1.3inch LCD HAT User Manual

OVERVIEW

This is an IPS LCD display HAT for Raspberry Pi, 1.3inch diagonal, 240x240 pixels, with embedded controller, communicating via SPI interface. Its size is similar to Raspberry Pi Zero. With basic functions, it can display pictures, texts and figures.

Demo codes provided for Raspberry Pi which are based on BCM2835 library, WiringPi library and Python separately.

SPECIFICATIONS

Driver:	ST7789VM
Interface:	SPI
Display color:	RGB, 65K color
Resolution:	240x240
Backlight:	LED
Operating voltage:	3.3V
Display size:	23.4(H) x 23.4(V) mm
Outline size:	65 x 30.2mm

INTERFACES

PIN	Raspberry Pi	Description
3V3	3V3	3.3V power
GND	GND	Ground
CLK	P11/P_SCLK	SPI clock input
DIN	P10/P_MOSI	SPI data input
CS	P8/P_CE0	Chip select, Low active
DC	P25	Data/Command select
RST	P27	Reset
BL	P24	Back light
KEY1	P21	Button 1GPIO
KEY2	P20	Button 2GPIO
KEY3	P16	Button 3GPIO
Joystick Up	P6	Joystick up
Joystick Down	P19	Joystick down
Joystick Left	P5	Joystick left
Joystick Right	P26	Joystick right
Joystick Press	P13	Joystick press

Note: The GPIO num of Raspberry Pi is based on the code number of BCM2835 libraries.

HARDWARE DESCRIPTION

LCD CONTROLLER

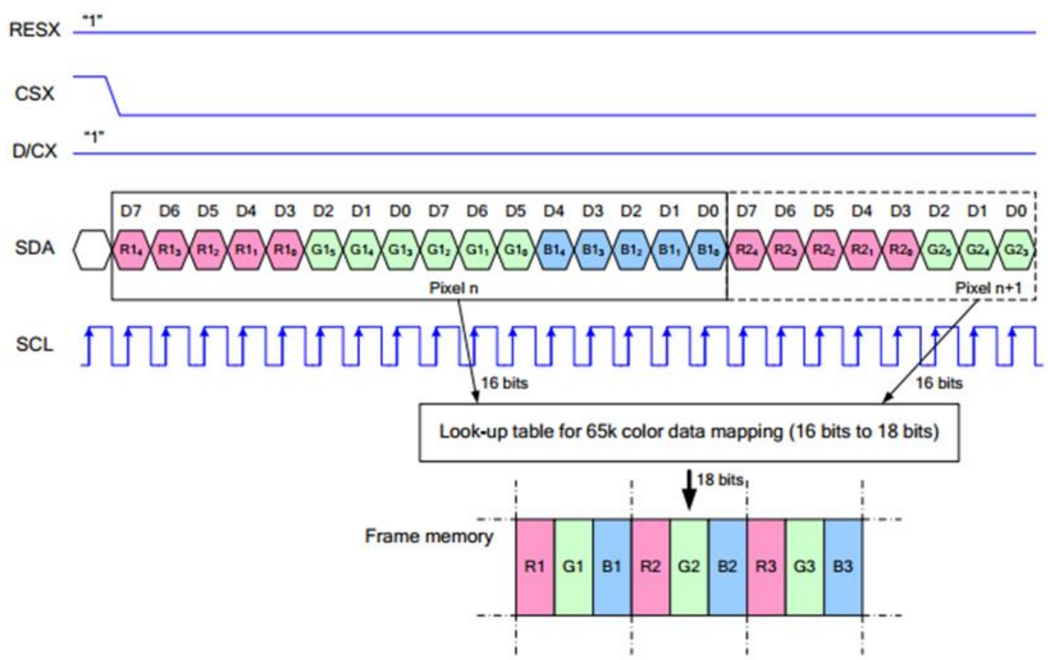
This LCD embedded ST7789VM, which is a controller for 24xRGBx320 resolution LCD. The resolution of this LCD is only 240(H)RGBx240(V) and supports initialize vertical display and horizontal display, thus the internal RAM of LCD is not full used.

Refer to the datasheet, ST7789VM controller supports 12bits (RGB444), 16bits (RGB565) and 18bit (RGB666) color formats. This LCD uses common RGB565 format.

Most of LCD controller can be configured to 8080, 3-wire SPI, 4-wires SPI interface and so on. This LCD uses 4-wire SPI interface to save GPIO and faster communicating.

COMMUNICATION PROTOCOL

We have known that this LCD use 4-wires SPI interface. The timing figure is provided on datasheet. The timing of RGB565 is as bellow:



Note: It is not like the tradition SPI protocol, it only uses MOSI to send data from master to slave for LCD display. For details please refer to Datasheet Page 105.

RESX: Reset, should be pull-down when power on, set to 1 other time.

CSX: Slave chip select. The chip is enabled only CS is set Low

D/CX: Data/Command selection; DC=0, write command; DC=1, write data

SDA: Data 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.

According to the figure above, data transmitting begins at the first falling edge, 8bit data are transmitted at one clock cycle. It is SPI0, Bitwise output, first high bits and low bits following.

USING WITH RASPBERRY

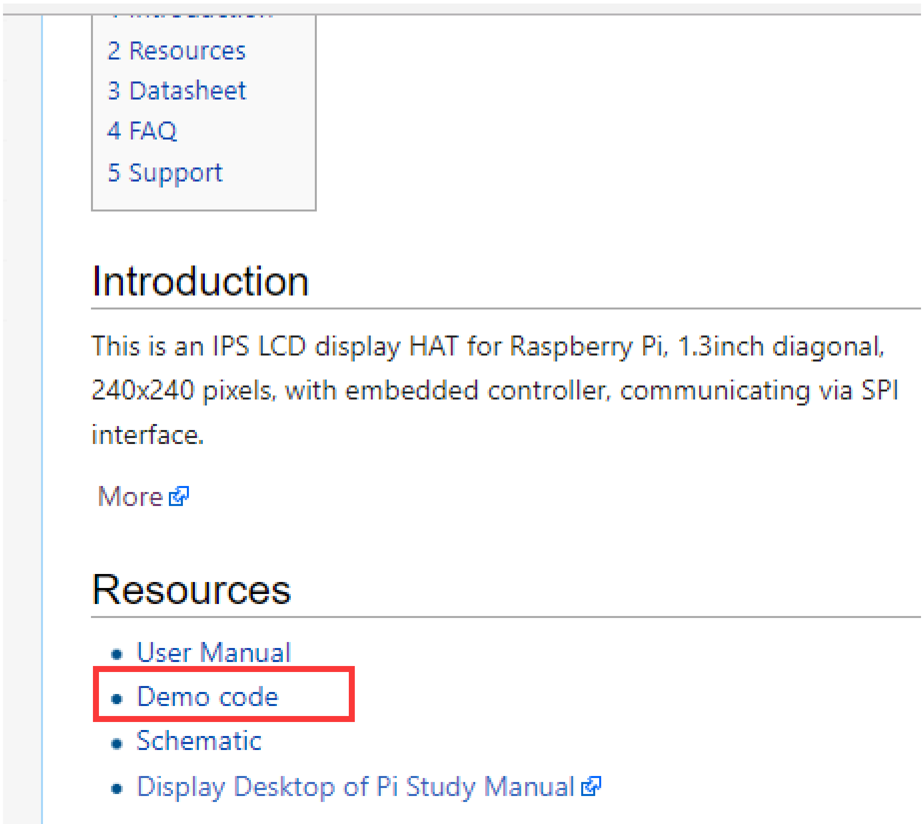
LIBRARIES INSTALLATION

To use the demo code properly, you should install support libraries first. The installation of libraries for wiringPi, bcm2835 and python is described on page: https://www.waveshare.com/wiki/Libraries_Installation_for_RPi

If you use python code, you need to install one more library:

```
sudo apt-get install python-imaging
```

Search 1.3inch LCD HAT on Waveshare Wiki, and download the demo code.



2 Resources
3 Datasheet
4 FAQ
5 Support

Introduction

This is an IPS LCD display HAT for Raspberry Pi, 1.3inch diagonal, 240x240 pixels, with embedded controller, communicating via SPI interface.

More

Resources

- User Manual
- Demo code
- Schematic
- Display Desktop of Pi Study Manual

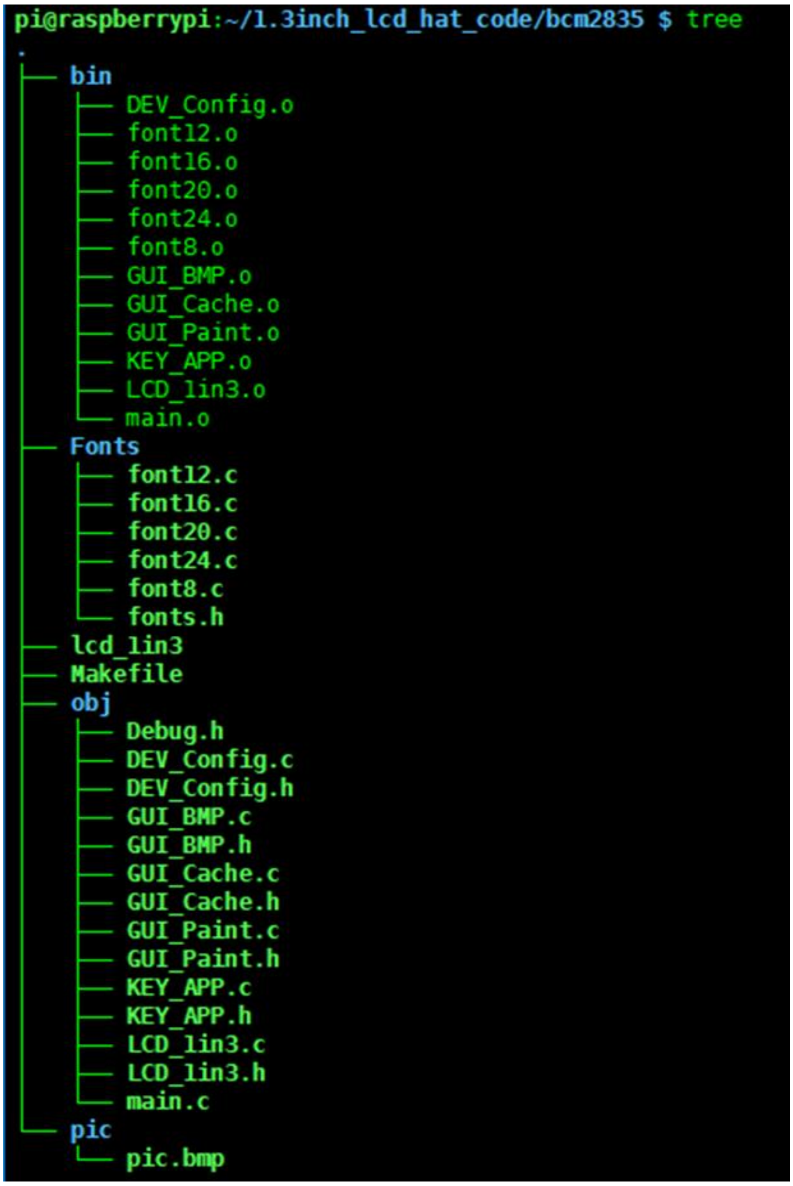
Unzip the files downloaded and copy to your Raspberry Pi

CODE ANALYSIS

The code is tested on Raspberry Pi 3 Mode B. Three codes provided, which are based on WiringPi, BCM2835 and Python separately.

C CODE

The BCM2835 code and WiringPi code are both written by C code. Their difference is the bottom hardware interfaces. Enter the directory of BCM2835 project or WiringPi project and execute command tree, you can get the list as below:



```
pi@raspberrypi:~/1.3inch_lcd_hat_code/bcm2835 $ tree
.
├── bin
│   ├── DEV_Config.o
│   ├── font12.o
│   ├── font16.o
│   ├── font20.o
│   ├── font24.o
│   ├── font8.o
│   ├── GUI_BMP.o
│   ├── GUI_Cache.o
│   ├── GUI_Paint.o
│   ├── KEY_APP.o
│   ├── LCD_lin3.o
│   └── main.o
├── Fonts
│   ├── font12.c
│   ├── font16.c
│   ├── font20.c
│   ├── font24.c
│   ├── font8.c
│   └── fonts.h
├── lcd_lin3
├── Makefile
├── obj
│   ├── Debug.h
│   ├── DEV_Config.c
│   ├── DEV_Config.h
│   ├── GUI_BMP.c
│   ├── GUI_BMP.h
│   ├── GUI_Cache.c
│   ├── GUI_Cache.h
│   ├── GUI_Paint.c
│   ├── GUI_Paint.h
│   ├── KEY_APP.c
│   ├── KEY_APP.h
│   ├── LCD_lin3.c
│   ├── LCD_lin3.h
│   └── main.c
└── pic
    └── pic.bmp
```

bin/: Folder for *.o target file

Fonts/: Folder for common font files like 0805, 1207 and 1611 fonts

Pic/: Folder for bmp image. Make sure the images you saved is 240x240.

Obj/: Workspace of functions file:

DEBUG.h: debug header file, if set USE_DEBUG as 1, can use DEBUG() function to print debug information. works like printf();

DEV_Config.c(.h): Define the PINs of Raspberry Pi and communication type. Different between BCM2835 and WiringPi code.

GUI_Paint.c(.h): Define several paint functions, like draw point, line, circle, string and so on. You can use it as libraries.

GUI_Cache.c(.h): Define a buffer for data, size 240x240xRGB

GUI_BMP.c(.h): function that read data of bmp image and save to buffer of Raspberry Pi

KEY_APP.c(.h): Application function of keys

LCD_1in3.c(.h): Driver functions of LCD

There are two files:

Makefile: For compiling

lcd_1in3: executable file, generated after compiling by make command.

To run the code, you need to execute the command:

```
sudo ./lcd_1in3
```

About the demo codes:

1. Initialize the SPI communication and the state of pins.

```
/* Module Init */
if(DEV_ModuleInit() != 0){
    DEV_ModuleExit();
    exit(0);
}
```

2. Initialize LCD and clear it to white

```
/* LCD Init */
printf("1.3inch LCD demo...\r\n");
LCD_Init(HORIZONTAL);
LCD_Clear(WHITE);
```

3. Initialize a RGB image, define its height, width, rotation angle 0°, color Non-inverted and set to white.

```
/* GUI */
printf("drawing...\r\n");
/*1.Create a new image cache named IMAGE_RGB and fill it with white*/
GUI_NewImage(IMAGE_RGB, LCD_WIDTH, LCD_HEIGHT, IMAGE_ROTATE_0, IMAGE_COLOR_POSITIVE);
GUI_Clear(WHITE);
```

4. Draw point, set its position, color, size and extension type.

```
/*2.Drawing on the image*/
GUI_DrawPoint(5, 10, BLACK, DOT_PIXEL_1X1, DOT_STYLE_DFT);//240 240
GUI_DrawPoint(5, 25, BLACK, DOT_PIXEL_2X2, DOT_STYLE_DFT);
GUI_DrawPoint(5, 40, BLACK, DOT_PIXEL_3X3, DOT_STYLE_DFT);
GUI_DrawPoint(5, 55, BLACK, DOT_PIXEL_4X4, DOT_STYLE_DFT);
```

5. Draw line, set it begin position, color, dotted/solid and width

```
GUI_DrawLine(20, 10, 70, 60, RED, LINE_STYLE_SOLID, DOT_PIXEL_1X1);
GUI_DrawLine(70, 10, 20, 60, RED, LINE_STYLE_SOLID, DOT_PIXEL_1X1);
GUI_DrawLine(170, 15, 170, 55, RED, LINE_STYLE_DOTTED, DOT_PIXEL_1X1);
GUI_DrawLine(150, 35, 190, 35, RED, LINE_STYLE_DOTTED, DOT_PIXEL_1X1);
```

6. Draw rectangle, set its begin position, color, full/empty and width of line

```
GUI_DrawRectangle(20, 10, 70, 60, BLUE, DRAW_FILL_EMPTY, DOT_PIXEL_1X1);
GUI_DrawRectangle(85, 10, 130, 60, BLUE, DRAW_FILL_FULL, DOT_PIXEL_1X1);
```

7. Draw circle, set its center, radius, color, full/empty and width of line

```
GUI_DrawCircle(170, 35, 20, GREEN, DRAW_FILL_EMPTY, DOT_PIXEL_1X1);
GUI_DrawCircle(170, 85, 20, GREEN, DRAW_FILL_FULL, DOT_PIXEL_1X1);
```

8. Display strings, set their being position, contents, font size, color and the background color

```
GUI_DrawNum(5, 120, 123456789, &Font20, BLUE, IMAGE_BACKGROUND);
```

9. Display numbers, set their begin position, parameters, size, color and background color

```
GUI_DrawNum(5, 120, 123456789, &Font20, BLUE, IMAGE_BACKGROUND);
```

10. Save data to the buffer of LCD and refresh to LCD

```
/*3.Refresh the picture in RAM to LCD*/
LCD_Display();
DEV_Delay_ms(2000);
```

11. Display image, write path of image and its name

```
/* show bmp */
printf("show bmp\r\n");
GUI_ReadBmp("./pic/time.bmp");
LCD_Display();
DEV_Delay_ms(2000);
```

12. Listening Keys:

```
/* Monitor button */
printf("Listening KEY\r\n");
KEY_Listen();
```

13. Exit

```
/* Module Exit */
DEV_ModuleExit();
return 0;
```

Notes:

The image defined by GUI_NewImage() function should be flipped, which is realized by invert coordinates. The LCD supports partial refresh, thus the invert has been done on LCD_DisplayWindows() function of LCD_1in3.c, user needn' t to invert it again. This feature is used in KEY_APP.c

```
if(GET_KEY_UP == 0) {
    while(GET_KEY_UP == 0) {
        GUI_DrawRectangle(65, 45, 115, 95, RED, DRAW_FILL_FULL, DOT_PIXEL_DFT);
        GUI_DrawString_EN(82, 62, "U", &Font24, IMAGE_BACKGROUND, BLUE);
        LCD_DisplayWindows(65, 45, 115, 95);
    }
    GUI_DrawRectangle(65, 45, 115, 95, WHITE, DRAW_FILL_FULL, DOT_PIXEL_DFT);
    GUI_DrawRectangle(65, 45, 115, 95, RED, DRAW_FILL_EMPTY, DOT_PIXEL_DFT);
    GUI_DrawString_EN(82, 62, "U", &Font24, IMAGE_BACKGROUND, BLUE);
    LCD_DisplayWindows(65, 45, 115, 95);
}
```

PYTHON

Python code will be much simply

Enter the directory of python code and use ls command:

```
main.py  ST7789.py  ST7789.pyc  time.bmp
```

ST7789.py is drive code, mian.py is main code and the time.bmp is bmp picture.

Execute command sudo pyton mian.py to run the code.

1. Initialize pins of ST7789 and corresponding registers, clear screen:

```
# 240x240 display with hardware SPI:
disp = ST7789.ST7789(SPI.SpiDev(bus, device),RST, DC, BL)

# Initialize library.
disp.Init()

# Clear display.
disp.clear()
```

2. Create RGB image by image libraries, define its length, width and full it with white

```
# Create blank image for drawing.
image1 = Image.new("RGB", (disp.width, disp.height), "WHITE")
```

3. Draw lines, frames and string with measure Draw

```
draw = ImageDraw.Draw(image1)
#font = ImageFont.truetype('/usr/share/fonts/truetype/freefont/FreeMonoBold.ttf', 16)
print "***draw line"
draw.line([(60,60),(180,60)], fill = "BLUE",width = 5)
draw.line([(180,60),(180,180)], fill = "BLUE",width = 5)
draw.line([(180,180),(60,180)], fill = "BLUE",width = 5)
draw.line([(60,180),(60,60)], fill = "BLUE",width = 5)
print "***draw rectangle"
draw.rectangle([(70,70),(170,80)],fill = "RED")

print "***draw text"
draw.text((90, 70), 'WaveShare ', fill = "BLUE")
draw.text((90, 120), 'Electronic ', fill = "BLUE")
draw.text((90, 140), '1.3inch LCD ', fill = "BLUE")
disp.ShowImage(image1,0,0)
time.sleep(3)
```

4. Save data to buffer of LCD and display

```
disp.ShowImage(image1,0,0)
time.sleep(3)
```

5. Open a BMP image and refresh

```
image = Image.open('pic.jpg')
disp.ShowImage(image,0,0)
```

FBTFT PORTING

Framebuffer is a portion of RAM containing a bitmap that drives a video display. It is a memory buffer containing a complete frame of data. That is it use a memory buffer to save data for display, if you want to change the display, you just need to change the data which is saved on the buffer.

There is a open-source project on Github, it realize the framebuffer driver for Raspberry Pi to use TFT LCD. Here we describe about how to use fbtft driver to drive 1.3inch LCD HAT

1. Open and edit configuration file to enable modules

```
sudo nano /etc/modules
```

Append three statements to the end. (first one is to enable SPI and another is to enable fbtft module)

```
spi-bcm2835
flexfb
fbtft_device
```

2. Create a new configure file

```
sudo nano /etc/modprobe.d/fbtft.conf
```

3. Save these sttements to the file

```
options fbtft_device name=flexfb
gpios=reset:27,dc:25,cs:8,led:24 speed=40000000 bgr=1 fps=60
custom=1 height=240 width=240
options flexfb setaddrwin=0 width=240 height=240 init=-1,0x11,-
2,120,-1,0x36,0x70,-1,0x3A,0x05,-
1,0xB2,0x0C,0x0C,0x00,0x33,0x33,-1,0xB7,0x35,-1,0xBB,0x1A,-
1,0xC0,0x2C,-1,0xC2,0x01,-1,0xC3,0x0B,-1,0xC4,0x20,-
1,0xC6,0x0F,-1,0xD0,0xA4,0xA1,-1,0x21,-
```

```
1,0xE0,0x00,0x19,0x1E,0x0A,0x09,0x15,0x3D,0x44,0x51,0x12,0x03,0
x00,0x3F,0x3F,-
1,0xE1,0x00,0x18,0x1E,0x0A,0x09,0x25,0x3F,0x43,0x52,0x33,0x03,0
x00,0x3F,0x3F,-1,0x29,-3
```

Note: There are two statements begin with "options"

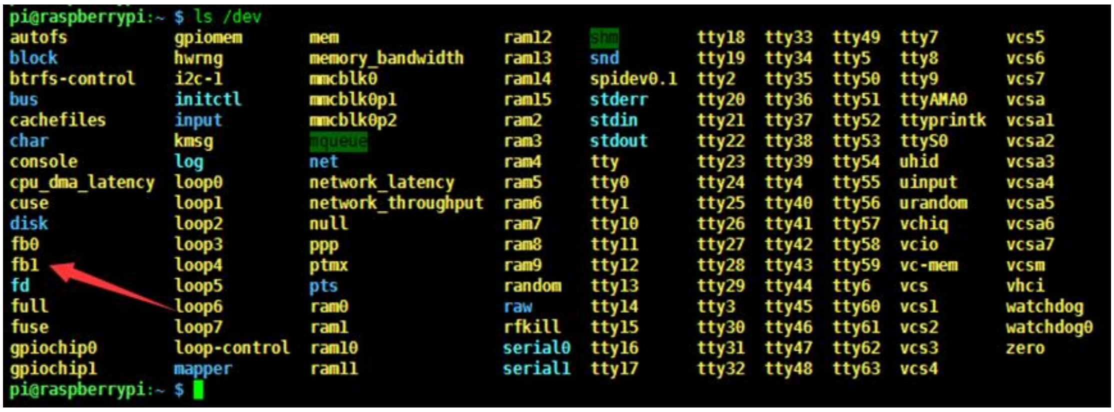
gpios=reset:27,dc:25,cs:8,led:24 This statement configure pins of LCD

height=240 width=240 This one set the resolution of LCD.

4. Restart your Raspberry Pi

```
sudo reboot
```

5. After restart you can find that a fb1 device is listed at /dev, it means the device has beed enabled successfully.



Display a picture

```
sudo python fb.py
```

Display Desktop

The resolution of this LCD is only 240x240, we can try to display the desktop of Raspbian to the screen,

To display the desktop, we just need to copy the data of fb0 to fb1, keep the fb0 and fb1 being same.

1) Install tools

```
sudo apt-get install cmake git
```

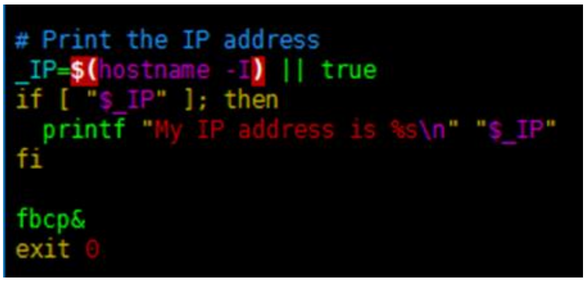
2) Download the open-source code

```
cd ~
git clone https://github.com/tasanakorn/rpi-fbcp
cd rpi-fbcp/
mkdir build
cd build/
cmake ..
make
sudo install fbcp /usr/local/bin/fbcp
```

3) Set the code to auto-run when booting

```
sudo nano /etc/rc.local
```

At fbcp& before exit 0. “&” is necessary for code run in background, otherwise the OS cannot boot anymore.



4) Set display size on config file

```
sudo vi /boot/config.txt
```

5) Append to the file:

```
hdmi_force_hotplug=1
hdmi_cvt=300 300 60 1 0 0 0
hdmi_group=2
hdmi_mode=1
```

```
hdmi_mode=87

display_rotate = 1
```

Here is to set the resolution for OS GUI, the result effect is that display on 1.3inch LCD in proportion. Here the best display is to set the resolution 300x300

After restart the Pi, the LCD will display desktop of Pi.

Display Video

1. There is a video on examples, we can use omxplayer to display it for a try

   Install omxplayer

   sudo apt-get install omxplayer

2. Display the video

   sudo omxplayer letitgo.mp4