User Guide

正点原子 littleVGL 开发指南

lv_conf 配置文件详解

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

lv_conf 配置文件详解

对于 littleVGL 而言,它的功能和特性基本都是由 lv_conf.h 文件来配置的,所以 lv_conf.h 文件的重要性就不言而喻了,此文件位于项目根目录的 GUI 目录下,给 lv_conf.h 文件进行合理的配置,可以对我们项目的性能和资源起到很大的作用 ,下面让我们对其配置项逐一讲解.

1.定义最大的液晶屏分辨率

和此功能相关的配置项有 2 个,也就是所谓的 2 个宏定义,分别为: LV_HOR_RES_MAX, LV_VER_RES_MAX

利用这俩个配置项,再加上 LCD_Display_Dir(dir)函数可以实现屏幕的横屏和竖屏

1.1 LV_HOR_RES_MAX 配置项

LV 是 littleVGL 的缩写,HOR 是 horizontal 的缩写,为水平的意思,RES 是 resolution 的缩写,为分辨率的意思,组合起来之后,就很容易猜到这是用于设置液晶屏水平长度的,单位为像素

1.2 LV_VER_RES_MAX 配置项

LV 是 littleVGL 的缩写,VER 是 vertical 的缩写,为垂直的意思,RES 是 resolution 的缩写,为分辨率的意思,组合起来之后,就很容易猜到这是用于设置液晶屏垂直长度的,单位为像素

2.定义颜色深度

和此功能相关的配置项有 2 个,为 LV_COLOR_DEPTH, LV_COLOR_16_SWAP

2.1 LV_COLOR_DEPTH

littleVGL 支持 4 种颜色深度,格式分为为 1 byte per pixel, RGB233, RGB565, ARGB8888,其对应的配置项值分别为 1, 8, 16, 32,在一般的实际项目中,最常用的为 1 和 16,当然如果你的处理器性能和资源都很高的话,那么你可以选择 32,占用 4 个字节,他会给你的项目带来更逼真的颜色效果,保证不失真,如果你选择的是 1,那么它只支持 2 种颜色,占用 1 个字节,在液晶屏上的表现就是显示与不显示的关系,常用于单色屏,比如 lcd12864,oled 等,当你选择的是 16 时,那么他支持 65536 种颜色,占用 2 个字节,显示效果还是很不错的,同时对处理器的要求也不是很高,因此 16 成为了我们实际项目中最常用的设置值.

2.2 LV_COLOR_16_SWAP

这个配置项只有在你选择颜色深度为 16 位时才会生效,在原本颜色值的基础上进行交换高低字节的操作,当你的屏幕有一个 8 位的并行接口时,可能会用到,一般不用理会

3 定义图片显示时的透明色

和此功能相关的配置项只有 1 个,为 LV_COLOR_TRANSP,它的默认值为 LV_COLOR_LIME 宏.而此宏的定义如下:

```
#define LV_COLOR_LIME       LV_COLOR_MAKE(0x00, 0xFF, 0x00)
```

也就是说 LV_COLOR_LIME 为纯绿色, LV_COLOR_TRANSP 宏只有当我们使用到 image 相关的控件时,才会有可能起到作用,注意了,我这里只是说有可能,因为 littleVGL 想要显示带有透明效果的图片,有 2 种方式来实现,分别为 Chrome keying 和 Alpha byte,只有当我们在”Online image converter”图片在线转换工具中选择带有 Chrome keying 的颜色格式时,LV_COLOR_TRANSP 就会起作用,具体作用就是当图片中的颜色值和 LV_COLOR_TRANSP 指定的颜色值相等时,此颜色值像素点就不会被绘制出来,从而达到了透明的效果.

举个例子,假如我们的 LV_COLOR_TRANS 不改动,即默认采用 LV_COLOR_LIME 纯绿色为透明色,当我们要显示  这张带有纯绿色的背景图片时,液晶屏上的实际效果是如下这样的:  ,因为纯绿色的背景没有被绘制,从而实现了透明显示的效果,当然这种方式也是有一定局限性的,因为它实现不了半透明的效果,它只能全透明或者完全不透明

4.是否使能抗锯齿功能

和此功能相关的配置项只有 1 个,为 LV_ANTIALIAS,它的默认值为 1,即默认是使能抗锯齿的,当设置为 0 时就禁止了抗锯齿功能,实际项目中,一般都是使能的,这样我们 UI 界面上的文字,线条,圆弧等元素就看不到毛刺现象了

5.定义屏幕的刷新周期

和此功能相关的配置项只有 1 个,为 LV_DISP_DEF_REFR_PERIOD,它的默认值为 30ms,此值设置的过大的话,就可能会出现卡顿的现象,设置的过小的话,就会有点浪费性能,我们直接采用默认值就可以了,不用过多理会

6.定义界面的缩放比例

和此功能相关的配置项只有 1 个,为 LV_DPI,它的默认值为 100,因为 littleVGL 是支持响应式布局的,通过调节此参数的值,可以很好的适应不同分辨率的布局,表面看起来就跟放大缩小了一样,其实本质是调节了 littleVGL 样式的内边距和外边距等,当此值越大时,控件所占的内边距和外边距值也就越大,导致整个控件所占据的位置也就越大,从而表现出了放大的效果,当此值越小时,表现效果与上面相反.

7.定义 littleVGL 的内存管理方式

和此功能相关的配置项有 8 个,为 LV_MEM_CUSTOM, LV_MEM_SIZE, LV_MEM_ATTR,LV_MEM_ADR,LV_MEM_AUTO_DEFRAG,LV_MEM_CUSTOM_INCLUDE,LV_MEM_CUSTOM_ALLOC, LV_MEM_CUSTOM_FREE

7.1 LV_MEM_CUSTOM

此配置项是用来选择使用 littleVGL 自带的内存管理还是使用自己自定义的内存管理,当其值为 0 时,是选择内部自带的,当其值为 1 时,是选择自定义的方式,默认值为 0 ,为了减少麻烦,我们直接使用内部自带的就可以了.

7.2 LV_MEM_ADR

当 LV_MEM_CUSTOM 为 0 时,此配置项才生效,当 LV_MEM_ADR 为 0 时,代表所管理的内存是直接定义在处理器内部 sram 的,也就是一个内部静态数组,当 LV_MEM_ADR 为其他地址值时,代表所管理的内存是定义在外部扩展的 sram 上, 通过打开 littleVGL 的 lv_mem.c 文件,找到 lv_mem_init 函数,可以看到如下定义:

```
#if LV_MEM_ADR == 0
    /*Allocate a large array to store the dynamically allocated data*/
    static LV_MEM_ATTR MEM_UNIT work_mem_int[LV_MEM_SIZE / sizeof(MEM_UNIT)];
    work_mem = (uint8_t *)work_mem_int;
#else
    work_mem = (uint8_t *)LV_MEM_ADR;
#endif
```

因为所管理的内存池一般不会很大,一般的应用 16KB-32KB 就可以满足了,所以我们一般都把 LV_MEM_ADR 的值设置为 0,即直接定义在处理器内部的 sram 上,这样会比外部的 sram 读写速度快很多

7.3 LV_MEM_SIZE

当 LV_MEM_CUSTOM 为 0 而且 LV_MEM_ADR 也为 0 时,此配置项才生效,用来设置所管理内存的大小,单位为字节,至少大于 2KB,通过打开 littleVGL 的 lv_mem.c 文件,找到 lv_mem_init 函数,可以看到如下定义:

```
static LV_MEM_ATTR MEM_UNIT work_mem_int[LV_MEM_SIZE / sizeof(MEM_UNIT)];
```

work_mem_int 就是函数内部定义的静态数组,也就是所被管理的内存

7.4 LV_MEM_ATTR

当 LV_MEM_CUSTOM 为 0 而且 LV_MEM_ADR 也为 0 时,此配置项才生效,是用来修饰所被管理的内存,通过打开 littleVGL 的 lv_mem.c 文件,找到 lv_mem_init 函数,可以看到如下定义:

```
static LV_MEM_ATTR MEM_UNIT work_mem_int[LV_MEM_SIZE / sizeof(MEM_UNIT)];
```

比如说用来设置字节对齐:

```
#define LV_MEM_ATTR    __align(8)    //以 8 字节对齐
```

7.5 LV_MEM_AUTO_DEFRAG

当 LV_MEM_CUSTOM 为 0 时,此配置项才生效,代表在空闲时候,是否自动进行内存碎片化整理,1 代表使能,0 代表不使能,我们默认使能即可

7.6 LV_MEM_CUSTOM_INCLUDE

当 LV_MEM_CUSTOM 为 1 时,此配置项才生效,用来指明你自己所实现的内存管理所在的头文件,如下所示:

```
#define   LV_MEM_CUSTOM_INCLUDE          <stdlib.h>
```

7.7 LV_MEM_CUSTOM_ALLOC

当 LV_MEM_CUSTOM 为 1 时,此配置项才生效,用来指向你自己所实现的 malloc 函数,如下所示:

```
#define   LV_MEM_CUSTOM_ALLOC      malloc
```

7.8 LV_MEM_CUSTOM_FREE

当 LV_MEM_CUSTOM 为 1 时,此配置项才生效,用来指向你自己所实现的 free 函数,如下所示:

```
#define   LV_MEM_CUSTOM_FREE      free
```

8. 是否使能垃圾回收器

和此功能相关的配置项有 4 个,为 LV_ENABLE_GC, LV_GC_INCLUDE, LV_MEM_CUSTOM_REALLOC, LV_MEM_CUSTOM_GET_SIZE

LV_ENABLE_GC 是总开关,当其值为 1 时,是使能垃圾回收器,一般只有当 littleVGL 绑定到其他的高级语言中时才有作用,此时的 LV_GC_INCLUDE, LV_MEM_CUSTOM_REALLOC, LV_MEM_CUSTOM_GET_SIZE 三个配置项指向相应的头文件和函数,当其值为 0 时,是禁止,我们一般是用不到此功能的,默认禁止就可以了

9.定义输入设备的轮询周期

和此功能相关的配置项有 1 个,为 LV_INDEV_DEF_READ_PERIOD ,其默认值为 30ms,代表每隔 30ms 采样一次输入设备(比如键盘,触摸屏等)的状态

10.定义 Drag 拖拽动作的阈值

和此功能相关的配置项有 1 个,为 LV_INDEV_DEF_DRAG_LIMIT ,其默认值为 10 个像素,代表按住不放拖动 10 个像素以上才认为是一次有效的 Drag 拖拽动作

11.定义 Drag 动作后,控件停下来的速度

和此功能相关的配置项有 1 个,为 LV_INDEV_DEF_DRAG_THROW ,其默认值为 20%,此值越大的话,那么拖动某个控件,待松手后,这个控件就会更快的停下来,类似于汽车紧急刹车后漂移的一种效果

12.定义长按事件的触发时间

和此功能相关的配置项有 1 个,为 LV_INDEV_DEF_LONG_PRESS_TIME ,其默认值为 400ms,代表按住某个控件 400ms 以上时,就会触发这个控件的长按事件

13. 定义长按重复事件的触发时间

和此功能相关的配置项有 1 个,为 LV_INDEV_DEF_LONG_PRESS_REP_TIME ,其默认值为 100ms,代表当某个控件的长按事件触发之后,再继续按住此控件 100ms 以上,就会触发这个控件的长按重复事件

14.是否使能动画特性

和此功能相关的配置项有 1 个,为 LV_USE_ANIMATION ,其默认值为 1,代表使能动画特性,其值为 0 时,代表禁止动画特性

15.是否使能阴影效果

和此功能相关的配置项有 1 个,为 LV_USE_SHADOW ,其默认值为 1,代表使能动画特性,其值为 0 时,代表禁止动画特性

16.是否使能 GROUP 分组功能

和此功能相关的配置项有 1 个,为 LV_USE_GROUP ,其默认值为 1,代表使能分组功能,常用于 keyboard 和 encoder 的导航场景,其值为 0 时,代表禁止

17.是否使用 GPU

和此功能相关的配置项有 1 个,为 LV_USE_GPU ,其默认值为 1,代表使用 GPU 进行加速,那么然后你还要在 lv_port_disp.c 文件中实现 gpu_blend 和 gpu_fill 接口,请根据自己的处理器是否具有 GPU 功能来相应的设置此配置项,我们一般设置为 0,即不使用 GPU 功能

18. 是否使用 littleVGL 自带的文件系统

和此功能相关的配置项有 1 个,为 LV_USE_FILESYSTEM ,当其值为 1 时,代表使用自带的文件系统,那么然后你还要移植 lv_port_fs_template.c 和.h 文件,我们一般设置为 0,即不使用自带的文件系统

19. 是否使用 user_data 子字段

和此功能相关的配置项有 1 个,为 LV_USE_USER_DATA ,当其值为 1 时,代表在 drivers 和 objects(就是控件)描述结构体上增加一个名为 user_data 的子字段,主要是用来携带用户自己的数据,当其值为 0 时,就是不增加此字段,请根据自己项目的实际需求来相应的设置,在 lv_obj.h 中可以找到如下定义:

```
typedef struct _lv_obj_t
{
   struct _lv_obj_t * par;
   //中间省略掉…
#if LV_USE_USER_DATA
      lv_obj_user_data_t user_data;
#endif

} lv_obj_t;
```

20.是否使能图片的调色板功能

和此功能相关的配置项有 1 个,为 LV_IMG_CF_INDEXED ,当其值为 1 时,代表支持图片的调色板方式显示,也就是先把一张图片的所有颜色值存放到一个数组(调色板)中,然后通过索引值来映射出颜色值,好处就是节省图片的存储空间,缺点就是调色板的元素个数不能太大,一般最大为 256 色

21. 是否使能图片的 alpha 透明功能

和此功能相关的配置项有 1 个，为 LV_IMG_CF_ALPHA ，其默认值为 1，代表使能 alpha 透明功能，0 代表不使能

22.设置图片缓存的大小

和此功能相关的配置项有 1 个，为 LV_IMG_CACHE_DEF_SIZE ，此值至少大于等于 1，其默认值为 1，此功能一般用在从外部存储器（如 SD 卡）显示图片的场景下，比如用第三方的 PNG 或者 JPG 库来解析存储器上的.png 或.jpg 图片时

23. 定义 lv_tick_inc 函数的修饰属性

和此功能相关的配置项有 1 个，为 LV_ATTRIBUTE_TICK_INC ，其默认值为空，通过打开 lv_hal_tick.c 文件，可以找到如下定义：

```
LV_ATTRIBUTE_TICK_INC void lv_tick_inc(uint32_t tick_period)
{
    tick_irq_flag = 0;
    sys_time += tick_period;
}
```

24.定义 lv_task_handler 函数的修饰属性

和此功能相关的配置项有 1 个，为 LV_ATTRIBUTE_TASK_HANDLER，其默认值为空，通过打开 lv_task.c 文件，可以找到如下定义：

```
LV_ATTRIBUTE_TASK_HANDLER void lv_task_handler(void)
{
//省略掉…
}
```

25.显式定义内存对齐的大小

和此功能相关的配置项有 1 个，为 LV_ATTRIBUTE_MEM_ALIGN ，其默认值为空，因为当编译器使用-Os 进行优化时，可能会导致我们的数据不是以 4 或者 8 字节对齐的，此时我们可以显式定义进行字节对齐，如下所示：

```
#define LV_ATTRIBUTE_MEM_ALIGN    __attribute__((aligned(4)))
```

26.定义常量关键字

和此功能相关的配置项有 1 个，为 LV_ATTRIBUTE_LARGE_CONST ，其默认值为空，littleVGL 默认自带了 const 常量关键字，如果你发现你的项目中不是用 const 的话，那么你就可以来设置 LV_ATTRIBUTE_LARGE_CONST 的值，如下所示（在 c51 中 code 为常量定义的关键字）：

```
#define LV_ATTRIBUTE_LARGE_CONST  code
```

27.是否使能 log 日志模块

和此功能相关的配置项有 3 个，为 LV_USE_LOG， LV_LOG_LEVEL， LV_LOG_PRINTF，其中 LV_USE_LOG 是总开关，代表是否使能 log 模块，其实内部就是通过 printf 或自实现的 print 函数来实现打印日志信息的，我们一般把 LV_USE_LOG 设置为 0，即不使能

28.是否使能 littleVGL 自带的主题

和此功能相关的配置项有 9 个，为 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

其中 LV_THEME_LIVE_UPDATE 是用来是否允许在运行状态下切换主题风格的，而其他的 8 个配置项表示是否使能其对应的 8 个主题风格

29.是否使能 littleVGL 自带的字体

和此功能相关的配置项有 5 个，为 LV_FONT_ROBOTO_12， LV_FONT_ROBOTO_16， LV_FONT_ROBOTO_22， LV_FONT_ROBOTO_28， LV_FONT_UNSCII_8 分别对应 12 号，16 号，22 号，28 号的 ROBOTO 字体和 8 号的 UNSCII 字体，这每一个字体都包含了 ASCII 和一些图标字体（Symbols），而且全都是 4bpp 抗锯齿的，设置为 1 就代表使能，设置为 0 就代表不使能

30.自定义字体申明

和此功能相关的配置项有 1 个，为 LV_FONT_CUSTOM_DECLARE，默认值为空，当我们自己自定义了多种字体时，就可以用此配置项来向外部进行申明，如下面所示：

```
#define LV_FONT_CUSTOM_DECLARE   LV_FONT_DECLARE(my_font_1) \
                                 LV_FONT_DECLARE(my_font_2) \
                                 LV_FONT_DECLARE(my_font_3)
```

其中 LV_FONT_DECLARE 是 littleVGL 用来申明字体的宏，其定义如下：

```
#define LV_FONT_DECLARE(font_name)      extern lv_font_t font_name;
```

31.定义 littleVGL 默认的字体

和此功能相关的配置项有 1 个,为 LV_FONT_DEFAULT，当某些控件没有显式指定字体时,那么此控件使用的就是默字体,默认字体的定义方式如下:

```
#define LV_FONT_DEFAULT          &lv_font_roboto_16
```

当然了,前提是你得保证 LV_FONT_ROBOTO_16 这个字体已经是使能了的

32.是否使能大容量字体

和此功能相关的配置项有 1 个,为 LV_FONT_FMT_TXT_LARGE，其默认值为 0,即不使能,当我们自定义的某种字体里面包含了特别多的字符时(大于 10000 个),如果显示出错了,那么此时应该使能 LV_FONT_FMT_TXT_LARGE 来避免错误.

33.定义文本的编码方式

和此功能相关的配置项有 1 个,为 LV_TXT_ENC，此配置项有 2 个可选值,分别为 LV_TXT_ENC_UTF8(默认值)和 LV_TXT_ENC_ASCII ,当我们要显示中文或者其他非 ascii 码字符时,那么我们一定得设置成 LV_TXT_ENC_UTF8,同时还得保证代码编辑器也是相应的 UTF8 编码

34.是否使能 lv_obj_realaign 函数

和此功能相关的配置项有 1 个,为 LV_USE_OBJ_REALIGN，其默认值为 1，即使能 lv_obj_realaign 函数,设置为 0 的话,就是不使能

35.是否使能 littleVGL 的某个控件

介绍到这里时,lv_conf.h 文件基本介绍完了,后面剩下的所有配置项基本都是一样的功能,都是用来是否使能某个控件的,举个例子:

#define LV_USE_BTN 1

意思就是使能按钮控件,剩下的其他配置项我就不介绍了,都是一样的道理

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