User Guide

正点原子 littleVGL 开发指南

使用外部 SARM 进行加速

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

使用外部 sram 进行加速

1.介绍

在我们的 littleVGL 移植章节中,为了减少移植的难度,我们是采用处理器内部的 sram 给 littleVGL 的显示缓冲区分配了一个 10*LV_HOR_RES_MAX 小空间,而在这一节中,我们将通过使用外部的大容量 sram 来给 GUI 应用增加流畅度,当然前提是你的开发板已经配备了外部 sram 模块,为了能让 littleVGL GUI 增加运行流畅度,其影响因素还是很多的,我这里主要列出如下 4 个主要的原因:

1.使用性能更好的处理器
2.使用 GPU 或者 DMA 进行 littleVGL 显示缓冲区的拷贝操作
3.尽量使用处理器内部的 sram
4.给 littleVGL 的显示缓冲区分配更大的空间,最好全屏大小的空间

在条件有限的情况下,第 1 点和第 2 点我们就不考虑了,我们都知道处理器内部的 sram 访问速度很快,但是容量却少的可怜,当然了如果你的处理器内部 sram 容量很大的话,那么你可以直接用内部 sram 给 littleVGL 分配一个全屏大小的显示缓冲区,就没必要再去使用外部 sram 了,虽然外部 sram 访问速度比内部 sram 慢,但是外部 sram 可以给 littleVGL 分配更大以至全屏的显示缓冲区,在这俩者的 PK 权衡之下,实验证明,使用外部 sram 是可以增加 littleVGL 的运行流畅度的.下面开始详细描述如何去实现这一功能

2.软件设计

本章节的代码改动不大,所以可以在移植那一节的代码基础上进行稍微修改.

2.1 添加外部 sram 驱动

我们可以直接从正点原子相应开发板上的”外部 SRAM 实验”中,把 sram 驱动程序直接拷贝到我们项目的 HARDWARE 目录下,如下图所示:

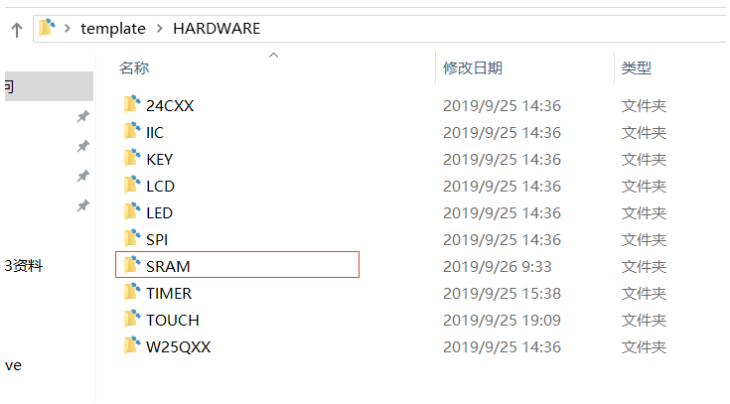

图 2.1.1 SRAM 驱动文件的位置

然后打开本项目的 Keil 工程,并在 HARDWARE 分组下添加 sram 的驱动文件,如下图所示:

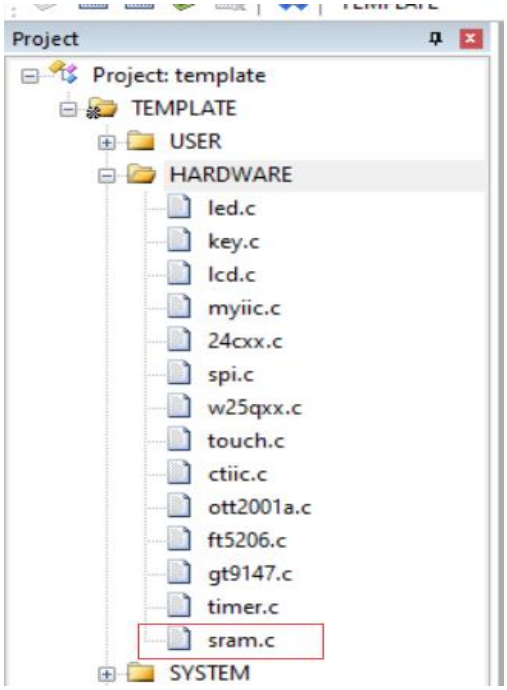

图 2.1.2 分组下添加 sram 的驱动程序

接着也需要注意在 Includes Paths 中添加 sram 驱动的头文件路劲,如下图所示:

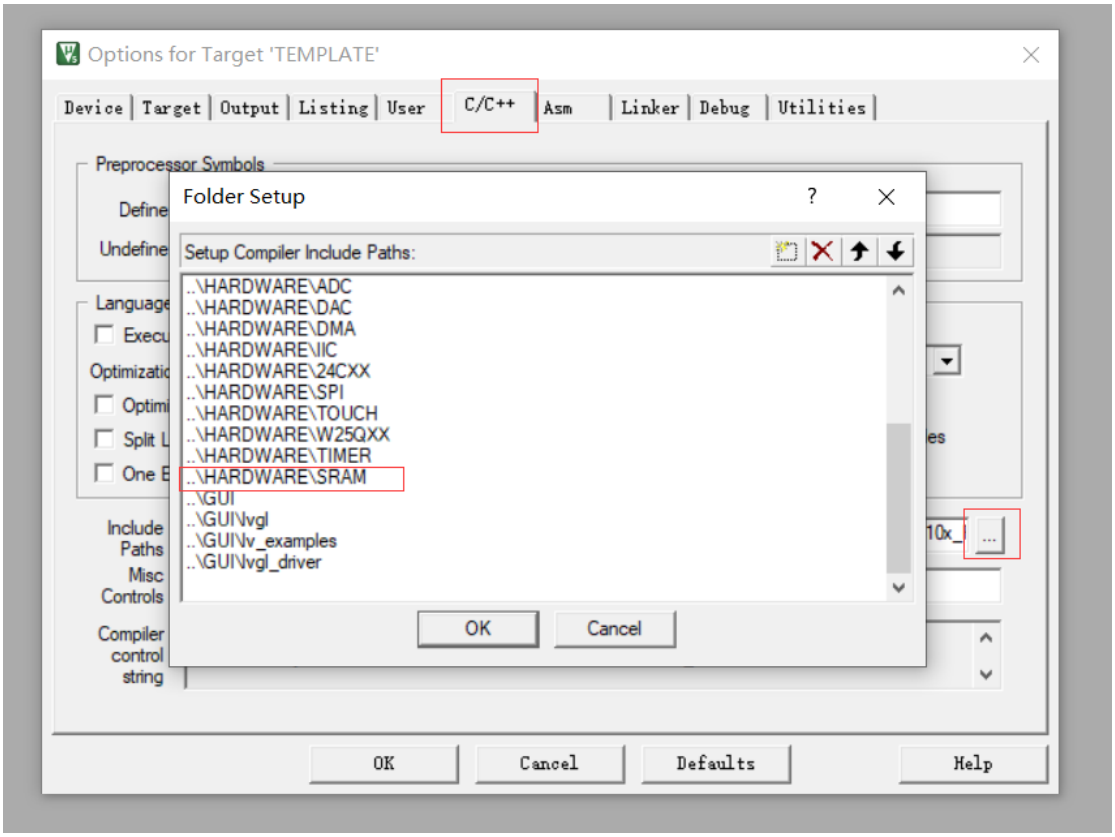

图 2.1.3 添加 sram 驱动的头文件路径

接着再到 main 函数中调用外部 sram 的初始化代码,如下图所示:

```
int main(void)
{
  delay_init();                    //延时函数初始化
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);//设置中断优先级
  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1m
  FSMC_SRAM_Init();                //外部1MB的sram初始化
  LCD_Init();                      //LCD初始化
  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
  lv_port_indev_init();            //lvgl输入接口初始化
```

图 2.1.4 main 函数中调用外部 sram 的初始化代码

注:以后像这种添加硬件驱动的简单操作,我后面的章节中就不再详细描述

2.2 修改 lv_port_disp.c 文件

我们在 keil 中打开 lv_port_disp.c 文件,在这个文件中,我们修改的内容很少,主要就是更换显示缓存区的定义方式,之前是在内部 sram 中分配的,以前的代码是下面这样的:

```
void lv_port_disp_init(void)
{
    …
static lv_color_t buf1_1[LV_HOR_RES_MAX * 10];
lv_disp_buf_init(&disp_buf_1, buf1_1, NULL, LV_HOR_RES_MAX * 10);
    …
}
```

而现在我们需要把改成如下这样的代码:

```
//全屏的大小
#define COLOR_BUF_SIZE          (LV_HOR_RES_MAX*LV_VER_RES_MAX)
//分配到外部
static lv_color_t color_buf[COLOR_BUF_SIZE]     __attribute__((at(0X68000000)));

void lv_port_disp_init(void)
{
    …
lv_disp_buf_init(&disp_buf, color_buf, NULL, COLOR_BUF_SIZE);
…
}
```

其中为了见名知意,显示缓存区的名字被我从 buf1_1 改为了 color_buf,,上面的 0X68000000 是外部 sram 的地址,这个请你根据你的项目来实际调整,,只要保证外部 sram 剩余的空间大小大于 COLOR_BUF_SIZE*2 个字节就行了,笔者为强迫症患者,所以还删除掉了 lv_port_disp.c 文件中一些无用的函数,比如 disp_init 函数就是没有用的,最后我给出 lv_port_disp.c 文件改完后的一个完整代码:

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


//变量定义
//全屏的大小
#define COLOR_BUF_SIZE          (LV_HOR_RES_MAX*LV_VER_RES_MAX)
//分配到外部 1MB sram 的最起始处
static lv_color_t color_buf[COLOR_BUF_SIZE]     __attribute__((at(0X68000000)));

//函数申明
static void disp_flush(lv_disp_drv_t * disp_drv, const lv_area_t * area, lv_color_t * color_p);
#if 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;

        //显示缓冲区初始化
        lv_disp_buf_init(&disp_buf, color_buf, NULL, COLOR_BUF_SIZE);

        //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,我这里是为了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

        //使用透明度混合俩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
```

按照上述改完后,直接编译,应该是无错误无警告的,把代码下载到开发板之后,可以对比发现会比”移植章节” 中的演示效果流畅很多.

3.littleVGL 堆的内存分配

littleVGL 的内存消耗主要体现在 2 个方面,第一个是显示缓冲区,第二个就是我们这里所要讲到的堆,而 littleVGL 堆的内存分配也是有 2 种方式,如下所示:

1)采用内部的 sram,原理定义一个静态的局部数组

2)和显示缓冲区一样,采用外部的大容量 sram

我们一般情况下都是使用第一种方式的,即采用内部的 sram,因为内部的 sram 速度快,而且 littleVGL 堆的内存消耗一般都不是很大的,对于市面上的大部分微处理器来说,是完全可以满足这个性能的,那什么时候我们去使用第二种方式呢?那就是当你的微处理器的内部 sram 真的是特别少的时候,当然了咯,前提是你的硬件已经具备了外部 sram,如果想要采用第二种方式,针对于软件层面来说是非常简单的,只需要把 lv_conf.h 配置文件中的 LV_MEM_ADR 宏指向我们的外部 sram 地址就行了,其他的什么都不需要改动,具体示例如下:

```
#define LV_MEM_SIZE        (16U * 1024U)   //此时是指占用外部 sram 的大小
#define LV_MEM_ADR         (0X68000000+LV_HOR_RES_MAX*LV_VER_RES_MAX*2)
```

至于 LV_MEM_ADR 的值请根据自己项目的实际情况来改动

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