1.28inch Touch LCD

1.28inch Touch LCD

1.28", 240x240
SPI, I2C

Overview

Introduction

We provide demos of Raspberry Pi, Raspberry Pi Pico, STM32, and Arduino for this product.

Parameters

Operating voltage	3.3V / 5V	**Resolution**	240 × 240 pixels
LCD controller	GC9A01	**Communication interface**	4-wire SPI
Touch controller	CST816S	**Communication interface**	I2C
Display panel	IPS	**Touch type**	Capacitive
Display dimensions	Φ32.4mm	**Pixel size**	0.135 × 0.135mm
Dimensions	39.15 × 37mm Φ38.5mm		

- **Please make sure the power supply is as same as the logic level, or it may cause abnormal operation.**

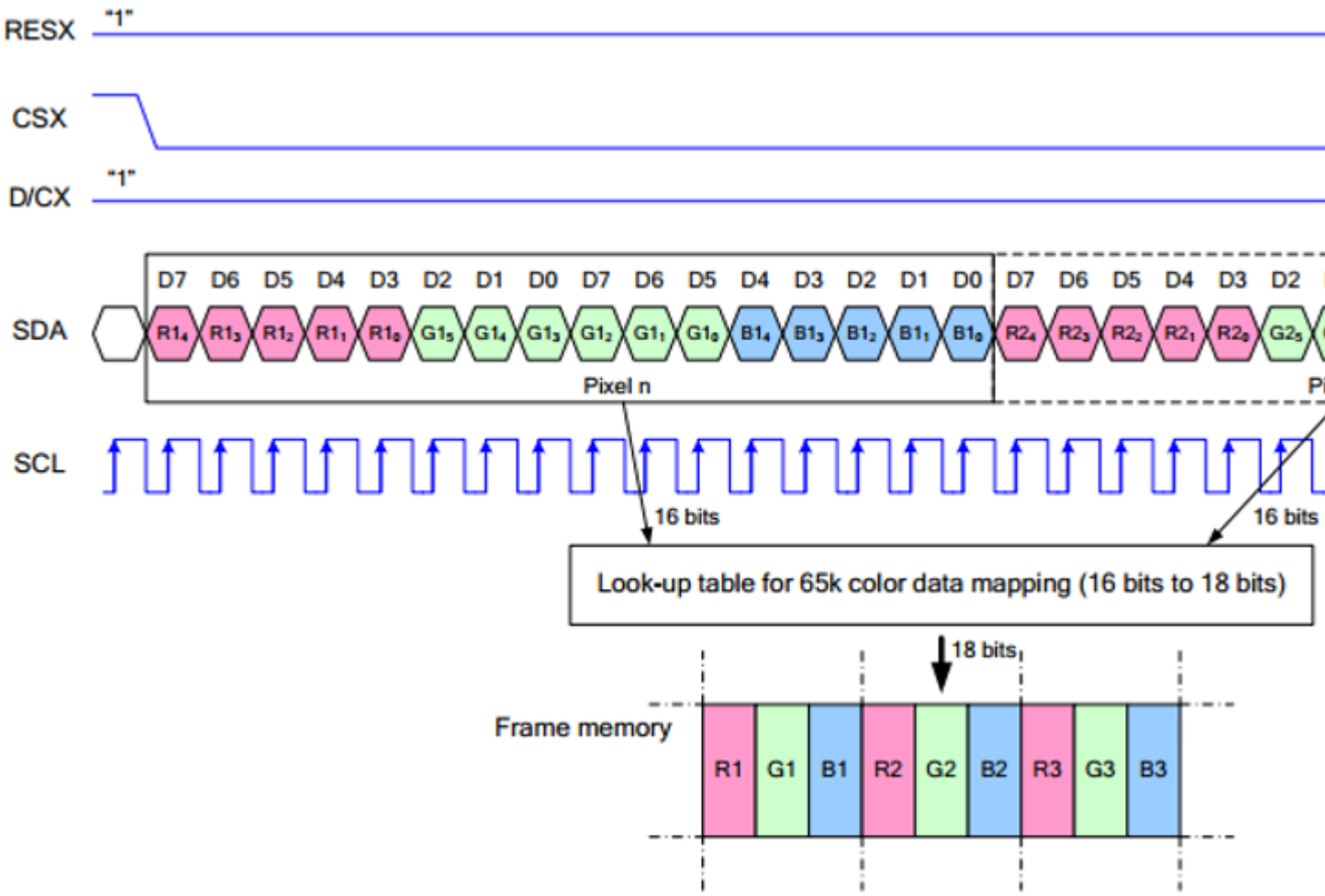
Interface Description

Pin	Function
VCC	Power input (3.3V/5V)
GND	Ground
MISO	SPI MISO pin
MOSI	SPI MOSI pin
SCLK	SPI CLK pin
LCS_CS	LCD chip selection pin, low active
LCS_DC	LCD data/command pin, low for command, high for data
LCS_RST	LCD reset pin, low active
LCS_BL	LCD backlight pin
TP_SDA	TP data pin
TP_SCL	TP clock pin
TP_INT	TP interrupt pin
TP_RST	TP reset pin, low active

LCD and Controller

- This LCD adopts built-in driver GC9A01 with a resolution of 240RGB × 240 dots, which has an internal GRAM of 129,600 bytes. It supports MCU port with 12/16/18 data bus, that is, the common RGB format of RGB444, RGB565, and RGB666.
- For most LCD controllers, the communication mode of the controller can be configured, usually with an 8080 parallel interface, three-wire SPI, four-wire SPI, and other communication methods. This LCD uses a four-wire SPI communication interface, which can greatly save GPIO ports, and the communication speed will be faster.
- Some friends may have doubts, the screen is circular, so which point is the first pixel of the screen? How to determine the coordinates?
    - In fact, you can understand it as a square screen with an inscribed circle drawn inside. We only display content in this inscribed circle, and the pixels in other positions are directly discarded. Most of the circular LCDs on the market are the same.

SPI Communication Protocol



- 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 105 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 selection, 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.

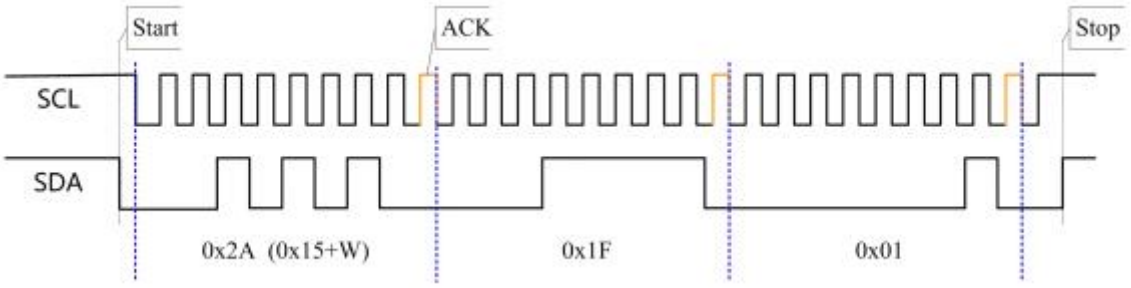
- As can be seen from the figure, when the first falling edge of SCLK starts to transmit data, 8-bit data is transmitted in one clock cycle, using SPI0, bit-by-bit transmission, high-order first, and low-order last.

Touch and Controller

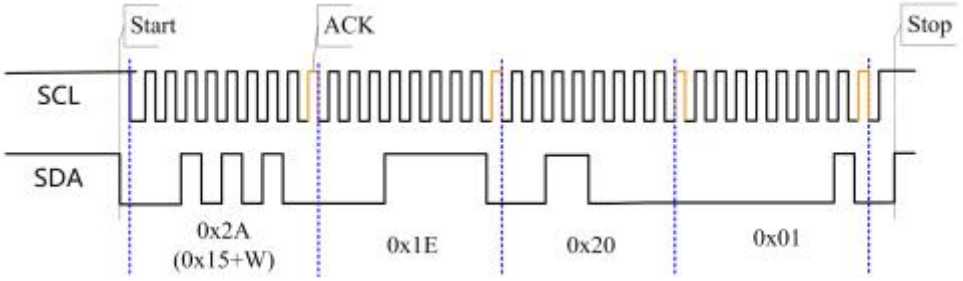
- 1.28-inch Touch LCD is composed of surface toughened glass + thin film FILM material, which has high strength, strong hardness, and good light transmittance. The matching driver chip is CST816S self-capacitance touch chip, which supports the standard I2C communication protocol standard, which can realize a 10Khz~400Khz configurable communication rate.

I2C Communication Protocol

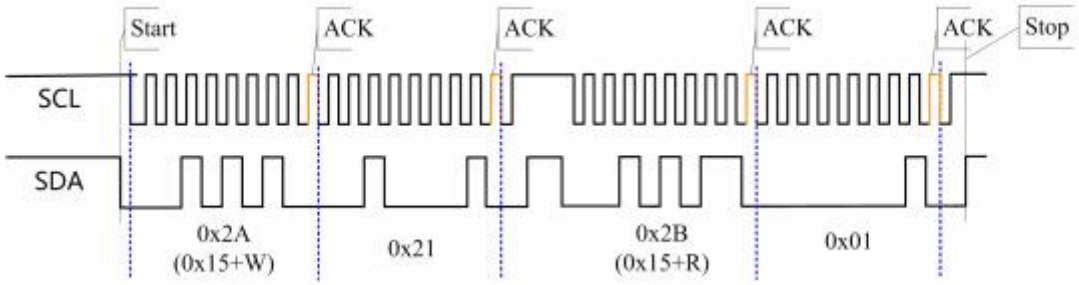
- The 7-bit device address of the chip is 0x15, that is, the device writing address is 0x2A, and reading address is 0x2B.
- Waveform introduction:
  - Write a single byte: (write 0x01 to 0x1F register)



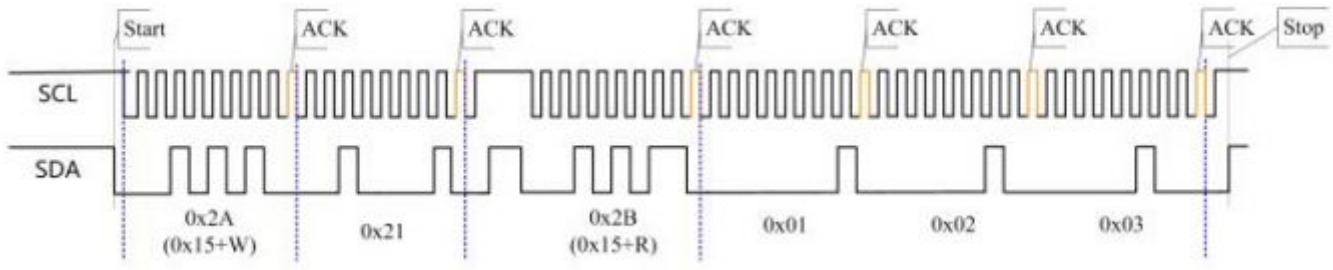
  - Write multiple bytes consecutively (write 0x20, 0x01 to 0x1E and 0x1F respectively)



  - Read a single byte (read a single byte from 0x21)



- Read multiple bytes consecutively (read 3 bytes from 0x21, 0x22, 0x23)



Raspberry Pi User Manual

Hardware Connection

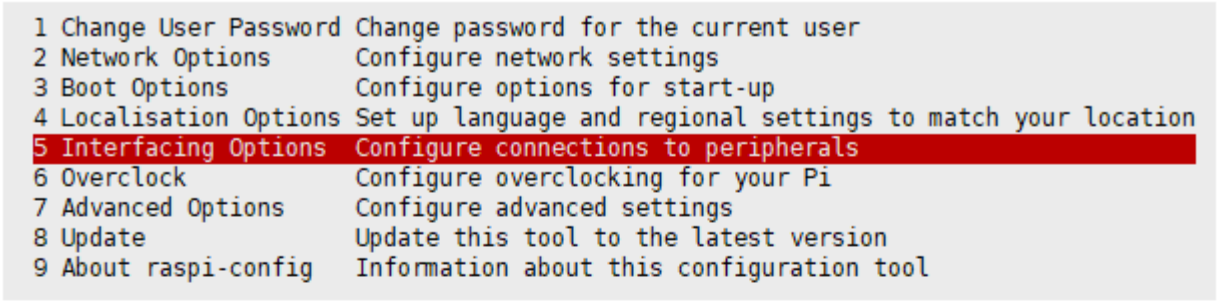
Module Pin	Raspberry Pi (BCM)
VCC	3.3V
GND	GND
MISO	9
MOSI	10
SCLK	11
LCS_CS	8
LCS_DC	25
LCS_RST	27
LCS_BL	18
TP_SDA	2
TP_SCL	3
TP_INT	4
TP_RST	17

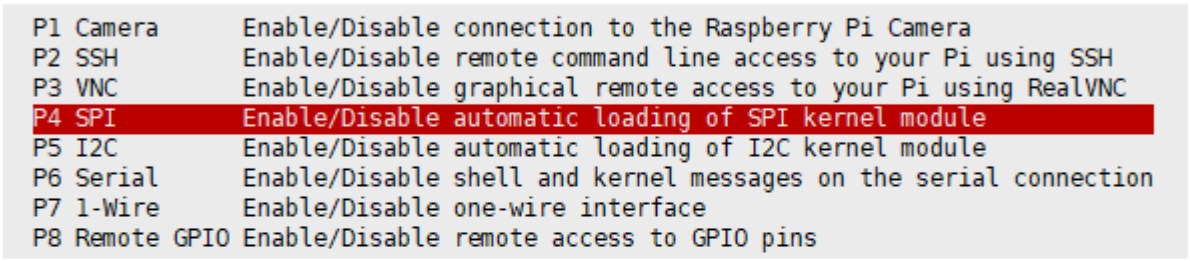
Example

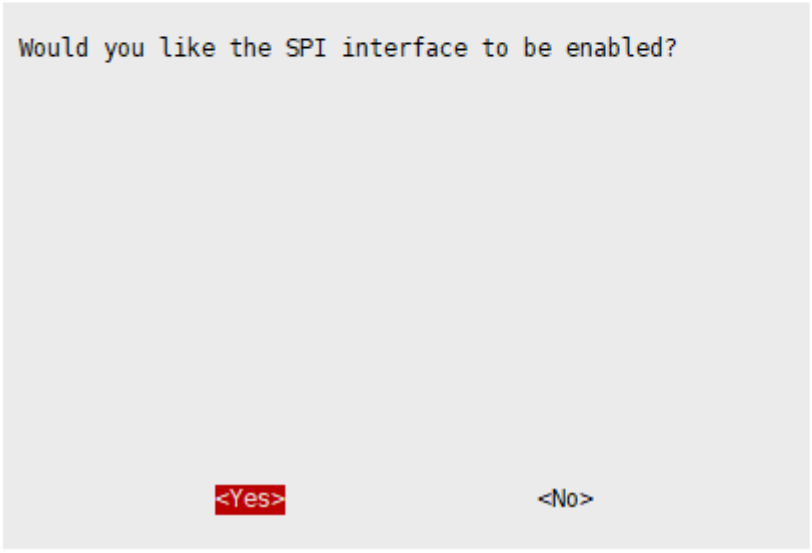
Enabled SPI and I2C Interface

- Open Raspberry Pi and enter the following commands in the config interface.

```
sudo raspi-config
Choose Interfacing Options -> SPI -> Yes Enable SPI interface
Choose Interfacing Options -> I2C -> Yes Enable I2C interface
```







And then reboot the Raspberry Pi:

```
sudo reboot
```

Please make sure SPI has not been occupied by other devices, and you can check it in /boot/config.txt.

Install Library

WiringPi

```
git clone https://github.com/WiringPi/WiringPi
cd WiringPi
./build
gpio -v
```

```
# Run gpio -v and version 2.70 will appear. If it does not appear, it means that there is an error in the installation
```

Python

```
#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 smbus
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 smbus
sudo pip3 install spidev
```

Download Test Demo

Open the Raspberry Pi, and execute:

```
sudo apt-get install unzip -y
cd ~
sudo wget https://files.waveshare.com/upload/f/fb/1.28inch_Touch_LCD_RPI.zip
sudo unzip ./1.28inch_Touch_LCD_RPI.zip
cd 1.28inch_Touch_LCD_RPI
```

Run Test Demo

Execute the following commands in the Raspberry Pi, otherwise, you cannot find the corresponding directory.

C

- Compile it again and it may take a few seconds:

```
cd ~
cd 1.28inch_Touch_LCD_RPI/c
sudo make clean
sudo make -j
sudo ./main
```

Python

```
cd ~
cd 1.28inch_Touch_LCD_RPI/python/examples
sudo python 1inch28_LCD_test.py
```

- Run the demo corresponding to the screen, and the demo supports python2/3.

API (C and Python are optional)

Raspberry Pi series can use the same demo as they have embedded systems with high compatibility.
The demo has three parts: the underlayer hardware interface, the middle LCD driver, and the upper application.

C

Bottom Layer Hardware Interface

We have carried out the bottom layer package, 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 are in the directory RaspberryPi\c\lib\Config.

```
1. There are three ways for C to drive: BCM2835 library, WiringPi library, and D
ev library respectively.
2. WiringPi library is used by default as there is no external interrupt in BCM2
835 and Dev library.
```

- Data type:

```
#define UBYTE   uint8_t
#define UWORD   uint16_t
```

```
#define UDOUBLE uint32_t
```

- Module initialization and exit:

```
void DEV_Module_Init(void);
void DEV_Module_Exit(void);
Note:
1. Here is how to address GPIO before or after using the LCD.
```

- GPIO read/write

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

- SPI read and write data:

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

Upper Application

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

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

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

名称	修改日期	类型
font8.c	2020/5/20 11:58	C 文件
font12.c	2020/5/20 11:58	C 文件
font12CN.c	2020/6/5 18:57	C 文件
font16.c	2020/5/20 11:58	C 文件
font20.c	2020/5/20 11:58	C 文件
font24.c	2020/5/20 11:58	C 文件
font24CN.c	2020/6/5 19:01	C 文件
fonts.h	2020/5/20 11:58	H 文件

- 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, UWORD Width, UWORD Height, UWORD Rotate, UWORD Color)

Parameters:

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

      Width: 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;
```

- Choose Image buffer: choose Image buffer for creating more image properties. The image buffer can be varied, and you can choose the image you created.

```
void Paint_SelectImage(UBYTE *image)

Parameter:

           image: the name of the image buffer is actually a pointer to its first address.
```

- Set the display position and color of the point in the buffer: here is the most important function in GUI, that is, about how to address the display position and color of the point in the buffer:

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

Parameters:
```

```
        Xpoint: the X position of a point in the image buffer
        Ypoint: Y position of a point in the image buffer
        Color: indicates the Color of the dot
```

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

```
void Paint_Clear(UWORD Color)
Parameters:
        Color: fill Color
```

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

```
void Paint_ClearWindows(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWOR
D 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, and the style of the point.

```
void Paint_DrawPoint(UWORD Xpoint, UWORD Ypoint, UWORD 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, the demo provides 8 size points by defau
lt.
              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 the line: In the image buffer, draw a line from (Xstart, Ystart) to (Xend, Yend), you can choose the color, the width, and the style of the line.

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Co
lor, LINE_STYLE Line_Style ,  LINE_STYLE Line_Style)
Parameters:
        Xstart: the x-starting coordinate of a line
        Ystart: the y-starting coordinate of 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  ,         // 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 straigh
t 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, UWO
RD 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 provi
des 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, and write an Ascii character, you can select Ascii visual character library, font foreground color, and font background color.

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

Parameters:
      Xstart: the x-coordinate of the left vertex of a character
      Ystart: the Y-coordinate of the left vertex of a character
      Ascii_Char: indicates the Ascii character
```

```
      Font: Ascii visual character library, in the Fonts folder the demo provide
s 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, and write a string of English characters, you can choose Ascii visual character library, font foreground color, or 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 the demo provide
s 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 character font, font foreground color, and font background color of the GB2312 encoding.

```
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 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, 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, UWOR
D Digit, UWORD Color_Foreground,   UWORD Color_Background)

Parameters:

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

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

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

      Digit: It's a decimal number

      Font: Ascii visual character library, in the Fonts folder the demo provide
s 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 numbers with decimals: In the image cache, at (Xstart Ystart) as the left vertex, write a string of numbers that can be numbered 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 Decima
l_Point, sFONT* Font, UWORD Color_Foreground, UWORD  Color_Background);

Parameter:

      Xstart: The left vertex X coordinate of the character

      Ystart: The left vertex Y coordinate of the font

      Nummber: The numbers shown, saved here using a double, are common enough
```

```
        Decimal_Point: Displays a few digits after the decimal point
      Font: Ascii code visual character font library, the following fonts are av
ailable in the Fonts folder:
              font8: A font of 5*8
              font12: 7*12 font
              font16: The font of 11*16
              font20: A font of 14*20
              font24: The font of 17*24
      Color_Foreground: Font color
      Color_Background: Background color
```

- Display time: in the image buffer, use (Xstart Ystart) as the left vertex, For 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, A time structure is defined here, as long as the hour
s, minutes, and seconds are passed to the parameters;
      Font: Ascii visual character library, in the Fonts folder the demo provide
s 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, UWORD Xstart, UWORD Ystart)
parameter:
```

```
path: the relative path of the BMP image

  Xstart: The X coordinate of the left vertex of the image, generally 0 is passe
d by default

  Ystart: The Y coordinate of the left vertex of the picture, generally 0 by def
ault
```

User Demo Test

The above three chapters have introduced three demo structures of Linux, here we introduce the user test demo.

For Raspberry Pi, in the directory: RPI\c\example; If you need to run the test demo, you need to run mian demo.

Under the Linux command mode, you need to execute the following commands again:

```
make clean

make -j

sudo ./main
```

Python (For Raspberry Pi)

Applicable for python and python3.

Python is as complicated as C.

Raspberry Pi: RPI\python\lib\

lcdconfig.py

- Module initialization and how to exit:

```
def module_init()

def Touch_module_init()

def module_exit()

Note:

1. Here is how to address GPIO before or after using the LCD.

2. The module_init() and Touch_module_init() functions are automatically called
in the init() function on the LCD and the touch screen respectively, but the mod
ule_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)
```

- I2C read and write data.

```
def i2c_write_byte( Addr, val)
def i2c_read_byte(Addr)
```

python in the following directories:
RPI\python\examples\
Execute the following commands again in the Linux command mode:

```
sudo python 1inch28_LCD_test.py
```

GUI Functions

Python has an image library PIL official library link, it does not need to write code from the logical layer like C and can directly call the image library for image processing. The following will take a 1.54-inch 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 to use. 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 two-element tuple with (40，50) as the left vertex, and use font2, fill is font color, fill = "WHITE" means that the font color is white. Also, you can use fill = (128,255,128), the number in the () corresponds to the RGB value, and then you can accurately control the color you want. The second sentence displays "微雪电子" in Font3, and the font color is white.

- Read local pictures:

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

The parameter is the image path.

- Other functions:

Python's image library is very powerful, if you need to implement more, you can learn on the website http://effbot.org/imagingbook pil

Raspberry Pi Pico

Hardware Connection

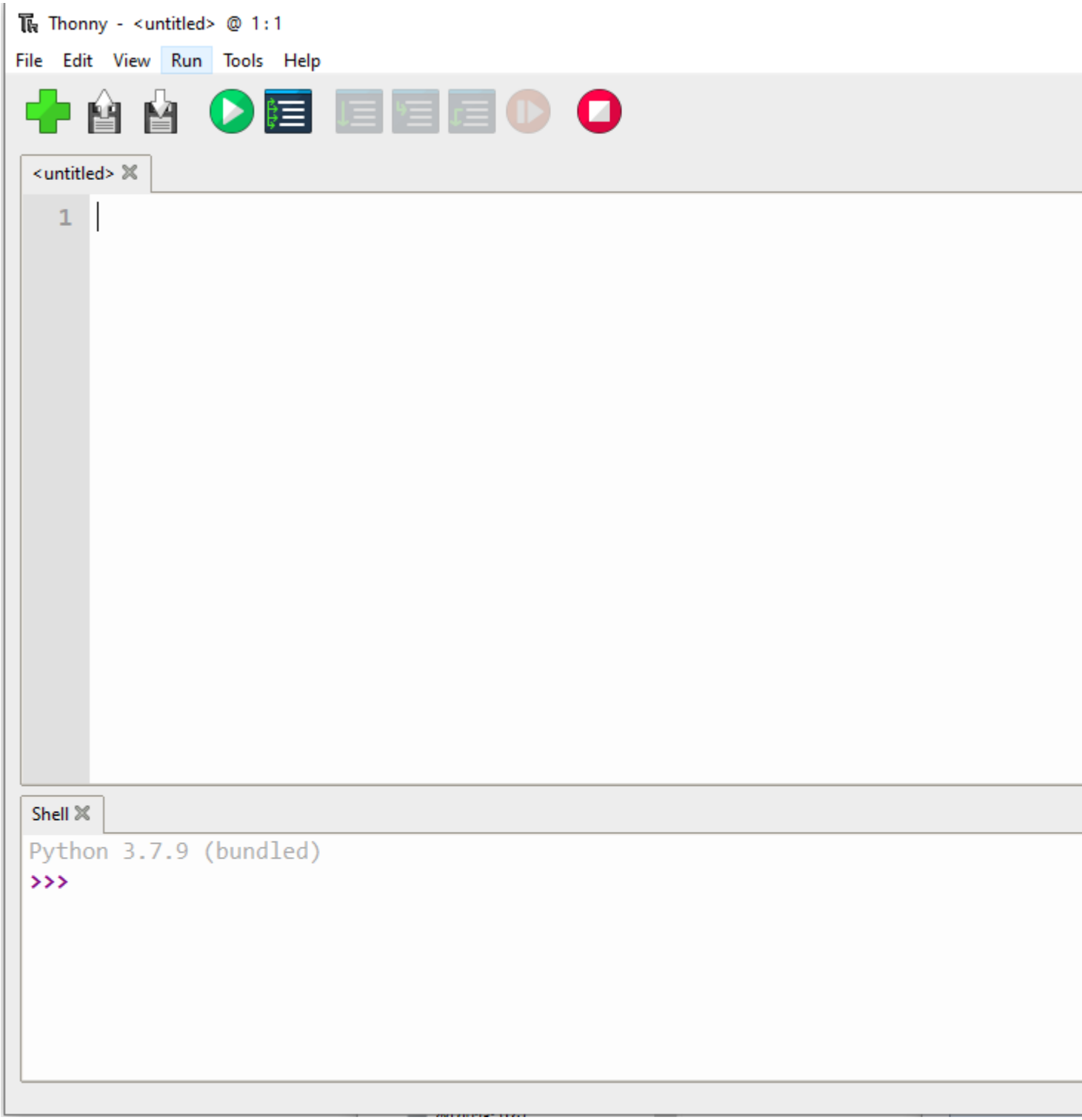
Pin	Raspberry Pi Pico
VCC	3.3V
GND	GND
MISO	GP12
MOSI	GP11
SCLK	GP10
LCS_CS	GP9
LCS_DC	GP14
LCS_RST	GP8
LCS_BL	GP15
TP_SDA	GP6
TP_SCL	GP7
TP_INT	GP17
TP_RST	GP16

C/C++ Development Environment Installation

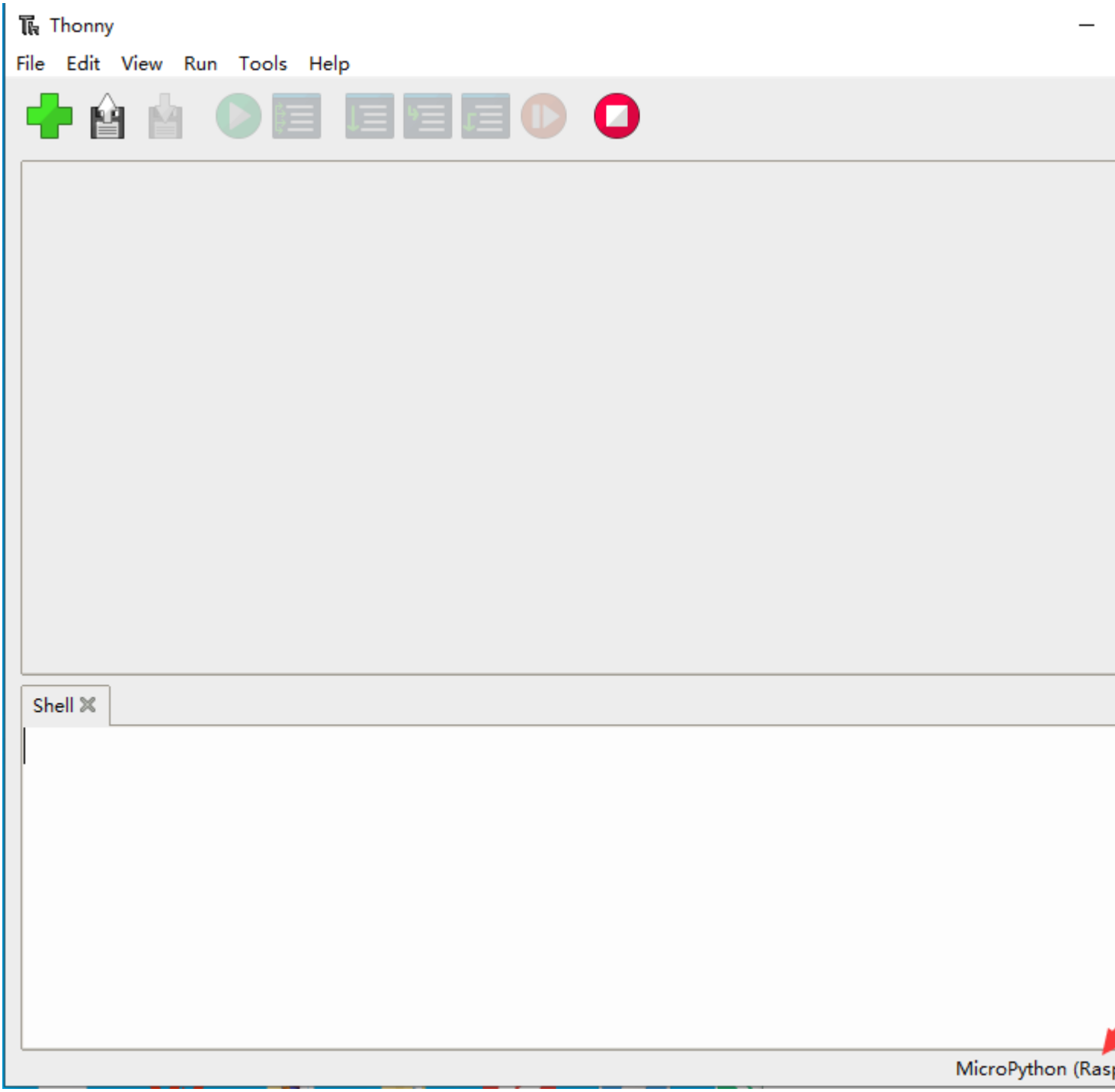
- Before using the demos and tutorials, you need to set up a development environment and learn the basic methods.
- Pico-Get-Start-Windows
- Pico-Get-Start-RPI

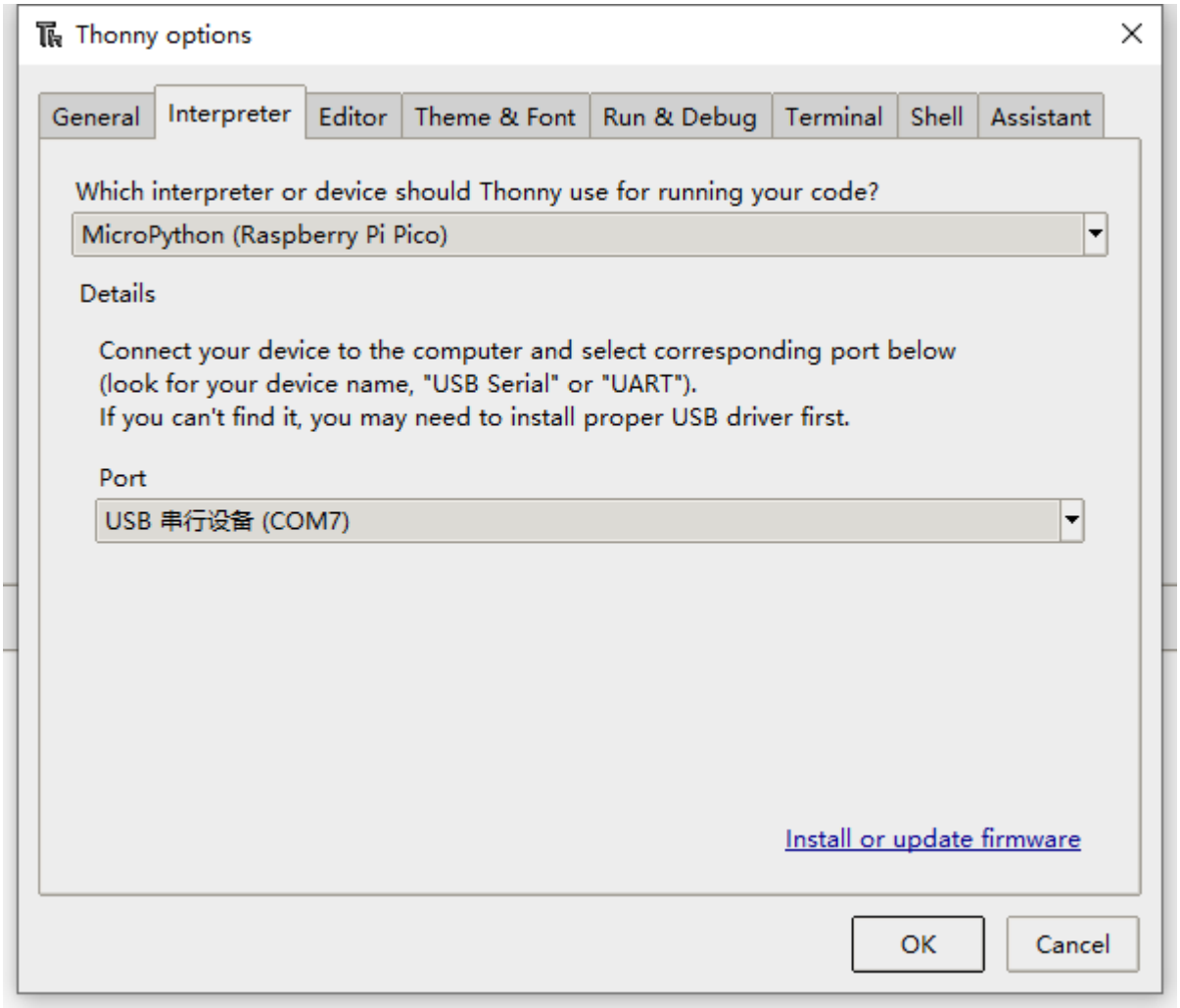
Software Environment Debugging

- In order to facilitate the development of Pico boards using MicroPython on the computer, it is recommended to download Thonny IDE.
- Download Thonny IDE and install it by steps.
    - Thonny IDE download link (windows)
    - Thonny website
- After installing, please configure the language and the environment for the first time. Note that we should choose the Raspberry Pi option in the board environment.



- Configure the Micrpython environment and select the Pico port.
  - First connect the Raspberry Pi Pico to the computer, left-click on the configuration environment option in the lower right corner of Thonny--> select configure an interpreter.
  - In the pop-up window bar, select MicroPython (Raspberry Pi Pico), and select the corresponding port.


Thonny
File Edit View Run Tools Help
Shell
MicroPython (Ras



- Click OK to return to the main interface of Thonny, download the firmware library to Pico, and then click the stop button to display the currently used environment in the Shell window.
- Pico download firmware library method in windows: Press and hold the BOOT button and connect to the computer, release the BOOT button, and a removable disk will appear on the computer and copy the firmware library into it.

Raspberry Pi

- Open the Raspberry Pi and execute:

```
sudo apt-get install p7zip-full
cd ~
sudo wget https://files.waveshare.com/upload/3/3e/1.28inch_Touch_LCD_Pico.zip
unzip 1.28inch_Touch_LCD_Pico.zip
cd ~/1.28inch_Touch_LCD_Pico
cd c/build/
```

How to Use Demos

C

- The following tutorial is operated on **Raspberry Pi**. As cMake features multi-platforms and portability, it can be sucessufully compiled on the PC. The operation my be a little different, you need to judge it by yourselves.

Compile, please make sure it is in the c directory:

```
cd ~/1.28inch_Touch_LCD_Pico/c/
```

Create and enter build directory, add SDK: ../../pico-sdk is your SDK directory. In our example demo, there is "build", you can directly enter:

```
cd build
export PICO_SDK_PATH=../../pico-sdk
(Note: please correctly write your SDK path)
```

Execute cmake and automatically generate Makefile.

```
cmake ..
```

Execute make to generate the executable file, you may wait for a long time as it is the first time to be compiled.

```
make -j9
```

After compiling, uf2 file will generate. Press the button on the Pico board, Pico can connect to the USB port of the Raspberry Pi via a Micro USB cable, and then release the buttons. After connecting, Raspberry Pi will automatically identify a movable disk (RPI-RP2), and copy main.uf2 in the build file to the recognizable movable disk (RPI-RP2).

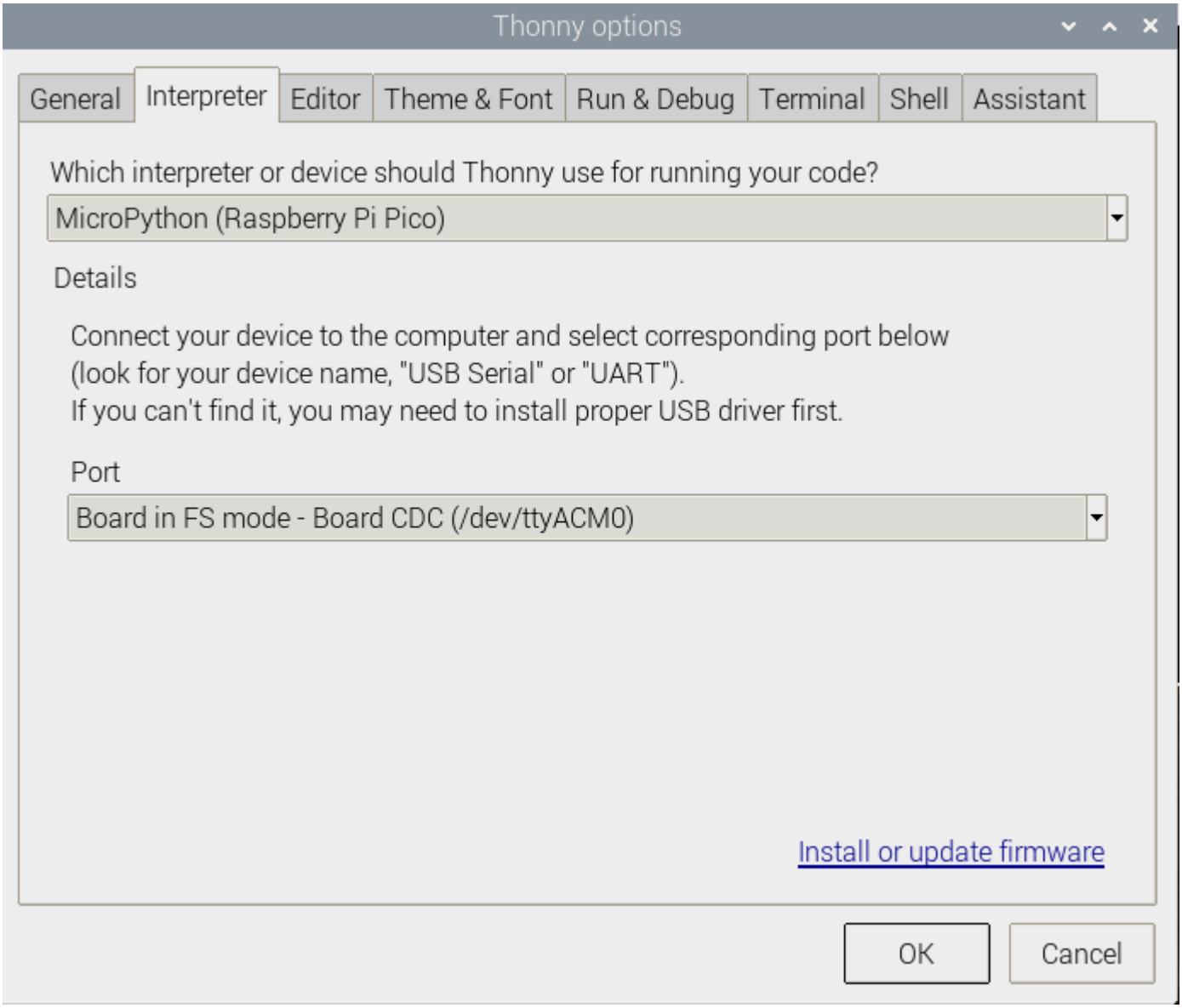
```
cp main.uf2 /media/pi/RPI-RP2/
```

Python

- 1. On the Raspberry Pi, copy ~/1.28inch_Touch_LCD_Pico/python/rp2-pico-20221125-v1.19.1.uf2 to Pico.

- 2. Open Thonny IDE on the Raspberry Pi (click Raspberry Pi logo -> Programming -> Thonny Python IDE), and you can check the version information in Help -> About Thonny.

To ensure your version that your version includes the package supported Pico, you can click Tools -> Options... -> Interpreter, choose MicroPython (Raspberry Pi Pico and ttyACM0 port). As shown below:



If your current Thonny version has no package supporting Pico, you can enter the following commands to update Thonny IDE.

```
sudo apt upgrade thonny
```

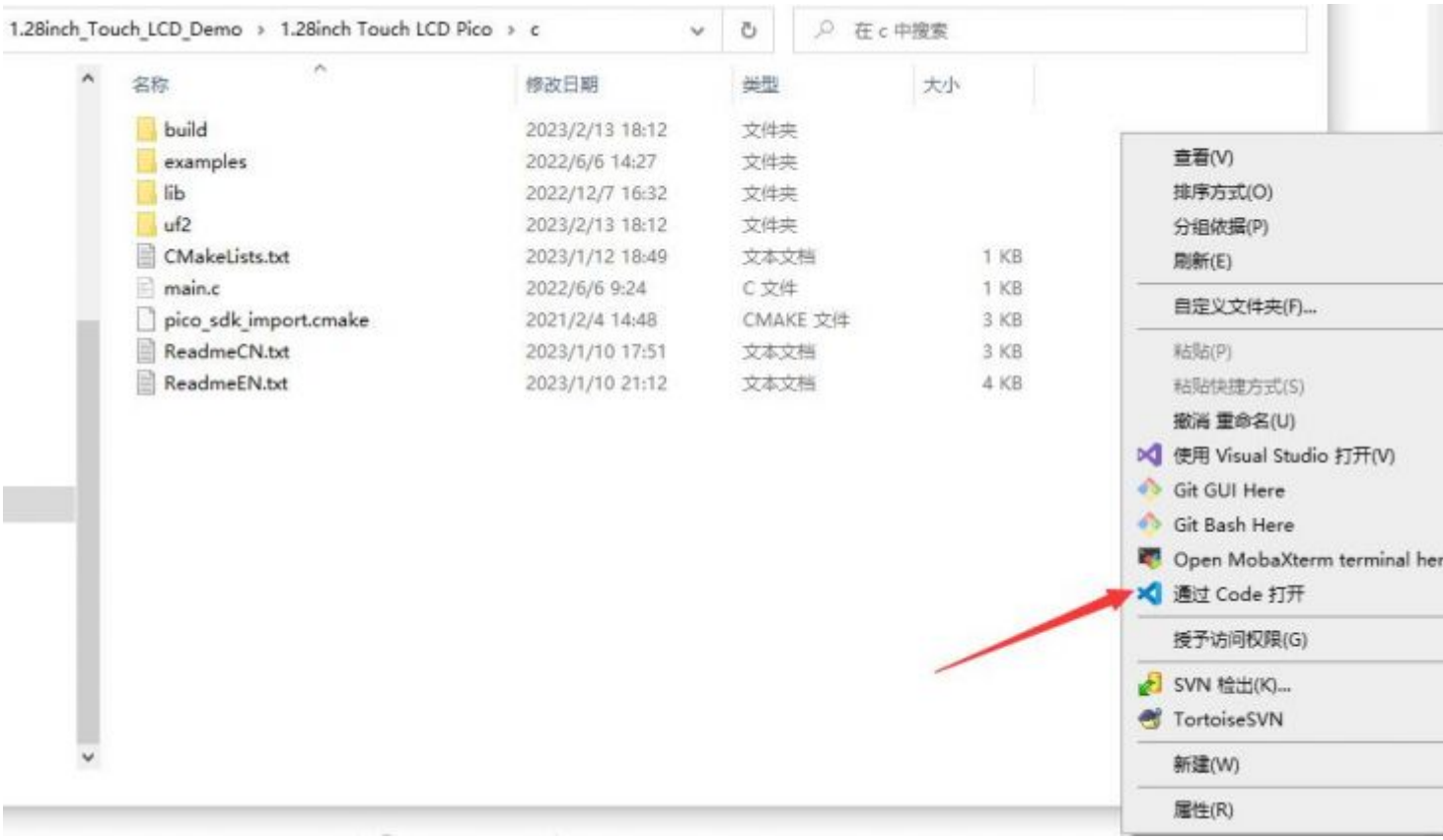
3. Click File->Open...->~/1.28inch Touch LCD Pico/python/1.28inch_Touch_LCD.py, and then run the script.

Windows

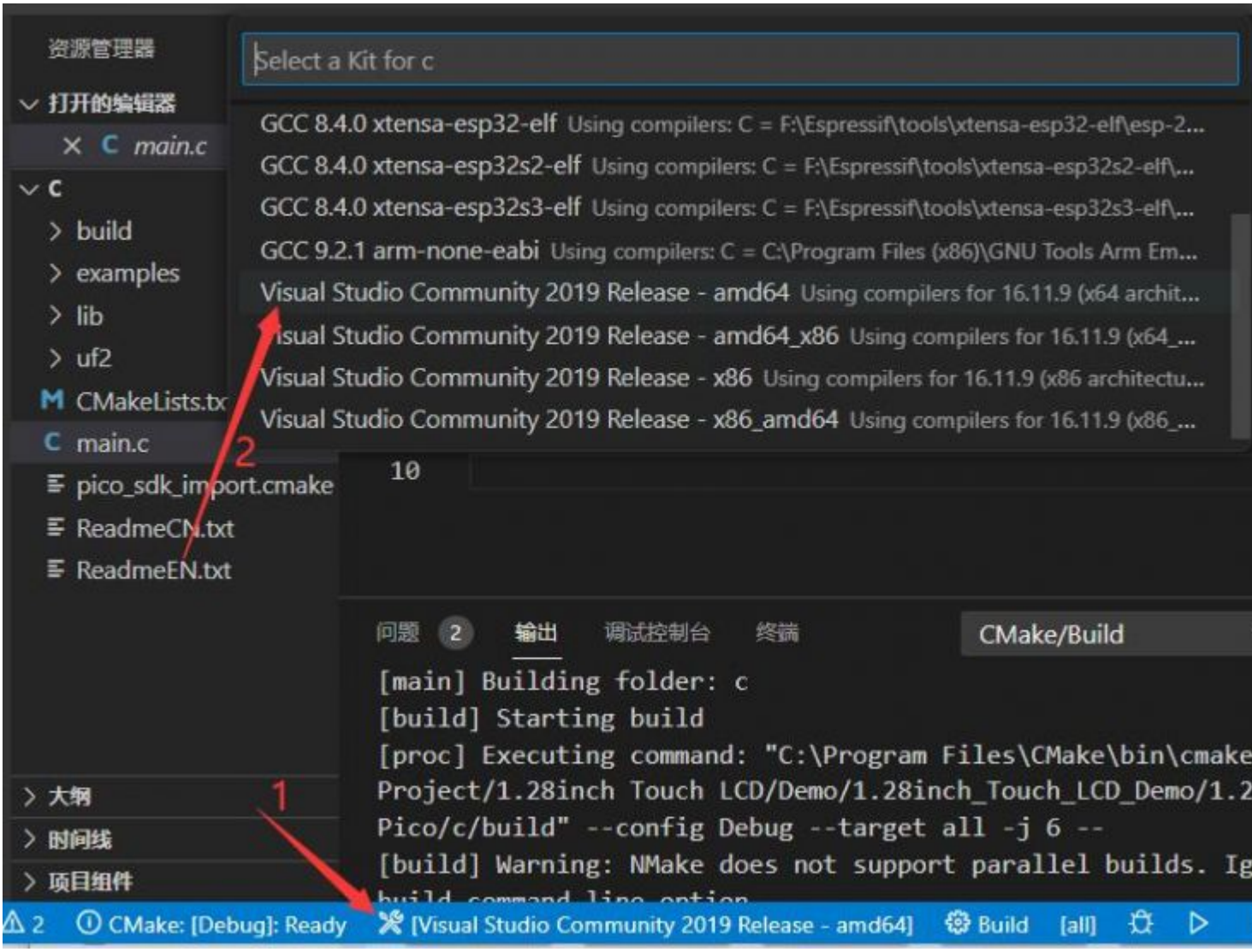
- Click to download 1.28inch Touch LCD Pico.zip, decompress the package and go to the 1.28-inch Touch LCD Pico folder.

C

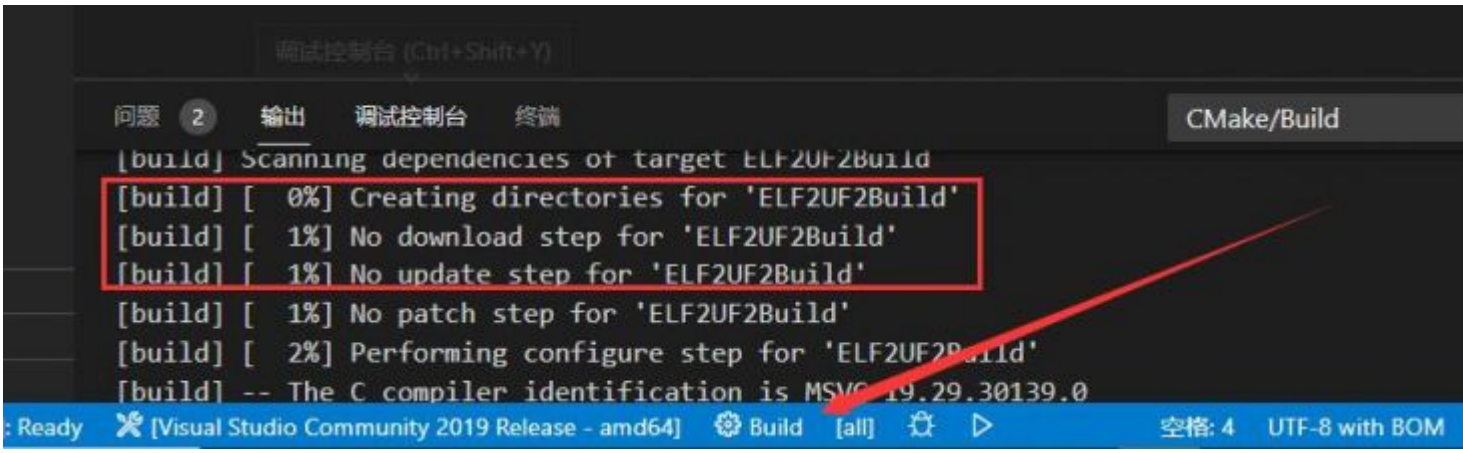
- After entering 1.28inch_Touch_LCD_Pico\c, you can open the project with vs code.



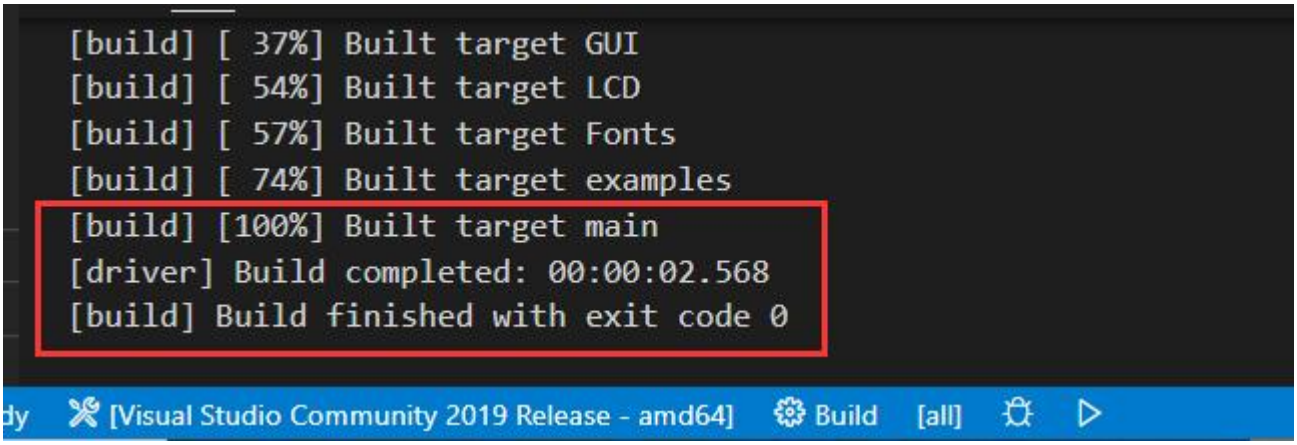
- Choose the Compiler.



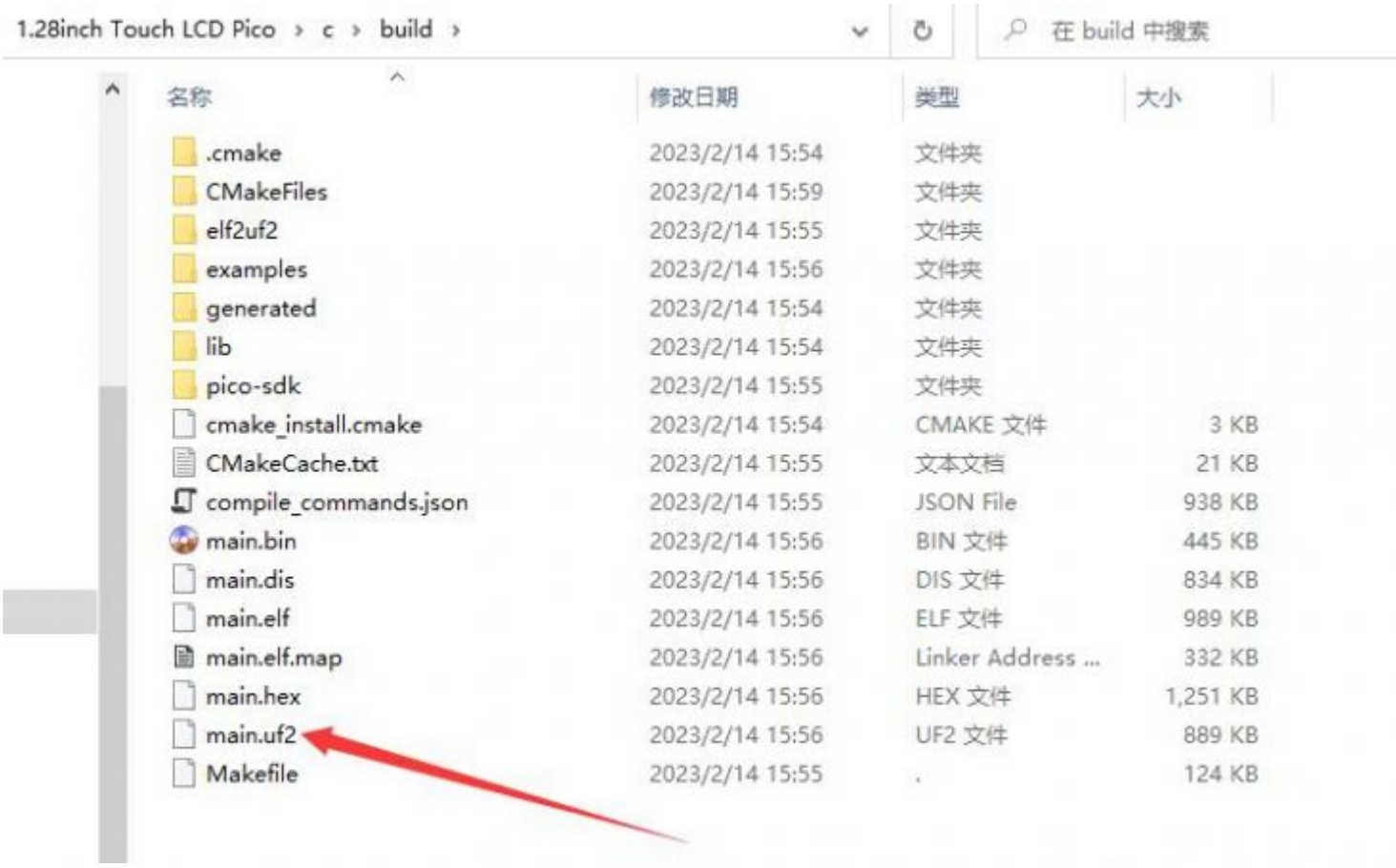
- Start to compile.



- Finish.



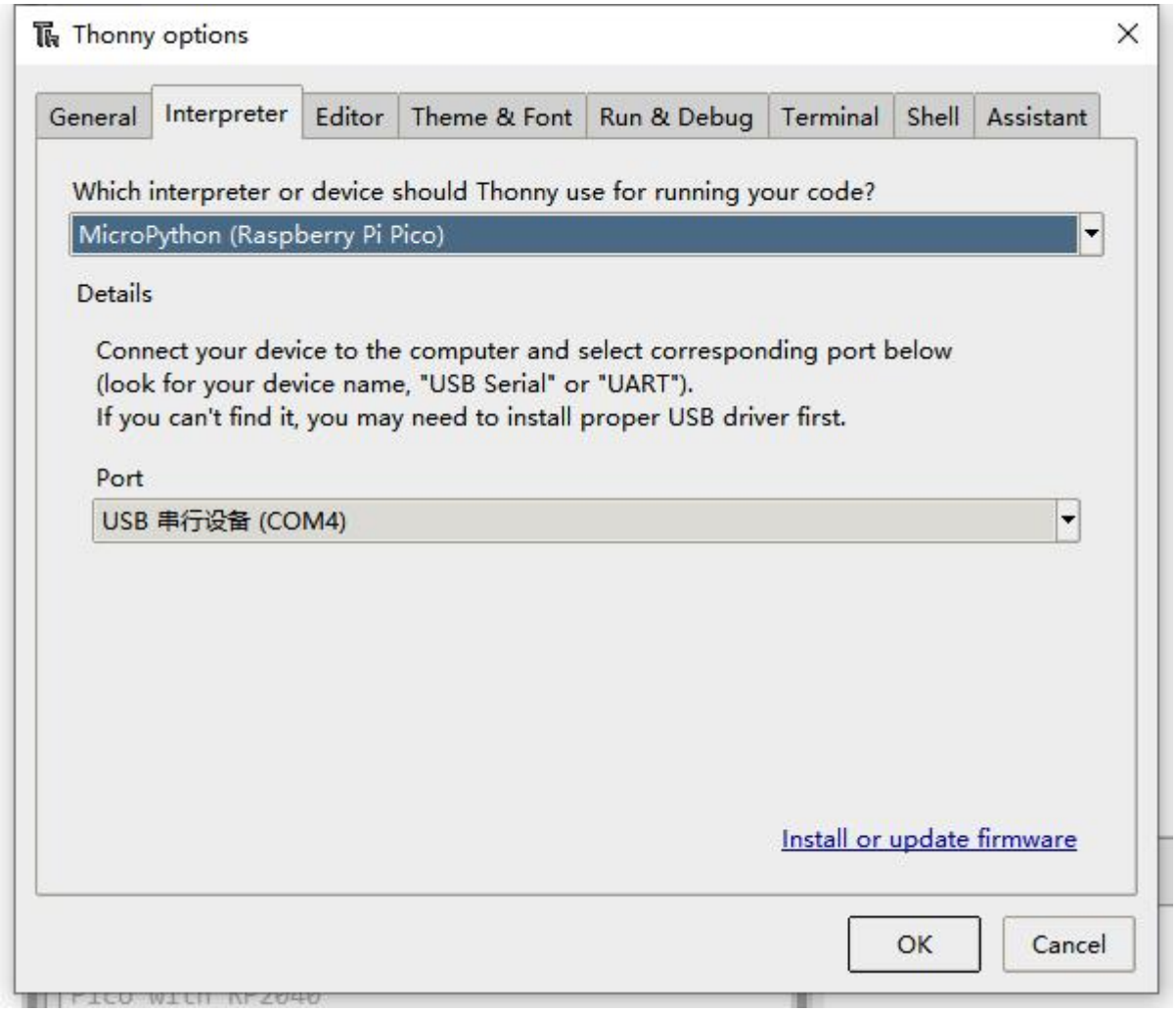
- Copy the main.uf2 filr in build to Pico, and then it can automatically run the demo.



Python

1. Press the BOOTSET button on the Pico and connect the pico to the USB port of the computer with a Micro USB cable. Release the button when the computer identifies a movable disk (RPI-RP2).
2. Copy rp2-pico-20221125-v1.19.1.uf2 file in the python directory to the recognizable movable disk (RPI-RP2).
3. Open Thonny IDE (Note: please use the latest version of Thonny, otherwise there is no Pico supporting package. Currently, the newest version in Windows is v3.3.3.)

4. Click Tool -> Setting ->Explainer. select the Pico and the corresponding port as shown below:



5. File -> Open -> 1.28inch_Touch_LCD.py, click to run, the effect is shown as below.

We provide a simple demo for you...

Demo Analysis

C

Bottom Hardware Interface

We package the bottom hardware layer for easily porting to the different hardware platforms.

DEV_Config.c(.h) in the directory:...\c\lib\Config

- Data type：

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

- Module initialize and exit:

```
void DEV_Module_Init(void);
void DEV_Module_Exit(void);
Note：
1. The functions above are used to initialize the display or exit handle.
```

- GPIO write/read：

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

- SPI transmits data.

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

- I2C writes and reads data.

```
void DEV_I2C_Write_Byte(uint8_t addr, uint8_t reg, uint8_t Value);
void DEV_I2C_Write_nByte(uint8_t addr, uint8_t *pData, uint32_t Len);
uint8_t DEV_I2C_Read_Byte(uint8_t addr, uint8_t reg);
void DEV_I2C_Read_nByte(uint8_t addr, uint8_t reg, uint8_t *pData, uint32_t Len)
;
```

Application functions

We provide basic GUI functions for testing, like draw point, line, string, and so on. The GUI function can be found in directory:..\c\lib\GUI\GUI_Paint.c(.h).

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

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

- Create a new image, you can set the image name, width, height, rotate angle, and color.

```
void Paint_NewImage(UWORD *image, UWORD Width, UWORD Height, UWORD Rotate, UWORD Color, UWORD Depth)
Parameter:
      image: Name of the image buffer, this is a pointer;
      Width: Width of the image;
      Height: Height of the image;
      Rotate: Rotate the angle of the Image;
      Color: The initial color of the image;
      Depth: Depth of the color
```

- Select image buffer: You can create multiple image buffers at the same time and select the certain one and draw by this function.

```
void Paint_SelectImage(UBYTE *image)
Parameter:
      image: The name of the image buffer, this is a pointer;
```

- Set point displays position and color in the cache: Here is the core function of the GUI, the processing point displays position and color in the cache.

```
void Paint_SetPixel(UWORD Xpoint, UWORD Ypoint, UWORD Color)
Parameter:
      Xpoint: Point X position in the image cache
      Ypoint: Point Y position in the image cache
```

```
      Color: The color in which the dot is displayed
```

- Image cache fill color: fill the image cache to a certain color, generally as the role of screen whitening

```
void Paint_Clear(UWORD Color)
Parameter:
      Color: The filling color
```

- Image cache part window fill color: fill a certain part of the image cache window with a certain color, generally as the role of window whitening, often used for the display of time, refreshing white for one second.

```
void Paint_ClearWindows(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWOR
D Color)
Parameter:
      Xstart: The X start coordinate of the window
      Ystart: The Y-start coordinate of the window
      Xend: The X endpoint coordinates of the window
      Yend: The Y-end coordinates of the window
      Color: The color of the fill
```

- Draw point: Draw a point at the position（Xpoint, Ypoint）of the image buffer, you can configure the color, size, and style.

```
void Paint_DrawPoint(UWORD Xpoint, UWORD Ypoint, UWORD Color, DOT_PIXEL Dot_Pixe
l, DOT_STYLE Dot_Style)
Parameter:
      Xpoint: X-axis position of the point.
      Ypoint: Y-axis position of the point
      Color: Color of the point
      Dot_Pixel: Size of the point, 8 sizes are available.
              typedef enum {
                     DOT_PIXEL_1X1  = 1,  // 1 x 1
                     DOT_PIXEL_2X2  ,            // 2 X 2
                     DOT_PIXEL_3X3  ,            // 3 X 3
                     DOT_PIXEL_4X4  ,            // 4 X 4
                     DOT_PIXEL_5X5  ,            // 5 X 5
```

```
                    DOT_PIXEL_6X6  ,            // 6 X 6
                    DOT_PIXEL_7X7  ,            // 7 X 7
                    DOT_PIXEL_8X8  ,            // 8 X 8
            } DOT_PIXEL;
      Dot_Style: The style of the point, defines the extended mode of the point.
            typedef enum {
               DOT_FILL_AROUND  = 1,
               DOT_FILL_RIGHTUP,
            } DOT_STYLE;
```

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

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Co
lor, LINE_STYLE Line_Style , LINE_STYLE Line_Style)
Parameter:
      Xstart: Xstart of the line
      Ystart: Ystart of the line
      Xend: Xend of the line
      Yend: Yend of the line
      Color: Color of the line
      Line_width: Width of the line, 8 sizes are available.
            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: Style of the line, Solid or Dotted.
            typedef enum {
                    LINE_STYLE_SOLID = 0,
                    LINE_STYLE_DOTTED,
```

```
            } LINE_STYLE;
```

- Draw a rectangle: Draw a rectangle from (Xstart, Ystart) to (Xend, Yend), you can configure the color, width, and style.

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

Parameter:

      Xstart: Xstart of the rectangle.

      Ystart: Ystart of the rectangle.

      Xend: Xend of the rectangle.

      Yend: Yend of the rectangle.

      Color: Color of the rectangle

      Line_width: The width of the edges. 8 sizes are available.

            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: Style of the rectangle, empty or filled.

            typedef enum {

                  DRAW_FILL_EMPTY = 0,

                  DRAW_FILL_FULL,

            } DRAW_FILL;
```

- Draw circle: Draw a circle in the image buffer, use (X_Center Y_Center) as the center and Radius as the radius. You can configure the color, width of the line and style of a circle.

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

Parameter:

      X_Center: X-axis of center
```

```
    Y_Center: Y-axis of center
    Radius: radius of the circle
    Color: The color of the circle
    Line_width: The width of the arc, 8 sizes are available.
        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: Style of the circle: empty or filled.
        typedef enum {
            DRAW_FILL_EMPTY = 0,
            DRAW_FILL_FULL,
        } DRAW_FILL;
```

- Show Ascii character: Show a character in (Xstart, Ystart) position, you can configure the font, foreground, and background.

```
void Paint_DrawChar(UWORD Xstart, UWORD Ystart, const char Ascii_Char, sFONT* Fo
nt, UWORD Color_Foreground, UWORD Color_Background)
Parameter:
    Xstart: Xstart of the character
    Ystart: Ystart of the character
    Ascii_Char: Ascii char
    Font: five fonts are available：
        font8：5*8
        font12：7*12
        font16：11*16
        font20：14*20
        font24：17*24
    Color_Foreground: foreground color
```

```
      Color_Background: background color
```

- Draw English string: In (Xstart Ystart) as the left vertex, write a string of English characters, you can choose Ascii code visual character library, font foreground color, or font background color.

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

Parameter:

      Xstart: Xstart of the string

      Ystart: Ystart of the string

      pString: String

      Font: five fonts are available：

             font8：5*8

             font12：7*12

             font16：11*16

             font20：14*20

             font24：17*24

      Color_Foreground: foreground color

      Color_Background: background color
```

- Draw Chinese string: Draw Chinese string at (Xstart Ystart) of the image buffer. You can configure fonts (GB2312), foreground, and background.

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

Parameter:

      Xstart: Xstart of string

      Ystart: Ystart of string

      pString: string

      Font: GB2312 fonts, two fonts are available

             font12CN：ascii 11*21，Chinese 16*21

             font24CN：ascii 24*41，Chinese 32*41

      Color_Foreground: Foreground color

      Color_Background: Background color
```

- Draw number: In the image buffer, at (Xstart Ystart) as the left vertex, write a string of numbers, you can choose the Ascii font, font foreground color, or font background color.

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

Parameter:

      Xstart: Left vertex X coordinate

      Ystart: Left vertex Y coordinate

      Nummber: The displayed number is stored in a 32-bit int type, which can be
displayed up to 2147483647.

              font8：5*8

              font12：7*12

              font16：11*16

              font20：14*20

              font24：17*24

        Digit: Display decimal places

      Color_Foreground: Foreground color

      Color_Background: Background color
```

- Display time: In (Xstart Ystart) as the left vertex, display for a period of time, you can choose Ascii code visual character library, font foreground color, and font background color.

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

Parameter:

      Xstart: The left vertex X coordinate of the character

      Ystart: The left vertex Y coordinate of the character

      pTime: The displayed time, here defines a time structure, just pass the nu
mber of hours, minutes, and seconds to the parameter

      Font: Ascii font, five fonts are available

              font8：5*8

              font12：7*12

              font16：11*16

              font20：14*20

              font24：17*24

      Color_Foreground: Foreground
```

```
    Color_Background: Background
```

MicroPython (Applicable for Raspberry Pi Pico)

- For basic library, you can refer to Pico python sdk.pdf, for more details, you can click here.
- See this link for graphic library usage.

STM32 User Guide

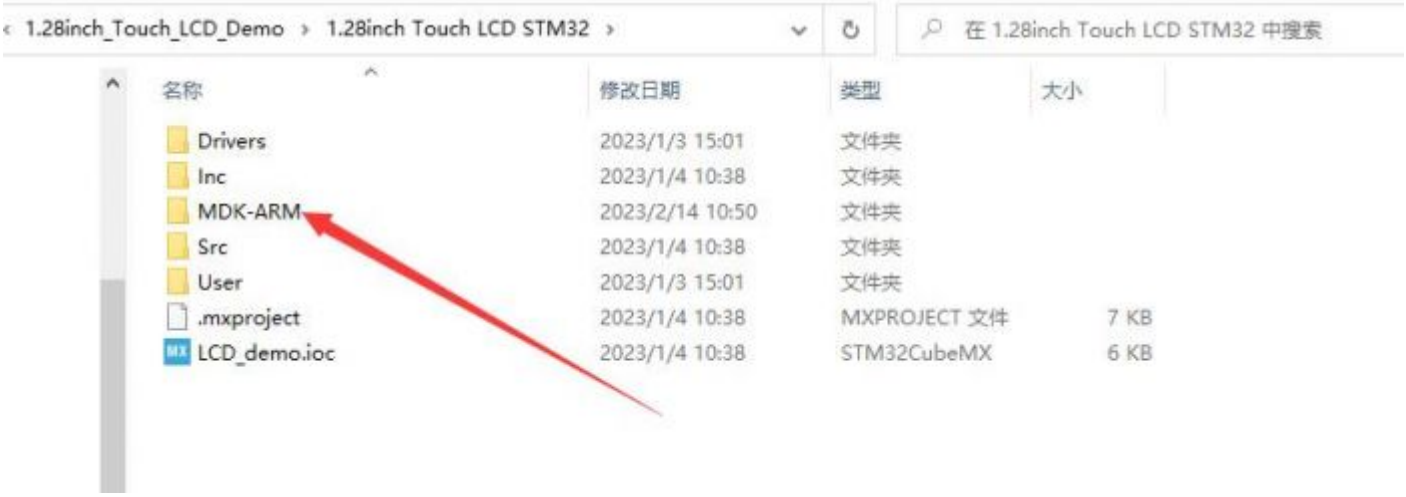
Hardware Connection

Module Pin	STM32F103RB
VCC	3.3V
GND	GND
MISO	PA6
MOSI	PA7
SCLK	PA5
LCS_CS	PB6
LCS_DC	PA8
LCS_RST	PA9
LCS_BL	PC7
TP_SDA	PB9
TP_SCL	PB8
TP_INT	PB10
TP_RST	PA10

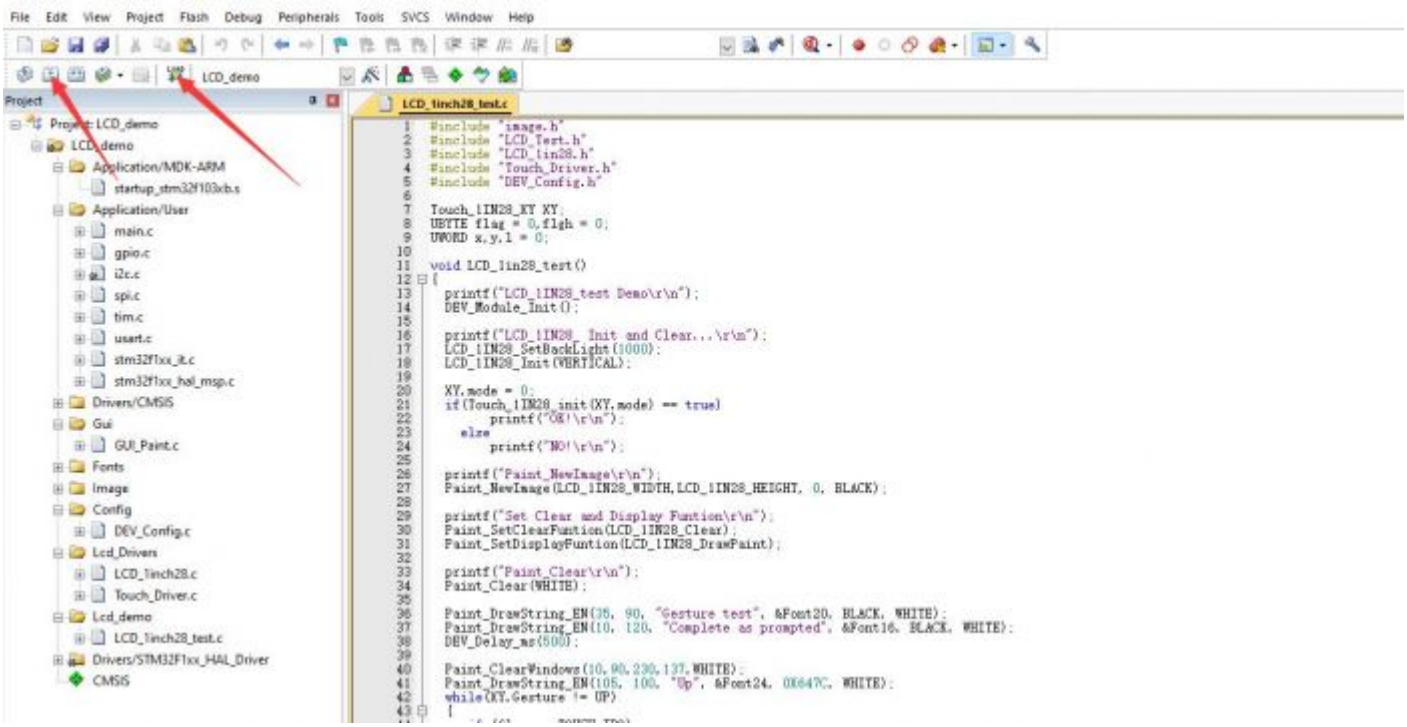
Demo Download

This demo is developed by the HAL library.

Click here to download, open LCD_demo.uvprojx in the 1.28-inch Touch LCD STM32/MDK-ARM directory, and then you can see the demo.



Open LCD_1inch28_test.c to see the demo and the compile it again.



Demo Description

Bottom Hardware Interface

- Data Type:

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

- Module initialization and exit:

```
void DEV_Module_Init(void);
void DEV_Module_Exit(void);
Note:
1. Here is how to address GPIO before or after using the LCD.
2. After using DEV_Module_Exit, LCD will close.
```

- GPIO reads and writes data.

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

- SPI reads and writes data:

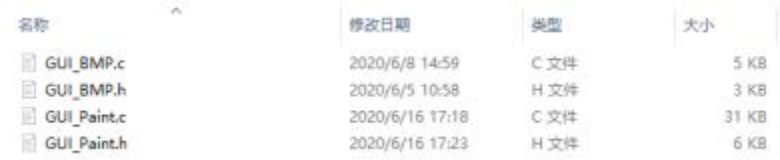
```
void    DEV_SPI_WRITE(UBYTE _dat);
```

- I2C reads and writes data:

```
void I2C_Write_Byte(uint8_t Cmd, uint8_t value);
int I2C_Read_Byte(uint8_t Cmd)
void I2C_Read_nByte(UBYTE Cmd,UBYTE *Buf,UBYTE num)
```

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 scree
n 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;
```

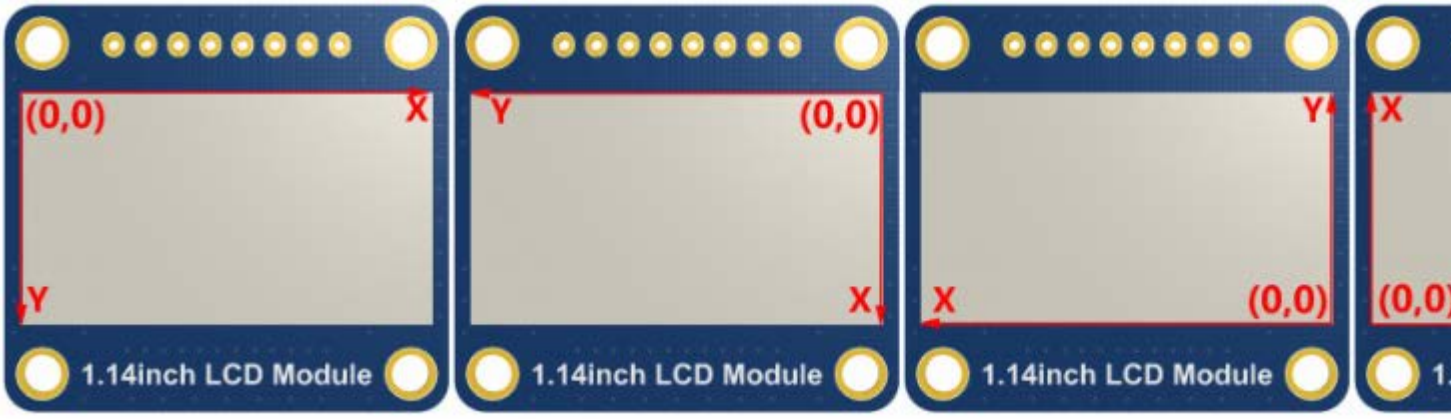
- Select image buffer: the purpose of the selection is that you can create multiple image attributes, an image buffer can exist multiple, and you can select each image you create.

```
void Paint_SelectImage(UBYTE *image)

Parameters:

    Image: the name of the image cache, which is actually a pointer to the first
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, verti
cal 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, UWOR
D 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_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
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, a line from (Xstart, Ystart) to (Xend, Yend), you can choose the color, line width, and line style.

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Co
lor, 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, UWO
RD 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 wi
dths 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* 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

    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
* 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 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
* font, UWORD Color_Foreground,  UWORD Color_Background)

Parameters:

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

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

    PString: string, string is a pointer

    Font: GB2312 encoding character Font library, in the Fonts folder provides 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, UWOR
D 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 foll
owing Fonts:

        Font8: 5*8 font

        Font12: 7*12 font

        Font16: 11*16 font

        Font20: 14*20 font

        Font24: 17*24 font
```

```
    Color_Foreground: Font color
    Color_Background: indicates the background color
```

- Write numbers with decimals: At the left vertex of (Xstart Ystart), write a string of numbers with decimals. You can select Ascii Visual Character library font foreground color font background color.

```
void Paint_DrawFloatNum(UWORD Xpoint, UWORD Ypoint, double Nummber, UBYTE Decima
l_Point, 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

        Nummber: the number displayed here is saved as a double

          Decimal_Point: Display the number after the decimal point

        Font: Ascii visual character font library, the following Fonts are prov
ided 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, 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 as a good time structure, as long as the h
our, minute, and second bits of data to the parameter;

    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
```

Arduino User Guide

Arduino IDE Installation Tutorial

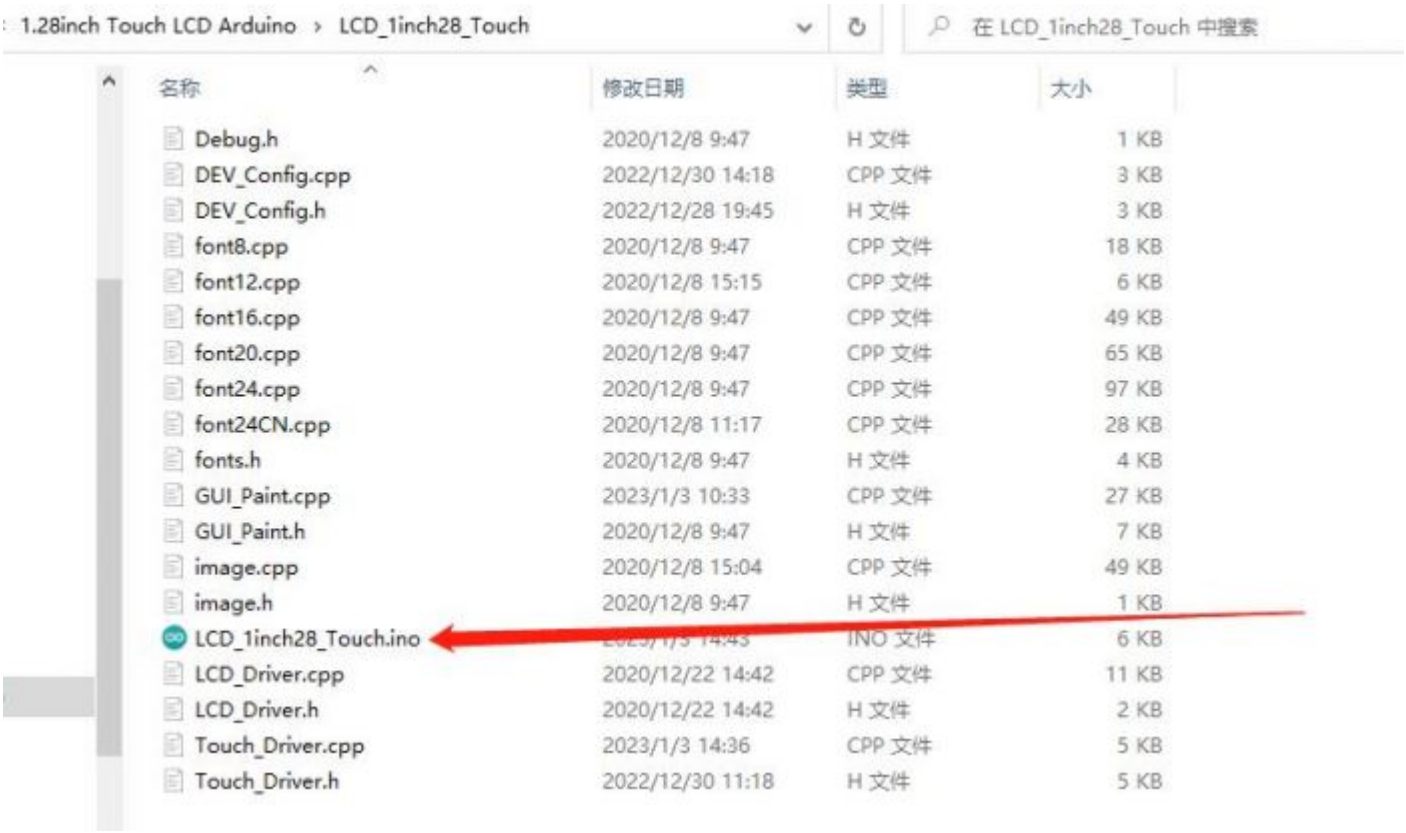
- Arduino IDE Installation Tutorial

Hardware Connection

Module Pin	Arduino uno
VCC	5V
GND	GND
MISO	12
MOSI	11
SCLK	13
LCS_CS	10
LCS_DC	7
LCS_RST	8
LCS_BL	9
TP_SDA	SDA
TP_SCL	SCL
TP_INT	3
TP_RST	4

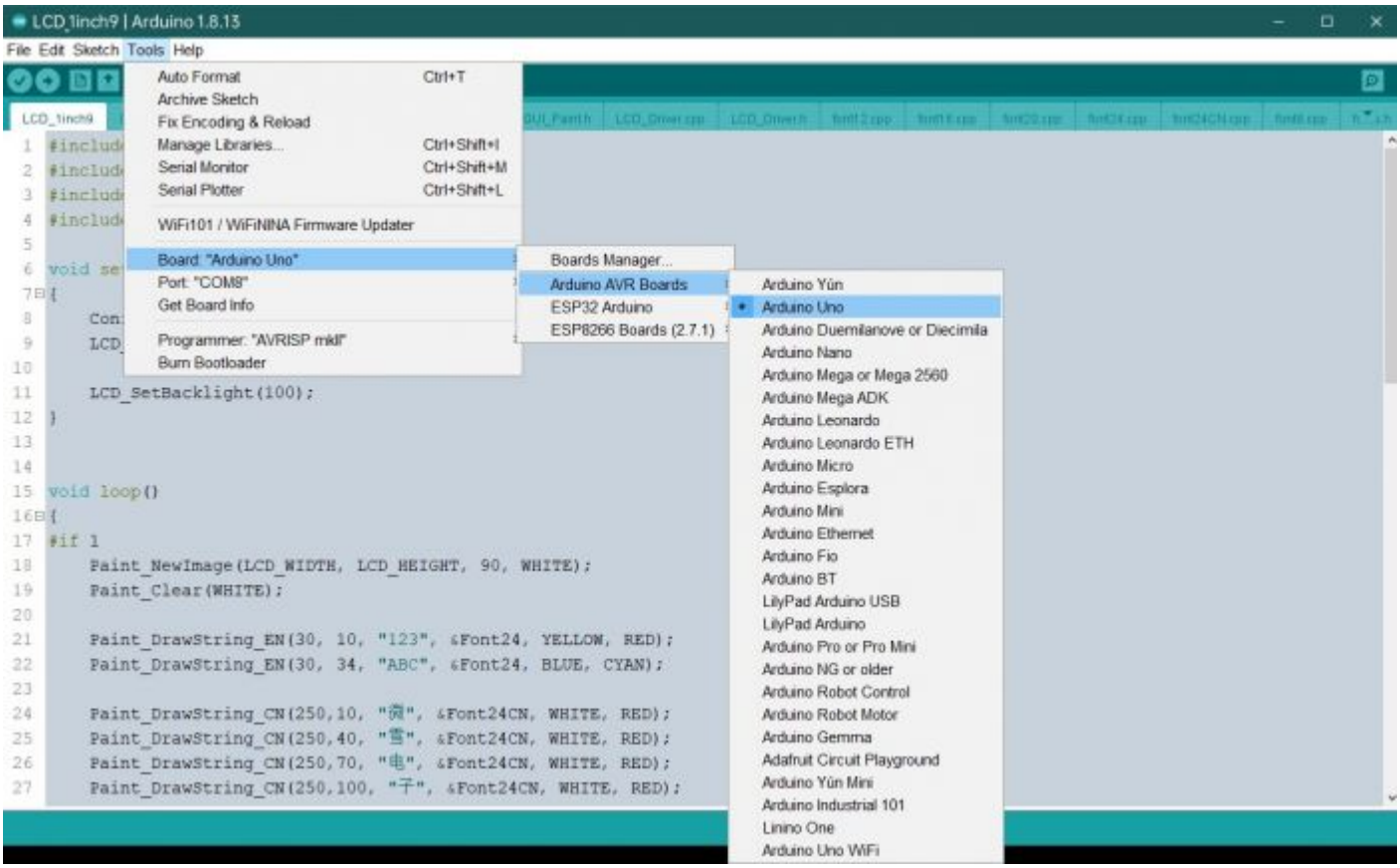
Run the Demo

Click to download the demo, and then decompress it. The demo is in 1.28-inch Touch LCD Arduino\LCD_1inch28_Touch.

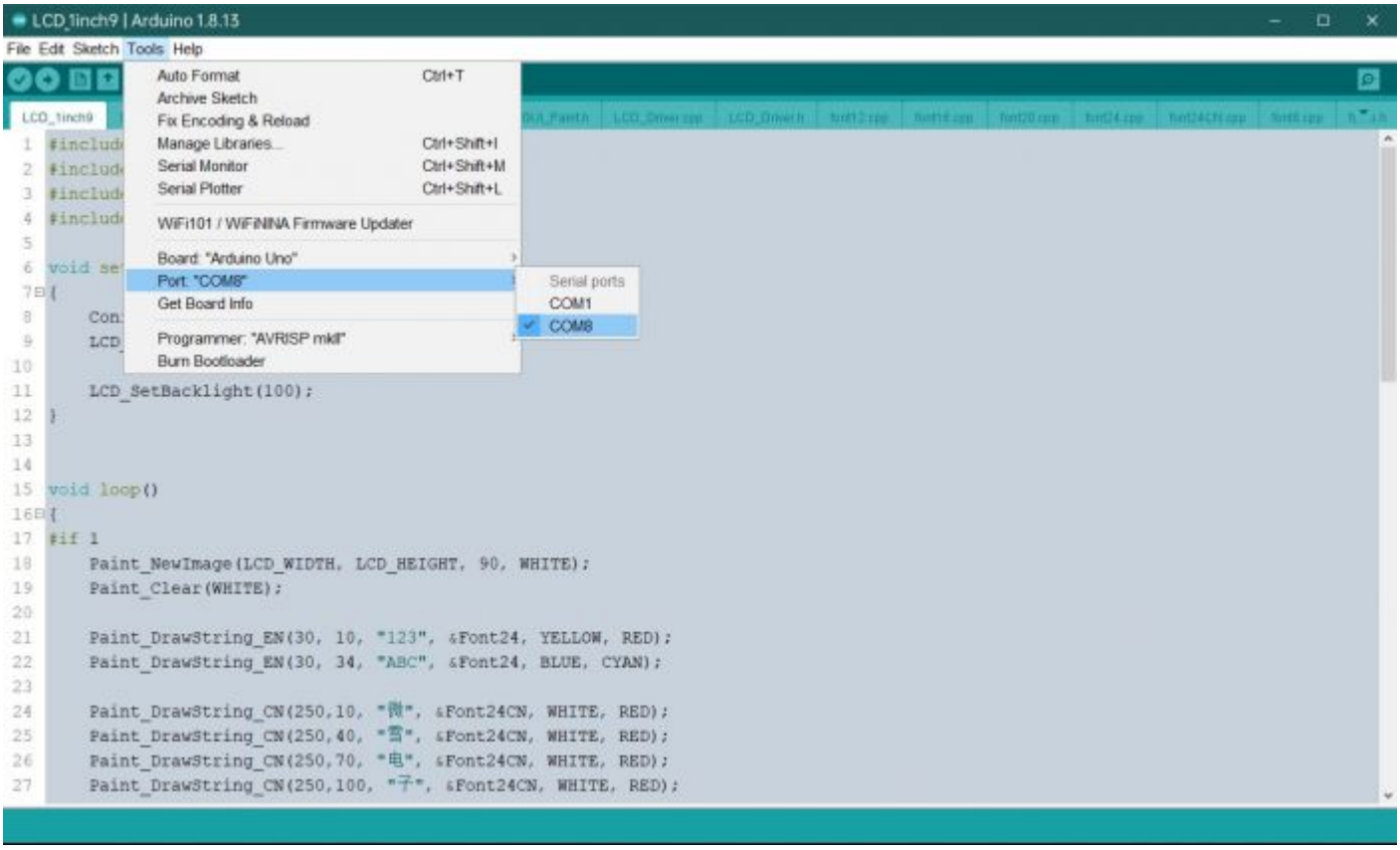


Insall Arduino IDE and then run LCD_1inch28_Touch.ino file.

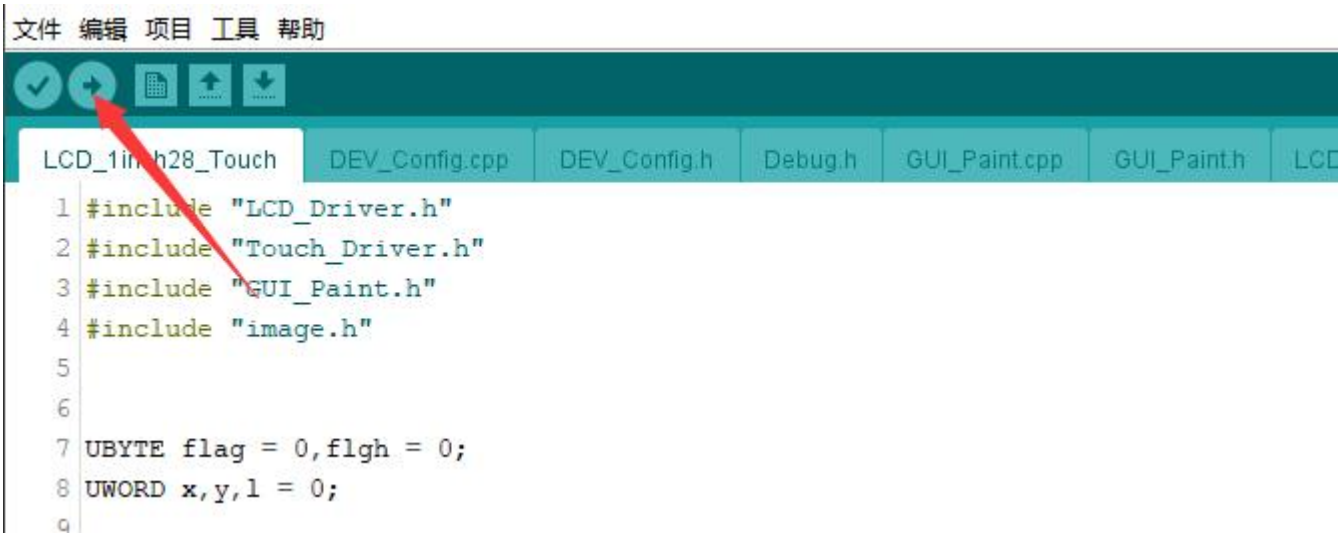
Open the demo, and select the development board as Arduino UNO.



Select the corresponding COM port.



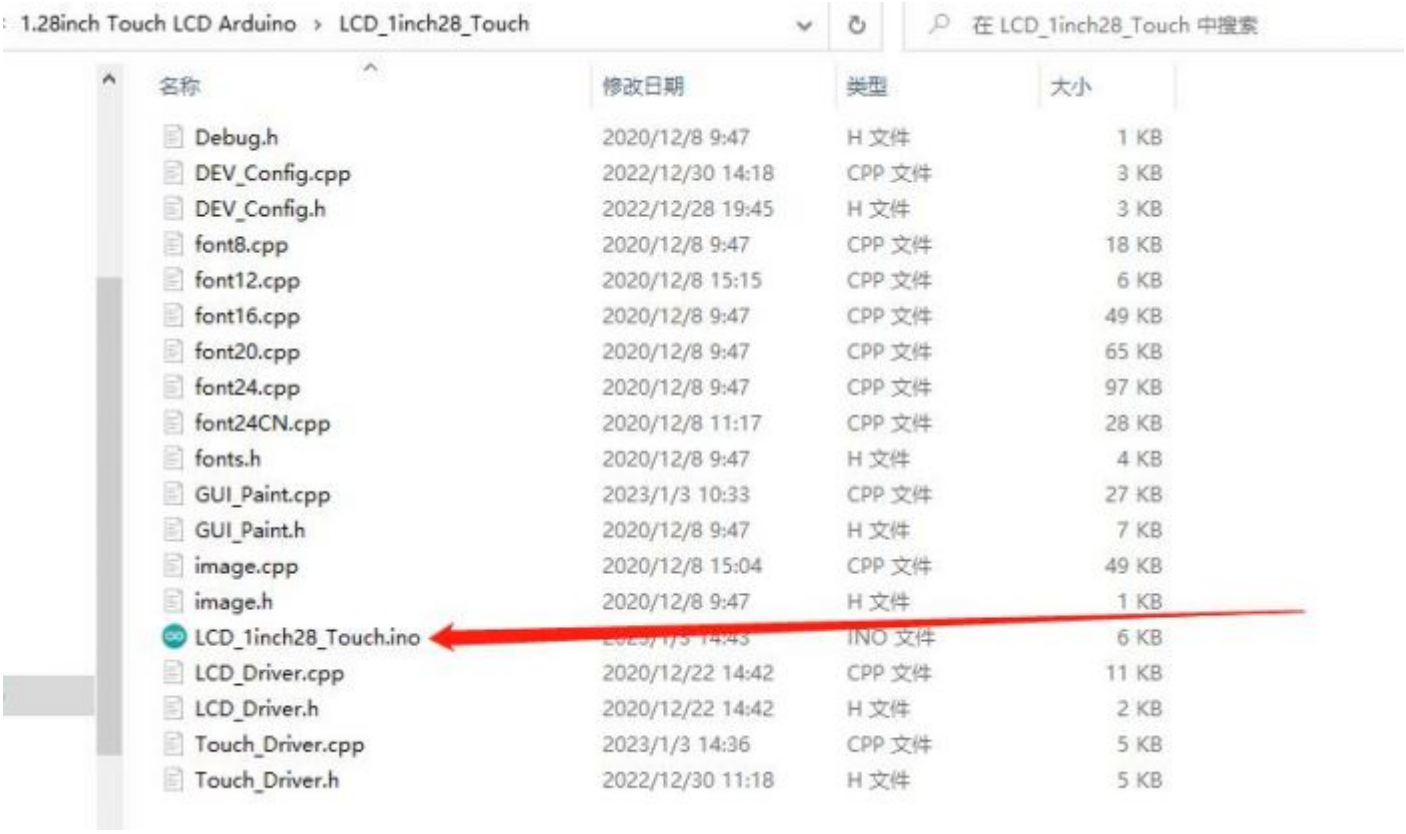
And then click compile and download.



Demo Description

File Introduction

Open ..\1.28inch Touch LCD Arduino\LCD_1inch28_Touch directory:



Among which:
LCD_1inch28_Touch.ino: Open it with Arduino IDE.
LCD_Driver.cpp(.h): the LCD driver demo.
Touch_Driver.cpp(.h): the driver demo for the touch screen.
DEV_Config.cpp(.h): the hardware interface definition, package the pin level of reading and writing, SPI data transmission, and the pin initialization.
font8.cpp, font12.cpp, font16.cpp, font20.cpp, font24.cpp, font24CN.cpp, fonts.h: different font sizes.
image.cpp(.h): image data, with Img2Lcd (download in #Resource), you can convert any BMP picture to 16-bit real RGB picture arrays.
The demo includes the bottom hardware interface, middle layer LCD driver, and the upper application.

Bottom Hardware Interface

In DEV_Config.cpp(.h), these two files define the hardware interface and package functions such as the reading and writing pin level, delay, SPI transmission, and I2C transmission.

Write Pin Level

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

The first parameter is a pin, the second one is the voltage level.

Read Pin Level

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

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

Delay

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

Millisecond level delay.

SPI Data Output

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

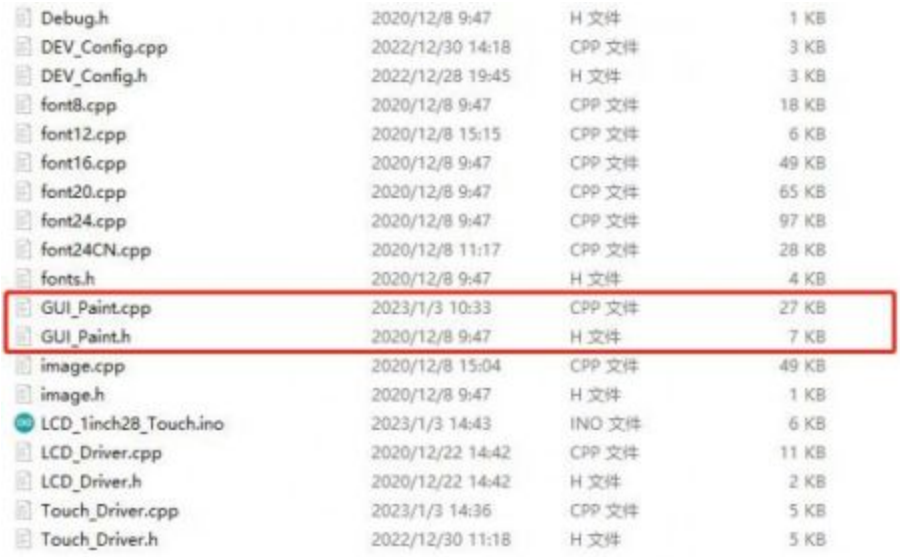
The parameter is char, occupying 8 bits.

I2C Reading and Writing Data

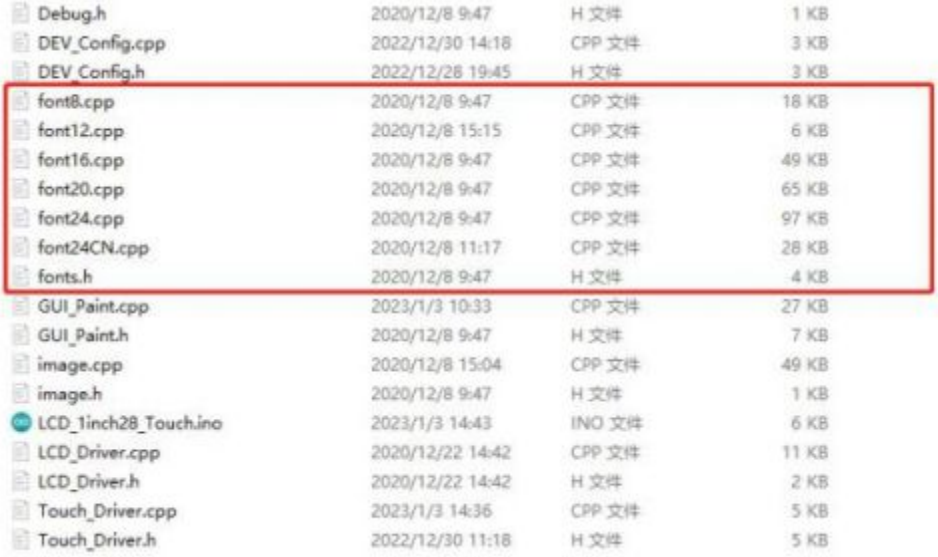
```
void DEV_I2C_Write_Byte(UBYTE DevAddr, UBYTE RegAddr, UBYTE value)
UBYTE DEV_I2C_Read_Byte(UBYTE DevAddr, UBYTE RegAddr)
void DEV_I2C_Read_nByte(UBYTE DevAddr,UBYTE Cmd, UBYTE *data, UBYTE num)
```

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.



The character font on which GUI dependent is font*.cpp (h) file.



- 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 scree
n 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;
```

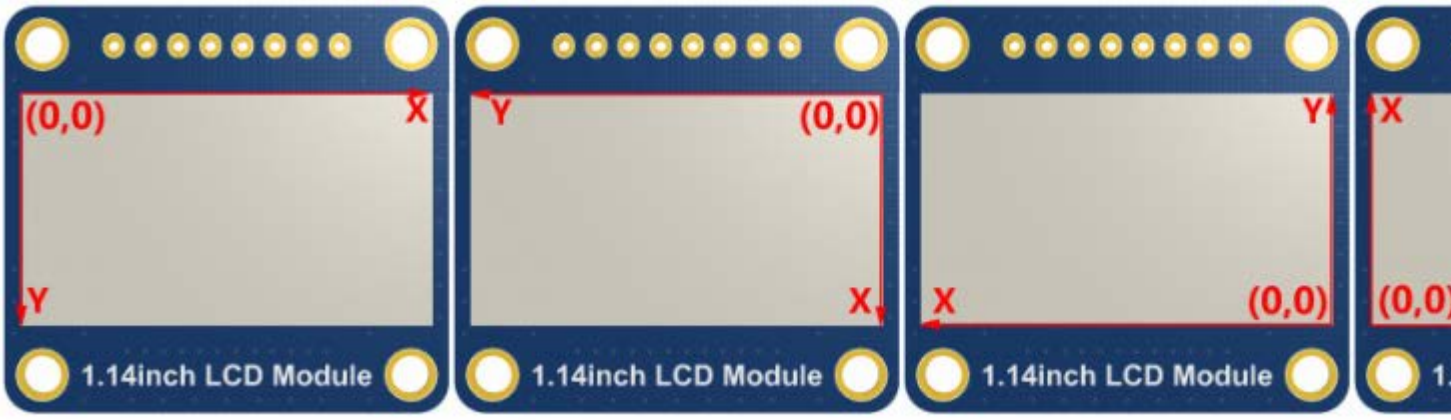
- Select image buffer: the purpose of the selection is that you can create multiple image attributes, an image buffer can exist multiple, and you can select each image you create.

```
void Paint_SelectImage(UBYTE *image)

Parameters:

    Image: the name of the image cache, which is actually a pointer to the first
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, verti
cal 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, UWOR
D 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_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
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, a line from (Xstart, Ystart) to (Xend, Yend), you can choose the color, line width, and line style.

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Co
lor, 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, UWO
RD 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 wi
dths 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, in 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, and font background color.

```
void Paint_DrawChar(UWORD Xstart, UWORD Ystart, const char Ascii_Char, 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

    Ascii_Char: indicates the Ascii character

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

        Font8: 5*8 font

        Font12: 7*12 font

        Font16: 11*16 font

        Font20: 14*20 font

        Font24: 17*24 font

    Color_Foreground: Font color

    Color_Background: indicates the background color
```

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

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

Parameters:

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

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

    PString: string, string is a pointer

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

        Font8: 5*8 font

        Font12: 7*12 font

        Font16: 11*16 font

        Font20: 14*20 font

        Font24: 17*24 font

     Color_Foreground: Font color

     Color_Background: indicates the background color
```

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

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

Parameters:

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

    Ystart: the Y coordinate of the 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, UWOR
D 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 foll
owing Fonts:

        Font8: 5*8 font

        Font12: 7*12 font

        Font16: 11*16 font

        Font20: 14*20 font

        Font24: 17*24 font

    Color_Foreground: Font color

    Color_Background: indicates the background color
```

- Write numbers with decimals: At the left vertex of (Xstart Ystart), write a string of numbers with decimals. You can select Ascii Visual Character library font foreground color font background color.

```
void Paint_DrawFloatNum(UWORD Xpoint, UWORD Ypoint, double Nummber, UBYTE Decima
l_Point, 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

        Nummber: The number displayed here is saved as a double

          Decimal_Point: Display the number after the decimal point

        Font: Ascii visual character font library, the following Fonts are prov
ided 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, 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 as a good time structure, as long as the h
our, minute, and second bits of data to the parameter;

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

        Font8: 5*8 font

        Font12: 7*12 font

        Font16: 11*16 font

        Font20: 14*20 font

        Font24: 17*24 font
```

```
    Color_Foreground: Font color
    Color_Background: indicates the background color
```

- Display image: At (Xstart Ystart) is the left vertex, an Image of W Image width and H Image height is displayed.

```
void Paint_DrawImage(const unsigned char *image, UWORD xStart, UWORD yStart, UWO
RD W_Image, UWORD H_Image)
Parameters:
      image: Image address, which points to the image information you want to di
splay
      Xstart: The left vertex X coordinate of the character
      Ystart: The left vertex Y coordinate of the character
      W_Image: image width
      H_Image: image height
```

Resource

Document

- 1.28inch_Touch_LCD_Schematic

Demo

- 1.28inch_Touch_LCD_Demo

Software

- Zimo221
- Image2Lcd
- Image2Lcd Image Modulo

Dimension drawing

- 1.28inch Touch LCD 3D file
- 1.28inch Touch LCD 2D file

Datasheet

- GC9A01A Datasheet

- CST816S Register Datasheet
- CST816S Datasheet