

正点原子 littleVGL 开发指南

littleVGL 移植

开发指南

正点原子
广州市星翼电子科技有限公司

修订历史

版本	日期	原因
V1.00	2020/05/01	第一次发布

littleVGL 移植

1. littleVGL 介绍

littleVGL 可以说是这 2 年才刚开始流行的一个小型开源嵌入式 GUI 库,具有界面精美,消耗资源小,可移植度高,响应式布局等特点,全库采用纯 c 语言开发,在 2018 年初时,笔者刚开始接触到它,就被它的界面给吸引了,而且 littleVGL 库的更新速度非常快,从当初我刚接触的 V5.1 版本到现在的 V6.0 版本,这两个版本之间功能相差还是很大的,随着 littleVGL 的认知度越来越大,官方资料也逐渐丰富起来,但在国内而言,资源还是比较缺乏的,现在市面上还是缺乏一套完整而易上手的教程,因此我们正点原子团队基于此行情,特意推出 littleVGL 教程.

littleVGL 的官方网址为: <https://littlevgl.com>

littleVGL 的 github 网址为: <https://github.com/littlevgl/lvgl>

littleVGL 的在线文档网址为: <https://docs.littlevgl.com/zh-CN/html/index.html>

littleVGL 的主要特性如下:

- 具有非常丰富的内置控件,像 buttons, charts, lists, sliders, images 等
- 高级图形效果: 动画, 反锯齿, 透明度, 平滑滚动
- 支持多种输入设备,像 touchpad, mouse, keyboard, encoder 等
- 支持多语言的 UTF-8 编码
- 支持多个和多种显示设备,例如同步显示在多个彩色屏或单色屏上
- 完全自定义的图形元素
- 硬件独立于任何微控制器或显示器
- 可以缩小到最小内存 (64 kB Flash, 16 kB RAM)
- 支持操作系统、外部储存和 GPU (非必须)
- 仅仅单个帧缓冲设备就可以呈现高级视觉特效
- 使用 C 编写以获得最大兼容性(兼容 C++)
- 支持 PC 模拟器
- 为加速 GUI 设计,提供教程,案例和主题,支持响应式布局
- 提供了在线和离线文档
- 基于自由和开源的 MIT 协议

littleVGL 的要求如下:

- 16、32 或 64 位的单片机(微控制器)或处理器
- 微处理器的主频最好高于 16MHZ
- Flash/ROM:如果只用 littleVGL 核心组件的话,则至少需要 64kB 的容量,如果想完整使用的话,最好保证 180kB 以上的容量
- RAM:
 - 静态 RAM: 大约 8 到 16 kB,这取决于你所用的组件功能和 objects 控件对象类型
 - 栈: 至少为 2Kb,一般推荐值为 4kB

- 动态数据(堆): 至少 4kB,如果你用到了多个或多种控件的话,那么最好设置为 16kB 以上,这个是通过 `lv_conf.h` 配置文件中的 `LV_MEM_SIZE` 宏来定义的
- 显示缓冲区: 至少要比“水平分辨率像素”要大,一般推介值为 10 倍的“水平分辨率像素”,举个例子,假如我们屏幕的水平分辨率为 480 个像素,采用 16 位的颜色深度进行显示,即一个像素占 2 个字节,那么推介的显示缓冲区大小为 $10 \times 480 \times 2 = 9600$ 个字节
- C99 或更新的编译器,如果是用 keil 开发的话,一定得勾选“c99”模式,否则编译会报错的
- 基本的 c(或者 c++)语言知识,如:指针,结构体,回调函数



图 1.1 官方效果图

还有很多界面显示效果,请自行上 littleVGL 官网查看.我这里就不一一列举了

2. littleVGL 移植

先声明一下,本教程适用于正点原子的所有 stm32 开发板,支持正点原子的全系列液晶屏,不同的开发板对液晶屏型号的支持力度大小可能会不同,这个会在相应开发板的配套例程代码中进行说明,为了方便笔者描述,本文档中的图片和素材都以正点原子的“战舰”开发板为参考,不同开发板之间如有区别,笔者将会在文档中进行指出,let's go,让我们现在开始跨进 littleVGL 的大门,那就先从移植开始吧,我把整个移植过程罗列成以下步骤,请按照如下步骤一一操作.

2.1 步骤 1,准备素材

首先我们需要从 https://littlevgl.com/download/lv_pc_simulator.zip 链接上下载到 lv_pc_simulator.zip 压缩包,或者直接使用我们正点原子已经下载好的也行,此压缩包里面包含了 littleVGL 库源代码和 littleVGL 官方演示例程,接着我们还要准备一个 Keil 工程项目,这里我不打算重头新建一个 Keil 工程,因为我觉得这些简单的操作步骤,大家在学正点原子的 stm32 开发板时,肯定已经学会了,我这里就准备在正点原子的 stm32 开发板例程上直接移植,减少一些重复造轮子的工作,那么选择哪一个实验来移植呢,肯定是选择“**标准例程-库函数版本/触摸屏实验**”了,因为此实验里面包含了触摸和液晶屏显示的基本功能,我这里以战舰开发板为参考进行移植演示,其他的开发板操作流程也是一样的,不需要去担心,最后我们拿到了如下图所示的 2 个文件,把它们放到桌面上,方便操作





图 2.1.1 素材

2.2 步骤 2,修改 Keil 工程名

我是一个有强迫症的人,不能允许 Keil 工程名和实际项目含义不一致,于是我们先开始对“触摸屏实验”工程进行整理一下,主要是修改名称,删除无用的代码

我们是移植,所以先就取名为 template 吧,先把顶层目录名改为 template,接着进入到其 USER 子目录中把 TOUCH.uvprojx, TOUCH.uvoptx 相应的改为 template.uvprojx 和 template.uvoptx,接着删除掉 TOUCH.uvguix.Administrator 文件,接着双击 template.uvprojx 文

件打开 Keil 工程,点击  打开对话框,把 Targets 名称从 TOUCH 改为 TEMPLATE,接着点击  魔术棒,切到 Output 选项卡,把 Name of Executable 的内容改为 template,接着打开 main.c 文件,删除掉无用代码,最后只留 main 函数,如下图所示:

```

23
24 int main(void)
25 {
26     delay_init();           //延时函数初始化
27     NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
28     uart_init(115200);      //串口初始化为115200
29
30     LED_Init();             //LED端口初始化
31     LCD_Init();
32     KEY_Init();
33     tp_dev.init();
34
35     while(1)
36     {
37
38     }
39 }

```

图 2.2.1 main 函数中的内容

接着我们编译一下,应该是没有任何错误和警告的

2.3 步骤 3,导入 littleVGL 库到 Keil 中

在 template 项目根目录下新建 **GUI** 和 **GUI_APP** 俩个子目录,即和 USER 目录是同级别的,GUI 目录是用来存放跟 littleVGL 库相关的所有文件的,而 GUI_APP 是用来放我们自己的 GUI 应用代码的,因为现在才刚开始移植,还来不及自己写 GUI 应用,所以 GUI_APP 目录里面先留空,我们的重点是来介绍 GUI 目录.

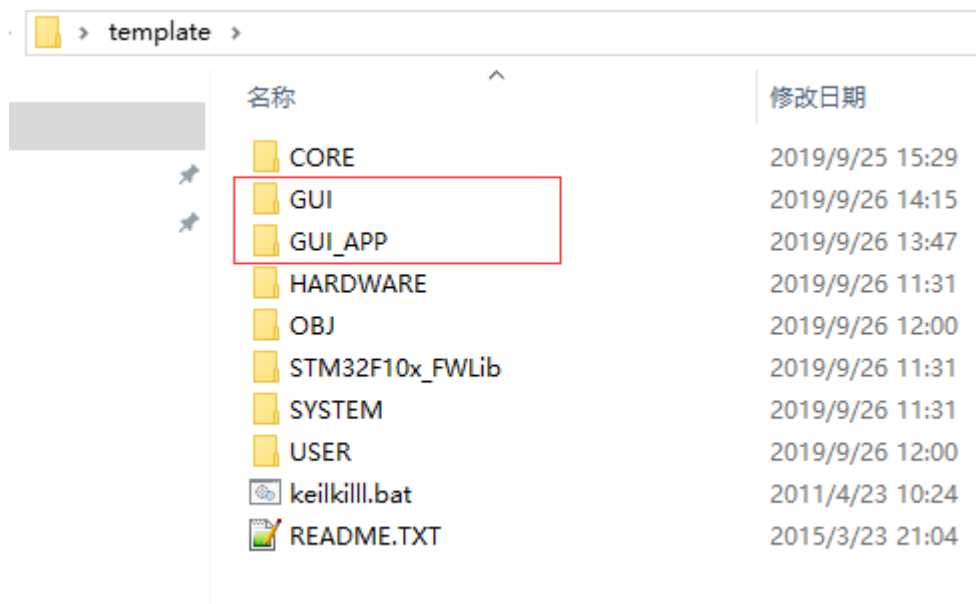


图 2.3.1 新建 GUI 和 GUI_APP 之后的目录结构

接着把 lv_pc_simulator.zip 压缩包里面的 lv_examples.zip 和 lvgl.zip 俩个子压缩包直接拷贝到 template/GUI 目录下,拷贝完成之后,接着分别对 lv_examples.zip 和 lvgl.zip 俩个子压缩包在当前目录下进行解压缩操作,解压缩完成后,可以把 lv_examples.zip 和 lvgl.zip 都删除了,接着把 template/GUI/lvgl/lv_conf_template.h 和 template/GUI/lv_examples/lv_ex_conf_tmpl.h 俩个配置模板文件统统拷贝到 template/GUI 目录下,然后对这个 2 文件分别重命名为 lv_conf.h

和 lv_ex_conf.h,接着还要在 template/GUI 目录下新建一个 lvgl_driver 子目录,这个目录是用来放底层显示驱动和触摸驱动文件的,最后 template/GUI 的目录结构如下图所示.

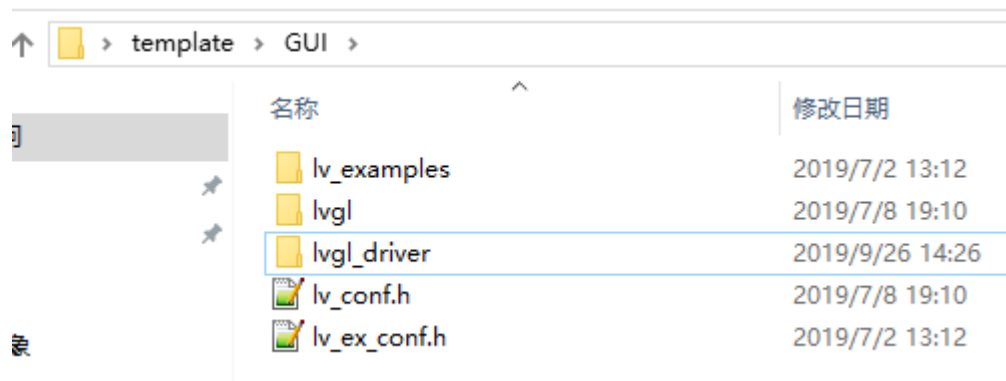



图 2.3.2 GUI 目录结构

注:在实际项目中,可以删除掉 lv_examples 目录来减少项目所占的磁盘空间,因为此目录就是专门用来存放官方的演示 demo,对我们的实际项目没有任何作用

接着我们打开 Keil 工程,点击  图标,打开分组管理面板,在 Groups 栏下新建 GUI 和 GUI_APP 两个分组,选中 GUI 分组,接着点击 Add Files 按钮,把

```
template\GUI\lvgl\src\lv_core
template\GUI\lvgl\src\lv_draw
template\GUI\lvgl\src\lv_font
template\GUI\lvgl\src\lv_hal
template\GUI\lvgl\src\lv_misc
template\GUI\lvgl\src\lv_objx
template\GUI\lvgl\src\lv_themes
```

所有目录下的所有.c 文件依次全部添加到 GUI 分组下面,最后如下图所示

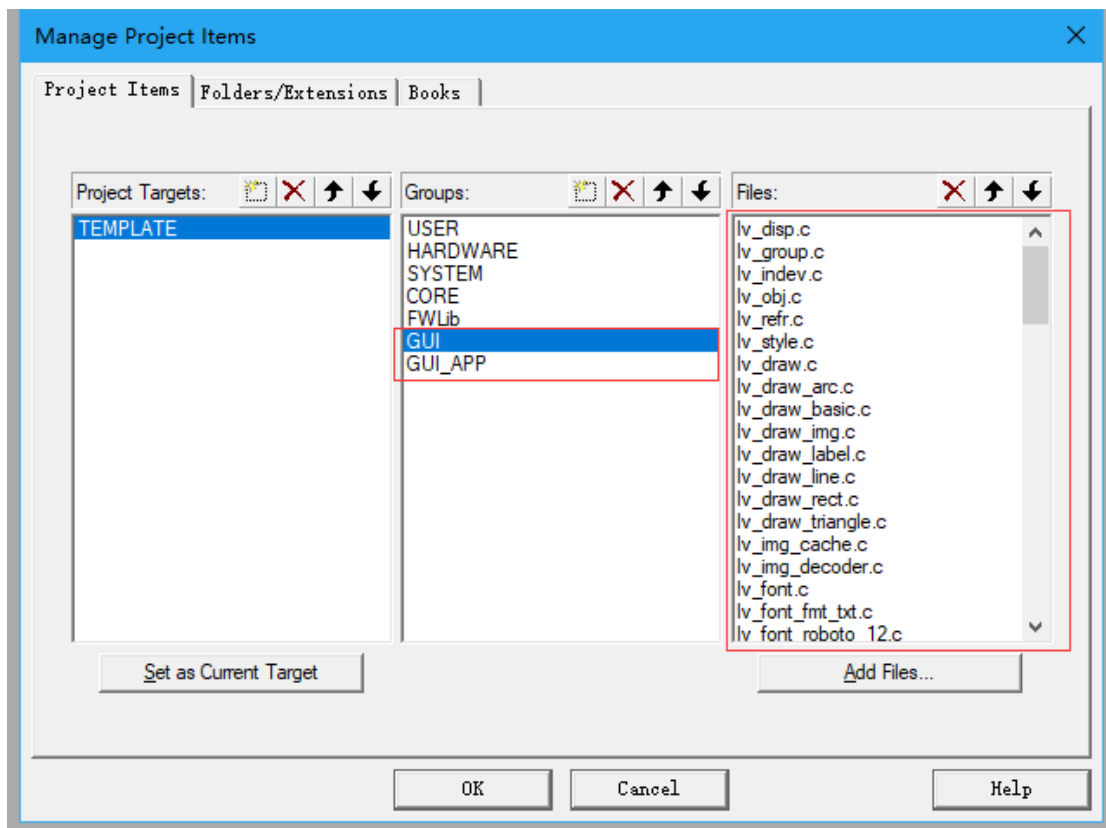



图 2.3.3 GUI 分组

最后点击 OK 按钮保存。

接着我们需要做 2 个比较重要的小操作,加大项目的栈空间到 2KB 和使能 C99 编译器功能,打开 Core 分组下的.s 启动文件,修改 Stack_Size 的值到 2KB(0x00000800),如

Stack_Size EQU 0x00000800 所示,然后点击  图标,打开面板之后,切换到 C/C++ 面板,选中 C99 Mode 复选框即可,因为 littleVGL 要求 C99 或更新的编译器,否则编译是会报错的。

2.4 步骤 4,修改 lv_conf.h 和 lv_ex_conf.h 配置文件

先打开 template\GUI 下的 lv_conf.h 文件,里面的配置项还是很多的,大家先不要纠结每个配置项到底什么意思,后面我会专门来讲解的,大家先按照我的步骤来,对于关键的配置项,我会先简单解释一下。

打开文件之后,第一个 #if 后面的 0 改为 1,使整个文件生效,接着修改 LV_HOR_RES_MAX 和 LV_VER_RES_MAX 宏的值,这个是告诉 littleVGL 你所用的液晶屏分辨率是多少,请根据自己手头液晶屏的实际分辨率大小相应设置,比如笔者手头就有如下 4 块不同型号的 TFTLCD 液晶屏,分别是 2.8 寸(320*240)电阻屏,3.5 寸(480*320)电阻屏,4.3 寸(800*480)电容屏,7 寸(800*480)电容屏,在这个 4 块 MCU 屏中,分辨率最高的为 800*480,因此我这里就设置如下:

```
#define LV_HOR_RES_MAX    (480)
#define LV_VER_RES_MAX    (800)
```

因为这样就可以一套代码兼容 4 块不分辨率的屏了,原理就是高分辨率可以兼容低分辨率。

接着我们修改 `LV_COLOR_DEPTH` 颜色深度,最常见的设置就是 1 或者 16 了,1 是用于单色屏,而 16 是用于彩色屏,这里我们设置成 16 即可,即如下所示

```
#define LV_COLOR_DEPTH 16
```

再接着我们来设置 `LV_DPI` 的值,默认值为 100,我们把他设置到 60,这个宏是用来调节界面缩放比例的,此值越大,控件分布的就越散,控件自身的间隔也会变大

```
#define LV_DPI 60
```

再接着修改 `LV_MEM_SIZE` 的大小,这个就是控制 littleVGL 中所谓的动态数据堆的大小,是用来给控件的创建动态分配空间的,我们这里设置为 16KB 的大小

```
#define LV_MEM_SIZE (16U * 1024U)
```

再接着修改 `LV_USE_GPU` 的值,默认值是 1,我们把它设置为 0,即不使能 GPU 功能

```
#define LV_USE_GPU 0
```

再接着修改 `LV_USE_FILESYSTEM` 的值,其默认值为 1,我们把他设置为 0,即不使能文件系统的功能

```
#define LV_USE_FILESYSTEM 0
```

再接着再把 `LV_THEME_LIVE_UPDATE`, `LV_USE_THEME_TEMPL`, `LV_USE_THEME_DEFAULT`, `LV_USE_THEME_ALIEN`, `LV_USE_THEME_NIGHT`, `LV_USE_THEME_MONO`, `LV_USE_THEME_MATERIAL`, `LV_USE_THEME_ZEN`, `LV_USE_THEME_NEMO` 等所有宏的值都设置为 1,即全部使能,这些宏都是跟 littleVGL 自带的主题相关的,因为后面我们要演示官方自带的例程效果,所以这里我们先全部使能,注意,在实际项目中,我们一般最多使能一个,如果我们项目根本就用不到其自带的主题,那么我们应该把这些宏全部禁止,因为这样可以节省 flash 和 ram。

至此 `lv_conf.h` 文件修改就完成了,接下来我们要修改 `lv_ex_conf.h` 文件,这个文件就简单很多了,而且这个文件也不是很重要,只有当我们要演示官方自带的例程时,才会用到,下面就简单做一下说明。

把 `LV_EX_KEYBOARD`, `LV_EX_MOUSEWHEEL`, `LV_USE_TESTS`, `LV_USE_TUTORIALS`, `LV_USE_BENCHMARK`, `LV_USE_DEMO`, `LV_USE_TERMINAL` 等宏的值全部设置为 1,其他宏保持默认即可。

2.5 步骤 5,添加定时器,为 littleVGL 提供心跳节拍

这里我打算采用定时器 3,设置其每隔 1ms 进入中断,为 littleVGL 提供 1ms 的心跳节拍,当然你也可以采用其他的定时器,原理都是一样的,定时器的代码可以直接从 `stm32 开发板的“定时器中断实验”`例程中拷贝过来,然后复制到 `template\HARDWARE` 目录下面,接着打开 Keil 工程,把 `timer.c` 文件添加到 `HARDWARE` 分组下面,接着修改 `timer.c` 文件中定时器中断服务函数,正确的如下所示:

```
#include "lvgl.h"
void TIM3_IRQHandler(void)
{
    if(TIM3->SR&TIM_IT_Update)//溢出中断
    {
        lv_tick_inc(1);//lvgl 的 1ms 心跳
    }
    TIM3->SR = (uint16_t)~TIM_IT_Update; //清除中断标志
}
```

然后在 `main` 函数中加入定时器的 `TIM3_Int_Init(arr, psc)`初始化代码,必须保证中断间隔为

1ms,不同开发板,传入的 arr, psc 参数可能会不同的,请注意!

以战舰开发板为例,此时的 main 函数代码差不多如下所示:


```
int main(void)
{
    delay_init();           //延时函数初始化
    //设置中断优先级分组为组 2: 2 位抢占优先级, 2 位响应优先级
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    uart_init(115200);      //串口初始化为 115200
    LED_Init();             //LED 端口初始化
    KEY_Init();             //按键初始化
    TIM3_Int_Init(999,71); //定时器初始化(1ms 中断),用于给 lvgl 提供 1ms 的心跳节拍
    LCD_Init();             //LCD 初始化
    tp_dev.init();          //触摸屏初始化

    lv_init();              //lvgl 系统初始化

    while(1)
    {
        tp_dev.scan(0);
        lv_task_handler();
    }
}
```

如果你的定时器代码是采用标准库或者 HAL 库开发的话,那么你还需要把定时器对应的标准库或 HAL 库驱动文件添加到 Keil 工程中.

接着我们还需要在 Keil 中配置 littleVGL 的头文件路径,点击  图标,面板打开之后,切

换到 C/C++ 面板,点击  图标,打开 Include Paths 面板,添加..\HARDWARE\TIMER, ..\GUI, ..\GUI\lvgl, ..\GUI\lv_examples, ..\GUI\lvgl_driver 几个头文件路径,如下图所示:

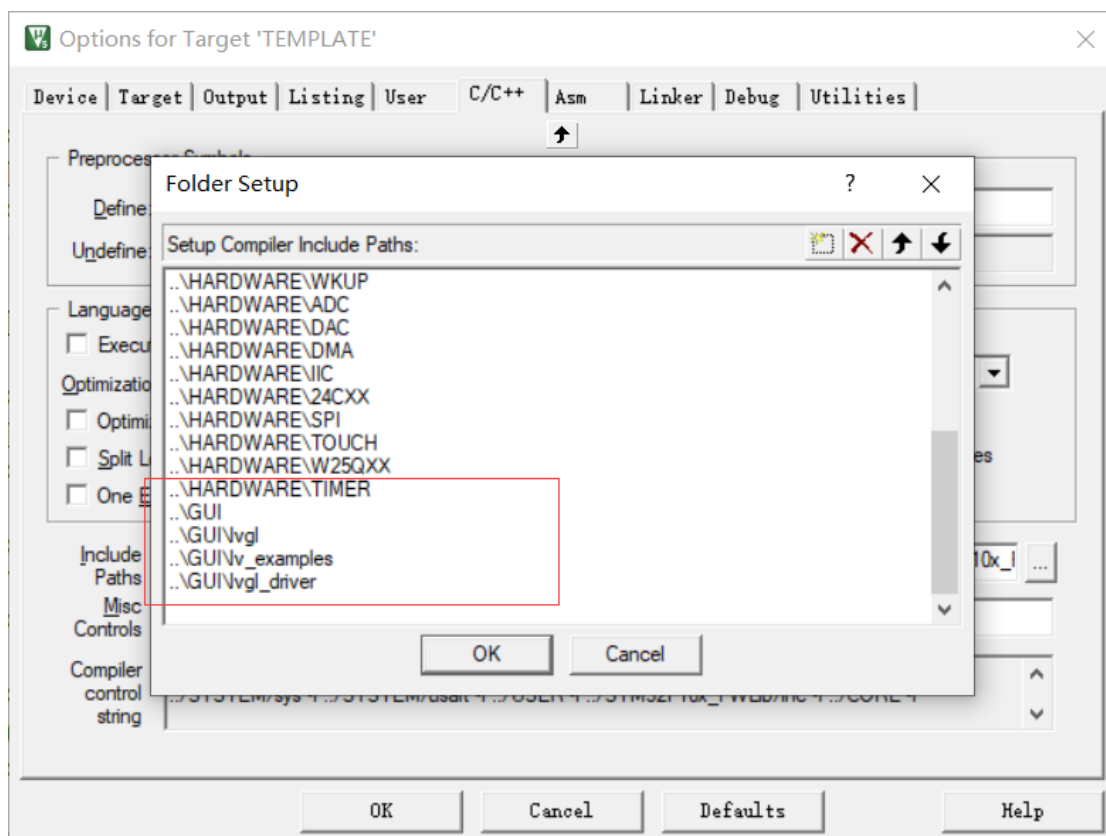



图 2.5.1 头文件路径添加

确认无误之后,哈哈,我们终于迎来了第一次编译,不出意外的话,会报一个错误和一百多个警告,错误提示应该是”..\SYSTEM\usart\usart.c(48): error: #260-D: explicit type is missing ("int" assumed)”,这个很好解决,打开 SYSTEM\usart\usart.c 文件,在 _sys_exit 函数的前面加入 void 返回值,再编译一下,就只剩下警告,没有错误了,接下来我们来解决警告,这一百多个警告中,仔细看其实就只有 68, 111, 550 这三种警告,我可以告诉大家,这个警告对我们项目没有任何影响的,但是强迫症患者看着就是难受,幸亏 Keil 可以通过设置,把某种警告给屏蔽掉,点击  图标,切换到 C/C++选项卡,在 Misc Controls 中填入 `--diag_suppress=68 --diag_suppress=111 --diag_suppress=550`,如下图所示。

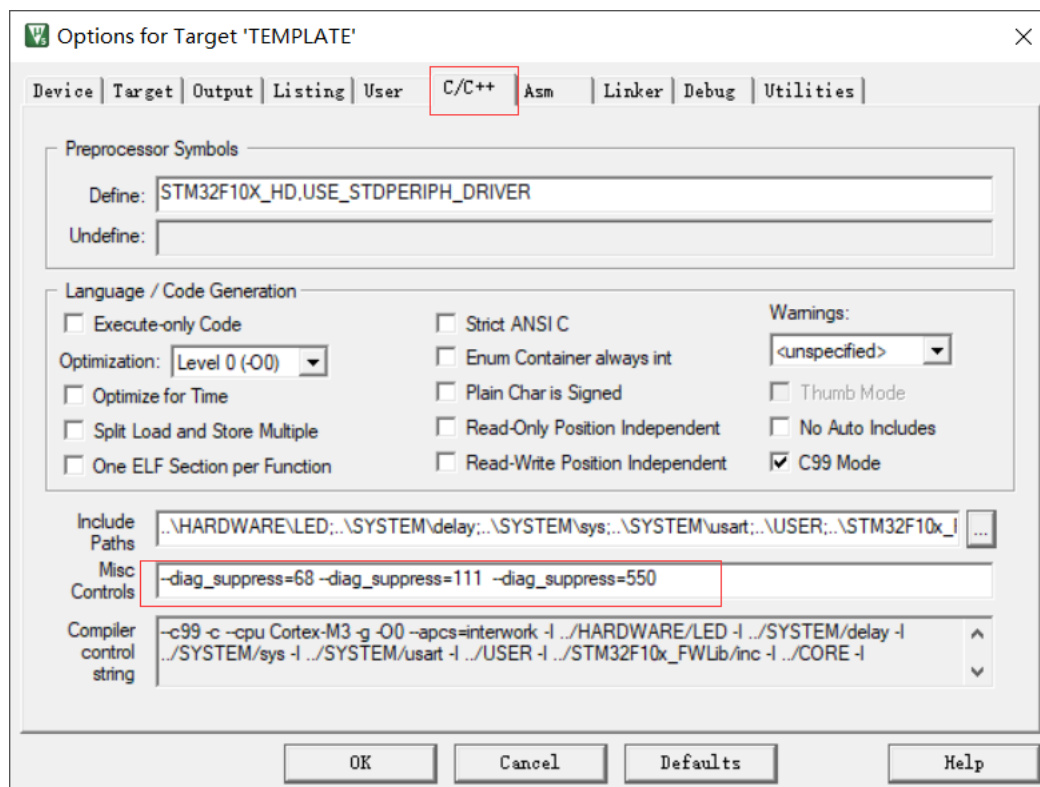


图 2.5.1 屏蔽警告

点击 OK 保存按钮之后,再来编译一下,就会发现没有任何错误和警告了.接下来我们要开始移植底层显示驱动了

2.6 步骤 6,移植底层显示驱动

littleVGL 官方给我们提供了显示驱动,输入驱动,文件系统驱动的模板文件,存放在 template\GUI\lvgl\porting 目录下,这里我们不需要文件系统,所以只需要把 lv_port_disp_template.c, lv_port_disp_template.h, lv_port_indev_template.c, lv_port_indev_template.h 四个文件拷贝到 template\GUI\lvgl_driver 目录下面,并分别重命名为 lv_port_disp.c, lv_port_disp.h, lv_port_indev.c, lv_port_indev.h,前面 2 个文件是跟显示驱动相关的,后面两个文件是跟触摸驱动相关的,我们先只需要修改 lv_port_disp.c, lv_port_disp.h 两个文件的内容就可以了,通过看注释,发现并不难移植,我这里直接给出正确的移植代码。

lv_port_disp.h 文件内容如下:

```
#ifndef LV_PORT_DISP_H
#define LV_PORT_DISP_H

#ifdef __cplusplus
extern "C" {
#endif

#include "lvgl/lvgl.h"
```

```
//函数申明
void lv_port_disp_init(void);

#ifdef __cplusplus
}
#endif

#endif
```

lv_port_disp.c 文件内容如下:

```
#include "lv_port_disp.h"
#include "lcd.h"

//函数申明
static void disp_flush(lv_disp_drv_t * disp_drv, const lv_area_t * area, lv_color_t * color_p);
#ifdef LV_USE_GPU
static void gpu_blend(lv_color_t * dest, const lv_color_t * src, uint32_t length, lv_opa_t opa);
static void gpu_fill(lv_color_t * dest, uint32_t length, lv_color_t color);
#endif

//lvgl 显示接口初始化
void lv_port_disp_init(void)
{
    static lv_disp_buf_t disp_buf;
    static lv_color_t color_buf[LV_HOR_RES_MAX * 10]; //显示缓冲区,静态的 sram
    //显示缓冲区初始化
    lv_disp_buf_init(&disp_buf, color_buf, NULL, LV_HOR_RES_MAX * 10);

    //显示驱动默认值初始化
    lv_disp_drv_t disp_drv;
    lv_disp_drv_init(&disp_drv);

    //设置屏幕的显示大小,我这里是支持正点原子的多个屏幕,采用动态获取的方式
    //如果你是用于实际项目的话,可以不用设置,那么其默认值就是 lv_conf.h 中
    //LV_HOR_RES_MAX 和 LV_VER_RES_MAX 宏定义的值
    disp_drv.hor_res = lcddev.width;
    disp_drv.ver_res = lcddev.height;
```

```
//注册显示驱动回调
disp_drv.flush_cb = disp_flush;

//注册显示缓冲区
disp_drv.buffer = &disp_buf;

#if LV_USE_GPU
//可选的,只要当使用到 GPU 时,才需要实现 gpu_blend 和 gpu_fill 接口

//使用透明度混合两个颜色数组时需要用到 gpu_blend 接口
disp_drv.gpu_blend = gpu_blend;

//用一个颜色填充一个内存数组时需要用到 gpu_fill 接口
disp_drv.gpu_fill = gpu_fill;
#endif

//注册显示驱动到 lvgl 中
lv_disp_drv_register(&disp_drv);
}

//把指定区域的显示缓冲区内容写入到屏幕上,你可以使用 DMA 或者其他的硬件加速器
//在后台去完成这个操作但是在完成之后,你必须得调用 lv_disp_flush_ready()
static void disp_flush(lv_disp_drv_t * disp_drv, const lv_area_t * area, lv_color_t * color_p)
{
    //把指定区域的显示缓冲区内容写入到屏幕
    LCD_Color_Fill(area->x1,area->y1,area->x2,area->y2,(u16*)color_p);
    //最后必须得调用,通知 lvgl 库你已经 flushing 拷贝完成了
    lv_disp_flush_ready(disp_drv);
}

//可选的
#if LV_USE_GPU

/*
    If your MCU has hardware accelerator (GPU) then you can use it to blend to memories
    using opacity
    * It can be used only in buffered mode (LV_VDB_SIZE != 0 in lv_conf.h)
    */
static void gpu_blend(lv_disp_drv_t * disp_drv, lv_color_t * dest, const lv_color_t * src,
```

```
uint32_t length, lv_opa_t opa)
{
    /*It's an example code which should be done by your GPU*/
    uint32_t i;
    for(i = 0; i < length; i++) {
        dest[i] = lv_color_mix(dest[i], src[i], opa);
    }
}

/* If your MCU has hardware accelerator (GPU) then you can use it to fill a memory with a
color
* It can be used only in buffered mode (LV_VDB_SIZE != 0 in lv_conf.h)*/
static void gpu_fill_cb(lv_disp_drv_t * disp_drv, lv_color_t * dest_buf, lv_coord_t
dest_width,
                        const lv_area_t * fill_area, lv_color_t color);
{
    /*It's an example code which should be done by your GPU*/
    uint32_t x, y;
    dest_buf += dest_width * fill_area->y1; /*Go to the first line*/

    for(y = fill_area->y1; y < fill_area->y2; y++) {
        for(x = fill_area->x1; x < fill_area->x2; x++) {
            dest_buf[x] = color;
        }
        dest_buf += dest_width; /*Go to the next line*/
    }
}

#endif
```

我们的 GUI 应用跑的流不流畅,主要是取决 lv_port_disp.c 文件了,这里需要注意一点,为了移植方便,我们的 color_buf 显示缓冲区是采用静态数组的方式定义的,优点是内部 sram 访问速度快,缺点是不能定义的过大,因为内部的 sram 少的可怜,后面的章节中我会介绍怎么把 color_buf 显示缓冲区定义到外部的大容量 sram 中,从而来提升 GUI 的流畅度!

2.7 步骤 7,移植底层触摸驱动

littleVGL 是支持很多种输入设备的,像 Touchpad, Mouse, Keypad, Encoder, Button 等统统支持,而通常情况下,我们用的最多的就是触摸屏了,他属于 Touchpad 类,通过打开 template\GUI\lvg_driver 目录下的 lv_port_indev.c 和 lv_port_indev.h,文件,不难发现只需要实现 lv_port_indev_init 和 touchpad_read 两个 API 接口就行了,其他的统统可以删除,下面我给出正确的移植代码.

lv_port_indev.h 文件内容如下:

```
#ifndef LV_PORT_INDEV_H
#define LV_PORT_INDEV_H

#ifdef __cplusplus
extern "C" {
#endif

#include "lvgl/lvgl.h"

//函数申明
void lv_port_indev_init(void);

#ifdef __cplusplus
}
#endif

#endif
```

lv_port_indev.c 文件的内容如下:

```
#include "lv_port_indev.h"
#include "touch.h"

//函数申明
static bool touchpad_read(lv_indev_drv_t * indev_drv, lv_indev_data_t * data);

//lvgl 的输入设备初始化
void lv_port_indev_init(void)
{
    lv_indev_drv_t indev_drv;

    //lvgl 支持很多种输入设备,但是我们一般常用的就是触摸屏,也就是 Touchpad
    lv_indev_drv_init(&indev_drv);
    indev_drv.type = LV_INDEV_TYPE_POINTER;
    indev_drv.read_cb = touchpad_read;
```



```
    lv_indev_drv_register(&indev_drv);
}
//将会被 lvgl 周期性调用,周期值是通过 lv_conf.h 中的 LV_INDEV_DEF_READ_PERIOD
//宏来定义的
//此值不要设置的太大,否则会感觉触摸不灵敏,默认值为 30ms
static bool touchpad_read(lv_indev_drv_t * indev_drv, lv_indev_data_t * data)
{
    static uint16_t last_x = 0;
    static uint16_t last_y = 0;

    if(tp_dev.sta & TP_PRES_DOWN) //触摸按下了
    {
        last_x = tp_dev.x[0];
        last_y = tp_dev.y[0];
        data->point.x = last_x;
        data->point.y = last_y;
        data->state = LV_INDEV_STATE_PR;
    } else { //触摸松开了
        data->point.x = last_x;
        data->point.y = last_y;
        data->state = LV_INDEV_STATE_REL;
    }

    //返回 false 代表没有缓冲的数据
    return false;
}
```

2.8 步骤 8,移植官方的演示例程

终于熬到最后一步了,我们只要把官方的演示例程代码添加到 Keil 工程中的 GUI_APP 分组下就可以了,官方的例程源码全部位于 template\GUI\lv_examples 目录下,这里我们先演示 2 个综合一点的例程。

demo 例程,其源码路径为: template\GUI\lv_examples\lv_apps\demo\

theme 例程,其源码路径为: template\GUI\lv_examples\lv_tests\lv_test_theme\

其中 theme 例程里面又包含了 2 个子例程

我们把 demo 和 theme 例程按照上面所描述的源码路径添加到 Keil 中的 GUI_APP 分组下,如图所示:

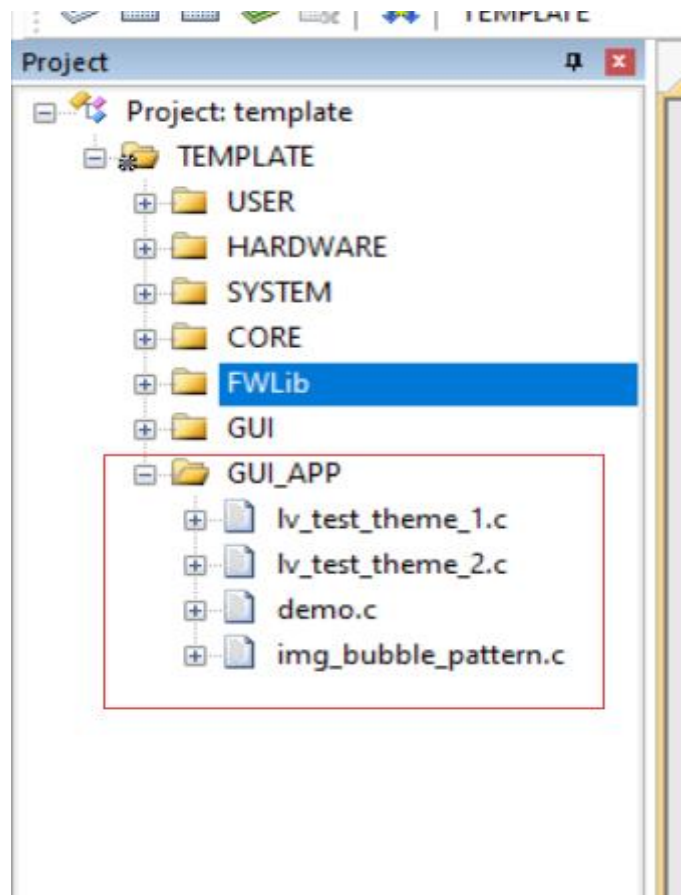


图 2.8.1 GUI_APP 分组

然后在 main 函数中添加显示驱动初始化和触摸驱动初始化的代码,总之最后 main.c 文件的代码如下:

```
#include "led.h"
#include "delay.h"
#include "key.h"
#include "sys.h"
#include "lcd.h"
#include "usart.h"
#include "touch.h"
#include "timer.h"
#include "lvgl.h"
#include "lv_port_disp.h"
#include "lv_port_indev.h"
#include "lv_apps\demo\demo.h"
#include "lv_tests\lv_test_theme\lv_test_theme_1.h"
#include "lv_tests\lv_test_theme\lv_test_theme_2.h"

#define TEST_NUM      1    //1,2,3 分别对应三个演示例程
```

```
int main(void)
{
    delay_init();           //延时函数初始化
    //设置中断优先级分组为组 2: 2 位抢占优先级, 2 位响应优先级
    NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
    uart_init(115200);      //串口初始化为 115200
    LED_Init();             //LED 端口初始化
    KEY_Init();             //按键初始化
    TIM3_Int_Init(999,71); //定时器初始化(1ms 中断),用于给 lvgl 提供 1ms 的心跳节拍
    LCD_Init();             //LCD 初始化,一定得放在 lv_init()的前面

    tp_dev.init();          //触摸屏初始化

    lv_init();              //lvgl 系统初始化
    lv_port_disp_init();    //lvgl 显示接口初始化,放在 lv_init()的后面
    lv_port_indev_init();   //lvgl 输入接口初始化,放在 lv_init()的后面

    //通过 TEST_NUM 的值来选择运行不同的例程
    #if(TEST_NUM==1)
        demo_create();
    #elif(TEST_NUM==2)
        lv_test_theme_1(lv_theme_night_init(210, NULL));
    #else
        lv_test_theme_2();
    #endif
    while(1)
    {
        tp_dev.scan(0);
        lv_task_handler();
    }
}
```

当运行 `demo_create()`;例程时,编译会发现报错的,这是因为官方的 `demo` 例程有一个小疏忽,我们打开 `GUI_APP` 分组下的 `demo.c` 文件,把 `a.user_data = NULL;`的代码给删除掉就行了,总共有 2 处,分别是在 215 行和 249 行。

因为我们的代码是支持电阻屏的,而电阻屏一般都需要先进行校准操作,所以我在 `touch.c` 文件的 `TP_Init` 函数中加入了一个小操作,那就是在开机之前如果先按住 `KEY0` 键不放,就可以先进入到电阻屏校准程序,校准完成之后,再进入到 `littleVGL` 演示例程,代码很简单,那就是直接把 `if(TP_Get_Adjdata())`改成 `if((KEY0==1)&&TP_Get_Adjdata())`就可以了,其他的地方不需要改动

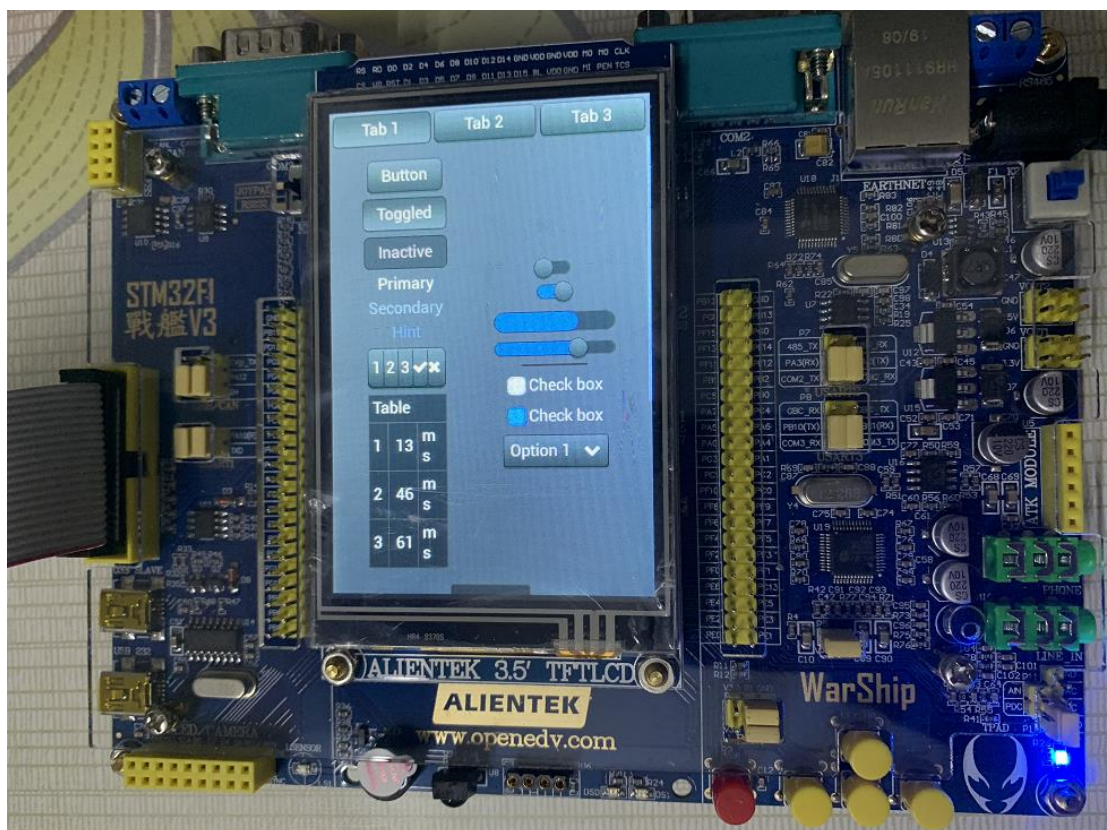
至此 `littleVGL` 的移植到此结束了,祝大家旅途愉快!

3. littleVGL 演示效果

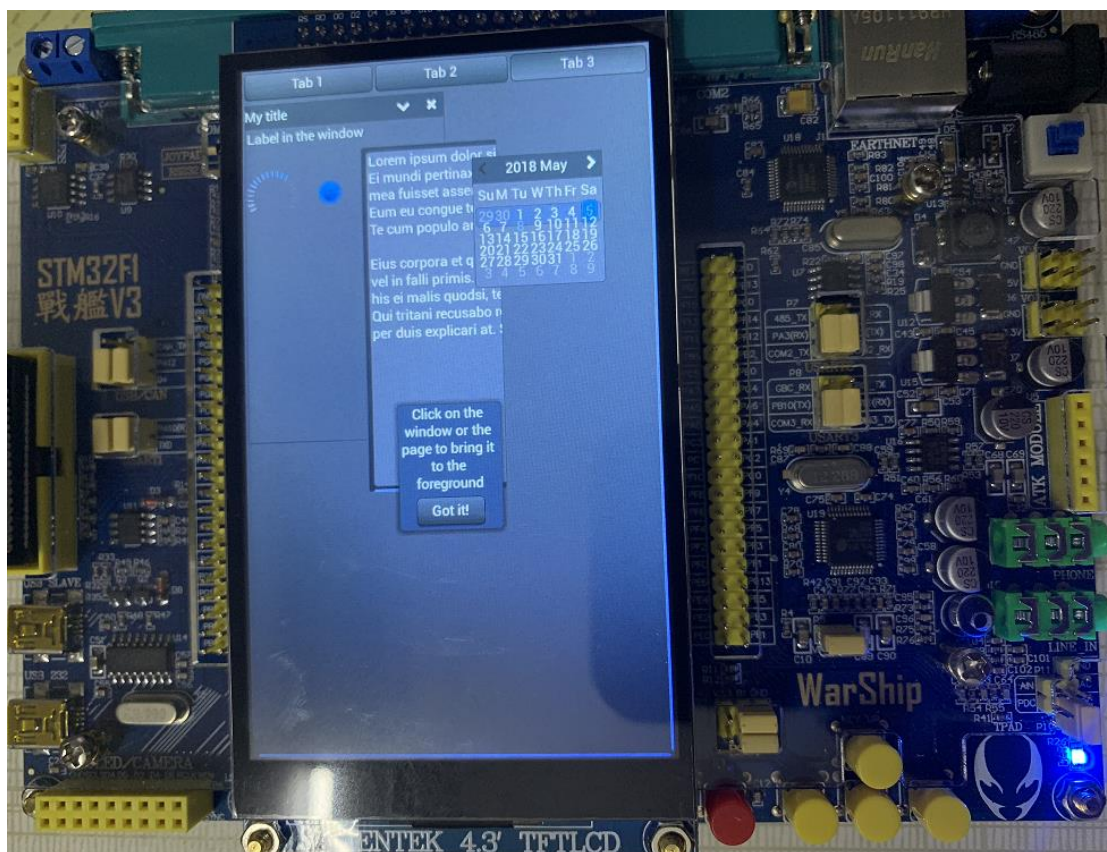
3.1 2.8 寸 (320*240) 电阻屏演示效果



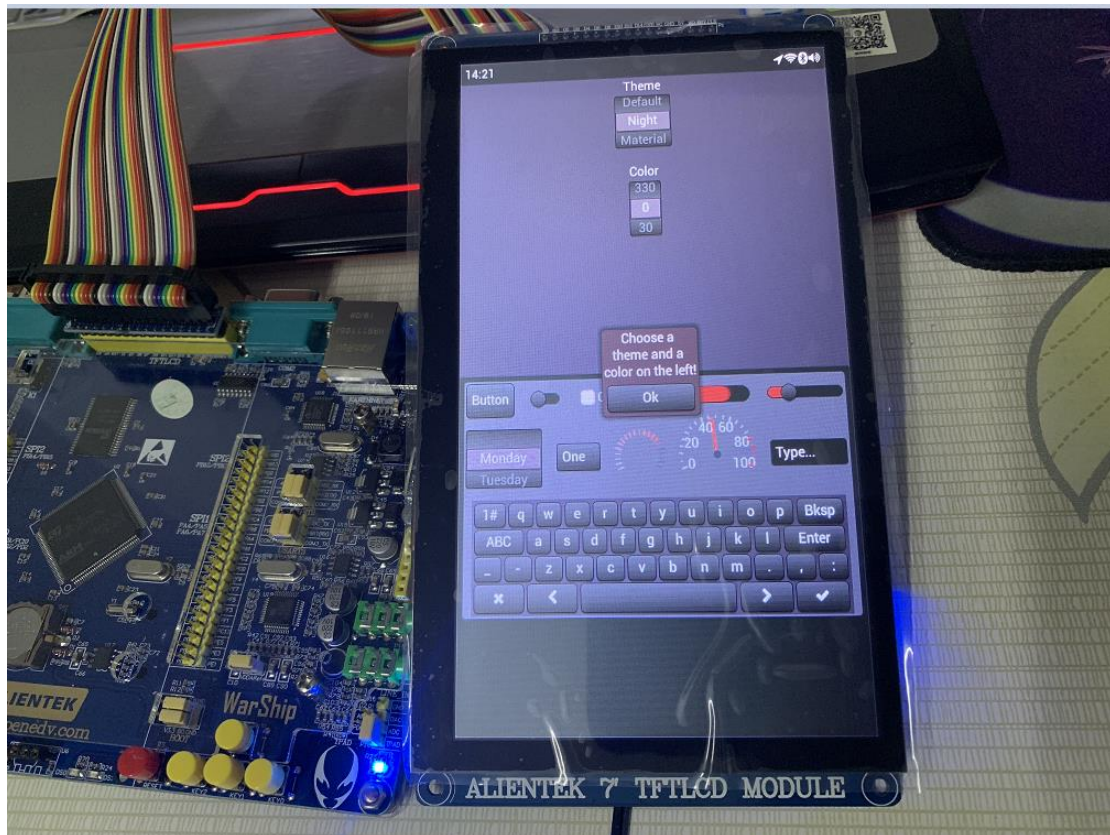
3.2 3.5 寸 (480*320) 电阻屏演示效果



3.3 4.3 寸(800*480) 电容屏



3.4 7 寸(800*480) 电容屏



3.5 总结

从上面的演示效果可以看出, littleVGL 的响应式布局确实很牛呀, 在不同分辨率的屏幕上跑, 界面不会变形, 当然了 littleVGL 支持的屏幕型号远不止上面列举的 4 种, 不同的正点原子开发板对液晶屏型号的支持力度大小是不同的, 而且运行的流畅度也是不同的, 下一节, 我将给大家介绍如果用外部的大容量 sram 给 littleVGL 增加运行流畅度

4. 资料下载

正点原子公司名称：广州市星翼电子科技有限公司

LittleVGL 资料连接：www.openedv.com/thread-309664-1-1.html

原子哥在线教学平台：www.yuanzige.com

正点原子淘宝店铺：<https://openedv.taobao.com>

正点原子官方网站：www.alientek.com

正点原子 B 站视频：<https://space.bilibili.com/394620890>

电话：020-38271790 传真：020-36773971

请下载原子哥 APP，数千讲视频免费学习，更快更流畅。

请关注正点原子公众号，资料发布更新我们会通知。



扫码下载“原子哥”APP



扫码关注正点原子公众号