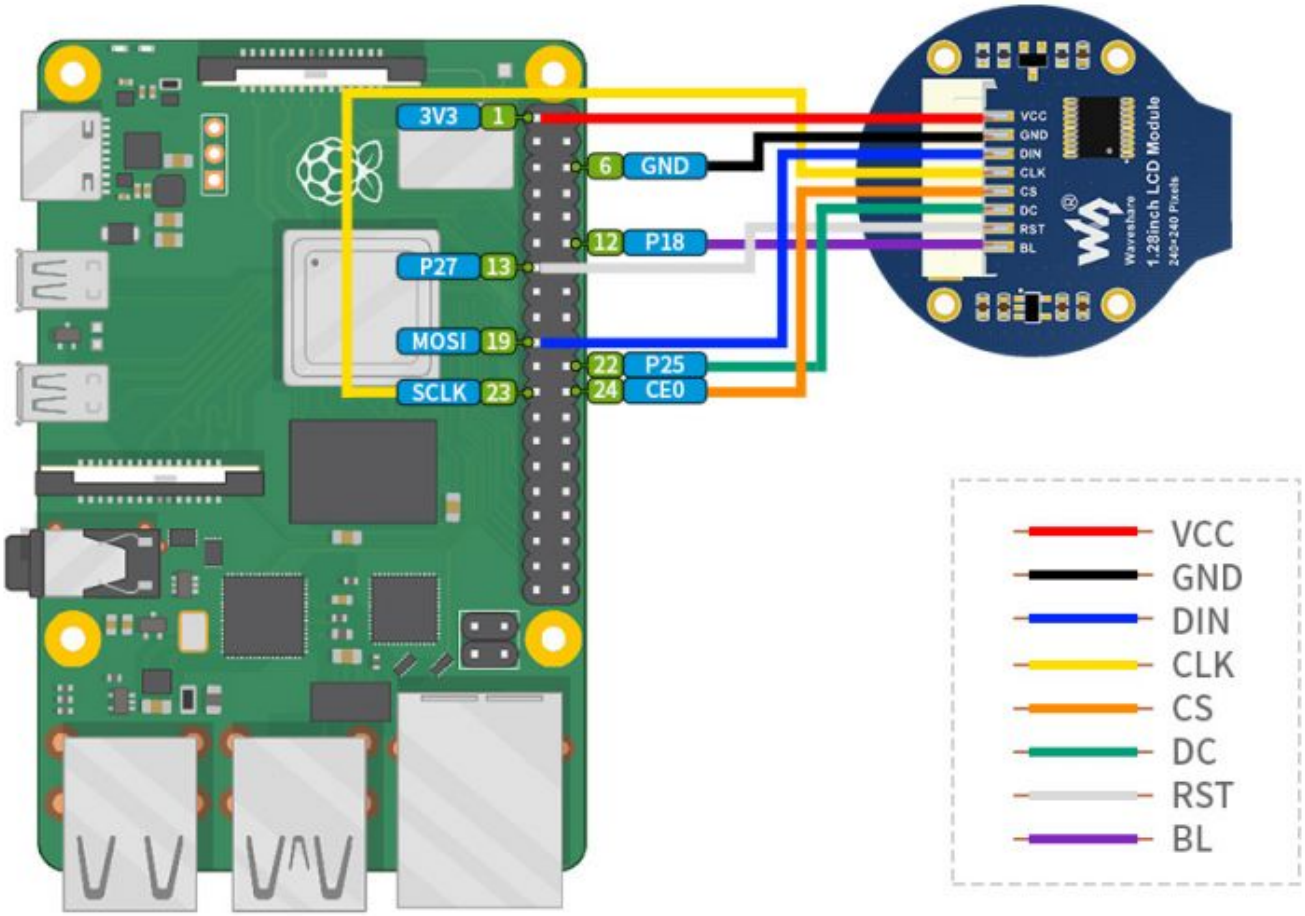
连接树莓派的时候，选择用8PIN排线连接，请参考下方的引脚对应表格
使用的是排针或者PH2.0 8PIN接口，需要对照以下表格连线

树莓派连接引脚对应关系

LCD	Raspberry Pi	
	BCM2835编码	Board物理引脚序号
VCC	3.3V	3.3V
GND	GND	GND
DIN	MOSI	19
CLK	SCLK	23
CS	CE0	24
DC	25	22

RST	27	13
BL	18	12

1.28inch LCD 使用的是PH2.0 8PIN接口，对照上述表格连接在树莓派上即可： （请按照引脚定义表格连接，图中排线颜色仅供参考，以实际颜色为准。）



(/wiki/%E6%96%87%E4%BB%B6:1.28-%E6%A0%91%E8%8E%93%E6%B4%BE.jpg)

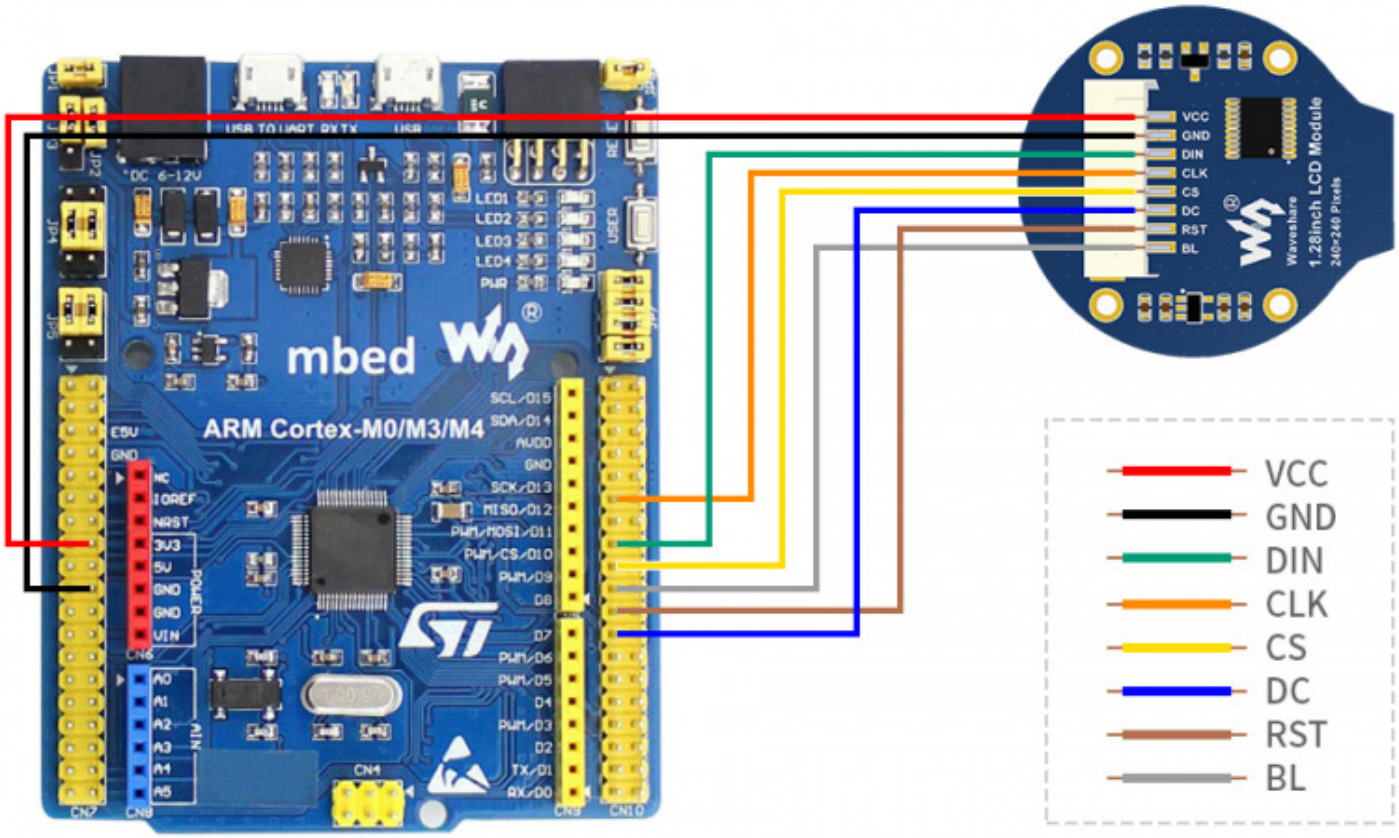
STM32硬件连接

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，如果有需要移植程序，请按实际引脚连接

STM32F103ZET连接引脚对应关系

LCD	STM32
VCC	3.3V
GND	GND
DIN	PA7
CLK	PA5
CS	PB6
DC	PA8
RST	PA9
BL	PC7

以本公司研发的XNUCLEO-F103RB开发板 (https://www.waveshare.net/shop/XNUCLEO-F103RB.htm)为例，连接如下图：



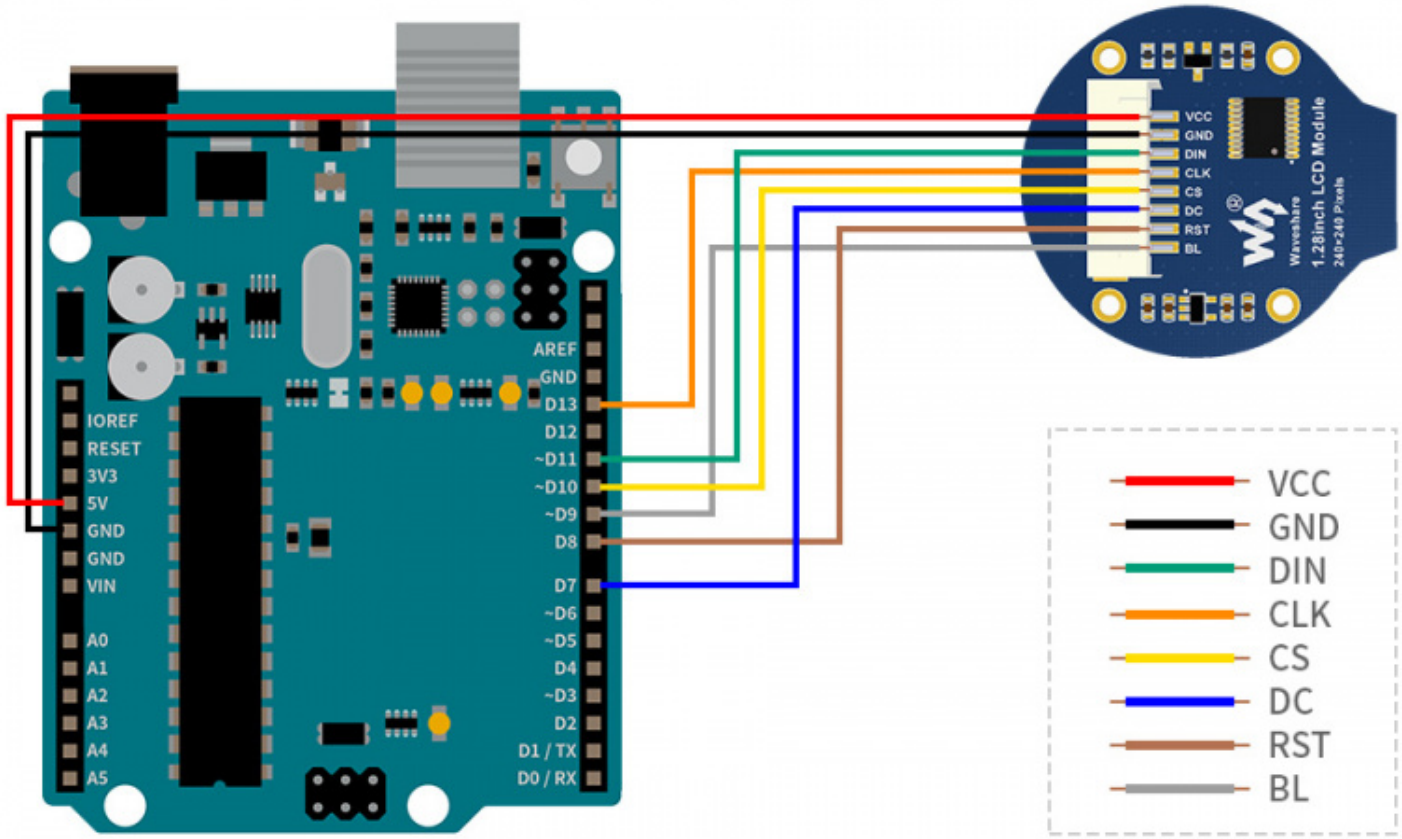
(/wiki/%E6%96%87%E4%BB%B6:1.28-STM32.jpg)

Arduino硬件连接

Arduino UNO连接引脚对应关系

LCD	UNO
VCC	5V
GND	GND
DIN	D11
CLK	D13
CS	D10
DC	D7
RST	D8
BL	D9

连接图如下(点击可放大)：



(/wiki/%E6%96%87%E4%BB%B6:1.28-Aduino.jpg)

LCD 及其控制器

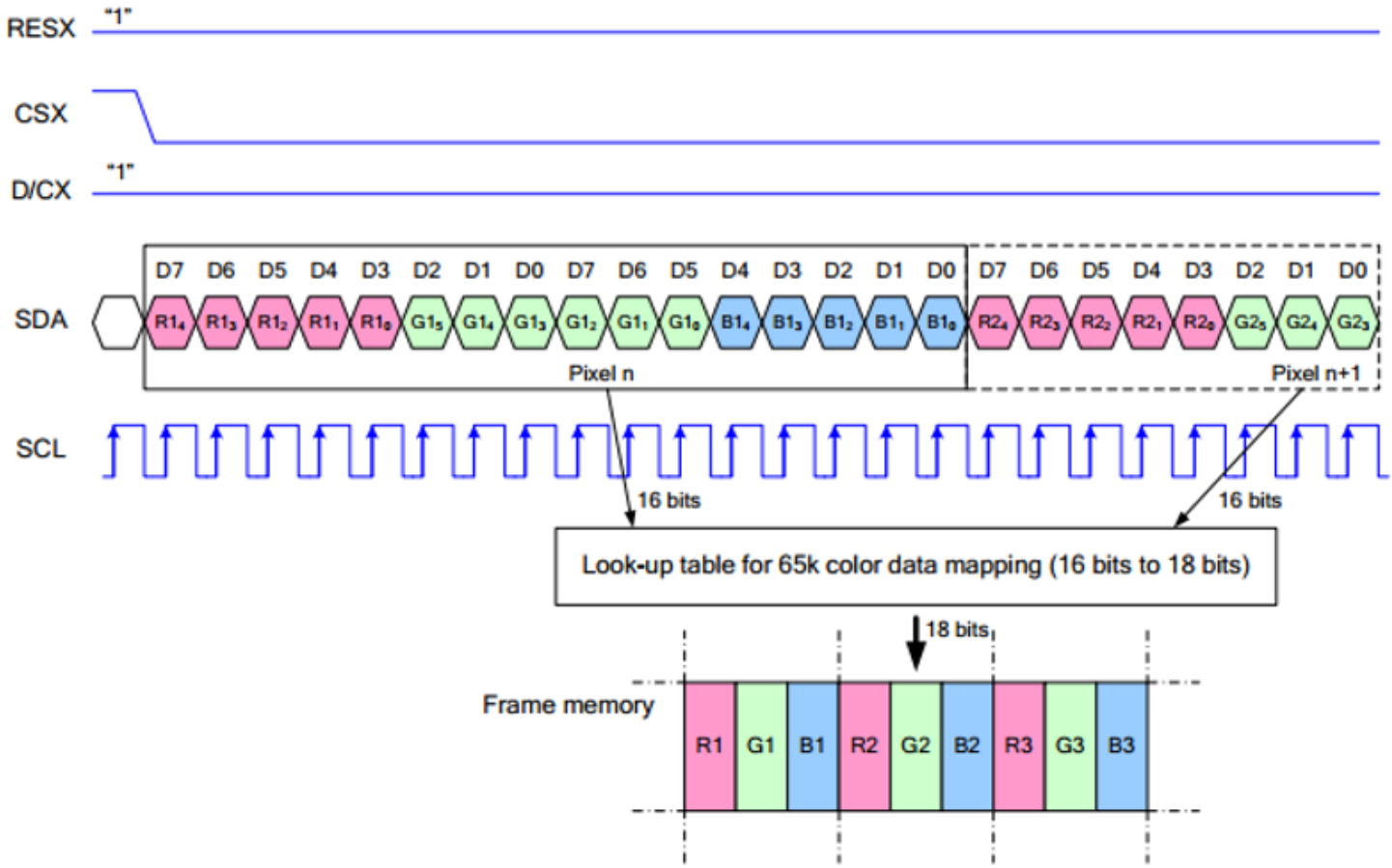
本款LCD使用的内置驱动器位GC9A01，分辨率为240RGB×240 dots，内部有129600字节的GRAM，支持12/16/18位数据总线MCU接口，即RGB444，RGB565，RGB666三种颜色格式，这也是常用的RGB格式。

对于大部分的LCD控制器而言，都可以配置控制器的通信方式，通常都有8080并行接口、三线SPI、四线SPI等通信方式。此LCD使用四线SPI通信接口，这样可以大大的节省GPIO口，同时通信速度也会比较快。

- 可能有的小伙伴们就会有疑问了，屏幕是圆形形状的，那哪一点才是屏幕的第一个像素点呢？怎么确定坐标呢？

- 其实呀你可以理解为就是一个正方形的屏幕在里面画了一个内切圆，我们只在这个内切圆中显示内容，其他位置的像素点就直接丢弃了，市面上的圆形LCD大多也是如此。

通信协议



(/wiki/%E6%96%87%E4%BB%B6:0.96inch_lcd_module_spi.png)

注：与传统的SPI协议不同的地方是：由于是只需要显示，故而将从机发往主机的数据线进行了隐藏，该表格详见Datasheet Page 105。

RESX为复位，模块上电时拉低，通常情况下置1；

CSX为从机片选， 仅当CS为低电平时，芯片才会被使能。

D/CX为芯片的数据/命令控制引脚，当DC = 0时写命令，当DC = 1时写数据

SDA为传输的数据，即RGB数据；

SCL为SPI通信时钟。

对于SPI通信而言，数据是有传输时序的，即时钟相位（CPHA）与时钟极性(CPOL)的组合：

CPHA的高低决定串行同步时钟是在第一时钟跳变沿还是第二个时钟跳变沿数据被采集，当CPHA = 0，在第一个跳变沿进行数据采集；

CPOL的高低决定串行同步时钟的空闲状态电平，CPOL = 0，为低电平。

从图中可以看出，当SCLK第一个下降沿时开始传输数据，一个时钟周期传输8bit数据，使用SPI0，按位传输,高位在前,低位在后。

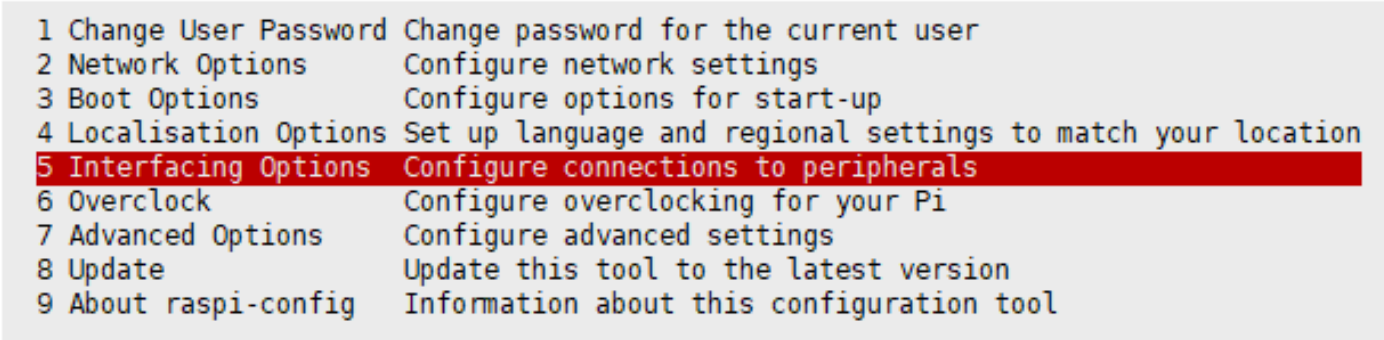
树莓派使用教程

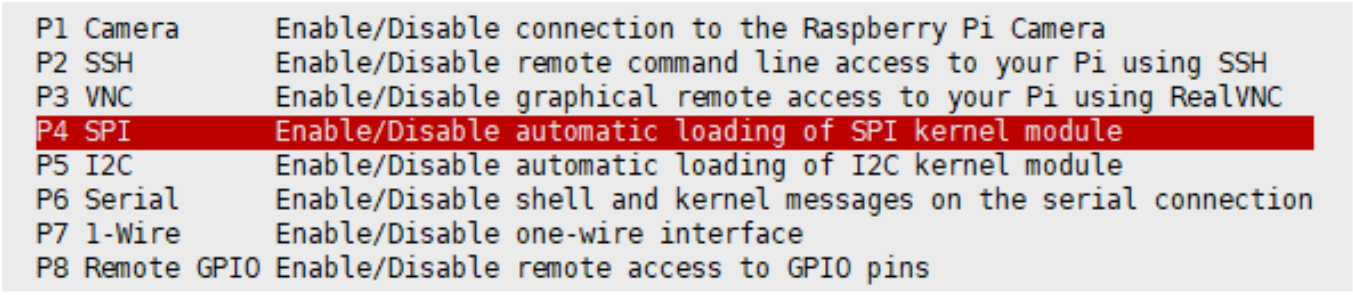
开启SPI接口

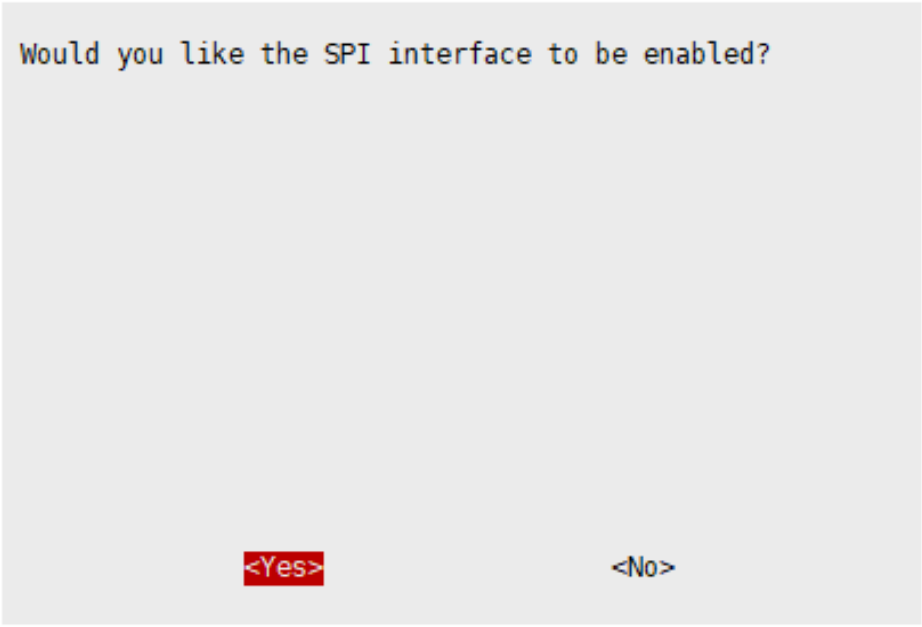
PS：如果使用的是Bullseye分支的系统,需要将”apt-get“改成“apt"，Bullseye分支的系统只支持Python3。

- 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
```







(/wiki/%E6%96%87%E4%BB%B6:RPI_open_spi.png)

然后重启树莓派：

```
sudo reboot
```

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

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
cd
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2.70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 unzip -y
sudo wget https://www.waveshare.net/w/upload/8/8d/LCD_Module_RPI_code.zip
sudo unzip ./LCD_Module_RPI_code.zip
cd LCD_Module_RPI_code/RaspberryPi/
```

运行测试程序

以下命令请在RaspberryPi下执行，否则不在索引不到目录；

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：

```
#0.96inch LCD Module
sudo ./main 0.96
#1.14inch LCD Module
sudo ./main 1.14
#1.28inch LCD Module
sudo ./main 1.28
#1.3inch LCD Module
sudo ./main 1.3
#1.47inch LCD Module
sudo ./main 1.47
#1.54inch LCD Module
sudo ./main 1.54
#1.8inch LCD Module
sudo ./main 1.8
#2inch LCD Module
sudo ./main 2
#2.4inch LCD Module
sudo ./main 2.4
```

python

- 进入python程序目录,并运行指令ls -l

```
cd python/examples
ls -l
```

```
pi@eng33:~/LCD_Module_code/RaspberryPi/python/example $ ls -l
total 24
-rw-r--r-- 1 pi pi 2830 Jun 16 17:59 0inch96_LCD_test.py
-rw-r--r-- 1 pi pi 2459 Jun 16 18:34 1inch14_LCD_test.py
-rw-r--r-- 1 pi pi 2701 Jun 16 18:33 1inch3_LCD_test.py
-rw-r--r-- 1 pi pi 2665 Jun 16 17:58 1inch54_LCD_test.py
-rw-r--r-- 1 pi pi 2678 Jun 16 18:34 1inch8_LCD_test.py
-rw-r--r-- 1 pi pi 2660 Jun 16 18:39 2inch_LCD_test.py
```

(/wiki/%E6%96%87%E4%BB%B6:LCD_rpi_python_examples.png)

可以查看到所有屏幕的测试程序，按照尺寸分类：

0inch96_LCD_test.py	0.96inch LCD测试程序
1inch14_LCD_test.py	1.14inch LCD测试程序
1inch28_LCD_test.py	1.28inch LCD测试程序
1inch3_LCD_test.py	1.3inch LCD测试程序
1inch47_LCD_test.py	1.47inch LCD测试程序
1inch54_LCD_test.py	1.54inchLCD测试程序
1inch8_LCD_test.py	1.8inch LCD测试程序
2inch_LCD_test.py	2inch LCD测试程序
2inch4_LCD_test.py	2.4inch LCD测试程序

- 运行对应屏幕的程序即可，程序支持python2/3

```
# python2
sudo python 0inch96_LCD_test.py
sudo python 1inch14_LCD_test.py
sudo python 1inch28_LCD_test.py
sudo python 1inch3_LCD_test.py
sudo python 1inch47_LCD_test.py
sudo python 1inch54_LCD_test.py
sudo python 1inch8_LCD_test.py
sudo python 2inch_LCD_test.py
sudo python 2inch4_LCD_test.py
# python3
sudo python3 0inch96_LCD_test.py
sudo python3 1inch14_LCD_test.py
sudo python3 1inch28_LCD_test.py
sudo python3 1inch3_LCD_test.py
sudo python3 1inch47_LCD_test.py
sudo python3 1inch54_LCD_test.py
sudo python3 1inch8_LCD_test.py
sudo python3 2inch_LCD_test.py
sudo python3 2inch4_LCD_test.py
```

API详解(请选读c或python部分)

RaspberryPi系列均可以共用一套程序，因为他们都是嵌入式系统，兼容性比较强。

程序分为底层硬件接口、中间层液晶屏驱动、上层应用；

C

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，在目录：RaspberryPi\c\lib\Config

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
```

(/wiki/%E6%96%87%E4%BB%B6:LCD_rpi_c_Makefile.png)

- 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。
```

- GPIO读写：

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

- SPI写数据

```
void DEV_SPI_WriteByte(UBYTE Value);
```

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，在目录：RaspberryPi\c\lib\GUI\GUI_Paint.c(.h)

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

(/wiki/%E6%96%87%E4%BB%B6:LCD_rpi_GUI.png)

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

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

(/wiki/%E6%96%87%E4%BB%B6:LCD_rpi_Font.png)

- 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 Color)
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;
```

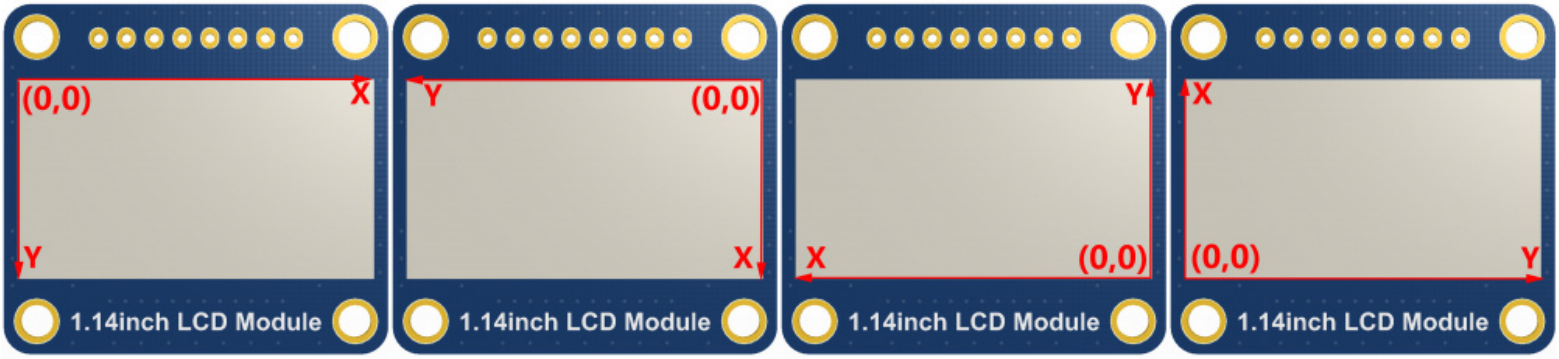
- 选择图像缓存:选择图像缓存，选择的目的是你可以创建多个图像属性，图像缓存可以存在多个，你可以选择你所创建的每一张图像

```
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
```

【说明】不同选择角度下，坐标对应起始像素点不同，这里以1.14为例，四张图，按顺序为0°， 90°， 180°，270°。仅做为参考



旋转0度　　旋转90度　　旋转180度　　旋转270度

(/wiki/%E6%96%87%E4%BB%B6:LCD_Rotate.jpg)

- 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 Color)
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 Color, LINE_STY
LE 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, DOT_PIXEL Li
ne_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 Col
or_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* 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
        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* 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
        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, int32_t Nummber, 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
        Nummber：显示的数字，这里使用的是32位长的int型保存，可以最大显示到2147483647
        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
```

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

```
void Paint_DrawFloatNum(UWORD Xpoint, UWORD Ypoint, double Nummber,  UBYTE Decimal_Point,
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
        Nummber：显示的数字，这里使用的是double型保存，足够普通需求
        Decimal_Point：显示小数点后几位数字
        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, UWORD Color_B
ackground, 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
```

- 读取本地的bmp图片并写到缓存中

对于Raspberry Pi这些Linux操作系统的，可以读写图片

对于Raspberry Pi，在目录：RaspberryPi\c\lib\GUI\GUI_BMPfile.c(.h)

```
UBYTE GUI_ReadBmp(const char *path, UWORD Xstart, UWORD Ystart)
参数：
        path：BMP图片的相对路径
        Xstart：图片的左顶点X坐标，一般默认传0
        Ystart：图片的左顶点Y坐标，一般默认传0
```

用户测试代码

前三个章节介绍了经典的linux三层代码结构，这里稍微讲解一下用户测试代码

对于Raspberry Pi，在目录：RaspberryPi\c\examples，为全部的测试代码；

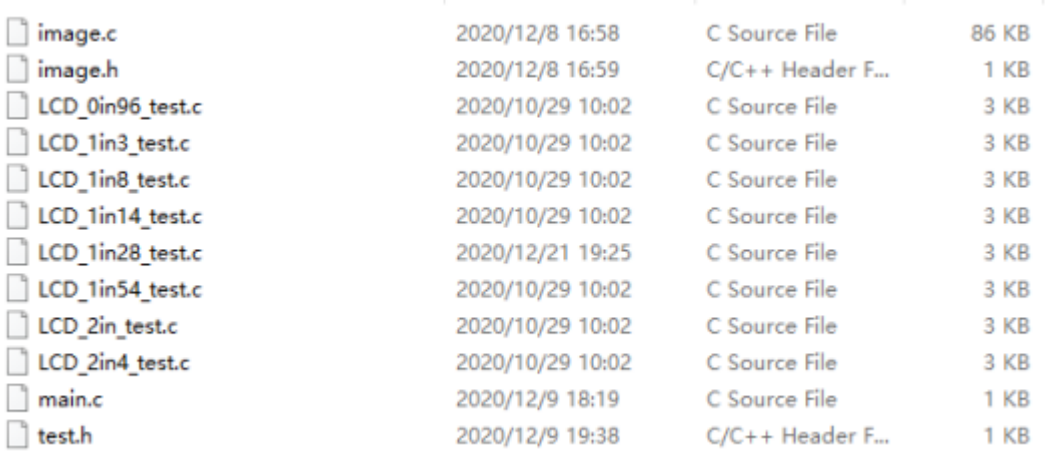


image.c	2020/12/8 16:58	C Source File	86 KB
image.h	2020/12/8 16:59	C/C++ Header F...	1 KB
LCD_0in96_test.c	2020/10/29 10:02	C Source File	3 KB
LCD_1in3_test.c	2020/10/29 10:02	C Source File	3 KB
LCD_1in8_test.c	2020/10/29 10:02	C Source File	3 KB
LCD_1in14_test.c	2020/10/29 10:02	C Source File	3 KB
LCD_1in28_test.c	2020/12/21 19:25	C Source File	3 KB
LCD_1in54_test.c	2020/10/29 10:02	C Source File	3 KB
LCD_2in_test.c	2020/10/29 10:02	C Source File	3 KB
LCD_2in4_test.c	2020/10/29 10:02	C Source File	3 KB
main.c	2020/12/9 18:19	C Source File	1 KB
test.h	2020/12/9 19:38	C/C++ Header F...	1 KB

(/wiki/%E6%96%87%E4%BB%B6:LCD_rpi_c_examples%26128.png)

如果需要运行0.96inch LCD测试程序，你需要运行mian程序时加入 0.96作为参数

在linux命令模式下重新执行如下：

```
make clean
make
sudo ./main 0.96
```

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

Raspberry Pi：RaspberryPi\python\lib\

名称	修改日期	类型	大小
__init__.py	2020/5/21 15:39	Python File	0 KB
LCD_0inch96.py	2020/6/16 17:41	Python File	6 KB
LCD_1inch3.py	2020/6/16 17:41	Python File	5 KB
LCD_1inch8.py	2020/6/16 17:41	Python File	9 KB
LCD_1inch14.py	2020/6/16 17:41	Python File	5 KB
LCD_1inch54.py	2020/6/16 17:41	Python File	5 KB
LCD_2inch.py	2020/6/8 14:17	Python File	6 KB
lcdconfig.py	2020/6/8 9:27	Python File	4 KB

(/wiki/%E6%96%87%E4%BB%B6:LCD_rpi_python_lib.png)

lcdconfig.py

- 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
```

- GPIO读写：

```
def  digital_write(pin, value)
def  digital_read(pin)
```

- SPI写数据

```
def spi_writebyte(data)
```

- xxx_LCD_test.py(xxx表示尺寸，若是0.96inch LCD，则为0inch96_LCD_test.py,依此类推)

python在如下目录：

Raspberry Pi：RaspberryPi\python\examples\

名称	修改日期	类型	大小
0inch96_LCD_test.py	2020/6/16 17:59	Python File	3 KB
1inch3_LCD_test.py	2020/6/16 18:33	Python File	3 KB
1inch8_LCD_test.py	2020/6/16 18:34	Python File	3 KB
1inch14_LCD_test.py	2020/6/16 18:34	Python File	3 KB
1inch54_LCD_test.py	2020/6/16 17:58	Python File	3 KB
2inch_LCD_test.py	2020/6/16 18:39	Python File	3 KB

(/wiki/%E6%96%87%E4%BB%B6:LCD_rpi_python_examples2.png)

如果你的python版本是python2，且需要运行0.96inch LCD测试程序，在linux命令模式下重新执行如下：

```
sudo python 0inch96_LCD_test.py
```

如果你的python版本是python3，且需要运行0.96inch LCD测试程序，在linux命令模式下重新执行如下：

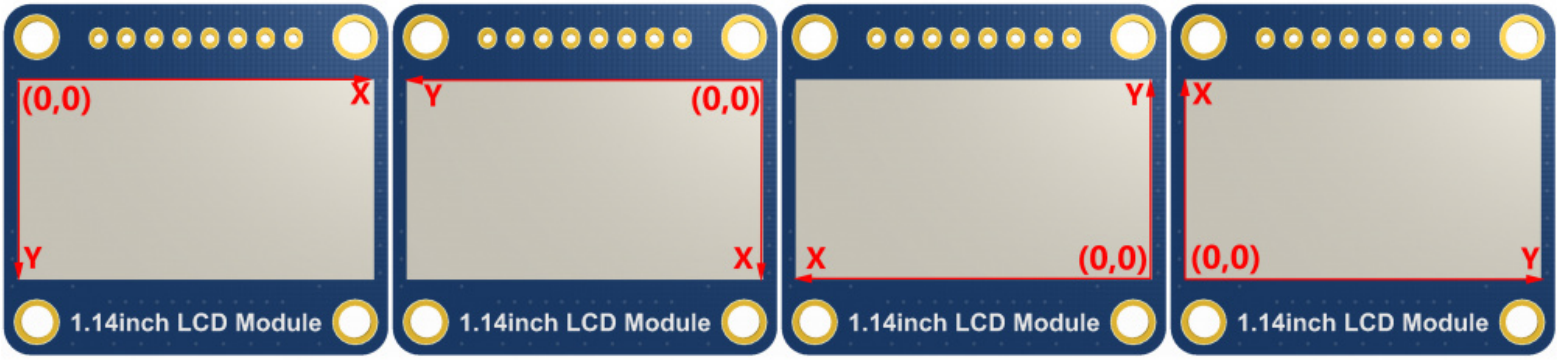
```
sudo python3 0inch96_LCD_test.py
```

关于旋转设置

如果在python程序中你需要设置屏幕旋转，可以通过语句im_r= image1.rotate(270)设置。

```
im_r= image1.rotate(270)
```

旋转效果，以1.54为例， 按顺序分别为0°， 90°，180°， 270°



旋转0度　旋转90度　旋转180度　旋转270度

(/wiki/%E6%96%87%E4%BB%B6:LCD_Rotate.jpg)

画图GUI

由于python有一个image库pil官方库链接 (http://effbot.org/imagingbook)，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("RGB", (disp.width, disp.height), "WHITE")
```

第一个参数定义图片的颜色深度，定义为"RGB"说明是RGB888彩色图，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([(20, 10),(70, 60)], fill = "RED",width = 1)
```

第一个参数为一个4个元素的元组，以（20，10）为起始点，（70，60）为终止点，画一条直线，fill="RED"表示线为红色，width=1表示线宽为1个像素。

- 画框

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

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="BLUE"表示外边框为蓝色。

- 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 =(0,255,0)表示线为绿色
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可以画圆之外，还有ellipse可以画实心圆

```
draw.ellipse((150,65,190,105), fill = (0,255,0))
```

实质是椭圆的绘制，第一个参数指定弦的圆外切矩形，fill =(0,255,0)表示内部填充颜色为绿色，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((40, 50), 'WaveShare', fill = (128,255,128),font=Font2)
text= u"微雪电子"
draw.text((74, 150),text, fill = "WHITE",font=Font3)
```

第一个参数为一个2个元素的元组，以（40，50）为左顶点，字体为Font2，fill为字体颜色，你可以直接让 fill = "WHITE"，因为常规的颜色的值已经定义好了，当然你也可以使用fill = (128,255,128)，括号里对应的是RGB三种颜色的值，这样你就能精确的控制你想要的颜色。第二句显示微雪电子，使用Font3，字体颜色为白色。

- 读取本地图片

```
image = Image.open('../pic/LCD_1inch28.jpg')
```

参数为图片路径。

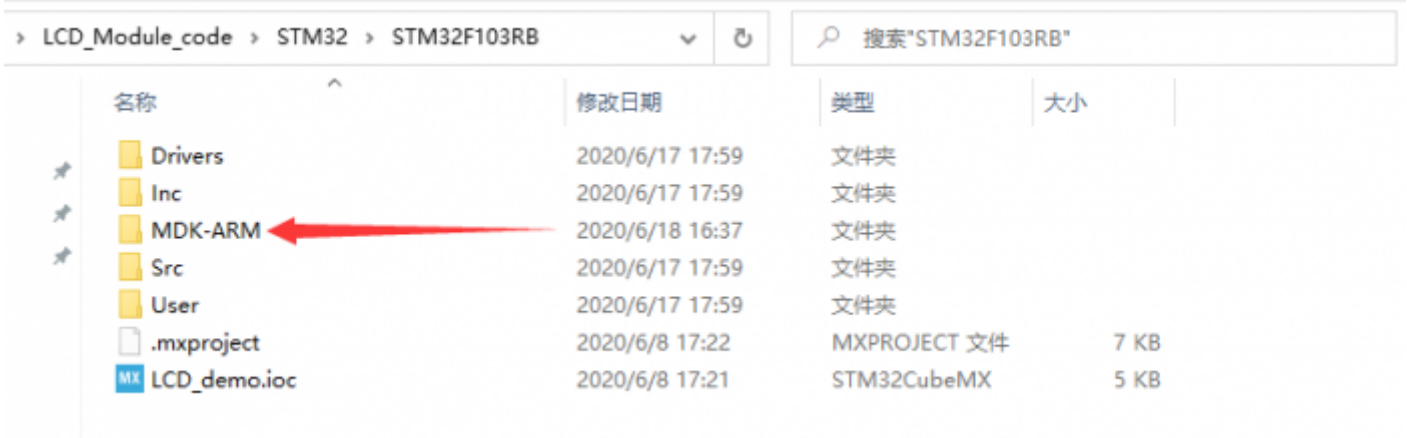
- 其他功能

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软件说明

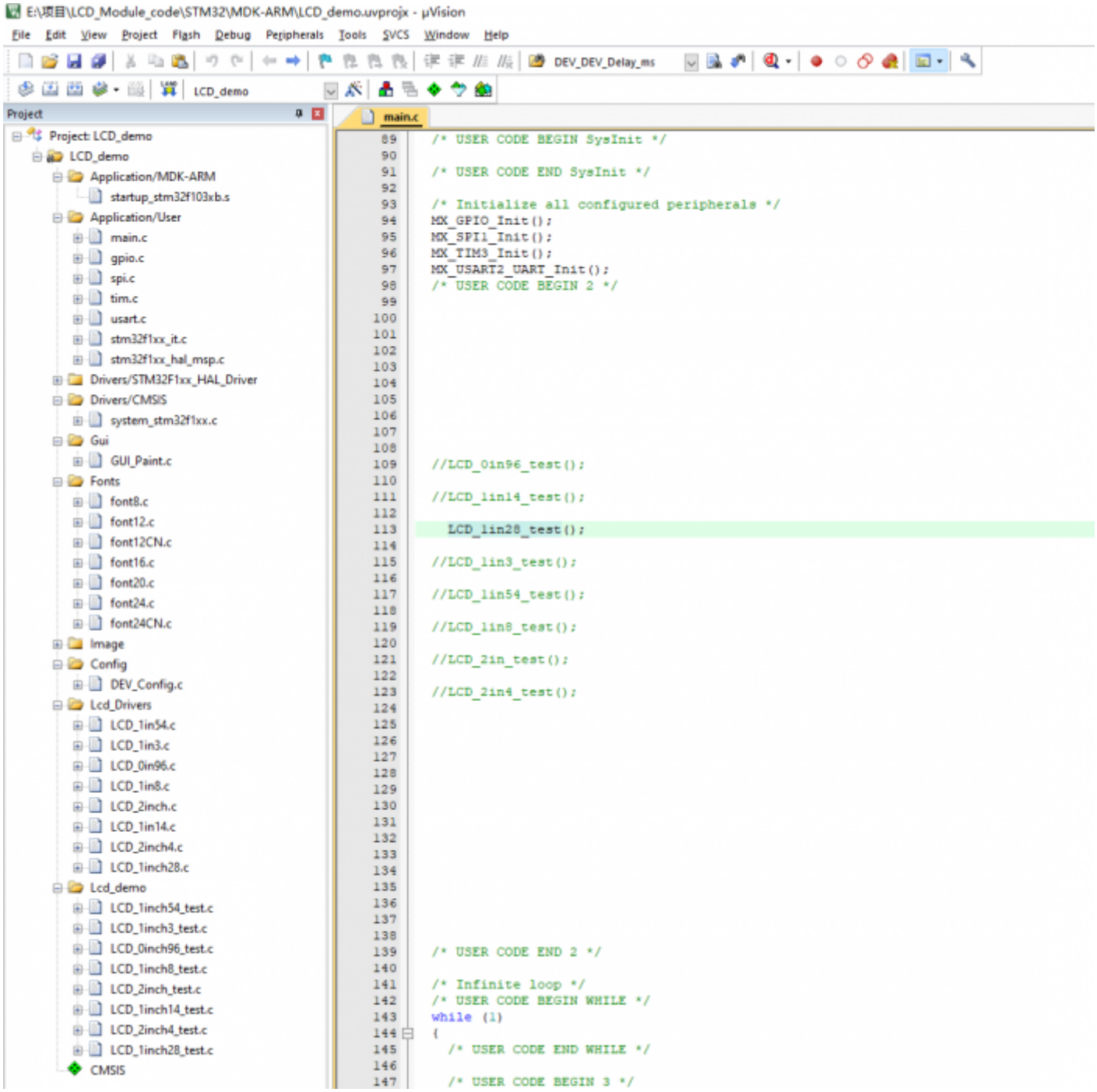
[折叠]

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LCD_demo.uvprojx，即可看到程序。



(/wiki/%E6%96%87%E4%BB%B6:LCD_STM32_CODE1.png)

打开main.c，可以看到所有的测试程序，把对应的屏幕的测试程序前面的注释去掉，重新编译下载即可。



(/wiki/%E6%96%87%E4%BB%B6:LCD_STM32_CODE128.png)

LCD_0in96_test()　0.96inch LCD测试程序
LCD_1in14_test()　1.14inch LCD测试程序
LCD_1in28_test()　1.28inch LCD测试程序
LCD_1in3_test()　1.3 inch LCD测试程序

LCD_1in47_test() 1.47 inch LCD测试程序
LCD_1in54_test() 1.54inch LCD测试程序
LCD_1in8_test()   1.8inch   LCD测试程序
LCD_2in_test()   2inch   LCD测试程序

程序说明

底层硬件接口

- 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；
2.DEV_Module_Exit函数使用后，会关闭LCD显示屏；
```

- GPIO读写：

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

- SPI写数据

```
void    DEV_SPI_WRITE(UBYTE _dat);
```

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，在目录：STM32\STM32F103RB\User\GUI_DEV\GUI_Paint.c(.h)

注：因为STM32和arduino内部RAM大小的原因，GUI都采用直接写入LCD的RAM中。

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

(/wiki/%E6%96%87%E4%BB%B6:LCD_rpi_GUI.png)

在如下目录下是GUI依赖的字符字体，在目录：STM32\STM32F103RB\User\Fonts

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

(/wiki/%E6%96%87%E4%BB%B6:LCD_rpi_Font.png)

- 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LCD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，通常直接调用LCD的DrawPaint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LCD内部RAM指定位置写入数据；
```

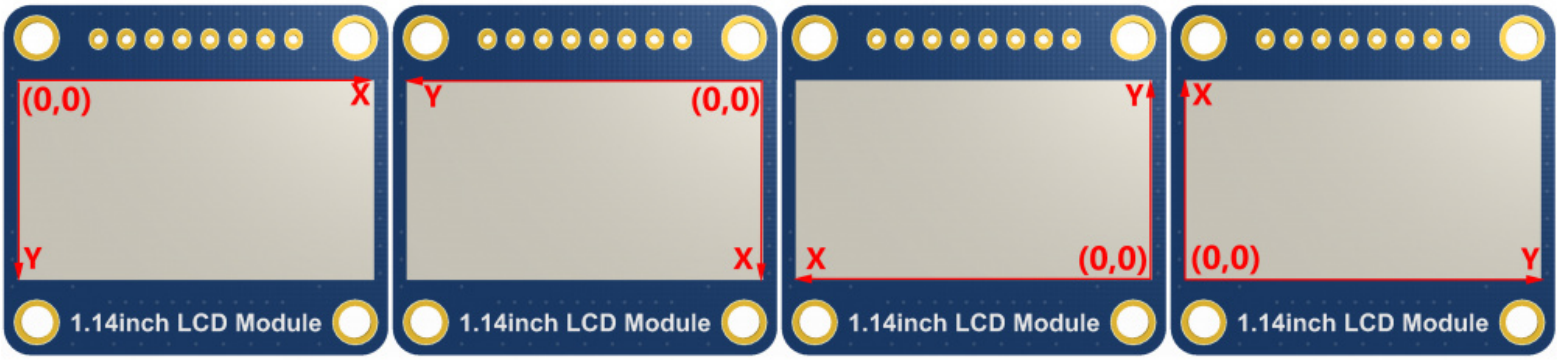
- 选择图像缓存:选择图像缓存，选择的目的是你可以创建多个图像属性，图像缓存可以存在多个，你可以选择你所创建的每一张图像

```
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
```

【说明】不同选择角度下，坐标对应起始像素点不同，这里以1.14为例，四张图，按顺序为0°， 90°， 180°，270°。仅做为参考



旋转0度　　旋转90度　　旋转180度　　旋转270度

(/wiki/%E6%96%87%E4%BB%B6:LCD_Rotate.jpg)

- 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 Color)
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 Color, LINE_STY
LE 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, DOT_PIXEL Li
ne_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 Col
or_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* 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
        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* 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
        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, int32_t Nummber, 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
        Nummber：显示的数字，这里使用的是32位长的int型保存，可以最大显示到2147483647
        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
```

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

```
void Paint_DrawFloatNum(UWORD Xpoint, UWORD Ypoint, double Nummber,  UBYTE Decimal_Point,
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
        Nummber：显示的数字，这里使用的是double型保存，足够普通需求
        Decimal_Point：显示小数点后几位数字
        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, UWORD Color_B
ackground, 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软件说明

[折叠]

注：例程均在Arduino uno上进行了测试，如果需要是其他型号的Arduino需要自己确定连接的管脚是否正确。

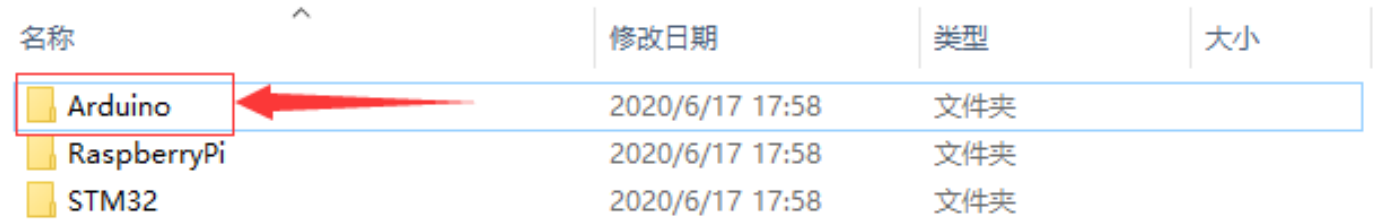
arduino IDE 安装教程

arduino IDE 安装教程 (https://www.waveshare.net/wiki/Arduino_ide_download)

运行程序

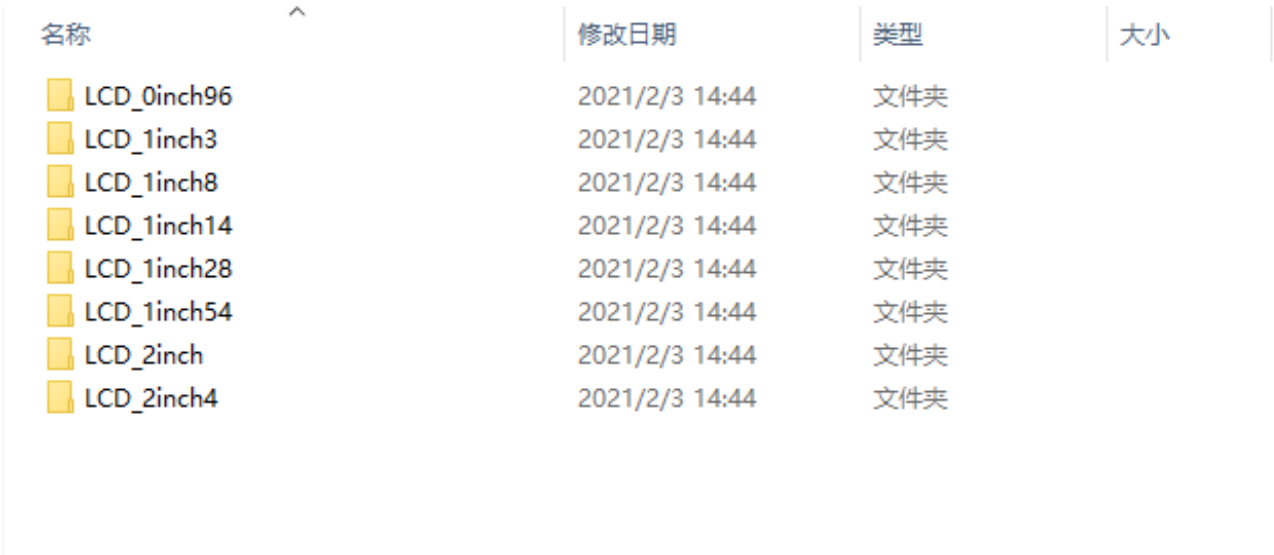
在产品百科界面下载程序 (https://www.waveshare.net/w/upload/f/fc/LCD_Module_code.zip)，然后

解压。Arduino程序位于 ~/Arduino/...



(/wiki/%E6%96%87%E4%BB%B6:LCD_arduino_cede1.png)

请根据液晶屏型号选择对应的程序打开

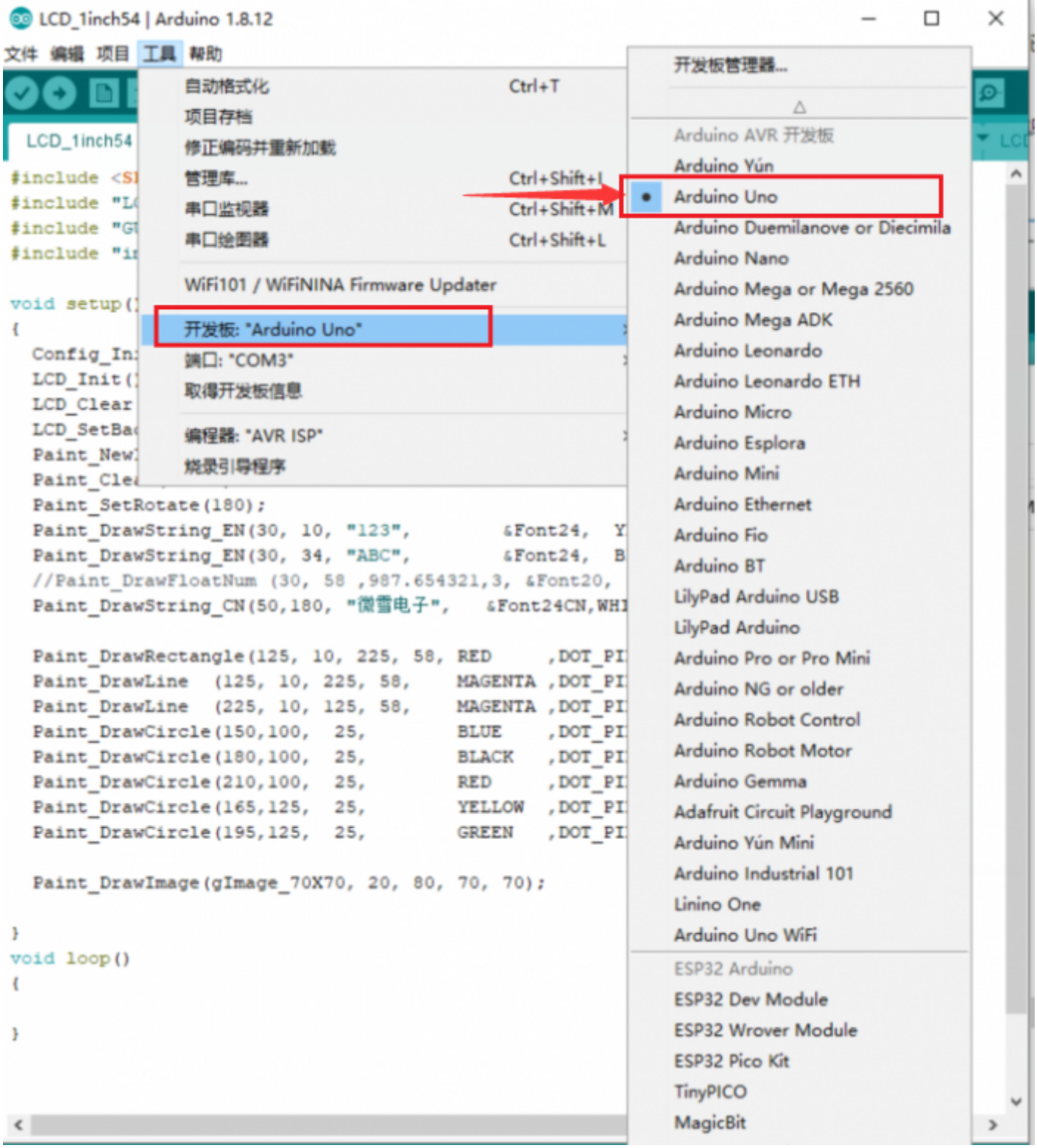


(/wiki/%E6%96%87%E4%BB%B6:1.28inch_LCD_Arduino.png)

可以查看到所有的屏幕尺寸的测试程序，按照尺寸分类：

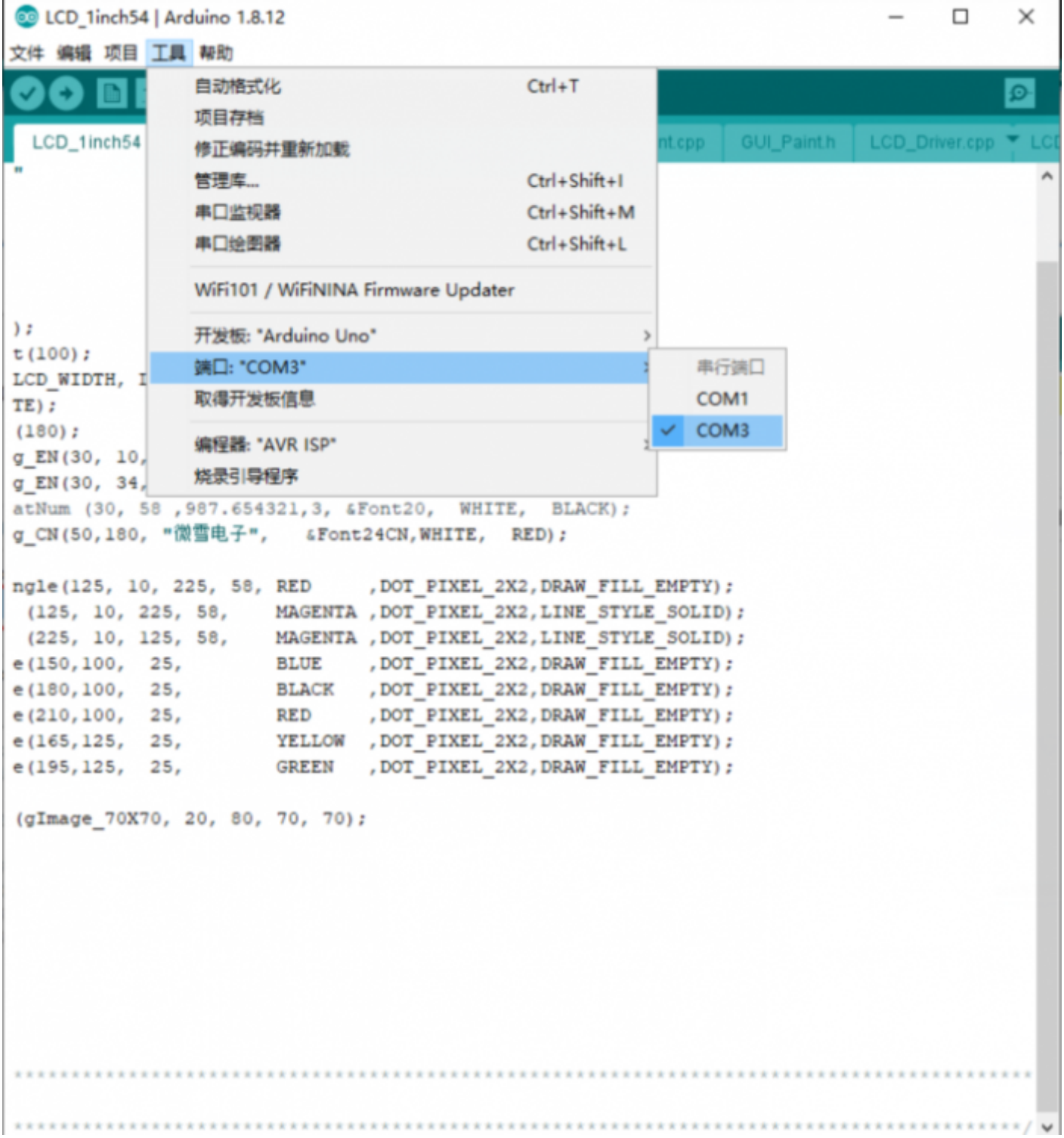
比如1.54inch LCD Module. 打开LCD_1inch54文件夹，并运行LCD_1inch54.ino文件。

打开程序，选择开发板型号Arduino UNO



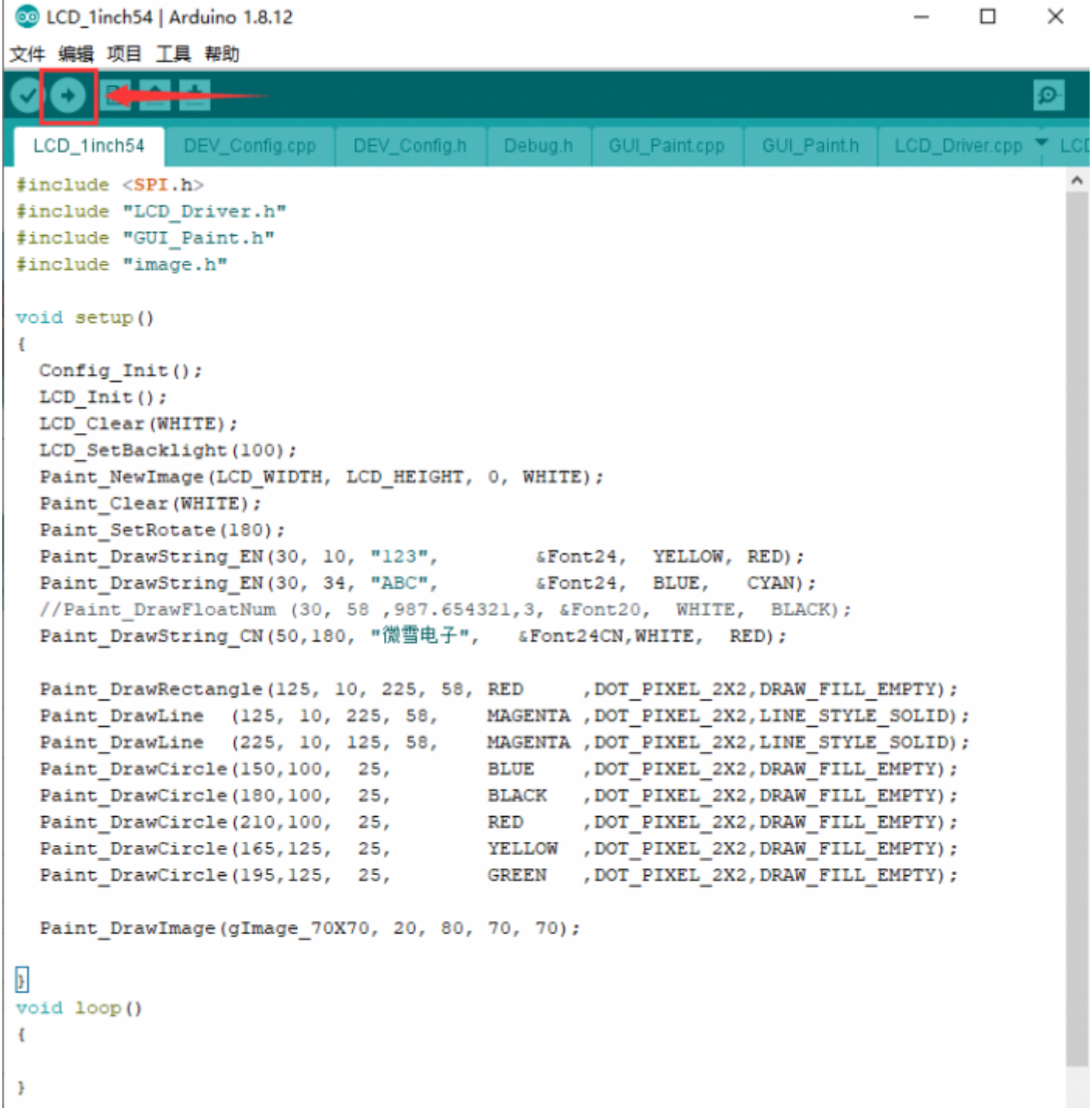
(/wiki/%E6%96%87%E4%BB%B6:LCD_arduino_cede3.png)

选择对应COM口



(/wiki/%E6%96%87%E4%BB%B6:LCD_arduino_cede4.png)

然后点击编译并下载即可



(/wiki/%E6%96%87%E4%BB%B6:LCD_arduino_cede5.png)

程序说明

文件介绍

以Arduino UNO控制1.54寸 LCD 为例，打开Arduino\LCD_1inch54目录：

名称	修改日期	类型	大小
Debug.h	2020/6/9 18:11	H 文件	1 KB
DEV_Config.cpp	2020/6/9 18:11	CPP 文件	2 KB
DEV_Config.h	2020/6/9 18:11	H 文件	3 KB
font8.cpp	2020/6/9 18:11	CPP 文件	19 KB
font12.cpp	2020/6/9 18:11	CPP 文件	6 KB
font16.cpp	2020/6/9 18:11	CPP 文件	51 KB
font20.cpp	2020/6/9 18:11	CPP 文件	67 KB
font24.cpp	2020/6/9 18:11	CPP 文件	100 KB
font24CN.cpp	2020/6/9 18:11	CPP 文件	28 KB
fonts.h	2020/6/9 18:11	H 文件	4 KB
GUI_Paint.cpp	2020/6/13 16:32	CPP 文件	27 KB
GUI_Paint.h	2020/6/10 14:25	H 文件	7 KB
image.cpp	2020/6/9 18:11	CPP 文件	50 KB
image.h	2020/6/9 18:11	H 文件	1 KB
LCD_1inch54.ino	2020/6/9 18:12	Arduino file	2 KB
LCD_Driver.cpp	2020/6/9 18:55	CPP 文件	8 KB
LCD_Driver.h	2020/6/9 18:11	H 文件	2 KB

(/wiki/%E6%96%87%E4%BB%B6:LCD_arduino_ide_codeDescription1.png)

其中：

LCD_1inch54.ino：使用Arduino IDE打开即可；

LCD_Driver.cpp(.h):是液晶屏的驱动程序；

DEV_Config.cpp(.h):是硬件接口定义，里面封装了读写管脚电平，SPI传输数据，以及管脚初始化；

font8.cpp、font12.cpp、font16.cpp、font20.cpp、font24.cpp、font24CN.cpp、fonts.h：为不同大小字符的字模；

image.cpp(.h)：是图片数据，这个可以通过Img2Lcd(在开发资料中可下载)把任意的BMP图片转换成16位真彩色图片数组。

程序分为底层硬件接口、中间层液晶屏驱动、上层应用；

底层硬件接口

在DEV_Config.cpp(.h)两个文件中定义了硬件接口，并封装好读写管脚电平、延时、SPI传输等函数。

写管脚电平

```
void DEV_Digital_Write(int pin, int value)
```

第一个参数为管脚、第二个为高低电平。

读管脚电平

```
int DEV_Digital_Read(int pin)
```

参数为管脚，返回值为读取管脚的电平。

延时

```
DEV_Delay_ms(unsigned int delaytime)
```

毫秒级别延时。

SPI输出数据

```
DEV_SPI_WRITE(unsigned char data)
```

参数为char型，占8位。

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 GUI_Paint.c(.h)

注：因为STM32和arduino内部RAM大小的原因，GUI都采用直接写入LCD的RAM中。

(/wiki/%E6%96%87%E4%BB%B6:LCD_arduino_ide_codeDescription_GUI.png)

GUI使用的字体均依赖于相同文件下的font*.cpp（h）文件

名称	修改日期	类型	大小
Debug.h	2020/6/9 18:11	H 文件	1 KB
DEV_Config.cpp	2020/6/9 18:11	CPP 文件	2 KB
DEV_Config.h	2020/6/9 18:11	H 文件	3 KB
font8.cpp	2020/6/9 18:11	CPP 文件	19 KB
font12.cpp	2020/6/9 18:11	CPP 文件	6 KB
font16.cpp	2020/6/9 18:11	CPP 文件	51 KB
font20.cpp	2020/6/9 18:11	CPP 文件	67 KB
font24.cpp	2020/6/9 18:11	CPP 文件	100 KB
font24CN.cpp	2020/6/9 18:11	CPP 文件	28 KB
fonts.h	2020/6/9 18:11	H 文件	4 KB
GUI_Paint.cpp	2020/6/13 16:32	CPP 文件	27 KB
GUI_Paint.h	2020/6/10 14:25	H 文件	7 KB
image.cpp	2020/6/9 18:11	CPP 文件	50 KB
image.h	2020/6/9 18:11	H 文件	1 KB
LCD_1inch54.ino	2020/6/9 18:12	Arduino file	2 KB
LCD_Driver.cpp	2020/6/9 18:55	CPP 文件	8 KB
LCD_Driver.h	2020/6/9 18:11	H 文件	2 KB

(/wiki/%E6%96%87%E4%BB%B6:LCD_arduino_ide_codeDescription_font.png)

- 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LCD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，通常直接调用LCD的DrawPaint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LCD内部RAM指定位置写入数据；
```

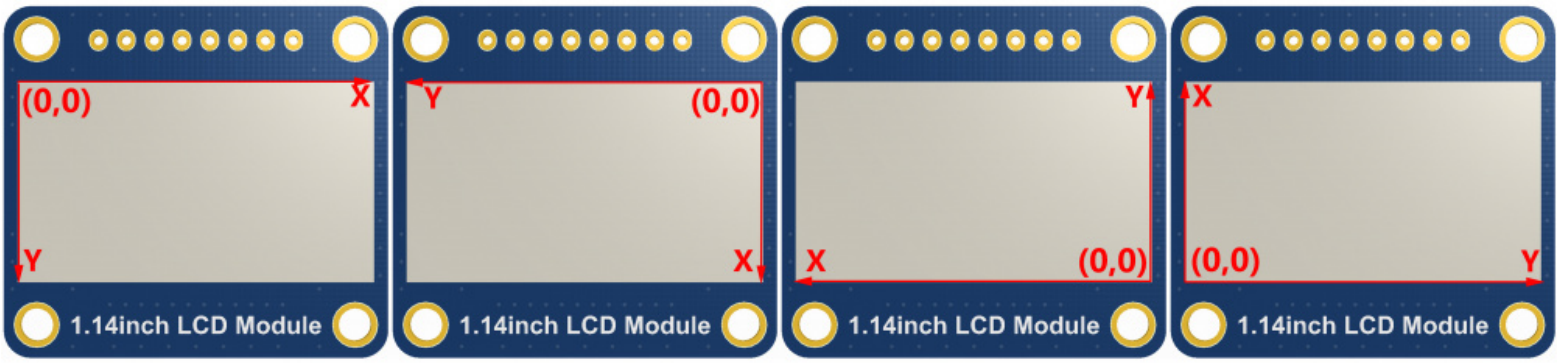
- 选择图像缓存:选择图像缓存，选择的目的是你可以创建多个图像属性，图像缓存可以存在多个，你可以选择你所创建的每一张图像

```
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
```

【说明】不同选择角度下，坐标对应起始像素点不同，这里以1.14为例，四张图，按顺序为0°， 90°， 180°，270°。仅做为参考



旋转0度　旋转90度　旋转180度　旋转270度

(/wiki/%E6%96%87%E4%BB%B6:LCD_Rotate.jpg)

- 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 Color)
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（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：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 Color, LINE_STY
LE 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: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: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, DOT_PIXEL Li
ne_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 (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 Col
or_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 (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* 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
        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 (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* 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
        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 (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, int32_t Nummber, 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
        Nummber：显示的数字，这里使用的是32位长的int型保存，可以最大显示到2147483647
        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
```

- 写带小数的数字:在 (Xstart Ystart) 为左顶点，写一串数字可以带小数的数字，可以选择Ascii码可视字符字库、字体前景色、字体背景色

```
void Paint_DrawFloatNum(UWORD Xpoint, UWORD Ypoint, double Nummber,  UBYTE Decimal_Point,
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
        Nummber：显示的数字，这里使用的是double型保存
        Decimal_Point：显示小数点后几位数字
        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 (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, UWORD Color_B
ackground, 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
```

- 显示图像:在 (Xstart Ystart) 为左顶点，显示一个宽为W_Image，高为H_Image的图像;

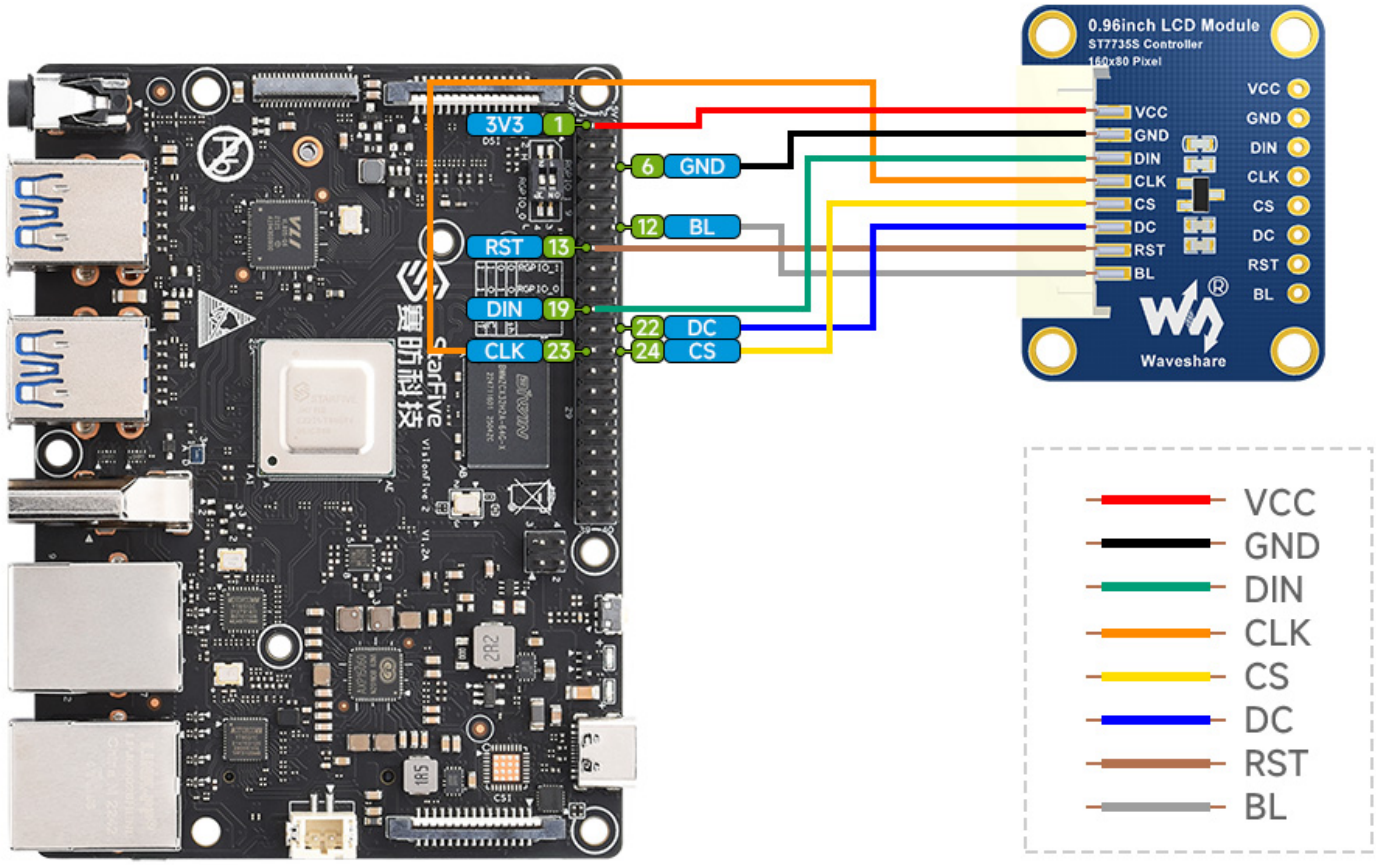
```
void Paint_DrawImage(const unsigned char *image, UWORD xStart, UWORD yStart, UWORD W_Image, U
WORD H_Image)
参数：
        image:图像地址，指向想要显示的图像信息
        Xstart：字符的左顶点X坐标
        Ystart：字体的左顶点Y坐标
        W_Image:图像宽度
        H_Image:图像高度
```


[折叠]

适配型号

[展开]

硬件连接



(/wiki/%E6%96%87%E4%BB%B6:0.96-VisionFive2.jpg)

VisionFive2连接引脚对应关系

LCD	VisionFive2 Board物理引脚序号
VCC	3.3V
GND	GND
DIN	19
CLK	23
CS	24
DC	22
RST	13
BL	12

安装对应的库文件

```
apt-get install pip
pip install VisionFive.gpio
apt-get install python3-numpy
apt-get install python3-pil
```

下载程序

```
apt-get install p7zip-full
wget https://www.waveshare.net/w/upload/e/e9/LCD_Module_code.7z (https://www.waveshare.net/w/upload/e/e9/LCD_Module_code.7z)
7z x LCD_Module_code.7z -o./LCD_Module_code
cd LCD_Module_code/VisionFive/python/example/
```

根据您所购买的屏幕运行下面对应的程序

```
python3 0inch96_LCD_test.py
python3 1inch14_LCD_test.py
python3 1inch28_LCD_test.py
python3 1inch3_LCD_test.py
python3 1inch54_LCD_test.py
python3 1inch8_LCD_test.py
python3 2inch_LCD_test.py
python3 2inch4_LCD_test.py
```

资料

文档

- 原理图 (https://www.waveshare.net/w/upload/b/b4/1.28inch_LCD_Module_SchDoc.pdf)

3D 图纸

- 1.28inch LCD Module 3D 图纸 (https://www.waveshare.net/w/upload/4/49/1.28inch_LCD_Module_3D_Drawing.zip)

程序

- 示例程序 (https://www.waveshare.net/w/upload/f/fc/LCD_Module_code.zip)

软件

- 汉字取模软件 (https://www.waveshare.net/w/upload/c/c6/Zimo221.7z)
- Image2Lcd 图片取模软件 (https://www.waveshare.net/w/upload/b/bd/Image2Lcd2.9.zip)
- 图片取模教程 (https://www.waveshare.net/wiki/Image_extraction)

数据手册

- GC9A01A 手册 (https://www.waveshare.net/w/upload/5/5e/GC9A01A.pdf)

FAQ

问题：1.28inch LCD Module 最大功耗是?

3.3V 31.2mA