Pico-OLED-1.3

来自Waveshare Wiki
跳转至： 导航、 搜索

功能简介	
1.3英寸	
64 × 128	
I2C	SPI

说明

产品概述

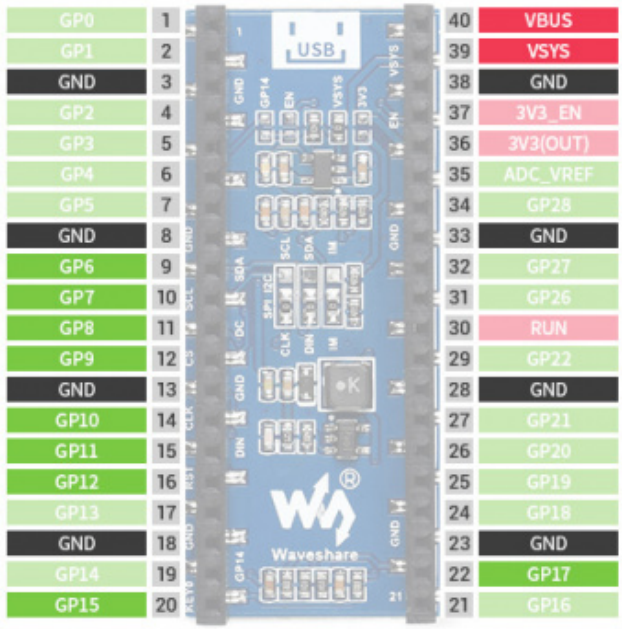
提供Pico C语言例程

产品参数

参数名称	参数
供电电压	2.6V ~ 5.5V
工作电流	40mA
控制芯片	SH1107
通信接口	4-wire SPI / I2C
分辨率	64 x 128 Pixels

像素大小	0.15 × 0.15mm
显示尺寸	14.70 × 29.42mm
产品尺寸	52.00 x25.00(mm)

接口说明

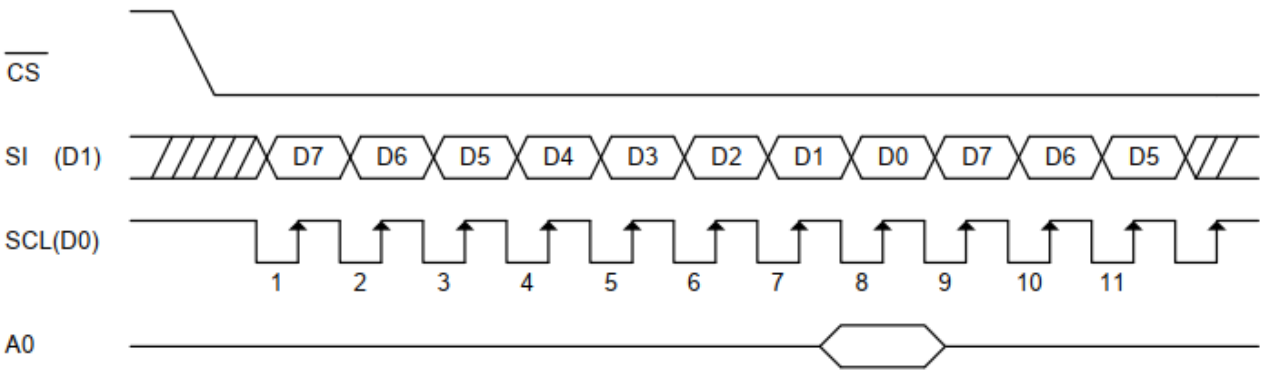


VBUS		电源供电 3.3V~5.5V
VSYS		电源供电 1.8V~5.5V
GND		电源地
GP6	I2C_SDA	数据输入
GP7	I2C_SCL	时钟输入
GP8	OLED_DC	数据/命令，低电平表示命令，高电平表示数据
GP9	OLED_CS	片选，低电平有效
GP10	OLED_CLK	时钟信号输入
GP11	OLED_DIN	数据输入
GP12	OLED_RST	复位，低电平有效
GP15	KEY0	用户按键0
GP17	KEY1	用户按键1

LCD 及其控制器

本款OLED使用的内置控制器为SH1107，它有128 × 128 bits SRAM，支持最大128 × 128像素屏幕，支持SPI/I2C/ 6800并口/8080并口，256级亮度设置，本屏幕为64 × 128 像素，所以内部SRAM没有完全使用。 本模块采用四线SPI和IIC两种接口，兼容性好，传输速度高。

通信协议



注：该模块的四线SPI是没有MISO的，不是比较流行的四线SPI，具体请见Datasheet Page 11。

CS为从机片选， 仅当CS为低电平时，芯片才会被使能；

SI（D1）即MOSI，为主设备数据输出，从设备数据输入；
SCL（D0）为SPI通信时钟；
A0即DC，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

对于SPI通信而言，数据是有传输时序的，即数据的捕获需要时钟信号的某个边沿触发，而这个边沿就是由时钟极性(CPOL)与时钟相位（CPHA）的组合决定的：
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；CPOL = 1，为高电平。
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；CPHA = 1，在第二个跳变沿进行数据采集。
从图中可以看出，SCL空闲时是高电平，在第二个边沿时开始传输数据，所以是模式三(0x11)，一个时钟周期传输8bit数据，按位传输,高位在前,低位在后(MSB)。

Pico快速上手

Pico百科

- 树莓派Pico百科（墙裂推荐）

固件下载

MicroPython固件下载

C_Blink固件下载 展开

视频教程（更新中）

PICO系列教程1——基础介绍 展开

PICO系列教程2——外设GPIO 展开

PICO系列教程3——PWM(脉冲宽度调制) 展开

PICO系列教程4——ADC（模拟数字转换器） 展开

PICO系列教程5——UART （异步收发传输器） 展开

PICO系列教程6——I2C（集成电路总线） 展开

PICO系列教程7——SPI（串行外设接口） 展开

文字教程（更新中）

基础介绍

Raspberry Pi Pico的基础介绍

MicroPython系列 展开

C/C++系列 展开

开源例程

MircoPython视频例程(github)
MicroPython固件/Blink例程（C）
树莓派官方C/C++示例程序 (github)
树莓派官方micropython示例程序 (github)

硬件连接

连接Pico的时候，请注意对应方向不要接反。可以观察模块上有USB丝印的一端与Pico的USB接口一端来判断方向（也可以根据墨水屏模块上的排母的引脚标号与Pico的引脚标号判断）
您可以对照以下表格连线。

Pico连接引脚对应关系

e-Paper	Pico	功能
VCC	VSYS	电源输入
GND	GND	电源地
DIN	GP11	SPI通信MOSI引脚，从设备数据输入
CLK	GP10	SPI通信SCK引脚，从设备时钟输入
CS	GP9	SPI片选引脚（低电平有效）
DC	GP8	数据/命令控制引脚（高电平数据，低电平命令）
RST	GP12	外部复位引脚（电平有效）

直连

注：请按左图所示，
连接模块和树莓派 Pico，勿接反。

扩展板连接

环境搭建

正在整理

请参照树莓派官方网站的Pico专题：

https://www.raspberrypi.org/documentation/pico/getting-started/

程序下载

打开树莓派终端，执行：

方法一：从我们官网下载，推荐使用。

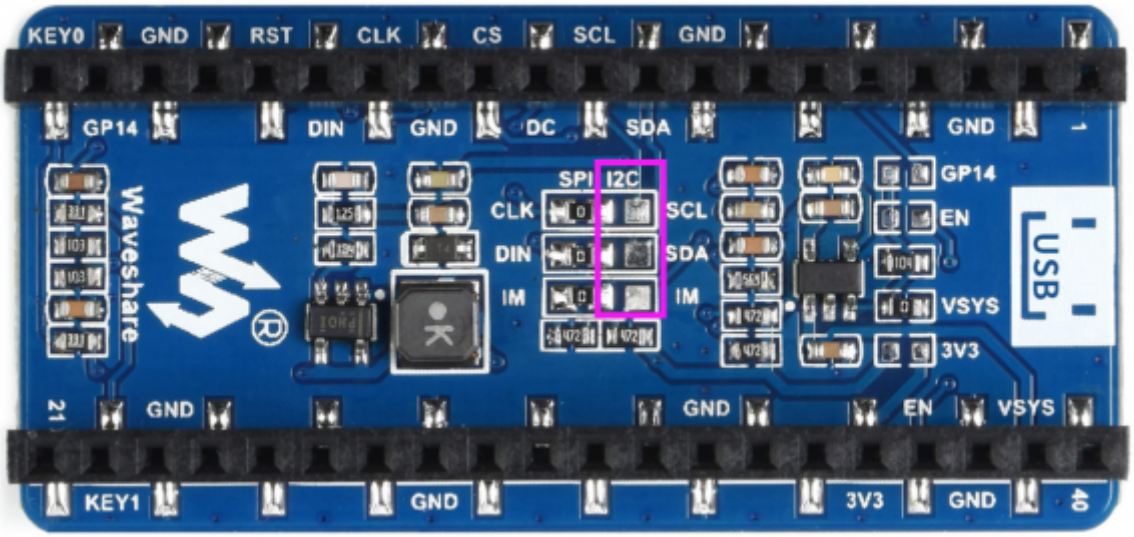
```
sudo apt-get install p7zip-full
cd ~
sudo wget  https://www.waveshare.net/w/upload/3/3d/Pico_OLED_code.7z
7z x Pico_OLED_code.7z -o./Pico_OLED_code
cd ~/Pico_OLED_code
cd c/build/
```

例程使用

以下教程为在树莓派上操作，但由于cmake的多平台、可移植的特点，在PC上也是能成功编译，但操作略有不同，需要您自行判断。 如果是测试可以使用uf2目录下编译好的文件即可

I2C

模块默认是使用SPI接口，如果需要使用I2C设备，需要修改背面电阻



C部分

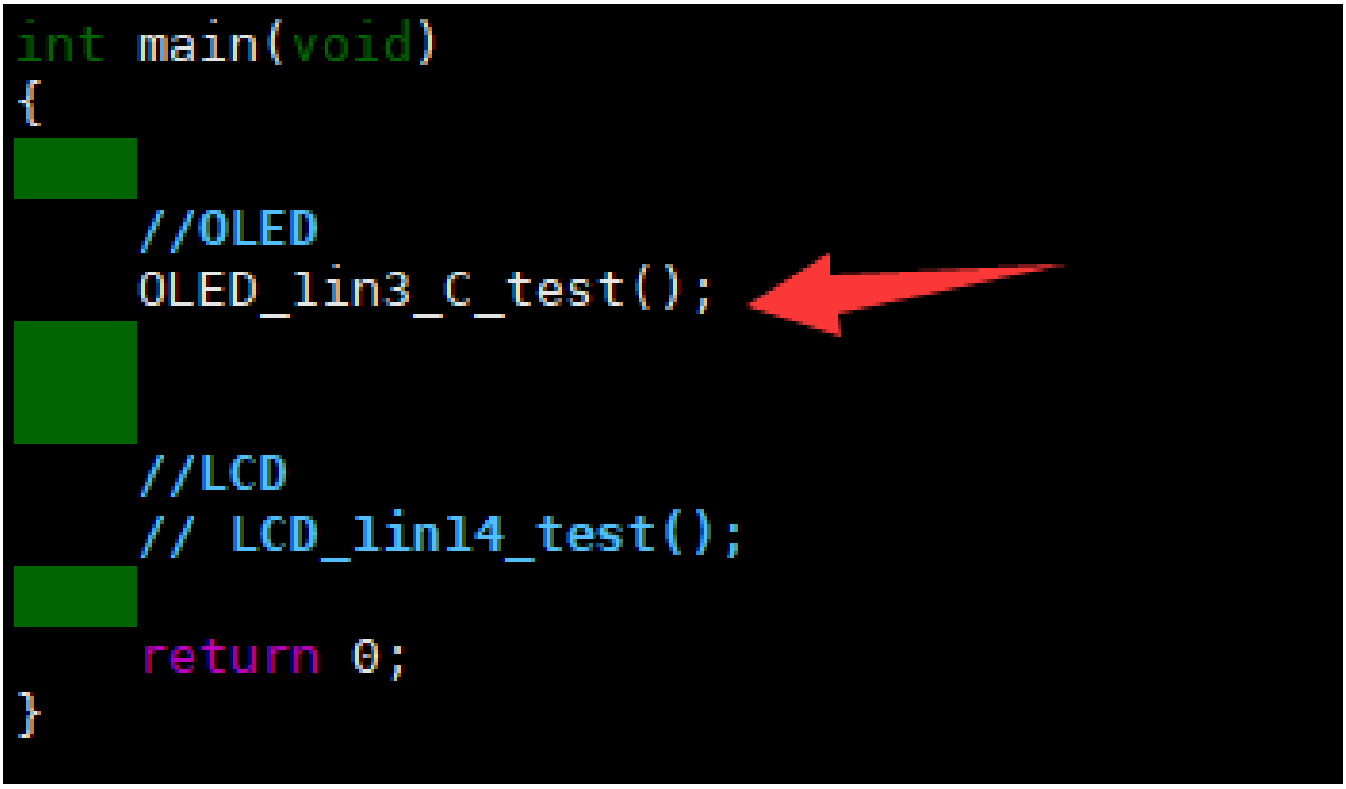
进行编译，请确保在c目录：

```
cd ~/Pico_OLED_code/c/
```

打开main.c选择对应的模块

```
sudo nano main.c
```

如果你使用的是Pico-OLED-1.3,那么就去掉OLED_1in3_C_test()函数前面的//，然后按ctrl+c，然后按Y键并回车保存并退出，具体如下：



创建并进入build目录,并添加SDK: 其中 ../../pico-sdk 是你的SDK的目录。 如果存在build，则直接进入

```
#mkdir build
cd build
export PICO_SDK_PATH=../../pico-sdk
#export PICO_SDK_PATH=/home/pi/pico/pico-sdk
```

执行cmake自动生成Makefile文件 如果build 目录下存在编译文件, 编译的环境和你的环境不一致会导致生成make文件失败,需要删除目录下的所有文件

```
cmake ..
```

执行make生成可执行文件，第一次编译时间比较久

```
make -j
```

编译完成，会生成uf2文件。 按住Pico板上的按键，将pico通过Micro USB线接到树莓派的USB接口，然后松开按键。接入之后，树莓派会自动识别到一个可移动盘（RPI-RP2），将build文件夹下的main.uf2 文件复制到识别的可移动盘（RPI-RP2）中即可。

```
cp main.uf2 /media/pi/RPI-RP2/
```

代码简析

如果您以前使用过我们的SPI屏幕应该会对这份例程比较熟悉

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：...\c\lib\Config

- 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：..\c\lib\GUI\GUI_Paint.c(.h)

GUI_Paint.c	2021/2/1 11:18	C 文件	32 KB
GUI_Paint.h	2021/2/1 11:17	H 文件	6 KB

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

- 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 *image, UWORD Width, UWORD Height, UWORD Rotate, UWORD Co
lor, UWORD Depth)
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；
        Depth ： 颜色深度
```

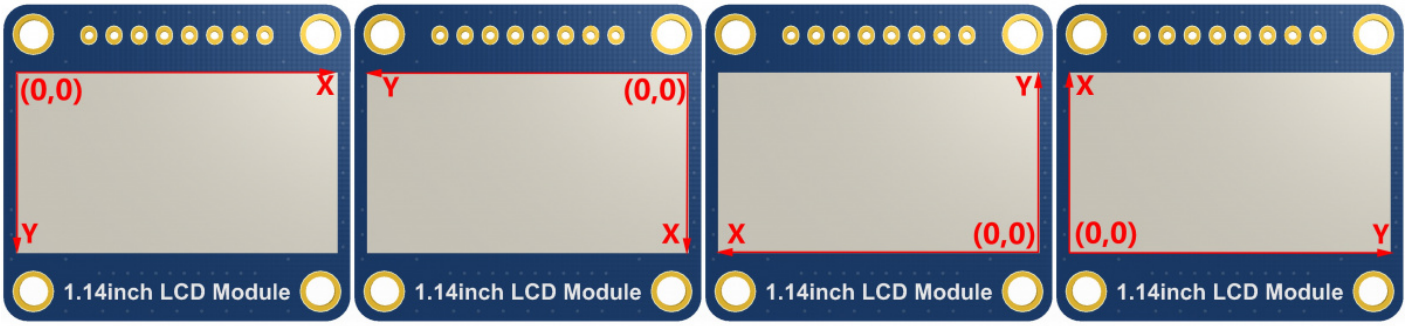
- 选择图像缓存:选择图像缓存，选择的目的是你可以创建多个图像属性，图像缓存可以存在多个，你可以选择你所创建的每一张图像

```
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



0 90 180 270

- 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, 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, 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

配套资料

文档

- 原理图

程序

- 示例程序

软件

- 汉字取模软件
- Image2Lcd 图片取模软件

数据手册

- SH1107 手册

开发软件

- Thonny Python IDE (Windows版本 V3.3.3)
- Pico环境搭建相关软件 (百度网盘提取码：prgc)

Pico系列教程

树莓派Pico 视频教程　展开

树莓派Pico C/C++ SDK 入门教程　展开

Pico开源例程

树莓派官方Pico资料链接
MircoPython视频例程(github)
MicroPython固件/Blink例程（C）
树莓派官方C/C++示例程序 (github)
树莓派官方micropython示例程序 (github)