

1.51inch Transparent OLED

From Waveshare Wiki

Jump to: navigation, search

Overview

Provides RPI, STM32, Arduino demos.

Specification

- Working Voltage: 5V/3.3V
- Communication Interface: SPI/I2C
- Controller: SSD1309
- Resolution: 128 × 64
- Display Size: 35.05 × 15.32 (mm)
- Pixel Size: 0.254 × 0.254 (mm)
- Board Size: 41 × 22.5 (mm)
- Display Color: Light Blue

Pin Function

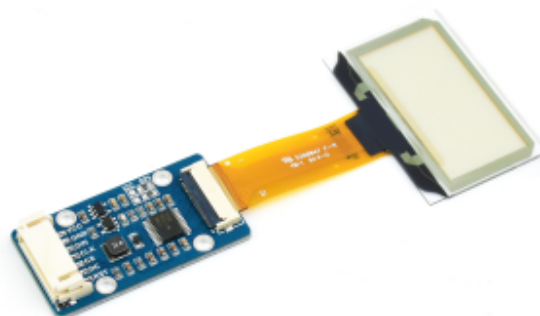
Pin Function	Description
VCC	5V/3.3V Power Input
GND	Ground
DIN	Data Input
CLK	Clock Input
CS	Chip selection, low active
DC	Data/Command selection (high for data, low for command)
RST	Reset, low active

Hardware Configuration

There are two driver interfaces on the OLED module: 4-wire SPI and I2C interfaces respectively. Two resistors can be soldered on the back of the module. The corresponding communication mode can be selected through the selection of resistors, as shown in the figure:

The module uses a 4-wire SPI communication mode by default, that is, the resistor is connected to the SPI position by default. The specific hardware configuration is as follows:

1.51inch Transparent OLED



(<https://www.waveshare.com/1.51inch-transparent-oled.htm>)

128 x 64
SPI, I2C

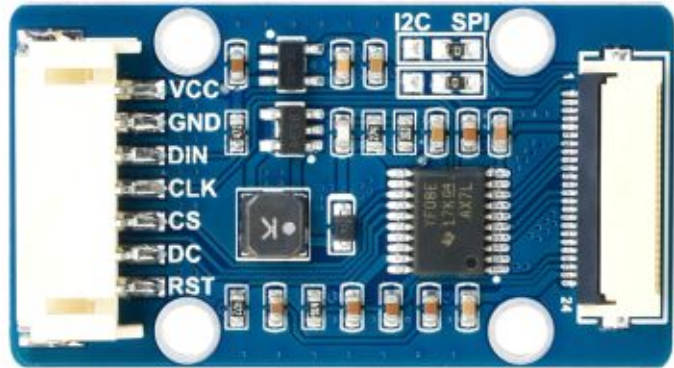
- Using 4-wire SPI:

That is, the factory program setting: two 0R resistors are connected to the SPI;

- Using I2C:

Two 0R resistors are connected to the I2C;

PS: The program defaults to SPI mode. If you need to switch the mode, please modify DEV_Config.h. For details, please refer to the program description - bottom hardware interface - interface selection.



(/wiki/File:1.51inch_Transparent_OLED001.jpg)

Hardware Configuration

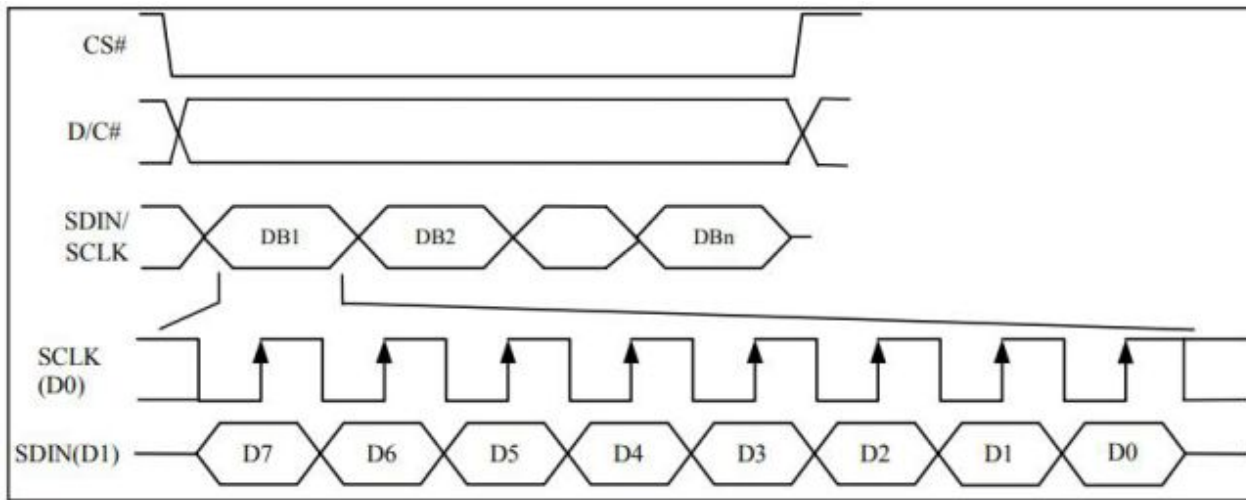
OLED & Controller

The integrated built-in controller used by this OLED is SSD1309, which has 128 × 64 bits SRAM, supports up to 128 × 64 pixel screen, supports SPI/I2C/ 6800 parallel port/8080 parallel port, 256-level brightness settings, this screen is 128 × 64 pixels, so the internal SRAM is not fully used.

This module adopts four-wire SPI and IIC interfaces, with good compatibility and high transmission speed.

SPI Communication Protocol

- The 4-wire serial interface consists of SCLK, SDIN, D/C#, and CS#. In 4-wire SPI mode, D0 is SCLK and D1 is SDIN. For unused data pins, D2 should be kept open. From pins D3 to D7, "E" and "R/W# (WR#)#" can be connected to an external ground.
- SDIN is shifted into an 8-bit shift register in the order D7, D6, ... on each rising edge of SCLK. D0. D/C# samples every 8 clocks and the data bytes in the shift register are written to the RAM (GDDRAM) or command register where the graphics display data is within the same clock.



(/wiki/File:1.51inch_Transparent_OLED002.jpg)

User Guides of RPI

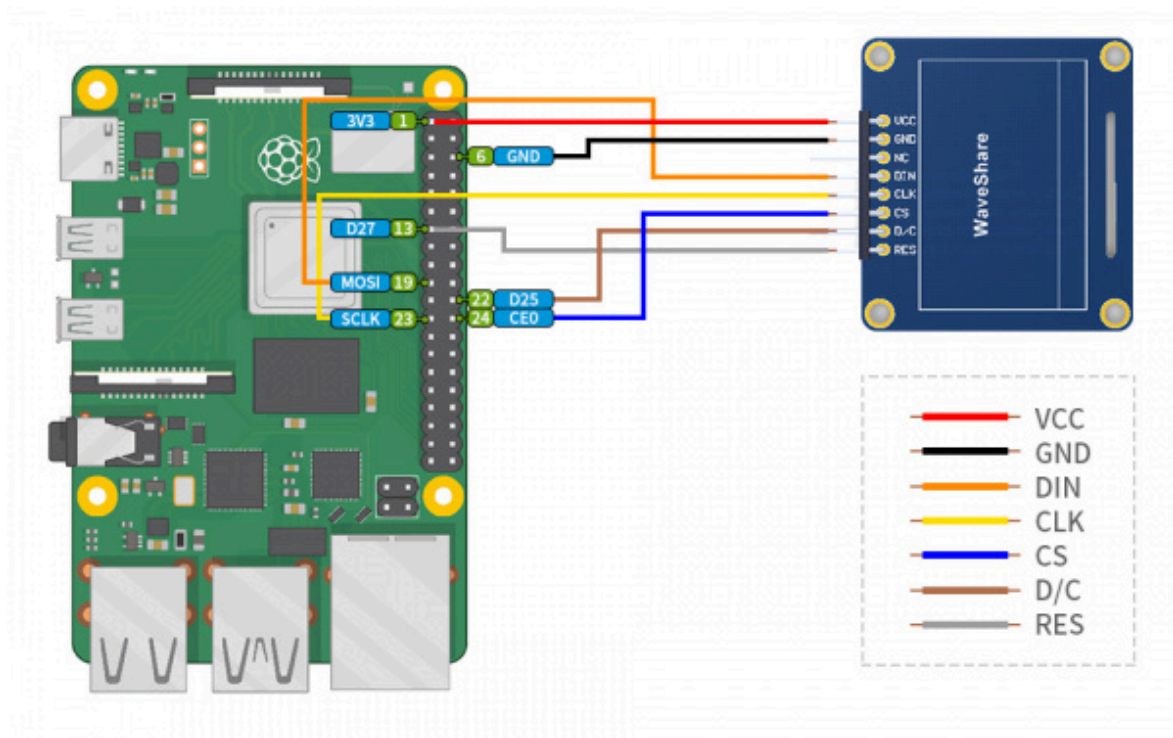
Hardware connection

Please connect the LCD to your Raspberry Pi with the 7Pin cable according to the table below.

Connect to Raspberry Pi

OLED	Raspberry Pi	
	BCM2835	Board
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
D/C	D25	GPIO.6
RES	D27	GPIO.2

- Four-wire SPI wiring diagram



(/wiki/File:1.3Four-wire_SPI_wiring02.jpg)

Enable SPI and I2C Interface

- Open the Raspberry Pi terminal and enter the following command to access the configuration interface:

```
sudo raspi-config
```

```
Select Interfacing Options -> SPI -> Yes to enable the SPI interface
```

```
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>

(/wiki/File:RPI_open_spi.png)

And then reboot the Raspberry Pi:

```
sudo reboot
```

Same for I2C, enter the configuration interface and select Interfacing Options -> I2C -> Yes to enable the IIC interface, then reboot.

Install Library

If you use bookworm system, you can only use I2C library, bcm2835 and wiringPi can't be installed and used.

BCM2835

```
#Open the Raspberry Pi terminal and run the following command:
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
#For more, you can refer to the official website: http://www.airspayce.com/mikem/bcm2835/
```

WiringPi

```
#Open the Raspberry Pi terminal and run the following command:
cd
sudo apt-get install wiringpi
#For Raspberry Pi systems after May 2019 (earlier than that do not need to execute), an u
pgrade may be required:
wget https://project-downloads.drogon.net/wiringpi-latest.deb
sudo dpkg -i wiringpi-latest.deb
gpio -v
#Run gpio -v and version 2.52 will appear, if it does not appear, it means there is an in
stallation error.

#The Bullseye branch system uses the following command:
git clone https://github.com/WiringPi/WiringPi
cd WiringPi
./build
gpio -v
#Run gpio -v and version 2.70 will appear, if it does not appear, it means there is an in
stallation error.
```

I2C

```
wget https://github.com/joan2937/lg/archive/master.zip
unzip master.zip
```

```
cd lg-master
sudo make install
##For more details, you can refer to https://github.com/gpiozero/lg
```

Python

```
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 spidev
sudo apt-get install python3-smbus
```

Download the Test Demo

Open the Raspberry Pi terminal and run:

```
sudo apt-get install p7zip-full
sudo wget https://files.waveshare.com/upload/2/2c/OLED_Module_Code.7z
7z x OLED_Module_Code.7z
cd OLED_Module_Code/RaspberryPi
```

Run the Test Demo

The following commands should be executed in the RaspberryPi directory, otherwise, the directory will not be indexed.

C

- Recompile, which may take a few seconds.

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

Test demos for all screens can be called directly by entering the corresponding size:

```
sudo ./main #Screen size
```

Depending on the OLED, one of the following commands should be entered.

#0.49inch OLED Module

sudo ./main 0.49

#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

#0.96inch OLED Module (C)/(D)/(E)

sudo ./main 0.96

#0.96inch RGB OLED Module

sudo ./main 0.96rgb

#1.27inch RGB OLED Module

sudo ./main 1.27rgb

#1.3inch OLED (A)/(B)

sudo ./main 1.3

#1.3inch OLED Module (C)

sudo ./main 1.3c

#1.32inch OLED Module

sudo ./main 1.32

#1.5inch OLED Module

sudo ./main 1.5

#1.5inch OLED Module (B)

sudo ./main 1.5b

#1.5inch RGB OLED Module

sudo ./main 1.5rgb

#1.51inch Transparent OLED

sudo ./main 1.51

#1.54inch OLED Module

sudo ./main 1.54

```
#2.42inch OLED Module  
sudo ./main 2.42
```

Python

- Enter the Python demo directory:

```
cd python/example
```

- **Run the demo corresponding to the OLED model**, the demo supports python2/3:

If you have purchased a 1.3inch OLED Module (C), please enter:

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

If you have purchased a 1.5inch RGB OLED Module, please enter:

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

- Model instruction correspondence table:

```
#0.49inch OLED Module
sudo python OLED_0in49_test.py
-----
#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
-----
#0.96inch OLED Module (C)/(D)/(E)
sudo python OLED_0in96_test.py
-----
#0.96inch RGB OLED Module
sudo python OLED_0in96_rgb_test.py
-----
#1.27inch RGB OLED Module
sudo python OLED_1in27_rgb_test.py
-----
#1.3inch OLED (A)/(B)
sudo python OLED_1in3_test.py
-----
#1.3inch OLED Module (C)
sudo python OLED_1in3_c_test.py
-----
#1.32inch OLED Module
sudo python OLED_1in32_test.py
-----
#1.5inch OLED Module
sudo python OLED_1in5_test.py
-----
#1.5inch OLED Module (B)
sudo python OLED_1in5_b_test.py
-----
#1.5inch RGB OLED Module
sudo python OLED_1in5_rgb_test.py
-----
#1.51inch Transparent OLED
sudo python OLED_1in51_test.py
-----
#1.54inch OLED Module
sudo python OLED_1in54_test.py
-----
```

```
#2.42inch OLED Module  
sudo python OLED_2in42_test.py
```

- Please make sure that the SPI is not occupied by other devices, you can check in the middle of /boot/config.txt.

Description of C codes (API)

Hardware Interface

1. There are three ways for C to drive: BCM2835 library, WiringPi library and Dev library.
2. We use Dev libraries by default. If you need to change to BCM2835 or WiringPi libraries, please open RaspberryPi\c\Makefile and modify lines 13-15 as follows:

```
13  #USELIB = USE_BCM2835_LIB  
14  #USELIB = USE_WIRINGPI_LIB  
15  USELIB = USE_DEV_LIB  
16  DEBUG = -D $(USELIB)  
17  ifeq ($(USELIB), USE_BCM2835_LIB)  
18      LIB = -lbcm2835 -lm  
19  else ifeq ($(USELIB), USE_WIRINGPI_LIB)  
20      LIB = -lwiringPi -lm  
21  else ifeq ($(USELIB), USE_DEV_LIB)  
22      LIB = -lpthread -lm  
23  endif
```

(/wiki/File:RPI_open_spi1.png)

We have carried out the underlying encapsulation because the hardware platform is different, the internal implementation is different. If you need to know the internal implementation, you can go to the corresponding directory.

You can see many definitions in DEV_Config.c(h), in the directory: RaspberryPi\c\lib\Config.

- Interface selection:

```
#define USE_SPI_4W 1  
#define USE_IIC 0  
Note: Modify here directly to switch SPI/I2C
```

- Data type:

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

- Module initialization and exit processing.

```
void DEV_Module_Init(void);
void DEV_Module_Exit(void);
```

Note:

Here are some GPIO processing before and after using the LCD screen.

- Write GPIO:

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

Parameter:

UWORD Pin: GPIO Pin number

UBYTE Value: level to be output, 0 or 1

- Read GPIO:

```
UBYTE DEV_Digital_Read(UWORD Pin)
```

Parameter:

UWORD Pin: GPIO Pin number

Return value:

level of GPIO, 0 or 1

- GPIO mode setting:

```
void DEV_GPIO_Mode(UWORD Pin, UWORD Mode)
```

Parameters:

UWORD Pin: GPIO Pin number

UWORD Mode: Mode, 0: input, 1: output

GUI Functions

If you need to draw pictures, display Chinese and English characters, display pictures, etc., we provide some basic functions here about some graphics processing, in the directory 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/File:C-GUI.png)

The fonts that GUI depends on can be found in RaspberryPi\c\lib\Fonts directory.

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/File:RPI_open_spi3.png)

- Create Image Properties: Create a new image property, this property includes the image buffer name, width, height, flip angle and color.

```
void Paint_NewImage(UBYTE *image, UWORD Width, UWORD Height, UWORD Rotate, UWORD Color)
```

Parameters:

image: the name of the image buffer, which is actually a pointer to the first address of the image buffer;
Width: the width of the image buffer;
Height: the height of the image buffer;
Rotate: the rotation angle of the image;
Color: the initial color of the image;

- Select image buffer: The purpose of the selection is that you can create multiple image buffers, there can be multiple image buffers, and you can select each image you create.

```
void Paint_SelectImage(UBYTE *image)
```

Parameters:

image: the name of the image buffer, which is actually a pointer to the first address of the image buffer;

- Image rotation: Set the rotation angle of the selected image, preferably after Paint_SelectImage(), you can choose to rotate 0, 90, 180, 270 degrees.

```
void Paint_SetRotate(UWORD Rotate)
```

Parameters:

Rotate: ROTATE_0, ROTATE_90, ROTATE_180 and ROTATE_270 correspond to 0, 90, 180 and 270 degrees.

- Sets the size of the pixels:

```
void Paint_SetScale(UBYTE scale)
```

Parameters:

scale: the size of pixels, 2: Each pixel occupies one bit; 4: Each pixel occupies two bits.

- Image mirror flip: Set the mirror flip of the selected image. You can choose no mirror, horizontal mirror, vertical mirror or image center mirror.

```
void Paint_SetMirroring(UBYTE mirror)
```

Parameters:

Mirror: indicates the image mirroring mode. MIRROR_NONE, MIRROR_HORIZONTAL, MIRROR_VERTICAL, MIRROR_ORIGIN correspond to no mirror, horizontal mirror, vertical mirror and image center mirror respectively.

- Set points of the display position and color in the buffer: Here is the core GUI function, processing points display position and color in the buffer.

```
void Paint_SetPixel(UWORD Xpoint, UWORD Ypoint, UWORD Color)
```

Parameters:

Xpoint: the X position of a point in the image buffer

Ypoint: the Y position of a point in the image buffer

Color: the color of the dot

- Image buffer fill color: Fills the image buffer with a color, usually used to flash the screen into blank.

```
void Paint_Clear(UWORD Color)
```

Parameters:

Color: fill color

- The fill color of a certain window in the image buffer: the image buffer part of the window filled with a certain color, usually used to fresh the screen into blank, often used for time display, fresh the last second of the screen.

```
void Paint_ClearWindows(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color)
```

Parameters:

Xstart: the x-starting coordinate of the window
Ystart: the y-starting coordinate of the window
Xend: the x-end coordinate of the window
Yend: the y-end coordinate of the window
Color: fill color

- Draw point: In the image buffer, draw a point on (Xpoint, Ypoint), you can choose the color, the size of the point and the style of the point.

```
void Paint_DrawPoint(UWORD Xpoint, UWORD Ypoint, UWORD Color, DOT_PIXEL Dot_Pixel, DOT_STYLE Dot_Style)
```

Parameters:

Xpoint: the X coordinate of the point.
Ypoint: the Y coordinate of the point.
Color: fill color
Dot_Pixel: The size of the dot, the demo provides 8 size points by default.

```
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: the style of a point that expands from the center of the point or from the bottom left corner of the point to the right and up.

```
typedef enum {  
    DOT_FILL_AROUND  = 1,  
    DOT_FILL_RIGHTUP ,  
} DOT_STYLE;
```

- Draw line: In the image buffer, draw a line from (Xstart, Ystart) to (Xend, Yend), you can choose the color, the width and the style of the line.

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color, LINE_STYLE Line_Style, LINE_STYLE Line_Style)
```

Parameters:

Xstart: the x-starting coordinate of the line

Ystart: the y-starting coordinate of the line

Xend: the x-end coordinate of the line

Yend: the y-end coordinate of the line

Color: fill color

Line_width: The width of the line, the demo provides 8 sizes of width by default.

```
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: line style. Select whether the lines are joined in a straight or dashed way.

```
typedef enum {
```

```
    LINE_STYLE_SOLID = 0,
```

```
    LINE_STYLE_DOTTED,
```

```
} LINE_STYLE;
```

- Draw rectangle: In the image buffer, draw a rectangle from (Xstart, Ystart) to (Xend, Yend), you can choose the color, the width of the line and whether to fill the inside of the rectangle.


```
void Paint_DrawRectangle(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color,
DOT_PIXEL Line_width, DRAW_FILL Draw_Fill)
```

Parameters:

Xstart: the starting X coordinate of the rectangle

Ystart: the starting Y coordinate of the rectangle

Xend: the x-end coordinate of the rectangle

Yend: the y-end coordinate of the rectangle

Color: fill color

Line_width: The width of the four sides of a rectangle. And the demo provides 8 sizes of width by default.

```
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: Fill, whether to fill the inside of the rectangle

```
typedef enum {
    DRAW_FILL_EMPTY = 0,
    DRAW_FILL_FULL,
} DRAW_FILL;
```

- Draw circle: In the image buffer, draw a circle with Radius as the radius and (X_Center, Y_Center) as the center. You can choose the color, the width of the line and whether to fill the inside of the circle.

```
void Paint_DrawCircle(UWORD X_Center, UWORD Y_Center, UWORD Radius, UWORD Color, DOT_PIXEL Line_width, DRAW_FILL Draw_Fill)
```

Parameters:

X_Center: the x-coordinate of the center of the circle

Y_Center: the y-coordinate of the center of the circle

Radius: the radius of the circle

Color: fill color

Line_width: The width of the arc, with a default of 8 widths

```
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: fill, whether to fill the inside of the circle

```
typedef enum {
    DRAW_FILL_EMPTY = 0,
    DRAW_FILL_FULL,
} DRAW_FILL;
```

- Write Ascii character: In the image buffer, use (Xstart, Ystart) as the left vertex, and write an Ascii character, you can select Ascii visual character library, font foreground color and font background color.

```
void Paint_DrawChar(UWORD Xstart, UWORD Ystart, const char Ascii_Char, sFONT* Font, UWORD Color_Foreground, UWORD Color_Background)
```

Parameters:

Xstart: the x-coordinate of the left vertex of a character

Ystart: the y-coordinate of the left vertex of a character

Ascii_Char: indicates the Ascii character

Font: Ascii visual character library, in the Fonts folder the demo provides the following Fonts:

Font8: 5*8 font

Font12: 7*12 font

Font16: 11*16 font

Font20: 14*20 font

Font24: 17*24 font

Color_Foreground: font color

Color_Background: background color

- Write English string: In the image buffer, use (Xstart Ystart) as the left vertex, and write a string of English characters, you can choose Ascii visual character library, font foreground color and font background color.

```
void Paint_DrawString_EN(UWORD Xstart, UWORD Ystart, const char * pString, sFONT* Font, UWORD Color_Foreground, UWORD Color_Background)
```

Parameters:

Xstart: the X coordinate of the left vertex of a character

Ystart: the Y coordinate of the left vertex of a character

PString: string, string is a pointer

Font: Ascii visual character library, in the Fonts folder the demo provides the following Fonts:

Font8: 5*8 font

Font12: 7*12 font

Font16: 11*16 font

Font20: 14*20 font

Font24: 17*24 font

Color_Foreground: font color

Color_Background: background color

- Write Chinese string: In the image buffer, use (Xstart, Ystart) as the left vertex, and write a string of Chinese characters, you can choose GB2312 encoding font, font foreground color and font background color.

```
void Paint_DrawString_CN(UWORD Xstart, UWORD Ystart, const char * pString, cFONT* font, UWORD Color_Foreground, UWORD Color_Background)
```

Parameters:

Xstart: the X coordinate of the left vertex of a character

Ystart: the Y coordinate of the left vertex of a character

PString: string, string is a pointer

Font: GB2312 encoding character Font library, in the Fonts folder the demo provides the following Fonts:

Font12CN: ASCII font 11*21, Chinese font 16*21

Font24CN: ASCII font 24*41, Chinese font 32*41

Color_Foreground: font color

Color_Background: background color

- Write numbers: In the image buffer, use (Xstart, Ystart) as the left vertex, and write a string of numbers, you can choose Ascii visual character library, font foreground color and font background color.

```
void Paint_DrawNum(UWORD Xpoint, UWORD Ypoint, double Nummber, sFONT* Font, UWORD Digit,
UWORD Color_Foreground, UWORD Color_Background)
```

Parameters:

Xpoint: the X coordinate of the left vertex of the character

Ypoint: the Y coordinate of the left vertex of the character

Nummber: indicates the number displayed, which can be a decimal

Digit: it's a decimal number

Font: Ascii visual character library, in the Fonts folder the demo provides the following Fonts:

Font8: 5*8 font

Font12: 7*12 font

Font16: 11*16 font

Font20: 14*20 font

Font24: 17*24 font

Color_Foreground: font color

Color_Background: background color

- Display time: In the image buffer, use (Xstart Ystart) as the left vertex, display time, you can choose Ascii visual character font, font foreground color and font background color.

```
void Paint_DrawTime(UWORD Xstart, UWORD Ystart, PAINT_TIME *pTime, sFONT* Font, UWORD Color_Background, UWORD Color_Foreground)
```

Parameters:

Xstart: the X coordinate of the left vertex of the character

Ystart: the Y coordinate of the left vertex of the character

PTime: display time, A time structure is defined here, as long as the hours, minutes and seconds are passed to the parameters;

Font: Ascii visual character library, in the Fonts folder the demo provides the following Fonts:

Font8: 5*8 font

Font12: 7*12 font

Font16: 11*16 font

Font20: 14*20 font

Font24: 17*24 font

Color_Foreground: font color

Color_Background: background color

Python (for Raspberry Pi)

It is compatible with python and python3.

The calls of python are less complex compared to the C demo.

Config.py

- Select interface.

```
Device_SPI = 1
```

```
Device_I2C = 0
```

Note: Modify here to switch SPI/I2C.

- Module initialization and exit processing.

```
def module_init()
```

```
def module_exit()
```

Note:

1. Here are some GPIO processing before and after using the LCD screen.

2. The module_init() function is automatically called in the init () initializer on the LCD, but the module_exit() function needs to be called itself.

- SPI writes data.

```
def spi_writebyte(data)
```

- IIC writes data.

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

Main.py

The main function, if your Python version is Python2, re-executed in Linux command mode as follows.

```
sudo python main.py
```

If your Python version is Python3, re-executed in Linux command mode as follows.

```
sudo python3 main.py
```

GUI Functions

Python has an image library PIL official library link (<https://pillow.readthedocs.io/en/stable/>), it does not need to write code from the logical layer like C and can directly call the image library for image processing. The following will take a 1.54-inch OLED as an example, we provide a brief description of the demo.

- It needs to use the image library and install the library.

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

And then import the library.

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

Among them, Image is the basic library, ImageDraw is the drawing function library, and ImageFont is the font library.

- Define an image buffer to facilitate drawing, writing and other functions on the image.

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

The first parameter defines the color depth of the image, which is defined as "1" to indicate the bitmap of one-bit depth. The second parameter is a tuple that defines the width and height of the image. The third parameter defines the default color of the buffer, which is defined as "WHITE".

- Create a drawing object based on image1, on which all drawing operations will be performed.

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

- Draw a line:

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

The first parameter is a four-element tuple, draw a line starting at (0, 0) and ending at (127,0). "fill =0" means the color of the line is white.

- Draw a rectangle:

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

The first parameter is a tuple of four elements. (20,10) is the coordinate value in the upper left corner of the rectangle, and (70,60) is the coordinate value in the lower right corner of the rectangle. fill = "WHITE" means white inside, and outline="BLACK" means the color of the outline is black.

- Draw a circle:

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

Draw an inscribed circle in the square, the first parameter is a tuple of 4 elements, with (150, 15) as the upper left corner vertex of the square, (190, 55) as the lower right corner vertex of the square, specifying the level median line of the rectangular frame is the angle of 0 degrees, and the angle becomes larger clockwise; the second parameter indicates the starting angle; the third parameter indicates the ending angle, and fill = 0 indicates that the color of the line is white.

If you are not using the square, what you draw will be an ellipse, which is a drawing of an arc.

Besides the arc function, you can also use the ellipse function for drawing a solid circle.

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

The essence is the drawing of the ellipse. The first parameter specifies the enclosing rectangle.

"fill = 0" means that the inner fill color is white. If its enclosing rectangle is a square, the ellipse is a circle.

- Write a character:

The ImageFont module needs to be imported and instantiated:

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

You can use the fonts of Windows or other fonts which is in ttc format.

Note: Each character library contains different characters; If some characters cannot be displayed, it is recommended that you can refer to the encoding set used.

To draw English characters, you can directly use the fonts; for Chinese characters, you need to add a symbol "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)
```

The first parameter is a two-element tuple with (5, 68) as the left vertex, and use font1, fill is the font color, fill = 0 means that the font color is white, and the second sentence shows '微雪电子', and its font color is white.

- Read local pictures:

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

The parameter is the image path.

- Other functions:

Python's image library is very powerful, if you need to achieve other features, you can go to the official website to learn: <https://pillow.readthedocs.io/en/stable/>
(<https://pillow.readthedocs.io/en/stable/>).

User Guides of STM 32

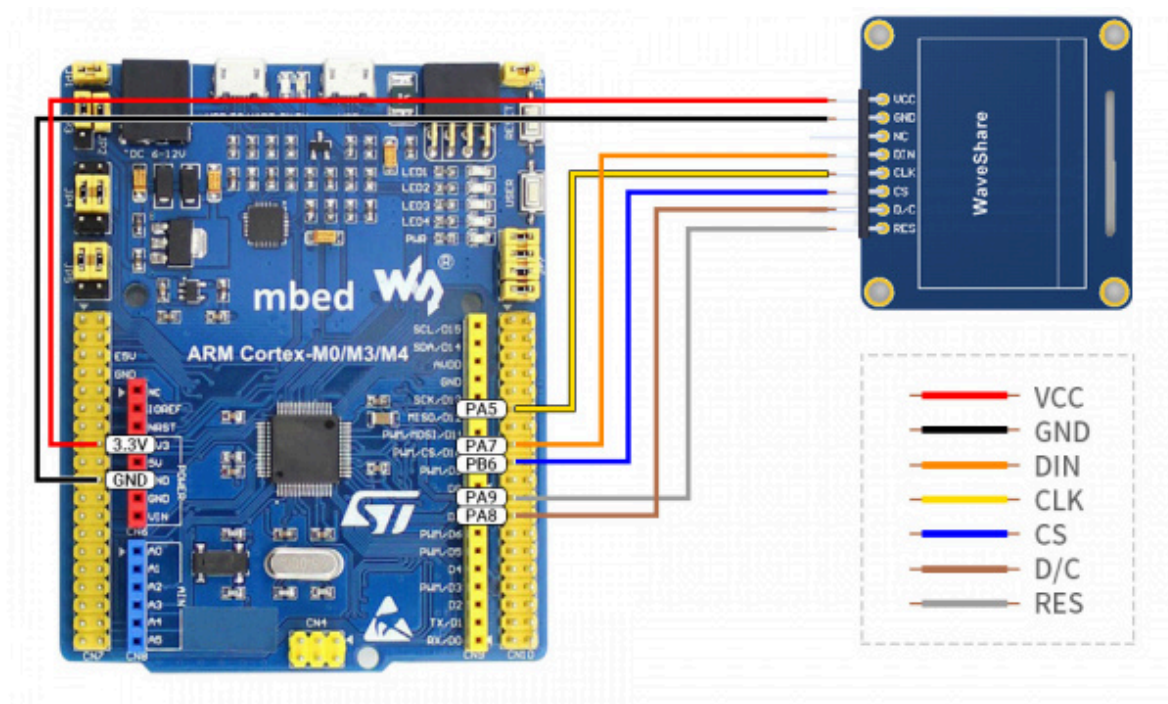
Hardware Connection

The examples are based on STM32F103RBT6 as well as the connection table. If you want to use other MCU, you need to port the project and change the connection according to the actual hardware.

Connect to STM32F103RBT6

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
D/C	PA8
RES	PA9

- Four-wire SPI wiring diagram



(/wiki/File:0.91-OLED-STM322.png)

Run the demo

- Download the program, find the STM32 program file directory, use STM32CubeIDE to open the .cproject in the \STM32\STM32F103RBT6\ directory.
- Then modify the corresponding function comment in the main.c according to the LCD model you are using, and then recompile and download to your board.

```
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_lin3_test(); // IIC must USE_IIC_SOFT
104
105 // OLED_lin3_c_test();
106
107 // OLED_lin5_test();
108
109 // OLED_lin5_rgb_test(); // Only SPI !!!
```

(/wiki/File:OLED_STM32_code0.png)

- **If** you are using a 1.3-inch OLED Module (C), you need to comment out line 105. (Note: there cannot be multiple sentences without comment at the same time; the line number may be changed, please modify it according to the actual situation.)
- The command of each OLED model can be found in the following table:

Model	Demo Function
0.49inch OLED Module	OLED_0in49_test();
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();
0.96inch OLED Module (C)/(D)/(E)	OLED_0in96_test();
0.96inch RGB OLED Module	OLED_0in96_rgb_test();
1.27inch RGB OLED Module	OLED_1in27_rgb_test();
1.3inch OLED (A)/(B)	OLED_1in3_test();
1.3inch OLED Module (C)	OLED_1in3_c_test();
1.32inch OLED Module	OLED_1in32_test();
1.5inch OLED Module	OLED_1in5_test();
1.5inch OLED Module (B)	OLED_1in5_b_test();
1.5inch RGB OLED Module	OLED_1in5_rgb_test();
1.54inch OLED Module	OLED_1in54_test();
2.42inch OLED Module	OLED_2in42_test();

Software description

- The demo is developed based on the HAL library, and the software used is STM32CubeIDE. Download the demo, find the STM32 demo file directory, and open the .cproject in the STM32\STM32F103RBT6\ directory to see the demo.

Core	2024/3/6 16:32	文件夹	
Drivers	2024/3/6 16:32	文件夹	
IDE .cproject	2024/3/6 19:31	CPROJECT 文件	26 KB
.mxproject	2023/8/22 17:21	MXPROJECT 文件	9 KB
IDE .project	2023/8/21 9:39	PROJECT 文件	2 KB
oled_demo.pdf	2020/8/28 10:59	看图王 PDF 文件	181 KB
readme_CN.txt	2020/8/28 11:01	文本文档	3 KB
readme_EN.txt	2020/8/28 11:01	文本文档	3 KB

(/wiki/File:OLED_STM32_code71.jpg)

- In addition, you can see the file directory of the project in the STM32\STM32-F103RBT6\User\ directory. The five folders are the underlying driver, sample demo, font, GUI and OLED driver.

Config	2024/3/6 16:32	文件夹	
Example	2024/3/6 16:42	文件夹	
Fonts	2024/3/6 16:32	文件夹	
GUI	2024/3/6 16:32	文件夹	
OLED	2024/3/6 16:34	文件夹	
Readme_CN.txt	2020/8/28 12:04	文本文档	5 KB
Readme_EN.txt	2020/8/28 12:04	文本文档	6 KB

(/wiki/File:OLED_STM32_code72.jpg)

Demo description

Hardware interface

We package the bottom for different hardware platforms. You can check the DEV_Config.c(h) file for more description.

■ Interface selection

```
#define USE_SPI_4W      1
#define USE_IIC         0
#define USE_IIC_SOFT    0
Note: Modify here directly to switch SPI/I2C
```

■ Data type

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

■ Module initialization and exit processing:

```
UBYTE  System_Init(void);
void    System_Exit(void);
Note:
1. Here are some GPIO processing before and after using the LCD screen.
2. After the System_Exit(void) function is used, the OLED display will be turned off;
```

■ Write and read GPIO:

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

- SPI writes data:

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

- IIC writes data:

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

Application Function

For the screen, if you need to draw pictures, display Chinese and English characters, display pictures, etc., you can use the upper application to do, and we provide some basic functions here about some graphics processing, you can check in the directory

STM32\STM32F103RB\User\GUI\GUI_Paint.c(h).

OLED_Module_Code > STM32 > STM32-F103RBT6 > User > GUI

名称	修改日期	类型	大小
GUI_Paint.c	2020/8/18 18:30	C 文件	37 KB
GUI_Paint.h	2020/8/18 18:37	H 文件	9 KB

(/wiki/File:OLED_STM32_code3.png)

The character font GUI dependent is in the directory STM32\STM32F103RB\User\Fonts.

OLED_Module_Code > STM32 > STM32-F103RBT6 > User > Fonts

名称	修改日期	类型	大小
font8.c	2018/7/4 17:24	C 文件	18 KB
font12.c	2018/7/4 17:24	C 文件	27 KB
font12CN.c	2018/3/6 15:52	C 文件	6 KB
font16.c	2018/7/4 17:24	C 文件	49 KB
font20.c	2018/7/4 17:24	C 文件	65 KB
font24.c	2018/7/4 17:24	C 文件	97 KB
font24CN.c	2018/3/6 16:02	C 文件	28 KB
fonts.h	2018/10/29 14:04	H 文件	4 KB

(/wiki/File:OLED_STM32_code4.png)

- Create Image Properties: Create a new image property, this property includes the image buffer name, width, height, flip angle and color.

```
void Paint_NewImage(UWORD Width, UWORD Height, UWORD Rotate, UWORD Color)
```

Parameters:

Width: the width of the image buffer;
 Height: the height of the image buffer;
 Rotate: the rotation angle of the image;
 Color: the initial color of the image;

- Set the clear screen function, usually directly call the OLED clear function:

```
void Paint_SetClearFuntion(void (*Clear)(UWORD));
```

parameter:

Clear: Pointer to the clear screen function, used to quickly clear the screen to a certain color;

- Set the drawing pixel function:

```
void Paint_SetDisplayFuntion(void (*Display)(UWORD,UWORD,UWORD));
```

parameter:

Display: Pointer to the drawing pixel function, which is used to write data to the specified location in the internal RAM of the OLED;

- Select image buffer: the purpose of the selection is that you can create multiple image attributes, there can be multiple images buffer, you can select each image you create.

```
void Paint_SelectImage(UBYTE *image)
```

Parameters:

Image: the name of the image buffer, which is actually a pointer to the first address of the image buffer;

- Image rotation: Set the selected image rotation angle, preferably after Paint_SelectImage(), you can choose to rotate 0, 90, 180 or 270 degrees.

```
void Paint_SetRotate(UWORD Rotate)
```

Parameters:

Rotate: ROTATE_0, ROTATE_90, ROTATE_180 and ROTATE_270 correspond to 0, 90, 180 and 270 degrees respectively;

- Image mirror flip: Set the mirror flip of the selected image. You can choose no mirror, horizontal mirror, vertical mirror or image center mirror.

```
void Paint_SetMirroring(UBYTE mirror)
```

Parameters:

Mirror: indicates the image mirroring mode. MIRROR_NONE, MIRROR_HORIZONTAL, MIRROR_VERTICAL, MIRROR_ORIGIN correspond to no mirror, horizontal mirror, vertical mirror and about image center mirror respectively.

- Set points of display position and color in the buffer: here is the core GUI function, processing points display position and color in the buffer.

```
void Paint_SetPixel(UWORD Xpoint, UWORD Ypoint, UWORD Color)
```

Parameters:

Xpoint: the X position of a point in the image buffer
Ypoint: the Y position of a point in the image buffer
Color: the color of the point

- Image buffer fill color: Fills the image buffer with a color, usually used to flash the screen into blank.

```
void Paint_Clear(UWORD Color)
```

Parameters:

Color: fill color

- The fill color of a certain window in the image buffer: the image buffer part of the window filled with a certain color, usually used to fresh the screen into blank, often used for time display, fresh the last second of the screen.

```
void Paint_ClearWindows(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color)
```

Parameters:

Xstart: the X starting point coordinate of the window
Ystart: the Y starting point coordinate of the window
Xend: the x-end coordinate of the window
Yend: the y-end coordinate of the window
Color: fill color

- Draw point: In the image buffer, draw a point on (Xpoint, Ypoint), you can choose the color, the size of the point and the style of the point.

```
void Paint_DrawPoint(UWORD Xpoint, UWORD Ypoint, UWORD Color, DOT_PIXEL Dot_Pixel, DOT_STYLE Dot_Style)
```

Parameters:

Xpoint: indicates the X coordinate of a point

Ypoint: indicates the Y coordinate of a point

Color: fill color

Dot_Pixel: The size of the dot, providing a default of eight size points

```
typedef enum {  
    DOT_PIXEL_1X1 ,           // 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: the style of a point, which determines expands from the center of the point or from the bottom left corner of the point to the right and up

```
typedef enum {  
    DOT_FILL_AROUND = 1,  
    DOT_FILL_RIGHTUP,  
} DOT_STYLE;
```

- Draw line: In the image buffer, draw a line from (Xstart, Ystart) to (Xend, Yend), you can choose the color, line width and line style.


```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color, LINE_STYLE Line_Style , LINE_STYLE Line_Style)
```

Parameters:

Xstart: the x-starting coordinate of the line

Ystart: the y-starting coordinate of the line

Xend: the x-end coordinate of the line

Yend: the y-end coordinate of the line

Color: fill color

Line_width: The width of the line, the demo provides 8 sizes of width by default.

```
typedef enum {
    DOT_PIXEL_1X1 ,          // 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: line style. Select whether the lines are joined in a straight or dashed way

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

- Draw rectangle: In the image buffer, draw a rectangle from (Xstart, Ystart) to (Xend, Yend), you can choose the color, the width of the line and whether to fill the inside of the rectangle.

```
void Paint_DrawRectangle(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color,
DOT_PIXEL Line_width, DRAW_FILL Draw_Fill)
```

Parameters:

Xstart: the starting X coordinate of the rectangle

Ystart: the starting Y coordinate of the rectangle

Xend: the x-end coordinate of the rectangle

Yend: the y-end coordinate of the rectangle

Color: fill color

Line_width: The width of the four sides of a rectangle. And the demo provides 8 sizes of width by default.

```
DOT_PIXEL_1X1 ,           // 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: Fill, whether to fill the inside of the rectangle

```
typedef enum {
    DRAW_FILL_EMPTY = 0,
    DRAW_FILL_FULL,
} DRAW_FILL;
```

- Draw circle: In the image buffer, draw a circle with Radius as the radius and (X_Center, Y_Center) as the center. You can choose the color, the width of the line and whether to fill the inside of the circle.

```
void Paint_DrawCircle(UWORD X_Center, UWORD Y_Center, UWORD Radius, UWORD Color, DOT_PIXEL Line_width, DRAW_FILL Draw_Fill)
```

Parameters:

X_Center: the x-coordinate of the center of a circle

Y_Center: the y-coordinate of the center of the circle

Radius: the radius of a circle

Color: fill color

Line_width: The width of the arc, with a default of 8 widths

```
typedef enum {
    DOT_PIXEL_1X1 ,          // 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: fill, whether to fill the inside of the circle

```
typedef enum {
    DRAW_FILL_EMPTY = 0,
    DRAW_FILL_FULL,
} DRAW_FILL;
```

- Write Ascii character: In the image buffer, use (Xstart, Ystart) as the left vertex, and write an Ascii character, you can select Ascii visual character library, font foreground color and font background color.

```
void Paint_DrawChar(UWORD Xstart, UWORD Ystart, const char Ascii_Char, sFONT* Font, UWORD Color_Foreground, UWORD Color_Background)
```

Parameters:

Xstart: the x-coordinate of the left vertex of a character

Ystart: the y-coordinate of the left vertex of a character

Ascii_Char: indicates the Ascii character

Font: Ascii visual character library, in the Fonts folder provides the following Font

s:

Font8: 5*8 font

Font12: 7*12 font

Font16: 11*16 font

Font20: 14*20 font

Font24: 17*24 font

Color_Foreground: font color

Color_Background: background color

- Write English string: In the image buffer, use (Xstart, Ystart) as the left vertex, and write a string of English characters, you can choose Ascii visual character library, font foreground color and font background color.

```
void Paint_DrawString_EN(UWORD Xstart, UWORD Ystart, const char * pString, sFONT* Font, UWORD Color_Foreground, UWORD Color_Background)
```

Parameters:

Xstart: the X-coordinate of the left vertex of a character

Ystart: the Y-coordinate of the left vertex of a character

PString: string, string is a pointer

Font: Ascii visual character library, in the Fonts folder provides the following Fonts:

Font8: 5*8 font

Font12: 7*12 font

Font16: 11*16 font

Font20: 14*20 font

Font24: 17*24 font

Color_Foreground: font color

Color_Background: background color

- Write Chinese string: In the image buffer, use (Xstart Ystart) as the left vertex, and write a string of Chinese characters, you can choose GB2312 encoding character font, font foreground color and font background color.

```
void Paint_DrawString_CN(UWORD Xstart, UWORD Ystart, const char * pString, cFONT* font, UWORD Color_Foreground, UWORD Color_Background)
```

Parameters:

Xstart: the X-coordinate of the left vertex of a character

Ystart: the Y-coordinate of the left vertex of a character

PString: string, string is a pointer

Font: GB2312 encoding character Font library, in the Fonts folder provides the following Fonts:

Font12CN: ASCII font 11*21, Chinese font 16*21

Font24CN: ASCII font 24*41, Chinese font 32*41

Color_Foreground: font color

Color_Background: background color

- Write numbers: In the image buffer, use (Xstart Ystart) as the left vertex, and write a string of numbers, you can choose Ascii visual character library, font foreground color and font background color.

```
void Paint_DrawNum(UWORD Xpoint, UWORD Ypoint, double Nummber, sFONT* Font, UWORD Digit,
UWORD Color_Foreground,    UWORD Color_Background)
```

Parameters:

Xstart: the X-coordinate of the left vertex of a character

Ystart: the Y-coordinate of the left vertex of a character

Nummber: indicates the number displayed, which can be a decimal

Digit: It's a decimal number

Font: Ascii visual character library, in the Fonts folder provides the following Font

s:

Font8: 5*8 font

Font12: 7*12 font

Font16: 11*16 font

Font20: 14*20 font

Font24: 17*24 font

Color_Foreground: font color

Color_Background: background color

- Display time: in the image buffer, use (Xstart Ystart) as the left vertex, display time, you can choose Ascii visual character font, font foreground color and font background color.

```
void Paint_DrawTime(UWORD Xstart, UWORD Ystart, PAINT_TIME *pTime, sFONT* Font, UWORD Col
or_Background,    UWORD Color_Foreground)
```

Parameters:

Xstart: the X-coordinate of the left vertex of a character

Ystart: the Y-coordinate of the left vertex of a character

PTime: display time, here defined a time structure, just pass the hour, minute and second bits of data to the parameter;

Font: Ascii visual character library, in the Fonts folder provides the following Font

s:

Font8: 5*8 font

Font12: 7*12 font

Font16: 11*16 font

Font20: 14*20 font

Font24: 17*24 font

Color_Foreground: font color

Color_Background: background color

User Guides of Arduino

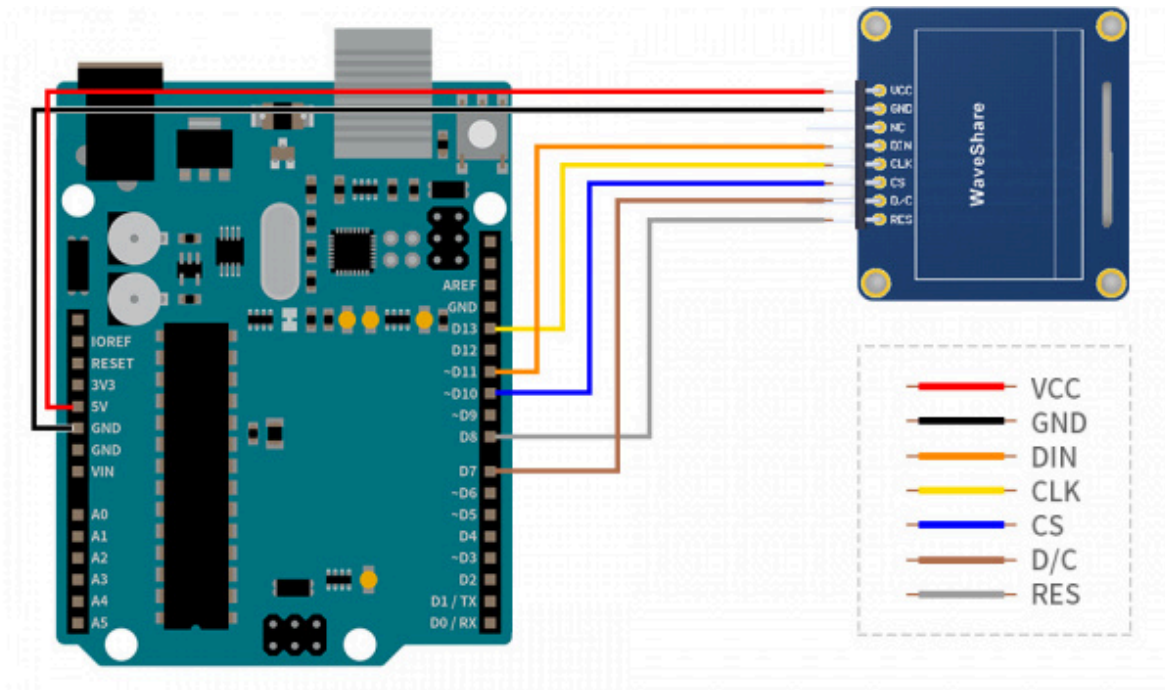
Hardware Connection

Arduino UNO connection

OLED	UNO
------	-----

VCC	3.3V/5V
GND	GND
DIN	SPI:D11 / I2C:SDA
CLK	SPI:D13 / I2C:SCL
CS	D10
DC	D7
RST	D8

- Four-wire SPI wiring diagram



(/wiki/File:0.91-OLED-Aduino2.png)

How to Install Arduino IDE








Arduino IDE Install Steps (https://www.waveshare.com/wiki/Template:Arduino_IDE_Installation_Steps)

Run the demo

- Download the demo and find the Arduino demo file directory.
- Use Arduino IDE to open the .ino file in the project folder of the corresponding model, recompile and download the demo to your board.
- For example, if you are using the 1.3inch OLED Module (C), open OLED_1in3_c.ino under the \Arduino\OLED_1in3_c directory.

Software description

- Download the demo on the Resources, open the Arduino demo file directory, you can see the Arduino program of different models of OLED.

名称	修改日期	类型
 OLED_0in91	2020/8/20 14:17	文件夹
 OLED_0in95_rgb	2020/8/20 10:49	文件夹
 OLED_0in96	2020/8/21 14:59	文件夹
 OLED_1in3	2020/8/21 15:05	文件夹
 OLED_1in3_c	2020/8/21 14:48	文件夹
 OLED_1in5	2020/8/21 14:50	文件夹
 OLED_1in5_rgb	2020/8/20 10:52	文件夹

(/wiki/File:OLED_Arduino_code1.png)

- Choose the folder according to the LCD model you are using, and open the xxx.ino file. Take the 1.3inch OLED Module (C) as an example: open OLED_1in3_c, then double-click OLED_1in3_c.ino to open the Arduino project.

名称	修改日期	类型	大小
 Debug.h	2020/8/19 19:44	H 文件	1 KB
 DEV_Config.cpp	2020/7/14 17:08	CPP 文件	3 KB
 DEV_Config.h	2020/8/21 14:48	H 文件	2 KB
 font8.cpp	2020/6/16 9:37	CPP 文件	18 KB
 font12.cpp	2020/6/16 9:37	CPP 文件	28 KB
 font12CN.cpp	2020/6/16 9:37	CPP 文件	6 KB
 font16.cpp	2020/6/16 9:37	CPP 文件	49 KB
 font20.cpp	2020/6/16 9:37	CPP 文件	65 KB
 font24.cpp	2020/6/16 9:37	CPP 文件	100 KB
 font24CN.cpp	2020/6/16 9:37	CPP 文件	28 KB
 fonts.h	2020/6/16 9:37	H 文件	4 KB
 GUI_Paint.cpp	2020/8/21 14:44	CPP 文件	33 KB
 GUI_Paint.h	2020/6/16 10:04	H 文件	8 KB
 ImageData.c	2020/8/19 15:49	C 文件	7 KB
 ImageData.h	2020/8/19 15:49	H 文件	2 KB
 OLED_1in3_c.ino	2020/8/21 14:47	Arduino file	3 KB
 OLED_Driver.cpp	2020/8/20 10:12	CPP 文件	8 KB
 OLED_Driver.h	2020/8/20 10:12	H 文件	3 KB
 Readme.txt	2020/7/1 15:12	文本文档	1 KB

(/wiki/File:OLED_Arduino_code2.png)

- The demo folder of each LCD model can find in the following table:

LCD Model	Demo folder
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

Program description

Underlying hardware interface

Because the hardware platform and the internal implementation are different. If you need to know the internal implementation, you can see many definitions in the directory DEV_Config.c(h).

- Interface selection

```
#define USE_SPI_4W          1
#define USE_IIC             0
#define USE_IIC_SOFT       0
Note:
    Switch SPI/I2C directly modified here
```

- Data type

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

- Module initialization and exit processing

```
UBYTE  System_Init(void);
void    System_Exit(void);
Note:
1.here is some GPIO processing before and after using the LCD screen.
2.After the System_Exit(void) function is used, the OLED display will be turned off;
```


- Write and read GPIO

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

- SPI write data

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

- IIC write data

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

The upper application

For the screen, if you need to draw pictures, display Chinese and English characters, display pictures, etc., you can use the upper application to do, and we provide some basic functions here about some graphics processing in the directory

STM32\STM32F103RB\User\GUI\GUI_Paint.c(.h)

OLED_Module_Code > STM32 > STM32-F103RBT6 > User > GUI			
名称	修改日期	类型	大小
GUI_Paint.c	2020/8/18 18:30	C 文件	37 KB
GUI_Paint.h	2020/8/18 18:37	H 文件	9 KB

(/wiki/File:OLED_STM32_code3.png)

The character font which GUI dependent is in the directory

STM32\STM32F103RB\User\Fonts

OLED_Module_Code > STM32 > STM32-F103RBT6 > User > Fonts			
名称	修改日期	类型	大小
font8.c	2018/7/4 17:24	C 文件	18 KB
font12.c	2018/7/4 17:24	C 文件	27 KB
font12CN.c	2018/3/6 15:52	C 文件	6 KB
font16.c	2018/7/4 17:24	C 文件	49 KB
font20.c	2018/7/4 17:24	C 文件	65 KB
font24.c	2018/7/4 17:24	C 文件	97 KB
font24CN.c	2018/3/6 16:02	C 文件	28 KB
fonts.h	2018/10/29 14:04	H 文件	4 KB

(/wiki/File:OLED_STM32_code4.png)

- New Image Properties: Create a new image property, this property includes the image buffer name, width, height, flip Angle, color.

```
void Paint_NewImage(UWORD Width, UWORD Height, UWORD Rotate, UWORD Color)
```

Parameters:

Width: image buffer Width;
Height: the Height of the image buffer;
Rotate: Indicates the rotation Angle of an image
Color: the initial Color of the image;

- Set the clear screen function, usually call the clear function of OLED directly.

```
void Paint_SetClearFuntion(void (*Clear)(UWORD));
```

parameter:

Clear : Pointer to the clear screen function, used to quickly clear the screen to a certain color;

- Set the drawing pixel function.

```
void Paint_SetDisplayFuntion(void (*Display)(UWORD,UWORD,UWORD));
```

parameter:

Display: Pointer to the pixel drawing function, which is used to write data to the specified location in the internal RAM of the OLED;

- Select image buffer:the purpose of the selection is that you can create multiple image attributes, image buffer can exist multiple, you can select each image you create.

```
void Paint_SelectImage(UBYTE *image)
```

Parameters:

Image: the name of the image cache, which is actually a pointer to the first address of the image buffer

- Image Rotation: Set the selected image rotation Angle, preferably after Paint_SelectImage(), you can choose to rotate 0, 90, 180, 270.

```
void Paint_SetRotate(UWORD Rotate)
```

Parameters:

Rotate: ROTATE_0, ROTATE_90, ROTATE_180, and ROTATE_270 correspond to 0, 90, 180, and 270 degrees respectively;

- Image mirror flip: Set the mirror flip of the selected image. You can choose no mirror, horizontal mirror, vertical mirror, or image center mirror.

```
void Paint_SetMirroring(UBYTE mirror)
```

Parameters:

Mirror: indicates the image mirroring mode. MIRROR_NONE, MIRROR_HORIZONTAL, MIRROR_VERTICAL, MIRROR_ORIGIN correspond to no mirror, horizontal mirror, vertical mirror, and about image center mirror respectively.

- Set points of display position and color in the buffer: here is the core GUI function, processing points display position and color in the buffer.

```
void Paint_SetPixel(UWORD Xpoint, UWORD Ypoint, UWORD Color)
```

Parameters:

Xpoint: the X position of a point in the image buffer

Ypoint: Y position of a point in the image buffer

Color: indicates the Color of the dot

- Image buffer fill color: Fills the image buffer with a color, usually used to flash the screen into blank.

```
void Paint_ClearWindows(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color)
```

Parameters:

Xstart: the x-starting coordinate of the window

Ystart: indicates the Y starting point of the window

Xend: the x-end coordinate of the window

Yend: indicates the y-end coordinate of the window

Color: fill Color

- Draw points: In the image buffer, draw points on (Xpoint, Ypoint), you can choose the color, the size of the point, the style of the point.

```
void Paint_DrawPoint(UWORD Xpoint, UWORD Ypoint, UWORD Color, DOT_PIXEL Dot_Pixel, DOT_STYLE Dot_Style)
```

Parameters:

Xpoint: indicates the X coordinate of a point

Ypoint: indicates the Y coordinate of a point

Color: fill Color

Dot_Pixel: The size of the dot, providing a default of eight size points

```
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: the size of a point that expands from the center of the point or from the bottom left corner of the point to the right and up

```
typedef enum {
    DOT_FILL_AROUND  = 1,
    DOT_FILL_RIGHTUP,
} DOT_STYLE;
```

- Line drawing: In the image buffer, line from (Xstart, Ystart) to (Xend, Yend), you can choose the color, line width, line style.

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color, LINE_STYLE Line_Style , LINE_STYLE Line_Style)
```

Parameters:

Xstart: the x-starting coordinate of a line

Ystart: indicates the Y starting point of a line

Xend: x-terminus of a line

Yend: the y-end coordinate of a line

Color: fill Color

Line_width: The width of the line, which provides a default of eight widths

```
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: line style. Select whether the lines are joined in a straight or dashed way

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

- Draw rectangle: In the image buffer, draw a rectangle from (Xstart, Ystart) to (Xend, Yend), you can choose the color, the width of the line, whether to fill the inside of the rectangle.

```
void Paint_DrawRectangle(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color,
DOT_PIXEL Line_width, DRAW_FILL Draw_Fill)
```

Parameters:

Xstart: the starting X coordinate of the rectangle

Ystart: indicates the Y starting point of the rectangle

Xend: X terminus of the rectangle

Yend: specifies the y-end coordinate of the rectangle

Color: fill Color

Line_width: The width of the four sides of a rectangle. Default eight widths are provided

```
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: Fill, whether to fill the inside of the rectangle

```
typedef enum {
    DRAW_FILL_EMPTY = 0,
    DRAW_FILL_FULL,
} DRAW_FILL;
```

- Draw circle: In the image buffer, draw a circle of Radius with (X_Center Y_Center) as the center. You can choose the color, the width of the line, and whether to fill the inside of the circle.

```
void Paint_DrawCircle(UWORD X_Center, UWORD Y_Center, UWORD Radius, UWORD Color, DOT_PIXEL Line_width, DRAW_FILL Draw_Fill)
```

Parameters:

X_Center: the x-coordinate of the center of a circle

Y_Center: Y coordinate of the center of a circle

Radius: indicates the Radius of a circle

Color: fill Color

Line_width: The width of the arc, with a default of 8 widths

```
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: fill, whether to fill the inside of the circle

```
typedef enum {
    DRAW_FILL_EMPTY = 0,
    DRAW_FILL_FULL,
} DRAW_FILL;
```

- Write Ascii character: In the image buffer, at (Xstart Ystart) as the left vertex, write an Ascii character, you can select Ascii visual character library, font foreground color, font background color.

```
void Paint_DrawChar(UWORD Xstart, UWORD Ystart, const char Ascii_Char, sFONT* Font, UWORD Color_Foreground, UWORD Color_Background)
```

Parameters:

Xstart: the x-coordinate of the left vertex of a character

Ystart: the Y coordinate of the font's left vertex

Ascii_Char: indicates the Ascii character

Font: Ascii visual character library, in the Fonts folder provides the following

Fonts:

Font8: 5*8 font

Font12: 7*12 font

Font16: 11*16 font

Font20: 14*20 font

Font24: 17*24 font

Color_Foreground: Font color

Color_Background: indicates the background color

- Write English string: In the image buffer, use (Xstart Ystart) as the left vertex, write a string of English characters, can choose Ascii visual character library, font foreground color, font background color.

```
void Paint_DrawString_EN(UWORD Xstart, UWORD Ystart, const char * pString, sFONT* Font, UWORD Color_Foreground, UWORD Color_Background)
```

Parameters:

Xstart: the x-coordinate of the left vertex of a character

Ystart: the Y coordinate of the font's left vertex

PString: string, string is a pointer

Font: Ascii visual character library, in the Fonts folder provides the following

Fonts:

Font8: 5*8 font

Font12: 7*12 font

Font16: 11*16 font

Font20: 14*20 font

Font24: 17*24 font

Color_Foreground: Font color

Color_Background: indicates the background color

- Write Chinese string: in the image buffer, use (Xstart Ystart) as the left vertex, write a string of Chinese characters, you can choose GB2312 encoding character font, font foreground color, font background color.

```
void Paint_DrawString_CN(UWORD Xstart, UWORD Ystart, const char * pString, cFONT* font, UWORD Color_Foreground, UWORD Color_Background)
```

Parameters:

Xstart: the x-coordinate of the left vertex of a character

Ystart: the Y coordinate of the font's left vertex

PString: string, string is a pointer

Font: GB2312 encoding character Font library, in the Fonts folder provides the fo

llowing Fonts:

Font12CN: ASCII font 11*21, Chinese font 16*21

Font24CN: ASCII font24 *41, Chinese font 32*41

Color_Foreground: Font color

Color_Background: indicates the background color

- Write numbers: In the image buffer,use (Xstart Ystart) as the left vertex, write a string of numbers, you can choose Ascii visual character library, font foreground color, font background color.


```
void Paint_DrawNum(UWORD Xpoint, UWORD Ypoint, double Nummber, sFONT* Font, UWORD Digit,
UWORD Color_Foreground, UWORD Color_Background)
```

Parameters:

Xpoint: the x-coordinate of the left vertex of a character

Ypoint: the Y coordinate of the left vertex of the font

Nummber: indicates the number displayed, which can be a decimal

Digit: It's a decimal number

Font: Ascii visual character library, in the Fonts folder provides the following

Fonts:

Font8: 5*8 font

Font12: 7*12 font

Font16: 11*16 font

Font20: 14*20 font

Font24: 17*24 font

Color_Foreground: Font color

Color_Background: indicates the background color

- Display time: in the image buffer, use (Xstart Ystart) as the left vertex, display time, you can choose Ascii visual character font, font foreground color, font background color.

```
void Paint_DrawTime(UWORD Xstart, UWORD Ystart, PAINT_TIME *pTime, sFONT* Font, UWORD Color_Background,
UWORD Color_Foreground)
```

Parameters:

Xstart: the x-coordinate of the left vertex of a character

Ystart: the Y coordinate of the font's left vertex

PTime: display time, here defined a good time structure, as long as the hour, minute and second bits of data to the parameter;

Font: Ascii visual character library, in the Fonts folder provides the following

Fonts:

Font8: 5*8 font

Font12: 7*12 font

Font16: 11*16 font

Font20: 14*20 font

Font24: 17*24 font

Color_Foreground: Font color

Color_Background: indicates the background color

Resource

Drawing

- 3D file (https://files.waveshare.com/wiki/1.51inch%20Transparent%20OLED/1.51inch_Transparent_OLED_3D_file%20.zip)

- 3D Preview (https://files.waveshare.com/wiki/1.51inch%20Transparent%20OLED/1.51inch_Transparent_OLED_3D_preview.zip)

Document

- Schematic (https://files.waveshare.com/upload/0/03/1.51inch_OLED_Transparent_Driver_Board_Schematic.pdf)

Demo Example

- Demo (https://files.waveshare.com/upload/2/2c/OLED_Module_Code.7z)

Software

- Zimo221 (<https://files.waveshare.com/upload/c/c6/Zimo221.7z>)
- Image2Lcd (https://files.waveshare.com/upload/b/bc/Image2Lcd2.9_%287%29.zip)

Datasheet

- SSD1309 Datasheet (<https://files.waveshare.com/upload/9/9c/SSD1309Datasheet.pdf>)

FAQ

Question:How many volts can an OLED module be used in a system?

Answer:

The OLED module is used in a 3.3V system by default, but after more than 24 hours of the burn-in test, it is found that it can also work normally in a 5V system.

Question:How many hours is the service life of an OLED module?

Answer:

Generally it is 50,000 hours under normal working conditions.

Question:What is the working current of the OLED module?

Answer:

At 3.3V operating voltage:

The working current of 0.95inch RGB OLED is about 38mA for full white display, and about 4mA for full black display.

The working current of 0.96inch OLED is about 25mA when fully on and 1.5mA when fully off.

The working current of 1.3inch OLED is about 29mA when fully on and 1.0mA when fully off.

Question:Why does the OLED module not turn on when it is connected to the power supply?

Answer:

OLED does not have a backlight, and the display is self-illuminating. Only connect VCC and GND, and the OLED will not light up.

Program control is required to brighten the OLED.

Question:What should I pay attention to when using this OLED module?

Answer:

1. Be careful not to connect the power supply in reverse.
2. The same picture cannot be displayed for a long time, or there will be afterimages and cause damage to the OLED.

Support

Technical Support

If you need technical support or have any feedback/review, please click the **Submit Now** button to submit a ticket, Our support team will check and reply to you within 1 to 2 working days. Please be patient as we make every effort to help you to resolve the issue.

Working Time: 9 AM - 6 PM GMT+8
(Monday to Friday)

Submit Now (<https://service.waveshare.com/>)

*Retrieved from "https://www.waveshare.com/w/index.php?title=1.51inch_Transparent_OLED&oldid=70813
(https://www.waveshare.com/w/index.php?title=1.51inch_Transparent_OLED&oldid=70813)"*
