Overview

1.69inch LCD Module

1.69inch, 240 x 280, SPI

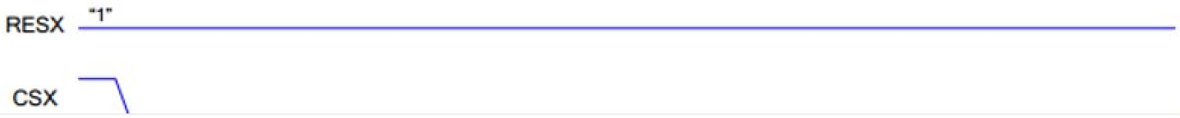
Specifications

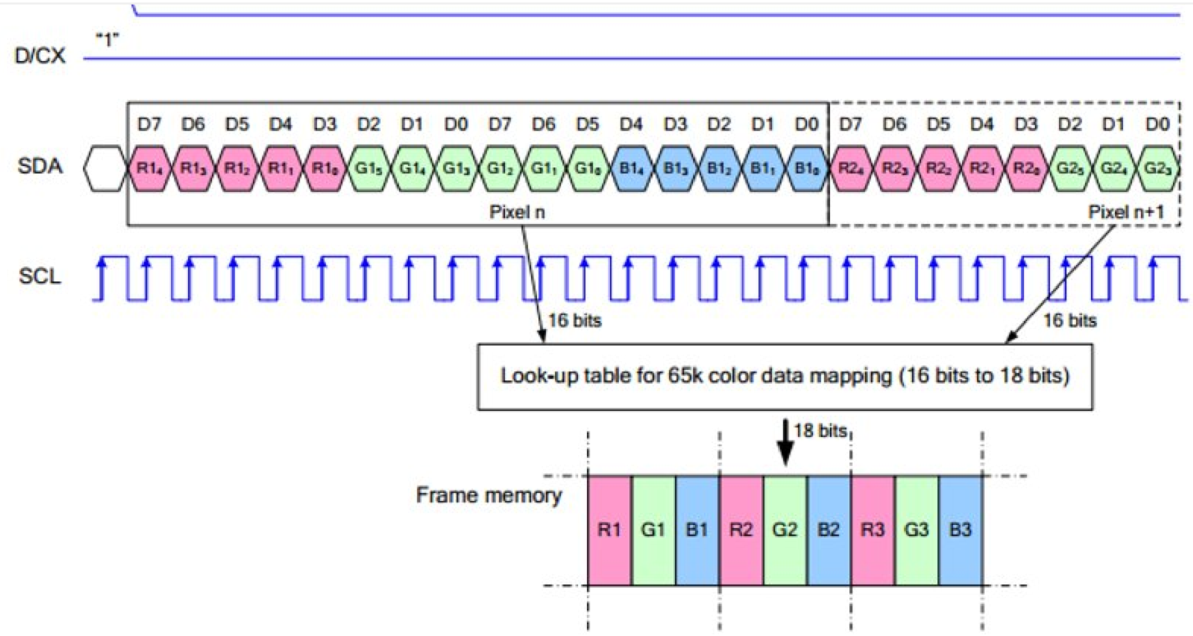
- Operating voltage: 3.3V/5V **(Please ensure that the power supply voltage and logic voltage are the same, otherwise it will not work properly.)**
- Communication interface: SPI
- Display Panel: IPS
- Control chip: ST7789V2
- Resolution: 240 (H)RGB × 280(V)
- Display size: 27.972mm × 32.634mm
- Display R angle: 4-R5 (mm)
- Pixel pitch: 0.11655mm × 0.11655mm
- Dimensions: 31.5mm × 39.0mm

Module & Control Chip

- The embedded controller used in this module is ST7789V2, an LCD controller with 240 × RGB × 320 pixels, and the pixels of the LCD itself is 240(H) × RGB × 280(V), so the internal RAM of the LCD is not fully used.
- This LCD supports input color formats of 12-bit, 16-bit, and 18-bit per pixel, which are RGB444, RGB565, and RGB666. This demo example uses the RGB565 color format, which is also a commonly used RGB format.
- The LCD uses a four-wire SPI communication interface, which greatly saves GPIO ports with faster communication speeds.
- TThe resolution of this module is 240 (H) x RGB x 280 (V), but due to the four round corners (see specifications for dimensions), some parts of the input images may not be displayed.

Communication Protocol




D/CX
"1"
SDA
D7 D6 D5 D4 D3 D2 D1 D0 D7 D6 D5 D4 D3 D2 D1 D0 D7 D6 D5 D4 D3 D2 D1 D0
R1₄ R1₃ R1₂ R1₁ R1₀ G1₅ G1₄ G1₃ G1₂ G1₁ G1₀ B1₄ B1₃ B1₂ B1₁ B1₀ R2₄ R2₃ R2₂ R2₁ R2₀ G2₅ G2₄ G2₃
Pixel n
Pixel n+1
SCL
16 bits
16 bits
Look-up table for 65k color data mapping (16 bits to 18 bits)
18 bits
Frame memory
R1 G1 B1 R2 G2 B2 R3 G3 B3

Note: The difference from the traditional SPI protocol is that the data line sent from the slave to the host is hidden because it only needs to be displayed. Please refer to Datasheet Page 66 for the table.
RESX is reset, it is pulled low when the module is powered on, usually set to 1;
CSX is the slave chip select, and the chip will be enabled only when CS is low.
D/CX is the data/command control pin of the chip, when DC = 0, write command, when DC = 1, write data.
SDA is the transmitted data, that is, RGB data;
SCL is 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 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, 8-bit data is transmitted in one clock cycle, using SPI0, bit-by-bit transmission, high-order first, and low-order last.

Raspberry Pi

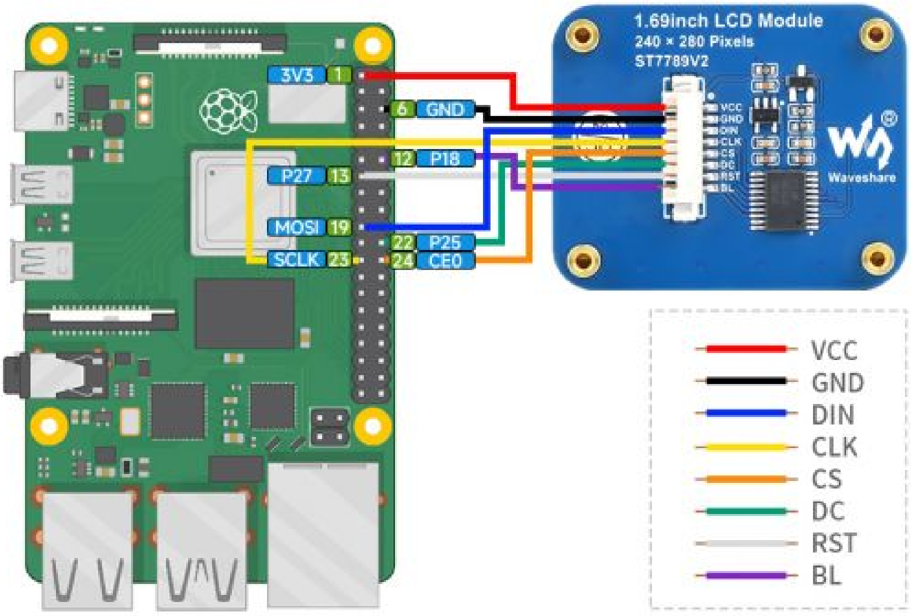
Hardware Connection

When connecting to the Raspberry Pi, you can choose the 8PIN cable to connect according to the following table:

Raspberry Pi Connection Pin Correspondence

LCD	Raspberry Pi	
	BCM2835	Board
VCC	3.3V	3.3V
GND	GND	GND
DIN	MOSI	19
CLK	SCLK	23
CS	CE0	24
DC	25	22
RST	27	13
BL	18	12

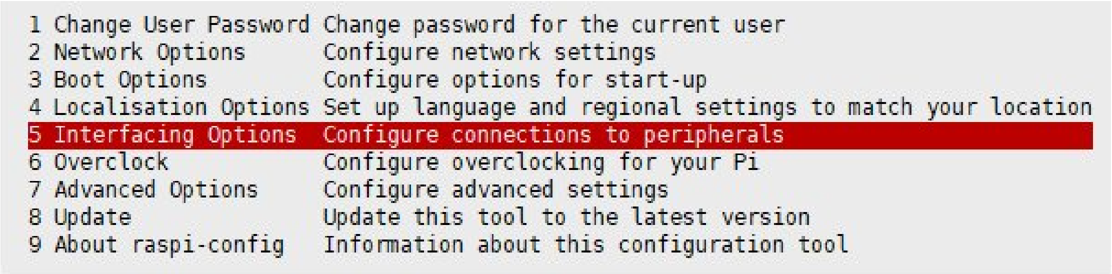
The 1.69inch LCD uses the GH1.25 8PIN interface, which can be connected to the Raspberry Pi according to the above table: (Please connect according to the pin definition table. The color of the wiring in the picture is for reference only, and the actual color shall prevail.)



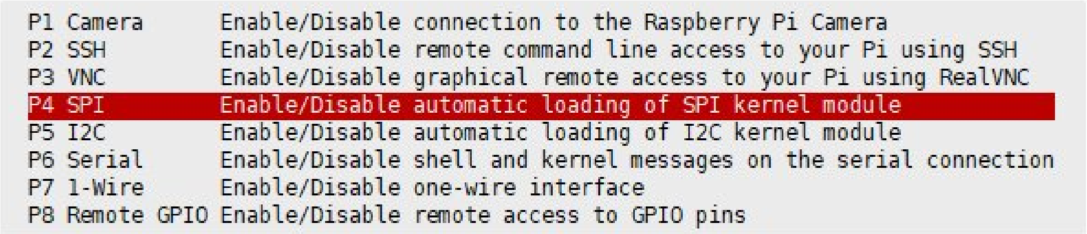
Enable SPI Interface

- Open the terminal, use the command to enter the configuration page:

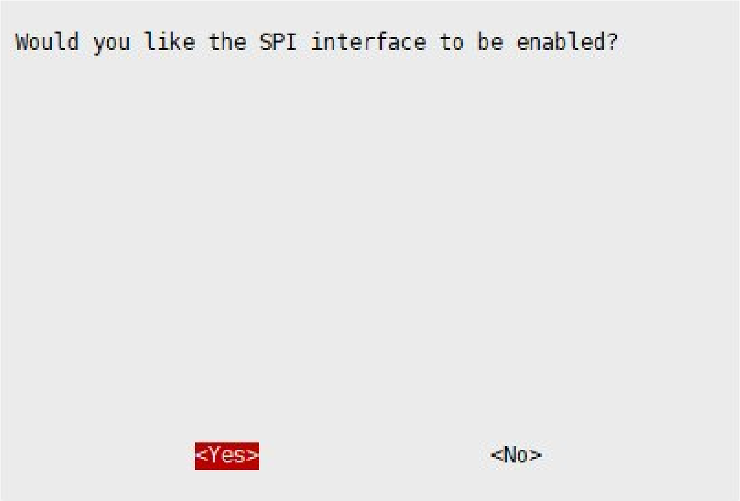
```
sudo raspi-config
Choose Interfacing Options -> SPI -> Yes  to enable SPI interface
```



```
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
```

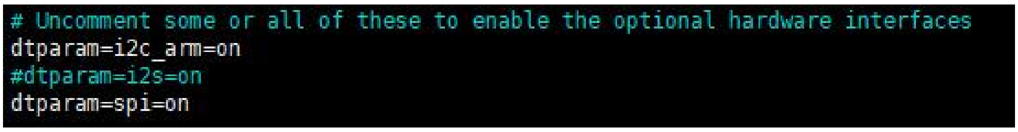


```
Would you like the SPI interface to be enabled?
```

Reboot Raspberry Pi:

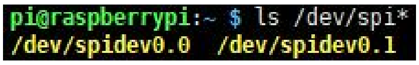
```
sudo reboot
```

- Check /boot/config.txt and you can see that 'dtparam=spi=on' has been written.



```
# Uncomment some or all of these to enable the optional hardware interfaces
dtparam=i2c_arm=on
#dtparam=i2s=on
dtparam=spi=on
```

- To make sure that the SPI is not occupied, it is recommended that other driver overrides be turned off for now. You can use ls /dev/spi* to check the SPI occupancy. The terminal output /dev/spidev0.0 and /dev/spidev0.1 indicates that the SPI status is normal.



```
pi@raspberrypi:~ $ ls /dev/spi*
/dev/spidev0.0  /dev/spidev0.1
```

Running C Demo

- Install 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 information, please refer to the official website: http://www.airspayce.c
om/mikem/bcm2835/
```

- Indtall wiringPi

```
# Open the Raspberry Pi terminal and run the following command:
sudo apt-get install wiringpi
#For Raspberry Pi systems after May 2019 (earlier than that can be executed without), 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 does not, there was an installation 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 2.60 will appear, if it doesn't, there is an installation error.
```

- Download the Demo

```
sudo apt-get install unzip -y
sudo wget https://files.waveshare.com/upload/f/fc/LCD_Module_code.zip
sudo unzip LCD_Module_code.zip -d ./LCD_Module_code
cd LCD_Module_code/RaspberryPi/
```

- Recompile, the compilation process may take a few seconds:

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

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

```
sudo ./main 1.69
```

Run Python Demo

- Install library:

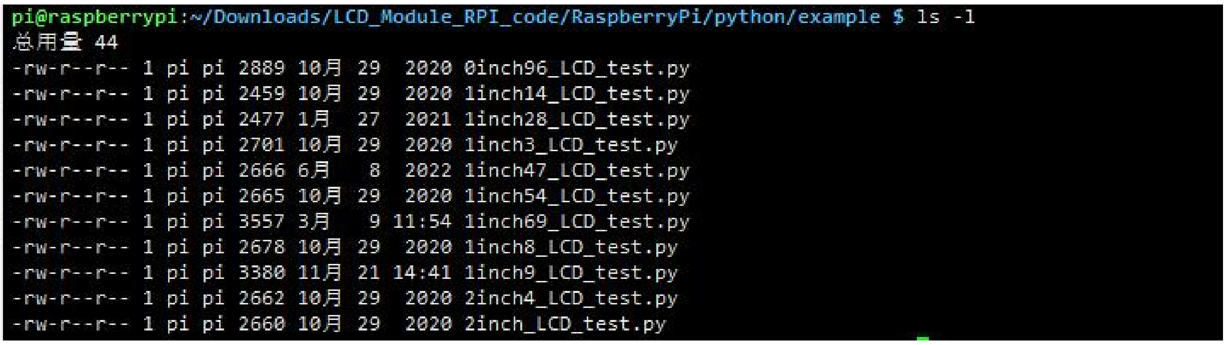
```
#python2
sudo apt-get update
sudo apt-get install python-pip
sudo apt-get install python-pil
sudo apt-get install python-numpy
sudo pip install RPi.GPIO
sudo pip install spidev
#python3
sudo apt-get update
sudo apt-get install python3-pip
sudo apt-get install python3-pil
sudo apt-get install python3-numpy
sudo pip3 install RPi.GPIO
sudo pip3 install spidev
```

- Download the demo

```
sudo apt-get install unzip -y
sudo wget https://files.waveshare.com/upload/f/fc/LCD_Module_code.zip
sudo unzip LCD_Module_code.zip -d ./LCD_Module_code
cd LCD_Module_code/RaspberryPi/
```

- Enter the python demo directory and run the "ls -l" commands:

```
cd python/example
ls -l
```



```
pi@raspberrypi:~/Downloads/LCD_Module_RPI_code/RaspberryPi/python/example $ ls -l
总用量 44
-rw-r--r-- 1 pi pi 2889 10月 29  2020 0inch96_LCD_test.py
-rw-r--r-- 1 pi pi 2459 10月 29  2020 1inch14_LCD_test.py
-rw-r--r-- 1 pi pi 2477 1月  27  2021 1inch28_LCD_test.py
-rw-r--r-- 1 pi pi 2701 10月 29  2020 1inch3_LCD_test.py
-rw-r--r-- 1 pi pi 2666 6月   8  2022 1inch47_LCD_test.py
-rw-r--r-- 1 pi pi 2665 10月 29  2020 1inch54_LCD_test.py
-rw-r--r-- 1 pi pi 3557 3月   9 11:54 1inch69_LCD_test.py
-rw-r--r-- 1 pi pi 2678 10月 29  2020 1inch8_LCD_test.py
-rw-r--r-- 1 pi pi 3380 11月 21 14:41 1inch9_LCD_test.py
-rw-r--r-- 1 pi pi 2662 10月 29  2020 2inch4_LCD_test.py
-rw-r--r-- 1 pi pi 2660 10月 29  2020 2inch_LCD_test.py
```

All test demos for screens can be viewed, sorted by size:

```
0inch96_LCD_test.py 0.96inch LCD test demo
1inch14_LCD_test.py 1.14inch LCD test demo
1inch28_LCD_test.py 1.28inch LCD test demo
1inch3_LCD_test.py 1.3inch LCD test demo
1inch47_LCD_test.py 1.47inch LCD test demo
1inch54_LCD_test.py 1.54inch LCD test demo
1inch69_LCD_test.py 1.69inch LCD test demo
1inch8_LCD_test.py 1.8inch LCD test demo
1inch9_LCD_test.py 1.9inch LCD test demo
2inch_LCD_test.py 2inch LCD test demo
2inch4_LCD_test.py 2.4inch LCD test demo
```

Just run the demo corresponding to the screen, the demo supports python2/3.

```
# python2
sudo python 1inch69_LCD_test.py
```

```
# python3
sudo python3 1inch69_LCD_test.py
```

FBCP Porting

Framebuffer uses a video output device to drive a video display device from a memory buffer containing complete frame data. Simply put, a memory area is used to store the display content, and the display content can be changed by changing the data in the memory.
There is an open source project on github: fbcp-ili9341. Compared with other fbcp projects, this project uses partial refresh and DMA to achieve a speed of up to 60fps.

Download Drivers

```
sudo apt-get install cmake -y
cd ~
wget https://files.waveshare.com/upload/1/18/Waveshare_fbcp.zip
unzip Waveshare_fbcp.zip
cd Waveshare_fbcp/
sudo chmod +x ./shell/*
```

Method 1: Use a script (recommended)

Here we have written several scripts that allow users to quickly use fbcp and run corresponding commands according to their own screen
If you use a script and do not need to modify it, you can ignore the second method below.
Note: The script will replace the corresponding /boot/config.txt and /etc/rc.local and restart, if the user needs, please back up the relevant files in advance.

```
#0.96inch LCD Module
sudo ./shell/waveshare-0inch96
#1.14inch LCD Module
sudo ./shell/waveshare-1inch14
#1.3inch LCD Module
sudo ./shell/waveshare-1inch3
#1.47inch LCD Module
sudo ./shell/waveshare-1inch47
#1.54inch LCD Module
sudo ./shell/waveshare-1inch54
#1.69inch LCD Module
sudo ./shell/waveshare-1inch69
#1.8inch LCD Module
sudo ./shell/waveshare-1inch8
#1.9inch LCD Module
sudo ./shell/waveshare-1inch9
```

```
#2inch LCD Module
sudo ./shell/waveshare-2inch
#2.4inch LCD Module
sudo ./shell/waveshare-2inch4
```

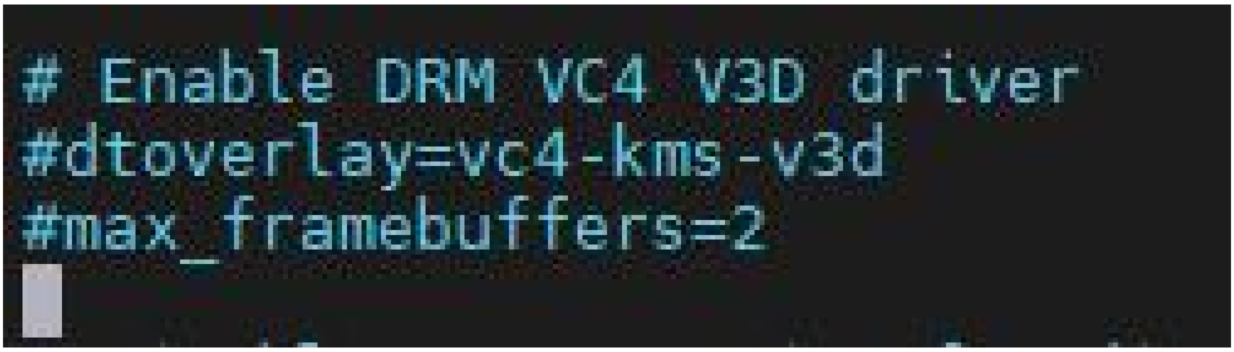
Method 2: Manual Configuration

Environment Configuration

Raspberry Pi's vc4-kms-v3d will cause fbcp to fail, so we need to close vc4-kms-v3d before installing it in fbcp.

```
sudo nano /boot/config.txt
```

Just block the statement corresponding to the picture below.



```
# Enable DRM VC4 V3D driver
#dtoverlay=vc4-kms-v3d
#max_framebuffers=2
```

A reboot is then required.

```
sudo reboot
```

Compile and Run

```
mkdir build
cd build
cmake [options] ..
sudo make -j
sudo ./fbcp
```

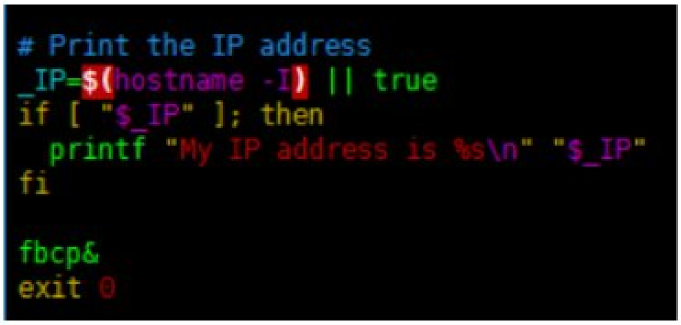
Replace it by yourself according to the LCD Module you use, above cmake [options] ..

```
#0.96inch LCD Module
sudo cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_0INCH96_LCD=ON -DBACKLIGHT_CONTROL
=ON -DSTATISTICS=0 ..
#1.14inch LCD Module
sudo cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_1INCH14_LCD=ON -DBACKLIGHT_CONTROL
=ON -DSTATISTICS=0 ..
#1.3inch LCD Module
sudo cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_1INCH3_LCD=ON -DBACKLIGHT_CONTROL=
ON -DSTATISTICS=0 ..
#1.47inch LCD Module
sudo cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_1INCH47_LCD=ON -DBACKLIGHT_CONTROL=ON -DSTATISTICS=0 ..
#1.54inch LCD Module
sudo cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_1INCH54_LCD=ON -DBACKLIGHT_CONTROL=ON -DSTATISTICS=0 ..
#1.69inch LCD Module
sudo cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_1INCH69_LCD=ON -DBACKLIGHT_CONTROL=ON -DSTATISTICS=0 ..
#1.8inch LCD Module
sudo cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_1INCH8_LCD=ON -DBACKLIGHT_CONTROL=ON -DSTATISTICS=0 ..
#1.9inch LCD Module
sudo cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_1INCH9_LCD=ON -DBACKLIGHT_CONTROL=ON -DSTATISTICS=0 ..
#2inch LCD Module
sudo cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_2INCH_LCD=ON -DBACKLIGHT_CONTROL=ON -DSTATISTICS=0 ..
#2.4inch LCD Module
sudo cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_2INCH4_LCD=ON -DBACKLIGHT_CONTROL=ON -DSTATISTICS=0 ..
```

Set up to Start Automatically

```
sudo cp ~/Waveshare_fbcp/build/fbcp /usr/local/bin/fbcp
sudo nano /etc/rc.local
```

Add fbcp& before exit 0. Note that you must add "&" to run in the background, otherwise the system may not be able to start.



```
# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi

fbcp&
exit 0
```

Set the Display Resolution

Set the user interface display size in the "/boot/config.txt" file.

```
sudo nano /boot/config.txt
```

Add the configuration statement for the resolution to the "config.txt" file.

```
hdmi_force_hotplug=1
hdmi_cvt=[options]
hdmi_group=2
hdmi_mode=1
hdmi_mode=87
display_rotate=0
```

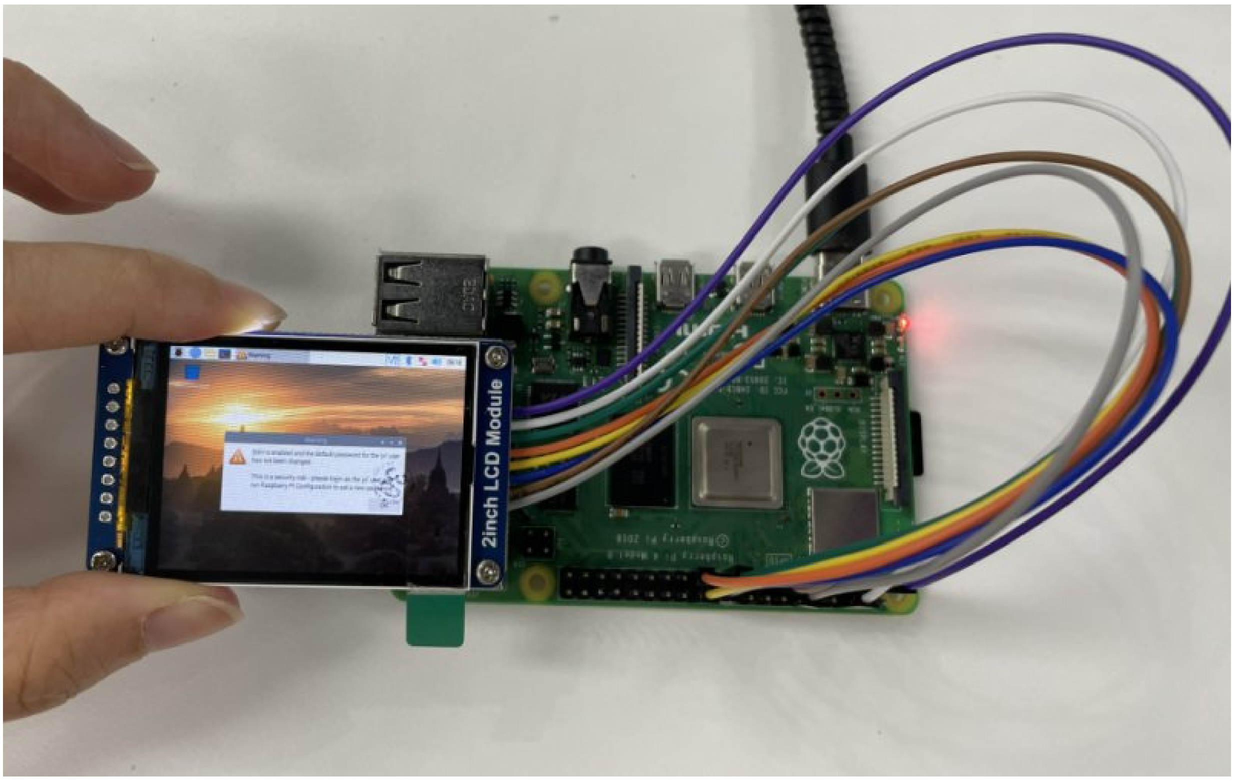
Replace the above "hdmi_cvt=[options]" according to the LCD Module that you are using.

```
#2.4inchinch LCD Module & 2inchinch LCD Module
hdmi_cvt=640 480 60 1 0 0 0
#1.9inch LCD Module
hdmi_cvt 640 340 60 6 0 0 0
#1.8inch LCD Module
hdmi_cvt=400 300 60 1 0 0 0
#1.69inch LCD Module
hdmi_cvt 560 480 60 6 0 0 0
#1.47inch LCD Module
hdmi_cvt 640 344 60 6 0 0 0
#1.3inch LCD Module & 1.54inch LCD Module
hdmi_cvt 480 480 60 6 0 0 0
#1.14inch LCD Module
hdmi_cvt 480 270 60 6 0 0 0
#0.96inch LCD Module
hdmi_cvt 320 160 60 6 0 0 0
```

And then reboot the system:

```
sudo reboot
```

After rebooting the system, the Raspberry Pi OS user interface will be displayed.



STM32

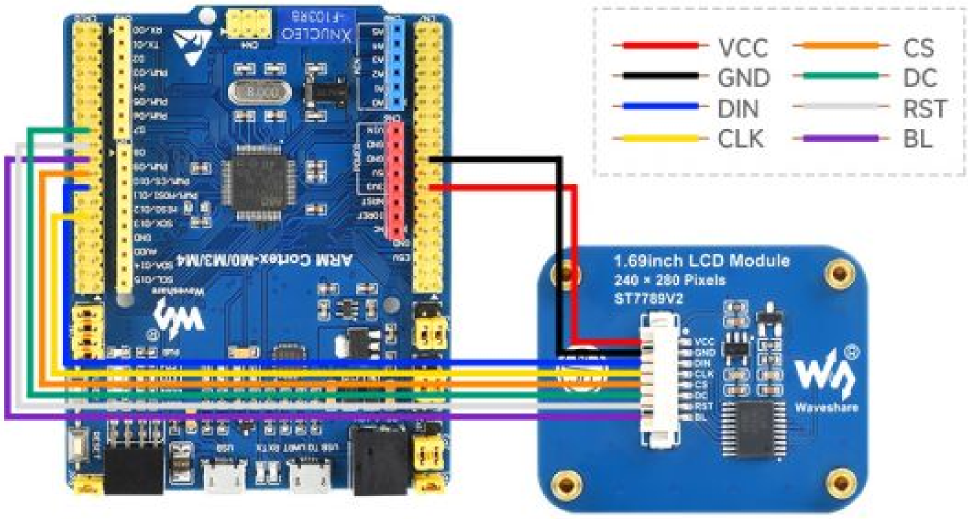
Hardware Connection

The demo we provided is based on STM32F103RBT6 and the connections provided correspond to the pins of the STM32F103RBT6, so if you need to port the demo, please connect the pins according to the actual pins:

STM32F103ZET Pin

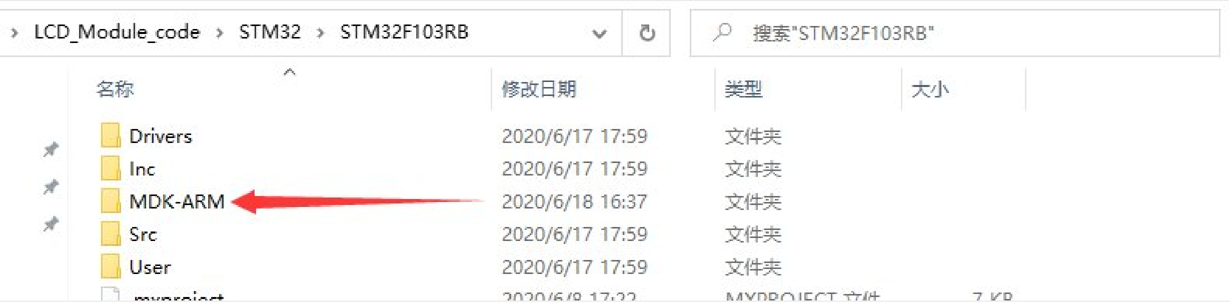
LCD	STM32
VCC	3.3V
GND	GND
DIN	PA7
CLK	PA5
CS	PB6
DC	PA8
RST	PA9
BL	PC7

Taking the XNUCLEO-F103RB development board developed by our company as an example, the connection is as follows:



Run the Demo

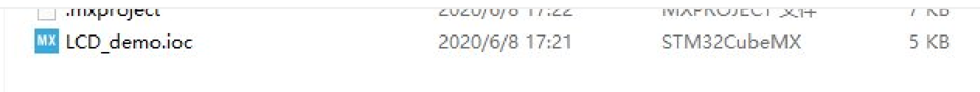
- Download the demo and find the STM32 demo file directory, open "LCD_demo.uvprojx" in the directory of "STM32\STM32F103RBT6\MDK-ARM", and then you can see the project.



› LCD_Module_code › STM32 › STM32F103RB

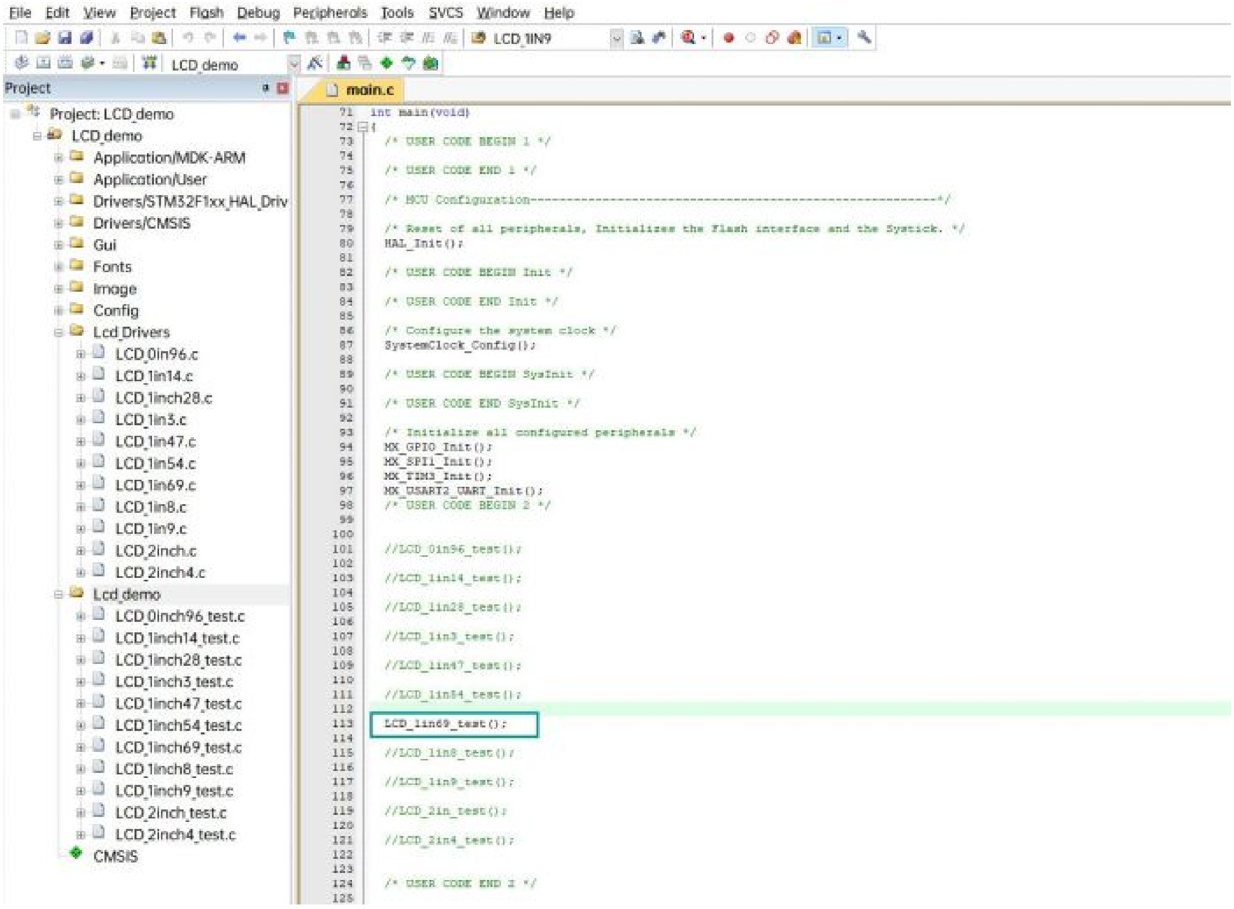
搜索"STM32F103RB"

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	
.myproject	2020/6/8 17:22	MYPROJECT 文件	7 KB



.mxproject 2020/6/8 17:22 MXPROJECT 文件 7 KB
LCD_demo.ioc 2020/6/8 17:21 STM32CubeMX 5 KB

- Open "main.c" and you can see all the test demos.
- As the demo we used is the 1.69-inch LCD module, you need to remove the comment in front of "LCD_1in69_test()"; and recompile and download it.



```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_SPI1_Init();
  MX_TIM3_Init();
  MX_USART2_UART_Init();
  /* USER CODE BEGIN 2 */


  //LCD_0in96_test();

  //LCD_1in14_test();

  //LCD_1in28_test();

  //LCD_1in3_test();

  //LCD_1in47_test();

  //LCD_1in54_test();

  LCD_1in69_test();

  //LCD_1in8_test();

  //LCD_1in9_test();

  //LCD_2in_test();

  //LCD_2in4_test();


  /* USER CODE END 2 */

```

Demo Description

Underlying Hardware Interface

- Data type:

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

- How to initialize and exit the module:

```
void DEV_Module_Init(void);
void DEV_Module_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 of
f;
```

- Write and read GPIO:

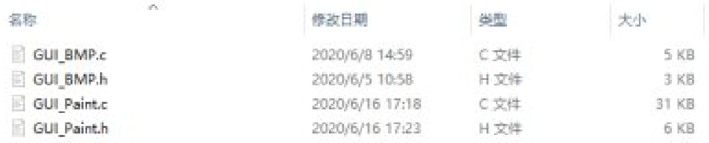
```
void    DEV_Digital_Write(UWORD Pin, UBYTE Value);
UBYTE   DEV_Digital_Read(UWORD Pin);
```

- SPI writes data:

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

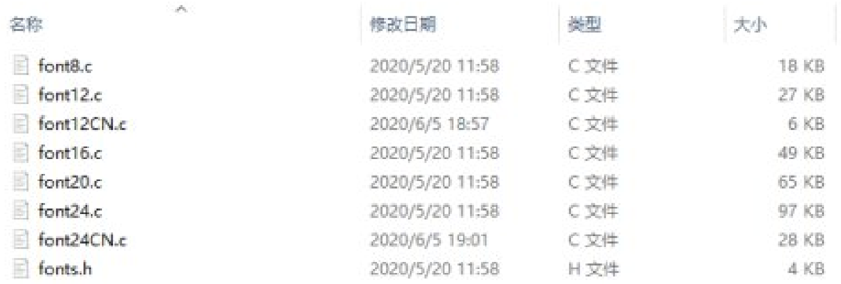
Upper Application

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

The character font on 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

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

```
void Paint_NewImage(UWORD Width, UWORD Height, UWORD Rotate, UWORD Color)
Parameters:
    Width: 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;
```

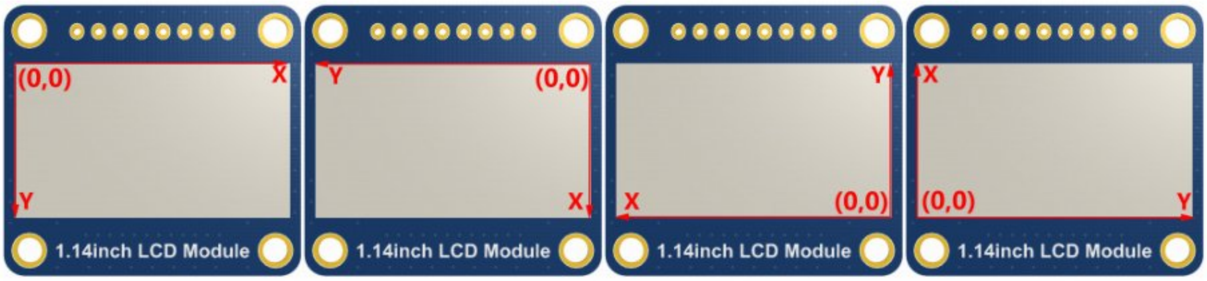
- Set the drawing pixel function

```
void Paint_SetDisplayFuntion(void (*Display)(UWORD,UWORD,UWORD));
parameter:
    Display: Pointer to the pixel drawing function, which is used to write data to t
he 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, an image buffer can exist multiple, and 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 add
ress 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, 18
0, 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, MIRR
OR_VERTICAL, MIRROR_ORIGIN correspond to no mirror, horizontal mirror, vertical mirr
or, 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 Co
lor)
Parameters:
    Xstart: the x-starting coordinate of the window
    Ystart: indicates the Y starting point of the window
    Xend: the x-end coordinate of the window
    Yend: indicates the y-end coordinate of the window
    Color: fill Color
```

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

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

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

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color,
LINE_STYLE Line_Style ,  LINE_STYLE Line_Style)
Parameters:
    Xstart: the x-starting coordinate of a line
    Ystart: indicates the Y starting point of a line
    Xend: x-terminus of a line
    Yend: the y-end coordinate of a line
    Color: fill Color
    Line_width: The width of the line, which provides a default of eight widths
        typedef enum {
            DOT_PIXEL_1X1   = 1,        // 1 x 1
            DOT_PIXEL_2X2  ,            // 2 X 2
            DOT_PIXEL_3X3  ,            // 3 X 3
            DOT_PIXEL_4X4  ,            // 4 X 4
            DOT_PIXEL_5X5  ,            // 5 X 5
            DOT_PIXEL_6X6  ,            // 6 X 6
            DOT_PIXEL_7X7  ,            // 7 X 7
            DOT_PIXEL_8X8  ,            // 8 X 8
        } DOT_PIXEL;
    Line_Style: line style. Select whether the lines are joined in a straight or das
hed way
        typedef enum {
            LINE_STYLE_SOLID = 0,
            LINE_STYLE_DOTTED,
        } LINE_STYLE;
```

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

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

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

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

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

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

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

```
void Paint_DrawString_EN(UWORD Xstart, UWORD Ystart, const char * pString, sFONT* Fo
nt, 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, and write a string of Chinese characters, you can choose GB2312 encoding character font, font foreground color, and font background color.

```
void Paint_DrawString_CN(UWORD Xstart, UWORD Ystart, const char * pString, cFONT* fo
nt, UWORD Color_Foreground,  UWORD Color_Background)
Parameters:
    Xstart: the x-coordinate of the left vertex of a character
    Ystart: the Y coordinate of the font's left vertex
    PString: string, string is a pointer
    Font: GB2312 encoding character Font library, in the Fonts folder provides the f
ollowing 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, and write a string of numbers, you can choose Ascii visual character library, font foreground color, or font background color.

```
void Paint_DrawNum(UWORD Xpoint, UWORD Ypoint, double Nummber, sFONT* Font, UWORD Digit, UWORD Color_Foreground,   UWORD Color_Background)
Parameters:
    Xpoint: the x-coordinate of the left vertex of a character
    Ypoint: the Y coordinate of the left vertex of the font
    Nummber: indicates the number displayed, which can be a decimal
    Digit: It's a decimal number
    Font: Ascii visual character library, in the Fonts folder provides the following Fonts:
        Font8: 5*8 font
        Font12: 7*12 font
        Font16: 11*16 font
        Font20: 14*20 font
        Font24: 17*24 font
    Color_Foreground: Font color
    Color_Background: indicates the background color
```

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

Arduino

Note: all the demos have been tested on the Arduino uno, you need to make sure the connection is correct if you want to use other Arduino.

IDE Installation

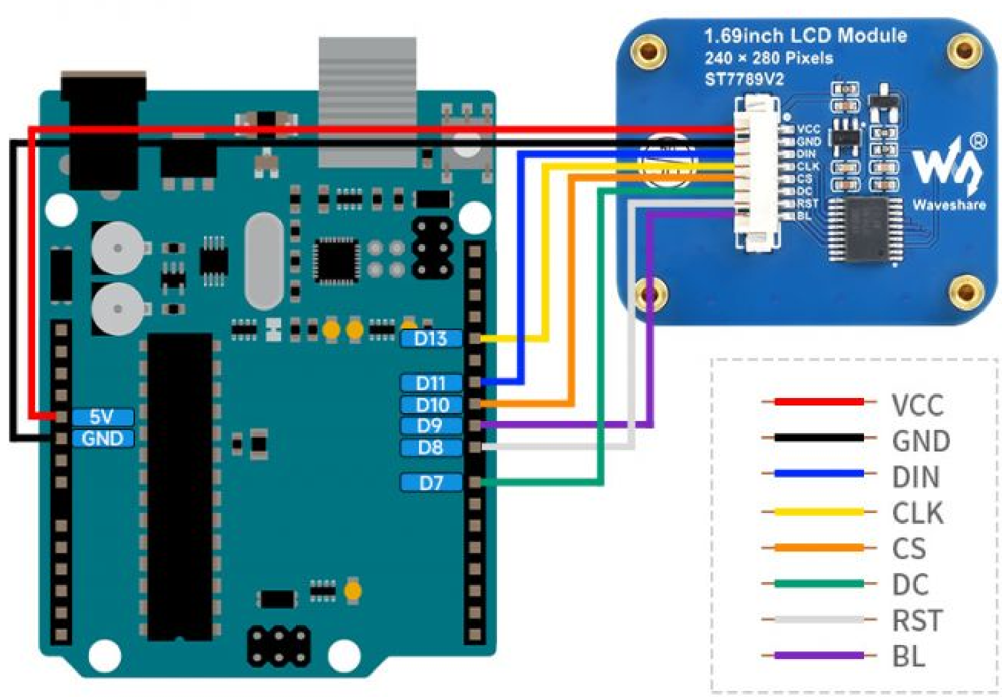
How to install Arduino IDE

Hardware Connection

Arduino UNO Pins

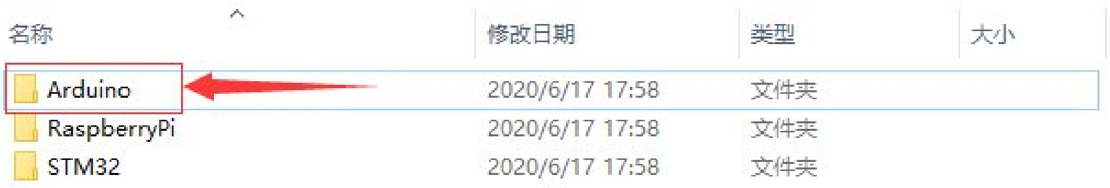
LCD	UNO
VCC	5V
GND	GND
DIN	D11
CLK	D13
CS	D10
DC	D7
RST	D8
BL	D9

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



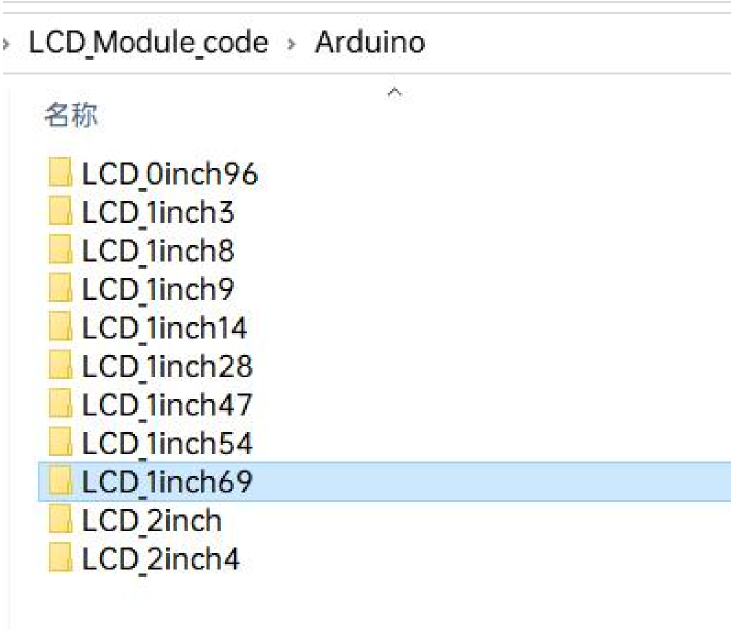
Run the Demo

- In the #Resource download the demo, and then unzip it. The Arduino demo is 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	

- The demo we used is 1.69inch LCD Module, so we need to open the LCD_1inch69 file folder and run the LCD_1inch69.ino file.

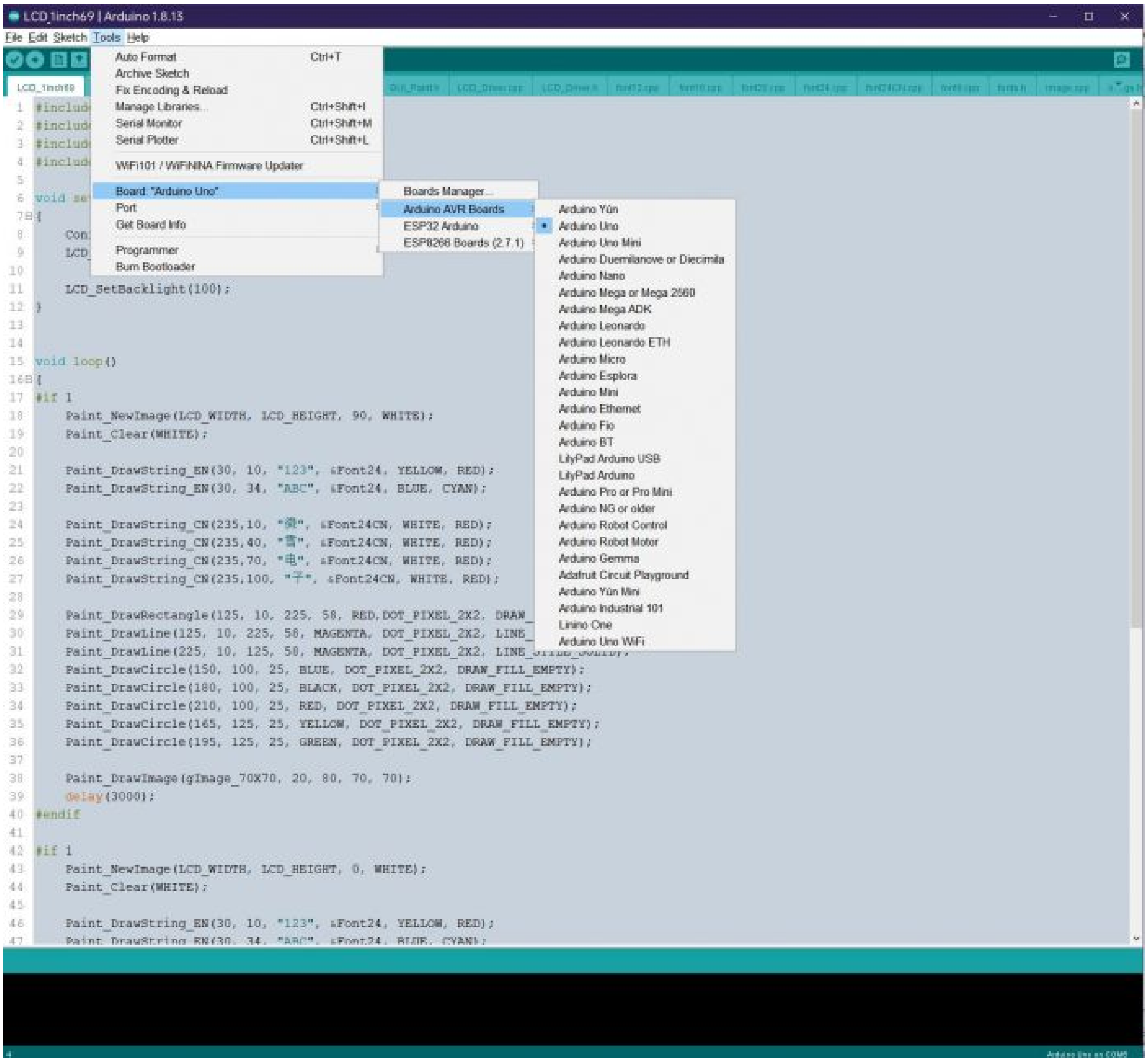


› LCD_Module_code › Arduino

名称

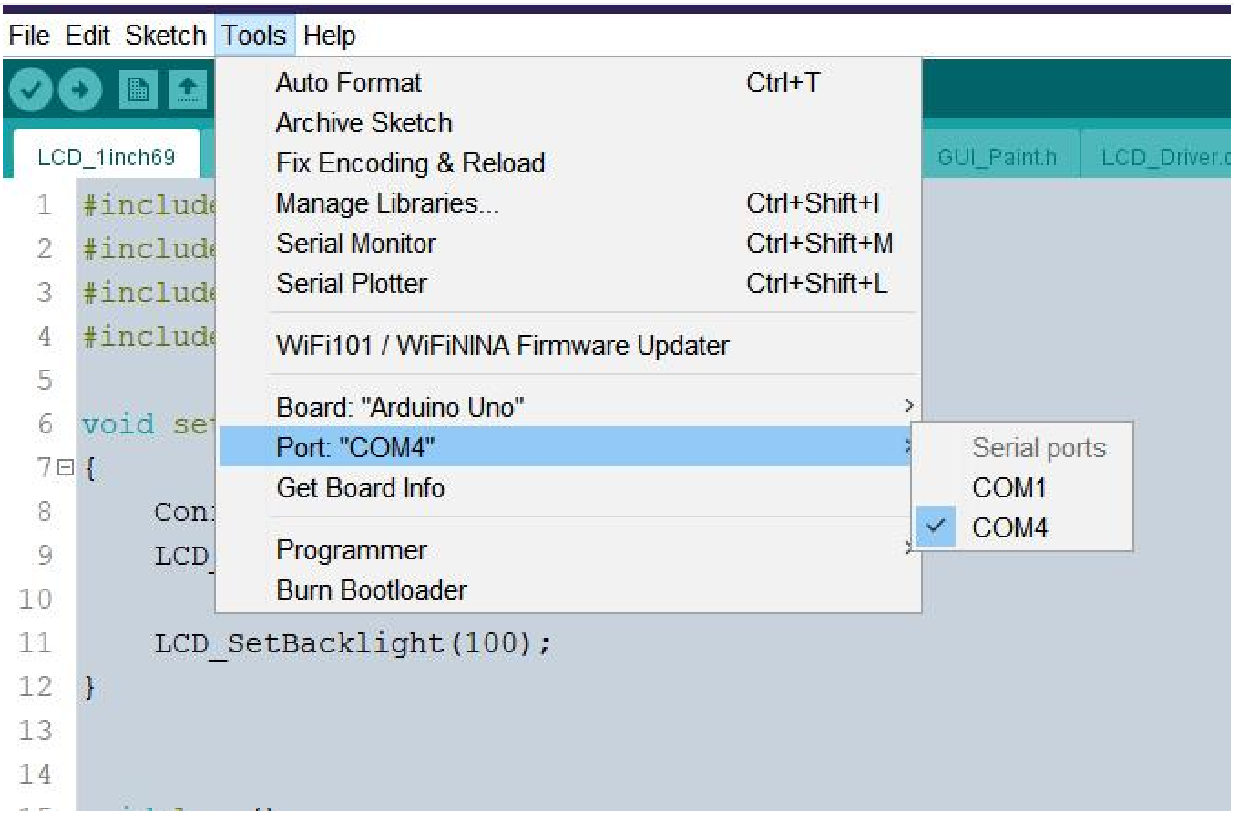
- LCD_0inch96
- LCD_1inch3
- LCD_1inch8
- LCD_1inch9
- LCD_1inch14
- LCD_1inch28
- LCD_1inch47
- LCD_1inch54
- LCD_1inch69
- LCD_2inch
- LCD_2inch4

- Open the demo and chose the Dev board model as Arduino UNO.



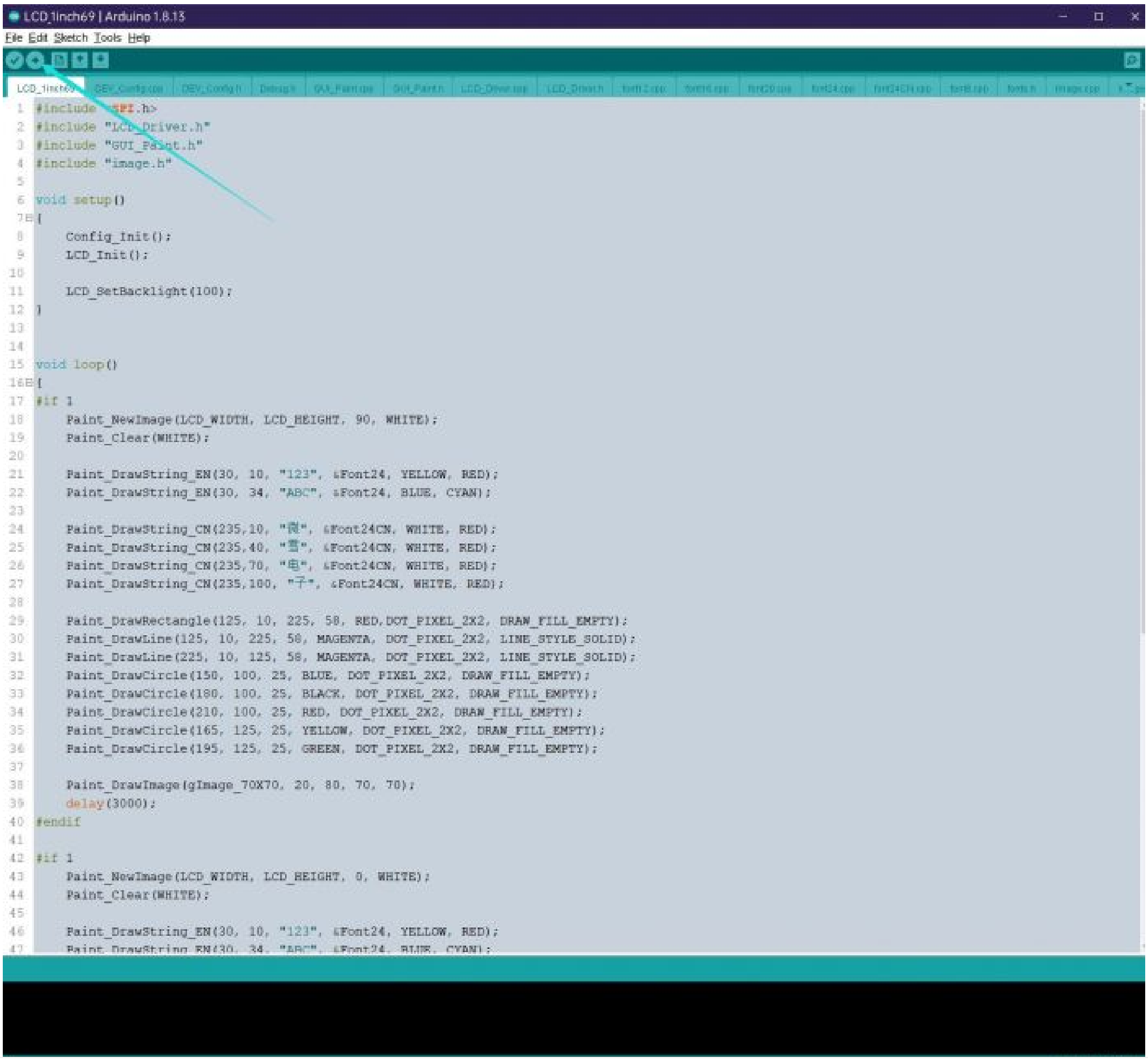
```
#includ
#includ
#includ
#includ

void se
{
    Con
    LCD_

    LCD_SetBacklight(100);
}


void loop()
{
#if 1
    Paint_NewImage(LCD_WIDTH, LCD_HEIGHT, 90, WHITE);
    Paint_Clear(WHITE);

    Paint_DrawString_EN(30, 10, "123", &Font24, YELLOW, RED);
    Paint_DrawString_EN(30, 34, "ABC", &Font24, BLUE, CYAN);

    Paint_DrawString_CN(235,10, "微", &Font24CN, WHITE, RED);
    Paint_DrawString_CN(235,40, "雪", &Font24CN, WHITE, RED);
    Paint_DrawString_CN(235,70, "电", &Font24CN, WHITE, RED);
    Paint_DrawString_CN(235,100, "子", &Font24CN, WHITE, RED);

    Paint_DrawRectangle(125, 10, 225, 58, RED,DOT_PIXEL_2X2, DRAW_
    Paint_DrawLine(125, 10, 225, 58, MAGENTA, DOT_PIXEL_2X2, LINE_
    Paint_DrawLine(225, 10, 125, 58, MAGENTA, DOT_PIXEL_2X2, LINE_
    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);
    delay(3000);
#endif

#if 1
    Paint_NewImage(LCD_WIDTH, LCD_HEIGHT, 0, WHITE);
    Paint_Clear(WHITE);

    Paint_DrawString_EN(30, 10, "123", &Font24, YELLOW, RED);
    Paint_DrawString_EN(30, 34, "ABC", &Font24, BLUE, CYAN);
```

- Choose the corresponding COM port.



```
#include
#include
#include
#include

void se
{
    Con
    LCD_

    LCD_SetBacklight(100);
}
```

- Then click to compile and download.

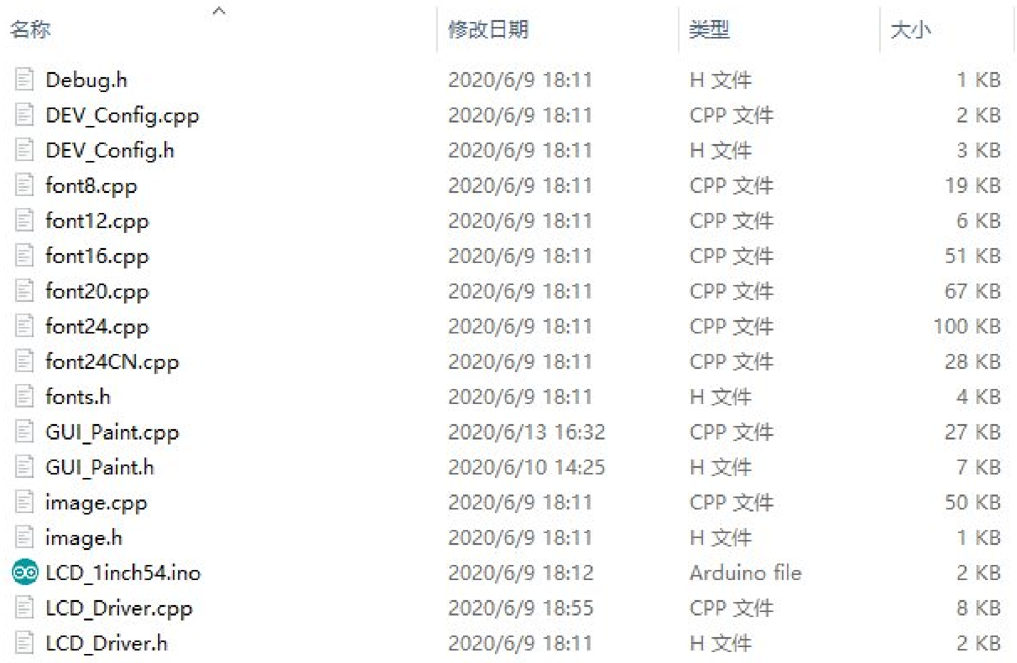


```
#include <SPI.h>
#include "LCD_Driver.h"
#include "GUI_Paint.h"
#include "image.h"

void setup()
{
    Config_Init();
    LCD_Init();

    LCD_SetBacklight(100);
}


void loop()
{
#if 1
    Paint_NewImage(LCD_WIDTH, LCD_HEIGHT, 90, WHITE);
    Paint_Clear(WHITE);

    Paint_DrawString_EN(30, 10, "123", &Font24, YELLOW, RED);
    Paint_DrawString_EN(30, 34, "ABC", &Font24, BLUE, CYAN);

    Paint_DrawString_CN(235,10, "微", &Font24CN, WHITE, RED);
    Paint_DrawString_CN(235,40, "雪", &Font24CN, WHITE, RED);
    Paint_DrawString_CN(235,70, "电", &Font24CN, WHITE, RED);
    Paint_DrawString_CN(235,100, "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);
    delay(3000);
#endif

#if 1
    Paint_NewImage(LCD_WIDTH, LCD_HEIGHT, 0, WHITE);
    Paint_Clear(WHITE);

    Paint_DrawString_EN(30, 10, "123", &Font24, YELLOW, RED);
    Paint_DrawString_EN(30, 34, "ABC", &Font24, BLUE, CYAN);
```

Demo 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

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 demo 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 levels.

- 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.

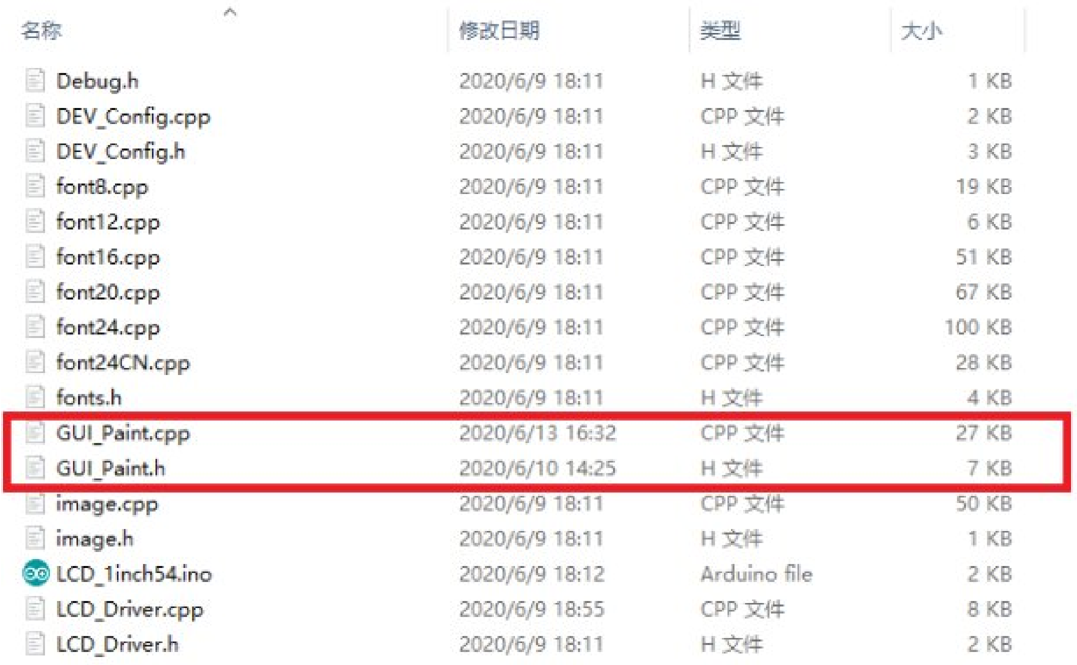
- SPI output data:

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

The parameter is char type, occupying 8 bits.

Upper Application

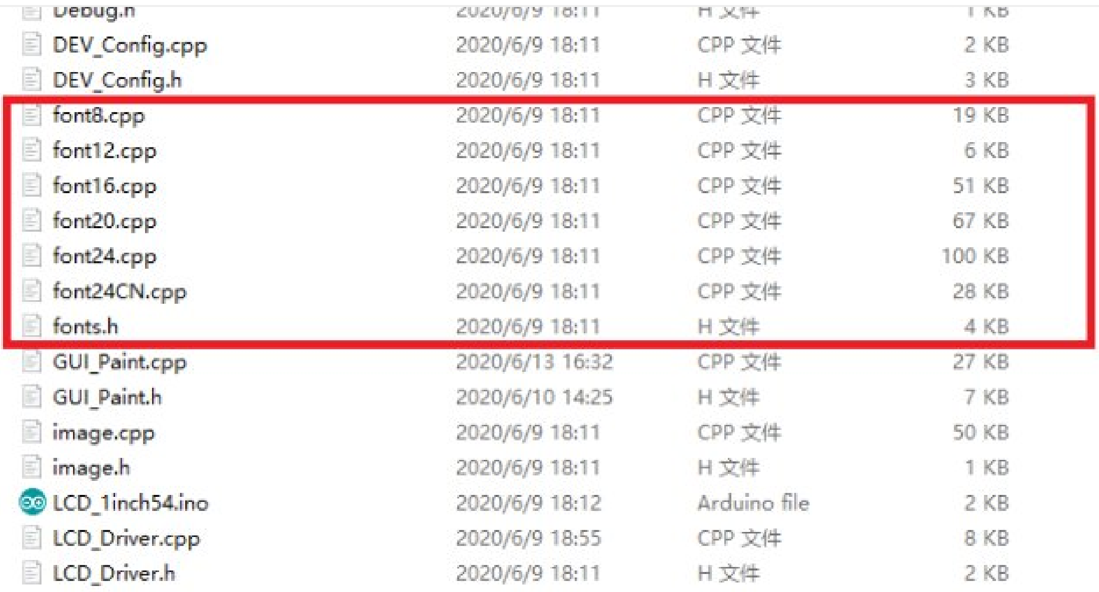
For the screen, if you need to draw pictures, display Chinese and English characters, display pictures, etc., you can use the upper application to do, and we provide some basic functions here about some graphics processing in the directory GUI_Paint.c(.h) Note: Because of the size of the internal RAM 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

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



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

- 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;
```

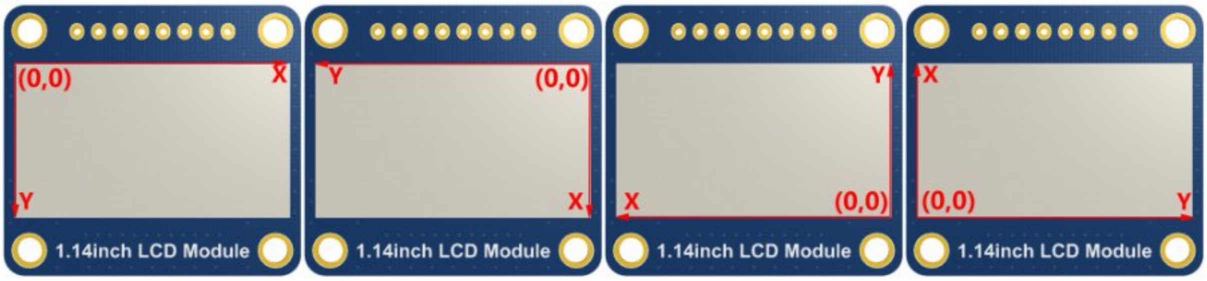
- Set the drawing pixel function.

```
void Paint_SetDisplayFuntion(void (*Display)(UWORD,UWORD,UWORD));
parameter:
    Display: Pointer to the pixel drawing function, which is used to write data to t
he 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, and 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 add
ress 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, 18
0, 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, MIRR
OR_VERTICAL, MIRROR_ORIGIN correspond to no mirror, horizontal mirror, vertical mirr
or, 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 Co
lor)
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, and the style of the point.

```
void Paint_DrawPoint(UWORD Xpoint, UWORD Ypoint, UWORD Color, DOT_PIXEL Dot_Pixel, D
OT_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, and line style.

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color,
LINE_STYLE Line_Style ,  LINE_STYLE Line_Style)
Parameters:
        Xstart: the x-starting coordinate of 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 a rectangle: In the image buffer, draw a rectangle from (Xstart, Ystart) to (Xend, Yend), you can choose the color, the width of the line, and whether to fill the inside of the rectangle.

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

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

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

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

```
void Paint_DrawChar(UWORD Xstart, UWORD Ystart, const char Ascii_Char, sFONT* Font,
UWORD Color_Foreground,  UWORD Color_Background)
Parameters:
        Xstart: the x-coordinate of the left vertex of a character
        Ystart: the Y coordinate of the font's left vertex
        Ascii_Char: indicates the Ascii character
        Font: Ascii visual character library, in the Fonts folder provides the follo
wing 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* Fo
nt, 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 follo
wing 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* fo
nt, UWORD Color_Foreground,  UWORD Color_Background)
Parameters:
        Xstart: the x-coordinate of the left vertex of a character
        Ystart: the Y coordinate of the font's left vertex
        PString: string, string is a pointer
        Font: GB2312 encoding character Font library, in the Fonts folder provides t
he 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, UWORD Di
git, 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 follo
wing 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(UWORD Xpoint, UWORD Ypoint, double Nummber, UBYTE Decimal_Po
int, sFONT* Font, UWORD Color_Foreground, UWORD Color_Background);
parameter:
         Xstart: the X coordinate of the left vertex of the character
         Ystart: Y coordinate of the left vertex of the font
         Nummber: 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, 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
```

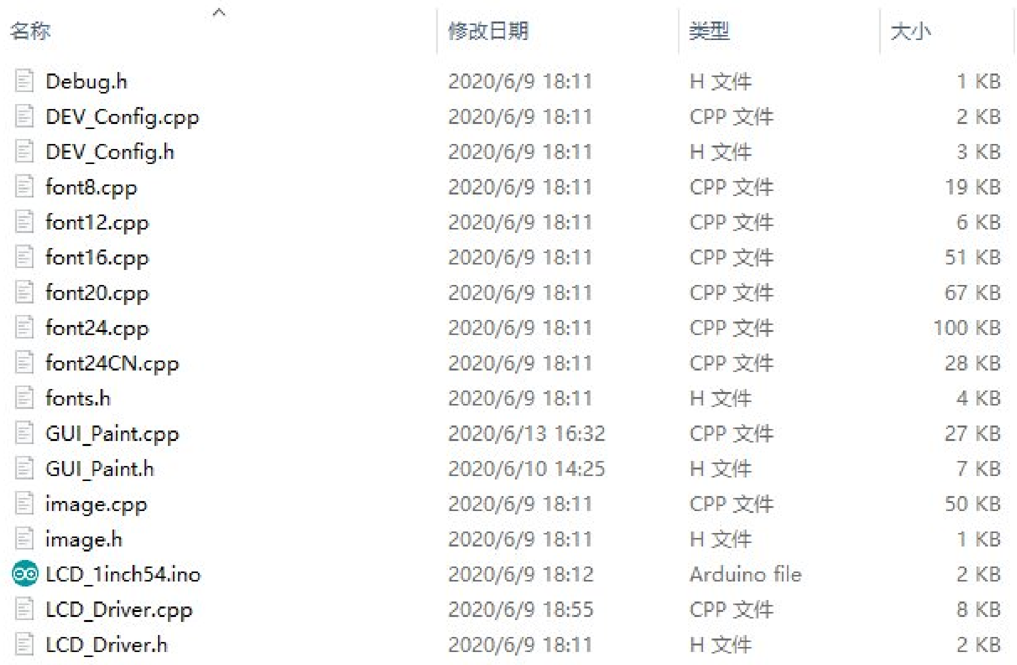
- Display image: at (Xstart Ystart) as the left vertex, display an image whose width is W_Image and height is H_Image;

```
void Paint_DrawImage(const unsigned char *image, UWORD xStart, UWORD yStart, UWORD W_Image, UWORD 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
```

Demo 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

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 demo 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 levels.

- 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.

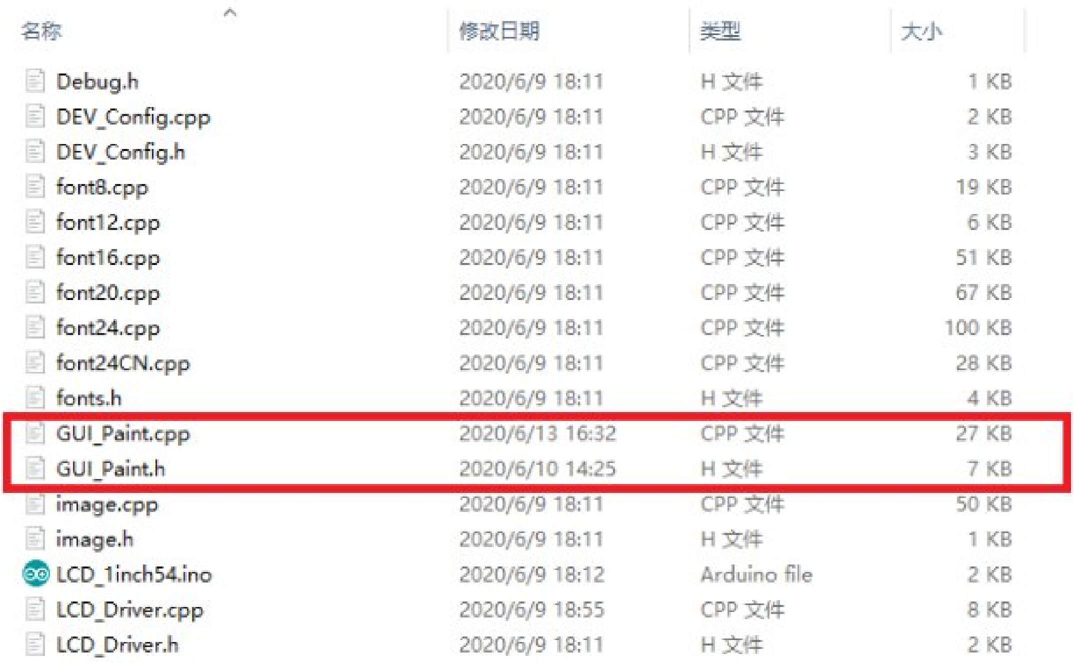
- SPI output data

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

The parameter is char type, occupying 8 bits.

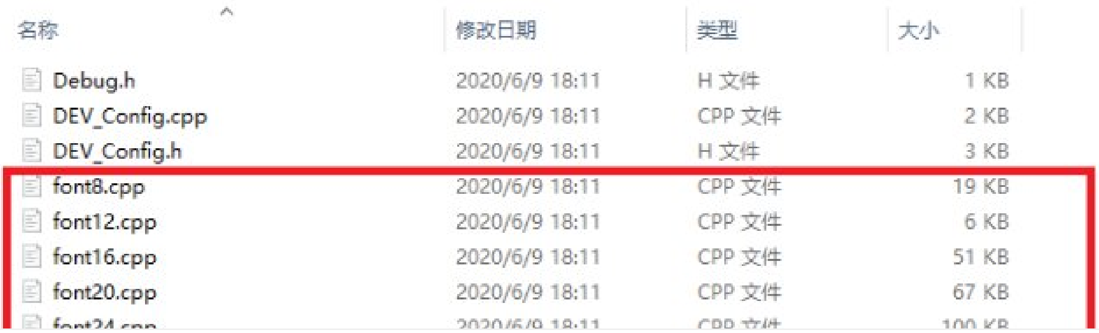
Upper Application

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.

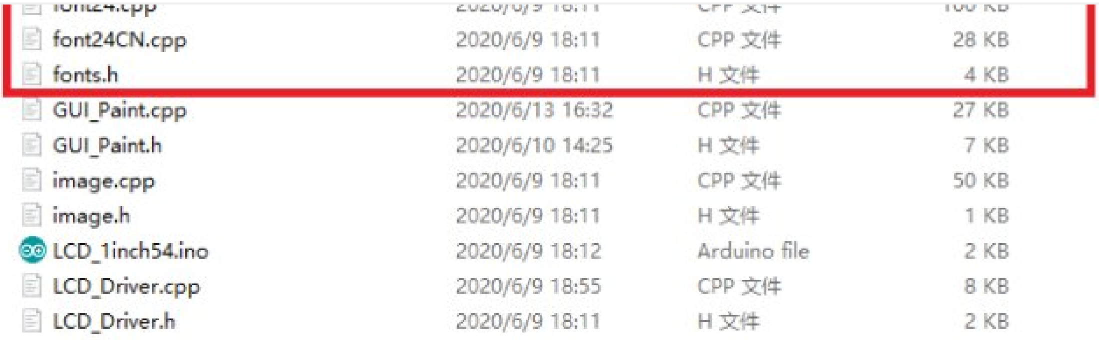


名称	修改日期	类型	大小
Debug.h	2020/6/9 18:11	H 文件	1 KB
DEV_Config.cpp	2020/6/9 18:11	CPP 文件	2 KB
DEV_Config.h	2020/6/9 18:11	H 文件	3 KB
font8.cpp	2020/6/9 18:11	CPP 文件	19 KB
font12.cpp	2020/6/9 18:11	CPP 文件	6 KB
font16.cpp	2020/6/9 18:11	CPP 文件	51 KB
font20.cpp	2020/6/9 18:11	CPP 文件	67 KB
font24.cpp	2020/6/9 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

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

- 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;
```

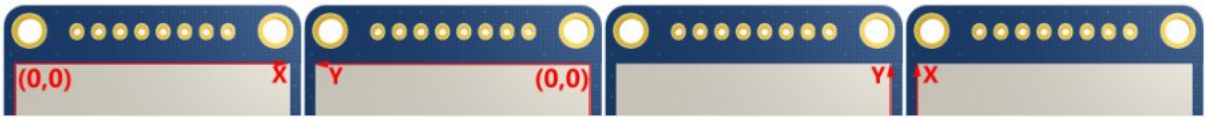
- Set the drawing pixel function.

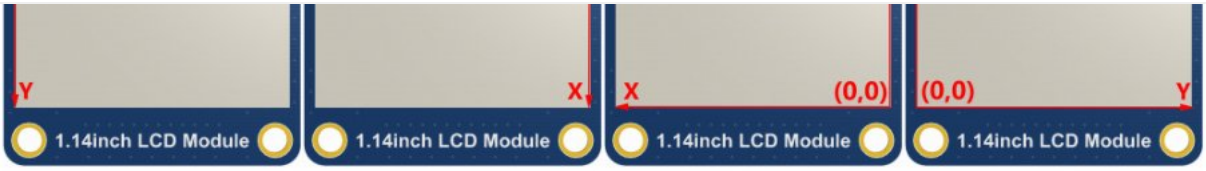
```
void Paint_SetDisplayFuntion(void (*Display)(UWORD,UWORD,UWORD));
parameter:
    Display: Pointer to the pixel drawing function, which is used to write data to t
he specified location in the internal RAM of the LCD;
```

- Select image buffer: the purpose of the selection is that you can create multiple image attributes, an image buffer can exist multiple, and 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 add
ress 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, and the style of the point.

```
void Paint_DrawPoint(UWORD Xpoint, UWORD Ypoint, UWORD Color, DOT_PIXEL Dot_Pixel, DOT_STYLE Dot_Style)
Parameters:
    Xpoint: indicates the X coordinate of a point
    Ypoint: indicates the Y coordinate of a point
    Color: fill Color
    Dot_Pixel: The size of the dot, providing a default of eight size points
        typedef enum {
                DOT_PIXEL_1X1  = 1,             // 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, and line style.

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color,
LINE_STYLE Line_Style ,  LINE_STYLE Line_Style)
Parameters:
        Xstart: the x-starting coordinate of 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 a rectangle: In the image buffer, draw a rectangle from (Xstart, Ystart) to (Xend, Yend), you can choose the color, the width of the line, and whether to fill the inside of the rectangle.

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

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

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

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

```
void Paint_DrawChar(UWORD Xstart, UWORD Ystart, const char Ascii_Char, sFONT* Font,
UWORD Color_Foreground,  UWORD Color_Background)
Parameters:
        Xstart: the x-coordinate of the left vertex of a character
        Ystart: the Y coordinate of the font's left vertex
        Ascii_Char: indicates the Ascii character
        Font: Ascii visual character library, in the Fonts folder provides the follo
wing 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* Fo
nt, 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 follo
wing 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, and write a string of Chinese characters, you can choose GB2312 encoding character font, font foreground color, and font background color.

```
void Paint_DrawString_CN(UWORD Xstart, UWORD Ystart, const char * pString, cFONT* fo
nt, UWORD Color_Foreground,  UWORD Color_Background)
Parameters:
        Xstart: the x-coordinate of the left vertex of a character
        Ystart: the Y coordinate of the font's left vertex
        PString: string, string is a pointer
        Font: GB2312 encoding character Font library, in the Fonts folder provides t
he 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, and write a string of numbers, you can choose Ascii visual character library, font foreground color, or font background color.

```
void Paint_DrawNum(UWORD Xpoint, UWORD Ypoint, double Nummber, sFONT* Font, UWORD Di
git, 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 follo
wing 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(UWORD Xpoint, UWORD Ypoint, double Nummber, UBYTE Decimal_Po
int, sFONT* Font, UWORD Color_Foreground, UWORD Color_Background);
parameter:
         Xstart: the X coordinate of the left vertex of the character
         Ystart: Y coordinate of the left vertex of the font
         Nummber: 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 pro
vided 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, 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 as 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, UWORD xStart, UWORD yStart, UWORD W_Image, UWORD 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
```

Resource

Document

- Schematic
- ST7789V2 datasheet
- 2D Drawing

Demo

- Demo example

Software

- lcd
- Image2Lcd
- Image Extraction

FAQ

Question: What is the maximum power consumption of 1.69inch LCD Module?

Answer:
3.3V 90mA

Question: The maximum brightness of 1.69inch LCD Module is?

Answer:
3.3V 500cd/㎡

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