1.54inch OLED SSD1309 SPI&IIC Module MSP154W&MSP154B User Manual

LCDWIKI

Introduction to OLED

OLED is an Organic Light-Emitting Diode (OLED). OLED display technology has the advantages of self-illumination, wide viewing angle, almost infinite contrast, low power consumption, high reaction speed, flexible panel, wide temperature range, simple structure and process, etc. A generation of flat panel display emerging application technology.

OLED display is different from traditional LCD display, it can self-illuminate, so no backlight is needed, which makes OLED display

The display is thinner than the LCD display and has a better display.

Product Description

The OLED module has a display size of 1.54 inches and has a 128x64 resolution. Three-wire system, 4-wire SPI and IIC communication modes can be selected, and the driver IC is SSD1309. Contains three modules in black, blue or yellow and blue.

Product Features

- 1.54 inch OLED screen with black and white, black or blue color display
- 128x64 resolution for clear display and high contrast
- Large viewing angle: greater than 160° (one screen with the largest viewing angle in the display)
- Wide voltage supply (3V~5V), compatible with 3.3V and 5V logic levels, no level shifting chip required
- The default is 4-wire SPI bus, which can choose IIC bus
- Ultra-low power consumption: normal display is only 0.06W (far below the TFT display)
- Military-grade process standards, long-term stable work

LCDWIKI

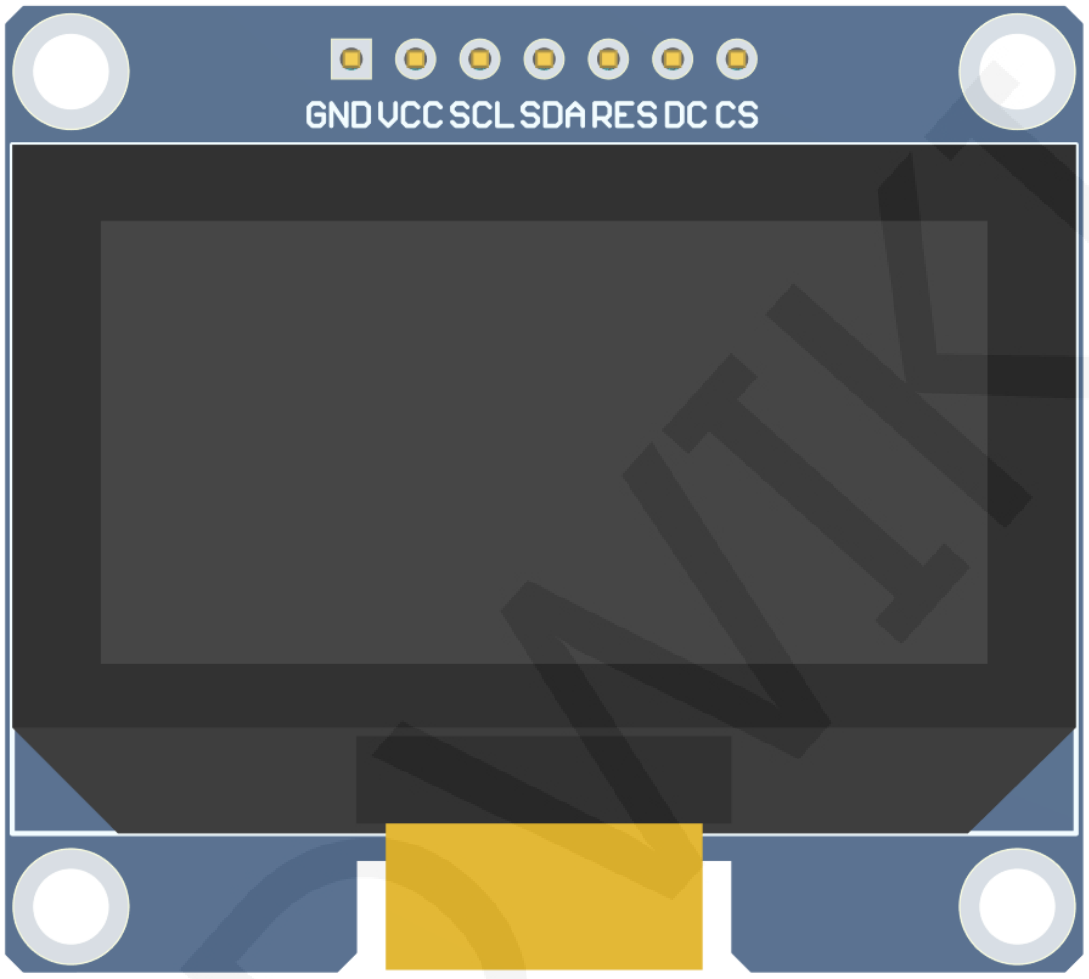
- Provides a rich sample program for STM32, C51, Arduino, Raspberry Pi and MSP430 platforms
- Provide underlying driver technical support

Product Parameters

Name	Description
Display Color	White, blue
SKU	MSP154W MSP154B
Screen Size	1.54(inch)
Type	OLED
Driver IC	SSD1309
Resolution	128*64(Pixel)
Module Interface	4-line SPI、IIC interface
Active Area	35.052x17.516(mm)
Touch Screen Type	have no touch screen
Touch IC	have no touch IC
Module PCB Size	42.40x38.00(mm)
Angle of view	>160°
Operating Temperature	-20℃~60℃
Storage Temperature	-30℃~70℃
Operating Voltage	3.3V/5V
Power Consumption	TBD
Product Weight(With packaging)	12(g)

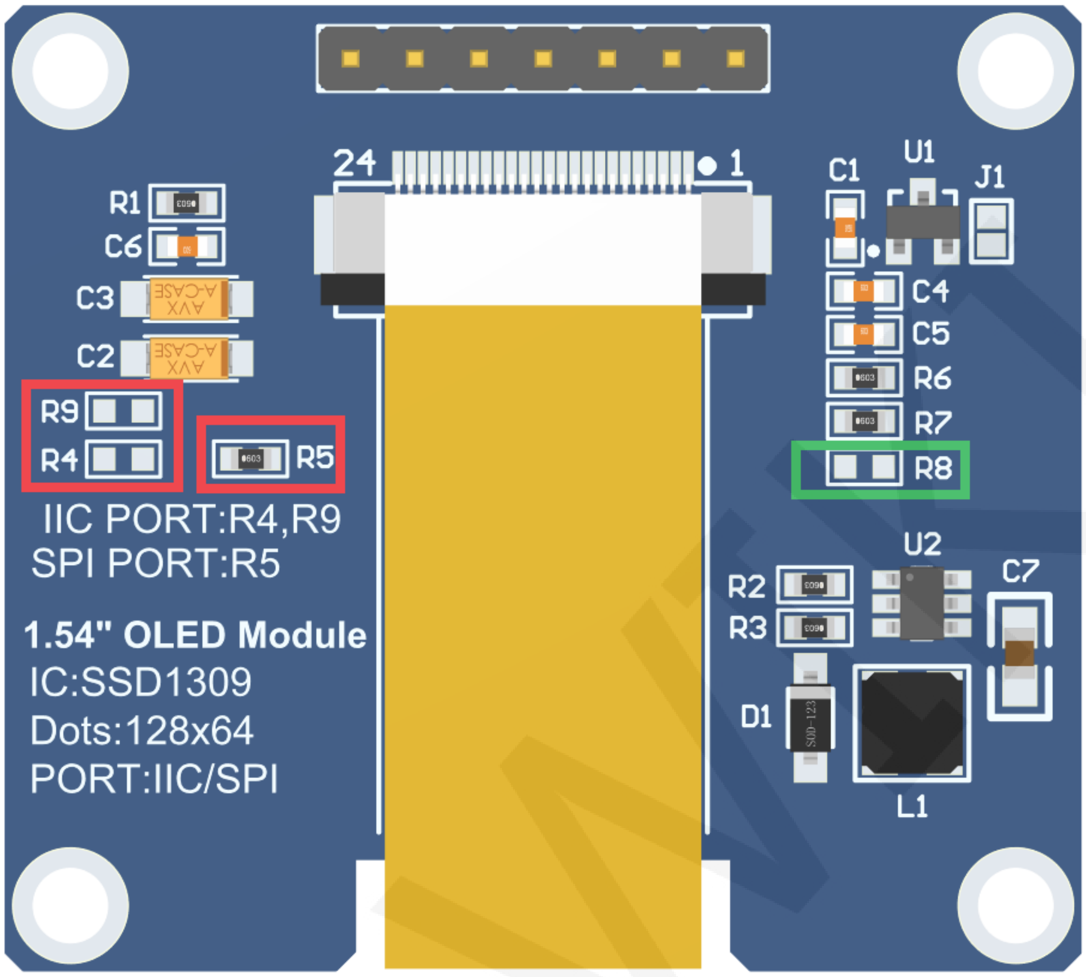
LCDWIKI

Interface Description



Picture1. Module pin Label picture

LCDWIKI



Picture 2. Rear view of the module

NOTE:

1. **This module supports IIC and 4-wire SPI interface bus mode switching (as shown in the red box in Picture 2). The details are as follows:**
   - A. **Using 4.7K resistance to solder only R5 resistors, then choose 4-wire SPI bus interface (default);**
   - B. **Using 4.7K resistance to solder only R4 and R9 resistors, then select the IIC bus interface;**
2. **After the interface bus mode is switched, you need to select the corresponding software and the corresponding wiring pins (as shown in Picture 1) for the module to operate normally. The corresponding wiring pins are described as follows:**

A. select the 4-wire SPI bus interface, all pins need to be used;

B. To select the IIC bus interface, only five pins GND, VCC, SCL, SDA and res need to be used. Connect the CS pin to the power supply and ground. The DC pin can be used to select the IIC slave device address. Select 0x7a for high level and 0x78 for low level;

3. Solder R8 resistor (as shown in the green box in Picture2), then CS pin does not need to be connected

important:

1. The following pin numbers 1~7 refer to the module pin number of our company with PCB backplane. If you purchase a bare screen, please refer to the pin definition of the bare screen specification, refer to the wiring according to the signal type instead of directly according to the following. The module pin number is used for wiring. For example: CS is 7 feet on our module. It may be x pin on different size bare screen. The following wiring instructions tell you that the CS signal is connected to the A5 pin of the MCU.
2. About VCC supply voltage: The OLED display module can be connected to 3.3V or 5V.

Number	Module Pin	Pin Description
1	**GND**	OLED power ground
2	**VCC**	OLED power positive (3.3V~5V)
3	**SCL**	OLED SPI and IIC bus clock signals
4	**SDA**	OLED SPI and IIC bus data signals
5	**RES**	OLED reset signal, low level reset (this pin need to connected to the high level (can be connected to the VCC) when selecting IIC bus)
6	**DC**	Select SPI bus as command / data input selection signal, high level: data, low level: command; When selecting the IIC bus, this pin can be used to select the address of the IIC slave device,

LCDWIKI

		which is connected to the high level selection 0x7a and the low level selection 0x78;
7	**CS**	OLED chip selection signal, low level enabled; Solder R8 resistor (as shown in the green box in Picture 2), then this pin does not need to be connected. Do not weld R8 resistor, select IIC interface, then this pin is grounded

Hardware Configuration

The hardware circuit of the module is composed of four parts: OLED display control circuit, OLED boost circuit, pin array interface, and power supply voltage stabilizing circuit.

OLED display control circuit is mainly used to control OLED display, including chip selection, reset, data and command transmission control, and interface selection.

The OLED boosting circuit is used to boost an input voltage to an OLED light emitting voltage.

The pin array interface is used for external connection of the main control development board.

The power supply voltage stabilizing circuit is used for 3.3V voltage stabilizing power supply.The OLED module adopts 4-wire SPI communication mode by default. In addition, it can also select IIC communication mode. The hardware is configured with 7 pins. Different communication methods are used, and the selected pins are different (see the interface description for details).

working principle

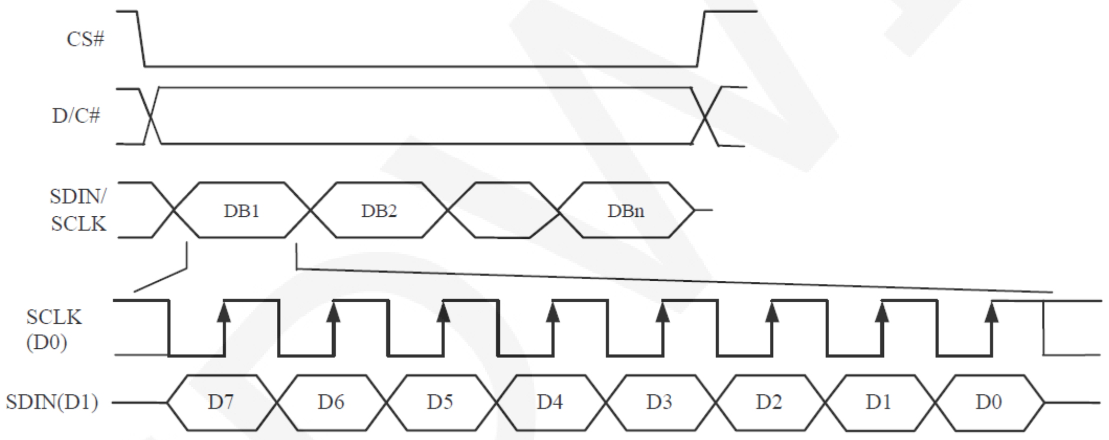
1. Introduction to SSD1309 Controller

The SSD1309 is an OLED/PLED controller that supports a maximum resolution of 128*64 and a 1024-byte GRAM. Support 8-bit 6800 and 8-bit 8080 parallel port data bus, also supports 3-wire and 4-wire SPI serial bus and I2C bus. Since parallel control requires a large number of IO ports, the most commonly used are the SPI serial bus and the I2C bus. It supports vertical scrolling and can be used in small portable devices such as mobile phones, MP3 players and more.

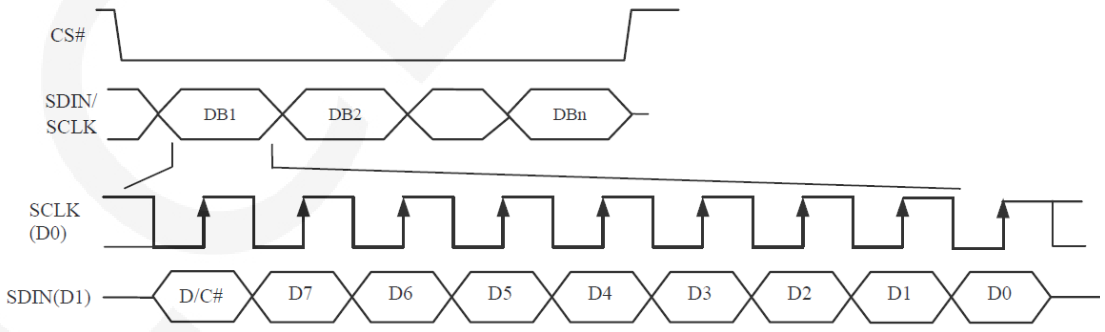
The SSD1309 controller uses 1 bit to control a pixel display, so each pixel can only display black and white or black and blue. The displayed RAM is divided into 8 pages, with 8 lines per page and 128 pixels per line. When setting pixel data, you need to specify the page address first, and then specify the column low address and column height address respectively, so set 8 pixels in the vertical direction at the same time. In order to be able to flexibly control the pixel points at any position, the software first sets a global one-dimensional array of the same size as the display RAM, first maps the pixel point data to the global array, and the process uses the OR or the operation to ensure that the global array is written before. The data is not corrupted, and the data of the global array is then written to the GRAM so that it can be displayed through the OLED.

2. Introduction to SPI communication protocol

The 4-wire SPI bus write mode timing is shown in the following figure:



The 3-wire SPI bus write mode timing is shown in the following figure:



As can be seen from the above timing diagram, the difference between the 3-wire SPI and the 4-wire SPI is as follows:

The 3-wire SPI does not have a D/C# signal, and its D/C# signal is input by SDIN, which first transmits 1 bit of D/C# data, followed by an 8-bit command or data. The 4-wire D/C# signal is directly input by D/C#.

CS# is a slave chip select, and the chip is enabled only when CSX is low.

D/C# is the data/command control pin of the chip. When DCX is low, the command is written. When it is high, the data is written.

SCLK is the SPI bus clock, and each rising edge transmits 1 bit of data;

SDIN is the data transmitted by SPI, and it transmits 8-bit data at a time.The high position is in front and transmitted first.For SPI communication, the data has a transmission timing, that is, a combination of clock phase (CPHA) and clock polarity (CPOL):

The CPOL level determines the idle state level of the serial synchronous clock, CPOL = 0, which is low. CPOL does not have a lot of impact on the transport protocol;

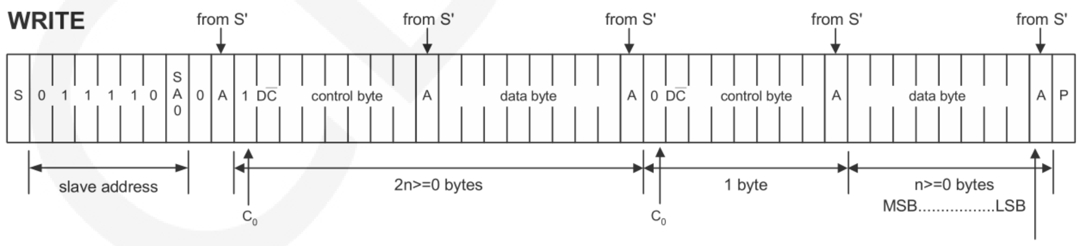
The level of CPHA determines whether the serial synchronous clock is acquired on the first clock transition edge or the second clock transition edge.

When CPHL = 0, data acquisition is performed on the first edge of the transition;

The combination of the two becomes the four SPI communication methods. SPI0 is usually used in China, that is, CPHL = 0, CPOL = 0.

3. Introduction to IIC Communication Protocol

The process of writing data on the IIC bus is shown in the following figure:



After the IIC bus starts working, the slave device address is sent first. After receiving the slave device response, it then sends a control byte to inform the slave device whether the next data to be sent is a command written to the IC register or written. The RAM data, after receiving the slave device response, then sends a value of multiple bytes until the transmission is completed and the IIC bus stops working.

among them:

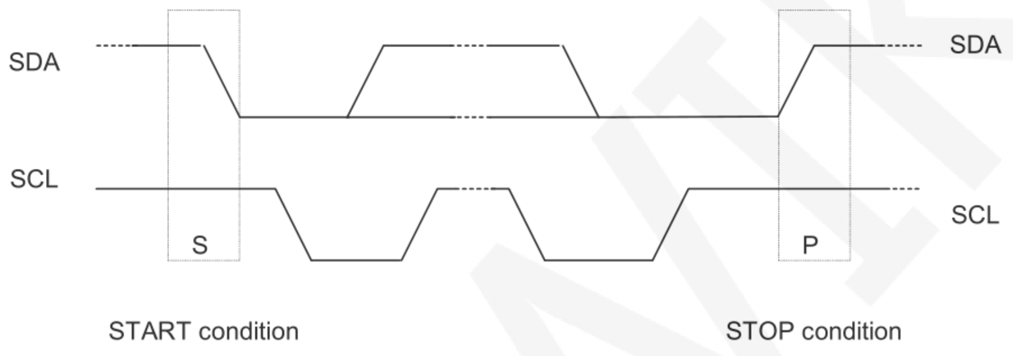
C0=0: This is the last control byte, and all the data bytes sent in the following are all data bytes.

C0=1: The next two bytes to be sent are the data byte and another control byte.

D/C(——)=0: is the register command operation byte

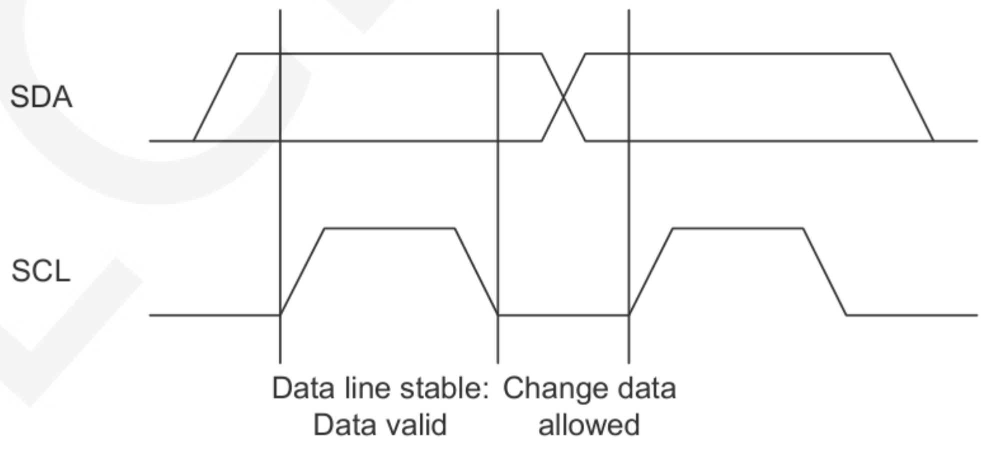
D/C(——)=1: operation byte for RAM data

The IIC start and stop timing diagrams are as follows:



When the data line and the clock line of the IIC are both kept at a high level, the IIC is in an idle state. At this time, the data line changes from a high level to a low level, and the clock line continues to be at a high level, and the IIC bus starts data transmission. When the clock line is held high, the data line changes from low to high, and the IIC bus stops data transmission.

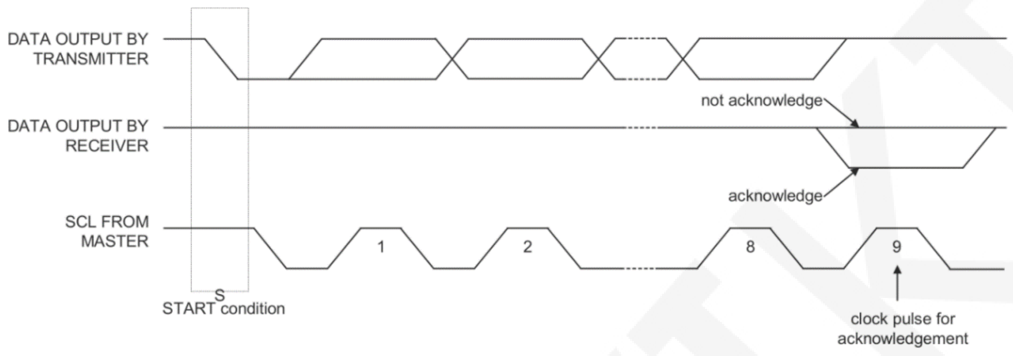
The timing diagram for the IIC to send a bit of data is as follows:



LCDWIKI

Each clock pulse (the process of pulling high and pulling low) sends 1 bit of data. When the clock line is high, the data line must remain stable, and the data line is allowed to change when the clock line is low.

The ACK transmission timing diagram is as follows:



When the master waits for the ACK of the slave, it needs to keep the clock line high. When the slave sends an ACK, keep the data line low.

Instructions for use

1. Arduino instructions

Wiring instructions:

See the interface description for pin assignments.

Arduino UNO microcontroller test program wiring instructions			
Number	Module Pin	Corresponding to UNO development board wiring pins	
		SPI	IIC
1	GND	GND	
2	VCC	5V/3.3V	
3	SCL	13	A5
4	SDA	11	A4
5	RES	3.3V	
6	DC	9	VCC or GND
7	CS	10	GND

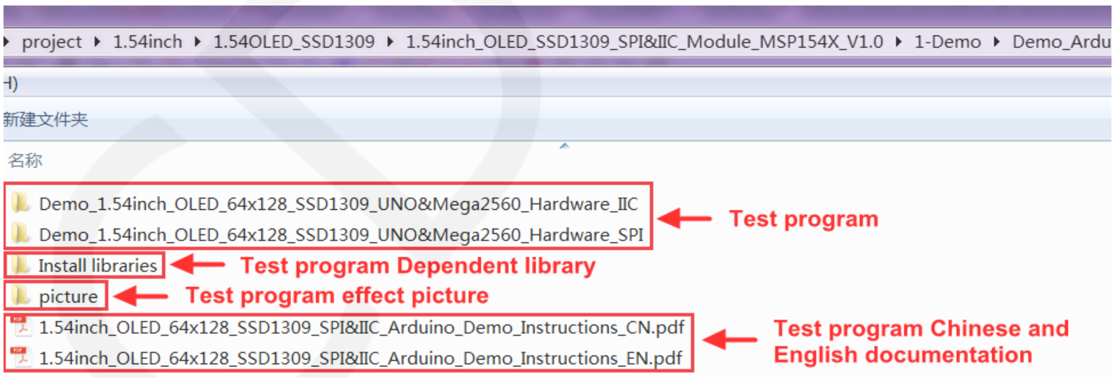
Arduino MEGA2560 microcontroller test program wiring instructions			
Number	Module Pin	Corresponding to MEGA2560 development board wiring pins	
		SPI	IIC
1	GND	GND	
2	VCC	5V/3.3V	
3	SCL	53	21
4	SDA	51	20
5	RES	3.3V	
6	DC	9	VCC or GND
7	CS	10	GND

Operating Steps:

A. Connect the OLED module and the Arduino MCU according to the above wiring instructions, and power on;

B. Select the example you want to test, as shown below:

(Please refer to the test program description document for test program description)


project ▸ 1.54inch ▸ 1.54OLED_SSD1309 ▸ 1.54inch_OLED_SSD1309_SPI&IIC_Module_MSP154X_V1.0 ▸ 1-Demo ▸ Demo_Ardu

新建文件夹

名称

- Demo_1.54inch_OLED_64x128_SSD1309_UNO&Mega2560_Hardware_IIC
- Demo_1.54inch_OLED_64x128_SSD1309_UNO&Mega2560_Hardware_SPI
- Install libraries
- picture
- 1.54inch_OLED_64x128_SSD1309_SPI&IIC_Arduino_Demo_Instructions_CN.pdf
- 1.54inch_OLED_64x128_SSD1309_SPI&IIC_Arduino_Demo_Instructions_EN.pdf

Test program
Test program Dependent library
Test program effect picture
Test program Chinese and English documentation

C. Open the selected sample project, compile and download.

The specific operation methods for the Arduino test program relying on library copy, compile and download are as follows:

http://www.lcdwiki.com/res/PublicFile/Arduino_IDE_Use_Illustration_EN.pdf

D. If the OLED module displays characters and graphics normally, the program runs Successfully;

2. RaspberryPi instructions

Wiring instructions:

See the interface description for pin assignments.

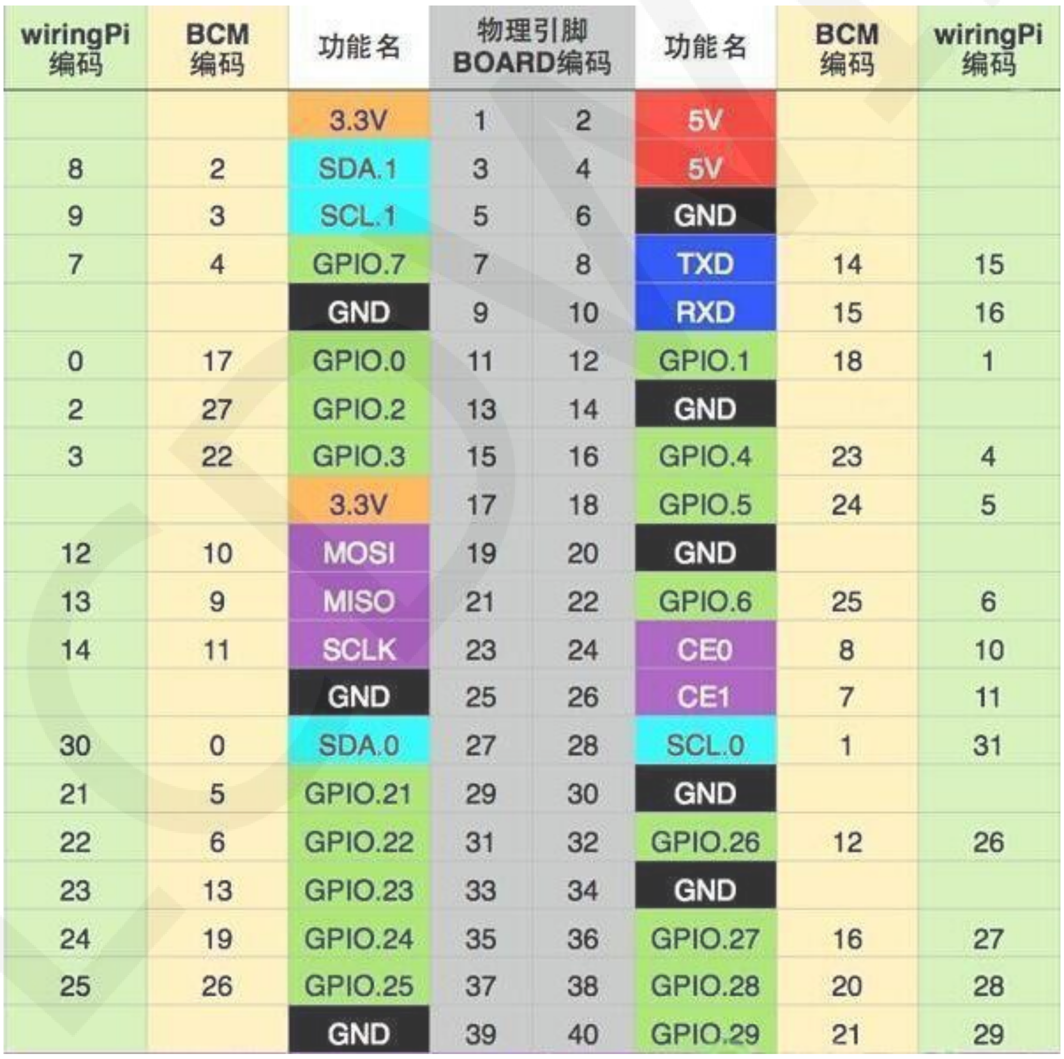
NOTE:

Physical pin refers to the GPIO pin code of the RaspBerry Pi development board.

BCM encoding refers to the GPIO pin coding when using the BCM2835 GPIO library.

WiringPi coding refers to the GPIO pin coding when using the wiringPi GPIO library.

Which GPIO library is used in the code, the pin definition needs to use the corresponding GPIO library code, see Picture 1 GPIO map table for details.



wiringPi 编码	BCM 编码	功能名	物理引脚 BOARD编码		功能名	BCM 编码	wiringPi 编码
		3.3V	1	2	5V		
8	2	SDA.1	3	4	5V		
9	3	SCL.1	5	6	GND		
7	4	GPIO.7	7	8	TXD	14	15
		GND	9	10	RXD	15	16
0	17	GPIO.0	11	12	GPIO.1	18	1
2	27	GPIO.2	13	14	GND		
3	22	GPIO.3	15	16	GPIO.4	23	4
		3.3V	17	18	GPIO.5	24	5
12	10	MOSI	19	20	GND		
13	9	MISO	21	22	GPIO.6	25	6
14	11	SCLK	23	24	CE0	8	10
		GND	25	26	CE1	7	11
30	0	SDA.0	27	28	SCL.0	1	31
21	5	GPIO.21	29	30	GND		
22	6	GPIO.22	31	32	GPIO.26	12	26
23	13	GPIO.23	33	34	GND		
24	19	GPIO.24	35	36	GPIO.27	16	27
25	26	GPIO.25	37	38	GPIO.28	20	28
		GND	39	40	GPIO.29	21	29

Picture4. GPIO map

Raspberry Pi test program wiring instructions			
Number	Module Pin	Corresponding to development board wiring pin	
		SPI	IIC
1	GND	GND (Physical pin：6,9,14,20,25,30,34,39)	
2	VCC	5V/3.3V （Physical pin：1,2,4）	
3	SCL	Physical pin：23 BCM coding：11 wiringPi coding：14	Physical pin：5 BCM coding：3 wiringPi coding：9
4	SDA	Physical pin：19 BCM coding：10 wiringPi coding：12	Physical pin：3 BCM coding：2 wiringPi coding：8
5	RES	Physical pin：5 BCM coding：3 wiringPi coding：9	Physical pin：23 BCM coding：11 wiringPi coding：14
6	DC	Physical pin：3 BCM coding：2 wiringPi coding：8	VCC or GND
7	CS	Physical pin：24 BCM coding：8 wiringPi coding：10	GND

Operating Steps：

A. open the SPI function of RaspberryPi

Log in to the RaspberryPi using a serial terminal tool (such as putty) and enter the following command:

```
sudo raspi-config
```

Select Interfacing Options->SPI->YES

Start RaspberryPi's SPI kernel driver

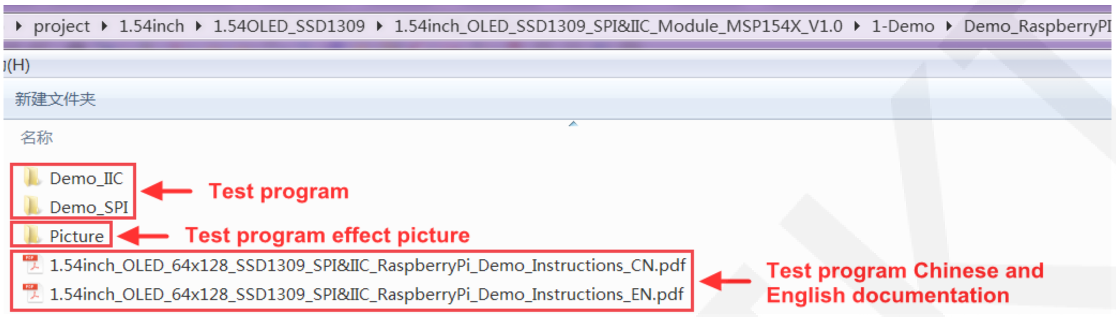
B. install the function library

For detailed installation methods of the bcm2835, wiringPi, and python function libraries of RaspberryPi, see the following documents:

http://www.lcdwiki.com/res/PublicFile/Raspberrypi_Use_Illustration_EN.pdf

C. select the example that needs to be tested, as shown below:

(Please refer to the test program description document for test program description)



project ▸ 1.54inch ▸ 1.54OLED_SSD1309 ▸ 1.54inch_OLED_SSD1309_SPI&IIC_Module_MSP154X_V1.0 ▸ 1-Demo ▸ Demo_RaspberryPI
(H)
新建文件夹
名称
Demo_IIC
Demo_SPI
Test program
Picture
Test program effect picture
1.54inch_OLED_64x128_SSD1309_SPI&IIC_RaspberryPi_Demo_Instructions_CN.pdf
1.54inch_OLED_64x128_SSD1309_SPI&IIC_RaspberryPi_Demo_Instructions_EN.pdf
Test program Chinese and English documentation

D. bcm2835 instructions(Take the 4-wire hardware SPI test program as an example)

a) Connect the OLED module to the RaspberryPi development board according to the above wiring

b) Copy the test program directory

Demo_1.54inch_OLED_64x128_SSD1309_bcm2835_Hardware_4-wire_SPI

to RaspberryPi (can be copied via SD card or via FTP tool (such as FileZilla))

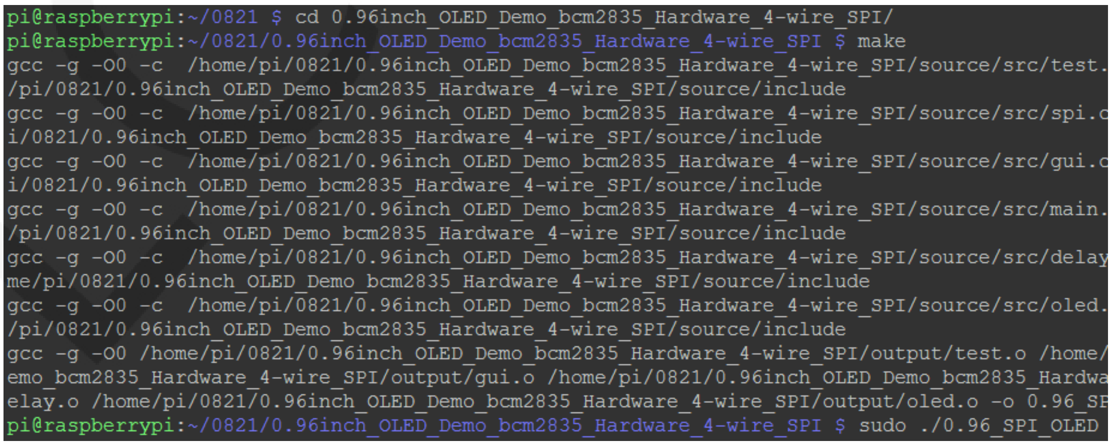
c) Run the following command to run the bcm2835 test program:

```
cd Demo_1.54inch_OLED_64x128_SSD1309_bcm2835_Hardware_4-wire_SPI

make

sudo ./ 1.54_SPI_OLED
```

As shown below:



```
pi@raspberrypi:~/0821 $ cd 0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI/
pi@raspberrypi:~/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI $ make
gcc -g -O0 -c  /home/pi/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI/source/src/test.
/pi/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI/source/include
gcc -g -O0 -c  /home/pi/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI/source/src/spi.c
i/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI/source/include
gcc -g -O0 -c  /home/pi/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI/source/src/gui.c
i/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI/source/include
gcc -g -O0 -c  /home/pi/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI/source/src/main.
/pi/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI/source/include
gcc -g -O0 -c  /home/pi/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI/source/src/delay
me/pi/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI/source/include
gcc -g -O0 -c  /home/pi/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI/source/src/oled.
/pi/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI/source/include
gcc -g -O0 /home/pi/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI/output/test.o /home/
emo_bcm2835_Hardware_4-wire_SPI/output/gui.o /home/pi/0821/0.96inch_OLED_Demo_bcm2835_Hardwa
elay.o /home/pi/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI/output/oled.o -o 0.96_SP
pi@raspberrypi:~/0821/0.96inch_OLED_Demo_bcm2835_Hardware_4-wire_SPI $ sudo ./0.96_SPI_OLED
```

E. wiringPi instructions(Take the 4-wire hardware SPI test program as an example)

a) Connect the OLED module to the RaspberryPi development board according to the above wiring

b) Copy the test program directory

Demo_1.54inch_OLED_64x128_SSD1309_wiringPi_Hardware_4-wire_SPI

to RaspberryPi (can be copied via SD card or via FTP tool (such as FileZilla))

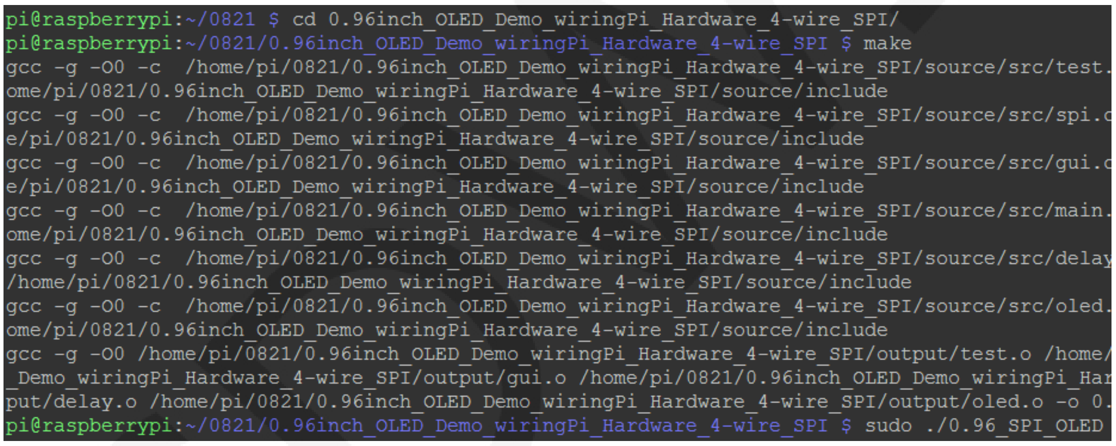
c) Run the following command to run the wiringPi test program:

```
cd Demo_1.54inch_OLED_64x128_SSD1309_wiringPi_Hardware_4-wire_SPI

make

sudo ./ 1.54_SPI_OLED
```

As shown below:



```
pi@raspberrypi:~/0821 $ cd 0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI/
pi@raspberrypi:~/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI $ make
gcc -g -O0 -c  /home/pi/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI/source/src/test.
ome/pi/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI/source/include
gcc -g -O0 -c  /home/pi/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI/source/src/spi.c
e/pi/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI/source/include
gcc -g -O0 -c  /home/pi/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI/source/src/gui.c
e/pi/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI/source/include
gcc -g -O0 -c  /home/pi/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI/source/src/main.
ome/pi/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI/source/include
gcc -g -O0 -c  /home/pi/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI/source/src/delay
/home/pi/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI/source/include
gcc -g -O0 -c  /home/pi/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI/source/src/oled.
ome/pi/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI/source/include
gcc -g -O0 /home/pi/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI/output/test.o /home/
_Demo_wiringPi_Hardware_4-wire_SPI/output/gui.o /home/pi/0821/0.96inch_OLED_Demo_wiringPi_Har
put/delay.o /home/pi/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI/output/oled.o -o 0.
pi@raspberrypi:~/0821/0.96inch_OLED_Demo_wiringPi_Hardware_4-wire_SPI $ sudo ./0.96_SPI_OLED
```

F. python instructions(Take the 4-wire hardware SPI test program as an example)

a) The image processing library PIL needs to be installed before running the python test program. The specific installation method is as follows:

http://www.lcdwiki.com/res/PublicFile/Python_Image_Library_Install_Illustration_EN.pdf

b) Connect the OLED module to the RaspberryPi development board as described above.

c) Copy the test program directory

Demo_1.54inch_OLED_64x128_SSD1309_python_Hardware_4-wire_SPI

to RaspberryPi (either via SD card or via FTP tool (such as FileZilla))

d) Run the following command to run 3 python test programs separately:

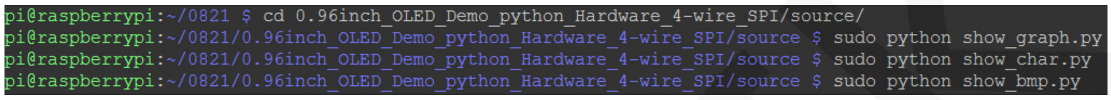
```
cd Demo_1.54inch_OLED_64x128_SSD1309_python_Hardware_4-wire_SPI/source

sudo python show_graph.py

sudo python show_char.py

sudo python show_bmp.py
```

As shown below:



```
pi@raspberrypi:~/0821 $ cd 0.96inch_OLED_Demo_python_Hardware_4-wire_SPI/source/
pi@raspberrypi:~/0821/0.96inch_OLED_Demo_python_Hardware_4-wire_SPI/source $ sudo python show_graph.py
pi@raspberrypi:~/0821/0.96inch_OLED_Demo_python_Hardware_4-wire_SPI/source $ sudo python show_char.py
pi@raspberrypi:~/0821/0.96inch_OLED_Demo_python_Hardware_4-wire_SPI/source $ sudo python show_bmp.py
```

3. STM32 instructions

Wiring instructions:

See the interface description for pin assignments.

STM32F103C8T6 microcontroller test program wiring instructions			
Number	**Module Pin**	**Corresponding to STM32F103C8T6 development board wiring pin**	
		SPI	**IIC**
1	**GND**	GND	
2	**VCC**	3.3V/5V	
3	**SCL**	PA5	
4	**SDA**	PA7	
5	**RES**	PB8	
6	**DC**	PB7	VCC or GND
7	**CS**	PB9	GND

STM32F103RCT6 microcontroller test program wiring instructions		
Number	**Module Pin**	**Corresponding to MiniSTM32 development board wiring pin**

		SPI	IIC
1	**GND**	GND	
2	**VCC**	3.3V/5V	
3	**SCL**	PB13	
4	**SDA**	PB15	
5	**RES**	PB12	
6	**DC**	PB10	VCC or GND
7	**CS**	PB11	GND

STM32F103ZET6 microcontroller test program wiring instructions

Number	Module Pin	**Corresponding to Elite STM32 development board wiring pin**	
		SPI	**IIC**
1	**GND**	GND	
2	**VCC**	3.3V/5V	
3	**SCL**	PB13	
4	**SDA**	PB15	
5	**RES**	PB12	
6	**DC**	PB10	VCC or GND
7	**CS**	PB11	GND

STM32F407ZGT6 microcontroller test program wiring instructions

Number	Module Pin	**Corresponding to Explorer STM32F4 development board wiring pin**	
		SPI	**IIC**
1	**GND**	GND	

LCDWIKI

2	**VCC**	3.3V/5V	
3	**SCL**	PB3	
4	**SDA**	PB5	
5	**RES**	PB12	
6	**DC**	PB14	VCC or GND
7	**CS**	PB15	GND

STM32F429IGT6 microcontroller test program wiring instructions

Number	Module Pin	**Corresponding to Apollo STM32F4/F7 development board wiring pin**	
		SPI	**IIC**
1	**GND**	GND	
2	**VCC**	3.3V/5V	
3	**SCL**	PF7	
4	**SDA**	PF9	
5	**RES**	PD12	
6	**DC**	PD5	VCC or GND
7	**CS**	PD11	GND

STM32F767IGT6 and STM32H743IIT6 microcontroller test program wiring instructions

Number	Module Pin	**Corresponding to Apollo STM32F4/F7 development board wiring pin**
		SPI
1	**GND**	GND
2	**VCC**	3.3V/5V
3	**SCL**	PB13
4	**SDA**	PB15

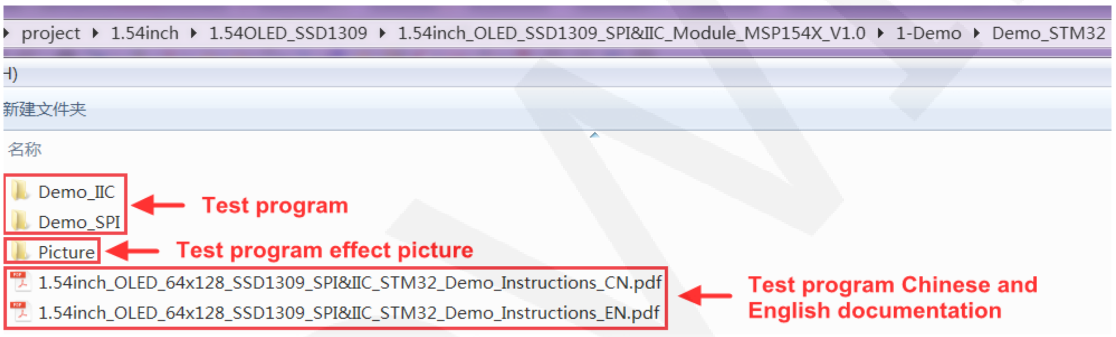
5	**RES**	PD12
6	**DC**	PD5
7	**CS**	PD11

Operating Steps:

A. Connect the IPS module and the STM32 MCU according to the above wiring instructions, and power on;

B. Select the test example according to the model of the microcontroller, as shown in the following figure:

(Please refer to the test program description document in the test package for the test program description)


project ▸ 1.54inch ▸ 1.54OLED_SSD1309 ▸ 1.54inch_OLED_SSD1309_SPI&IIC_Module_MSP154X_V1.0 ▸ 1-Demo ▸ Demo_STM32

新建文件夹

名称

Demo_IIC
Demo_SPI
Picture
1.54inch_OLED_64x128_SSD1309_SPI&IIC_STM32_Demo_Instructions_CN.pdf
1.54inch_OLED_64x128_SSD1309_SPI&IIC_STM32_Demo_Instructions_EN.pdf

Test program
Test program effect picture
Test program Chinese and English documentation

C. Open the selected test program project, compile and download;

detailed description of the STM32 test program compilation and download can be found in the following document:

http://www.lcdwiki.com/res/PublicFile/STM32_Keil_Use_Illustration_EN.pdf

D. If the OLED module displays characters and graphics normally, the program runs successfully;

4. C51 instructions

Wiring instructions:

See the interface description for pin assignments.

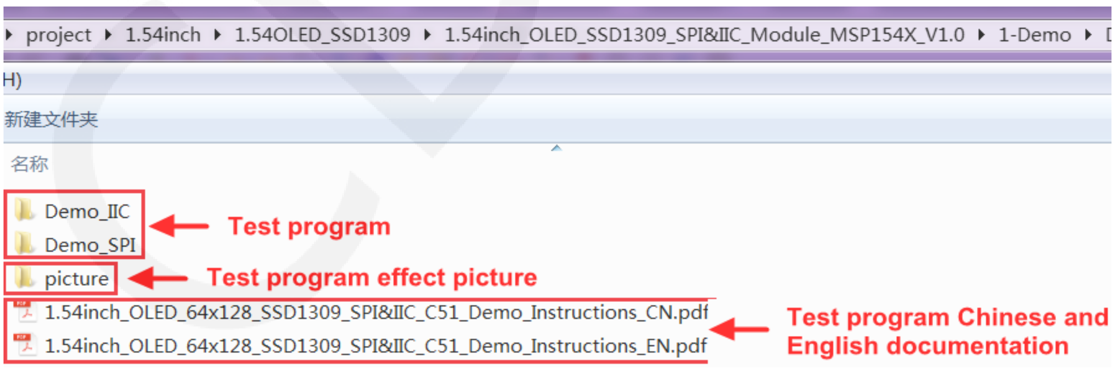
STC89C52RC and STC12C5A60S2 microcontroller test program wiring instructions			
Number	Module Pin	Corresponding to STC89/STC12 development board wiring pin	
		SPI	IIC
1	GND	GND	
2	VCC	3.3V/5V	
3	SCL	P17	
4	SDA	P15	
5	RES	P33	
6	DC	P12	VCC or GND
7	CS	P13	GND

Operating Steps:

A. Connect the IPS module and the C51 MCU according to the above wiring instructions, and power on;

B. Select the C51 test program to be tested, as shown below:

(Please refer to the test program description document in the test package for the test program description)


project ▸ 1.54inch ▸ 1.54OLED_SSD1309 ▸ 1.54inch_OLED_SSD1309_SPI&IIC_Module_MSP154X_V1.0 ▸ 1-Demo ▸

新建文件夹

名称

Demo_IIC
Demo_SPI
picture
1.54inch_OLED_64x128_SSD1309_SPI&IIC_C51_Demo_Instructions_CN.pdf
1.54inch_OLED_64x128_SSD1309_SPI&IIC_C51_Demo_Instructions_EN.pdf

Test program
Test program effect picture
Test program Chinese and English documentation

C. Open the selected test program project, compile and download;

detailed description of the C51 test program compilation and download can be found in the following document:

http://www.lcdwiki.com/res/PublicFile/C51_Keil%26stc-isp_Use_Illustration_EN.pdf

D. If the OLED module displays characters and graphics normally, the program runs successfully;

5. MSP430 instructions

Wiring instructions:

See the interface description for pin assignments.

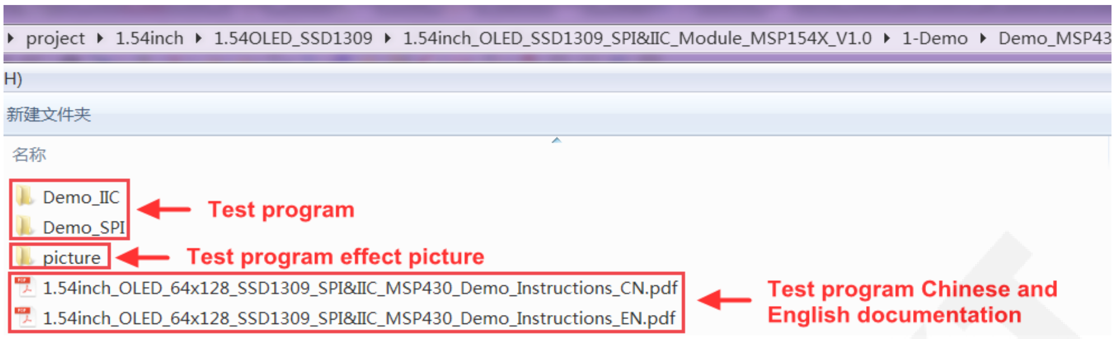
MSP430F149 microcontroller test program wiring instructions			
Number	Module Pin	Corresponding to MSP430 development board wiring pin	
		SPI	IIC
1	GND	GND	
2	VCC	3.3V/5V	
3	SCL	P33	
4	SDA	P31	
5	RES	P22	
6	DC	P21	VCC or GND
7	CS	P20	GND

Operating Steps:

A. Connect the IPS module and the MSP430 MCU according to the above wiring instructions, and power on;

B. Select the MSP430 test program to be tested, as shown below:
(Please refer to the test program description document in the test package for the test program description)

LCDWIKI



▸ project ▸ 1.54inch ▸ 1.54OLED_SSD1309 ▸ 1.54inch_OLED_SSD1309_SPI&IIC_Module_MSP154X_V1.0 ▸ 1-Demo ▸ Demo_MSP43

H)

新建文件夹

名称

Demo_IIC

Demo_SPI

picture

1.54inch_OLED_64x128_SSD1309_SPI&IIC_MSP430_Demo_Instructions_CN.pdf

1.54inch_OLED_64x128_SSD1309_SPI&IIC_MSP430_Demo_Instructions_EN.pdf

Test program
Test program effect picture
Test program Chinese and English documentation

C. Open the selected test program project, compile and download;

detailed description of the MSP430 test program compilation and download can be found in the following document:

http://www.lcdwiki.com/res/PublicFile/IAR_IDE%26MspFet_Use_Illustration_EN.pdf

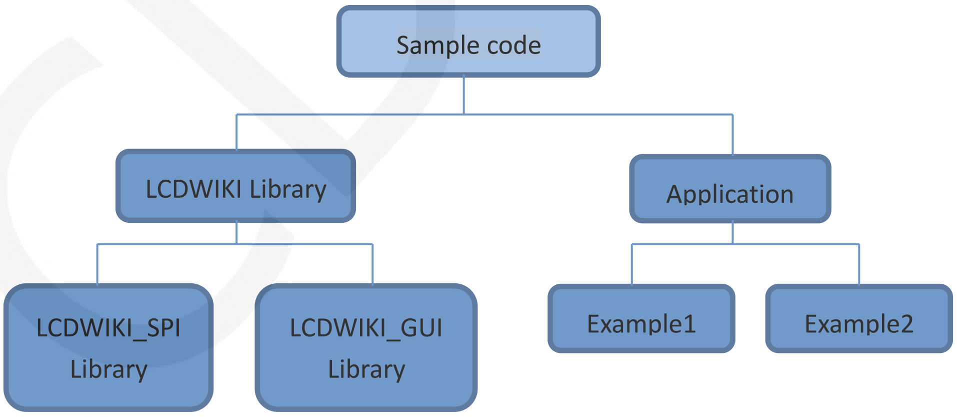
D. If the IPS module displays characters and graphics normally, the program runs successfully;

Software Description

1. Code Architecture

A. Arduino code architecture description

The code architecture is shown below



Arduino's test program code consists of two parts: the LCDWIKI library and application code.

The LCDWIKI library consists of two parts: the LCDWIKI_SPI library and the LCD_GUI library.

The application contains several test examples, each with different test content

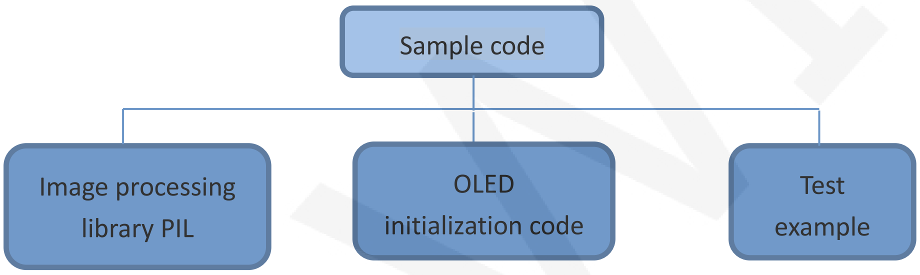
LCDWIKI_SPI is the underlying library, which is associated with hardware. It is mainly responsible for operating registers, including hardware module initialization, data and command transmission, pixel coordinates and color settings, and display mode configuration.

LCDWIKI_GUI is a middle-tier library, which is mainly responsible for drawing graphics and displaying characters using the API provided by the underlying library.

The application is to use the API provided by the LCDWIKI library to write some test examples to achieve some aspects of the test function.

B. RaspberryPi code architecture description

The python test program code architecture is shown below:



The python test program consists of but part: PIL image processing library, OLED initialization code, test sample code

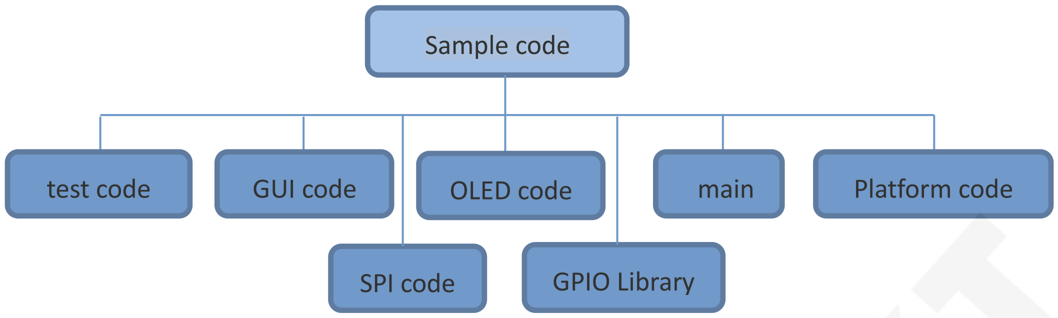
PIL image processing library is responsible for image drawing, character and text display operations, etc.

OLDE initialization code is responsible for operating registers, including hardware module initialization, data and command transfer, pixel coordinates and color settings, display mode configuration, etc.

The test example is to use the API provided by the above two parts of the code to implement some test functions.

The bcm2835 and wiringPi test program code architecture is as follows:

LCDWIKI



The Demo API code for the main program runtime is included in the test code;

OLED initialization and related operations are included in the OLED code;

Drawing points, lines, graphics, and Chinese and English character display related operations are included in the GUI code;

The GPIO library provides GPIO operations;

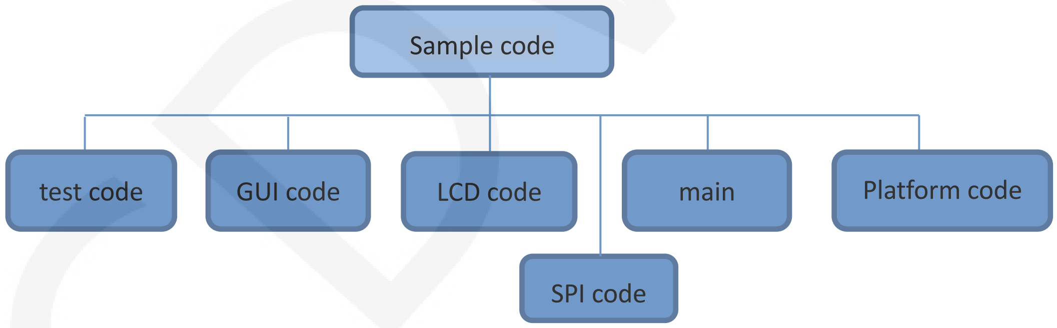
The main function implements the application to run;

Platform code varies by platform;

SPI initialization and configuration related operations are included in the SPI code;

C. C51, STM32 and MSP430 code architecture description

The code architecture is shown below:



The Demo API code for the main program runtime is included in the test code;

OLED initialization and related bin parallel port write data operations are included in the OLED code;

Drawing points, lines, graphics, and Chinese and English character display related operations are included in the GUI code;

The main function implements the application to run;

Platform code varies by platform;

SPI initialization and configuration related operations are included in the SPI code;

2. software SPI and hardware SPI description

The IPS module provides software SPI and hardware SPI sample code (except STC89C52RC, because it does not have hardware SPI function), the two sample code does not make any difference in the display content, but the following aspects are different:

A. display speed

The hardware SPI is significantly faster than the software SPI, which is determined by the hardware.

B. GPIO definition

The software SPI all control pins must be defined, any idle pin can be used, the hardware SPI data and clock signal pins are fixed (depending on the platform), other control pins should be defined by themselves, or any idle reference can be used. foot.

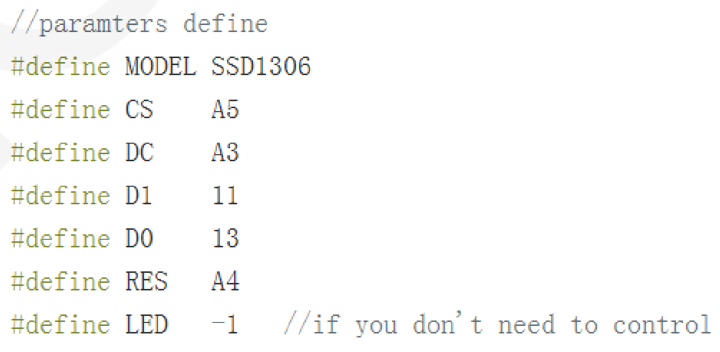
C. initialization

When the software SPI is initialized, only the GPIO for pin definition needs to be initialized (not required by the C51 platform). When the hardware SPI is initialized, the relevant control registers and data registers need to be initialized.

3. GPIO definition description

A. Arduino test program GPIO definition description

The Arduino test program GPIO definitions are placed in the application examples, and each application example can define GPIO. As shown in the figure below (take UNO MCU 4-wire software SPI test program as an example):



```
//paramters define
#define MODEL SSD1306
#define CS    A5
#define DC    A3
#define D1    11
#define D0    13
#define RES   A4
#define LED   -1   //if you don't need to control
```

LCDWIKI

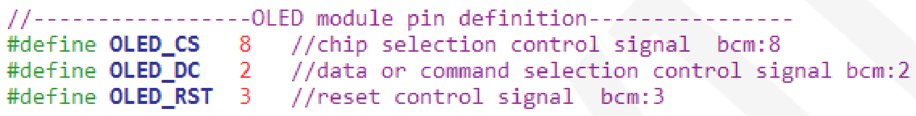
If using the software SPI, all pin definitions can be modified to any other free GPIO.

If hardware SPI is used, D0 and D1 cannot be modified and do not need to be defined. Other GPIOs can be modified.

If a 3-wire SPI is used, the DC does not need to be defined.

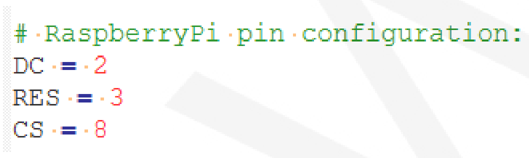
B. RaspberryPi test program GPIO definition description

The RaspberryPi test program uses hardware SPI, so only three GPIO ports need to be defined. The bcm2835 and WiringPi test programs place the GPIO definition in the oled.h file, as shown in the following figure (take the 4-wire SPI test program as an example):



```
//-----------------OLED module pin definition----------------
#define OLED_CS   8   //chip selection control signal  bcm:8
#define OLED_DC   2   //data or command selection control signal bcm:2
#define OLED_RST  3   //reset control signal  bcm:3
```

The Python test program places the GPIO definition in each test example, as shown in the following figure (take the 4-wire SPI test program as an example):



```
# RaspberryPi pin configuration:
DC = 2
RES = 3
CS = 8
```

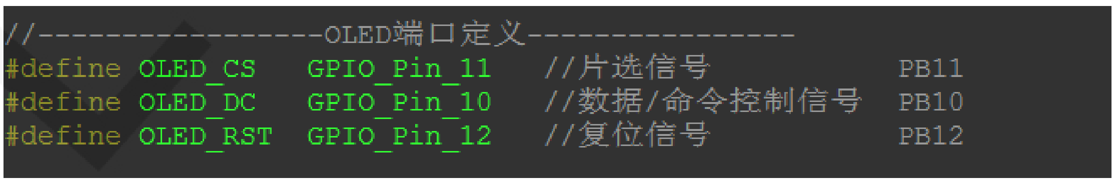
These three GPIOs can be modified according to the corresponding GPIO library code.

If a 3-wire SPI is used, the OLED_DC or DC does not need to be defined.

C. STM32 test program GPIO definition description

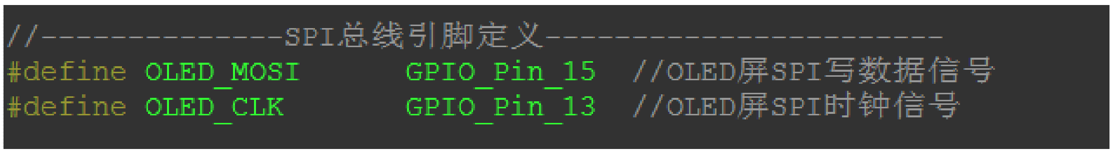
The STM32 test program GPIO definition is divided into two parts: control GPIO definition and SPI GPIO definition

The control GPIO definition is placed in oled.h, and the SPI GPIO definition is placed in spi.h, as shown in the following figure (take the STM32F103RCT6 software 4-wire SPI test program as an example):



```
//------------------OLED端口定义----------------
#define OLED_CS    GPIO_Pin_11   //片选信号          PB11
#define OLED_DC    GPIO_Pin_10   //数据/命令控制信号  PB10
#define OLED_RST   GPIO_Pin_12   //复位信号          PB12
```

LCDWIKI



```
//--------------SPI总线引脚定义-----------------------
#define OLED_MOSI       GPIO_Pin_15   //OLED屏SPI写数据信号
#define OLED_CLK        GPIO_Pin_13   //OLED屏SPI时钟信号
```

If using the software SPI, all pin definitions can be modified to any other free GPIO.

If hardware SPI is used, OLED_MOSI and OLED_CLK cannot be modified and do not need to be defined. Other GPIOs Can be modified.

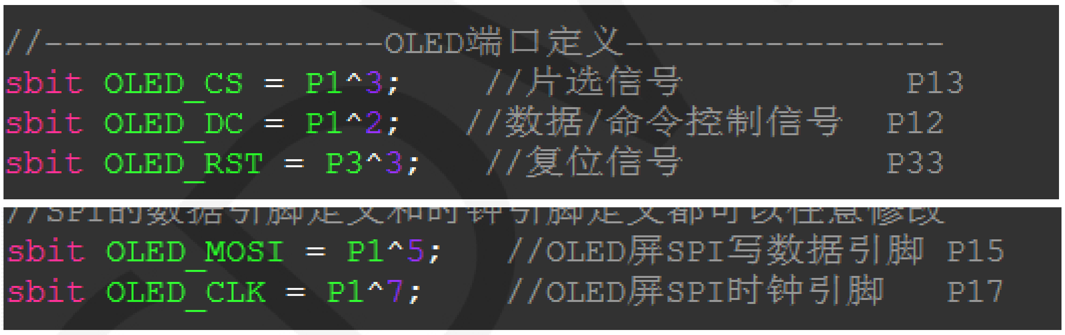
If you use a 3-wire SPI, OLED_DC does not need to be defined.

After modifying the GPIO definition, you need to initialize the GPIO to the OLED_Init_GPIO function in the oled.c file.

D. C51 test program GPIO definition description

The C51 test program GPIO definition is divided into two parts: control GPIO definition and SPI GPIO definition

The control GPIO definition is placed in oled.h, and the SPI GPIO definition is placed in spi.h, as shown in the following figure (take the STC12C5A60S2 software 4-wire SPI test program as an example):



```
//-----------------OLED端口定义----------------
sbit OLED_CS = P1^3;     //片选信号            P13
sbit OLED_DC = P1^2;    //数据/命令控制信号   P12
sbit OLED_RST = P3^3;    //复位信号            P33
//SPI的数据引脚定义和时钟引脚定义都可以任意修改
sbit OLED_MOSI = P1^5;    //OLED屏SPI写数据引脚  P15
sbit OLED_CLK = P1^7;     //OLED屏SPI时钟引脚    P17
```

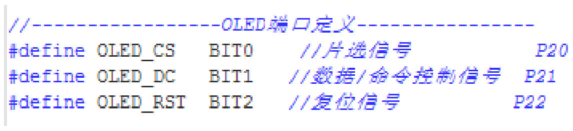
If using the software SPI, all pin definitions can be modified to any other free GPIO.

If hardware SPI is used, OLED_MOSI and OLED_CLK cannot be modified and do not need to be defined. Other GPIOs can be modified. (Only STC12C5A60S2 microcontroller has hardware SPI function)

If you use a 3-wire SPI, OLED_DC does not need to be defined.

E. MSP430 test program GPIO definition description

MSP430's LCD non-SPI GPIO definition is placed in lcd.h, as shown below (take MSP430F149 software 4-wire SPI test program as an example):

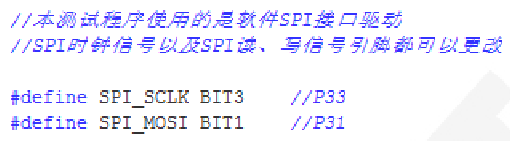


```
//-----------------OLED端口定义----------------
#define OLED_CS   BIT0    //片选信号          P20
#define OLED_DC   BIT1   //数据/命令控制信号  P21
#define OLED_RST  BIT2   //复位信号          P22
```

All pin definitions can be modified and can be defined as any other free GPIO.

If you use a 3-wire SPI, OLED_DC does not need to be defined.

The GPIO definition of the MSP430 LCD SPI is placed in spi.h, as shown in the following figure (take the MSP430F149 software 4-wire SPI test program as an example):



```
//本测试程序使用的是软件SPI接口驱动
//SPI时钟信号以及SPI读、写信号引脚都可以更改

#define SPI_SCLK BIT3    //P33
#define SPI_MOSI BIT1    //P31
```

If using the software SPI, all pin definitions can be modified and can be defined as any other free GPIO.

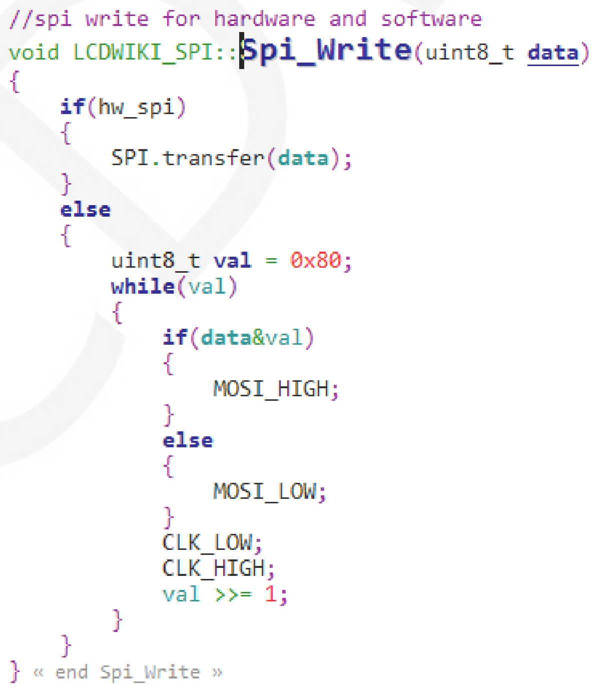
If you use hardware SPI, these pins do not need to be defined.

4. SPI communication code implementation

A. Arduino test program SPI communication code implementation

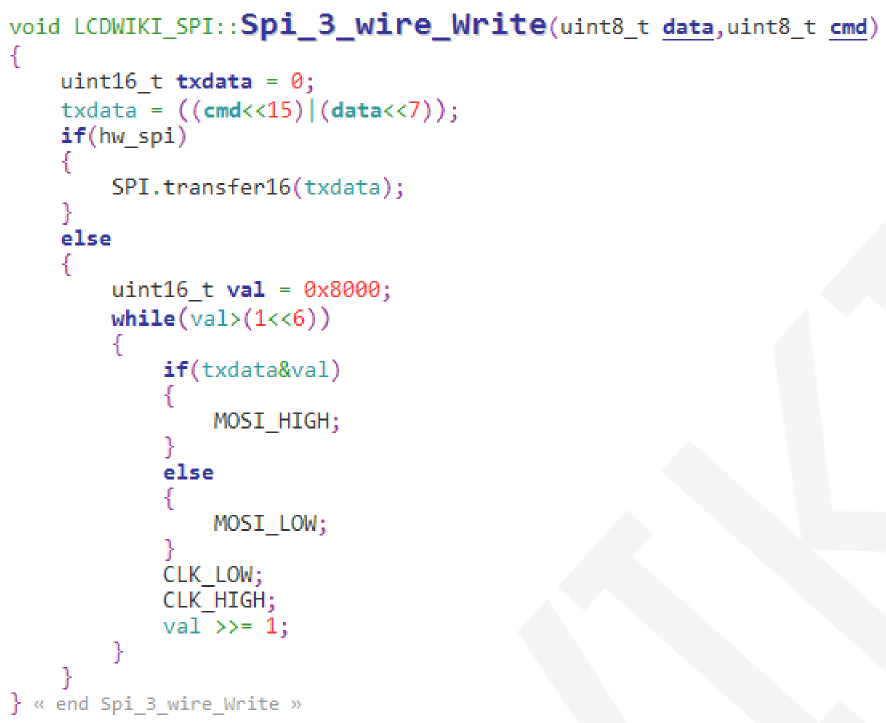
The SPI communication code is implemented in the LCDWIKI_SPI library.

The 4-wire software and hardware SPI code implementation is shown below:



```
//spi write for hardware and software
void LCDWIKI_SPI::Spi_Write(uint8_t data)
{
    if(hw_spi)
    {
        SPI.transfer(data);
    }
    else
    {
        uint8_t val = 0x80;
        while(val)
        {
            if(data&val)
            {
                MOSI_HIGH;
            }
            else
            {
                MOSI_LOW;
            }
            CLK_LOW;
            CLK_HIGH;
            val >>= 1;
        }
    }
} « end Spi_Write »
```

The 3-wire software and hardware SPI code implementation is shown below:



```
void LCDWIKI_SPI::Spi_3_wire_Write(uint8_t data,uint8_t cmd)
{
    uint16_t txdata = 0;
    txdata = ((cmd<<15)|(data<<7));
    if(hw_spi)
    {
        SPI.transfer16(txdata);
    }
    else
    {
        uint16_t val = 0x8000;
        while(val>(1<<6))
        {
            if(txdata&val)
            {
                MOSI_HIGH;
            }
            else
            {
                MOSI_LOW;
            }
            CLK_LOW;
            CLK_HIGH;
            val >>= 1;
        }
    }
} « end Spi_3_wire_Write »
```

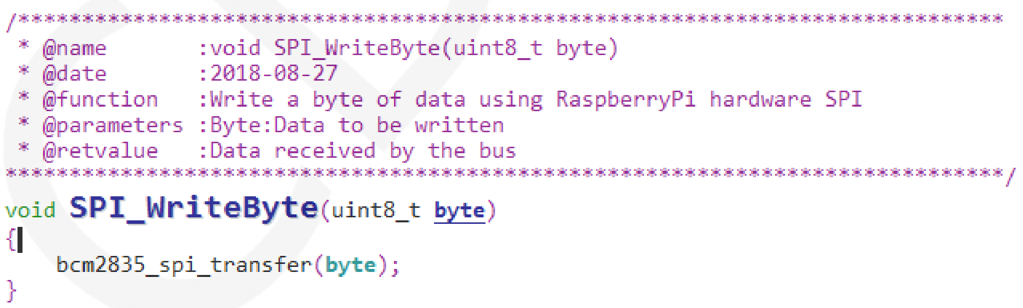
It is through the flag bit to decide whether to use software SPI or hardware SPI.

B. RaspberryPi test program SPI communication code implementation

The SPI communication code for the bcm2835 and wiringPi test programs is implemented in spi.c.

The SPI communication code for the python test program is implemented in oled.py.

The bcm2835 test program 4-wire hardware SPI code is implemented as shown below:



```
/*****************************************************************************
 * @name       :void SPI_WriteByte(uint8_t byte)
 * @date       :2018-08-27
 * @function   :Write a byte of data using RaspberryPi hardware SPI
 * @parameters :Byte:Data to be written
 * @retvalue   :Data received by the bus
******************************************************************************/
void SPI_WriteByte(uint8_t byte)
{
    bcm2835_spi_transfer(byte);
}
```

The bcm2835 test program 3-wire hardware SPI code is implemented as shown below:

LCDWIKI

```
/*****************************************************************************
 * @name       :void SPI_WriteByte(uint8_t byte, uint8_t cmd)
 * @date       :2018-08-27
 * @function   :Write a byte of data using RaspberryPi hardware SPI
 * @parameters :Byte:Data to be written
                cmd:0-command
                    1-data
 * @retvalue   :Data received by the bus
******************************************************************************/
void SPI_WriteByte(uint8_t byte, uint8_t cmd)
{
    uint16_t data=0;
    char txbuf[2]={0};
    data=((cmd<<15)|(byte<<7));
    txbuf[0]=(char)(data>>8);
    txbuf[1]=(char)(data&0xFF);
        bcm2835_spi_transfern(txbuf,2);
}
```

The wiringPi test program 4-wire hardware SPI code is implemented as shown below:

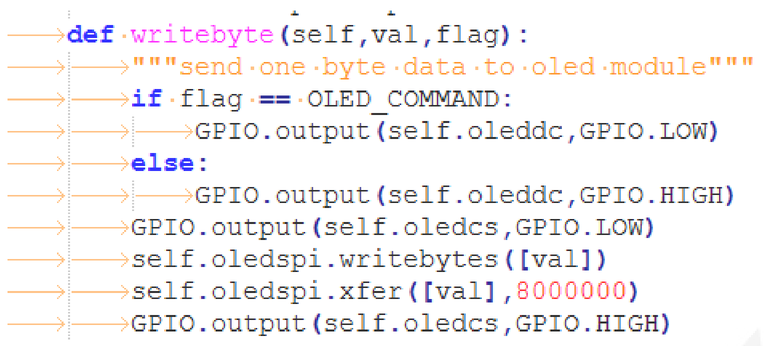
```
/*****************************************************************************
 * @name       :void SPI_WriteByte(uint8_t byte)
 * @date       :2018-08-27
 * @function   :Write a byte of data using RaspberryPi hardware SPI
 * @parameters :Byte:Data to be written
 * @retvalue   :Data received by the bus
******************************************************************************/
void SPI_WriteByte(uint8_t byte)
{
    wiringPiSPIDataRW(CHANNEL,&byte,1);
}
```

The wiringPi test program 3-wire hardware SPI code is implemented as shown below:

```
/*****************************************************************************
 * @name       :void SPI_WriteByte(uint8_t byte, uint8_t cmd)
 * @date       :2018-08-27
 * @function   :Write a byte of data using RaspberryPi hardware SPI
 * @parameters :Byte:Data to be written
                cmd:0-command
                    1-data
 * @retvalue   :Data received by the bus
******************************************************************************/
void SPI_WriteByte(uint8_t byte, uint8_t cmd)
{
    uint16_t data;
    unsigned char txbuf[2]={0};
    data=((cmd<<15)|(byte<<7));
    txbuf[0]=(unsigned char)(data>>8);
    txbuf[1]=(unsigned char)(data&0xFF);
    wiringPiSPIDataRW(CHANNEL,txbuf,2);
}
```

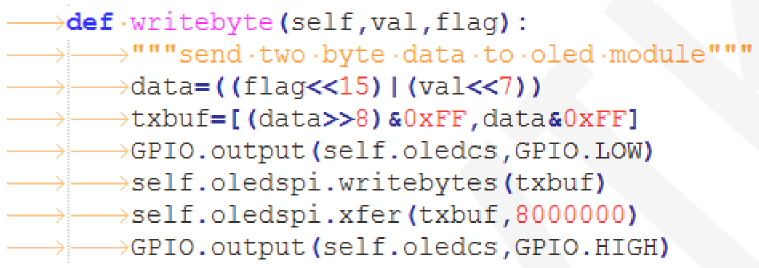
The python test program 4-wire hardware SPI code is implemented as shown below:

LCDWIKI



```
    def writebyte(self,val,flag):
        """send one byte data to oled module"""
        if flag == OLED_COMMAND:
            GPIO.output(self.oleddc,GPIO.LOW)
        else:
            GPIO.output(self.oleddc,GPIO.HIGH)
        GPIO.output(self.oledcs,GPIO.LOW)
        self.oledspi.writebytes([val])
        self.oledspi.xfer([val],8000000)
        GPIO.output(self.oledcs,GPIO.HIGH)
```

The python test program 3-wire hardware SPI code is implemented as shown below:



```
    def writebyte(self,val,flag):
        """send two byte data to oled module"""
        data=((flag<<15)|(val<<7))
        txbuf=[(data>>8)&0xFF,data&0xFF]
        GPIO.output(self.oledcs,GPIO.LOW)
        self.oledspi.writebytes(txbuf)
        self.oledspi.xfer(txbuf,8000000)
        GPIO.output(self.oledcs,GPIO.HIGH)
```

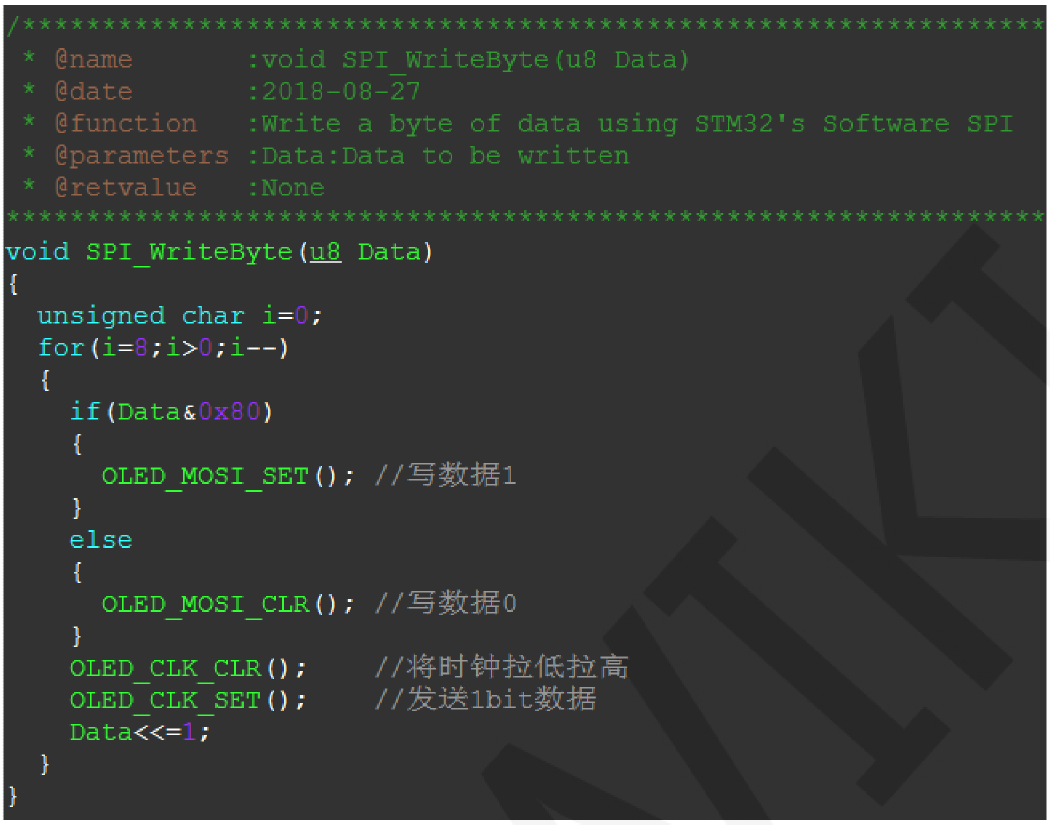
C. STM32 test program SPI communication code implementation

The SPI communication code is implemented in spi.c. (take STM32F103RCT6 test program as an example)

The 4-wire software and hardware SPI communication code implementation is as follows:

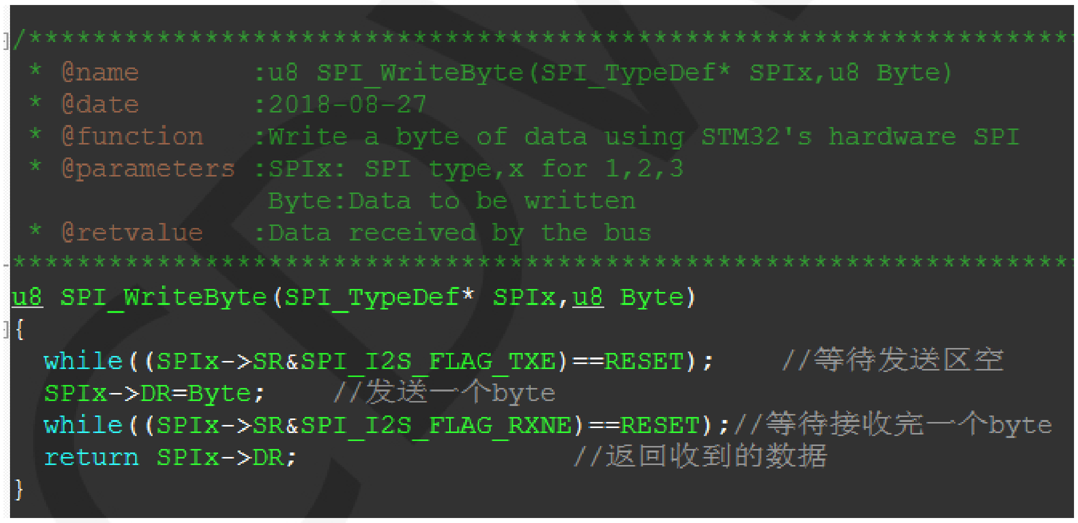
Software SPI:

LCDWIKI



```
/*****************************************************************
 * @name       :void SPI_WriteByte(u8 Data)
 * @date       :2018-08-27
 * @function   :Write a byte of data using STM32's Software SPI
 * @parameters :Data:Data to be written
 * @retvalue   :None
******************************************************************/
void SPI_WriteByte(u8 Data)
{
  unsigned char i=0;
  for(i=8;i>0;i--)
  {
    if(Data&0x80)
    {
      OLED_MOSI_SET(); //写数据1
    }
    else
    {
      OLED_MOSI_CLR(); //写数据0
    }
    OLED_CLK_CLR();     //将时钟拉低拉高
    OLED_CLK_SET();     //发送1bit数据
    Data<<=1;
  }
}
```

Hardware SPI:

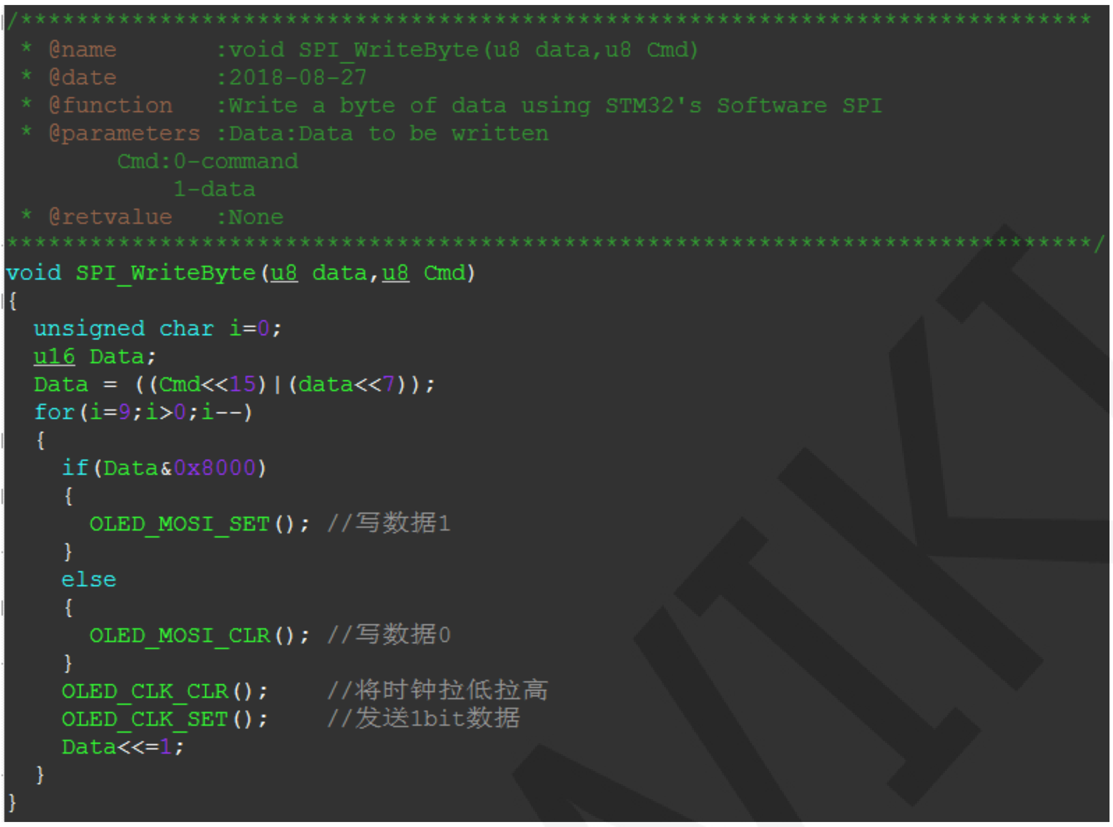


```
/*****************************************************************
 * @name       :u8 SPI_WriteByte(SPI_TypeDef* SPIx,u8 Byte)
 * @date       :2018-08-27
 * @function   :Write a byte of data using STM32's hardware SPI
 * @parameters :SPIx: SPI type,x for 1,2,3
                Byte:Data to be written
 * @retvalue   :Data received by the bus
******************************************************************/
u8 SPI_WriteByte(SPI_TypeDef* SPIx,u8 Byte)
{
  while((SPIx->SR&SPI_I2S_FLAG_TXE)==RESET);     //等待发送区空
  SPIx->DR=Byte;     //发送一个byte
  while((SPIx->SR&SPI_I2S_FLAG_RXNE)==RESET);//等待接收完一个byte
  return SPIx->DR;                 //返回收到的数据
}
```

The 3-wire software and hardware SPI communication code implementation is as follows:

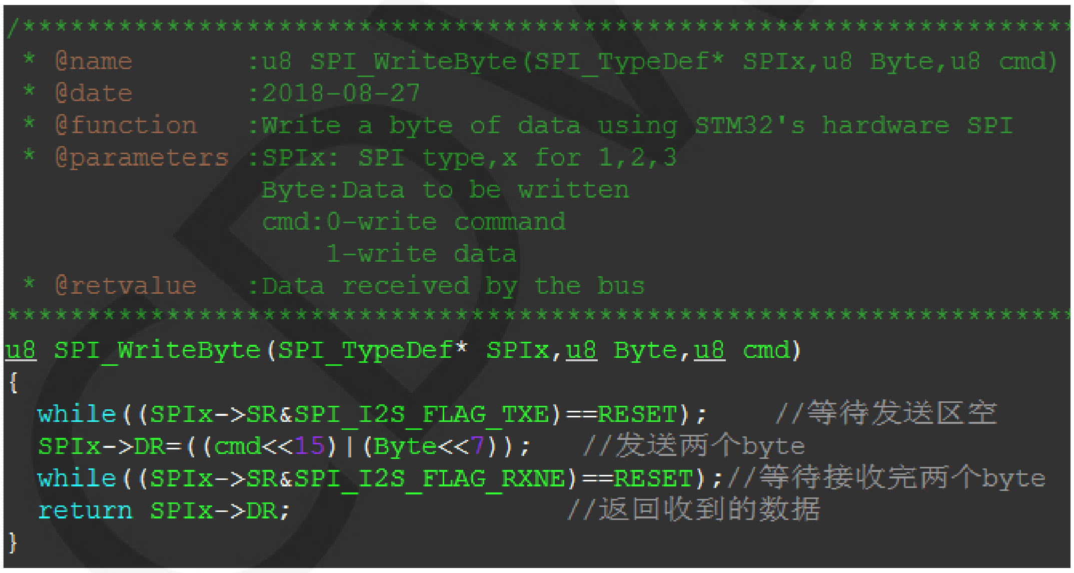
Software SPI:

LCDWIKI



```
/**************************************************************************
 * @name        :void SPI_WriteByte(u8 data,u8 Cmd)
 * @date        :2018-08-27
 * @function    :Write a byte of data using STM32's Software SPI
 * @parameters :Data:Data to be written
       Cmd:0-command
           1-data
 * @retvalue    :None
**************************************************************************/
void SPI_WriteByte(u8 data,u8 Cmd)
{
  unsigned char i=0;
  u16 Data;
  Data = ((Cmd<<15)|(data<<7));
  for(i=9;i>0;i--)
  {
    if(Data&0x8000)
    {
      OLED_MOSI_SET(); //写数据1
    }
    else
    {
      OLED_MOSI_CLR(); //写数据0
    }
    OLED_CLK_CLR();     //将时钟拉低拉高
    OLED_CLK_SET();     //发送1bit数据
    Data<<=1;
  }
}
```

Hardware SPI:



```
/*************************************************************************
 * @name        :u8 SPI_WriteByte(SPI_TypeDef* SPIx,u8 Byte,u8 cmd)
 * @date        :2018-08-27
 * @function    :Write a byte of data using STM32's hardware SPI
 * @parameters :SPIx: SPI type,x for 1,2,3
                Byte:Data to be written
                cmd:0-write command
                    1-write data
 * @retvalue    :Data received by the bus
*************************************************************************
u8 SPI_WriteByte(SPI_TypeDef* SPIx,u8 Byte,u8 cmd)
{
  while((SPIx->SR&SPI_I2S_FLAG_TXE)==RESET);     //等待发送区空
  SPIx->DR=((cmd<<15)|(Byte<<7));    //发送两个byte
  while((SPIx->SR&SPI_I2S_FLAG_RXNE)==RESET);//等待接收完两个byte
  return SPIx->DR;                  //返回收到的数据
}
```

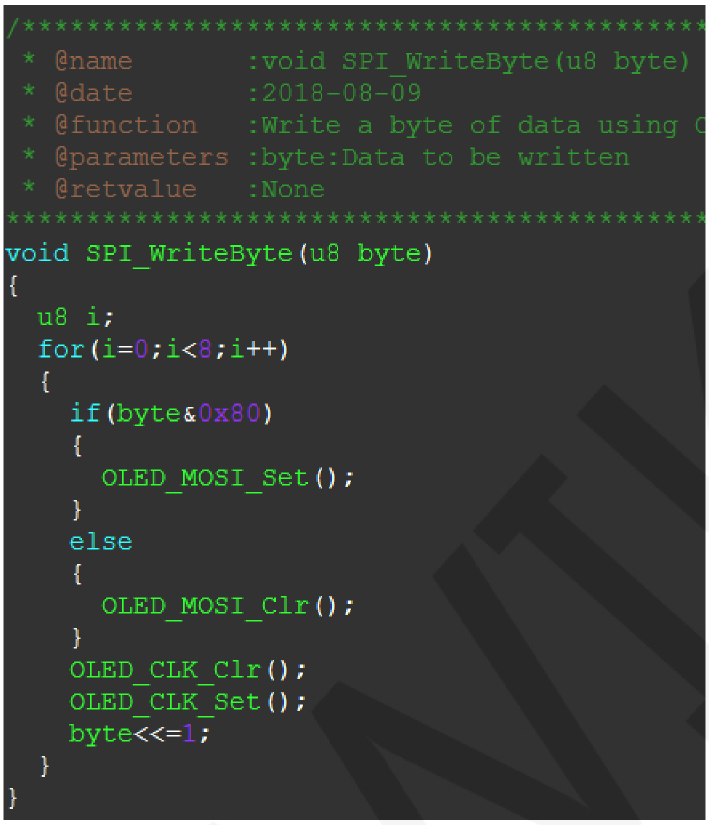
D. C51 test program SPI communication code implementation

The SPI communication code is implemented in spi.c. (taking the STC12C5A60S2 test program as an example)

The 4-wire software and hardware SPI communication code implementation is as follows:

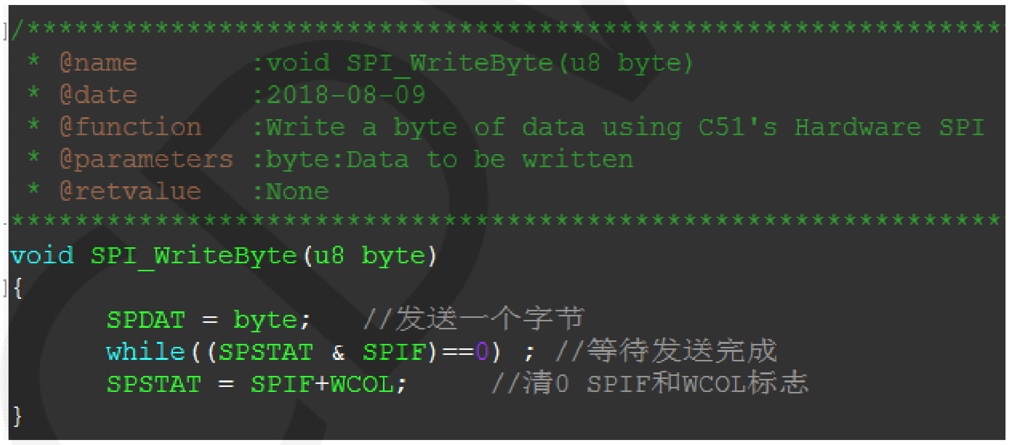
LCDWIKI

Software SPI:



```
/*********************************************
 * @name        :void SPI_WriteByte(u8 byte)
 * @date        :2018-08-09
 * @function    :Write a byte of data using C
 * @parameters  :byte:Data to be written
 * @retvalue    :None
*********************************************
void SPI_WriteByte(u8 byte)
{
  u8 i;
  for(i=0;i<8;i++)
  {
    if(byte&0x80)
    {
      OLED_MOSI_Set();
    }
    else
    {
      OLED_MOSI_Clr();
    }
    OLED_CLK_Clr();
    OLED_CLK_Set();
    byte<<=1;
  }
}
```

Hardware SPI:

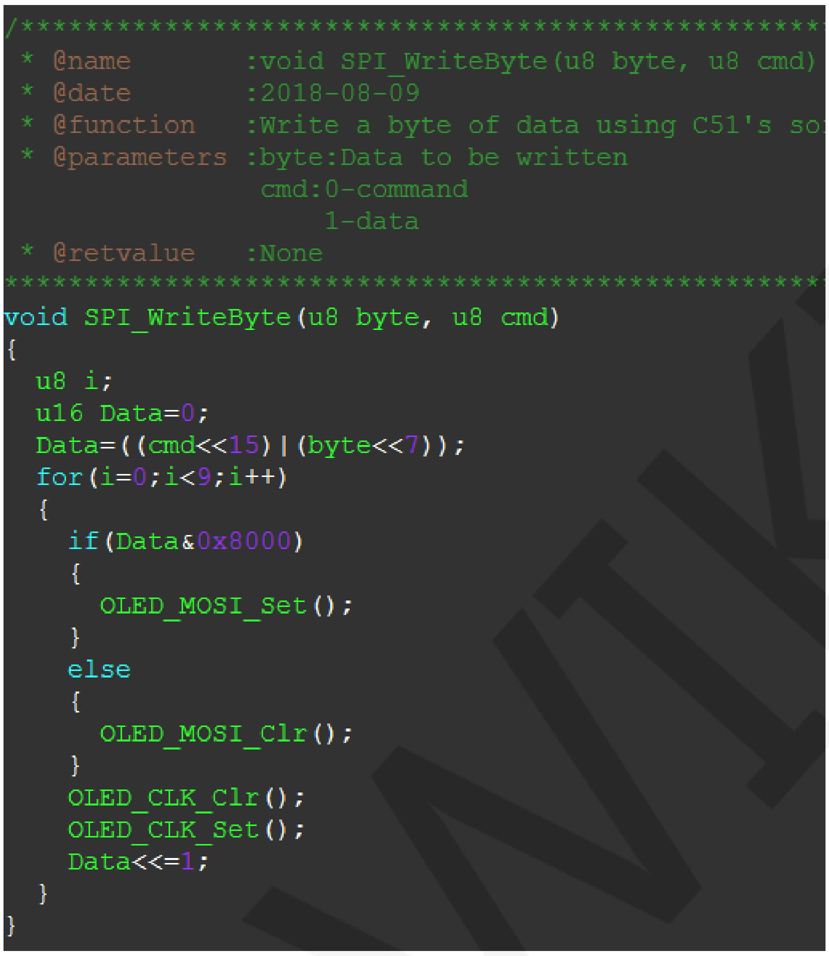


```
/***************************************************************
 * @name        :void SPI_WriteByte(u8 byte)
 * @date        :2018-08-09
 * @function    :Write a byte of data using C51's Hardware SPI
 * @parameters  :byte:Data to be written
 * @retvalue    :None
****************************************************************
void SPI_WriteByte(u8 byte)
{
      SPDAT = byte;    //发送一个字节
      while((SPSTAT & SPIF)==0) ; //等待发送完成
      SPSTAT = SPIF+WCOL;      //清0 SPIF和WCOL标志
}
```

The 3-wire software and hardware SPI communication code implementation is as follows:

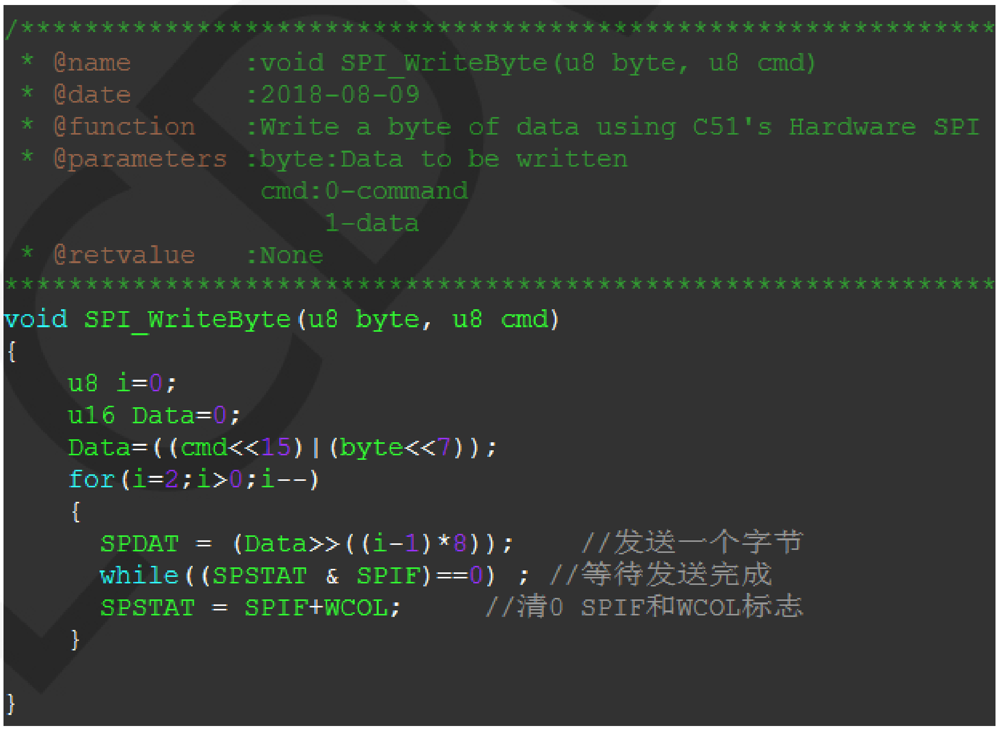
Software SPI:

LCDWIKI



```
/*************************************************
 * @name        :void SPI_WriteByte(u8 byte, u8 cmd)
 * @date        :2018-08-09
 * @function    :Write a byte of data using C51's so
 * @parameters  :byte:Data to be written
                 cmd:0-command
                     1-data
 * @retvalue    :None
**************************************************
void SPI_WriteByte(u8 byte, u8 cmd)
{
  u8 i;
  u16 Data=0;
  Data=((cmd<<15)|(byte<<7));
  for(i=0;i<9;i++)
  {
    if(Data&0x8000)
    {
      OLED_MOSI_Set();
    }
    else
    {
      OLED_MOSI_Clr();
    }
    OLED_CLK_Clr();
    OLED_CLK_Set();
    Data<<=1;
  }
}
```

Hardware SPI:



```
/**************************************************************
 * @name        :void SPI_WriteByte(u8 byte, u8 cmd)
 * @date        :2018-08-09
 * @function    :Write a byte of data using C51's Hardware SPI
 * @parameters  :byte:Data to be written
                 cmd:0-command
                     1-data
 * @retvalue    :None
***************************************************************
void SPI_WriteByte(u8 byte, u8 cmd)
{
    u8 i=0;
    u16 Data=0;
    Data=((cmd<<15)|(byte<<7));
    for(i=2;i>0;i--)
    {
      SPDAT = (Data>>((i-1)*8));    //发送一个字节
      while((SPSTAT & SPIF)==0) ; //等待发送完成
      SPSTAT = SPIF+WCOL;      //清0 SPIF和WCOL标志
    }

}
```

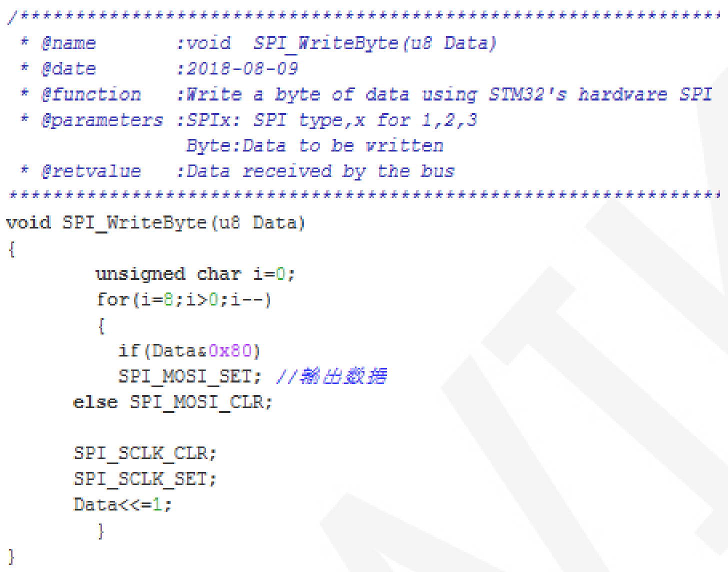
E. MSP430 test program SPI communication code implementation

LCDWIKI

The software SPI communication code is implemented in spi.c.

The 4-wire software and hardware SPI communication code implementation is as follows:

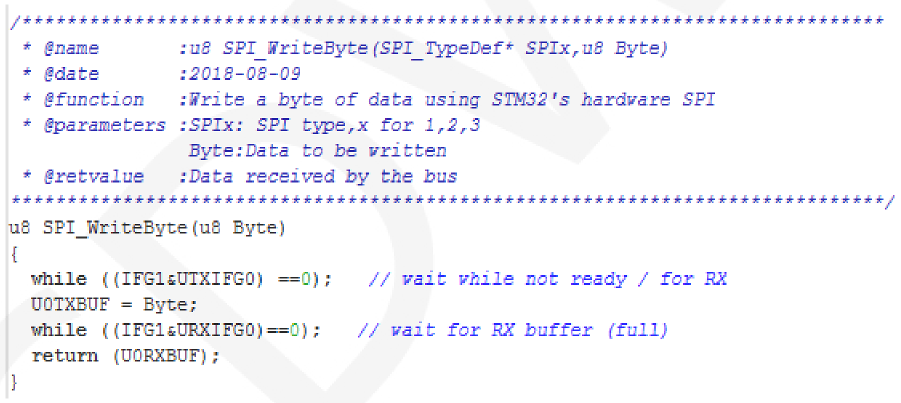
Software SPI:



```
/*****************************************************************
 * @name       :void  SPI_WriteByte(u8 Data)
 * @date       :2018-08-09
 * @function   :Write a byte of data using STM32's hardware SPI
 * @parameters :SPIx: SPI type,x for 1,2,3
                Byte:Data to be written
 * @retvalue   :Data received by the bus
******************************************************************/
void SPI_WriteByte(u8 Data)
{
        unsigned char i=0;
        for(i=8;i>0;i--)
        {
          if(Data&0x80)
          SPI_MOSI_SET; //输出数据
      else SPI_MOSI_CLR;

      SPI_SCLK_CLR;
      SPI_SCLK_SET;
      Data<<=1;
        }
}
```

Hardware SPI:



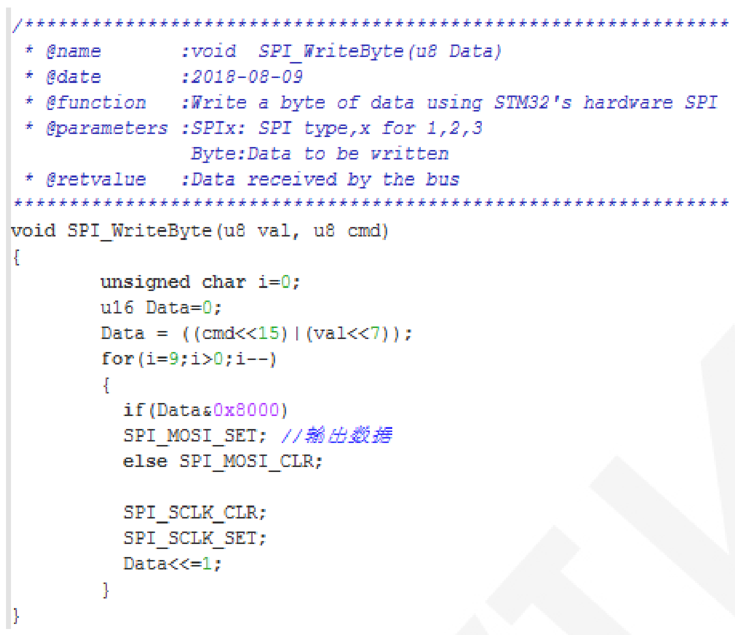
```
/******************************************************************************
 * @name       :u8 SPI_WriteByte(SPI_TypeDef* SPIx,u8 Byte)
 * @date       :2018-08-09
 * @function   :Write a byte of data using STM32's hardware SPI
 * @parameters :SPIx: SPI type,x for 1,2,3
                Byte:Data to be written
 * @retvalue   :Data received by the bus
******************************************************************************/
u8 SPI_WriteByte(u8 Byte)
{
  while ((IFG1&UTXIFG0) ==0);   // wait while not ready / for RX
  U0TXBUF = Byte;
  while ((IFG1&URXIFG0)==0);   // wait for RX buffer (full)
  return (U0RXBUF);
}
```

The software SPI communication code is implemented in spi.c.

The 3-wire software and hardware SPI communication code implementation is as follows:

Software SPI:

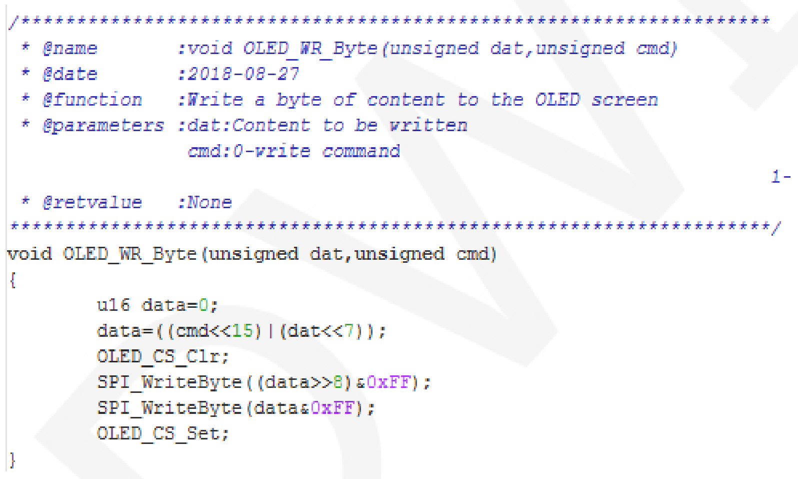
LCDWIKI



```
/******************************************************************
 * @name       :void  SPI_WriteByte(u8 Data)
 * @date       :2018-08-09
 * @function   :Write a byte of data using STM32's hardware SPI
 * @parameters :SPIx: SPI type,x for 1,2,3
                Byte:Data to be written
 * @retvalue   :Data received by the bus
******************************************************************
void SPI_WriteByte(u8 val, u8 cmd)
{
        unsigned char i=0;
        u16 Data=0;
        Data = ((cmd<<15)|(val<<7));
        for(i=9;i>0;i--)
        {
          if(Data&0x8000)
          SPI_MOSI_SET; //输出数据
          else SPI_MOSI_CLR;

          SPI_SCLK_CLR;
          SPI_SCLK_SET;
          Data<<=1;
        }
}
```

Hardware SPI:



```
/*******************************************************************
 * @name       :void OLED_WR_Byte(unsigned dat,unsigned cmd)
 * @date       :2018-08-27
 * @function   :Write a byte of content to the OLED screen
 * @parameters :dat:Content to be written
                cmd:0-write command
                                                               1-
 * @retvalue   :None
********************************************************************/
void OLED_WR_Byte(unsigned dat,unsigned cmd)
{
        u16 data=0;
        data=((cmd<<15)|(dat<<7));
        OLED_CS_Clr;
        SPI_WriteByte((data>>8)&0xFF);
        SPI_WriteByte(data&0xFF);
        OLED_CS_Set;
}
```

Common software

This set of test examples needs to display Chinese and English, symbols and pictures, so PCtoLCD2002 modulo software is used. Here, the setting of the modulo software is explained only for the test program.

The **PCtoLCD2002** modulo software settings are as follows:

Dot matrix format select Dark code

the modulo mode select the progressive mode(C51 and MSP430 test programs need to choose determinant)

Take the model to choose the direction (high position first)(C51 and MSP430 test

LCDWIKI

procedures need to choose reverse (low position first))

Output number system selects hexadecimal number

Custom format selection C51 format

The specific setting method is as follows:

http://www.lcdwiki.com/Chinese_and_English_display_modulo_settings

LCDWIKI