Waveshare Core357608000

Waveshare Luckfox Core3576 Module User Manual

Model: Core357608000

1. INTRODUCTION

The Waveshare Luckfox Core3576 Module is a high-performance, low-power AIoT processor core module based on Rockchip's RK3576. It features a sophisticated big.LITTLE architecture, integrating quad-core Arm Cortex-A72 at 2.2GHz and quad-core Arm Cortex-A53 at 2.0GHz. This module is equipped with an ARM Mali G52 MC3 GPU for graphics and an NPU delivering 6 TOPS (Tera Operations Per Second) of computing power, making it suitable for a wide range of embedded applications requiring advanced AI capabilities and multimedia processing.

The Core3576 supports 4K encoding and decoding, various display outputs including HDMI, MIPI DSI, and eDP, and a rich set of peripheral interfaces such as PCIe, CSI, USB, I2C, SPI, and GPIO, providing extensive connectivity and expansion options for diverse project requirements.

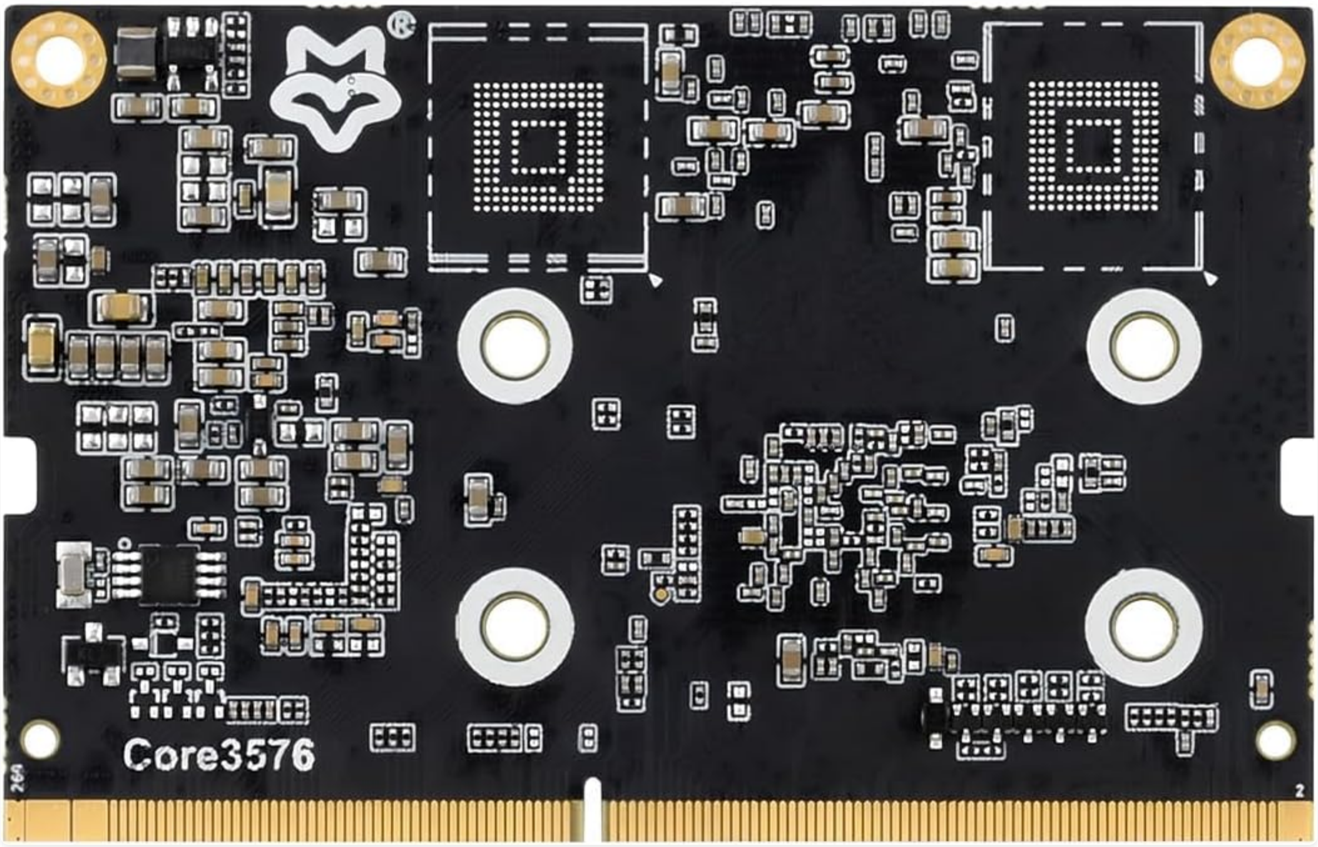


Figure 1.1: Top view of the Waveshare Luckfox Core3576 Module.

2. Package Contents

Verify that all items are present and in good condition upon opening the package. If any items are missing or damaged, please contact customer support.

- Core357608000 Module x 1

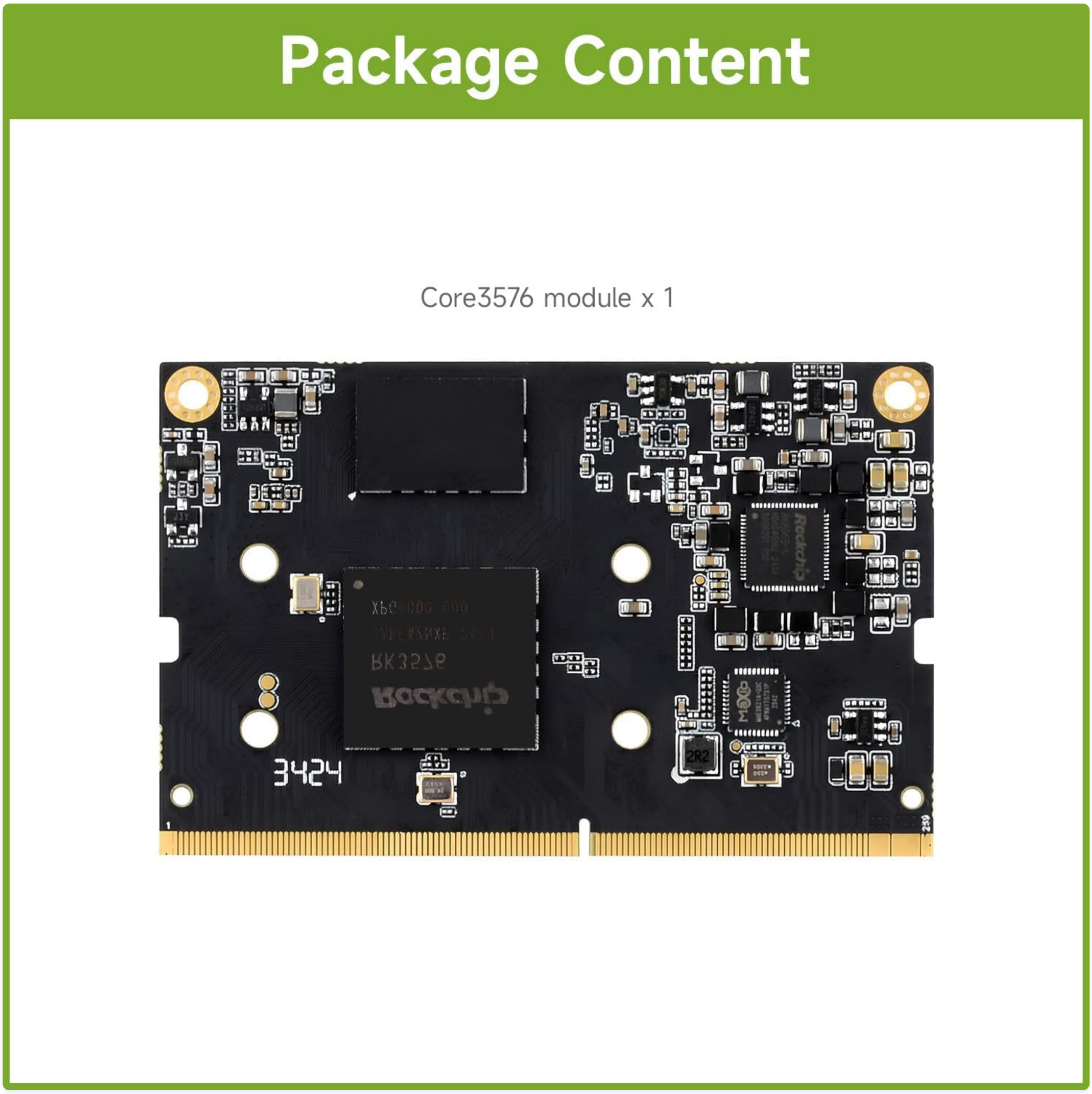
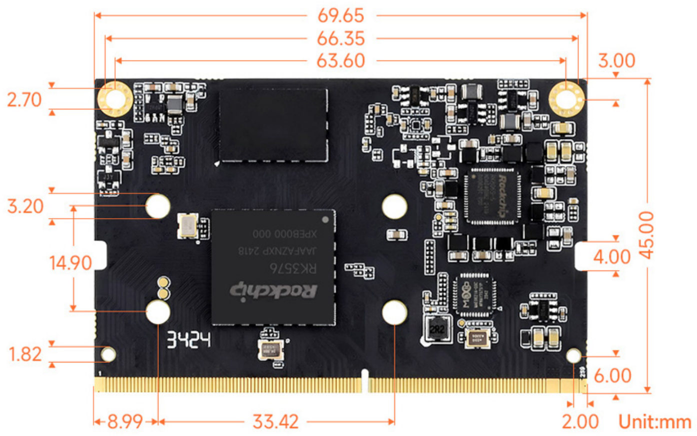

Package Content
Core3576 module x 1
RK3576
Rockchip
3424
2R2

Figure 2.1: Illustration of the package content, showing the Core3576 module.

3. SPECIFICATIONS

The following table details the technical specifications of the Core3576 Module:

Outline dimensions



Core3576 Module Technical Specifications

Feature	Description
Processor	Quad-core Arm Cortex-A72@2.2GHz + Quad-core Arm Cortex-A53@2.0GHz
RAM	8GB LPDDR4X RAM
Storage	0GB eMMC Flash (Options for 0 / 32 / 64GB eMMC Flash)
NPU	6 TOPS@INT8, supports INT4, INT16, FP16, BF16, FP32 mixed operation
GPU	ARM Mali G52 MC3@0.9GHz, supports OpenGL ES 1.1/2.0/3.2, OpenCL 2.0 and Vulkan 1.1, and built-in high-performance 2D/3D acceleration hardware
ISP	Built-in 16MP ISP, supports multiple algorithm accelerators such as HDR, 3A, CAC, 3DNR, 2DNR, Sharpening, Dehaze, Enhance, Debayer, and small-angle lens distortion correction, etc.
Video	Decoding: 4K@120fps (H.265/HEVC, VP9, AVS2, AV1), 4K@60fps (H.264/AVC) Encoding: 4K@60fps (H.265/HEVC, H.264/AVC)
Connectivity	Supports 1000M Ethernet × 2 Expandable for 2.4GHz/5GHz dual-band WiFi 6, Bluetooth 5.2
Video Output	HDMI2.1 (4K@120fps) × 1 / eDP1.3 (4K@60fps) × 1 4-lane MIPI DSI (2K@60fps) × 1
Video Input	MIPI CSI DPHY × 2 MIPI CSI D/CPHY
PCIe/SATA	PCIe 2.1/SATA3.1 × 2
USB	USB3.2 Gen1 OTG (5Gbps), supports DP protocol USB2.0 interface
SDIO	SDIO3.0 × 2
CAN	CAN FD × 2
ADC	SARADC × 8

Feature	Description
Expansion Interfaces	I2C × 10; I3C × 2; UART × 12; SPI × 5; CAN × 2; Multiple GPIOs
Connector	260 Pin SO-DIMM Connector
Power Input	5V DC
Outline Dimensions	69.6mm × 45mm
Others	Supports 38KHz infrared input Low power RTC chip with backup battery input

Note: The actual number of available interfaces may be reduced due to pin multiplexing.

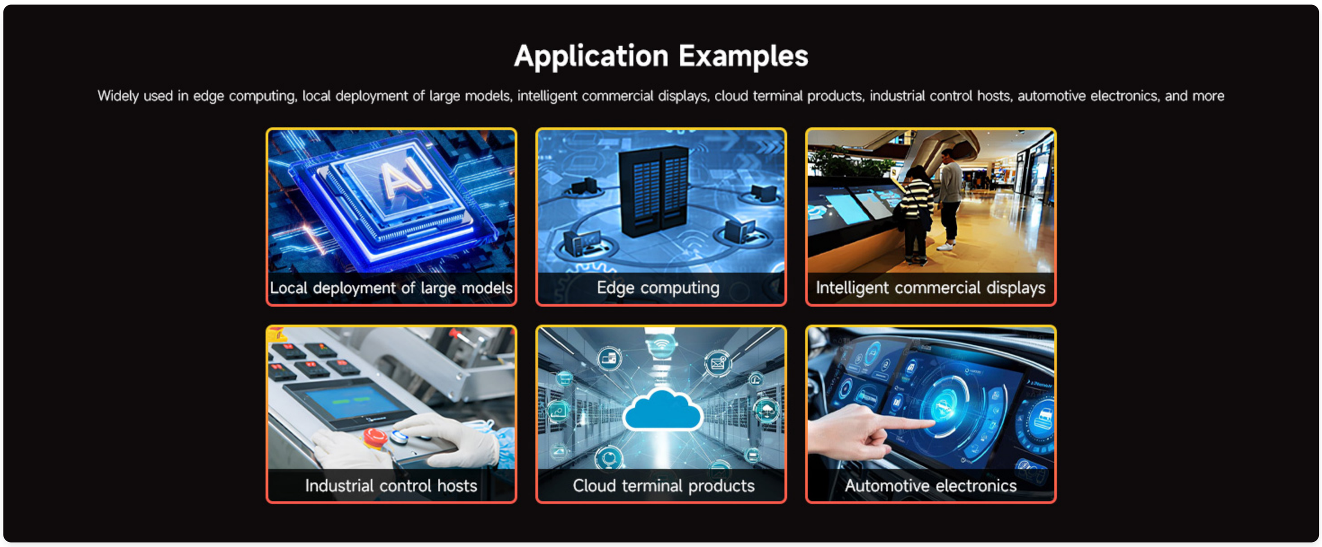
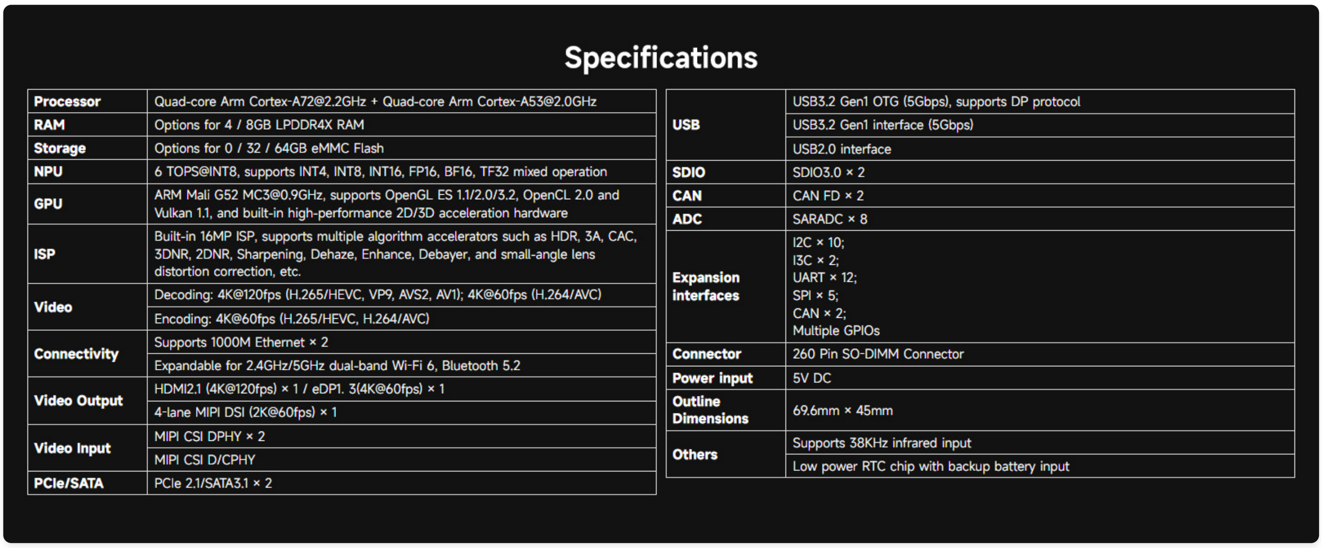

Application Examples
Widely used in edge computing, local deployment of large models, intelligent commercial displays, cloud terminal products, industrial control hosts, automotive electronics, and more
AI
Local deployment of large models
Edge computing
Intelligent commercial displays
Industrial control hosts
Cloud terminal products
Automotive electronics

Figure 3.1: Outline dimensions of the Core3576 Module.

4. Key Features

4.1 Low Power Octa-Core AIoT Processor

The Core3576 is equipped with the high-performance Rockchip RK3576 processor, integrating quad-core Cortex-A72 and quad-core Cortex-A53. This configuration provides robust performance while maintaining high energy efficiency, making it ideal for AIoT applications.



Specifications

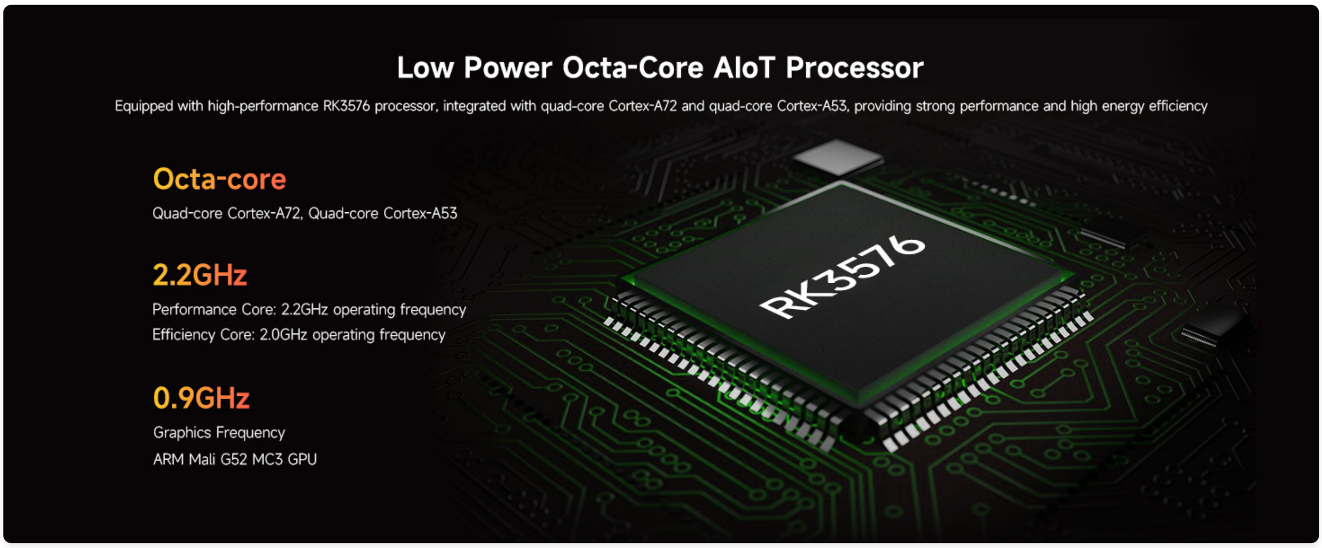
Processor	Quad-core Arm Cortex-A72@2.2GHz + Quad-core Arm Cortex-A53@2.0GHz
RAM	Options for 4 / 8GB LPDDR4X RAM
Storage	Options for 0 / 32 / 64GB eMMC Flash
NPU	6 TOPS@INT8, supports INT4, INT8, INT16, FP16, BF16, TF32 mixed operation
GPU	ARM Mali G52 MC3@0.9GHz, supports OpenGL ES 1.1/2.0/3.2, OpenCL 2.0 and Vulkan 1.1, and built-in high-performance 2D/3D acceleration hardware
ISP	Built-in 16MP ISP, supports multiple algorithm accelerators such as HDR, 3A, CAC, 3DNR, 2DNR, Sharpening, Dehaze, Enhance, Debayer, and small-angle lens distortion correction, etc.
Video	Decoding: 4K@120fps (H.265/HEVC, VP9, AVS2, AV1); 4K@60fps (H.264/AVC)
	Encoding: 4K@60fps (H.265/HEVC, H.264/AVC)
Connectivity	Supports 1000M Ethernet × 2
	Expandable for 2.4GHz/5GHz dual-band Wi-Fi 6, Bluetooth 5.2
Video Output	HDMI2.1 (4K@120fps) × 1 / eDP1. 3(4K@60fps) × 1
	4-lane MIPI DSI (2K@60fps) × 1
Video Input	MIPI CSI DPHY × 2
	MIPI CSI D/CPHY
PCIe/SATA	PCIe 2.1/SATA3.1 × 2

USB	USB3.2 Gen1 OTG (5Gbps), supports DP protocol
	USB3.2 Gen1 interface (5Gbps)
	USB2.0 interface
SDIO	SDIO3.0 × 2
CAN	CAN FD × 2
ADC	SARADC × 8
Expansion interfaces	I2C × 10; I3C × 2; UART × 12; SPI × 5; CAN × 2; Multiple GPIOs
Connector	260 Pin SO-DIMM Connector
Power input	5V DC
Outline Dimensions	69.6mm × 45mm
Others	Supports 38KHz infrared input
	Low power RTC chip with backup battery input

Figure 4.1: Overview of the RK3576 processor architecture.

4.2 6 TOPS Computing Power NPU

The integrated Neural Processing Unit (NPU) offers 6 TOPS of computing power, facilitating the efficient conversion and execution of various neural network models based on popular frameworks such as TensorFlow, MXNet, PyTorch, and Caffe. This enables advanced AI applications like object recognition, face recognition, vehicle recognition, pedestrian detection, text recognition, and semantic segmentation.



Low Power Octa-Core AIoT Processor

Equipped with high-performance RK3576 processor, integrated with quad-core Cortex-A72 and quad-core Cortex-A53, providing strong performance and high energy efficiency

Octa-core

Quad-core Cortex-A72, Quad-core Cortex-A53

2.2GHz

Performance Core: 2.2GHz operating frequency

Efficiency Core: 2.0GHz operating frequency

0.9GHz

Graphics Frequency

ARM Mali G52 MC3 GPU

Figure 4.2: AI applications enabled by the 6 TOPS NPU.

4.3 4K Encoding and Decoding Support

The module supports high-definition video tasks with 4K@120fps decoding (H.265/HEVC, VP9, AVS2, AV1) and 4K@60fps encoding (H.265/HEVC, H.264/AVC). This capability ensures smooth handling of demanding multimedia content.

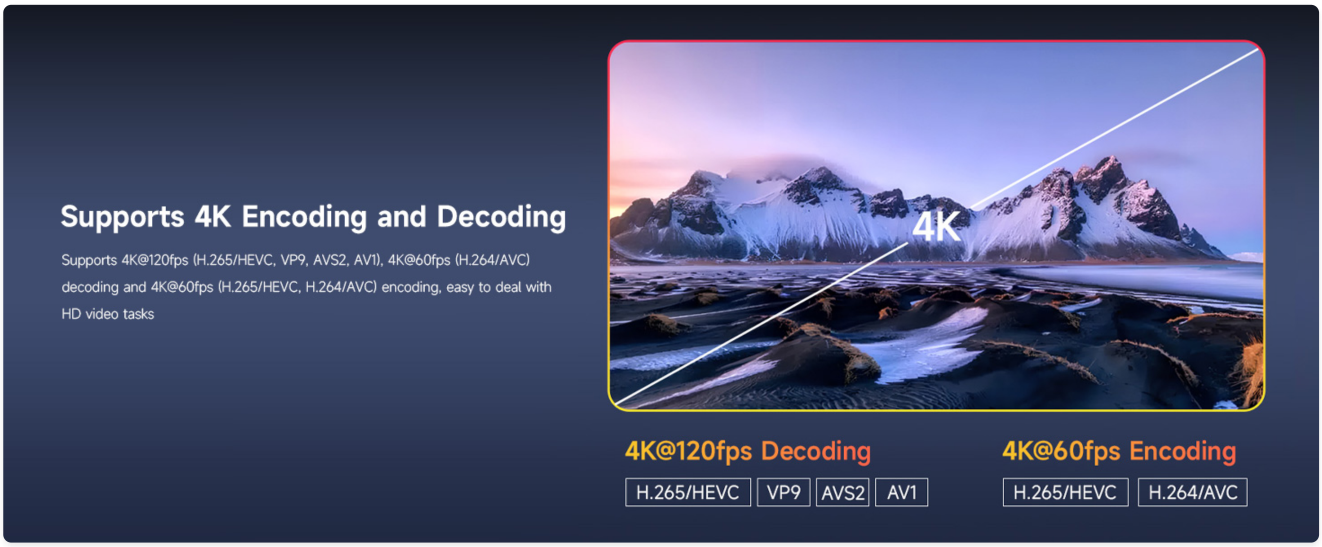

Supports 4K Encoding and Decoding
Supports 4K@120fps (H.265/HEVC, VP9, AVS2, AV1), 4K@60fps (H.264/AVC) decoding and 4K@60fps (H.265/HEVC, H.264/AVC) encoding, easy to deal with HD video tasks
4K
4K@120fps Decoding
H.265/HEVC
VP9
AVS2
AV1
4K@60fps Encoding
H.265/HEVC
H.264/AVC

Figure 4.3: 4K video encoding and decoding capabilities.

4.4 Dual Gigabit Ethernet Interfaces

The Core3576 module features dual Gigabit Ethernet interfaces, allowing for flexible network configurations. Different types of network traffic can be distributed to separate interfaces, enhancing security and management flexibility for various applications, such as one for external internet connection and another for internal LAN.

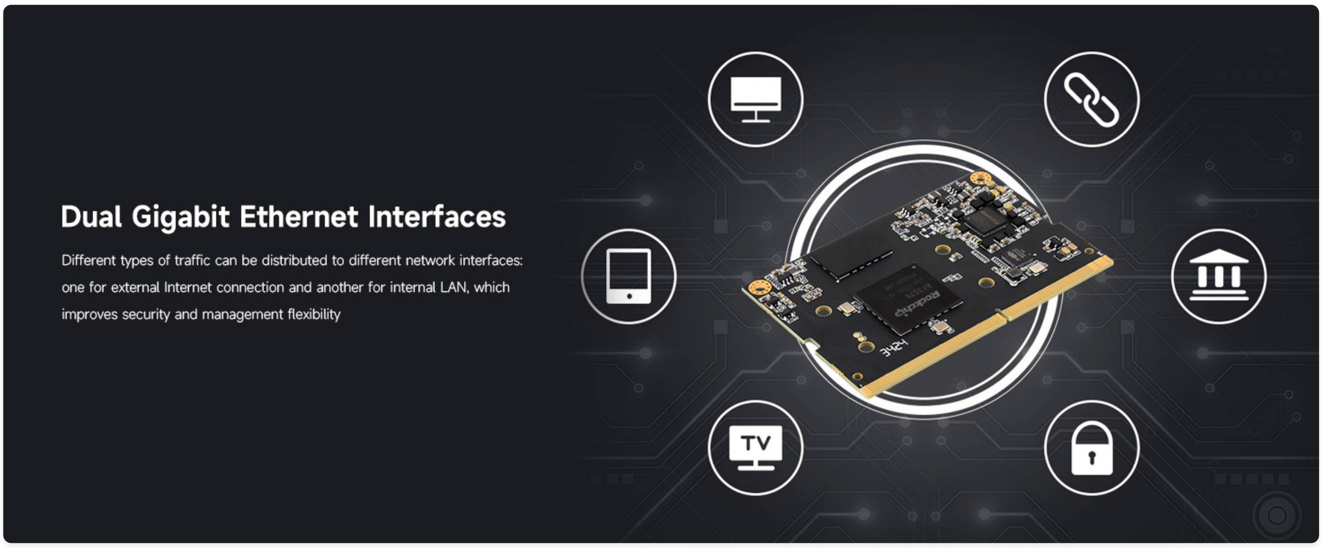
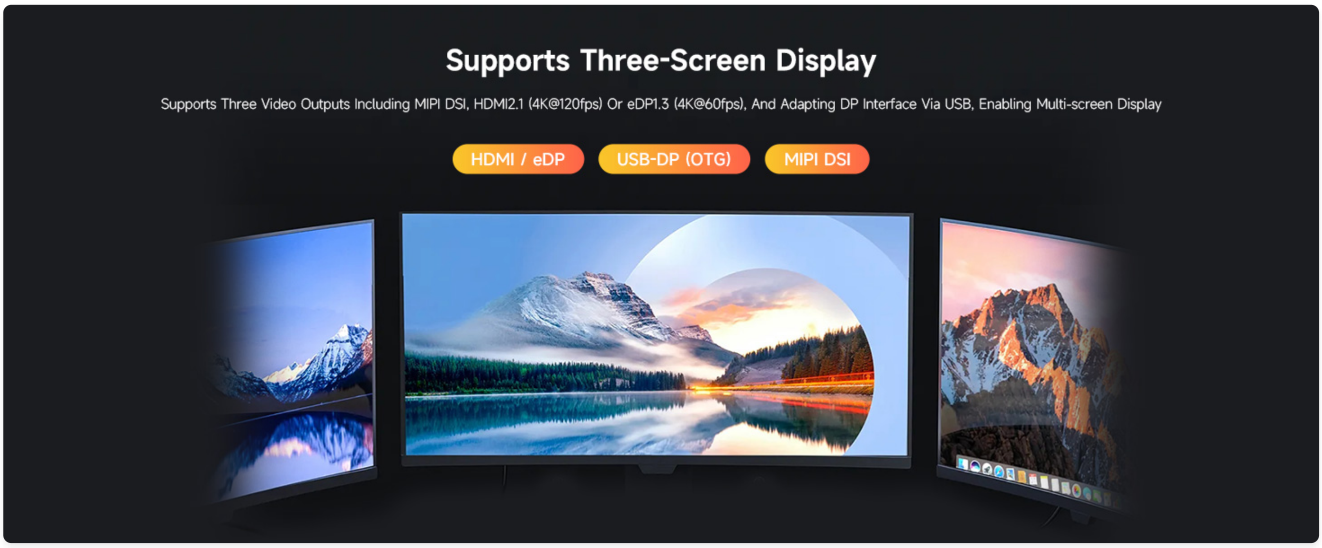

Dual Gigabit Ethernet Interfaces
Different types of traffic can be distributed to different network interfaces: one for external Internet connection and another for internal LAN, which improves security and management flexibility
TV

Figure 4.4: Dual Gigabit Ethernet interfaces for enhanced network flexibility.

4.5 Three-Screen Display Support

The module supports simultaneous output to three screens via MIPI DSI, HDMI2.1 (4K@120fps), or eDP1.3 (4K@60fps). Additionally, it can adapt to DP interfaces via USB, enabling versatile multi-screen display setups for various visual applications.

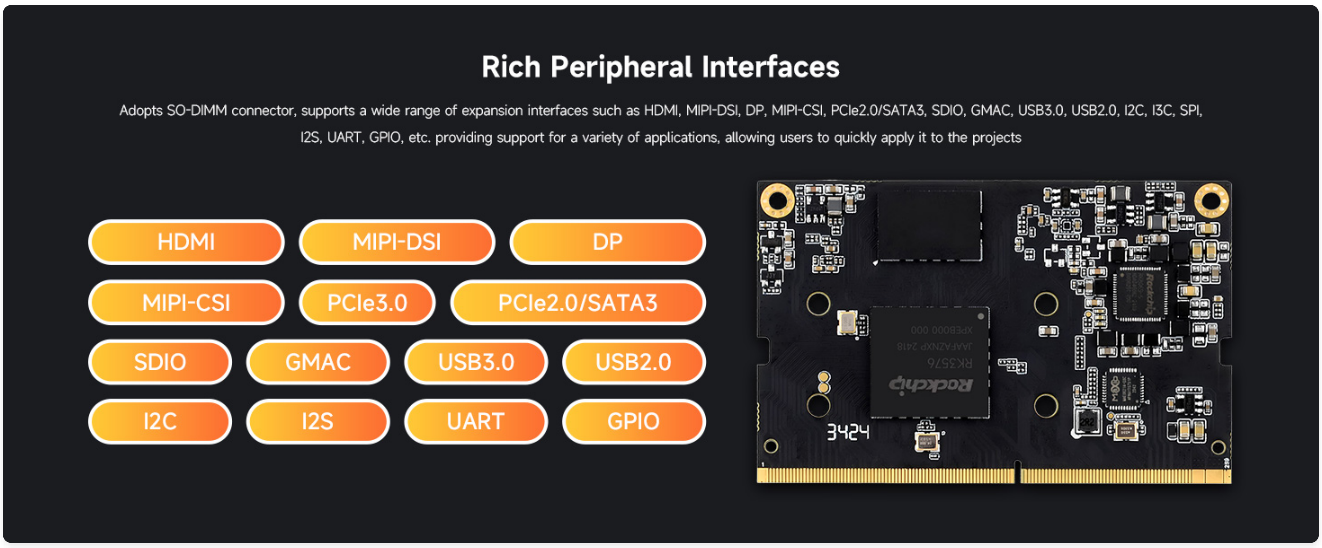

Supports Three-Screen Display
HDMI / eDP
USB-DP (OTG)
MIPI DSI

Supports Three Video Outputs Including MIPI DSI, HDMI2.1 (4K@120fps) Or eDP1.3 (4K@60fps), And Adapting DP Interface Via USB, Enabling Multi-screen Display

Figure 4.5: Multi-screen display capabilities.

4.6 Rich Peripheral Interfaces

Adopting a SO-DIMM connector, the Core3576 module supports a wide array of expansion interfaces. These include HDMI, MIPI-DSI, DP, MIPI-CSI, PCIe2.0/SATA3, SDIO, GMAC, USB3.0, USB2.0, I2C, I3C, SPI, I2S, UART, and GPIO. This extensive range of interfaces provides robust support for diverse applications, allowing users to quickly integrate the module into their projects.


Rich Peripheral Interfaces
HDMI
MIPI-DSI
DP
MIPI-CSI
PCIe3.0
PCIe2.0/SATA3
SDIO
GMAC
USB3.0
USB2.0
I2C
I2S
UART
GPIO
Rockchip
RK3576
3424

Adopts SO-DIMM connector, supports a wide range of expansion interfaces such as HDMI, MIPI-DSI, DP, MIPI-CSI, PCIe2.0/SATA3, SDIO, GMAC, USB3.0, USB2.0, I2C, I3C, SPI, I2S, UART, GPIO, etc. providing support for a variety of applications, allowing users to quickly apply it to the projects

Figure 4.6: Available peripheral interfaces on the Core3576 Module.

5. APPLICATION EXAMPLES

The versatility and powerful capabilities of the Core3576 Module make it suitable for a broad spectrum of applications, including:

- Edge computing
- Local deployment of large models
- Intelligent commercial displays
- Cloud terminal products
- Industrial control hosts

- Automotive electronics

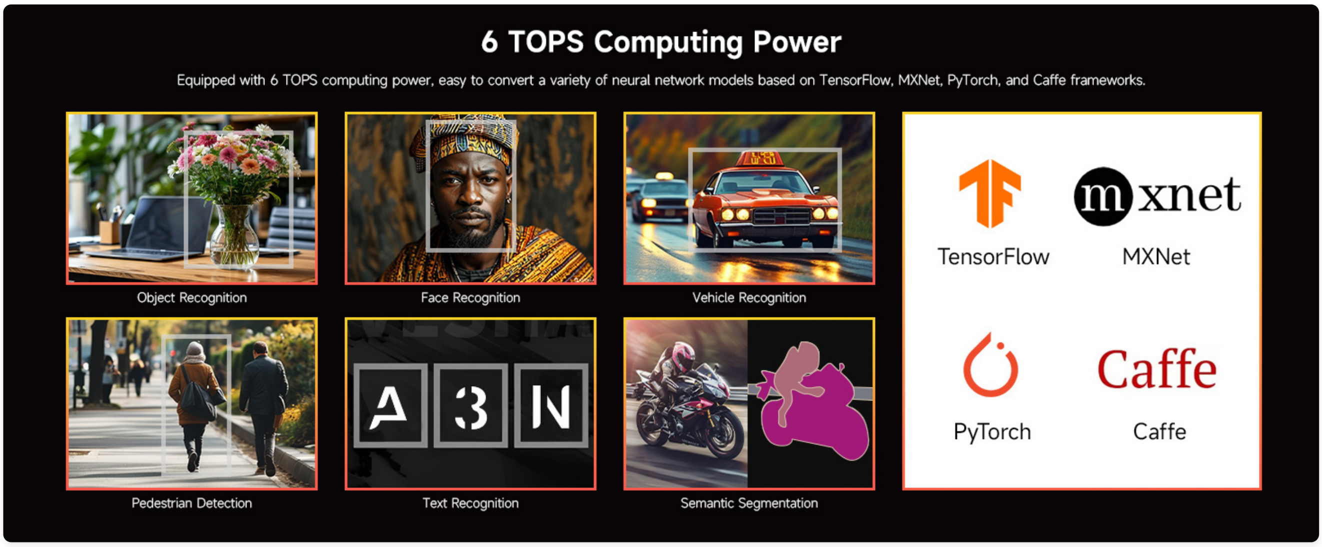

6 TOPS Computing Power
Equipped with 6 TOPS computing power, easy to convert a variety of neural network models based on TensorFlow, MXNet, PyTorch, and Caffe frameworks.
Object Recognition
Face Recognition
Vehicle Recognition
Pedestrian Detection
Text Recognition
Semantic Segmentation
TensorFlow
MXNet
PyTorch
Caffe

Figure 5.1: Diverse application scenarios for the Core3576 Module.

6. SETUP GUIDE

As a core module, the Core3576 requires integration with a carrier board or custom PCB for full functionality. The setup process typically involves:

1. **Hardware Integration:** Carefully insert the Core3576 module into the compatible SO-DIMM slot on your carrier board. Ensure proper alignment and secure seating.
2. **Power Supply:** Connect a stable 5V DC power supply to your carrier board. Refer to your carrier board's documentation for specific power requirements and connection points.
3. **Peripheral Connections:** Connect necessary peripherals such as display units (via HDMI, MIPI DSI, or eDP), USB devices, network cables, and other sensors or actuators to the carrier board's corresponding ports.
4. **Software Preparation:** Prepare the operating system image (e.g., Linux distribution) compatible with the RK3576 processor. This typically involves flashing the image onto an SD card or eMMC storage if available on your carrier board.
5. **Initial Boot:** Power on the system. Monitor the boot process via a serial console or connected display.

For detailed instructions on specific carrier boards or development kits, please refer to the official Waveshare Wiki resources or the documentation provided with your specific hardware setup.

7. OPERATING INSTRUCTIONS

Operating the Core3576 Module primarily involves interacting with the software running on it. Key aspects include:

1. **Operating System Interaction:** Once booted, interact with the installed operating system (e.g., Linux) via command line, graphical user interface (if available), or remote access (SSH).
2. **Software Development:** Utilize the provided SDKs and toolchains to develop applications. Leverage the NPU for AI inference tasks using frameworks like TensorFlow, PyTorch, etc.

3. **Peripheral Control:** Access and control connected peripherals through the operating system's drivers and APIs.
4. **Network Configuration:** Configure the dual Gigabit Ethernet interfaces and any wireless connectivity (if an expansion module is used) for network access and communication.
5. **Multimedia Processing:** Utilize the module's video encoding and decoding capabilities for multimedia applications.

Refer to the software documentation and examples provided by Waveshare for specific programming guides and application development.

8. MAINTENANCE

To ensure the longevity and optimal performance of your Core3576 Module, consider the following maintenance guidelines:

- **Environmental Conditions:** Operate the module within its specified temperature and humidity ranges. Avoid extreme temperatures, direct sunlight, and high humidity.
- **Dust and Debris:** Keep the module and carrier board free from dust and debris. Use compressed air or a soft brush for cleaning.
- **Power Supply:** Use a stable and appropriate power supply (5V DC) to prevent damage from voltage fluctuations.
- **Firmware/Software Updates:** Regularly check for and apply firmware and software updates from Waveshare to benefit from performance improvements, bug fixes, and security enhancements.
- **Physical Handling:** Handle the module by its edges to avoid touching sensitive components. Use anti-static precautions when handling to prevent electrostatic discharge (ESD) damage.

9. TROUBLESHOOTING

This section provides general troubleshooting tips for common issues. For more specific problems, consult the Waveshare Wiki or community forums.

- **Module Not Powering On:**
    - Verify the power supply is connected correctly and providing the required 5V DC.
    - Ensure the module is properly seated in the SO-DIMM slot.
    - Check for any visible damage to the module or carrier board.
- **No Display Output:**
    - Confirm the display cable is securely connected to both the carrier board and the monitor.
    - Ensure the correct display output (HDMI, MIPI DSI, eDP) is selected in the software configuration.
    - Try a different display cable or monitor to rule out external issues.
- **Network Connectivity Issues:**
    - Check Ethernet cable connections.

  - Verify network configurations in the operating system.
  - Ensure network drivers are correctly installed.
- **System Instability/Crashes:**
  - Ensure adequate cooling for the module, especially under heavy load.
  - Verify the power supply is stable and sufficient.
  - Check for corrupted software installations or incompatible drivers.

If issues persist, consider reinstalling the operating system or contacting Waveshare technical support.

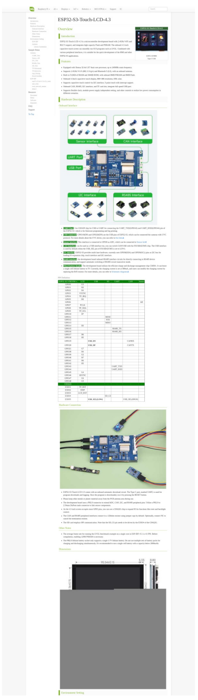
10. Warranty and Support

For information regarding product warranty, technical support, and additional resources, please refer to the official Waveshare website and Wiki. Waveshare provides extensive documentation and community support for their products.

- **Official Waveshare Store:** For product information and purchases, visit the Waveshare Amazon Store.
- **Wiki Resources:** Waveshare provides rich Wiki resources with detailed tutorials, code examples, and technical documentation. Please contact Waveshare for more information on accessing these resources.
- **Technical Support:** For technical inquiries or assistance, refer to the support channels listed on the official Waveshare website.

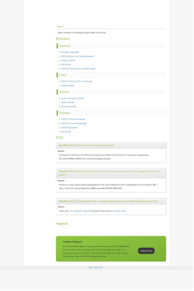
© 2024 Waveshare. All rights reserved.
This manual is subject to change without notice.

Related Documents - Core357608000



[Waveshare ESP32-S3-Touch-LCD-4.3 Development Board: Features & Guide](#)

Explore the Waveshare ESP32-S3-Touch-LCD-4.3, a powerful microcontroller development board featuring a 4.3-inch capacitive touch display, WiFi, BLE 5, and multiple interfaces like CAN, RS485, and I2C. Learn about its hardware, setup, and sample demos for HMI development.



demo1

Resource

Document

Demo

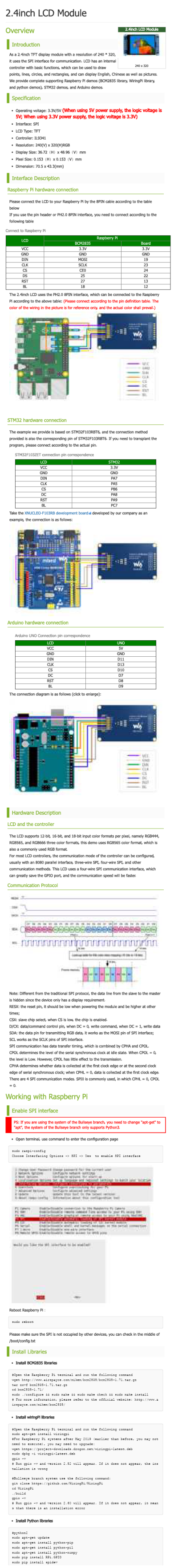
Software

Datasheet

FAQ

Support

Technical Support



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
```

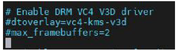
Method 2: Manual Configuration

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.



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

A reboot is then required.

```
sudo reboot
```

Compile and run

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

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

```
#0.96inch LCD Module
sudo cmake -DSPI_BUS_CLOCK_DIVISOR=20 -DWAVESHARE_0INCH96_LCD=ON -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
```

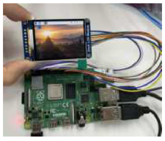
Replace the above hdmi_cvt=[options] according to the LCD Module you are using.

```
#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

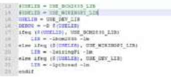
The RaspberryPi series can share a set of programs, because they are all embedded systems, and the compatibility is relatively strong.
The program is divided into bottom-layer hardware interface, middle-layer LCD screen driver, and upper-layer application;

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 Wi
ringPi libraries, please open RaspberryPi\c\Makefile and modify lines 13-1
5 as follows:
```



```
13  #USELIB = USE_BCM2835_LIB
14  #USELIB = USE_WIRINGPI_LIB
15  USELIB = USE_DEV_LIB
16  DEBUG = -D $(USELIB)
17  ifeq ($(USELIB), USE_BCM2835_LIB)
18      LIB = -lbcm2835 -lm
19  else ifeq ($(USELIB), USE_WIRINGPI_LIB)
20      LIB = -lwiringPi -lm
21  else ifeq ($(USELIB), USE_DEV_LIB)
22      LIB = -lpthread -lm
23  endif
```

- 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.

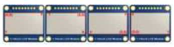
- New Image Properties: Create a new image buffer, this property includes the image buffer name, width, height, flip Angle, and color.

```
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.
```

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

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

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

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

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

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

- The fill color of a certain window in the image buffer: the image buffer part of the window filled with a certain color, usually used to 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, 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 Dot_Pixel, DOT_STYLE Dot_Style)
Parameters:
        Xpoint: indicates the X coordinate of a point.
        Ypoint: indicates the Y coordinate of a point.
        Color: fill Color
        Dot_Pixel: The size of the dot, the demo provides 8 size pointss by default.
              typedef enum {
                DOT_PIXEL_1X1  = 1,    // 1 x 1
                DOT_PIXEL_2X2  ,               // 2 X 2
                DOT_PIXEL_3X3  ,               // 3 X 3
                DOT_PIXEL_4X4  ,               // 4 X 4
                DOT_PIXEL_5X5  ,               // 5 X 5
                DOT_PIXEL_6X6  ,               // 6 X 6
                DOT_PIXEL_7X7  ,               // 7 X 7
                DOT_PIXEL_8X8  ,               // 8 X 8
              } DOT_PIXEL;
        Dot_Style: the size of a point that expands from the center of the point or from the bottom left corner of the point to the right and up.
                typedef enum {
                    DOT_FILL_AROUND  = 1,
                    DOT_FILL_RIGHTUP,
                 } DOT_STYLE;
```

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

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

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

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

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

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

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

```
void Paint_DrawChar(UWORD Xstart, UWORD Ystart, const char Ascii_Char, sFONT* Font, 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* Font, UWORD Digit, UWORD Color_Foreground,   UWORD Color_Background)
Parameters:
        Xpoint: the x-coordinate of the left vertex of a character
        Ypoint: the Y coordinate of the left vertex of the font
        Nummber: indicates the number displayed, which can be a decimal
        Digit: It's a decimal number
        Font: Ascii visual character library, in the Fonts folder 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, generally 0 by default
```

Testing Code for Users

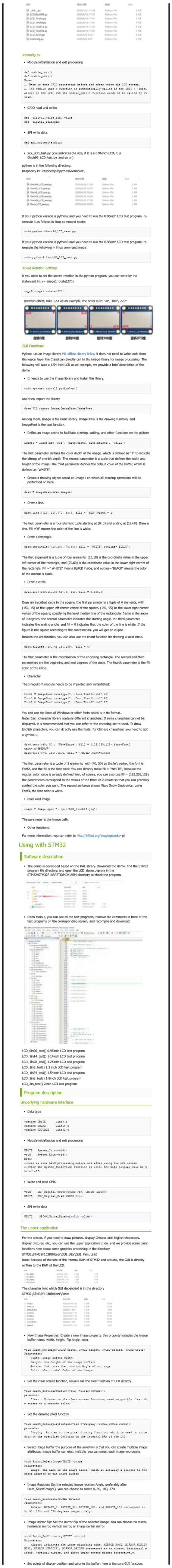
For Raspberry Pi, in the directory: RaspberryPi\c\examples, for all the test code;

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
alizer 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:

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

About Rotation Settings

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

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

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

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 to the image library for image processing. The following will take a 1.54-inch LCD as an example, we provide a brief description of the demo.

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

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

And then import the library

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

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

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

```
image1 = Image.new("RGB", (disp.width, disp.height), "WHITE")
```

The first parameter defines the color depth of the image, which is defined as "1" to indicate the bitmap of one-bit depth. The second parameter is a tuple that defines the width and height of the image. The third parameter defines the default color of the buffer, which is defined as "WHITE".

- Create a drawing object based on Image1 on which all drawing operations will be performed on here.

```
draw = ImageDraw.Draw(image1)
```

- Draw a line.

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

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

- Draw a rectangle.

```
draw.rectangle([(20,10),(70,60)],fill = "WHITE",outline="BLACK")
```

The first argument is a tuple of four elements. (20,10) is the coordinate value in the upper left corner of the rectangle, and (70,60) is the coordinate value in the lower right corner of the rectangle. Fill =" WHITE" means BLACK inside, and outline="BLACK" means the color of the outline is black.

- Draw a circle.

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

Draw an inscribed circle in the square, the first parameter is a tuple of 4 elements, with (150, 15) as the upper left corner vertex of the square, (190, 55) as the lower right corner vertex of the square, specifying the level median line of the rectangular frame is the angle of 0 degrees, the second parameter indicates the starting angle, the third parameter indicates the ending angle, and fill = 0 indicates that the color of the line is white. If the figure is not square according to the coordination, you will get an ellipse.

Besides the arc function, you can also use the chord function for drawing a solid circle.

```
draw.ellipse((150,65,190,105), fill = 0)
```

The first parameter is the coordination of the enclosing rectangle. The second and third parameters are the beginning and end degrees of the circle. The fourth parameter is the fill color of the circle.

- Character.

The ImageFont module needs to be imported and instantiated:

```
Font1 = ImageFont.truetype("../Font/Font01.ttf",25)
Font2 = ImageFont.truetype("../Font/Font01.ttf",35)
Font3 = ImageFont.truetype("../Font/Font02.ttf",32)
```

You can use the fonts of Windows or other fonts which is in ttc format..
Note: Each character library contains different characters; If some characters cannot be displayed, it is recommended that you can refer to the encoding set ro used. To draw English characters, you can directly use the fonts; for Chinese characters, you need to add a symbol u:

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

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

- read local image

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

The parameter is the image path.

- Other functions.

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

Using with STM32

Software description

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

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

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

Program description

Underlying hardware interface

- Data type

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

- Module initialization and exit processing

```
UBYTE   System_Init(void);
void    System_Exit(void);
Note:
1.here is some GPIO processing before and after using the LCD screen.
2.After the System_Exit(void) function is used, the OLED display will be t
urned off;
```

- Write and read GPIO

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

- SPI write data

```
UBYTE    SPI4W_Write_Byte(uint8_t value);
```

The upper application

For the screen, if you need to draw pictures, display Chinese and English characters, display pictures, etc., you can use the upper application to do, and we provide some basic functions here about some graphics processing in the directory STM32\STM32F103RB\User\GUI_DEV\GUI_Paint.c(.h)
Note: Because of the size of the internal RAM of STM32 and arduino, the GUI is directly written to the RAM of the LCD.

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

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

```
void Paint_NewImage(UWORD Width, UWORD Height, UWORD Rotate, UWORD Color)
Parameters:
    Width: image buffer Width;
    Height: the Height of the image buffer;
    Rotate: Indicates the rotation Angle of an image
    Color: the initial Color of the image;
```

- Set the clear screen function, 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 th
e screen to a certain color;
```

- Set the drawing pixel function

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

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

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

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

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

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

```
void Paint_SetMirroring(UBYTE mirror)
Parameters:
    Mirror: indicates the image mirroring mode. MIRROR_NONE, MIRROR_HORIZO
NTAL, MIRROR_VERTICAL, MIRROR_ORIGIN correspond to no mirror, horizontal m
irror, vertical mirror, and about image center mirror respectively.
```

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

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.

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 Yen
d, 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 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, providing a default of eight size poin
ts
        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 poi
nt or from the bottom left corner of the point to the right and up
        typedef enum {
            DOT_FILL_AROUND  = 1,
            DOT_FILL_RIGHTUP,
        } DOT_STYLE;
```

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

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, 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.

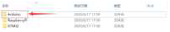
```
void Paint_DrawTime(UWORD Xstart, UWORD Ystart, PAINT_TIME *pTime, sFONT*
Font, UWORD Color_Background,  UWORD Color_Foreground)
Parameters:
    Xstart: the x-coordinate of the left vertex of a character
    Ystart: the Y coordinate of the font's left vertex
    PTime: display time, here defined 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

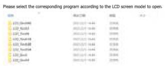
Note: The demos are all tested on Arduino uno. If you need other types of Arduino, you need to determine whether the connected pins are correct.

Run program

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



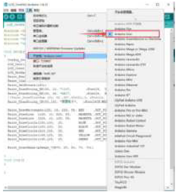
Please select the corresponding program according to the LCD screen model to open.



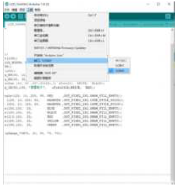
You can view test programs for all screen sizes, sorted by size:

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

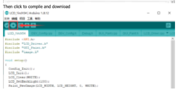
Open the program, select the development board model Arduino UNO.



Select the corresponding COM port



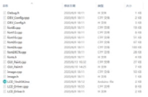
Then click to compile and download



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 delaytime)
```

millisecond level delay.

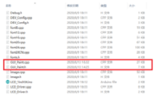
- SPI output data

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

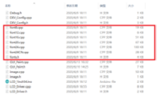
The parameter is char type, occupying 8 bits.

The upper application

For the screen, if you need to draw pictures, display Chinese and English characters, display pictures, etc., you can use the upper application to do, and we provide some basic functions here about some graphics processing in the directory GUI_Paint.c(.h)
Note: Because of the size of the internal RAM of STM32 and Arduino, the GUI is directly written to the RAM of the LCD.



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 Width, UWORD Height, UWORD Rotate, UWORD Color)
Parameters:
    Width: image buffer Width;
    Height: the Height of the image buffer;
    Rotate: Indicates the rotation Angle of an image
    Color: the initial Color of the image;
```

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

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

- Set the drawing pixel function.

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

- Select image buffer: the purpose of the selection is that you can create multiple image attributes, image buffer can exist multiple, 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_HORIZO
NTAL, MIRROR_VERTICAL, MIRROR_ORIGIN correspond to no mirror, horizontal m
irror, 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 Yen
d, 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 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, providing a default of eight size poin
ts
        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 poi
nt or from the bottom left corner of the point to the right and up
        typedef enum {
                DOT_FILL_AROUND  = 1,
                DOT_FILL_RIGHTUP,
        } DOT_STYLE;
```

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

```
void Paint_DrawLine(UWORD Xstart, UWORD Ystart, UWORD Xend, UWORD Yend, 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 eig
ht 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 s
traight 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 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.

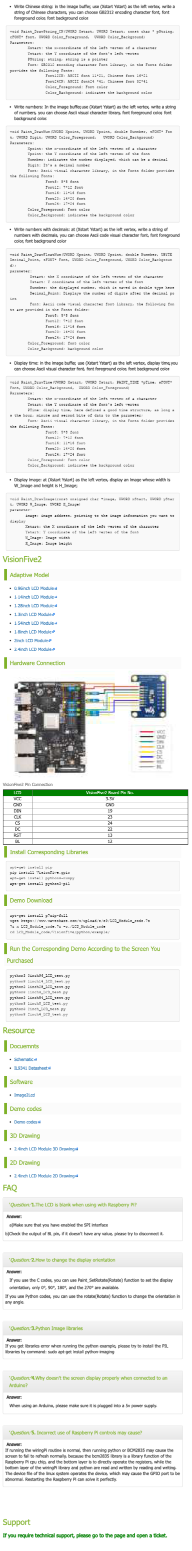
```
void Paint_DrawCircle(UWORD X_Center, UWORD Y_Center, UWORD Radius, UWORD
Color, DOT_PIXEL Line_width,  DRAW_FILL Draw_Fill)
Parameters:
        X_Center: the x-coordinate of the center of a circle
        Y_Center: Y coordinate of the center of a circle
        Radius: indicates the Radius of a 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
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 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 numbers with decimals: at (Xstart Ystart) as the left vertex, write a string of numbers with decimals, you can choose Ascii code visual character font, font foreground color, font background color

```
void Paint_DrawFloatNum(UWORD Xpoint, UWORD Ypoint, double Nummber, UBYTE
Decimal_Point, sFONT* Font, UWORD Color_Foreground, UWORD Color_Backgroun
d);
parameter:
         Xstart: the X coordinate of the left vertex of the character
         Ystart: Y coordinate of the left vertex of the font
         Nummber: the displayed number, which is saved in double type here
         Decimal_Point: Displays the number of digits after the decimal po
int
         Font: Ascii code visual character font library, the following fon
ts are provided in the Fonts folder:
                Font8: 5*8 font
                Font12: 7*12 font
                Font16: 11*16 font
                Font20: 14*20 font
                Font24: 17*24 font
        Color_Foreground: font color
        Color_Background: background color
```

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

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

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

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

VisionFive2

Adaptive Model

- 0.96inch LCD Module
- 1.14inch LCD Module
- 1.28inch LCD Module
- 1.3inch LCD Module
- 1.54inch LCD Module
- 1.8inch LCD Module
- 2inch LCD Module
- 2.4inch LCD Module

Hardware Connection

VisionFive2 Pin Connection

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

Install Corresponding Libraries

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

Demo Download

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

Run the Corresponding Demo According to the Screen You Purchased

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

Resource

Docuemnts

- Schematic
- IL9341 Datasheet

Software

- Image2Lcd

Demo codes

- Demo codes

3D Drawing

- 2.4inch LCD Module 3D Drawing

2D Drawing

- 2.4inch LCD Module 2D Drawing

FAQ

'*Question:*'**1.**The LCD is blank when using with Raspberry Pi?

Answer:

a)Make sure that you have enabled the SPI interface

b)Check the output of BL pin, if it doesn't have any value, please try to disconnect it.

'*Question:*'**2.**How to change the display orientation

Answer:

If you use the C codes, you can use Paint_SetRotate(Rotate) function to set the display orientation, only 0°, 90°, 180°, and the 270° are available.

If you use Python codes, you can use the rotate(Rotate) function to change the orientation in any angle.

'*Question:*'**3.**Python Image libraries

Answer:
If you get libraries error when running the python example, please try to install the PIL libraries by command: sudo apt-get install python-imaging

'*Question:*'**4.**Why doesn't the screen display properly when connected to an Arduino?

Answer:

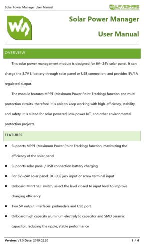
When using an Arduino, please make sure it is plugged into a 5v power supply.

'*Question:*'**5.** Incorrect use of Raspberry Pi controls may cause?

Answer:
If running the wiringPi routine is normal, then running python or BCM2835 may cause the screen to fail to refresh normally, because the bcm2835 library is a library function of the Raspberry Pi cpu chip, and the bottom layer is to directly operate the registers, while the bottom layer of the wiringPi library and python are read and written by reading and writing. The device file of the linux system operates the device, which may cause the GPIO port to be abnormal. Restarting the Raspberry Pi can solve it perfectly.

Support

If you require technical support, please go to the page and open a ticket.



Solar Power Manager User Manual

Solar Power Manager
User Manual

OVERVIEW

This solar power management module is designed for 6V~24V solar panel. It can charge the 3.7V Li battery through solar panel or USB connection, and provides 5V/1A regulated output.

The module features MPPT (Maximum Power Point Tracking) function and multi protection circuits, therefore, it is able to keep working with high-efficiency, stability, and safety. It is suited for solar powered, low-power IoT, and other environmental protection projects.

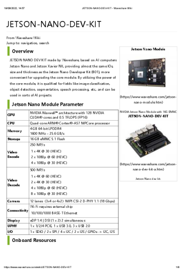
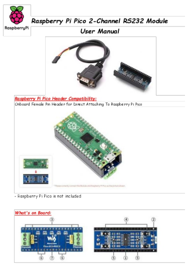
FEATURES

- Supports MPPT (Maximum Power Point Tracking) function, maximizing the efficiency of the solar panel
- Supports solar panel / USB connection battery charging
- For 6V~24V solar panel, DC-002 jack input or screw terminal input
- Onboard MPPT SET switch, select the level closed to input level to improve charging efficiency
- Two 5V output interfaces: pinheaders and USB port
- Onboard high capacity aluminum electrolytic capacitor and SMD ceramic capacitor, reducing the ripple, stable performance

Version: V1.0 **Date:** 2019.02.20 **1 / 6**

Waveshare Solar Power Manager User Manual

User manual for the Waveshare Solar Power Manager module, detailing its features, specifications, board components, usage instructions, and applications for solar energy charging with 6V-24V solar panels and 3.7V Li batteries.

 JETSON-NANO-DEV-KIT - Waveshare Wiki JETSON-NANO-DEV-KIT From Waveshare Wiki Jump to: navigation, search Overview JETSON NANO DEV KIT made by Waveshare, based on AI computers Jetson Nano and Jetson Xavier NX, providing almost the same IOs, size and thickness as the Jetson Nano Developer Kit (B01), more convenient for upgrading the core module. By utilizing the power of the core module, it is qualified for fields like image classification, object detection, segmentation, speech processing, etc, and can be used in sorts of AI projects. Jetson Nano Module Parameter GPU NVIDIA Maxwell™ architecture with 128 NVIDIA CUDA® cores and 0.5 TFLOPS (FP16) CPU Quad-core ARM® Cortex®-A57 MPCore processor Memory 4GB 64-bit LPDDR4 1600 MHz – 25.6 GB/s Storage 16 GB eMMC 5.1 Flash Video Encode 250 MP/s 1 x 4K @ 30 (HEVC) 2 x 1080p @ 60 (HEVC) 4 x 1080p @ 30 (HEVC) Video Decode 500 MP/s 1 x 4K @ 60 (HEVC) 2 x 4K @ 30 (HEVC) 4 x 1080p @ 60 (HEVC) 8 x 1080p @ 30 (HEVC) Camera 12 lanes (3x4 or 4x2) MIPI CSI-2 D-PHY 1.1 (18 Gbps) Connectivity Wi-Fi requires external chip 10/100/1000 BASE-T Ethernet Display eDP 1.4 | DSI (1 x 2) 2 simultaneous UPHY 1 x 1/2/4 PCIE, 1 x USB 3.0, 3 x USB 2.0 I/O 1 x SDIO / 2 x SPI / 4 x I2C / 2 x I2S / GPIOs -> I2C, I2S Onboard Resources Jetson Nano Module (https://www.waveshare.com/jetson-nano-module.htm) NVIDIA Jetson Nano Module with 16G EMMC JETSON-NANO-DEV-KIT (https://www.waveshare.com/jetson-nano-dev-kit-a.htm) Jetson Nano dev kit	Waveshare Jetson Nano Dev Kit: Overview, Setup, and Resources A comprehensive guide to the Waveshare Jetson Nano Developer Kit, covering its overview, hardware specifications, software setup using SDK Manager, camera configuration, and troubleshooting.
 RaspberryPi Raspberry Pi Pico 2-Channel RS232 Module User Manual Raspberry Pi Pico Header Compatibility: Onboard Female Pin Header for Direct Attaching To Raspberry Pi Pico - Raspberry Pi Pico is not included What's on Board:	Raspberry Pi Pico 2-Channel RS232 Module User Manual User manual for the Raspberry Pi Pico 2-Channel RS232 Module, detailing its features, compatibility, onboard components, and pinout definition for connecting RS232 devices to the Raspberry Pi Pico.
 RS232/485 To ETH 用户手册 微雪电子 RS232/485 To ETH 用户手册 深圳市微雪电子有限公司 1 / 44 www.waveshare.net	Waveshare RS232/485 to Ethernet Converter User Manual This is an industrial-grade RS232/485 to Ethernet module from Waveshare, enabling bidirectional transparent data transmission between RS232, RS485, and RJ45 Ethernet ports. It features a 32-bit ARM M4 processor, 10/100M Ethernet, multiple operating modes (TCP Server/Client, UDP, HTTPD), Modbus gateway, and Websocket support, offering high speed, low power consumption, and stability for industrial IoT applications.