1.44inch LCD HAT USER MANUAL

OVERVIEW

This product is 1.44-inch resistive screen module with resolution 128 x 128. It has internal controller and uses SPI communication interface. It has already basic functions: setting the point size, the line thickness, drawing circle, rectangle, and displaying English characters.

FEATURES

Display type:	TFT
Interface:	SPI
Driver:	ST7735S
Colors:	256K
Resolution:	128 x 128 (Pixel)
Product size:	65*30.2 (mm)
Display size:	25.5*26.5 （mm）
Pixel size:	0.129(W)*0.219(H) （MM）
Operating temperature:	-30°C ~ 85°C

INTERFACE

Pins	Description
3V3	3.3V power
GND	ground
DIN	SPI data input
CLK	SPI clock
CS	chip select
DC	data/command
RST	reset
BL	背光

PROGRAM ANALYSIS

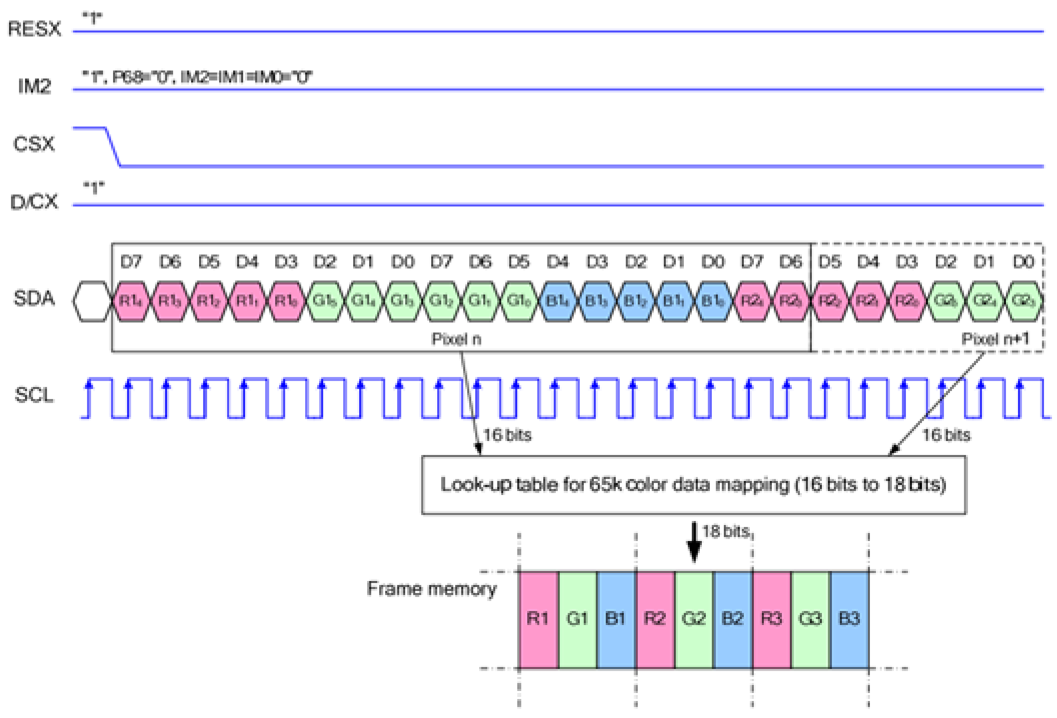
1. Working principles:

ST7735S is 132 x 162 pixel LCD panel, but the product is 128 x 128 pixel LCD display. In the display there are two processes: the horizontal direction scanning –use the 2nd pixel s the first pixel for display. So, you can see that positions of pixels in RAM correspond to their actual positions.

The LCD supports 12-bit, 16-bit and 18-bit per pixel input formats. They correspond to RGB444, RGB565 and RGB666 color formats. This routine uses the RGB565 color format, which is commonly used as well.

LCD uses 4-wired SPI communication interface, which can save a lot of GPIO ports and provides fast data transfer to LCD at the same time.

2. Communication protocol


RESX
"1"
IM2
"1", P68="0", IM2=IM1=IM0="0"
CSX
D/CX
"1"
SDA
SCL
Pixel n
Pixel n+1
16 bits
16 bits
Look-up table for 65k color data mapping (16 bits to 18 bits)
18 bits
Frame memory
R1 G1 B1 R2 G2 B2 R3 G3 B3

Note: there is a difference from traditional SPI, which is made because it is used for this display only. In this protocol some wires from slave to the host are hidden. The detailed information can be seen in datasheet at page 58.

RESX: reset. Set it to 0 while powering the module. Usually set it to 1

IM2: data communication mode pin, which define usage of SPI

CSX: chip selection control pin. If CS=0 – the chip is selected

D/CX: data/command control pin, if DC=0 – command is written. And if DC=1 – data are written

SDA: transmitted RGB data

SCL: SPI clock

The SPI communication protocol of the data transmission uses control bits: clock phase (CPHA) and clock polarity (CPOL):

CPOL defines the level while synchronization clock is idle. If CPOL=0, then it is LOW.

CPHA defines at whish clock's tick the data transmission starts. CPHL=0 – at the first one, otherwise at the second one

This combination of two bits provides 4 modes of SPI data transmission. The commonly used is SPI0 mode, i.e. GPHL=0 and CPOL=0.

From the picture it is seen that data transmission in SPI0 mode starts at the second falling edge of SCLK.

DEMO

We provide Raspberry Pi program: BCM2835, WiringPi and python programs. It implements common graphical functions as drawing dot, line, rectangle, circle, setting their sizes and line width; filling arias, and displaying English characters of 5 common fonts and other display's functions.

Following instructions are offered for you convenience

1. Initialize SPI function of the Raspberry Pi

   sudo raspi-config

   Select: ***Advanced Options -> SPI -> yes***

   Activate SPI hardware driver

2. Installation of function libraries

   For detailed information about libraries installation for Raspberry Pi, you can refer to this page:

   https://www.waveshare.com/wiki/Libraries_Installation_for_RPi

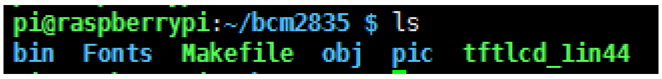
   It is detailed description of WiringPi, bcm2835 and python installation.

3. Usage

BCM2835 and WiringPi program should be only copied into correspondent directory of Raspberry Pi (by samba or directly copy to the SD). The following code example are copied directly to /home/pi of Raspbian.

3.1 BCM2835

Run *ls* command as you can see below:



```
pi@raspberrypi:~/bcm2835 $ ls
bin  Fonts  Makefile  obj  pic  tftlcd_1in44
```

There are: Folder ***bin*** contains generated ".o" project files: normally we don't need change it.

Folder ***Fonts*** contains 5 commonly used fonts.

Folder ***Obj*** contains object files: ***mian.c, OLED_Driver.c and OLED_Driver.h, DEV_Config.c and DEV_Config.c, OLED_GUI.c OLED_GUI.h***.

main.c: the main function. Important: even OLED_ScanDir function is intended to change the scan direction of the display, its usage doesn't make any effect, because this code came from Raspberry Pi's and uses just for compatibility.

DEV_Config.c: definitions of Raspberry Pi's pins and communication modes.

LCD_Driver.c: Driver code of LCD, it doesn't need any changes;

LCD_GUI.c: graphical primitives: dot, line, bitmap, text; normally you only need to change ***GUI_Show()*** function, it is display calling function;

LCD_BMP.c: Reading and analysis of bmp files and displaying them

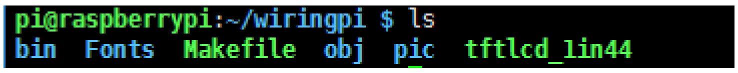
Makefile: this file contains compilation rules, if there are some changes in code please run ***make clear***, do changes if only you understand clearly dependences between files and generation of executable file, then run ***make*** again, it will compile whole project and generate executable file.

oled_1in44: this file is generate by command ***make***.

User needs only run ***sudo ./oled_1in44***

3.2 WiringPi

Input *ls* command, now you can see following:



```
pi@raspberrypi:~/wiringpi $ ls
bin  Fonts  Makefile  obj  pic  tftlcd_1in44
```

In cases of WiringPi and BCM2835 the file directory is the similar, there are two differences only:

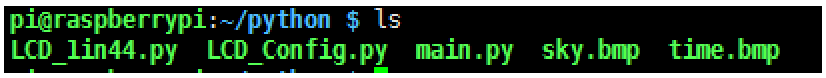
1. WiringPi operates by reading device files as in Linux OS, but bcm2835 operates Raspberry Pi chip's registers by its library functions. Thus, if you have used bcm2835 code before, wiringpi usage can be failed. This case you need to reboot the system to start successfully;

2. Due to the first difference their underlying configurations are not the same, in ***DEV_Config.c*** the usage of underlying WiringPi and WiringPi SPI interfaces are described.

The same is the program execution by command ***sudo ./oled_lin8***

3.3 Python

Input ***ls*** command, now you can see following:



```
pi@raspberrypi:~/python $ ls
LCD_1in44.py  LCD_Config.py  main.py  sky.bmp  time.bmp
```

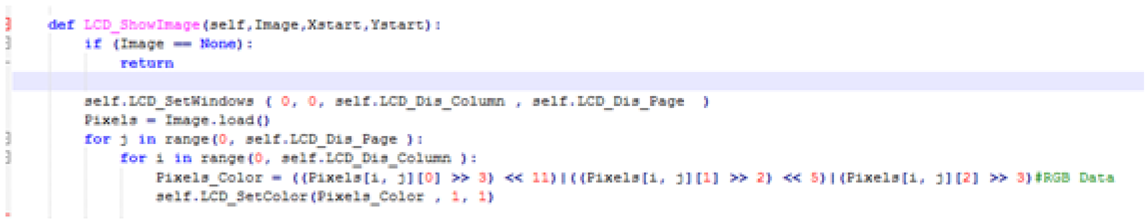
LCD_1in44.py: LCD driver

LCD_Config.py: configuration of hardware underlying interface

Execute ***sudo python main.py*** to run the program.

Note: some Raspbian OS hasn't library for images. Run ***sudo apt-get install python-imaging*** to install ***python-imaging*** library

Image is a graphic process library for python. Any sub-images are represented by an object Image. Thus, using operator ***new*** you can create blank image， and draw the picture according sizes of the object and LCD. After that the data are moved to LCD. The function ***Image.load()*** reads data in RGB888 pixel format and converts them to RGB565 pixel format. To display all of images on the screen the following code is used:



```
def LCD_ShowImage(self,Image,Xstart,Ystart):
    if (Image == None):
        return

    self.LCD_SetWindows ( 0, 0, self.LCD_Dis_Column , self.LCD_Dis_Page  )
    Pixels = Image.load()
    for j in range(0, self.LCD_Dis_Page ):
        for i in range(0, self.LCD_Dis_Column ):
            Pixels_Color = ((Pixels[i, j][0] >> 3) << 11)|((Pixels[i, j][1] >> 2) << 5)|(Pixels[i, j][2] >> 3)#RGB Data
            self.LCD_SetColor(Pixels_Color , 1, 1)
```

3.4 Auto-run option

Initialize autorun in Raspberry Pi by configuring code of /etc/rc.local file:

sudo vim /etc/rc.local

Before ***exit0*** add:

sudo python /home/pi/python/demo.py &

Important: /***home/pi/python/demo.py*** is the path of the code, , you can input command ***pwd*** to get this path. Character ***&*** is necessary at the end of command line, otherwise you probable need to reinstall the system (impossible exit the process by pressing ctrl+c, impossible to login Raspberry Pi's users).

COMPATIBLE CODE PORTING

Offered demo is the commonly used programs, which are able to be ported. They can be used with two screens and their difference is only in initialization of display sizes.

The usage method is defined by macros. To setup a certain configuration, select 1.44-inch screen initial configuration in LCD_Driver.h or in LCD.h:

```
//#define LCD_1IN44
#define LCD_1in8.
```

After saving run ***make clean*** to remove dependency files, and then run ***make*** again to generate executable file.