

1.28inch LCD Module

From Waveshare Wiki

Jump to: navigation, search

Overview

Introduction

1.28inch LCD Display Module, IPS Screen, 65K RGB, 240 × 240 Resolution, SPI Interface.

More (<https://www.waveshare.com/product/1.28inch-lcd-module.htm?amazon>)

1.28inch LCD Module



(<https://www.waveshare.com/product/1.28inch-lcd-module.htm>)

1.28inch LCD, SPI interfaces

Specification

- Operating voltage: 3.3V/5V
- Interface: SPI
- LCD type: IPS
- Controller: GC9A01
- Resolution: 240 (H)RGB x 240(V)
- Display size: Φ32.4mm
- Pixel size: 0.135 (H) x 0.135 (V) mm
- Dimension: 40.4 × 37.5(mm) Φ37.5(mm)

Interface Description

Raspberry Pi

Please connect the LCD to your Raspberry Pi by the 8PIN cable according to the table below

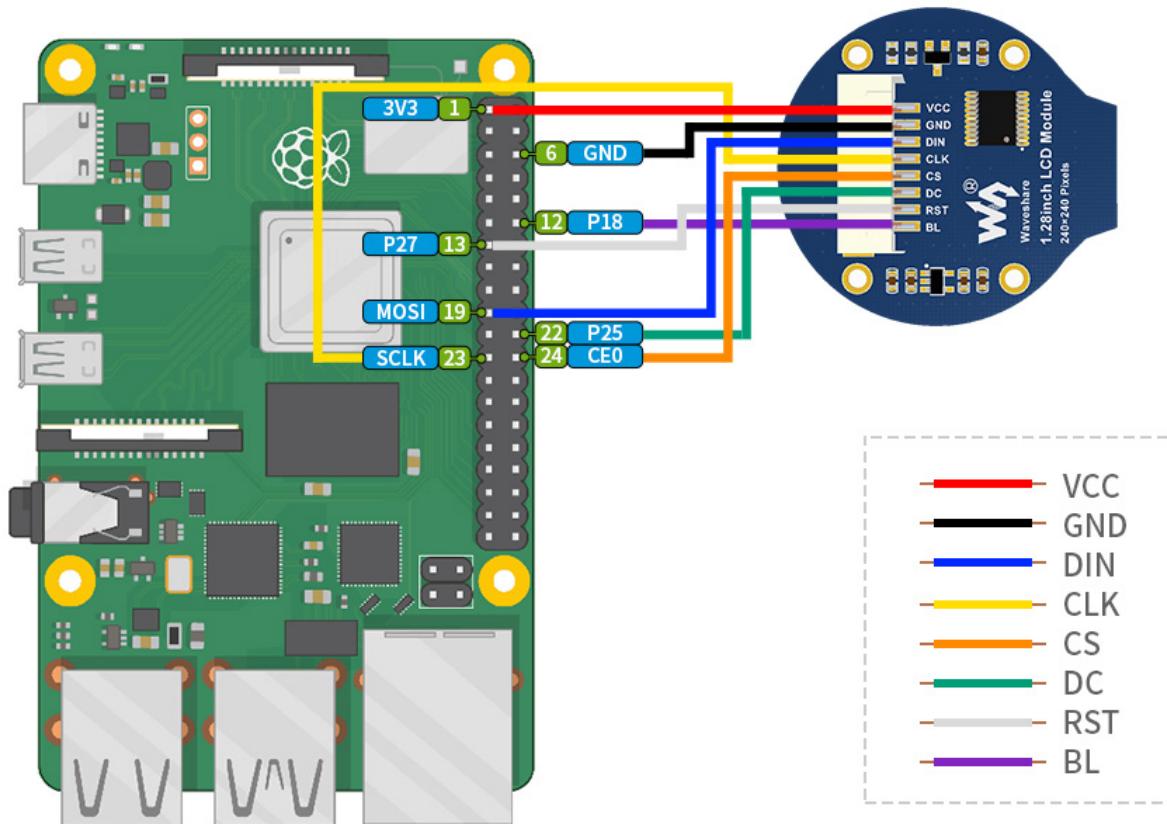
Use the pin header or PH2.0 8PIN interface, you need to connect according to the following table

Connect to Raspberry Pi

LCD	Raspberry Pi	
	BCM2835	Board
VCC	3.3V	3.3V
GND	GND	GND
DIN	MOSI	19
CLK	SCLK	23
CS	CE0	24

DS	25	22
RST	27	13
BL	18	12

The 1.28inch LCD uses the PH2.0 8PIN interface, which can be connected to the Raspberry Pi according to the above table: (Please connect according to the pin definition table. The color of the wiring in the picture is for reference only, and the actual color shall prevail.)



(/wiki/File:1.28-%E6%A0%91%E8%8E%93%E6%B4%BE.jpg)

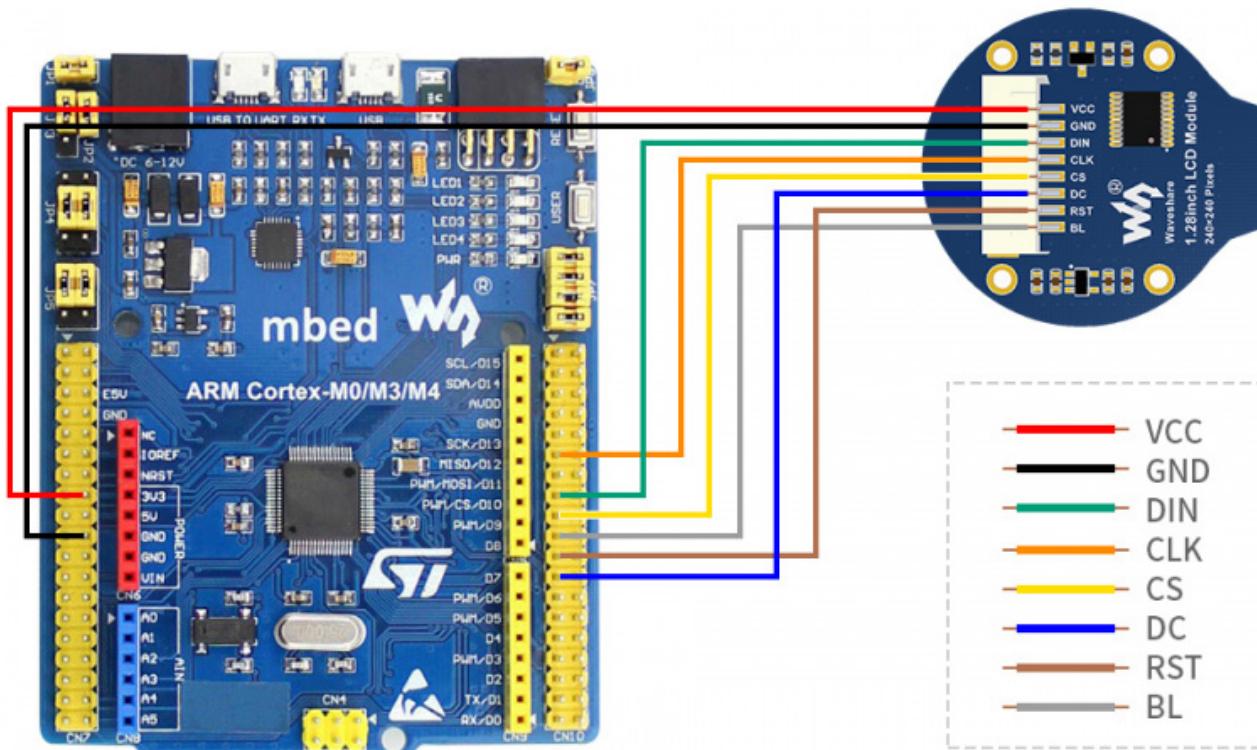
STM32

The example we provide is based on STM32F103RBT6, and the connection method provided is also the corresponding pin of STM32F103RBT6. If you need to transplant the program, please connect according to the actual pin.

STM32F103ZET connection pin correspondence

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

Take the XNUCLEO-F103RB development board (<https://www.waveshare.com/xnucleo-f103rb.htm>) developed by our company as an example, the connection is as follows:



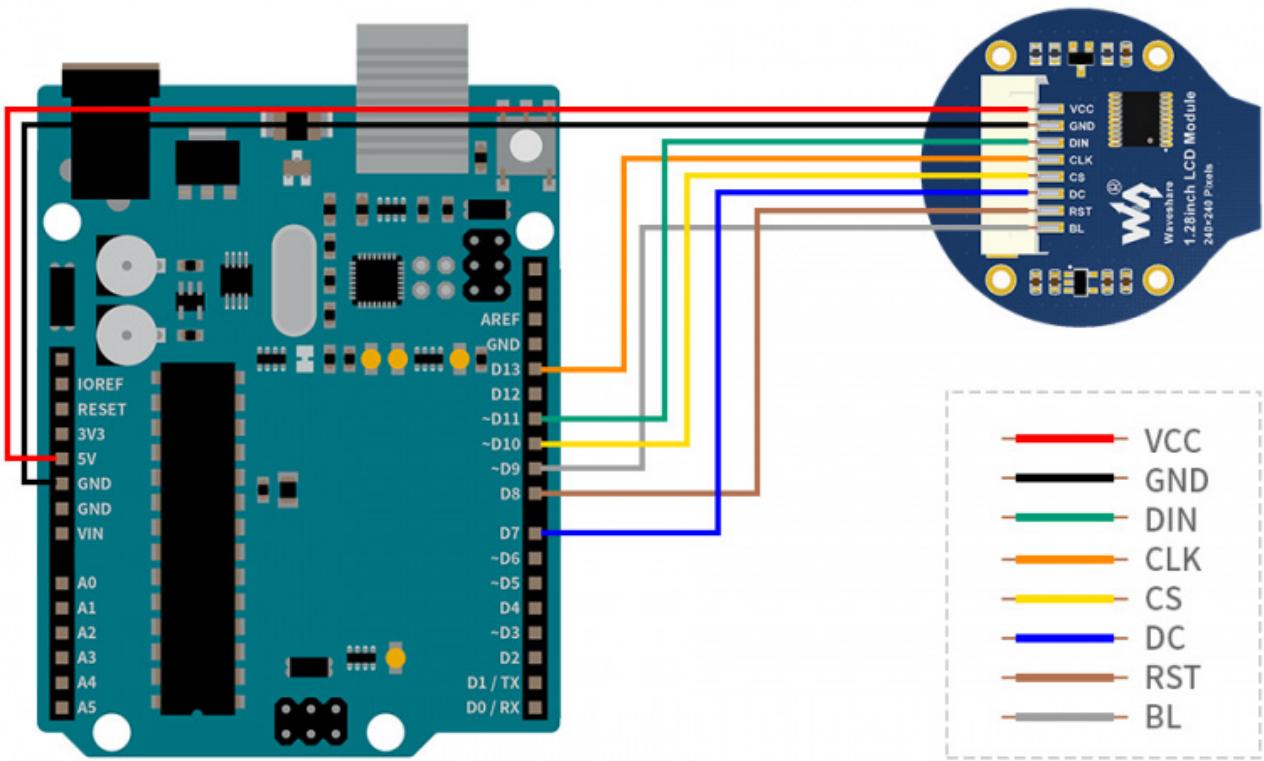
(/wiki/File:1.28-STM32.jpg)

Arduino

Arduino UNO Connection pin correspondence

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

The connection diagram is as follows (click to enlarge):



(/wiki/File:1.28-Aduino.jpg)

LCD & Controller

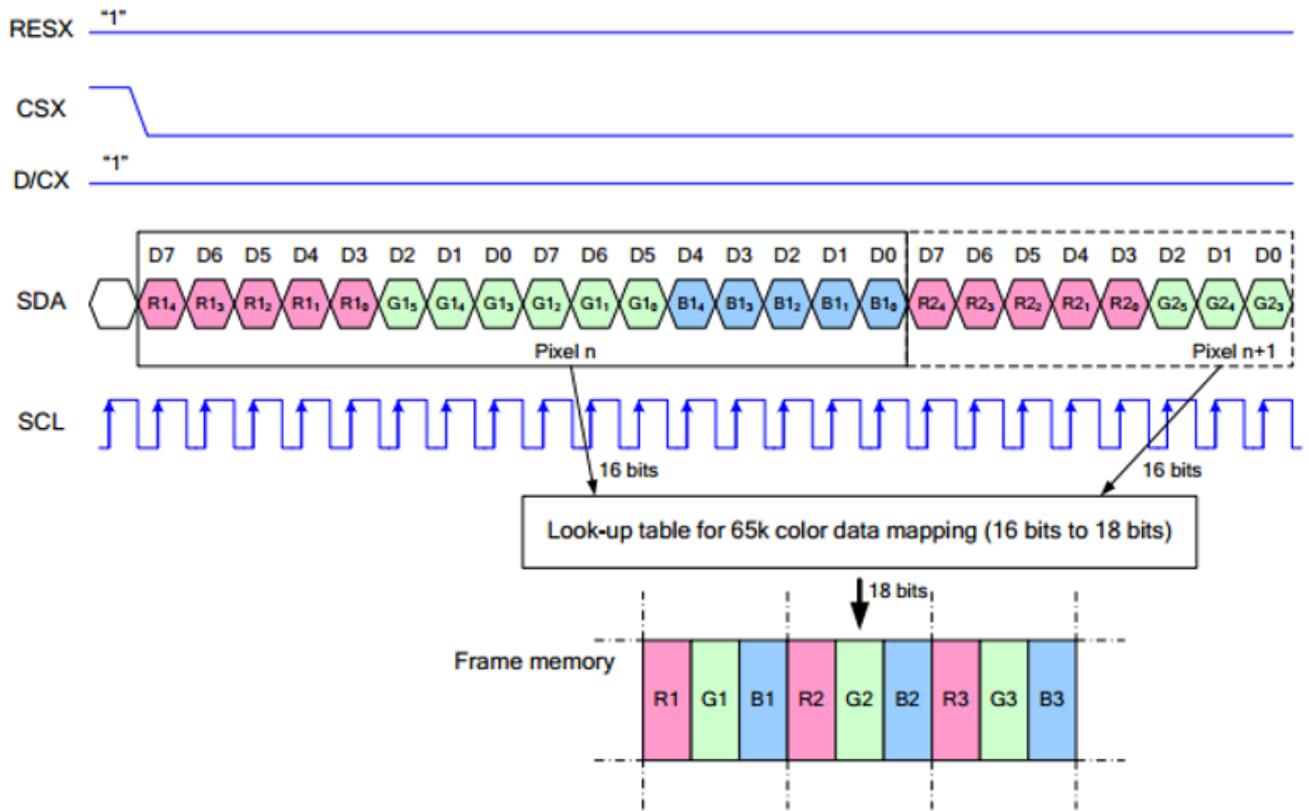
The built-in driver used in this LCD is GC9A01, with a resolution of 240RGB×240 dots, 129,600 bytes of GRAM inside. This LCD supports 12/16/18-bit data bus MCU interface, namely the formats of RGB444, RGB565, RGB666, which are commonly used.

Most of the LCD controllers can debug their communication ways such as by 8080 parallel interfaces, three-wire SPI, four-wire SPI, and so on. This LCD adopts a four-wire SPI communication interface, which can greatly save the GPIO port, and the communication speed will be faster.

- Maybe you have doubts that as the screen is circular, which point is the first pixel? How to determine the coordinates?

- In fact, you can presume that a square screen draws an inscribed circle in it. We only display the content in this inscribed circle, and the pixels beside the square are deleted. The same is true for most of the circular LCDs on the market.

Communication Protocol



(/wiki/File:0.96inch_lcd_module_spi.png)

Note: Different from the traditional SPI protocol: the data line sent from the slave to the host is hidden as the LCD only needs to be displayed. Please refer to the table on Datasheet Page 105.

RESX: reset, it is pulled low when the module is powered on, and it is usually set to 1.

CSX: the slave chip selection and the chip will be enabled only when CS is low.

D/CX: the data/command control pin of the chip, write command when DC = 0, write data when DC = 1

SDA: the transmitted data, that is, RGB data;

SCL: the SPI communication clock.

For SPI communication, data is transmitted with timing, that is, the combination of clock phase (CPHA) and clock polarity (CPOL):

The level of CPHA determines whether the data of the serial synchronization clock is collected on the first clock transition edge or the second clock transition edge. When CPHA = 0, data acquisition is performed on the first transition edge;

The level of CPOL determines the idle state level of the serial synchronous clock. CPOL = 0, which is a low level.

As can be seen from the figure, when the first falling edge of SCLK starts to transmit data, 8bit data is transmitted in one clock cycle, using SPI0, bit-by-bit transmission, high-order first, and low-order at last.

Working with Raspberry Pi

Enable SPI interface

PS: If you are using the system of the Bullseye branch, you need to change "apt-get" to "apt", the system of the Bullseye branch only supports Python3.

- Open terminal, use command to enter the configuration page

```
sudo raspi-config
Choose Interfacing Options -> SP
I -> 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
```

Reboot Raspberry Pi :

```
sudo reboot
```

Would you like the SPI interface to be enabled?

<Yes>

<No>

(/wiki/File:RPI_open_spi.png)

Install Library

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

BCM2835

```
#Open the Raspberry Pi terminal and run the following command
wget http://www.airspayce.com/mikem/bcm2835/bcm2835-1.71.tar.gz
tar zxvf bcm2835-1.71.tar.gz
cd bcm2835-1.71/
sudo ./configure && sudo make && sudo make check && sudo make install
# For more, you can refer to the official website at: http://www.airspayce.com/mikem/bcm2835/
```

WiringPi

```
#Open the Raspberry Pi terminal and run the following command
cd
sudo apt-get install wiringpi
#For Raspberry Pi systems after May 2019 (earlier than that can be executed witho
```

```
ut), an upgrade may be required:  
wget https://project-downloads.drogon.net/wiringpi-latest.deb  
sudo dpkg -i wiringpi-latest.deb  
gpio -v  
# Run gpio -v and version 2.52 will appear, if it doesn't it means there was an i  
nstallation error  
  
# Bullseye branch system using 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 doesn't it means there was an i  
nstallation error
```

lgpio

```
#Open the Raspberry Pi terminal and run the following command  
wget https://github.com/joan2937/lg/archive/master.zip  
unzip master.zip  
cd lg-master  
sudo make install  
  
# You can refer to the official website for more: https://github.com/gpiozero/lg
```

Python

```
sudo apt-get update  
sudo apt-get install python3-pip  
sudo apt-get install python3-pil  
sudo apt-get install python3-numpy  
sudo pip3 install spidev
```

Download Examples

Open Raspberry Pi terminal and run the following command

```
sudo apt-get install unzip -y  
sudo wget https://files.waveshare.com/upload/8/8d/LCD_Module_RPI_code.zip  
sudo unzip ./LCD_Module_RPI_code.zip  
cd LCD_Module_RPI_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
```

- The test program of all screens can be called directly by entering the corresponding size

```
sudo ./main Screen Size
```

Depending on the LCD, one of the following commands should be entered:

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

python

- Enter the python program directory and run the command ls -l

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

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

(/wiki/File:LCD_rpi_python_examples.png)

Test programs for all screens can be viewed, sorted by size:

0inch96_LCD_test.py: 0.96inch LCD test program

1inch14_LCD_test.py: 1.14inch LCD test program

1inch28_LCD_test.py: 1.28inch LCD test program

1inch3_LCD_test.py: 1.3inch LCD test program

1inch47_LCD_test.py: 1.47inch LCD test program

1inch54_LCD_test.py: 1.54inchLCD test program

1inch8_LCD_test.py: 1.8inch LCD test program

2inch_LCD_test.py: 2inch LCD test program

2inch4_LCD_test.py: 2inch4 LCD test program

- Just run the program corresponding to the screen, the program supports python2/3

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

API Description

The RaspberryPi series can share a set of programs, because they are all embedded systems, and the compatibility is relatively strong.

The program is divided into bottom-layer hardware interface, middle-layer LCD screen driver, and upper-layer application;

C

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 WiringPi libraries, please open RaspberryPi\c\Makefile and modify lines 13-15 as follows:

```
13 #USELIB = USE_BCM2835_LIB
14 #USELIB = USE_WIRINGPI_LIB
15 USELIB = USE_DEV_LIB
16 DEBUG = -D $(USELIB)
17 ifeq ($(USELIB), USE_BCM2835_LIB)
18     LIB = -l bcm2835 -lm
19 else ifeq ($(USELIB), USE_WIRINGPI_LIB)
20     LIB = -lwiringPi -lm
21 else ifeq ($(USELIB), USE_DEV_LIB)
22     LIB = -lpthread -lm
23 endif
```

(/wiki/File:RPI_open_spi1.png)

- Data type:

```
#define UBYTE      uint8_t
#define UWORLD     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.

```

- GPIO read and write:

```

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

```

- SPI write data:

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

Upper application

If you need to draw pictures or display Chinese and English characters, we provide some basic functions here about some graphics processing in the directory RaspberryPi\c\lib\GUI\GUI_Paint.c(.h).

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

(/wiki/File:LCD_rpi_GUI.png)

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

(/wiki/File:RPI_open_spi3.png)

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

```
void Paint_NewImage(UBYTE *image, UWORLD Width, UWORLD Height, UWORLD Rotate, UWORLD Color)
Parameters:
    Image: the name of the image buffer, which is actually a pointer to the first 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;
```

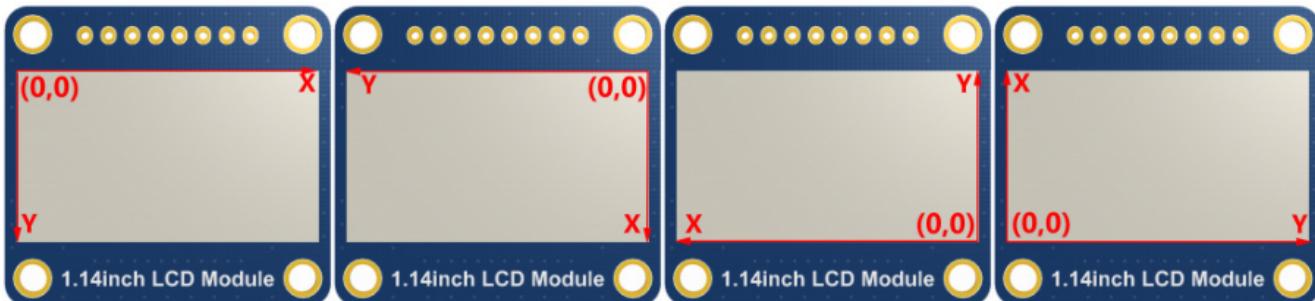
- Select image buffer: The purpose of the selection is that you can create multiple image attributes, there can be multiple images buffer, you can select each image you create.

```
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.



(/wiki/File:Rotation-lcd.png)

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

Parameters:

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

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

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

Parameters:

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

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

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

Parameters:

Xpoint: the X position of a point in the image buffer
Ypoint: 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 refresh the screen into blank, often used for time display, refresh the last second of the screen.

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

Color)

Parameters:

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

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

```
void Paint_DrawPoint(UWORD Xpoint, WORD Ypoint, WORD Color, DOT_PIXEL Dot_Pixel,
    DOT_STYLE Dot_Style)
Parameters:
    Xpoint: indicates the X coordinate of a point.
    Ypoint: indicates the Y coordinate of a point.
    Color: fill Color
    Dot_Pixel: The size of the dot, the demo provides 8 size pointss by default.
```

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

Dot_Style: the 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, the width and the style of the line.

```
void Paint_DrawLine(UWORD Xstart, WORD Ystart, WORD Xend, WORD Yend, WORD Color, 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 by default.
```

```
typedef enum {
    DOT_PIXEL_1X1 = 1,      // 1 x 1
    DOT_PIXEL_2X2 ,
    DOT_PIXEL_3X3 ,
    DOT_PIXEL_4X4 ,
    DOT_PIXEL_5X5 ,
    DOT_PIXEL_6X6 ,
    DOT_PIXEL_7X7 ,
    DOT_PIXEL_8X8 ,
} DOT_PIXEL;
```

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

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

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

```

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

Parameters:
    Xstart: the starting X coordinate of the rectangle
    Ystart: the starting Y coordinate of the rectangle
    Xend: the x-end coordinate of the rectangle
    Yend: the y-end coordinate of the rectangle
    Color: fill Color
    Line_width: The width of the four sides of a rectangle. And the demo provides 8 sizes of width by default.
    typedef enum {
        DOT_PIXEL_1X1 = 1,           // 1 x 1
        DOT_PIXEL_2X2 ,             // 2 x 2
        DOT_PIXEL_3X3 ,             // 3 x 3
        DOT_PIXEL_4X4 ,             // 4 x 4
        DOT_PIXEL_5X5 ,             // 5 x 5
        DOT_PIXEL_6X6 ,             // 6 x 6
        DOT_PIXEL_7X7 ,             // 7 x 7
        DOT_PIXEL_8X8 ,             // 8 x 8
    } DOT_PIXEL;
    Draw_Fill: Fill, whether to fill the inside of the rectangle
    typedef enum {
        DRAW_FILL_EMPTY = 0,
        DRAW_FILL_FULL,
    } DRAW_FILL;

```

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

```

void Paint_DrawCircle(UWORD X_Center, UWORD Y_Center, UWORD Radius, UWORD Color,
DOT_PIXEL Line_width, DRAW_FILL Draw_Fill)
Parameters:
    X_Center: the x-coordinate of the center of 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,      // 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* Font,
t, UWORD Color_Foreground, UWORD Color_Background)
Parameters:
    Xstart: the x-coordinate of the left vertex of a character
    Ystart: the Y-coordinate of the left vertex of a character
    Ascii_Char: indicates the Ascii character
    Font: Ascii visual character library, in the Fonts folder the demo provides the following Fonts:
        Font8: 5*8 font
        Font12: 7*12 font
        Font16: 11*16 font
        Font20: 14*20 font
        Font24: 17*24 font
    Color_Foreground: Font color
    Color_Background: 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, WORD Ystart, const char * pString, sFONT*  
Font, WORD Color_Foreground, WORD 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 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 character font, font foreground color, and font background color of the GB2312 encoding.

```
void Paint_DrawString_CN(UWORD Xstart, WORD Ystart, const char * pString, cFONT*  
font, WORD Color_Foreground, WORD 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 demo 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, WORD Ypoint, double Number, SFONT* Font, WORD  
Digit, WORD Color_Foreground, WORD 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

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

Digit: It's a decimal number

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

Font8: 5*8 font

Font12: 7*12 font

Font16: 11*16 font

Font20: 14*20 font

Font24: 17*24 font

Color_Foreground: Font color

Color_Background: 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(WORD Xstart, WORD Ystart, PAINT_TIME *pTime, SFONT* Font, WORD  
Color_Background, WORD 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 hours, minutes, and seconds are passed to the parameters;

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

Font8: 5*8 font

Font12: 7*12 font

Font16: 11*16 font

Font20: 14*20 font

Font24: 17*24 font

Color_Foreground: Font color

Color_Background: indicates the background color

- Read the local bmp image and write it to the cache.

For Linux operating systems such as Raspberry Pi, you can read and write pictures. For Raspberry Pi, in the directory: RaspberryPi\c\lib\GUI\GUI_BMPfile.c(h).

```

UBYTE GUI_ReadBmp(const char *path, UWORLD Xstart, UWORLD Ystart)
parameter:
    path: the relative path of the BMP image
    Xstart: The X coordinate of the left vertex of the image, generally 0 is passed by default
    Ystart: The Y coordinate of the left vertex of the picture, generally 0 by default

```

Testing Code for Users

For Raspberry Pi, in the directory: RaspberryPi\c\examples, for all the test code;

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

(/wiki/File:LCD_rpi_c_examples%26128.png)

If you need to run the 0.96-inch LCD test program, you need to add 0.96 as a parameter when running the main demo.

Re-execute in Linux command mode as follows:

```

make clean
make
sudo ./main 0.96

```

Python (for Raspberry Pi)

Works with python and python3.

For python, his calls are not as complicated as C.

Raspberry Pi: RaspberryPi\python\lib\

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

(/wiki/File:LCD_rpi_python_lib.png)

lcdconfig.py

- 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 () initializer on the LCD, but the `module_exit()` function needs to be called by itself.

- GPIO read and write:

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

- SPI write data.

```
def spi_writebyte(data)
```

- `xxx_LCD_test.py` (xxx indicates the size, if it is a 0.96inch LCD, it is `0inch96_LCD_test.py`, and so on)

python is in the following directory:

Raspberry Pi: RaspberryPi\python\examples\

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

(/wiki/File:LCD_rpi_python_examples2.png)

If your python version is python2 and you need to run the 0.96inch LCD test program, re-execute it as follows in linux command mode:

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

If your python version is python3 and you need to run the 0.96inch LCD test program, re-execute the following in linux command mode:

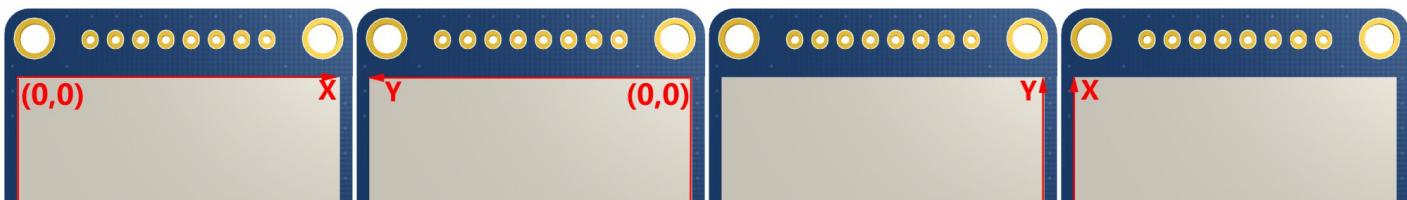
```
sudo python3 0inch96_LCD_test.py
```

About Rotation Settings

If you need to set the screen rotation in the python program, you can set it by the statement `im_r = image1.rotate(270)`.

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

Rotation effect, take 1.54 as an example, the order is 0°, 90°, 180°, 270°



(/wiki/File:LCD_Rotate.jpg)

GUI Functions

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

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

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

And then import the library

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

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

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

```
image1 = Image.new("RGB", (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 a line.

```
draw.line([(20, 10), (70, 60)], fill = "RED", width = 1)
```

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 a 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 a circle.

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

Draw an inscribed circle in the square, the first parameter is a tuple of 4 elements, with (150, 15) as the upper left corner vertex of the square, (190, 55) as the lower right corner vertex of the square, specifying the level median line of the rectangular frame is the angle of 0 degrees, the second parameter indicates the starting angle, the third parameter indicates the ending angle, and fill = 0 indicates that the color of the line is white. If 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 a 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 characters, you can directly use the fonts; for Chinese characters, you need to add a symbol u:

```
draw.text((40, 50), 'WaveShare', fill = (128,255,128),font=Font2)
text= u"微雪电子"
draw.text((74, 150),text, fill = "WHITE",font=Font3)
```

The first parameter is a tuple of 2 elements, with (40, 50) as the left vertex, the font is Font2, and the fill is the font color. You can directly make fill = "WHITE", because the regular color value is already defined Well, of course, you can also use fill = (128,255,128), the parentheses correspond to the values of the three RGB colors so that you can precisely control the color you want. The second sentence shows Micro Snow Electronics, using Font3, the font color is white.

- read local image

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

The parameter is the image path.

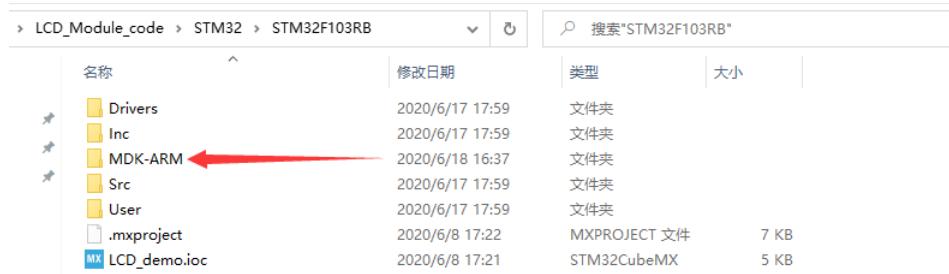
- Other functions.

For more information, you can refer to <http://effbot.org/imagingbook> (<http://effbot.org/imagingbook>) pil

Using with STM32

Software description

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



名称	修改日期	类型	大小
Drivers	2020/6/17 17:59	文件夹	
Inc	2020/6/17 17:59	文件夹	
MDK-ARM	2020/6/18 16:37	文件夹	
Src	2020/6/17 17:59	文件夹	
User	2020/6/17 17:59	文件夹	
.mxproject	2020/6/8 17:22	MXPROJECT 文件	7 KB
LCD_demo.ioc	2020/6/8 17:21	STM32CubeMX	5 KB

(/wiki/File:LCD_STM32_CODE1.png)

- Open main.c, you can see all the test programs, remove the comments in front of the test programs on the corresponding screen, and recompile and download.

(/wiki/File:LCD_STM32_CODE128.png)

LCD_0in96_test() 0.96inch LCD test program
LCD_1in14_test() 1.14inch LCD test program
LCD_1in28_test() 1.28inch LCD test program
LCD_1in3_test() 1.3 inch LCD test program
LCD_1in54_test() 1.54inch LCD test program
LCD_1in8_test() 1.8inch LCD test program
LCD_2in_test() 2inch LCD test program

Program Description

Underlying hardware interface

- 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);
```

The upper application

For the screen, if you need to draw pictures, display Chinese and English characters, display pictures, etc., you can use the upper application to do, and we provide some basic functions here about some graphics processing in the directory STM32\STM32F103RB\User\GUI_DEV\GUI_Paint.c(.h)

Note: Because of the size of the internal RAM of STM32 and arduino, the GUI is directly written to the RAM of the LCD.

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

(/wiki/File:LCD_rpi_GUI.png)

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

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

(/wiki/File:LCD_rpi_Font.png)

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

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

Parameters:

Width: image buffer Width;

Height: the Height of the image buffer;

Rotate: Indicates the rotation Angle of an image

Color: the initial Color of the image;

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

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

parameter:

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

- Set the drawing pixel function

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

parameter:

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

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

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

Parameters:

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

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

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

Parameters:

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

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

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

Parameters:

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

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

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

Parameters:

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

Ypoint: Y position of a point in the image buffer

Color: indicates the Color of the dot

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

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

Parameters:

Color: fill Color

- Image buffer part of the window filling color: the image buffer part of the window filled with a certain color, generally as a window whitewashing function, often used for time display, whitewashing on a second

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

Parameters:

Xstart: the x-starting coordinate of the window

Ystart: indicates the Y starting point of the window

Xend: the x-end coordinate of the window

Yend: indicates the y-end coordinate of the window

Color: fill Color

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

```

void Paint_DrawPoint(UWORD Xpoint, WORD Ypoint, WORD Color, DOT_PIXEL Dot_Pixel,
    DOT_STYLE Dot_Style)
Parameters:
    Xpoint: indicates the X coordinate of a point
    Ypoint: indicates the Y coordinate of a point
    Color: fill Color
    Dot_Pixel: The size of the dot, providing a default of eight size points
        typedef enum {
            DOT_PIXEL_1X1    = 1,           // 1 x 1
            DOT_PIXEL_2X2    ,           // 2 X 2
            DOT_PIXEL_3X3    ,           // 3 X 3
            DOT_PIXEL_4X4    ,           // 4 X 4
            DOT_PIXEL_5X5    ,           // 5 X 5
            DOT_PIXEL_6X6    ,           // 6 X 6
            DOT_PIXEL_7X7    ,           // 7 X 7
            DOT_PIXEL_8X8    ,           // 8 X 8
        } DOT_PIXEL;
    Dot_Style: the size of a point that expands from the center of the point or from the bottom left corner of the point to the right and up
        typedef enum {
            DOT_FILL_AROUND = 1,
            DOT_FILL_RIGHTUP,
        } DOT_STYLE;

```

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

```

void Paint_DrawLine(UWORD Xstart, WORD Ystart, WORD Xend, WORD Yend, WORD Col
or, LINE_STYLE Line_Style , LINE_STYLE Line_Style)

Parameters:
    Xstart: the x-starting coordinate of a line
    Ystart: indicates the Y starting point of a line
    Xend: x-terminus of a line
    Yend: the y-end coordinate of a line
    Color: fill Color
    Line_width: The width of the line, which provides a default of eight widths

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

    Line_Style: line style. Select whether the lines are joined in a straight or
dashed way
    typedef enum {
        LINE_STYLE_SOLID = 0,
        LINE_STYLE_DOTTED,
    } LINE_STYLE;

```

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

```

void Paint_DrawRectangle(UWORD Xstart, WORD Ystart, WORD Xend, WORD Yend, WORD
Color, DOT_PIXEL Line_width, DRAW_FILL Draw_Fill)
Parameters:
    Xstart: the starting X coordinate of the rectangle
    Ystart: indicates the Y starting point of the rectangle
    Xend: X terminus of the rectangle
    Yend: specifies the y-end coordinate of the rectangle
    Color: fill Color
    Line_width: The width of the four sides of a rectangle. Default eight widths are provided
    typedef enum {
        DOT_PIXEL_1X1 = 1,      // 1 x 1
        DOT_PIXEL_2X2 ,         // 2 X 2
        DOT_PIXEL_3X3 ,         // 3 X 3
        DOT_PIXEL_4X4 ,         // 4 X 4
        DOT_PIXEL_5X5 ,         // 5 X 5
        DOT_PIXEL_6X6 ,         // 6 X 6
        DOT_PIXEL_7X7 ,         // 7 X 7
        DOT_PIXEL_8X8 ,         // 8 X 8
    } DOT_PIXEL;
    Draw_Fill: Fill, whether to fill the inside of the rectangle
    typedef enum {
        DRAW_FILL_EMPTY = 0,
        DRAW_FILL_FULL,
    } DRAW_FILL;

```

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

```

void Paint_DrawCircle(UWORD X_Center, UWORD Y_Center, UWORD Radius, UWORD Color,
DOT_PIXEL Line_width, DRAW_FILL Draw_Fill)
Parameters:
    X_Center: the x-coordinate of the center of a circle
    Y_Center: Y coordinate of the center of a circle
    Radius: indicates the Radius of a circle
    Color: fill Color
    Line_width: The width of the arc, with a default of 8 widths
    typedef enum {
        DOT_PIXEL_1X1 = 1,           // 1 x 1
        DOT_PIXEL_2X2 ,             // 2 X 2
        DOT_PIXEL_3X3 ,             // 3 X 3
        DOT_PIXEL_4X4 ,             // 4 X 4
        DOT_PIXEL_5X5 ,             // 5 X 5
        DOT_PIXEL_6X6 ,             // 6 X 6
        DOT_PIXEL_7X7 ,             // 7 X 7
        DOT_PIXEL_8X8 ,             // 8 X 8
    } DOT_PIXEL;
    Draw_Fill: fill, whether to fill the inside of the circle
    typedef enum {
        DRAW_FILL_EMPTY = 0,
        DRAW_FILL_FULL,
    } DRAW_FILL;

```

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

```

void Paint_DrawChar(UWORD Xstart, UWORD Ystart, const char Ascii_Char, sFONT* Font,
t, 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, WORD Ystart, const char * pString, sFONT*
Font, WORD Color_Foreground, WORD 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, WORD Ystart, const char * pString, cFONT*
font, WORD Color_Foreground, WORD Color_Background)
Parameters:
    Xstart: the x-coordinate of the left vertex of a character
    Ystart: the Y coordinate of the font's left vertex
    PString: string, string is a pointer
    Font: GB2312 encoding character Font library, in the Fonts folder provides the 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, WORD Ypoint, double Nummber, SFONT* Font, WORD  
Digit, WORD Color_Foreground, WORD 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, WORD Ystart, PAINT_TIME *pTime, sFONT* Font, WORD Color_Background, WORD 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

```

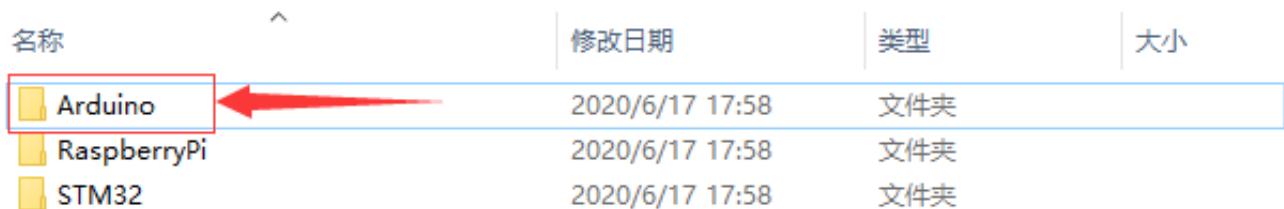
Arduino Software Description

Note: The demos are all tested on Arduino uno. If you need other types of Arduino, you need to determine whether the connected pins are correct.

Template: Arduino IDE Installation Steps (/wiki/Template:Arduino_IDE_Installation_Steps)

Run program

In the product encyclopedia interface download the program (https://files.waveshare.com/upload/e/e9/LCD_Module_code.7z), and then unzip it. The Arduino program is located at `~/Arduino/...`



名称	修改日期	类型	大小
Arduino	2020/6/17 17:58	文件夹	
RaspberryPi	2020/6/17 17:58	文件夹	
STM32	2020/6/17 17:58	文件夹	

(/wiki/File:LCD_arduino_cede1.png)

Please select the corresponding program according to the LCD screen model to open

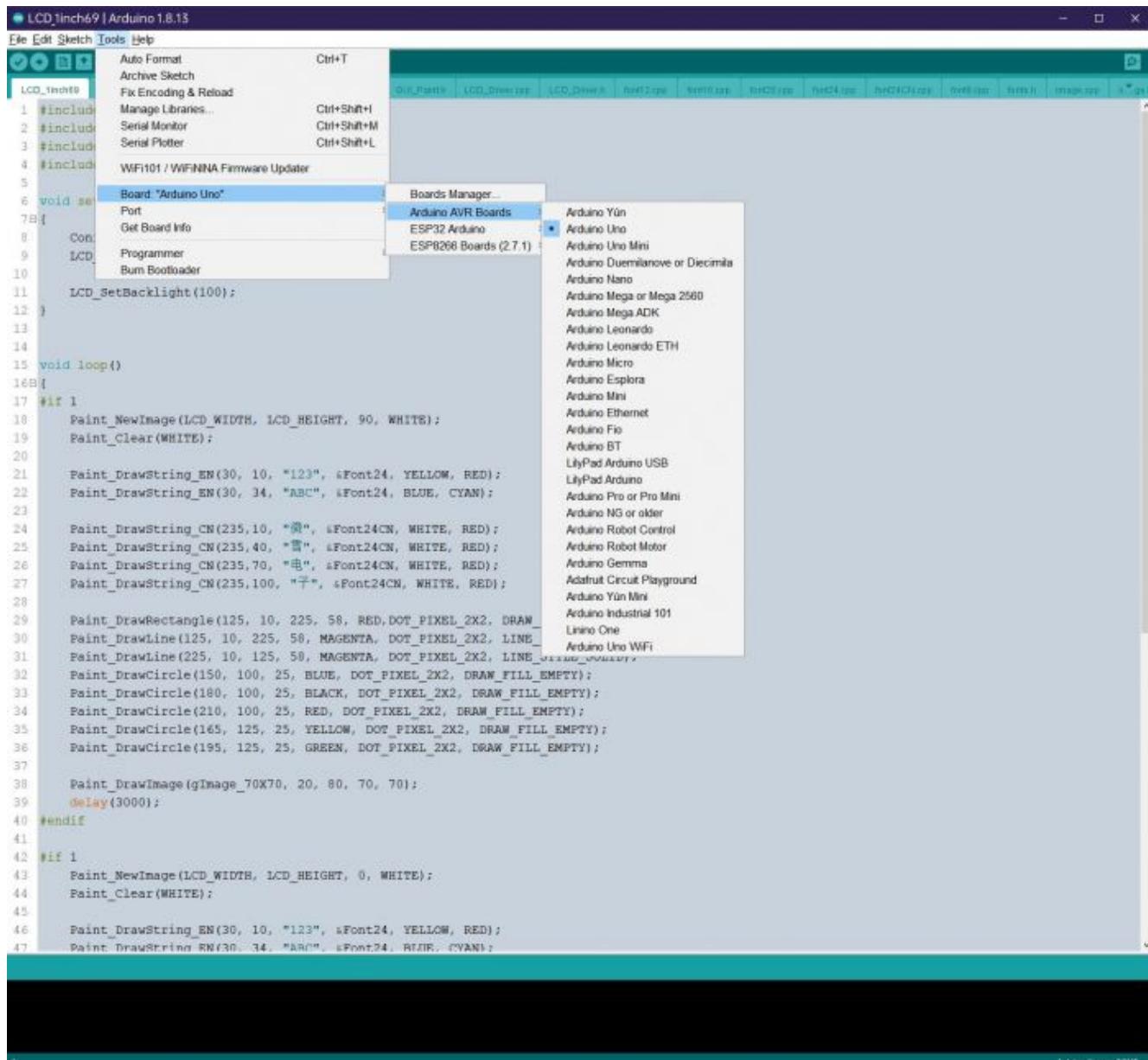
名称	修改日期	类型	大小
LCD_0inch96	2021/2/3 14:44	文件夹	
LCD_1inch3	2021/2/3 14:44	文件夹	
LCD_1inch8	2021/2/3 14:44	文件夹	
LCD_1inch14	2021/2/3 14:44	文件夹	
LCD_1inch28	2021/2/3 14:44	文件夹	
LCD_1inch54	2021/2/3 14:44	文件夹	
LCD_2inch	2021/2/3 14:44	文件夹	
LCD_2inch4	2021/2/3 14:44	文件夹	

(/wiki/File:1.28inch_LCD_Arduino.png)

You can view test programs for all screen sizes, sorted by size:

For example, 1.54inch LCD Module. Open the LCD_1inch54 folder and run the LCD_1inch54.ino file.

Open the program, select the development board model Arduino UNO



(/wiki/File:Arduino_for_1.69inch_lcd_module03.jpg)

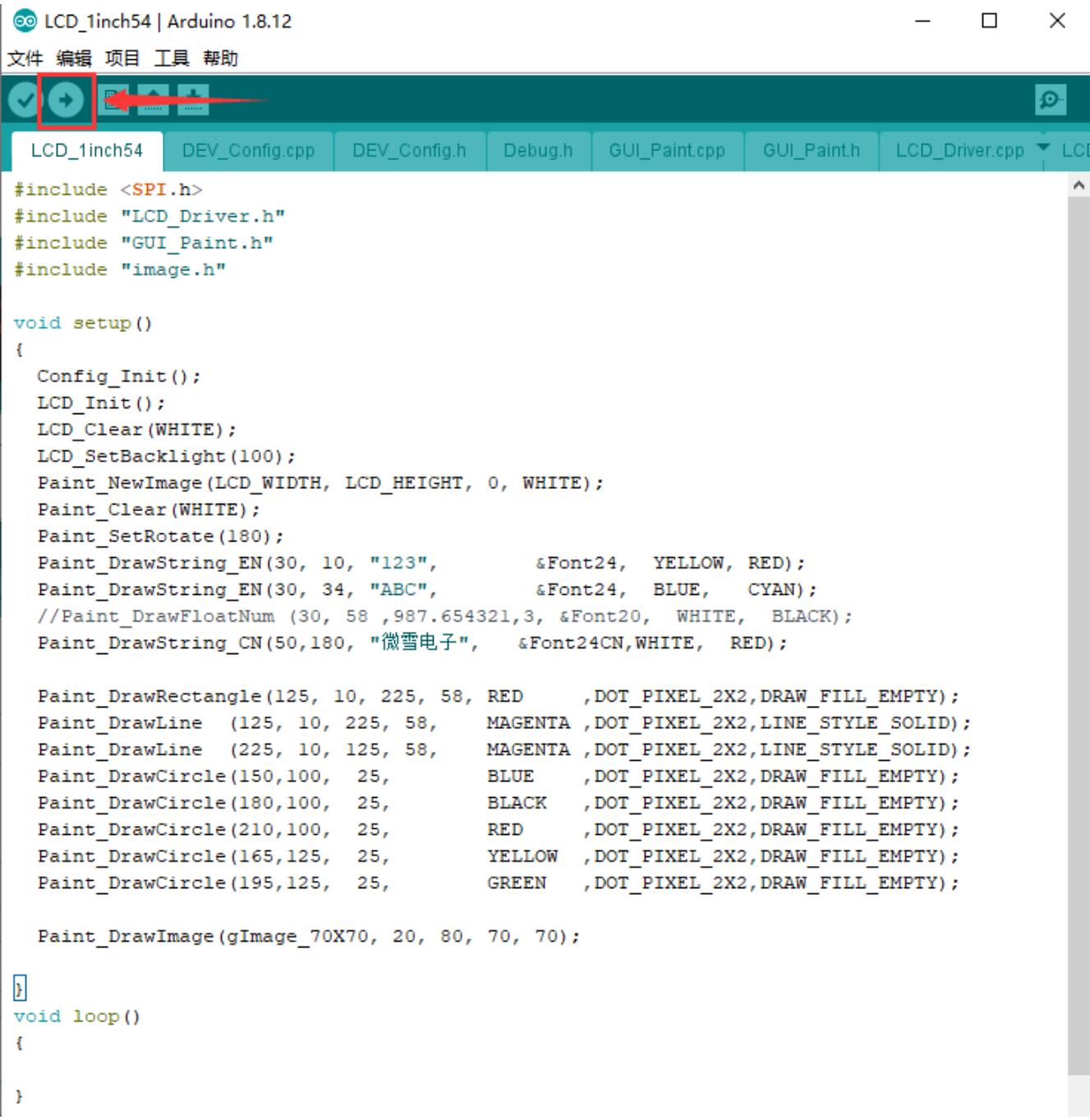
Select the corresponding COM port

The screenshot shows the Arduino IDE interface. The title bar reads "LCD_1inch69 | Arduino 1.8.13". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". The "Tools" menu is open, displaying various options like "Auto Format", "Archive Sketch", and "Serial Monitor". A submenu for "Port" is open, showing "Serial ports", "COM1", and "COM4", with "COM4" highlighted. The main code editor window contains a sketch named "LCD_1inch69" with the following code:

```
1 #include<LiquidCrystal.h>
2 #include<SPI.h>
3 #include<Adafruit_NeoPixel.h>
4 #include<WiFiNINA.h>
5
6 void setup() {
7     // Initialize serial:
8     // Serial.begin(9600);
9     // LCD.begin(16, 2);
10    // LCD.setBacklight(true);
11    LCD_SetBacklight(100);
12 }
13
14 
```

(/wiki/File:Arduino_for_1.69inch_lcd_module04.jpg)

Then click to compile and download



```
#include <SPI.h>
#include "LCD_Driver.h"
#include "GUI_Paint.h"
#include "image.h"

void setup()
{
    Config_Init();
    LCD_Init();
    LCD_Clear(WHITE);
    LCD_SetBacklight(100);
    Paint_NewImage(LCD_WIDTH, LCD_HEIGHT, 0, WHITE);
    Paint_Clear(WHITE);
    Paint_SetRotate(180);
    Paint_DrawString_EN(30, 10, "123", &Font24, YELLOW, RED);
    Paint_DrawString_EN(30, 34, "ABC", &Font24, BLUE, CYAN);
    //Paint_DrawFloatNum (30, 58 ,987.654321,3, &Font20, WHITE, BLACK);
    Paint_DrawString_CN(50,180, "微雪电子", &Font24CN,WHITE, RED);

    Paint_DrawRectangle(125, 10, 225, 58, RED, DOT_PIXEL_2X2,DRAW_FILL_EMPTY);
    Paint.DrawLine (125, 10, 225, 58, MAGENTA, DOT_PIXEL_2X2,LINE_STYLE_SOLID);
    Paint.DrawLine (225, 10, 125, 58, MAGENTA, DOT_PIXEL_2X2,LINE_STYLE_SOLID);
    Paint_DrawCircle(150,100, 25, BLUE, DOT_PIXEL_2X2,DRAW_FILL_EMPTY);
    Paint_DrawCircle(180,100, 25, BLACK, DOT_PIXEL_2X2,DRAW_FILL_EMPTY);
    Paint_DrawCircle(210,100, 25, RED, DOT_PIXEL_2X2,DRAW_FILL_EMPTY);
    Paint_DrawCircle(165,125, 25, YELLOW, DOT_PIXEL_2X2,DRAW_FILL_EMPTY);
    Paint_DrawCircle(195,125, 25, GREEN, DOT_PIXEL_2X2,DRAW_FILL_EMPTY);

    Paint_DrawImage(gImage_70X70, 20, 80, 70, 70);
}

void loop()
{}
```

(/wiki/File:LCD_arduino_cede5.png)

Program Description

Document Introduction

Take Arduino UNO controlling a 1.54-inch LCD as an example, open the Arduino\LCD_1inch54 directory:

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

(/wiki/File:LCD_arduino_ide_codeDescription1.png)

Of which:

LCD_1inch54.ino: open with Arduino IDE;

LCD_Driver.cpp(.h): is the driver of the LCD screen;

DEV_Config.cpp(.h): It is the hardware interface definition, which encapsulates the read and write pin levels, SPI transmission data, and pin initialization;

font8.cpp, font12.cpp, font16.cpp, font20.cpp, font24.cpp, font24CN.cpp, fonts.h: fonts for characters of different sizes;

image.cpp(.h): is the image data, which can convert any BMP image into a 16-bit true color image array through Img2Lcd (downloadable in the development data).

The program is divided into bottom-layer hardware interface, middle-layer LCD screen driver, and upper-layer application;

Underlying hardware interface

The hardware interface is defined in the two files DEV_Config.cpp(.h), and functions such as read and write pin level, delay, and SPI transmission are encapsulated.

- write pin level

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

The first parameter is the pin, and the second is the high and low level.

- Read pin level

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

The parameter is the pin, and the return value is the level of the read pin.

- Delay

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

millisecond level delay.

- SPI output data

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

The parameter is char type, occupying 8 bits.

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 GUI_Paint.c.h)

Note: Because of the size of the internal RAM of STM32 and arduino, the GUI is directly written to the RAM of the LCD.

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

(/wiki/File:LCD_arduino_ide_codeDescription_GUI.png)

The fonts used by the GUI all depend on the font*.cpp(h) files under the same file

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

(/wiki/File:LCD_arduino_ide_codeDescription_font.png)

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

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

Parameters:

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

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

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

parameter:

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

- Set the drawing pixel function.

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

parameter:

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

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

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

Parameters:

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

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

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

Parameters:

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

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

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

Parameters:

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

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

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

Parameters:

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

Ypoint: Y position of a point in the image buffer

Color: indicates the Color of the dot

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

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

Parameters:

Xstart: the x-starting coordinate of the window

Ystart: indicates the Y starting point of the window

Xend: the x-end coordinate of the window

Yend: indicates the y-end coordinate of the window

Color: fill Color

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

```

void Paint_DrawPoint(UWORD Xpoint, WORD Ypoint, WORD Color, DOT_PIXEL Dot_Pixe
l, DOT_STYLE Dot_Style)
Parameters:
    Xpoint: indicates the X coordinate of a point
    Ypoint: indicates the Y coordinate of a point
    Color: fill Color
    Dot_Pixel: The size of the dot, providing a default of eight size points
        typedef enum {
            DOT_PIXEL_1X1 = 1,                      // 1 x 1
            DOT_PIXEL_2X2 ,                         // 2 X 2
            DOT_PIXEL_3X3 ,                         // 3 X 3
            DOT_PIXEL_4X4 ,                         // 4 X 4
            DOT_PIXEL_5X5 ,                         // 5 X 5
            DOT_PIXEL_6X6 ,                         // 6 X 6
            DOT_PIXEL_7X7 ,                         // 7 X 7
            DOT_PIXEL_8X8 ,                         // 8 X 8
        } DOT_PIXEL;
    Dot_Style: the size of a point that expands from the center of the point or f
rom 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, WORD Ystart, WORD Xend, WORD Yend, WORD Col
or, 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 width
hs
    typedef enum {
        DOT_PIXEL_1X1 = 1,           // 1 x 1
        DOT_PIXEL_2X2 ,             // 2 X 2
        DOT_PIXEL_3X3 ,             // 3 X 3
        DOT_PIXEL_4X4 ,             // 4 X 4
        DOT_PIXEL_5X5 ,             // 5 X 5
        DOT_PIXEL_6X6 ,             // 6 X 6
        DOT_PIXEL_7X7 ,             // 7 X 7
        DOT_PIXEL_8X8 ,             // 8 X 8
    } DOT_PIXEL;
    Line_Style: line style. Select whether the lines are joined in a straight
or dashed way
    typedef enum {
        LINE_STYLE_SOLID = 0,
        LINE_STYLE_DOTTED,
    } LINE_STYLE;

```

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

```

void Paint_DrawRectangle(UWORD Xstart, WORD Ystart, WORD Xend, WORD Yend, WORD
Color, DOT_PIXEL Line_width, DRAW_FILL Draw_Fill)
Parameters:
    Xstart: the starting X coordinate of the rectangle
    Ystart: indicates the Y starting point of the rectangle
    Xend: X terminus of the rectangle
    Yend: specifies the y-end coordinate of the rectangle
    Color: fill Color
    Line_width: The width of the four sides of a rectangle. Default eight widths are provided
    typedef enum {
        DOT_PIXEL_1X1 = 1,                      // 1 x 1
        DOT_PIXEL_2X2 ,                         // 2 X 2
        DOT_PIXEL_3X3 ,                         // 3 X 3
        DOT_PIXEL_4X4 ,                         // 4 X 4
        DOT_PIXEL_5X5 ,                         // 5 X 5
        DOT_PIXEL_6X6 ,                         // 6 X 6
        DOT_PIXEL_7X7 ,                         // 7 X 7
        DOT_PIXEL_8X8 ,                         // 8 X 8
    } DOT_PIXEL;
    Draw_Fill: Fill, whether to fill the inside of the rectangle
    typedef enum {
        DRAW_FILL_EMPTY = 0,
        DRAW_FILL_FULL,
    } DRAW_FILL;

```

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

```

void Paint_DrawCircle(UWORD X_Center, UWORD Y_Center, UWORD Radius, UWORD Color,
DOT_PIXEL Line_width, DRAW_FILL Draw_Fill)
Parameters:
    X_Center: the x-coordinate of the center of a circle
    Y_Center: Y coordinate of the center of a circle
    Radius: indicates the Radius of a circle
    Color: fill Color
    Line_width: The width of the arc, with a default of 8 widths
    typedef enum {
        DOT_PIXEL_1X1 = 1,                      // 1 x 1
        DOT_PIXEL_2X2 ,                         // 2 X 2
        DOT_PIXEL_3X3 ,                         // 3 X 3
        DOT_PIXEL_4X4 ,                         // 4 X 4
        DOT_PIXEL_5X5 ,                         // 5 X 5
        DOT_PIXEL_6X6 ,                         // 6 X 6
        DOT_PIXEL_7X7 ,                         // 7 X 7
        DOT_PIXEL_8X8 ,                         // 8 X 8
    } DOT_PIXEL;
    Draw_Fill: fill, whether to fill the inside of the circle
    typedef enum {
        DRAW_FILL_EMPTY = 0,
        DRAW_FILL_FULL,
    } DRAW_FILL;

```

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

```

void Paint_DrawChar(UWORD Xstart, UWORD Ystart, const char Ascii_Char, sFONT* Font,
t, UWORD Color_Foreground, UWORD Color_Background)
Parameters:
    Xstart: the x-coordinate of the left vertex of a character
    Ystart: the Y coordinate of the font's left vertex
    Ascii_Char: indicates the Ascii character
    Font: Ascii visual character library, in the Fonts folder provides the following Fonts:
        Font8: 5*8 font
        Font12: 7*12 font
        Font16: 11*16 font
        Font20: 14*20 font
        Font24: 17*24 font
    Color_Foreground: Font color
    Color_Background: indicates the background color

```

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

```

void Paint_DrawString_EN(UWORD Xstart, UWORD Ystart, const char * pString, sFONT*
Font, UWORD Color_Foreground, UWORD Color_Background)
Parameters:
    Xstart: the x-coordinate of the left vertex of a character
    Ystart: the Y coordinate of the font's left vertex
    PString: string, string is a pointer
    Font: Ascii visual character library, in the Fonts folder provides the fo
llowing Fonts:
        Font8: 5*8 font
        Font12: 7*12 font
        Font16: 11*16 font
        Font20: 14*20 font
        Font24: 17*24 font
    Color_Foreground: Font color
    Color_Background: indicates the background color

```

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

```

void Paint_DrawString_CN(UWORD Xstart, UWORD Ystart, const char * pString, cFONT*
font, UWORD Color_Foreground, UWORD Color_Background)
Parameters:
    Xstart: the x-coordinate of the left vertex of a character
    Ystart: the Y coordinate of the font's left vertex
    PString: string, string is a pointer
    Font: GB2312 encoding character Font library, in the Fonts folder provide
s 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, WORD Ypoint, double Number, SFONT* Font, WORD
Digit, WORD Color_Foreground, WORD 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
    Number: 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 fo
llowing 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 numbers with decimals: at (Xstart Ystart) as the left vertex, write a string of numbers with decimals, you can choose Ascii code visual character font, font foreground color, font background color

```

void Paint_DrawFloatNum(WORD Xpoint, WORD Ypoint, double Number, BYTE Decimal
_Point, SFONT* Font, WORD Color_Foreground, WORD Color_Background);
parameter:
    Xstart: the X coordinate of the left vertex of the character
    Ystart: Y coordinate of the left vertex of the font
    Number: the displayed number, which is saved in double type here
    Decimal_Point: Displays the number of digits after the decimal point
    Font: Ascii code visual character font library, the following fonts are
provided in the Fonts folder:
        Font8: 5*8 font
        Font12: 7*12 font
        Font16: 11*16 font
        Font20: 14*20 font
        Font24: 17*24 font
    Color_Foreground: font color
    Color_Background: background color

```

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

```

void Paint_DrawTime(UWORD Xstart, WORD Ystart, PAINT_TIME *pTime, sFONT* Font, WORD Color_Background, WORD 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

```

- Display image: at (Xstart Ystart) as the left vertex, display an image whose width is W_Image and height is H_Image;

```

void Paint_DrawImage(const unsigned char *image, WORD xStart, WORD yStart, WORD W_Image, WORD H_Image)
parameter:
    image: image address, pointing to the image information you want to display
    Xstart: the X coordinate of the left vertex of the character
    Ystart: Y coordinate of the left vertex of the font
    W_Image: Image width
    H_Image: Image height

```

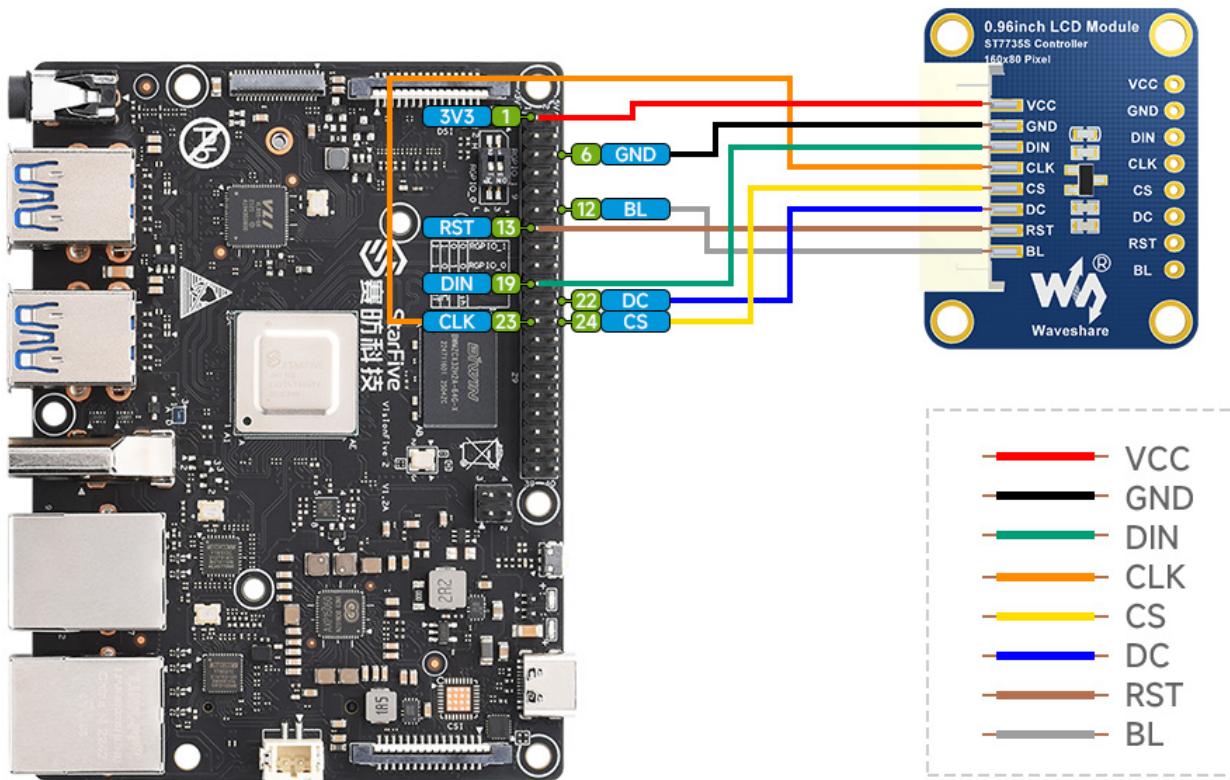
VisionFive2

Adaptive Model

- 0.96inch LCD Module (<https://www.waveshare.com/0.96inch-lcd-module.htm>)
- 1.14inch LCD Module (<https://www.waveshare.com/1.14inch-lcd-module.htm>)
- 1.28inch LCD Module (<https://www.waveshare.com/1.28inch-lcd-module.htm>)
- 1.3inch LCD Module (<https://www.waveshare.com/1.3inch-lcd-module.htm>)
- 1.54inch LCD Module (<https://www.waveshare.com/1.54inch-lcd-module.htm>)
- 1.8inch LCD Module (<https://www.waveshare.com/1.8inch-lcd-module.htm>)
- 2inch LCD Module (<https://www.waveshare.com/2inch-lcd-module.htm>)

- 2.4inch LCD Module (<https://www.waveshare.com/2.4inch-lcd-module.htm>)

Hardware Connection



(/wiki/File:LCD_VisionFive201.jpg)

VisionFive2 Pin Connection

LCD	VisionFive2 Board Pin No.
VCC	3.3V
GND	GND
DIN	19
CLK	23
CS	24
DC	22
RST	13
BL	12

Install Corresponding Libraries

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

Demo Download

```
apt-get install p7zip-full  
wget https://files.waveshare.com/upload/e/e9/LCD_Module_code.7z  
7z x LCD_Module_code.7z -o./LCD_Module_code  
cd LCD_Module_code/VisionFive/python/example/
```

Run the Corresponding Demo According to the Screen You Purchased

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

Resources

Documents

- Schematic (https://files.waveshare.com/upload/b/b4/1.28inch_LCD_Module_SchDoc.pdf)
- GC9A01A manual (<https://files.waveshare.com/upload/5/5e/GC9A01A.pdf>)

Software

- lcd (<https://files.waveshare.com/upload/7/78/LcmZimo.zip>)
- Image2Lcd (<https://files.waveshare.com/upload/b/bd/Image2Lcd2.9.zip>)

Demo Codes

- Sample demo (https://files.waveshare.com/upload/8/8d/LCD_Module_code.zip)

3D Drawing

- 1.28inch LCD Module 3D Drawing (https://files.waveshare.com/upload/4/49/1.28inch_LCD_Module_3D_Drawing.zip)

FAQ

Question: What is the maximum power consumption of the 1.28-inch LCD Module?

Answer:

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)

[Submit Now \(https://service.waveshare.com/\)](https://service.waveshare.com/)

*Retrieved from "https://www.waveshare.com/w/index.php?title=1.28inch_LCD_Module&oldid=75051
[https://www.waveshare.com/w/index.php?title=1.28inch_LCD_Module&oldid=75051"](https://www.waveshare.com/w/index.php?title=1.28inch_LCD_Module&oldid=75051)"*