Waveshare LC76G GNSS Module

Waveshare LC76G Multi-GNSS Module User Manual

Model: LC76G GNSS Module

1. Introduction

The Waveshare LC76G is a compact and high-performance Multi-GNSS (Global Navigation Satellite System) module designed for precise positioning applications. It supports a wide range of satellite systems including GPS, BeiDou (BDS), GLONASS, Galileo, and QZSS, ensuring robust and accurate location data. This module integrates a built-in Low-Noise Amplifier (LNA) for enhanced reception sensitivity and supports both UART and I2C communication interfaces, making it versatile for various embedded projects. It also features EPO (Extended Predictive Orbit) and EASY (Embedded Assist System) functions for faster time-to-first-fix (TTFF) and improved performance.
This manual provides essential information for setting up, operating, and maintaining your LC76G Multi-GNSS Module.

2. Package Contents

Verify that all items listed below are present in your package:

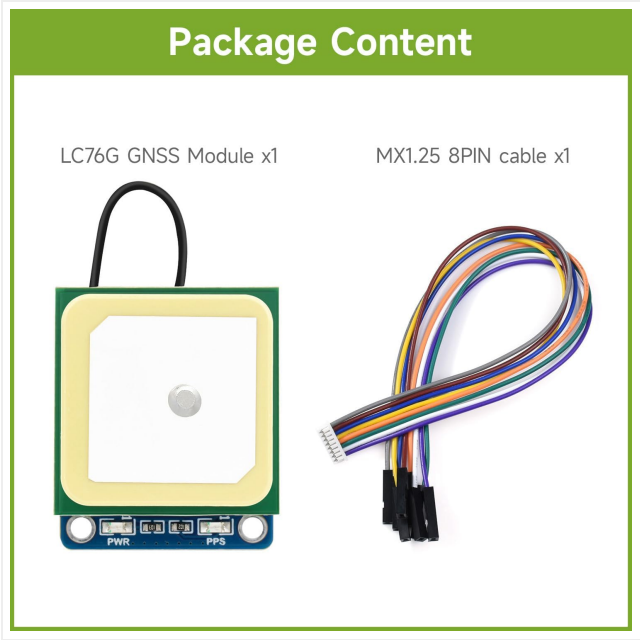


Image: Package contents showing the LC76G GNSS Module and an MX1.25 8PIN cable.

- 1x LC76G GNSS Module
- 1x MX1.25 8PIN cable

3. Key Features

The LC76G GNSS Module offers the following key features:

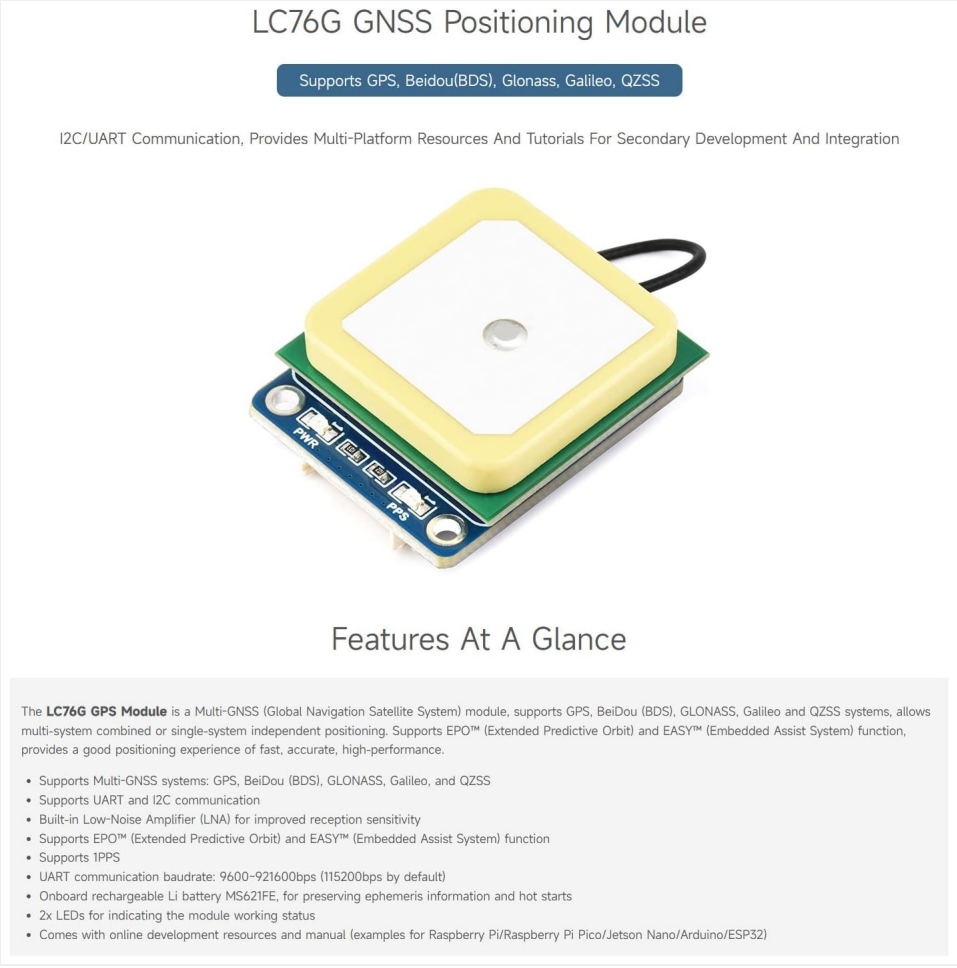
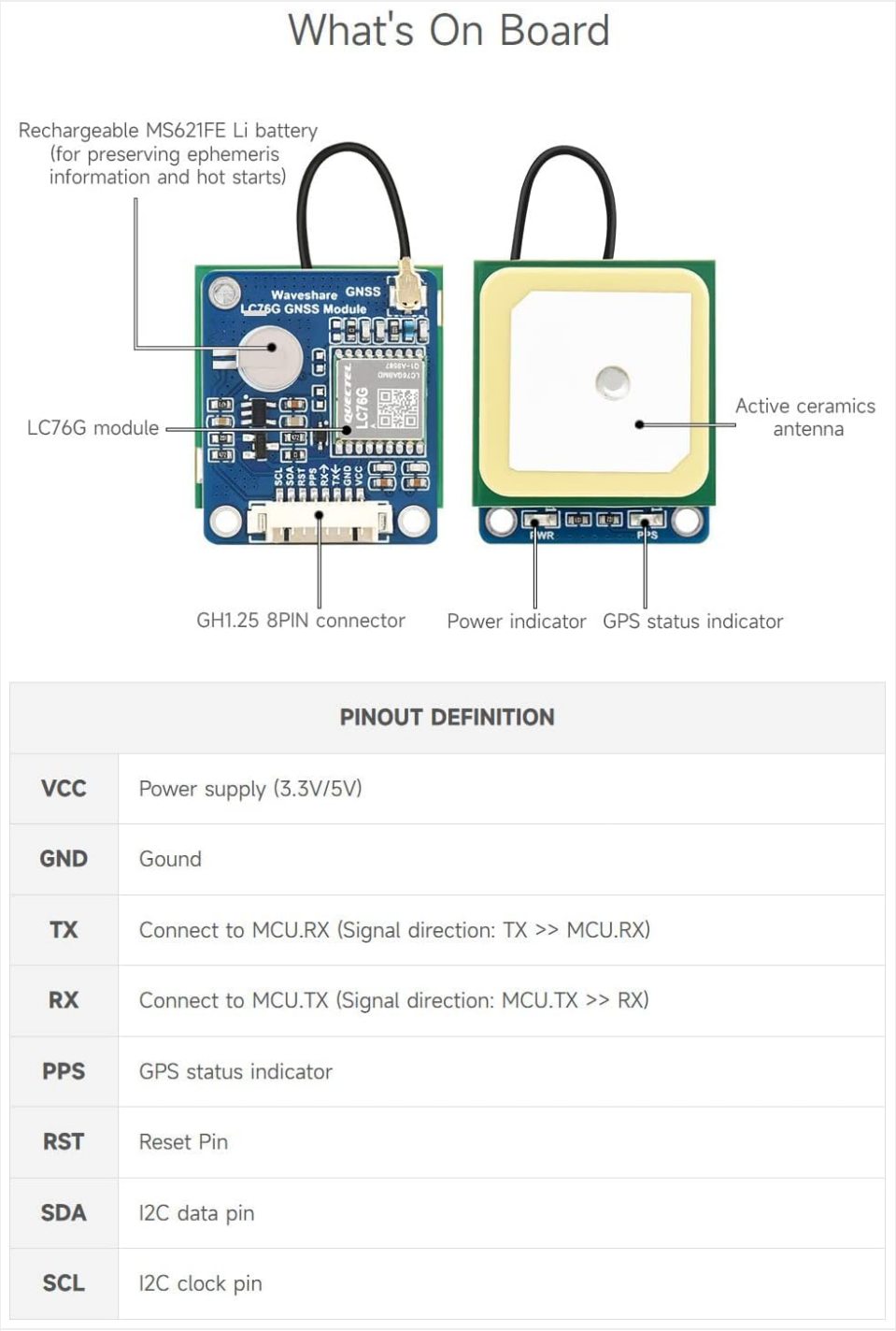


Image: Visual representation of the LC76G GNSS module with its main features listed.

- Supports Multi-GNSS systems: GPS, BeiDou (BDS), GLONASS, Galileo, and QZSS.
- Supports UART and I2C communication interfaces.
- Built-in Low-Noise Amplifier (LNA) for improved reception sensitivity.
- Supports EPO (Extended Predictive Orbit) and EASY (Embedded Assist System) functions for faster acquisition.
- Supports 1PPS (One Pulse Per Second) output for precise timing.
- UART communication baud rate: Configurable from 9600 to 921600bps (default 115200bps).
- Onboard rechargeable Li battery (MS621FE) for preserving ephemeris information and enabling hot starts.
- Two LEDs for indicating module working status (Power and PPS).
- Comes with online development resources and manual examples for various platforms (Raspberry Pi, Raspberry Pi Pico, Jetson Nano, Arduino, ESP32).

4. Pinout Definition

Understanding the pinout is crucial for proper integration of the LC76G module with your host device.



PINOUT DEFINITION	
VCC	Power supply (3.3V/5V)
GND	Gound
TX	Connect to MCU.RX (Signal direction: TX >> MCU.RX)
RX	Connect to MCU.TX (Signal direction: MCU.TX >> RX)
PPS	GPS status indicator
RST	Reset Pin
SDA	I2C data pin
SCL	I2C clock pin

Image: Detailed diagram of the LC76G module's components and pinout, including the GH1.25 8PIN connector.

Pin	Description
VCC	Power supply (3.3V/5V)
GND	Ground
TX	Connect to MCU.RX (Signal direction: TX >> MCU.RX)
RX	Connect to MCU.TX (Signal direction: MCU.TX >> RX)
PPS	GPS status indicator / 1PPS output
RST	Reset Pin
SDA	I2C data pin

Pin	Description
SCL	I2C clock pin

5. Specifications and Parameters

The following table outlines the key technical specifications and performance parameters for the LC76G GNSS Module.

Parameters Comparison

PRODUCT		LC76G(AB)	L76K
GNSS		GPS/GLONASS/Galileo/BDS/QZSS	GPS/GLONASS/BDS/QZSS
CHIP PLATFORM		AG3352	AT6558R
L1 BAND RECEIVER	CHANNELS	47 Tracking ch	32 tracking ch/ 72 acquisition ch
	SBAS	Support	N/A
A-GNSS		Support	Support
SENSITIVITY	ACQUISITION	-147dBm	-148dBm
	RE-ACQUISITION	-159dBm	-160dBm
	TRACKING	-166dBm	-162dBm
TTFF (TIME TO FIRST FIX)	COLD STARTS	26s, Autonomous 12s, With EASY™ 5s, With EPO™	30s, Autonomous 5.5s, With AGNSS
	WARM STARTS	20s, Autonomous 2s, with EASY™	-
	HOT STARTS	1s	2s
POSITION ACCURACY		1.5m CEP	2.0m CEP
SPEED ACCURACY		0.1m/s	0.1m/s
ACCELERATION ACCURACY(MAX)		0.1m/s^2	0.1m/s^2
TIMING ACCURACY		30ns	30ns
UPDATE RATE (MAX)		10Hz	5Hz
I2C RATE		400Kbps (Max)	-
UART BAUDRATE		Range: 9600~921600bps Default: 115200bps	Range: 4800~115200bps Default: 9600bps
GNSS DATA UPDATE RATE		Max: 10Hz Default: 1Hz	Max: 5Hz Default: 1Hz

Image: Detailed comparison of technical parameters for LC76G and L76K GNSS modules.

Parameter	LC76G(AB)
GNSS	GPS/GLONASS/Galileo/BDS/QZSS
Chip Platform	AG3352
L1 Band Receiver	47 Tracking channels
SBAS	Support

Parameter	LC76G(AB)
A-GNSS	Support
Acquisition Sensitivity	-147dBm
Re-acquisition Sensitivity	-159dBm
Tracking Sensitivity	-166dBm
Cold Starts	26s, Autonomous; 12s, With EASY™; 5s, With EPO™
Warm Starts	20s, Autonomous; 2s, With EASY™
Hot Starts	1s
Position Accuracy	1.5m CEP
Speed Accuracy	0.1m/s
Acceleration Accuracy (MAX)	0.1m/s²
Timing Accuracy	30ns
Update Rate (MAX)	10Hz
I2C Rate	400Kbps (Max)
UART Baudrate	Range: 9600-921600bps; Default: 115200bps
GNSS Data Update Rate	Max: 10Hz; Default: 1Hz

6. Compatible Platforms

The LC76G module is designed for broad compatibility, supporting integration with various popular development platforms.

Image: Visual representation of the LC76G module's compatibility with multiple development boards.

The module is compatible with platforms such as:

- Raspberry Pi
- Raspberry Pi Pico
- Jetson Nano
- ESP32
- Arduino

7. Product Overview and Dimensions

A visual overview of the LC76G module and its physical dimensions.

Image: Top view of the LC76G GNSS module, showing the ceramic antenna.

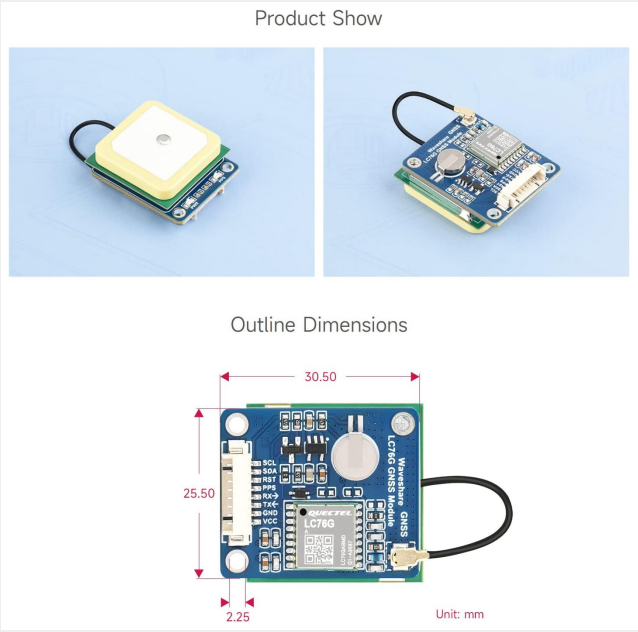


Image: Two angled views of the LC76G GNSS module, showcasing its compact design.

Outline Dimensions

Image: Technical drawing illustrating the physical dimensions of the LC76G module in millimeters.

Dimensions: Approximately 30.50mm x 25.50mm x 2.25mm (excluding antenna cable and connector height).

8. Setup Instructions

Follow these steps to set up your LC76G Multi-GNSS Module:

1. **Power Supply:** Connect the VCC pin to a 3.3V or 5V power source and the GND pin to ground. Ensure the power supply is stable and within the specified voltage range.
2. **Communication Interface:**
    - **UART:** Connect the module's TX pin to your host MCU's RX pin, and the module's RX pin to your host MCU's TX pin. Configure your host MCU's UART to the desired baud rate (default 115200bps).
    - **I2C:** Connect the module's SDA pin to your host MCU's SDA pin, and the module's SCL pin to your host MCU's SCL pin. Ensure appropriate pull-up resistors are used if not already present on your host board.
3. **Antenna:** The module includes an integrated ceramic antenna. Ensure the module has a clear view of the sky for optimal satellite reception. Avoid placing it near metal objects or other sources of interference.
4. **Optional Connections:**
    - **PPS:** If precise timing is required, connect the PPS pin to an input on your host MCU.
    - **RST:** The RST pin can be connected to a GPIO on your host MCU for software-controlled resets.
5. **Initial Power-Up:** Once connections are secure, apply power. The PWR LED should illuminate. The PPS LED will blink once a valid GPS fix is acquired.

9. Operating Instructions

The LC76G module outputs NMEA 0183 sentences, which are standard for GNSS receivers. These sentences contain information such as latitude, longitude, altitude, speed, and satellite status.

1. **Data Reception:**
    - **UART:** Read data from the module's TX pin using your host MCU's UART receiver. The data will be a stream of NMEA sentences.
    - **I2C:** Access the module's data registers via the I2C interface. Refer to the detailed communication protocol in the online documentation for specific register addresses and data formats.
2. **NMEA Sentence Parsing:** Implement NMEA parser code on your host MCU to extract meaningful data from the received sentences. Common sentences include GPGGA (Global Positioning System Fix Data), GPRMC (Recommended Minimum Specific GNSS Data), and GPVTG (Track Made Good and Ground Speed).
3. **Configuration:** The module can be configured using specific NMEA commands sent via the UART or I2C interface. This allows adjustment of parameters such as update rate, enabled GNSS constellations, and output sentence types. Refer to the Waveshare online resources for detailed command sets.
4. **1PPS Output:** The 1PPS signal provides a highly accurate pulse once per second, synchronized with UTC time. This is useful for applications requiring precise time synchronization.
5. **Hot Start** / **Warm Start:** The onboard rechargeable battery helps retain ephemeris data, allowing for faster position fixes (warm or hot starts) after a brief power loss or restart.

For detailed programming examples and libraries for specific platforms (Raspberry Pi, Arduino, etc.), please consult the official Waveshare online development resources.

10. Maintenance

The LC76G Multi-GNSS Module is designed for reliable operation with minimal maintenance. Observe the following guidelines:

- **Environmental Conditions:** Operate the module within its specified temperature and humidity ranges. Avoid exposure to extreme temperatures, direct sunlight, moisture, or corrosive environments.
- **Physical Handling:** Handle the module with care to prevent physical damage. Avoid bending or stressing the PCB or connectors.
- **Cleaning:** If necessary, gently clean the module with a soft, dry cloth. Do not use liquid cleaners or solvents.
- **Antenna Placement:** Ensure the integrated antenna has an unobstructed view of the sky for optimal performance. Avoid covering the antenna with metallic objects or placing it near sources of electromagnetic interference.
- **Battery Life:** The onboard MS621FE battery is rechargeable and designed for long life. If the module consistently experiences cold starts after short power cycles, the battery may need to be charged by keeping the module powered for an extended period.

11. Troubleshooting

If you encounter issues with your LC76G module, consider the following troubleshooting steps:

- **No Power LED:**
    - Check power connections (VCC and GND) and ensure the power supply is providing the correct voltage (3.3V or 5V).

- Verify the power supply is capable of providing sufficient current.

- **No PPS LED Blinking / No Fix:**
  - Ensure the module has a clear, unobstructed view of the sky. Indoor environments or areas with tall buildings can block satellite signals.
  - Allow sufficient time for a cold start (up to 26 seconds or more in challenging conditions).
  - Check for strong electromagnetic interference nearby.
  - Verify the antenna is not covered or damaged.
- **No Data Output (UART/I2C):**
  - **UART:** Double-check TX/RX connections (cross-over). Ensure the baud rate on your host MCU matches the module's baud rate (default 115200bps). Verify your UART receiver code is correctly implemented.
  - **I2C:** Confirm I2C address, SDA/SCL connections, and pull-up resistors. Verify your I2C communication code.
  - Ensure the module is powered and the PWR LED is on.
- **Inaccurate Position Data:**
  - Ensure the module has a good satellite fix (PPS LED blinking steadily).
  - Check for multipath interference (signals reflecting off buildings).
  - Verify the module is configured to use all available GNSS constellations for best accuracy.
- **Module Not Responding:**
  - Try resetting the module by momentarily pulling the RST pin low, or by power cycling it.

12. Warranty and Support

For warranty information, technical support, and access to the latest documentation, drivers, and example code, please visit the official Waveshare website or their dedicated product page for the LC76G Multi-GNSS Module. Waveshare provides comprehensive online resources to assist with development and integration. These resources often include:

- Detailed datasheets and user manuals.
- Example code for various microcontrollers and single-board computers (e.g., Raspberry Pi, Arduino, ESP32).
- Forum or community support.

Contact information for technical support can typically be found on the Waveshare website.

Related Documents - LC76G GNSS Module

	Waveshare Industrial 8-Channel Relay Module for Raspberry Pi Pico User Manual User manual for the Waveshare Industrial 8-Channel Relay Module for Raspberry Pi Pico (Pico-Relay-B). Details features, compatibility, enclosure, and pinout for industrial control applications.

2.4inch LCD Module

Overview

Introduction

As a 2.4inch TFT display module with a resolution of 240 * 320, it uses the SPI interface for communication. LCD has an internal controller with basic functions, which can be used to draw points, lines, circles, and rectangles, and can display English, Chinese as well as pictures.
We provide complete supporting Raspberry Pi demos (BCM2835 library, WiringPi library, and python demos), STM32 demos, and Arduino demos.

Specification

- Operating voltage: 3.3V/5V **(When using 5V power supply, the logic voltage is 5V; When using 3.3V power supply, the logic voltage is 3.3V)**
- Interface: SPI
- LCD Type: TFT
- Controller: IL9341
- Resolution: 240(V) x 320(H)RGB
- Display Size: 36.72（H）x 48.96（V）mm
- Pixel Size: 0.153（H）x 0.153（V）mm
- Dimension: 70.5 x 43.3(mm)

Interface Description

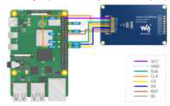
Raspberry Pi hardware connection

Please connect the LCD to your Raspberry Pi by the 8PIN cable according to the table below
If you use the pin header or PH2.0 8PIN interface, you need to connect according to the following table

Connect to Raspberry Pi

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

The 2.4inch LCD uses the PH2.0 8PIN interface, which can be connected to the Raspberry Pi according to the above table: (Please connect according to the pin definition table. The color of the wiring in the picture is for reference only, and the actual color shall prevail.)



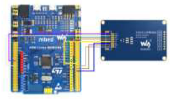
STM32 hardware connection

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

STM32F103ZET connection pin correspondence

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

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

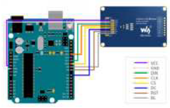


Arduino hardware connection

Arduino UNO Connection pin correspondence

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

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

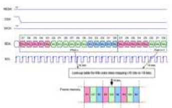


Hardware Description

LCD and the controller

The LCD supports 12-bit, 16-bit, and 18-bit input color formats per pixel, namely RGB444, RGB565, and RGB666 three color formats, this demo uses RGB565 color format, which is also a commonly used RGB format.
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 the GPIO port, and the communication speed will be faster.

Communication Protocol



Note: Different from the traditional SPI protocol, the data line from the slave to the master is hidden since the device only has a display requirement.
RESX: the reset pin, it should be low when powering the module and be higher at other times;
CSX: slave chip select, when CS is low, the chip is enabled.
D/CX: data/command control pin, when DC = 0, write command, when DC = 1, write data
SDA: the data pin for transmitting RGB data, it works as the MOSI pin of SPI interface;
SCL works as the SCLK pins of SPI interface.
SPI communication has data transfer timing, which is combined by CPHA and CPOL.
CPOL determines the level of the serial synchronous clock at idle state. When CPOL = 0, the level is Low. However, CPOL has little effect to the transmission.
CPHA determines whether data is collected at the first clock edge or at the second clock edge of serial synchronous clock; when CPHL = 0, data is collected at the first clock edge.
There are 4 SPI communication modes. SPI0 is commonly used, in which CPHL = 0, CPOL = 0.

Working with Raspberry Pi

Enable SPI interface

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

- Open terminal, use command to enter the configuration page

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

Reboot Raspberry Pi：

```
sudo reboot
```

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

Install Libraries

- **Install BCM2835 libraries**

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

- **Install wiringPI libraries**

```
#Open the Raspberry Pi terminal and run the following command
sudo apt-get install wiringpi
#For Raspberry Pi systems after May 2019 (earlier than before, you may not
need to execute), you may need to upgrade:
wget https://project-downloads.drogon.net/wiringpi-latest.deb
sudo dpkg -i wiringpi-latest.deb
gpio -v
# Run gpio -v and version 2.52 will appear. If it does not appear, the ins
tallation is wrong

#Bullseye branch system use 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 does not appear, it mean
s that there is an installation error
```

- **Install Python libraries**

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

Open Raspberry Pi terminal and run the following command

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

Run the demo codes

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

C codes

- Re-compile the demo codes

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

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

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

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

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

python

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

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

Test programs for all screens can be viewed, sorted by size:
0inch96_LCD_test.py: 0.96inch LCD test program
1inch14_LCD_test.py: 1.14inch LCD test program
1inch28_LCD_test.py: 1.28inch LCD test program
1inch3_LCD_test.py: 1.3inch LCD test program
1inch47_LCD_test.py: 1.47inch LCD test program
1inch54_LCD_test.py: 1.54inchLCD test program
1inch8_LCD_test.py: 1.8inch LCD test program
2inch_LCD_test.py: 2inch LCD test program
2inch4_LCD_test.py: 2inch4 LCD test program

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

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

FBCP Porting

PS: FBCP is currently not compatible with 64-bit Raspberry Pi system, it is recommended to use 32-bit system.

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://www.waveshare.com/w/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
#1.44inch LCD Module
sudo ./shell/waveshare-1inch44
#1.54inch LCD Module
sudo ./shell/waveshare-1inch54
#1.8inch LCD Module
sudo ./shell/waveshare-1inch8
#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.

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 -DBACKLIG
HT_CONTROL=ON -DSTATISTICS=0 ..
#1.14inch LCD Module
sudo cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_1INCH14_LCD=ON -DBACKLIG
HT_CONTROL=ON -DSTATISTICS=0 ..
#1.3inch LCD Module
sudo cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_1INCH3_LCD=ON -DBACKLIGH
T_CONTROL=ON -DSTATISTICS=0 ..
#1.54inch LCD Module
sudo cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_1INCH54_LCD=ON -DBACKLIG
HT_CONTROL=ON -DSTATISTICS=0 ..
#1.8inch LCD Module
sudo cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_1INCH8_LCD=ON -DBACKLIGH
T_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 -DBACKLIGH
T_CONTROL=ON -DSTATISTICS=0 ..
```

Set up to start automatically

```
sudo cp ~/Waveshare_fbcp/buil
d/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.

Set the Display Resolution

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

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

Then add the following lines at the end of the config.txt.

```
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 you are using.

```
#2.4inchinch LCD Module & 2inchinch LCD Module
hdmi_cvt=640 480 60 1 0 0 0

#1.8inch LCD Module
hdmi_cvt=400 300 60 1 0 0 0

#1.3inch LCD Module & 1.54inch LCD Module
hdmi_cvt=300 300 60 1 0 0 0

#1.14inch LCD Module
hdmi_cvt=300 170 60 1 0 0 0

#0.96inch LCD Module
hdmi_cvt=300 150 60 1 0 0 0
```

And then reboot the system

```
sudo reboot
```

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

API Description

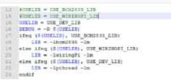
The RaspberryPi series can share a set of programs, because they are all embedded systems, and the compatibility is relatively strong.
The program is divided into bottom-layer hardware interface, middle-layer LCD screen driver, and upper-layer application;

C

Hardware Interface

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

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



- Data type:

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

- Module initialization and exit processing.

```
void DEV_Module_Init(void);
void DEV_Module_Exit(void);
Note:
  Here is some GPIO processing before and after using the LCD screen.
```

- GPIO read and write:

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

- SPI write data:

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

Upper application

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

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

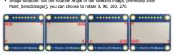
- New Image Properties: Create a new image buffer, this property includes the image buffer name, width, height, flip Angle, and color.

```
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;
```

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

```
void Paint_SelectImage(UBYTE *image)
Parameters:
       Image: the name of the image buffer, which is actually a pointer to the first address of the image buffer;
```

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



```
void Paint_SetRotate(UWORD Rotate)
Parameters:
        Rotate: ROTATE_0, ROTATE_90, ROTATE_180, and ROTATE_270 correspond to 0, 90, 180, and 270 degrees.
```

```
be 0, 90, 180, and 270 degrees.
```

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

```
void Paint_SetMirroring(UBYTE mirror)
Parameters:
        Mirror: indicates the image mirroring mode. MIRROR_NONE, MIRROR_HO
RIZONTAL, MIRROR_VERTICAL, MIRROR_ORIGIN correspond to no mirror, horizont
al mirror, vertical mirror, and image center mirror respectively.
```

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

```
void Paint_SetPixel(UWORD Xpoint, UWORD Ypoint, UWORD Color)
Parameters:
        Xpoint: the X position of a point in the image buffer
        Ypoint: Y position of a point in the image buffer
        Color: indicates the Color of the dot
```

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

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

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

```
void Paint_ClearWindows(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yen
d, UWORD Color)
Parameters:
        Xstart: the x-starting coordinate of the window
        Ystart: the y-starting coordinate of the window
        Xend: the x-end coordinate of the window
        Yend: the y-end coordinate of the window
        Color: fill Color
```

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

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

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

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UW
ORD Color, LINE_STYLE Line_Style ,  LINE_STYLE Line_Style)
Parameters:
        Xstart: the x-starting coordinate of a line
        Ystart: the y-starting coordinate of the a line
        Xend: the x-end coordinate of a line
        Yend: the y-end coordinate of a line
        Color: fill Color
        Line_width: The width of the line, the demo provides 8 sizes of wi
dth 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 s
traight or dashed way.
                typedef enum {
                   LINE_STYLE_SOLID = 0,
                   LINE_STYLE_DOTTED,
                } LINE_STYLE;
```

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

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

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

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

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

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

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

```
void Paint_DrawString_EN(UWORD Xstart, UWORD Ystart, const char * pString,
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
provides the following Fonts:
                Font8: 5*8 font
                Font12: 7*12 font
                Font16: 11*16 font
                Font20: 14*20 font
                Font24: 17*24 font
        Color_Foreground: Font color
        Color_Background: indicates the background color
```

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

```
void Paint_DrawString_CN(UWORD Xstart, 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, 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* Fon
t, 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 the demo
provides the following Fonts:
                Font8: 5*8 font
                Font12: 7*12 font
                Font16: 11*16 font
                Font20: 14*20 font
                Font24: 17*24 font
        Color_Foreground: Font color
        Color_Background: indicates the background color
```

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

```
void Paint_DrawTime(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 hours, minutes, and seconds are passed to the parameters;
        Font: Ascii visual character library, in the Fonts folder the demo
provides the following Fonts:
                Font8: 5*8 font
                Font12: 7*12 font
                Font16: 11*16 font
                Font20: 14*20 font
                Font24: 17*24 font
        Color_Foreground: Font color
        Color_Background: indicates the background color
```

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

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

```
UBYTE GUI_ReadBmp(const char *path, 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 passed by default
       Ystart: The Y coordinate of the left vertex of the picture, general
ly 0 by default
```

Testing Code for Users

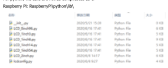
For Raspberry Pi, in the directory: RaspberryPi\c\examples, for all the test code;

If you need to run the 0.96-inch LCD test program, you need to add 0.96 as a parameter when running the main demo.
Re-execute in Linux command mode as follows:

```
make clean
make
sudo ./main 0.96
```

Python (for Raspberry Pi)

Works with python and python3.
For python, his calls are not as complicated as C.
Raspberry Pi: RaspberryPi\python\lib\



lcdconfig.py

- Module initialization and exit processing.

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

- GPIO read and write:

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

- SPI write data.

```
def spi_writebyte(data)
```

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

python is in the following directory:
Raspberry Pi: RaspberryPi\python\examples\

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

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

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

[Waveshare 2.4-inch LCD Module User Manual]

A comprehensive guide to the Waveshare 2.4-inch LCD TFT display module, detailing its features, specifications, and usage with Raspberry Pi, STM32, and Arduino. Learn about SPI interface, IL9341 controller, hardware connections, and software examples for integrating this 240x320 resolution display into your projects.

- 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, UW
ORD 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 w
idths
        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 strai
ght or dashed way
        typedef enum {
            LINE_STYLE_SOLID = 0,
            LINE_STYLE_DOTTED,
        } LINE_STYLE;
```

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

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

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

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

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

```
void Paint_DrawString_CN(UWORD Xstart, UWORD Ystart, const char * pString,
cFONT* font, UWORD Color_Foreground,  UWORD Color_Background)
Parameters:
    Xstart: the x-coordinate of the left vertex of a character
    Ystart: the Y coordinate of the font's left vertex
    PString: string, string is a pointer
    Font: GB2312 encoding character Font library, in the Fonts folder prov
ides the following Fonts:
        Font12CN: ASCII font 11*21, Chinese font 16*21
        Font24CN: ASCII font24 *41, Chinese font 32*41
    Color_Foreground: Font color
    Color_Background: indicates the background color
```

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

```
void Paint_DrawNum(UWORD Xpoint, UWORD Ypoint, double Nummber, sFONT* Fon
t, UWORD Digit, UWORD Color_Foreground,   UWORD Color_Background)
Parameters:
    Xpoint: the x-coordinate of the left vertex of a character
    Ypoint: the Y coordinate of the left vertex of the font
    Nummber: indicates the number displayed, which can be a decimal
    Digit: It's a decimal number
    Font: Ascii visual character library, in the Fonts folder provides the
following Fonts:
        Font8: 5*8 font
        Font12: 7*12 font
        Font16: 11*16 font
        Font20: 14*20 font
        Font24: 17*24 font
    Color_Foreground: Font color
    Color_Background: indicates the background color
```

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

```
void Paint_DrawTime(UWORD Xstart, UWORD Ystart, PAINT_TIME *pTime, sFONT*
Font, UWORD Color_Background,  UWORD Color_Foreground)
Parameters:
    Xstart: the x-coordinate of the left vertex of a character
    Ystart: the Y coordinate of the font's left vertex
    PTime: display time, here defined a good time structure, as long as th
e hour, minute and second bits of data to the parameter;
    Font: Ascii visual character library, in the Fonts folder provides the
following Fonts:
        Font8: 5*8 font
        Font12: 7*12 font
        Font16: 11*16 font
        Font20: 14*20 font
        Font24: 17*24 font
    Color_Foreground: Font color
    Color_Background: indicates the background color
```

Arduino software description

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

Run program

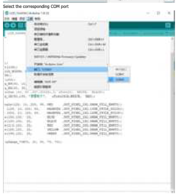
In the product encyclopedia interface download the program, and then unzip it. The Arduino program is located at ~/Arduino/...

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

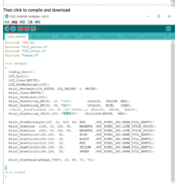
You can view test programs for all screen sizes, sorted by size:

For example, 1.54inch LCD Module. Open the LCD_1inch54 folder and run the LCD_1inch54.ino file.
Open the program, select the development board model Arduino UNO.

Select the corresponding COM port



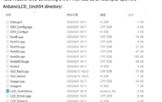
Then click to compile and download



Program Description

Document Introduction

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



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 writes pin levels, SPI transmission data, and pin initialization;
font8.cpp, font12.cpp, font16.cpp, font20.cpp, font24.cpp, font24CN.cpp, fonts.h: fonts for characters of different sizes;
image.cpp(.h): is the image data, which can convert any BMP image into a 16-bit true color image array through Img2Lcd (downloadable in the development data).
The program is divided into bottom-layer hardware interface, middle-layer LCD screen driver, and upper-layer application;

Underlying hardware interface

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

- write pin level

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

The first parameter is the pin, and the second is the high and low 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 xms)
```

millisecond level delay.

- SPI output data

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

The parameter is char type, occupying 8 bits

The upper application

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

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

The fonts used by the GUI all depend on the font*.cpp(.h) Files under the same 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 *Image, UWORD Width, UWORD Height, UWORD Rotate, UWORD Color)
Parameters:
    Image: image buffer Width
    Width: Set the width of the image buffer
    Rotate: Indicates the rotation Angle of an image
    Color: the initial Color of the image
```

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

```
void Paint_SetClearFuntion(void (*Clear)(UWORD));
parameter:
    Clear: Pointer to the clear screen function, used to quickly clear the screen to a certain color
```

- Set the drawing pixel function.

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

- Select image buffer: the purpose of the selection is that you can create multiple image attributes, image buffer can exist multiple, 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 corresponds to no mirror, horizontal mirror, vertical mirror, and about image center mirror respectively.
```

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

```
void Paint_SetPixel(UWORD Xpoint, UWORD Ypoint, UWORD Color)
Parameters:
    Xpoint: the X position of a point in the image buffer
    Ypoint: Y position of a point in the image buffer
    Color: indicates the Color of the dot
```

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

```
void Paint_ClearWindows(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, UWORD Color)
Parameters:
    Xstart: the x-starting coordinate of the window
    Ystart: indicates the Y starting point of the window
    Xend: the x-end coordinate of the window
    Yend: indicates the y-end coordinate of the window
    Color: fill Color
```

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

```
void Paint_DrawPoint(UWORD Xpoint, UWORD Ypoint, UWORD Color, DOT_PIXEL Dot_Pixel, DOT_STYLE Dot_Style )
Parameters:
    Xpoint: indicates the X coordinate of a point
    Ypoint: indicates the Y coordinate of a point
    Color: fill Color
    Dot_Pixel: The size of the dot, providing a default of eight size points
        typedef enum {
            DOT_PIXEL_1X1 = 1,          // 1 x 1
            DOT_PIXEL_2X2 ,             // 2 X 2
            DOT_PIXEL_3X3 ,             // 3 X 3
            DOT_PIXEL_4X4 ,             // 4 X 4
            DOT_PIXEL_5X5 ,             // 5 X 5
            DOT_PIXEL_6X6 ,             // 6 X 6
            DOT_PIXEL_7X7 ,             // 7 X 7
            DOT_PIXEL_8X8 ,             // 8 X 8
        } DOT_PIXEL;
    Dot_Style: the size of a point that expands from the center of the point as to draw 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 so visible
            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 = 1,
                    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: the x-end coordinate 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. Defaults eight visible 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 = 1,
                DRAW_FILL_FULL,
            } DRAW_FILL
```

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

```
void Paint_DrawCircle(UWORD X_Center, UWORD Y_Center, UWORD Radius, UWORD 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 visible
        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 = 1,
                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: 8*8 Font
                Font12: 7*12 Font
                Font16: 11*16 Font
                Font20: 14*20 Font
                Font24: 17*24 Font
        Color_Foreground: Font color
        Color_Background: indicates the background color
```

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

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

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

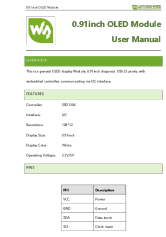
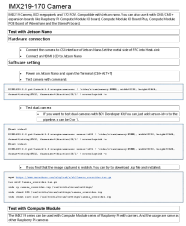
```
void Paint_DrawString_CN(UWORD Xstart, UWORD Ystart, const char * pString, cFONT* Font, UWORD Color_Foreground,  UWORD Color_Background)
Parameters:
        Xstart: the x-coordinate of the left vertex of a character
        Ystart: the Y coordinate of the font's left vertex
        pString: string, string is a pointer
        Font: GB2312 encoding character Font Library, in the Fonts folder provides the following fonts:
                Font12CN: ASCII font 11*21, Chinese font 16*21
                Font24CN: ASCII font 24, 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 Number, 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
        Number: indicates the number displayed, which can be a decimal number
        Digit: Displays the number of digits after the decimal point
        Font: Ascii visual character Library, in the Fonts folder provides the following fonts:
                Font8: 8*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 Number, UBYTE Decimal_Point, 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
        Number: the displayed number. Which is saved in double type data
        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: 8*8 font
                Font12: 7*12 font
                Font16: 11*16 font
                Font20: 14*20 font
                Font24: 17*24 font
```

	[0.91inch OLED Module User Manual - Waveshare] User manual for the Waveshare 0.91inch OLED Module (128x32 pixels) with SSD1306 controller. Covers overview, features, pinout, I2C communication, and demo code for STM32, Raspberry Pi (BCM2835, WiringPi, Python), and Arduino.
	[IMX219-170 Camera User Guide for Jetson Nano and Compute Module] A guide to using the IMX219-170 camera with Jetson Nano and Raspberry Pi Compute Modules, including hardware connection, software setup, and troubleshooting.

	[WAVESHARE UART Fingerprint Sensor (F) Command Manual: Protocol and Command Reference] This manual details the serial communication protocol, command list, and data packet formats for the WAVESHARE UART Fingerprint Sensor (F) module. It serves as a comprehensive guide for developers integrating fingerprint recognition capabilities into their projects.
	[Waveshare RG500U-CN & RM500U-CN Linux Driver User Guide] A comprehensive guide for integrating Waveshare's RG500U-CN and RM500U-CN 5G modules with Linux systems, covering USB-to-serial and network card driver setup, AT command testing, and troubleshooting.