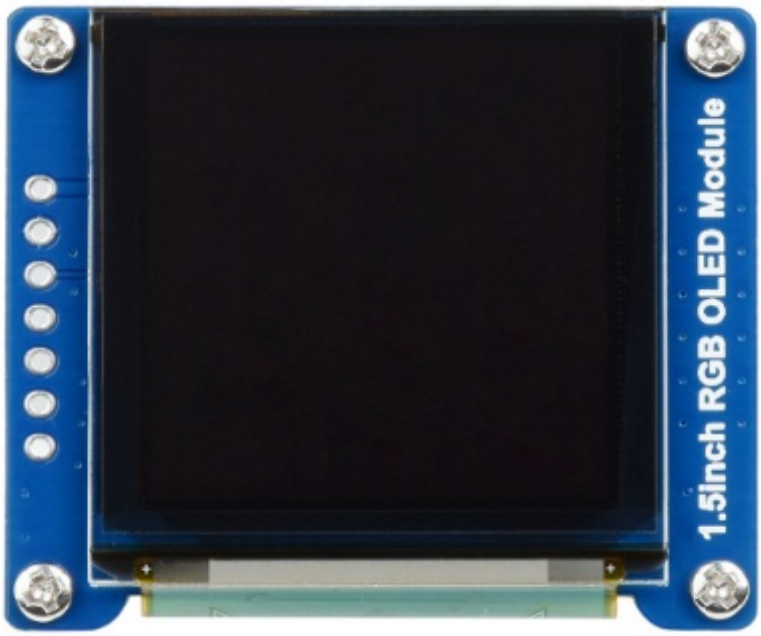
LUCKFOX 1.5 Inch 65K Color OLED Display Module User Guide

Contents

1 LUCKFOX 1.5 Inch 65K Color OLED Display Module
2 Product Usage Instructions
3 Introduction
4 Specification
5 OLED and its controller
6 4WIRE-SPI Communication Protocol
7 Modulo settings
8 Raspberry Pi Software Instructions
9 STM32 Usage Tutorial
10 Software description
11 Demo Description
12 Arduino Tutorial
13 Resources
14 FAQ
15 Support
16 Documents / Resources
16.1 References

LUCKFOX

LUCKFOX 1.5 Inch 65K Color OLED Display Module



Specifications

- Operating voltage: 3.3V/5V (Ensure power supply voltage and logic voltage are consistent)
- Support interface: 4-wire SPI, 3-wire SPI
- Controller: SSD1351
- Resolution: 128(H)RGB x 128(V)
- Dimension: 26.855H x 26.855V mm
- Pixel size: 0.045(H) x 0.194(V) mm
- Display color: 65K color
- 128 x 128, 1.5inch

Product Usage Instructions

Hardware Configuration
This OLED module supports both 4-wire SPI and 3-wire SPI communication methods. The default mode is 4-wire SPI. The communication method can be changed by soldering the appropriate resistor on the back of the module.

OLED and Controller Information
The OLED module uses the SSD1351 controller, supporting 262k and 65k color depths. It utilizes a 128x128x18bit SRAM for the display buffer area and offers various communication methods.

4-Wire SPI Communication Protocol
In 4-wire SPI mode, set the DC pin to indicate command or data transmission. Refer to the SSD1351 datasheet for detailed information on sending commands and data.

3-Wire SPI Communication Protocol
3-wire SPI mode eliminates the need for separate DC pins by adding a bit before each transfer byte to identify command or data transmission. The DC pin needs to be grounded, and each transfer is 9 bits.

Module Settings
The default settings include horizontal scan, 16-bit true color with high order first.

Raspberry Pi Software Instructions
This product offers demos for BCM2835, WiringPi, file IO, and RPI (Python) libraries for Raspberry Pi integration.

Hardware Connection with Raspberry Pi
When connecting to Raspberry Pi, use a 7-pin cable and refer to the pin correspondence table for proper connections.

FAQ

- Q: How do I change the communication method on the OLED module?
  A: To change the communication method, solder the appropriate resistor on the back of the module as per the instructions provided.
- Q: Can this OLED module be used with Arduino?
  A: Yes, this OLED module provides demos for Arduino integration along with Raspberry Pi and STM32.

Introduction

1.5-inch RGB OLED Module provides Raspberry Pi, STM32, and Arduino demos.

Specification

- Operating voltage: 3.3V/5(Please ensure that the power supply voltage and logic voltage are consistent, otherwise it will not work properly)
- Support interface: 4-wire SPI, 3-wire SPI
- Controller: SSD1351
- Resolution: 128(H)RGB x 128(V)
- Dimension: 26.855(H) x 26.855 (V) mm
- Pixel size: 0.045(H) x 0.194(V) mm
- Display color: 65K color

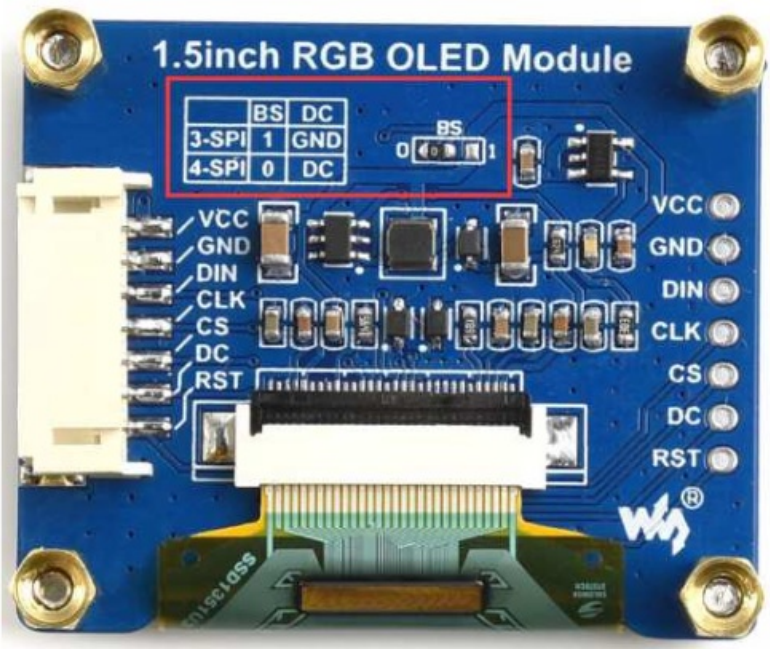
Pinout

PS: This module only has SPI interface, please pay attention when using it.

PIN	Description
VCC	3.3V/5V Power input
GND	Ground
DIN	Data input
SCL	Clock input
CS	Chip select, active low
DC	Data/command signal selection, low level indicates command, high level indicates data
RST	Reset signal, active low

Hardware Configuration

- This OLED module provides two communication methods: 4-wire SPI and 3wire-SPI
- On the back of the module there is an optional solderable resistor through which the communication method is selected.



As shown below:

The module uses 4-wire SPI communication by default, that is, BSO is connected to 0 by default

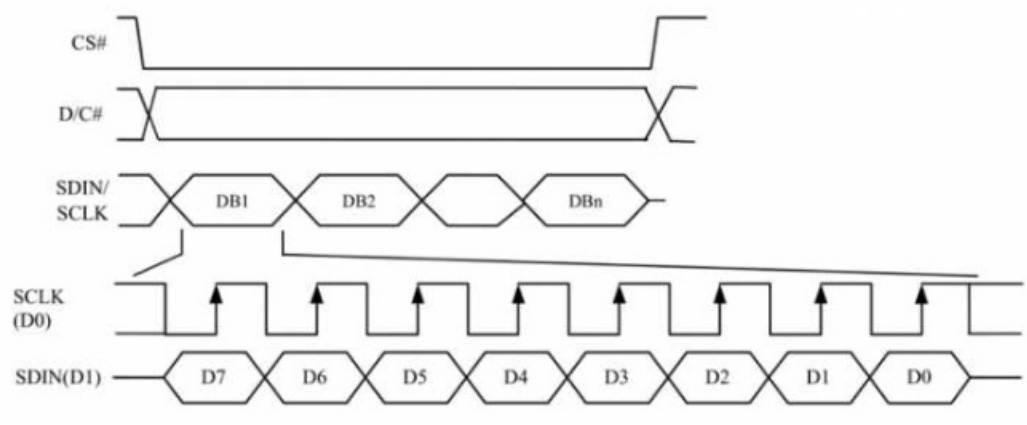
Note: The following table shows the interface connections.

Letter of agreement	BS	CS	DC	DIN	CLK
4Wire SPI	0	Chip Select	DC	MOSI	SCK
3Wire SPI	1	Chip Select	GND	MOSI	SCK

OLED and its controller

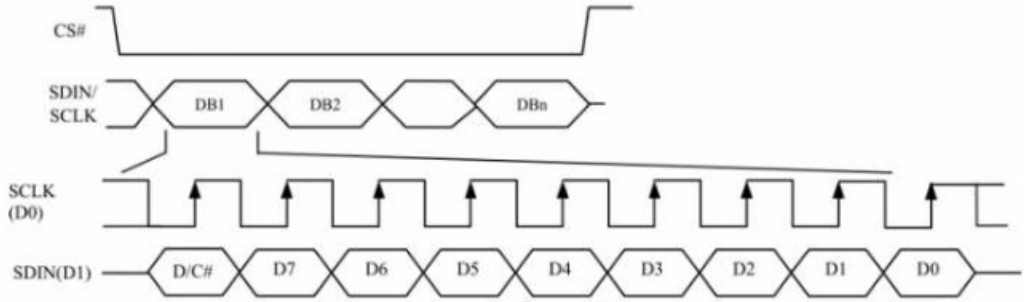
- The built-in driver used by this OLED is SSD1351, which is a 128RGB * 128 Dot Matrix OLED/PLED controller with a 128*128*18bit SRAM as a display buffer area, which supports two color depths of 262k and 65k. And supports 8bit 8080 parallel, 8bit 6800 parallel, 3wire-SPI and 4wire-SPI and other communication methods.
- This module selects 4wire-SPI and 3wire-SPI as the communication method, which reduces the area of the module and saves the 10 resources of the controller.

4WIRE-SPI Communication Protocol



- In 4wire-SPI communication, first set DC to 1 or 0, and then send one or more bytes of data
- When DC is set to 0, the sent byte will be used as a command to control the OLED. When sending a command, generally only one byte is sent at a time.
- When DC is set to 1, the transmitted bytes will be stored in the designated register or SRAM SSD1351 as data. When sending data, multiple bytes can be sent in succession.
- See SSD1351 Datasheet Figure 8-5 for details.

3WIRE-SPI communication protocol



- The only difference between 3wire-SPI and 4wire-SPI is that it removes the DC pins that control sending commands and data
- A bit is added before each SPI transfer byte to identify whether the byte is a command or dat
- Therefore, in 3wire-SPI, the DC pin needs to be grounded. In addition, the data transmitted each time is not 8bit, but 9bit.

Modulo settings

The reference settings are as shown: horizontal scan, 16-bit true color, high order first

The reference settings are as shown: horizontal scan, 16-bit true color, high order first

Raspberry Pi Software Instructions

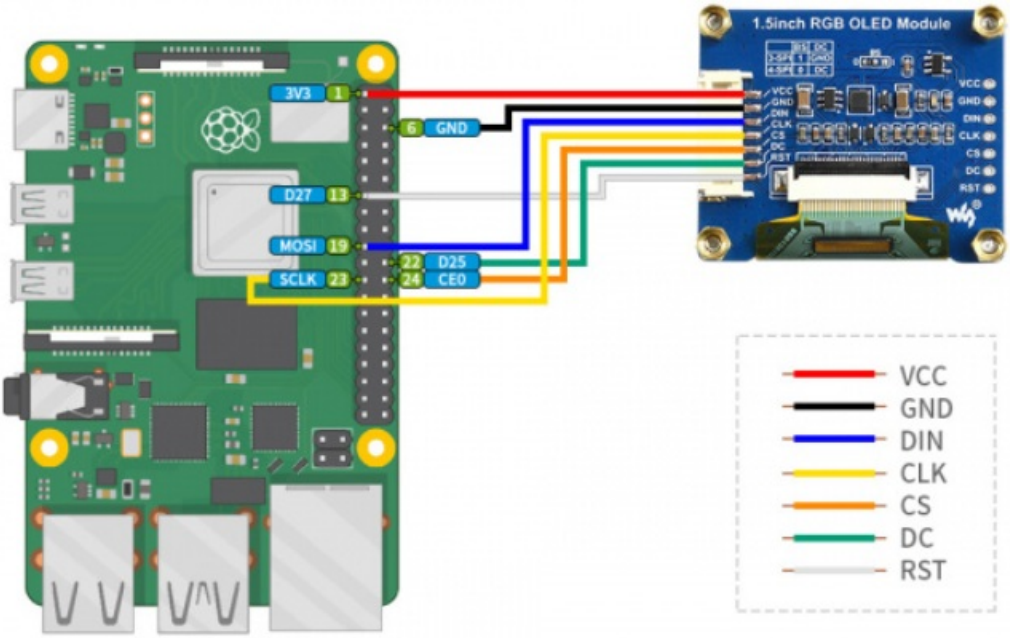
This product provides BCM2835, WiringPi, file IO, RPI (Python) library demos

Hardware connection

When connecting the Raspberry Pi, choose to use the 7PIN cable to connect, please refer to the pin correspondence table below.
Raspberry Pi connection pin correspondence

OLED	Raspberry Pi	
	BCM2835	Board
VCC	3.3V	3.3V
GND	GND	GND
DIN	MOSI / SDA	19 / 3
CLK	SCLK / SCL	23 / 5
CS	CE0	24
DC	25	22
RST	27	13

Four-wire SPI wiring diagram



Enable SPI and I2C interfaces

Open terminal, use command to enter the configuration page

```
sudo raspi-config
Choose Interfacing Options -> SPI -> Yes  to enable 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>
```

Reboot Raspberry P

Please make sure that SPI interface was not used by other devices
I2C is the same, enter the configuration interface and select Interfaceing Options -> I2C -> Yes to open the IIC interface, and then restart

Install Libraries

Install BCM2835 libraries

```
#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 at http://www.airspayce.com/m
ikem/bcm2835/
```

Install wiringPi libraries

```
#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 may be implemented
without), an upgrade may be required to:
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 the
re is an installation 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 ther
e is an installation error.
```

Install Python libraries

```
#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
sudo apt-get install python-smbus
#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
sudo apt-get install python3-smbus
```

Download Examples

Open Raspberry Pi terminal and run the following command

```
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 -O./OLED_Module_Code
cd OLED_Module_Code/RaspberryPi
```

Run the demo codes

Please go into the RaspberryPi directory (demo codes) first and run the commands in terminal

C Codes

- Re-compile the demo codes

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

- After the compilation is complete, the main file is generateendt,e r the command according to the OLED model you are usin gl.f you have purchased a 1.3inch OLED Module (C), please enter the following command:

```
sudo ./main 1.3c
```

- If you have purchased a 1.5inch RGB OLED Module, please enter the following command:

```
sudo ./main 1.5rgb
```

- The command of each LCD model can check in the following table:

```
#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
------------------------------
#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 RGB OLED Module
sudo ./main 1.5rgb
------------------------------
#1.51inch OLED Module
sudo ./main 1.51
```

Python

- Enter the python directory and run

```
cd python/example
```

- Run the demo of the corresponding model OL,E tDhe program 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
```

- The command of each LCD model can check in the following table:

```
#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
------------------------------------
#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 RGB OLED Module
sudo python OLED_1in5_rgb_test.py
------------------------------------
#1.51inch OLED Module
sudo python OLED_1in51_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)

We have carried out the low-level encapsulation, if you need to know the internal implementation can go to the corresponding directory to check, for the reason the hardware platform and the internal implementation are different.
You can open DEV_Config.c(.h) to see definitions,which in the directory RaspberryPi\c\lib\Config

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

```
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
```

- Interface selection:

```
#define USE_SPI_4W  1
#define USE_IIC     0
Note: Modified 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.

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

- 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

The fonts 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

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

```
void Paint_NewImage(UBYTE *image, UWORD Width, UWORD Height, UWORD Rotate, UWOR
D Color)
Parameters:
      Image: the name of the image buffer, which is actually a pointer to the f
irst address of the image buffer;
      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;
```

- 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 rotation Angle of the selected image, 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.
```

- Sets the size of the pixels

```
void Paint_SetMirroring(UBYTE mirror)
Parameters:
        Mirror: indicates the image mirroring mode. MIRROR_NONE, MIRROR_HORIZON
TAL, MIRROR_VERTICAL, MIRROR_ORIGIN correspond to no mirror, horizontal mirror,
vertical mirror, and image center mirror 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_HORIZON
TAL, 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_SetMirroring(UBYTE mirror)
Parameters:
        Mirror: indicates the image mirroring mode. MIRROR_NONE, MIRROR_HORIZON
TAL, MIRROR_VERTICAL, MIRROR_ORIGIN correspond to no mirror, horizontal mirror,
vertical mirror, and image center mirror respectively.
```

- 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, UWO
RD 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 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_Pix
el, 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, the demo provides 8 size pointss by def
ault.
              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 size of a point that expands from the center of the poin
t 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 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 C
olor, LINE_STYLE Line_Style ,  LINE_STYLE Line_Style)
Parameters:
        Xstart: the x-starting coordinate of a line
        Ystart: the y-starting coordinate of the a line
        Xend: the x-end coordinate of a line
        Yend: the y-end coordinate of a line
        Color: fill Color
        Line_width: The width of the line, the demo provides 8 sizes of width b
y 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 straig
ht 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, UW
ORD 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 pr
ovides 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;
         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 Colo
r, 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: 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 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, 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* F
ont, 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 prov
ides 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, you can choose Ascii visual character library, font foreground color, font background color.

```
void Paint_DrawString_EN(UWORD Xstart, UWORD Ystart, const char * pString, sFON
T* 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 the demo prov
ides 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 character font, font foreground color, font background color of the GB2312 encoding

```
void Paint_DrawString_CN(UWORD Xstart, UWORD Ystart, const char * pString, cFON
T* 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 the d
emo provides the following 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, UWO
RD 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 the demo prov
ides 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, A time structure is defined here, as long as the h
ours, minutes and seconds are passed to the parameters;
        Font: Ascii visual character library, in the Fonts folder the demo prov
ides 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
```

Python(for Raspberry Pi)
It is compatible with python2.7 and python3.
The calls of the python are less complex compared to C demo.
config.py

Select interface.

```
Device_SPI = 1
Device_I2C = 0
Note: Switch SPI/I2C modified here
```

Module initialization and exit processing

```
def module_init()
def module_exit()
Note:
1. Here is some GPIO processing before and after using the LCD screen.
2. The module_init() function is automatically called in the INIT () initialize
r on the LCD, but the module_exit() function needs to be called by itself
```

- SPI write data.
  - def spi_writebyte(data)
- IIC write data.
  - i2c_writebyte(reg, value):

main.py
The main function, if your Python version is Python2, is re-executed in Linux command mode as follows. sudo python main.py
If your Python version is Python3, run the following command in Linux. sudo python3 main.py

GUI functions
Python has an image librarPy IL official library link , it do not need to write code from the logical layer like C, can directly call to the image library for image processing. The following will take 1.54inch LCD as an example, we provide a brief description for 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, and ImageFont.
Among them, Image is the basic library, ImageDraw is the drawing function, and ImageFont is the text function.

Define an image buffer
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 on here.
draw = ImageDraw.Draw(image1)

Draw line.

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

Draw rectangle

draw.rectangle([(20,10),(70,60)],fill = "WHITE",outline="BLACK")
The first argument 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 BLACK inside, and outline="BLACK" means the color of the outline is black.

Draw 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, the second parameter indicates the starting angle, the third parameter indicates the ending angle, and fill = 0 indicates that the the color of the line is white. If the figure is not square according to the coordination, you will get an ellipse.
Besides the arc function, you can also use the chord function for drawing solid circle.

draw.ellipse((150,65,190,105), fill = 0)
The first parameter is the coordination of the enclosing rectangle. The second and third parameters are the beginning and end degrees of the circle. The fourth parameter is the fill color of the circle.

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 ro used. To draw English character, you can directly use the fonts; for Chinese character, you need to add a symbol u:

- draw.text((5, 68), 'Hello world', fill = 0, font=Font1)
- text= u" "
- 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 font color, fill = 0 means that the font color is white, and the second sentence sh o ws ' ', font color is white.

Read local picture.
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 implement more, you can learn on the website http://effbot.org/imagingbook pil.

STM32 Usage Tutorial

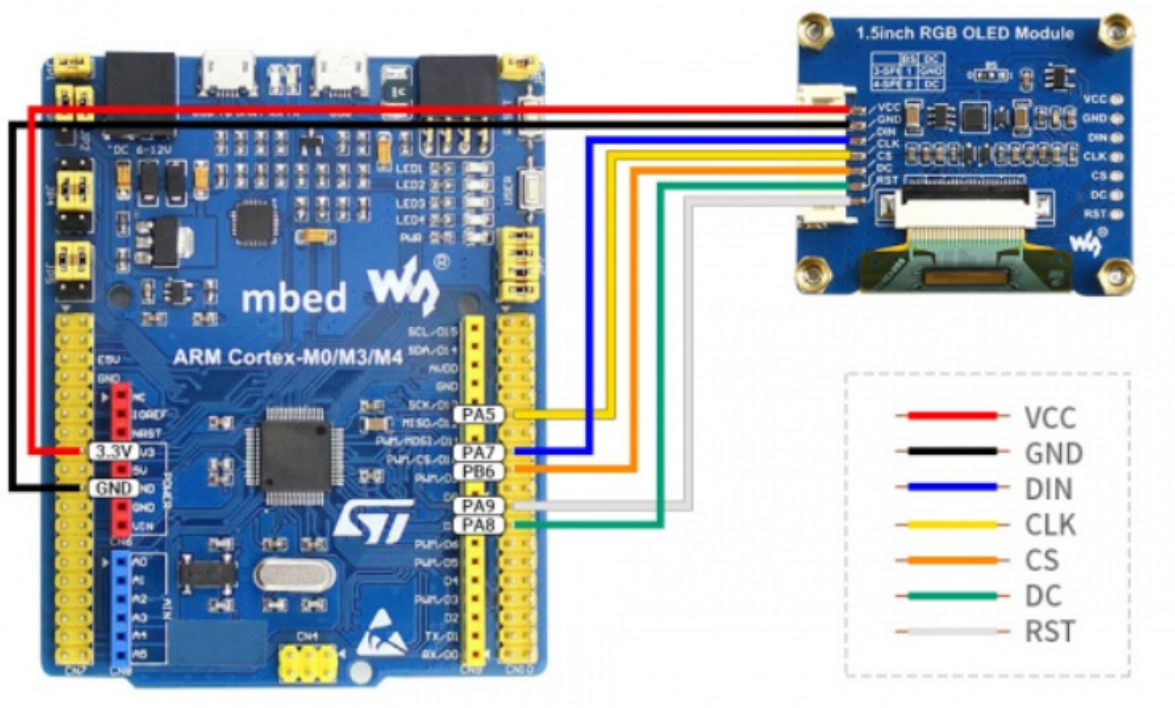
Provides demos based on STM32F103RBT6

Hardware Coonnection
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
SCL	SPI:PA5/I2C:PB8 / I2C_SOFT PC6
CS	PB6
DC	PA8
RST	PA9

Four-wire SPI wiring diagram



Run the demo

- Download the demo, find the STM32 demo file directory, use Keil5 to open oled_demo.uvprojx in the \STM32\STM32-F103RBT6\MDK-ARM directory.
- Then modify the corresponding function comment in 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_1in3_test();   //  IIC must USE_IIC_SOFT
104
105  //  OLED_1in3_c_test();
106
107  //  OLED_1in5_test();
108
109  //  OLED_1in5_rgb_test(); //  Only SPI !!!
```

- For examples, if you are using 1.3inch OLED Modu leC ,you need to comment out the 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 demo folder of each LCD model can fine in the following table:

LCD Model	Demo function
0.91inch OLED Module	`OLED_0in91_test();`
0.95inch RGB OLED (A)/(B)	`OLED_0in95_rgb_test();`
0.96inch OLED (A)/(B)	`OLED_0in96_test();`
1.3inch OLED (A)/(B)	`OLED_1in3_test();`
1.3inch OLED Module (C)	`OLED_1in3_c_test();`
1.5inch OLED Module	`OLED_1in5_test();`
1.5inch RGB OLED Module	`OLED_1in5_rgb_test();`

Software description

The demo is developed based on the HAL library. Download the demo, find the STM32 program file directory, and open the oled_demo.uvprojx in the STM32\STM32F103RBT6\MDK-ARM directory to check the program

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 program, font, GUI, and OLED driver

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: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:
Here is some GPIO processing before and after using the LCD screen.
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);

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)

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

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

- void Paint_SetClearFuntion(void (*Clear)(UWORD));
- parameter:
- Clear : Pointer to the clear screen function, used to quickly clear the scr
- een 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, 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 cache, which is actually a pointer to the firs
- t 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, 9
- 0, 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, vert
- ical 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_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, UWO RD 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 point: 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_Pix
el, 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 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;
```

Draw line: In the image buffer, draw 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 C
olor, 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 de
fault.
        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 o
r 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, UW
ORD 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 pr
ovides 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 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 Colo
r, 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: 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 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, 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* F
ont, 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 foll
owing 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, you can choose Ascii visual character library, font foreground color, font background color.

void Paint_DrawString_EN(UWORD Xstart, UWORD Ystart, const char * pString, sFON
T* 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 foll
owing 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, cFON
T* 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 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, UWO
RD 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 foll
owing 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 left vertex of a character
PTime: display time, here defined a good time structure, as long as the hou
r, minute and second bits of data to the parameter;

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

Arduino Tutorial

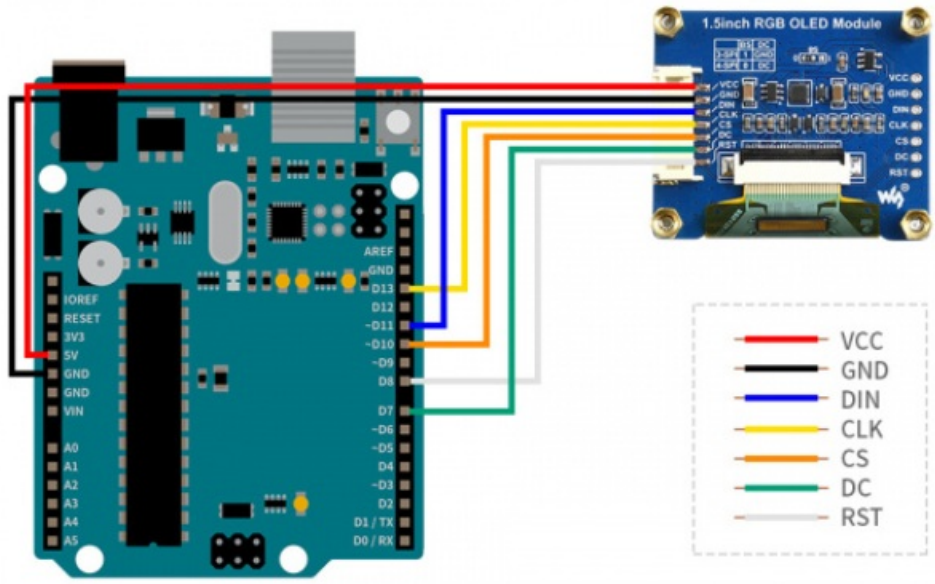
Provides demos based on UNO PLUS

Hardware connection
The demos we provide are based on UNO PLUS, and the connection method provided is also the corresponding UNO PLUS pins. If you need to transplant the program, please connect according to the actual pins.

Arduino UNO connection

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

- Four-wire SPI wiring diagram



- How to install Arduino IDE

Run the demo

- Download the demo, find the STM32 demo file directory, use Keil5 to open oled_demo.uvprojx in the \STM32\STM32-F103RBT6\MDK-ARM directory.

- Then modify the corresponding function comment in 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_1in3_test();   //  IIC must USE_IIC_SOFT
104
105  //  OLED_1in3_c_test();
106
107  //  OLED_1in5_test();
108
109  //  OLED_1in5_rgb_test(); //  Only SPI !!!
```

- For examples, if you are using 1.3inch OLED Module C ,you need to comment out the 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 demo folder of each LCD model can fine in the following table:

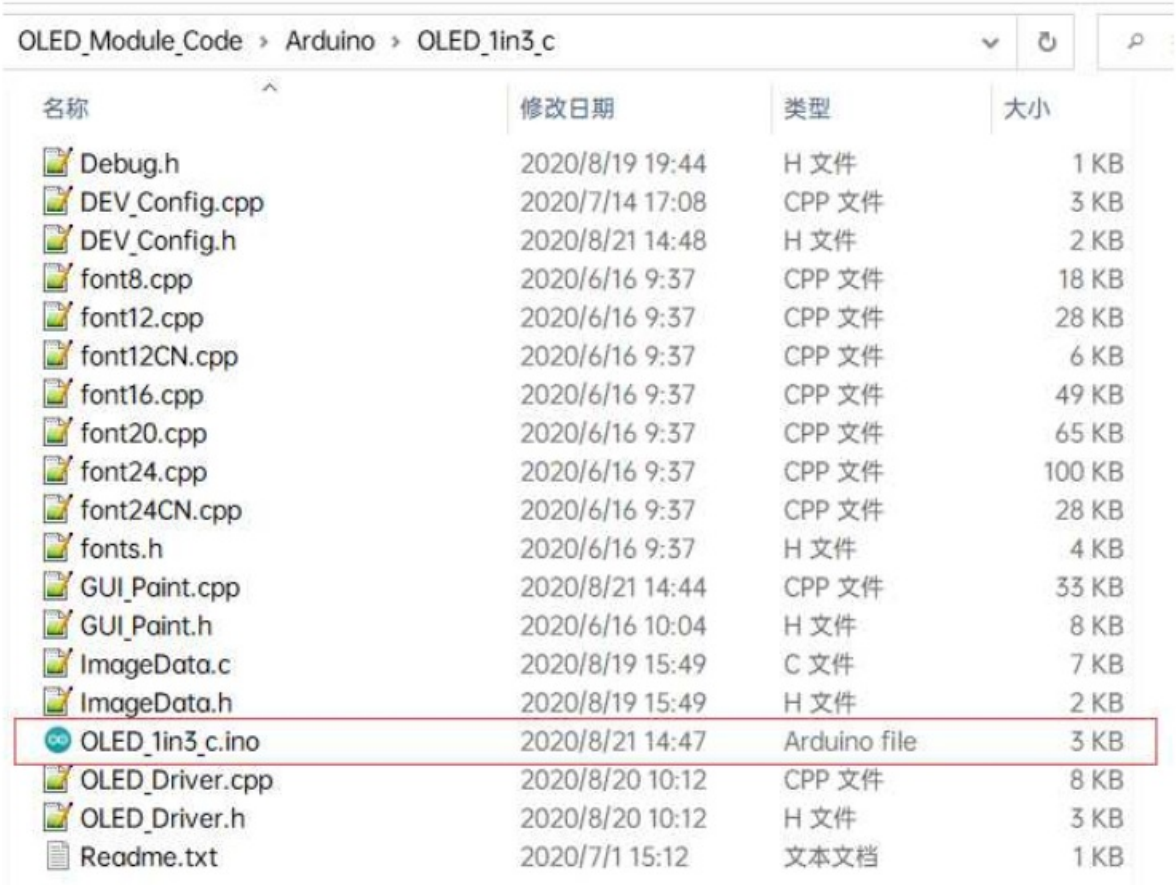
Software Description

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

OLED_Module_Code › Arduino

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

- 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.



OLED_Module_Code › Arduino › OLED_1in3_c

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

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

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
- 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:

- here is some GPIO processing before and after using the LCD screen.
- 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: Arduino\OLED_xxx\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

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

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_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;
void Paint_SetClearFuntion(void (*Clear)(UWORD));
parameter:
Clear : Pointer to the clear screen function, used to quickly clear the scr
een 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 firs
t 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, 9
0, 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, vert
ical 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, UWO
RD 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_Pix
el, 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 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 C
olor, 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 wi
dths
                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 straig
ht 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, UW
ORD 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 w
idths are provided
        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 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 Colo
r, 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 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* F
ont, 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, sFON
T* 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, cFON
T* 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 provi
des the following 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, UWO
RD 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
```

Resources

Provide a full set of documents, procedures, data sheets, etc.

- Documentation
  Schematic
- Program
  Sample Program
- 3D drawings
  3D Drawing

- Software

  LCD

  Image2lcd
- Data sheet

  SDD1351 Datasheet

  1.5inch RGB OLED Datasheet

FAQ

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

  Answer: At the operating voltage of 3.3V: about 60mA for full white and 4mA for full black.
- **Question: Why does the OLED module not turn on when it is connected to the power supply?**

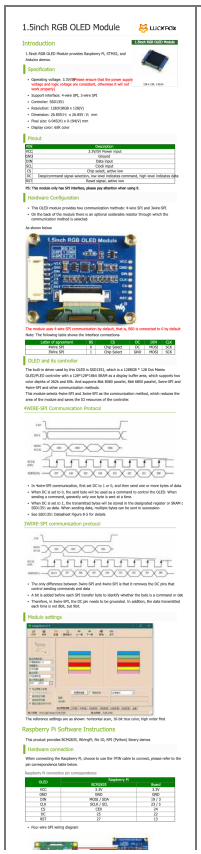
  Answer: There is no backlight, and the display is self-illuminating. Only connect VCC and GND, the OLED will not light up. Program control is required to brighten the OLED.

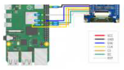
Support

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 AM GMT+8 (Monday to Friday)

Documents / Resources

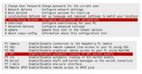




Enable SPI and I2C interfaces

- Open terminal, use command to enter the configuration page

```
sudo raspi-config
Choose Interfacing Options -> SPI -> Yes  to enable SPI interface
```



Reboot Raspberry Pi

```
sudo reboot
```

Please make sure that SPI interface was not used by other devices
I2C is the same, enter the configuration interface and select Interfaceing Options -> I2C -> Yes to open the IIC interface, and then restart

Install Libraries

- Install BCM2835 libraries

```
#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 at http://www.airspayce.com/m
ikem/bcm2835/
```

- Install wiringPi libraries

```
#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 may be implemented
without), an upgrade may be required to:
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 the
re is an installation 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 ther
e is an installation error.
```

- Install Python libraries

```
#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
sudo apt-get install python-smbus
#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
sudo apt-get install python3-smbus
```

Download Examples

Open Raspberry Pi terminal and run the following command

```
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 -O./OLED_Module_Code
cd OLED_Module_Code/RaspberryPi
```

Run the demo codes

Please go into the RaspberryPi directory (demo codes) first and run the commands in terminal

C Codes

- Re-compile the demo codes

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

- After the compilation is complete, the main file is generated**, enter the command according to the OLED model you are using.** If you have purchased a 1.3inch OLED Module (C), please enter the following command:

```
sudo ./main 1.3c
```

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

```
sudo ./main 1.5rgb
```

The command of each LCD model can check in the following table:

```
#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
------------------------------
#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 RGB OLED Module
sudo ./main 1.5rgb
------------------------------
#1.51inch OLED Module
sudo ./main 1.51
```

Python

- Enter the python directory and run

```
cd python/example
```

- **Run the demo of the corresponding model OLED**, the program 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
```

The command of each LCD model can check in the following table:

```
#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
-----------------------------------
#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 RGB OLED Module
sudo python OLED_1in5_rgb_test.py
-----------------------------------
#1.51inch OLED Module
sudo python OLED_1in51_test.py
```

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

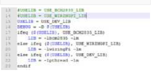
Description of C codes (API)

Hardware interface

We have carried out the low-level encapsulation, if you need to know the internal implementation can go to the corresponding directory to check, for the reason the hardware platform and the internal implementation are different.

You can open DEV_Config.c(.h) to see definitions,which in the directory RaspberryPi\c\lib\Config

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



```
#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
```

- Interface selection:

```
#define USE_SPI_4W  1
#define USE_IIC     0
Note: Modified 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 is 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).

The fonts can be found in RaspberryPi\c\lib\Fonts directory.

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

```
void Paint_NewImage(UBYTE *image, UWORD Width, UWORD Height, UWORD Rotate, UWOR
D Color)
Parameters:
      Image: the name of the image buffer, which is actually a pointer to the f
irst address of the image buffer;
      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;
```

- 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 rotation Angle of the selected image, 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.
```

- 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 pixe
l 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_HORIZON
TAL, 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: 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_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, UWO
RD 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 points on (Xpoint, Ypoint), you can choose the color, th size of the point, the style of the point.

```
void Paint_DrawPoint(UWORD Xpoint, UWORD Ypoint, UWORD Color, DOT_PIXEL Dot_Pix
el, 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, the demo provides 8 size pointss by def
ault.
              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 size of a point that expands from the center of the poin
t or from the bottom left corner of the point to the right and up.
                 typedef enum {
```

LUCKFOX 1.5 Inch 65K Color OLED Display Module [pdf] User Guide
1.5 Inch 65K Color OLED Display Module, 65K Color OLED Display Module, OLED Display Module, Display Module, Module

- In addition, you can see the file directory of the project in the STM32CT6D-FU38B7E51user directory. The five folders are the underlying drive, sample program, font, GB2, and OLED driver.

Demo Description

Hardware Interface

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

- Interface selection

```
#define USE_SPI_4W              1
#define USE_SPI_3W              0
#define USE_IIC_OPT             0
Note:Priority SPI/IIC Already enabled here
```

- Data type

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

- Module initialization and exit processing

```
UBYTE System_Init(void);
void System_Exit(void);
Note:
1. Make it work GPIO processing before and after using the LCD screen;
2. When the System_Init() function is used, the SPI 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 DEV_SPI_Write(uint8_t Value);
```

- IIC write data

```
void I2C_Write_Byte(uint8_t Value, uint8_t Add);
```

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
STM32CT6D-FU38B7E51user/GUI/GUI_Paint.c(.h)

The character font which GUI dependent is in the directory STM32CT6D-FU38B7E51user/Fonts

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

```
void Paint_NewImage(UBYTE *image, UWORD Width, UWORD Height, UWORD Rotate,
Parameters:
    image: image buffer header;
    Width: the width of the image buffer;
    Height: Enclosure the creation height of an image
    Color: the initial Color of the image;
```

- Set the clear screen function

```
void Paint_SetClearFunction(void (*Clear)(UWORD));
parameters:
    Clear: Pointer to the clear screen function, used to quickly clear the scr
een to a specific color.
```

- Set the drawing pixel function.

```
void Paint_SetDisplayFunction(void (*Display)(UWORD,UWORD,UBYTE));
parameters:
    Display: Pointer to the pixel drawing function, which is used to write data
to the specified location in the external 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 cache. which is actually a pointer to the fir
st 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, 9
0, 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: indicate the image mirroring mode. MIRROR_NONE, MIRROR_HORIZONTAL,
MIRROR_VERTICAL, MIRROR_ORIGIN correspond to no mirror, horizontal mirror, ver
tical mirror, and about image center mirror respectively.
```

- Set pointer of display position and color in the buffer: here is the core GUI function, processing centre 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: Fill 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 flash the screen into blank, often used for time display. Flash the last second of the screen.

```
void Paint_ClearWindows(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UW
ORD 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;
    line_color: The color of the line, the demo provides 5 types of colors by de
fault.
        typedef enum {
            DOT_PIXEL_1X1  =1,  // 1 x 1
            DOT_PIXEL_2X2  =2,  // 2 x 2
            DOT_PIXEL_3X3  =3,  // 3 x 3
            DOT_PIXEL_4X4  =4,  // 4 x 4
            DOT_PIXEL_5X5  =5,  // 5 x 5
            DOT_PIXEL_6X6  =6,  // 6 x 6
            DOT_PIXEL_7X7  =7,  // 7 x 7
            DOT_PIXEL_8X8  =8,  // 8 x 8
        } DOT_PIXEL;
    Dot_Pixel: the size of a point (the separate from the center of the point to
four sides, for example, the size of the point on the right and up
        typedef enum {
            DOT_FILL_AROUND =1,
            DOT_FILL_RIGHTUP=2,
        } DOT_FILL;
```

- Draw dot: In the image buffer, draw one 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_PIXEL Line_Pixel)
Parameters:
    Xstart: the starting coordinate of the line
    Ystart: the starting coordinate of the line
    Xend: the end coordinate of the line
    Yend: the end coordinate of the line
    Color: Fill Color
    Line_color: The color of the line, the demo provides 5 types of colors by de
fault.
        typedef enum {
            DOT_PIXEL_1X1  =1,  // 1 x 1
            DOT_PIXEL_2X2  =2,  // 2 x 2
            DOT_PIXEL_3X3  =3,  // 3 x 3
            DOT_PIXEL_4X4  =4,  // 4 x 4
            DOT_PIXEL_5X5  =5,  // 5 x 5
            DOT_PIXEL_6X6  =6,  // 6 x 6
            DOT_PIXEL_7X7  =7,  // 7 x 7
            DOT_PIXEL_8X8  =8,  // 8 x 8
        } DOT_PIXEL;
    Line_Style: Line style, which defines the line style and is divided into s
olid and dotted lines.
        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, UW
ORD 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 ending X coordinate of the rectangle
    Yend: the ending Y coordinate of the rectangle
    Color: Fill Color
    Line_width: The width of the four sides of a rectangle, but the demo pr
ovides 8 kinds of widths by default.
        typedef enum {
            DOT_PIXEL_1X1  =1,  // 1 x 1
            DOT_PIXEL_2X2  =2,  // 2 x 2
            DOT_PIXEL_3X3  =3,  // 3 x 3
            DOT_PIXEL_4X4  =4,  // 4 x 4
            DOT_PIXEL_5X5  =5,  // 5 x 5
            DOT_PIXEL_6X6  =6,  // 6 x 6
            DOT_PIXEL_7X7  =7,  // 7 x 7
            DOT_PIXEL_8X8  =8,  // 8 x 8
        } DOT_PIXEL;
    Draw_Fill: Whether to fill the inside of the rectangle
        typedef enum {
            DRAW_FILL_EMPTY =1,
            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 Colo
r,DOT_PIXEL Line_width, DRAW_FILL Draw_Fill)
Parameters:
    X_Center: the circumscribed of the centre of a circle
    Y_Center: the circumscribed of the centre 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 kinds
        typedef enum {
            DOT_PIXEL_1X1  =1,  // 1 x 1
            DOT_PIXEL_2X2  =2,  // 2 x 2
            DOT_PIXEL_3X3  =3,  // 3 x 3
            DOT_PIXEL_4X4  =4,  // 4 x 4
            DOT_PIXEL_5X5  =5,  // 5 x 5
            DOT_PIXEL_6X6  =6,  // 6 x 6
            DOT_PIXEL_7X7  =7,  // 7 x 7
            DOT_PIXEL_8X8  =8,  // 8 x 8
        } DOT_PIXEL;
    Draw_Fill: Draw_Fill: Whether to fill the inside of the circle
        typedef enum {
            DRAW_FILL_EMPTY =1,
            DRAW_FILL_Full,
        } DRAW_FILL;
```

- Write dot character: In the image buffer, use (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, char const Ascii_Char, sFONT *
font, UWORD Color_Foreground, UWORD Color_Background)
Parameters:
    Xstart: the coordinates of the left vertex of a character
    Ystart: the coordinates of the left vertex of a character
    Ascii_Char: indicates the Ascii character
    font: Ascii visual character library, in the Fonts folder provides the fol
lowing fonts
        font8: 5*8 font
        font12: 7*12 font
        font16: 11*16 font
        font24: 16*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, you can choose dash visual character library, font foreground color, font background color.

```
void Paint_DrawString_EN(UWORD Xstart, UWORD Ystart, const char * pString, sFO
NT *font, UWORD Color_Foreground, UWORD Color_Background)
Parameters:
    Xstart: the coordinates of the left vertex of a character
    Ystart: the coordinates of the left vertex of a character
    pString: string, string to be printed
    font: Ascii visual character library, in the Fonts folder provides the fol
lowing fonts
        font8: 5*8 font
        font12: 7*12 font
        font16: 11*16 font
        font24: 16*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, sFO
NT *font, UWORD Color_Foreground, UWORD Color_Background)
Parameters:
    Xstart: the coordinates of the left vertex of a character
    Ystart: the coordinates of the left vertex of a character
    pString: string, string to be printed
    font: GB2312 encoding character font library, in the Fonts folder provides
the following fonts
        font12CN: GB2312 Simple Font 11*12 dot
        font24CN: GB2312 Simple Font 16*24 dot
    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 dash visual character library, font foreground color, font background color

```
void Paint_DrawNum(UWORD Xpoint, UWORD Ypoint, double Number, sFONT *Font, UW
ORD Color, UWORD Color_Background)
Parameters:
    Xstart: the coordinates of the left vertex of a character
    Ystart: the coordinates of the left vertex of a character
    Number: Number, the number to be displayed, which can be a decimal
    Digit: Bit a decimal number
    font: Ascii visual character library, in the Fonts folder provides the fol
lowing fonts
        font8: 5*8 font
        font12: 7*12 font
        font16: 11*16 font
        font24: 16*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 dash visual character font, font foreground color, font background color

```
void Paint_DrawTime(UWORD Xstart, UWORD Ystart, PAINT_TIME *pTime, sFONT *Fon
t, UWORD Color_Foreground, UWORD Color_Background)
Parameters:
    Xstart: the coordinates of the left vertex of a character
    Ystart: the coordinates of the left vertex of a character
    pTime: display time, take defined a good time structure, as long as the for
mat is consistent with the time structure of this demo.
    font: Ascii visual character library, in the Fonts folder provides the fol
lowing fonts
        font8: 5*8 font
        font12: 7*12 font
        font16: 11*16 font
        font24: 16*24 font
    Color_Foreground: Font color
    Color_Background: indicates the background color
```

Arduino Tutorial

Provides demos based on GND PLUS

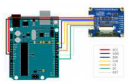
Hardware connection

The demos we provide are based on GND PLUS, and the connection method provided is also the corresponding GND PLUS pins. If you need to transplant the program, please connect according to the actual pins.

Arduino GND connection:

TFT	PIN
VCC	5V
GND	GND
SDA	SPI:D11 / I2C:SDA
CLK	SPI:D13 / I2C:SCL
CS	D10
DC	D9
RST	D8

- Four-wire SPI wiring diagram:



- How to install Arduino IDE

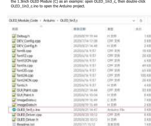
Run the demo:

- 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 examples of different models of OLED.

- 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.



- The demo folder of each LCD model can fine 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
Note:Switch SPI/IIC 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: Arduino\OLED_xxx\GUI_Paint.c(.h)

- 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 scr
een 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 firs
t 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, 9
0, 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, vert
ical 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, UWO
RD 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_Pix
el, 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 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 C
olor, 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 wi
dths
                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 straig
ht 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, UW
ORD 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 w
idths are provided
        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 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 Colo
r, 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 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* F
ont, 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 o 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, sFON
T* 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 c 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, cFON
T* 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 provi
des the following 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, UWO
RD 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
```

Resources

Provide a full set of documents, procedures, data sheets, etc.

Documentation

- Schematic

Program

- Sample Program

3D drawings

- 3D Drawing

Software

- LCD
- Image2lcd

Data sheet

- SDD1351 Datasheet
- 1.5inch RGB OLED Datasheet

FAQ **Question:**What is the working current of the OLED module? **Answer:** At the operating voltage of 3.3V, about 80mA for full white and 4mA for full black. **Question:**Why does the OLED module not turn on when it is connected to the power source? **Answer:** There is no backlight, and the display is self-illuminating. Only connect VCC and GND, the OLED will not light up. Program control is required to brighten the OLED. Support **Technical Support** If you need technical support or have any feedback/review, please click **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 and wait for our reply. Thank you for your time! Working Time: 9:00 - 6:00 GMT+8 (Monday to Friday) Submit Now	

References

- airspayce.com/mikem/bcm2835/bcm2835-1.71.tar.gz
- GitHub - WiringPi/WiringPi: The arguably fastest GPIO Library for the Raspberry Pi
- project-downloads.drogon.net/wiringpi-latest.deb
- User Manual

Manuals+, Privacy Policy

This website is an independent publication and is neither affiliated with nor endorsed by any of the trademark owners. The "Bluetooth®" word mark and logos are registered trademarks owned by Bluetooth SIG, Inc. The "Wi-Fi®" word mark and logos are registered trademarks owned by the Wi-Fi Alliance. Any use of these marks on this website does not imply any affiliation with or endorsement.