



# 1.3inch LCD HAT 用户手册

## 产品概述

1.3inch LCD HAT 是微雪电子专为树莓派开发的 1.3 寸 LCD 显示屏模块，分辨率为 240 \* 240，LCD 带有内部控制器，使用 SPI 接口通信，与树莓派 ZERO 大小一致，已封装好基本函数，可以实现图片旋转、画点、线、圆、矩形，显示英文字符，显示 BMP 图片。

提供树莓派 BCM2835 库，WiringPi 库，以及 python 例程。

## 产品参数

工作电压:	3.3V
通信接口:	SPI
屏幕类型:	TFT
控制芯片:	ST7789VM
分辨率:	240(H)RGB x 240(V)
显示尺寸:	23.4 (H) x 23.4 (V) mm
像素大小:	0.0975 (H) x 0.0975 (V) mm
产品尺寸	65 x 30.2(mm)

## 接口说明

功能引脚	树莓派接口 (BCM)	描述
3V3	3V3	3.3V 电源正
GND	GND	电源地
CLK	P11/P_SCK	SPI 时钟输入
DIN	P10/P_MOSI	SPI 数据输入
CS	P8/P_CEO	片选, 低电平有效
DC	P25	数据/命令选择
RST	P27	复位
BL	P24	背光
KEY1	P21	按键 1GPIO
KEY2	P20	按键 2GPIO
KEY3	P16	按键 3GPIO
摇杆 Up	P6	摇杆上
摇杆 Down	P19	摇杆下
摇杆 Left	P5	摇杆左
摇杆 Right	P26	摇杆右
摇杆 Press	P13	摇杆按下

## 硬件说明

### 1. LCD 及其控制器

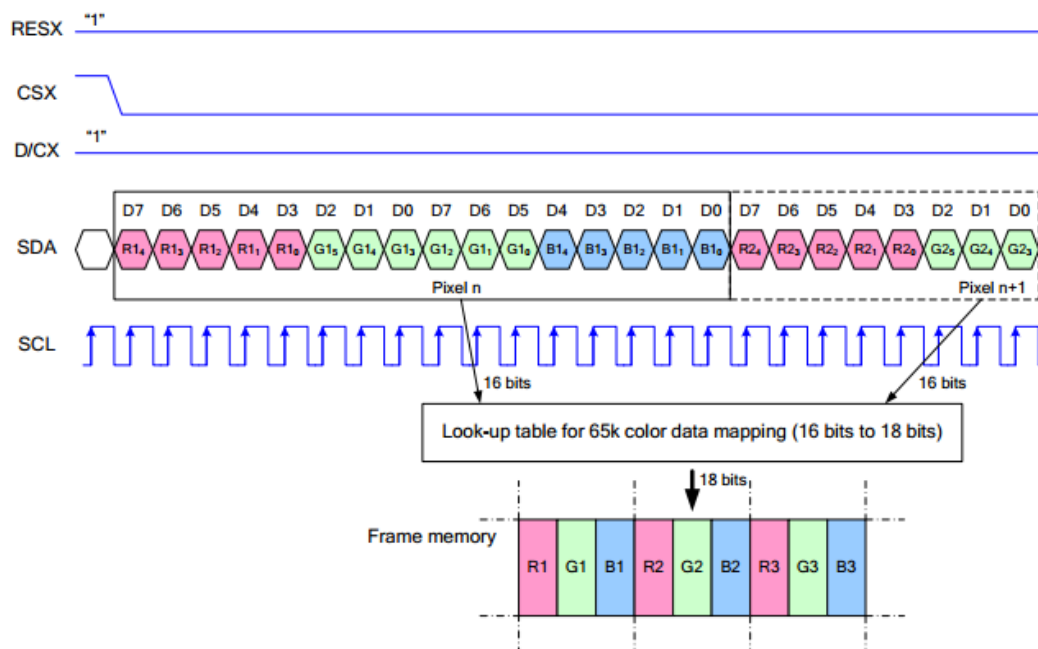
本款 LCD 使用的内置控制器为 ST7789VM，是一款 240 x RGB x 320 像素的 LCD 控制器，而本 LCD 本身的像素为 240(H)RGB x 240(V)，同时由于初始化控制可以初始化为横屏和竖屏两种，因此 LCD 的内部 RAM 并未完全使用。

查看数据手册可以得知该控制器支持 12 位，16 位以及 18 位每像素的输入颜色格式，即 RGB444, RGB565, RGB666 三种颜色格式，本屏幕使用 RGB565 格式的颜色格式，这也是常用的 RGB 格式。

对于大部分的 LCD 控制器而言，都可以配置控制器的通信方式，通常都有 8080 并行接口、三线 SPI、四线 SPI 等通信方式。此 LCD 使用四线 SPI 通信接口，这样可以大大的节省 GPIO 口，同时通信速度也会比较快。

### 2. 通信协议

从上的得知使用的是 4 线 SPI 通信，查阅数据手册可以得到如下的通信时序图，以传输 RGB565 为例：



注：与传统的 SPI 协议不同的地方是：由于是只需要显示，故而将从机发往主机的数据线进行了隐藏，该表格详见 Datasheet Page 105。

RESX 为复位，模块上电时拉低，通常情况下置 1；

CSX 为从机片选， 仅当 CS 为低电平时，芯片才会被使能。

D/CX 为芯片的数据/命令控制引脚，当 DC = 0 时写命令，当 DC = 1 时写数据

SDA 为传输的数据，即 RGB 数据；

SCL 为 SPI 通信时钟。

对于 SPI 通信而言，数据是有传输时序的，即时钟相位（CPHA）与时钟极性(CPOL)的组合：

CPHA 的高低决定串行同步时钟是在第一时钟跳变沿还是第二个时钟跳变沿数据被采集，当 CPHL = 0，在第一个跳变沿进行数据采集；

CPOL 的高低决定串行同步时钟的空闲状态电平，CPOL = 0，为低电平。CPOL 对传输协议没有很多的影响；

这两者组合就成为四种 SPI 通信方式，国内通常使用 SPI0,即 CPHL = 0，CPOL = 0

从图中可以看出，当 SCLK 第一个下降沿时开始传输数据，一个时钟周期传输 8bit 数据，使用 SPI0，按位传输,高位在前,低位在后。

## 树莓派使用

### 安装必要的函数库

需要安装必要的函数库（wiringPi、bcm2835、python 库），否则以下的示例程序可能无法正常工作。安装方法详见：

[http://www.waveshare.net/wiki/Pioneer600\\_Datasheets](http://www.waveshare.net/wiki/Pioneer600_Datasheets)

如若使用 python 库，装完上述基本的库之后，还有如下库需要安装

```
sudo apt-get install python-imageing
```

在官网上找到对应产品，在产品资料打开下载路径，在 wiki 中下载示例程序：

#### 文档

- [用户手册](#)
- [原理图](#)

#### 程序

- [示例程序](#)

得到解压包，解压并将程序复制到树莓派中。

### 程序分析

本演示程序使用树莓派 3B 进行了验证。

提供 BCM2835、WiringPi、python 三种例程，其本质为 C 语言与 python 的区别

#### C 语言

无论是使用 BCM2835 驱动还是使用 WiringPi 驱动，其都为 C 语言编写的，唯一不同的只是底层硬件接口的不同，无论是进入 BCM2835 工程目录，还是在在 WiringPi 工程目录下，运行命令 tree，将得到如下的工程目录图：

```
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/:目录下均为目标文件.o，用户不需要理会；

Fonts/:为常见的字体库，例如 0805、1207、1611 等字体；

Pic/:为存放的 bmp 图片，保证图片的像素为 240\*240,这样显示函数就不要做任何改动；

Obj/: 为工作目录，其下定义了多个函数，其中：

**DEBUG.h:** 此函数为调试头文件，把其中 USE\_DEBUG 定义为 1,即可使用 DEBUG()来打印调试信息，与 printf()的使用是一样的；

**DEV\_Config.c(.h):** 定义了树莓派的管脚及通信方式, 根据 BCM2835 与 WiringPi 的不同而不同;

**GUI\_Paint.c(.h):** 为基本的图片创建, 画点、线、圈、框、数字、字符等功能, 当成库使用即可;

**GUI\_Cache.c(.h):** 定义了一个 240 X 240 X RGB 的一个数据缓存区域;

**GUI\_BMP.c(.h):** 为读取 bmp 图片并将其写入到树莓派上的图片缓存中

**KEY\_APP.c(.h):** 为按键的应用程序;

**LCD\_1in3.c(.h):** 为屏幕的驱动程序。

在工作目录上还可以看到另外两个文件:

**Makefile:** 工程的编译规则, 如果更改了代码, 需要先执行 `make clear` 清楚全部文件依赖以及生产的可执行文件, 然后再执行 `make`, 这样 `makefile` 就会自动编译整个项目从而生成 可执行文件。

`lcd_1in3:` 可执行文件, 通过 `make` 命令生成, 通过执行: `sudo ./lcd_1in3` 运行程序。

介绍完了整个工程目录, 下面是如何调用函数来实现功能:

1. 初始化整个模块的 SPI 通信, 管脚状态, 若未定义成功则退出:

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

2. 初始化 LCD, 并清除 LCD 为白色:

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

3. 初始化一张 RGB 图片, 定义其长、宽、图片翻转 0 度、不翻转颜色, 并清空为白色:

```

/* 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. 画点，设置点的坐标，颜色、大小、扩展方式：

```

/*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. 画线，设置线的起始坐标、颜色、虚线还是实线、宽度：

```

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. 画框、设置框的对角线起始坐标、颜色、内部是否填充、线的宽度：

```

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. 画圈，设置圈的圆心坐标、半径、颜色、内部是否填充、线的宽度：

```

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. 显示字符，设置字符的起点坐标、内容、字体大小、字体颜色、字体背景色：

```

GUI_DrawString_EN(5, 70, "hello world", &Font16, WHITE, BLACK);
GUI_DrawString_EN(5, 90, "waveshare", &Font20, RED, IMAGE_BACKGROUND);

```

9. 显示数字，设置数字的起点坐标、一个数字常量、字体大小、字体颜色、字体背景色：

```

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

```

10. 把图片的数据发送到 LCD 缓存上，并打开显示：

```

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

```

11. 显示图片，写入图片的路径及图片的名称：

```

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

```

12. 按键监听：



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

### 13. 退出模块:

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

注意事项:

使用 GUI\_NewImage()函数定义的图片是可以翻转的, 通过坐标的转换来实现的, 而 LCD 是支持局部开窗刷新的, 因此在 LCD\_1in3.c 中 LCD\_DisplayWindows(), 也进行了相应的转换, 不需要再次转换, 在 KEY\_APP.c 中得到了很好的体现:

```
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

对于 python 而言, 是比较简单的。

运行 ls:

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

ST7789.py 为驱动程序, main.py 为主函数, time.bmp 为图片

运行 `sudo python main.py` 即可以在硬件上实现

### 1. 初始化 ST7789 的管脚, 并初始化寄存器, 并清屏:

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

# Initialize library.
disp.Init()

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

2. 使用 image 库，创建一个 RGB 图片，定义长与宽、并填充为白色

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

3. 使用 draw 的一些方法，来画线、框、字体

```
draw = ImageDraw.Draw(imgel)
#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(imgel,0,0)
time.sleep(3)
```

4. 将图片刷新到 LCD 缓存中

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

5. 打开 bmp 图片，并将其刷新到 LCD 上

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

## Fbtf 移植

**Framebuffer** 是用一个视频输出设备从包含完整的帧数据的一个内存缓冲区中来驱动一个视频显示设备。简单的来说，就是使用一个内存区来存储显示内容，改变内存的数据就可以改变显示的内容。

在 [github](https://github.com/notro/fbtf) 上有一个开源工程 <https://github.com/notro/fbtf> 完整的实现了 **framebuffer** 驱动，让树莓派完美支持 **tft** 液晶。下面来介绍一下如何使用 **fbtf** 驱动 1.3inch LCD HAT。

1. 打开编辑配置文件，启用一些模块。

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

2. 在文件后面添加如下三个语句，第一行是确保屏幕的 **SPI** 已经启动并正在运行，第二个命令实际是启动 **fbtf** 模块。

```
spi-bcm2835
flexfb
fbtf_device
```

3. 新建另外一个配置文件，配置 **fbtf**

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

4. 将下面语句添加到新建的空白文件中，

```
options fbtf_device name=flexfb gpios=reset:27,dc:25,cs:8,led:
24 speed=4000000 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,0x44,0x51,0x12,0x03,0x00,0x3F,0x3F,-
1,0xE1,0x00,0x18,0x1E,0x0A,0x09,0x25,0x3F,0x43,0x52,0x33,0x03,0x00,0x3F,0x3F,-1,0x29,-3
```

这里需要注意一下，这里是以 **options** 开头的两行语句。

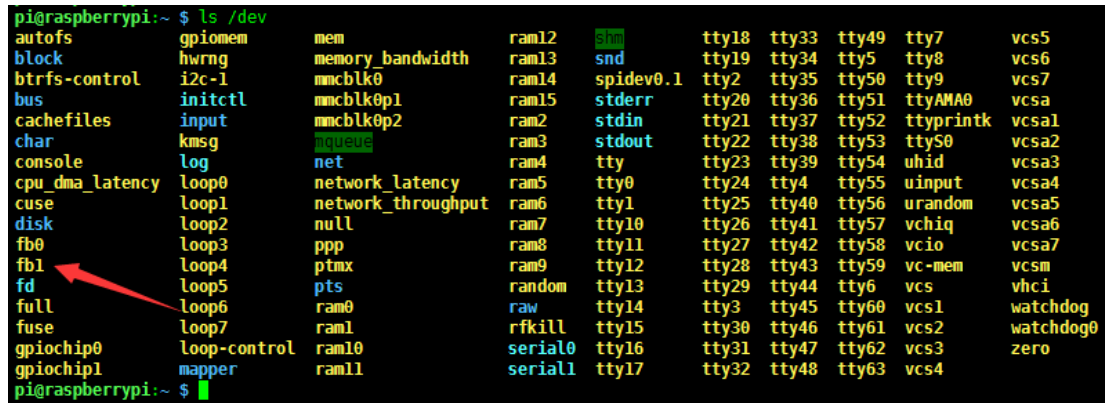
**gpios=reset:27,dc:25,cs:8,led:24** 这个设置屏幕对应的引脚，

height=240 width=240 设置屏幕分辨率大小。

5. 此时重启一下树莓派。

```
sudo reboot
```

6. 查看设备可以发现多了一个 fb1 设备，则说明设备已经成功启动了



```

pi@raspberrypi:~$ ls /dev
autofs          gpiomem        mem             ram12           shm             tty18           tty33           tty49           tty7            vcs5
block           hwrng          memory_bandwidth ram13           snd             tty19           tty34           tty5            tty8            vcs6
btrfs-control   i2c-1          mmcblk0         ram14           spidev0.1       tty2            tty35           tty50           tty9            vcs7
bus             initctl        mmcblk0p1       ram15           stderr          tty20           tty36           tty51           ttyAMA0         vcsa
cachefiles      input          mmcblk0p2       ram2            stdin           tty21           tty37           tty52           ttyprintk       vcsa1
char            kmsg           mqqueue         ram3            stdout          tty22           tty38           tty53           ttyS0           vcsa2
console         log            net             ram4            tty             tty23           tty39           tty54           uhid            vcsa3
cpu_dma_latency loop0           network_latency ram5            tty0            tty24           tty4            tty55           uinput          vcsa4
cuse            loop1          network_throughput ram6            tty1            tty25           tty40           tty56           urandom          vcsa5
disk            loop2          null            ram7            tty10           tty26           tty41           tty57           vchiq           vcsa6
fb0             loop3          ppp             ram8            tty11           tty27           tty42           tty58           vcio            vcsa7
fb1             loop4          ptmx            ram9            tty12           tty28           tty43           tty59           vc-mem          vcsm
fd              loop5          pts             random          tty13           tty29           tty44           tty6            vcs             vhci
full            loop6          ram0            raw             tty14           tty3            tty45           tty60           vcs1            watchdog
fuse            loop7          ram1            rkill           tty15           tty30           tty46           tty61           vcs2            watchdog0
gpiochip0       loop-control   ram10           serial0         tty16           tty31           tty47           tty62           vcs3            zero
gpiochip1       mapper         ram11           serial1         tty17           tty32           tty48           tty63           vcs4
pi@raspberrypi:~$
  
```

- 显示一张图片

```
sudo python fb.py
```

- 显示用户界面

最后我们将用户界面到 1.3inch LCD HAT 上，虽然这个屏幕只有 240x240 分辨率，

我们还是试一下将用户界面显示到屏幕上看下有什么样的效果。

显示用户界面只需将 fb0 上的内容直接拷贝到 fb1 上，fb0 和 fb1 同步。

首先安装一下工具软件

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

使用 `github` 上的开源代码来实现这个功能，下载代码并安装。

```
cd ~

git clone https://github.com/tasanakorn/rpi-fbcp

cd rpi-fbcp/

mkdir build

cd build/

cmake ..

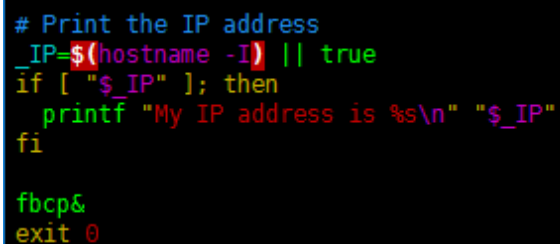
make

sudo install fbcp /usr/local/bin/fbcp
```

设置开机启动。

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

设置开机启动。在 `exit 0` 前面添加 `fbcp&`。注意一定要添加"&" 后台运行，否则可能会出现系统不能启动的情况。



```
# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
    printf "My IP address is %s\n" "$_IP"
fi

fbcp&
exit 0
```

最后在 `/boot/config.txt` 文件中设置用户界面显示尺寸。

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

在文件最后面添加上

```
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
```

这里是设置系统界面分辨率，最后显示的效果是按照比例缩放显示在 1.3inch LCD 上。这里设置分辨率为 300x300 显示最佳。

启动树莓派后可以发现屏幕已经出现用户界面了。

- [播放视频](#)

示例中有一段视频，下面我们可以试一下用屏幕播放视频。首先安装 omxplayer

```
sudo apt-get install omxplayer
```

播放视频

```
sudo omxplayer letitgo.mp4
```