VM3300 软件源程序

```
#include "Speed.h"

/**
 * @brief 编码器初始化
 *
 */
void Encoder_Init(void)
{
    HAL_TIM_Base_Start_IT(&htim17);
    HAL_TIM_IC_Start_IT(&htim17, TIM_CHANNEL_1); // 开启 time17 通道 1 输入捕获
}

/**
 * @brief 检测速度是否停止-0.05s
 *
 * @param dT 任务周期
 */
void Check_Speed(float dT)
{
    Speed.Stop_Cnt += dT;          // 每 50ms 进入
    if (Speed.Stop_Cnt >= 1.0) // 0.5s 发现没出发输入捕获
    {
        Speed.Rel = 0;          // 将速度清零
        Speed.Stop_Cnt = 0; // 计数清零
    }
}

uint32_t capture, capture1, capture2;
float rel1;
/**
 * @brief Tim17 通道 1 的输入捕获回调函数
 *
 */
void TIM17CaptureChannel1Callback(void)
{
    capture1 = __HAL_TIM_GET_COMPARE(&htim17, TIM_CHANNEL_1); // 获取 Tim14
通道 1 的输入捕获
    if (capture1 > capture2)
        capture = capture1 - capture2;
    else
        capture = capture1 + (0xFFFF - capture2);
    rel1 = 10000.0f / (float)capture; // 计算速度
    capture2 = capture1;
    Speed.Rel = rel1 * 60 / 5; // 将速度赋值给 L1 的实际速度
    Speed.Stop_Cnt = 0;
}

/**
 * @brief TIM_IC 回调函数
 *
```

```
 * @param htim
 */
void Speed_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
	if (htim->Instance == TIM17)
	{
		if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
		{
			TIM17CaptureChannel1Callback();
		}
	}
}
#include "Interrupt.h"

/**
 * @brief  定时器计数回调函数
 *
 * @param htim
 */
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
	Infrared_TIM_Interrupt(htim);//红外发送定时计数中断函数
}

/**
 * @brief  定时器捕获回调函数
 *
 * @param htim
 */
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
	Infrared_IC_CaptureCallback(htim);
	Speed_IC_CaptureCallback(htim);
}

/**
 * @brief PWM 信号传输完成回调函数，该函数非常之重要
 *
 * @param htim
 */
void HAL_TIM_PWM_PulseFinishedCallback(TIM_HandleTypeDef *htim)
{
	WS2812B_PulseFinishedCallback(htim);
}
#include "Param.h"

/**********结构体**********/
struct _Save_Param_ Param; //  原始数据

/**********全局变量声明******/
```

```
uint8_t Save_Param_En; //  保存标志位

/**
 * @brief  初始化硬件中的参数
 *
 */
void Param_Reset(void)
{
    Param.Flash_Check_Start = FLASH_CHECK_START;

    Param.Speed = 100;

    Param.Flash_Check_End = FLASH_CHECK_END;
}

/**
 * @brief  保存硬件中的参数
 *
 */
void Param_Save(void)
{
    Flash_Write((uint8_t *)(&Param), sizeof(Param));
}

/**
 * @brief  读取硬件中的参数，判断是否更新
 *
 */
void Param_Read(void)
{
    Flash_Read((uint8_t *)(&Param), sizeof(Param));

    // 板子从未初始化
    if (Param.Flash_Check_Start != FLASH_CHECK_START || Param.Flash_Check_End != FLASH_CHECK_END)
    {
        Param_Reset();
        Speed.Set = Param.Speed;// 将 Flash 中的速度赋值
        SetOK_Flag = 1;
        Save_Param_En = 1;
    }
    else
    {
        Speed.Set = Param.Speed;// 将 Flash 中的速度赋值
        SetOK_Flag = 1;
    }

    // 保存参数
    if (Save_Param_En)
    {
```

```
            Save_Param_En = 0;
            Param_Save();
        }
}

/**
 * @brief 保存标志位置 1，0.5s 后保存
 *
 * @param dT 任务周期
 */
void Param_Save_Overtime(float dT)
{
    static float time;

    if (Save_Param_En)
    {
        time += dT;

        if (time >= 0.5f)
        {
            Param_Save();
            Save_Param_En = 0;
        }
    }
    else
        time = 0;
}
#include "PID.h"

/**
 * @brief 微分先行 PID 计算
 *
 * @param dT 周期（单位：秒）
 * @param Expect 期望值（设定值）
 * @param Freedback 反馈值
 * @param PID_Arg PID 参数结构体
 * @param PID_Val PID 数据结构体
 * @param Error_Lim 误差限幅
 * @param Integral_Lim 积分误差限幅
 */
void AltPID_Calculation(float dT, float Expect, float Freedback, _PID_Arg_ *PID_Arg, _PID_Val_ *PID_Val, float Error_Lim, float Integral_Lim)
{
    PID_Val->Error = Expect - Freedback; // 误差 = 期望值-反馈值

    PID_Val->Proportion = PID_Arg->Kp * PID_Val->Error; // 比例 = 比例系数*误差
    PID_Val->Fb_Differential = -PID_Arg->Kd * ((Freedback - PID_Val->Freedback_Old) * safe_div(1.0f, dT, 0)); // 微分 = -（微分系数） *（当前反馈值-上一次反馈值） *频率
    PID_Val->Integral += PID_Arg->Ki * LIMIT(PID_Val->Error, -Error_Lim, Error_Lim) * dT;
```

```
// 积分 = 积分系数*误差*周期
    PID_Val->Integral = LIMIT(PID_Val->Integral, -Integral_Lim, Integral_Lim);
// 积分限幅

    PID_Val->Out = PID_Val->Proportion + PID_Val->Integral + PID_Val->Fb_Differential; //
PID 输出

    PID_Val->Freedback_Old = Freedback; // 将当前反馈值赋值给上一次反馈值
}

/**
 * @brief 增量式 PID 计算
 *
 * @param dT dT：周期（单位：秒）
 * @param Expect 期望值（设定值）
 * @param Freedback 反馈值
 * @param PID_Arg PID 参数结构体
 * @param PID_Val PID 数据结构体
 * @param Integral_Lim 积分误差限幅
 */
void IncPID_Calculation(float dT, float Expect, float Freedback, _PID_Arg_ *PID_Arg,
_PID_Val_ *PID_Val, float Integral_Lim)
{
    PID_Val->Error = Expect - Freedback; // 误差 = 期望值-反馈值

    PID_Val->Proportion = PID_Arg->Kp * (PID_Val->Error - PID_Val->Error_Last);
// 比例 = 比例系数*（当前误差-上一次误差）
    PID_Val->Integral = PID_Arg->Ki * PID_Val->Error * dT;
// 积分 = 积分系数*误差*周期
    PID_Val->Integral = LIMIT(PID_Val->Integral, -Integral_Lim, Integral_Lim);
// 积分限幅
    PID_Val->Differential = PID_Arg->Kd * (PID_Val->Error - 2.0f * PID_Val->Error_Last +
PID_Val->Error_Previous) * safe_div(1.0f, dT, 0); // 微分 = 微分系数 *（当前误差-2*上一次
误差+上上次误差）*频率

    PID_Val->Out += PID_Val->Proportion + PID_Val->Integral + PID_Val->Differential; //
PID 输出

    PID_Val->Error_Previous = PID_Val->Error_Last; // 将上一次误差赋值给上上次误差
    PID_Val->Error_Last = PID_Val->Error;          // 将当前误差赋值给上一次误差
}
#include "SetVal.h"

/**********全局变量声明******/
uint8_t SetOK_Flag; // 检测是否按下按键

/**
 * @brief 检测设置
 *
 * @param dT 任务周期
```

```
 */
void Check_Set(float dT)
{
    if (SetOK_Flag)
    {
        if (Speed.Ctrl != Speed.Set)
        {
            Speed.Ctrl = Speed.Set;
            Param.Speed = Speed.Set;
            if (Speed.ADDMode != 0)
                Speed.ADDMode = 0;
        }
        Save_Param_En = 1; //  保存
        SetOK_Flag = 0;
    }
}
#include "Show.h"

/**
 * @brief  速度显示处理
 *
 * @param dT  任务周期
 */
void Deal_Speed(float dT)
{
    if (sys.Run_Status == 1 || Speed.Rel > 10)
    {
        if (Speed.ADDMode == 0) //  在电机控制中，速度未处理
        {
            Speed.Display = Speed.Rel;
            if (Speed.Ctrl >= Speed.Display) //  控制速度大于实际速度
            {
                Speed.ADDMode = 1; //  进入加速模式下
            }
            else if (Speed.Ctrl < Speed.Display) //  控制速度小于实际速度
            {
                Speed.ADDMode = 2; //  进入减速模式下
            }
        }
        if (Speed.ADDMode == 1) //  在进入加速模式下
        {
            if (Speed.Rel > Speed.Display) //  当前速度大于显示速度
            {
                if (Speed.Display < Speed.Rel)
                    Speed.Display += 1; //  显示当前速度
            }
            else //  当前速度小于上一次速度
            {
                Speed.Display = Speed.Display; //  显示上一次速度，不让速度小于当前速
度。呈现攀升速度的现象
```

```
                }
                if (Speed.Display >= Speed.Ctrl) //  实际速度大于等于控制速度
                {
                    Speed.ADDMode = 3; //  进入稳定模式
                    return;
                }
            }
            if (Speed.ADDMode == 2) //  速度下降模式下
            {
                if (Speed.Rel < Speed.Display) //  当前速度小于上一次速度
                {
                    if (Speed.Display > Speed.Rel)
                        Speed.Display -= 1; //  显示当前速度
                }
                else //  当前速度大于上一次速度
                {
                    Speed.Display = Speed.Display; //  显示上一次速度，不让速度大于当前速度。呈现下降速度的现象
                }

                if (Speed.Display <= Speed.Ctrl) //  实际速度小于等于控制速度
                {
                    Speed.ADDMode = 3; //  进入稳定模式
                    return;
                }
            }
            else if (Speed.ADDMode == 3) //  速度稳定模式下
            {
                Speed.Display = Speed.Ctrl; //  显示控制速度
            }
        }
        else
        {
            Speed.Display = Speed.Set; //  显示设定转速
            Speed.ADDMode = 0;
        }
}

/**
 * @brief  显示屏幕内容
 *
 */
void Show_Display(float dT)
{
    Deal_Speed(dT);
    DisplayNumber_4BitDig(Speed.Display);

}
#include "Drv_WS2812.h"
```

```
/**********结构体***********/
_WS2812_RGB_ WS2812[2];
_WS2812_ WS2812_Status[2];

/**********局部变量声明******/
uint16_t WS2812_Buf1[RESET_WORD+RGB_NUM1 * 24+DUMMY_WORD]; // 存放的数组
uint16_t WS2812_Buf2[RESET_WORD+RGB_NUM2 * 24+DUMMY_WORD]; // 存放的数组

uint16_t Memaddr = 0;                    // 数组的地址值

/**
 * @brief WS2812 初始化
 *
 */
void WS2812_Init(void)
{
    WS2812[0].Tim = &WS2812_Tim;          // 定时器选择
    WS2812[0].CHANNEL = TIM_CHANNEL_4; // 定时器通道数
    WS2812[1].Tim = &WS2812_Tim;          // 定时器选择
    WS2812[1].CHANNEL = TIM_CHANNEL_1; // 定时器通道数
}

/**
 * @brief 更新目前灯的状态
 *
 */
void WS2812_Update(void)
{
    if (WS2812_Status[0].Update)
    {
        HAL_TIM_PWM_Start_DMA(WS2812[0].Tim,    WS2812[0].CHANNEL,(uint32_t
*)&WS2812_Buf1[0], RESET_WORD+RGB_NUM1 * 24+DUMMY_WORD); // 开始传输字
节数据

        WS2812_Status[0].Sending = 1; // 发送
        WS2812_Status[0].Update = 0;  // 更新完成，标志位清零
    }

    if (WS2812_Status[1].Update)
    {
        HAL_TIM_PWM_Start_DMA(WS2812[1].Tim,    WS2812[1].CHANNEL,(uint32_t
*)&WS2812_Buf2[0], RESET_WORD+RGB_NUM2 * 24+DUMMY_WORD); // 开始传输字
节数据

        WS2812_Status[1].Sending = 1; // 发送
        WS2812_Status[1].Update = 0;  // 更新完成，标志位清零
    }
}

/**
```

```
 * @brief  设置第几个灯的颜色
 *
 * @param num  要设置的第几个灯
 * @param r  红色的色值
 * @param g  绿色的色值
 * @param b  蓝色的色值
 */
void WS2812_SetColor(uint8_t num, uint8_t r, uint8_t g, uint8_t b)
{
    uint8_t cnt = 0;

    WS2812_Status[0].Number = num; // 灯的序号

    WS2812[0].Red_Out = WS2812[0].Red = r; // 颜色赋值
    WS2812[0].Green_Out = WS2812[0].Green = g;
    WS2812[0].Blue_Out = WS2812[0].Blue = b;

    WS2812[0].Red_Out = (WS2812[0].Red_Out + 1) / 16 * MAX_RATE * 10;          // 乘等于最大亮度值
    WS2812[0].Green_Out = (WS2812[0].Green_Out + 1) / 16 * MAX_RATE * 10; // 乘等于最大亮度值
    WS2812[0].Blue_Out = (WS2812[0].Blue_Out + 1) / 16 * MAX_RATE * 10;  // 乘等于最大亮度值

    if (WS2812_Status[0].Number == 0xFF) // 同时改变所有灯的颜色
    {
        Memaddr = 0;          // 从头赋值
        while (cnt < RGB_NUM1) // 一个一个发
        {
            for (uint8_t i = 0; i < 8; i++)
                WS2812_Buf1[Memaddr++] = ((WS2812[0].Green_Out << i) & 0x80) ? TIMING_ONE : TIMING_ZERO; // 将设定值拆分放入数组中
            for (uint8_t i = 0; i < 8; i++)
                WS2812_Buf1[Memaddr++] = ((WS2812[0].Red_Out << i) & 0x80) ? TIMING_ONE : TIMING_ZERO;
            for (uint8_t i = 0; i < 8; i++)
                WS2812_Buf1[Memaddr++] = ((WS2812[0].Blue_Out << i) & 0x80) ? TIMING_ONE : TIMING_ZERO;
            cnt++;
        }
    }
    else if (WS2812_Status[0].Number <= RGB_NUM1) // 只改变指定序号灯的颜色，看RGB_NUM 的值
    {
        Memaddr = WS2812_Status[0].Number * 24;
        for (uint8_t i = 0; i < 8; i++)
            WS2812_Buf1[Memaddr++] = ((WS2812[0].Green_Out << i) & 0x80) ? TIMING_ONE : TIMING_ZERO;
        for (uint8_t i = 0; i < 8; i++)
            WS2812_Buf1[Memaddr++] = ((WS2812[0].Red_Out << i) & 0x80) ?
```

```
TIMING_ONE : TIMING_ZERO;
            for (uint8_t i = 0; i < 8; i++)
                WS2812_Buf1[Memaddr++] = ((WS2812[0].Blue_Out << i) & 0x80) ? TIMING_ONE : TIMING_ZERO;
        }

        WS2812_Status[0].Update = 1; // 更新显示
}

/**
 * @brief 设置第几个灯的颜色
 *
 * @param num 要设置的第几个灯
 * @param r 红色的色值
 * @param g 绿色的色值
 * @param b 蓝色的色值
 */
void WS28121_SetColor(uint8_t num, uint8_t r, uint8_t g, uint8_t b)
{
        uint8_t cnt = 0;

        WS2812_Status[1].Number = num; // 灯的序号

        WS2812[1].Red_Out = WS2812[1].Red = r; // 颜色赋值
        WS2812[1].Green_Out = WS2812[1].Green = g;
        WS2812[1].Blue_Out = WS2812[1].Blue = b;

        WS2812[1].Red_Out = (WS2812[1].Red_Out + 1) / 16 * MAX_RATE * 10;          // 乘等于最大亮度值
        WS2812[1].Green_Out = (WS2812[1].Green_Out + 1) / 16 * MAX_RATE * 10; // 乘等于最大亮度值
        WS2812[1].Blue_Out = (WS2812[1].Blue_Out + 1) / 16 * MAX_RATE * 10;  // 乘等于最大亮度值

        if (WS2812_Status[1].Number == 0xFF) // 同时改变所有灯的颜色
        {
            Memaddr = 0;            // 从头赋值
            while (cnt < RGB_NUM2) // 一个一个发
            {
                for (uint8_t i = 0; i < 8; i++)
                    WS2812_Buf2[Memaddr++] = ((WS2812[1].Green_Out << i) & 0x80) ? TIMING_ONE : TIMING_ZERO; // 将设定值拆分放入数组中
                for (uint8_t i = 0; i < 8; i++)
                    WS2812_Buf2[Memaddr++] = ((WS2812[1].Red_Out << i) & 0x80) ? TIMING_ONE : TIMING_ZERO;
                for (uint8_t i = 0; i < 8; i++)
                    WS2812_Buf2[Memaddr++] = ((WS2812[1].Blue_Out << i) & 0x80) ? TIMING_ONE : TIMING_ZERO;
                cnt++;
            }
```

```
    }
    else if (WS2812_Status[1].Number <= RGB_NUM2) // 只改变指定序号灯的颜色，看RGB_NUM 的值
    {
        Memaddr = WS2812_Status[1].Number * 24;
        for (uint8_t i = 0; i < 8; i++)
            WS2812_Buf2[Memaddr++] = ((WS2812[1].Green_Out << i) & 0x80) ? TIMING_ONE : TIMING_ZERO;
        for (uint8_t i = 0; i < 8; i++)
            WS2812_Buf2[Memaddr++] = ((WS2812[1].Red_Out << i) & 0x80) ? TIMING_ONE : TIMING_ZERO;
        for (uint8_t i = 0; i < 8; i++)
            WS2812_Buf2[Memaddr++] = ((WS2812[1].Blue_Out << i) & 0x80) ? TIMING_ONE : TIMING_ZERO;
    }

    WS2812_Status[1].Update = 1; // 更新显示
}

/**
 * @brief PWM 信号传输完成回调函数，该函数非常之重要
 *
 * @param htim
 */
void WS2812B_PulseFinishedCallback(TIM_HandleTypeDef *htim)
{
    if (htim->Instance == TIM1)
    {
        if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_4)
        {
            WS2812_Status[0].Sending = 0;                                   // 发送完成，标志位清零
            HAL_TIM_PWM_Stop_DMA(WS2812[0].Tim, WS2812[0].CHANNEL);         // 关闭 DMA 传输
        }

        if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
        {
            WS2812_Status[1].Sending = 0;                                   // 发送完成，标志位清零
            HAL_TIM_PWM_Stop_DMA(WS2812[1].Tim, WS2812[1].CHANNEL);         // 关闭 DMA 传输
        }
    }

}
/*
*********************************************************************
 * 函数原型：static void WS2812_Breath(float dT, uint8_t num, uint8_t f, uint8_t r, uint8_t g, uint8_t b)
```

```
 * 功      能：呼吸灯
 * 输      入：num：第几个 RGB 灯，f：是否点亮，r、g、b：表示颜色的值
 * 参      数：float dT, uint8_t num, uint8_t f, uint8_t r, uint8_t g, uint8_t b
 * 调      用：内部调用
**********************************************************************
*/
static void WS2812_Breath(float dT, uint8_t num, uint8_t r, uint8_t g, uint8_t b)
{
    static float T;
    float Run_Time;

    T += dT;
    if(T >= (float)(BREATH_TIME / 1000)) T = 0;
    Run_Time = (T * 1000) / BREATH_TIME * 6.28f;
    Run_Time = (cosf(Run_Time) + 1) * 0.5;

    WS2812_SetColor(num, r * Run_Time,g * Run_Time,b * Run_Time);
}

/*
**********************************************************************
 * 函数原型：static void WS2812_Colorful(float dT, uint8_t num, uint8_t f)
 * 功      能：七彩变幻
 * 输      入：num：第几个 RGB 灯，f：是否点亮
 * 参      数：float dT, uint8_t num, uint8_t f
 * 调      用：内部调用
**********************************************************************
*/
static void WS2812_Colorful(float dT, uint8_t num)
{
    static float T;
    float Run_Time;
    uint8_t Red,Green,Blue;
    T += dT;
    if(T >= (float)(COLORFUL_TIME / 1000)) T = 0;
    Run_Time = (T * 1000) / COLORFUL_TIME * 6.28f;

    Red = ((cosf(Run_Time - 1.04f) + 1) * 0.5f) * 0xFF;
    Green = ((cosf(Run_Time + 0.00f) + 1) * 0.5f) * 0xFF;
    Blue = ((cosf(Run_Time + 1.04f) + 1) * 0.5f) * 0xFF;

    WS2812_SetColor(num, Red,Green,Blue);

}

/**
 * @brief WS2812 显示任务
 *
 * @param dT  任务周期
 */
```

```
void WS2812_Duty(float dT)
{
    static uint16_t time = 0;
    static uint8_t led_check = 1;
    static uint32_t time1 = 0;
//  if (Save_Param_En)
//      return;
    uint8_t val1,val2,val3,val4,val5,val6,val7,val8,val9,val10;
    if (led_check) //开机灯光自检
    {
        time += 20;
        uint8_t step = (time / 500) % 4;
        if (step == 0)
            WS2812_SetColor(0xFF, 0xFF, 0, 0);
        else if (step == 1)
            WS2812_SetColor(0xFF, 0, 0xFF, 0);
        else
            WS2812_SetColor(0xFF, 0, 0, 0), led_check = 0, WS2812_Status[0].Mode = 1;
    }
    else
    {
        switch (WS2812_Status[0].Mode)
        {
        case 1:
//          time1 += 20;
//          uint8_t step1 = (time1 / 25) % 12;
//          WS2812_Colorful(dT,step1);
//          WS2812_Colorful(dT,24-step1);

//          time1 += 20;
//          uint8_t step1 = (time1 / 25) % 24;
//          WS2812_Colorful(dT,step1);


//          time1 += 20;
//          uint8_t step1 = (time1 / 25) % (RGB_NUM1/2);
//          if(time1 > 25*(RGB_NUM1/2))
//              time1 = 0;
//          WS2812_Colorful(dT,step1);
//          WS2812_Colorful(dT,12+step1);

//      WS2812_SetColor(0xFF, 0, 0, 0);
        /**************************************/
            if (Infrared[0].Someone || Infrared[1].Someone)
            {
                WS2812_SetColor(0xFF, 0, 0, 0);
                time1 += 20;
                uint8_t step1 = (time1 / 20) % 24;
                val1 = step1;
                if(val1 > 23) val1 = 0;
```

```
            WS2812_SetColor(val1, 0xFF, 0xFF, 0xFF);
            val2 = val1+1;
            if(val2 > 23) val2 = 0;
            WS2812_SetColor(val2, 0xFF, 0xFF, 0xFF);
            val3 = val2+1;
            if(val3 > 23) val3 = 0;
            WS2812_SetColor(val3, 0xFF, 0xFF, 0xFF);
            val4 = val3 + 1;
            if(val4 > 23) val4 = 0;
            WS2812_SetColor(val4, 0xFF, 0xFF, 0xFF);
            val5 = val4 + 1;
            if(val5 > 23) val5 = 0;
            WS2812_SetColor(val5, 0xFF, 0xFF, 0xFF);
            val6 = val5 + 1;
            if(val6 > 23) val6 = 0;
            WS2812_SetColor(val6, 0xFF, 0xFF, 0xFF);

            val7 = val6 + 1;
            if(val7 > 23) val7 = 0;
            WS2812_SetColor(val7, 0xFF, 0xFF, 0xFF);
            val8= val7 + 1;
            if(val8 > 23) val8 = 0;
            WS2812_SetColor(val8, 0xFF, 0xFF, 0xFF);
            val9 = val8 + 1;
            if(val9 > 23) val9 = 0;
            WS2812_SetColor(val9, 0xFF, 0xFF, 0xFF);
            val10= val9 + 1;
            if(val10 > 23) val10 = 0;
            WS2812_SetColor(val10, 0xFF, 0xFF, 0xFF);


            if(step1 > 23)
                time1 = 0;
        }
        else
        {
            WS2812_SetColor(0xFF, 0xFF, 0xFF, 0xFF);
            time1 = 0;
        }



//      time1 += 20;
//      uint8_t step1 = (time1 / 50) % ((RGB_NUM1/2)-1);
//      if(time1 > 50*((RGB_NUM1/2)-1))
//          time1 = 0;
//      WS2812_SetColor(step1, 0, 0, 0);
//      WS2812_SetColor(step1+1, 0, 0, 0);
//      WS2812_SetColor(24-step1, 0, 0, 0);
//      WS2812_SetColor(24-(step1 + 1), 0, 0, 0);
```

```
//                  WS2812_SetColor(0, 0, 0, 0);
//                  WS2812_SetColor(1, 0, 0, 0);
//                  WS2812_SetColor(23-0, 0, 0, 0);
//                  WS2812_SetColor(23-1, 0, 0, 0);
                    break; // 正常情况下
            case 2:
                    break; // 按键变换检测
            case 3:
                    break; // 按键变换检测
            }
    }

    WS28121_SetColor(1, 0xFF, 0xFF, 0xFF);
    WS28121_SetColor(2, 0xFF, 0xFF, 0xFF);
    WS28121_SetColor(3, 0xFF, 0xFF, 0xFF);

    if (!WS2812_Status[0].Sending || !WS2812_Status[1].Sending)
        WS2812_Update();
}
#include "Drv_Flash.h"

/********************用法********************/
// Flash_Write((uint8_t *)(&Param),sizeof(Param));
// Flash_Read((uint8_t *)(&Param),sizeof(Param));

/**
 * @brief 写入 Flash
 *
 * @param addr 需要写入结构体的地址
 * @param len 结构体长度
 * @return uint8_t 1：成功，0：失败
 */
uint8_t Flash_Write(uint8_t *addr, uint16_t len)
{
    uint16_t FlashStatus;                   // 定义写入 Flash 状态
    FLASH_EraseInitTypeDef My_Flash; // 声明 FLASH_EraseInitTypeDef 结构体为 My_Flash

    HAL_FLASH_Unlock(); // 解锁 Flash

    My_Flash.TypeErase = FLASH_TYPEERASE_PAGES;       // 标明 Flash 执行页面只做擦除操作
    My_Flash.PageAddress = PARAMFLASH_BASE_ADDRESS; // 声明要擦除的地址
    My_Flash.NbPages = 1;                                   // 说明要擦除的页数，此参数必须是 Min_Data = 1 和 Max_Data =(最大页数-初始页的值)之间的值

    uint32_t PageError = 0; // 设置 PageError,如果出现错误这个变量会被设置为出错的 FLASH 地址

    FlashStatus = HAL_FLASHEx_Erase(&My_Flash, &PageError); // 调用擦除函数（擦除
```

```
Flash）
    if (FlashStatus != HAL_OK)
        return 0;

    for (uint16_t i = 0; i < len; i = i + 2)
    {
        uint16_t temp; // 临时存储数值
        if (i + 1 <= len - 1)
            temp = (uint16_t)(addr[i + 1] << 8) + addr[i];
        else
            temp = 0xff00 + addr[i];
        // 对 Flash 进行烧写，FLASH_TYPEPROGRAM_HALFWORD 声明操作的 Flash 地址的 16 位的，此外还有 32 位跟 64 位的操作，自行翻查 HAL 库的定义即可
        FlashStatus = HAL_FLASH_Program(FLASH_TYPEPROGRAM_HALFWORD, PARAMFLASH_BASE_ADDRESS + i, temp);
        if (FlashStatus != HAL_OK)
            return 0;
    }
    HAL_FLASH_Lock(); // 锁住 Flash
    return 1;
}

/**
 * @brief 读取 Flash
 *
 * @param addr 需要写入结构体的地址
 * @param len 结构体长度
 * @return uint8_t 1：成功
 */
uint8_t Flash_Read(uint8_t *addr, uint16_t len)
{
    for (uint16_t i = 0; i < len; i = i + 2)
    {
        uint16_t temp;
        if (i + 1 <= len - 1)
        {
            temp = (*(__IO uint16_t *)(PARAMFLASH_BASE_ADDRESS + i)); //*(__IO uint16_t *)是读取该地址的参数值,其值为 16 位数据,一次读取两个字节
            addr[i] = BYTE0(temp);
            addr[i + 1] = BYTE1(temp);
        }
        else
        {
            temp = (*(__IO uint16_t *)(PARAMFLASH_BASE_ADDRESS + i));
            addr[i] = BYTE0(temp);
        }
    }
    return 1;
}
#include "Drv_Infrared.h"
```

```
/**********结构体*************/
_Infrared_ Infrared[2];

/**********局部变量声明*******/
uint32_t Infrared_Time;//发送的时间
uint8_t Infrared_Step;//发送的步骤

/**
 * @brief 红外驱动初始化
 *
 */
void Drv_Infrared_Init(void)
{
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);//开启 tim3 通道一
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2);//开启 tim3 通道二

    HAL_TIM_IC_Start_IT(&htim15, TIM_CHANNEL_1);    //开启 tim15 通道 1 的捕获（中断方式）
    HAL_TIM_IC_Start_IT(&htim15, TIM_CHANNEL_2);    //开启 tim15 通道 2 的捕获（中断方式）
    __HAL_TIM_ENABLE_IT(&htim15, TIM_IT_UPDATE);    //更新使能中断

    HAL_TIM_Base_Start_IT(&htim6);//开始定时器
}

/**
 * @brief 检测速度是否停止-0.05s
 *
 * @param dT 任务周期
 */
void Check_Infrared(float dT)
{
    Infrared[0].Someone_Time += dT;//每 100ms 进入
    Infrared[1].Someone_Time += dT;//每 100ms 进入
    if (Infrared[0].Someone_Time >= 0.5f)// 0.5s 发现没出发输入捕获
    {
        Infrared[0].Someone = 0;// 将有人清零
        Infrared[0].Someone_Time = 0;//计数清零
    }
    if (Infrared[1].Someone_Time >= 0.5f)// 0.5s 发现没出发输入捕获
    {
        Infrared[1].Someone = 0;// 将有人清零
        Infrared[1].Someone_Time = 0;//计数清零
    }
}

/**
 * @brief 定时器计数中断
 *
```

```
 */
void Infrared_TIM_Interrupt(TIM_HandleTypeDef *htim)
{
    if(htim->Instance == htim6.Instance)
    {
        if(Infrared_Step == 0)
        {
            Infrared_Send1 = 79;
            Infrared_Send2 = 79;
            Infrared_Time++;
            if(Infrared_Time >= 900)
            {
                Infrared_Step = 1;
                Infrared_Time = 0;
            }
        }
        if(Infrared_Step == 1)
        {
            Infrared_Send1 = 0;
            Infrared_Send2 = 0;
            Infrared_Time++;
            if(Infrared_Time >= 225)
            {
                Infrared_Step = 2;
                Infrared_Time = 0;
            }
        }
        if(Infrared_Step == 2)
        {
            Infrared_Send1 = 79;
            Infrared_Send2 = 79;
            Infrared_Time++;
            if(Infrared_Time >= 56)
            {
                Infrared_Step = 3;
                Infrared_Time = 0;
            }
        }
        if(Infrared_Step == 3)
        {
            Infrared_Send1 = 0;
            Infrared_Send2 = 0;
            Infrared_Time++;
            if(Infrared_Time >= 9819)
            {
                Infrared_Step = 0;
                Infrared_Time = 0;
            }
        }
    }
```

```
}

/**
 * @brief 红外检测信号变化定时器捕获
 *
 */
void Infrared_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Instance == TIM15)
    {
        if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2)
        {
            if(IR1_IN)//capture 了上升沿
            {
                TIM_RESET_CAPTUREPOLARITY(&htim15, TIM_CHANNEL_2);//清除捕获上升沿
                TIM_SET_CAPTUREPOLARITY(&htim15, TIM_CHANNEL_2, TIM_ICPOLARITY_FALLING);//开始捕获下降沿
                __HAL_TIM_SET_COUNTER(&htim15, 0);//清空定时器值
                Infrared[0].IRSta |= 0x10;//[4]置 1，即标志上升沿已捕获
            }
            else  //捕获下降沿
            {
                Infrared[0].Dval = HAL_TIM_ReadCapturedValue(&htim15, TIM_CHANNEL_2);//下降沿计数
                TIM_RESET_CAPTUREPOLARITY(&htim15,TIM_CHANNEL_2);// 清 除捕获下降沿

    TIM_SET_CAPTUREPOLARITY(&htim15,TIM_CHANNEL_2,TIM_ICPOLARITY_RISING);//开始捕获上升沿
                if(Infrared[0].IRSta & 0x10)  //如果完成了一次高电平捕获，接下来看是否有引导码
                {
                    if(Infrared[0].Dval>2100 && Infrared[0].Dval<2400)//2.25ms 高电平，引导码
                    {
                        Infrared[0].Someone = 1;
                        Infrared[0].Someone_Time = 0;
                        Infrared[0].IRSta &= 0xF0;
                    }
                }
                Infrared[0].IRSta &=~(1<<4);//清空[4]，即高电平计数结束
            }
        }
        if (htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
        {
            if(IR2_IN)//capture 了上升沿
            {
                TIM_RESET_CAPTUREPOLARITY(&htim15, TIM_CHANNEL_1);//清除捕获上升沿
```

```
                    TIM_SET_CAPTUREPOLARITY(&htim15, TIM_CHANNEL_1, TIM_ICPOLARITY_FALLING);//开始捕获下降沿
                    __HAL_TIM_SET_COUNTER(&htim15, 0);//清空定时器值
                    Infrared[1].IRSta |= 0x10;//[4]置 1，即标志上升沿已捕获
                }
                else  //捕获下降沿
                {
                    Infrared[1].Dval = HAL_TIM_ReadCapturedValue(&htim15, TIM_CHANNEL_1);//下降沿计数
                    TIM_RESET_CAPTUREPOLARITY(&htim15,TIM_CHANNEL_1);// 清 除捕获下降沿

    TIM_SET_CAPTUREPOLARITY(&htim15,TIM_CHANNEL_1,TIM_ICPOLARITY_RISING);//开始捕获上升沿
                    if(Infrared[1].IRSta & 0x10)//如果完成了一次高电平捕获，接下来看是否有引导码
                    {
                        if(Infrared[1].Dval>2100 && Infrared[1].Dval<2600)//2.25ms 高电平，引导码
                        {
                            Infrared[1].Someone = 1;
                            Infrared[1].Someone_Time = 0;
                            Infrared[1].IRSta &= 0xF0;
                        }
                    }
                    Infrared[1].IRSta &=~(1<<4); //清空[4]，即高电平计数结束
                }
            }
        }
}

#include "Drv_TM1650.h"

/**********局部变量声明******/
static uint8_t s_7number[10] = {0xF5,0x84,0xB3,0x97,0xC6,0x57,0x77,0x85,0xF7,0xD7};

/**
 * @brief TM1650 开始函数
 *
 */
void TM1650_Start(void)
{
    SDA_OUT();
    TM1650_SDA_H;
    TM1650_SCL_H;
    Delay_us(4);
    TM1650_SDA_L;
    Delay_us(4);
    TM1650_SCL_L;
    Delay_us(4);
```

```
}

/**
 * @brief TM1650 停止函数
 *
 */
void TM1650_Stop(void)
{
    SDA_OUT();
    TM1650_SDA_L;
    TM1650_SCL_H;
    Delay_us(4);
    TM1650_SDA_H;
}

/**
 * @brief TM1650 等待响应
 *
 */
uint8_t TM1650_Wait_Ack(void)
{
    uint8_t ucErrTime=0;
    TM1650_SDA_H;
    Delay_us(4);
    SDA_IN();
    TM1650_SCL_H;
    Delay_us(4);
    while(TM1650_SDA_RD())
    {
        ucErrTime++;
        if(ucErrTime>250)
        {
            TM1650_Stop();
            return 1;
        }
    }
    TM1650_SCL_L;
    return 0;
}

/**
 * @brief TM1650Ack 信号
 *
 */
void TM1650_Ack(void)
{
    TM1650_SCL_L;
    SDA_OUT();
    TM1650_SDA_L;
    Delay_us(4);
```

```
    TM1650_SCL_H;
    Delay_us(4);
    TM1650_SCL_L;
}

/**
 * @brief TM1650NAck 信号
 *
 */
void TM1650_NAck(void)
{
    TM1650_SCL_L;
    SDA_OUT();
    TM1650_SDA_H;
    Delay_us(4);
    TM1650_SCL_H;
    Delay_us(4);
    TM1650_SCL_L;
}

/**
 * @brief TM1650 发送一个字节数据
 *
 * @param oneByte  字节值
 */
void TM1650_Send_Byte(uint8_t oneByte)
{
    uint8_t t;
    SDA_OUT();
    TM1650_SCL_L;
    TM1650_SDA_L;
    for(t=0;t<8;t++)
    {
        TM1650_SCL_L;
        Delay_us(2);
        if((oneByte&0x80)==0x80)
        {
            TM1650_SDA_H;
            Delay_us(4);
        }
        else
        {
            TM1650_SDA_L;
            Delay_us(4);
        }
        oneByte<<=1;
        Delay_us(4);
        TM1650_SCL_H;
        Delay_us(4);
        TM1650_SCL_L;
```

```
            Delay_us(4);
        }
        TM1650_SCL_L;
        TM1650_SDA_L;
}

/**
 * @brief TM1650 读一个字节数据
 *
 */
uint8_t TM1650_Read_Byte(void)
{
        uint8_t i,rekey=0;
        SDA_IN();
        for(i=0;i<8;i++ )
        {
            TM1650_SCL_L;
            Delay_us(4);
            TM1650_SCL_H;
            rekey<<=1;
            if(TM1650_SDA_RD()) rekey++;
            Delay_us(4);
        }
        return rekey;
}

/**
 * @brief TM1650 发送命令
 *
 * @param add  地址
 * @param dat  字节值
 */
void TM1650_SendCommand(uint8_t add,uint8_t dat)
{
        TM1650_Start();
        TM1650_Send_Byte(add);
        TM1650_Wait_Ack();
        TM1650_Send_Byte(dat);
        TM1650_Wait_Ack();
        TM1650_Stop();
}

/**
 * @brief TM1650 显示函数
 *
 * @param index  要设置的第几个数码管
 * @param num  设置的数值
 */
void TM1650_SendDigData(uint16_t index,uint16_t num)
{
```

```
    uint8_t indexAddr =    0;
    uint8_t numValue    =    0;
    switch(index)
    {
        case 1:indexAddr = 0x68;break;
        case 2:indexAddr = 0x6A;break;
        case 3:indexAddr = 0x6C;break;
        case 4:indexAddr = 0x6E;break;
        default:break;
    }
    numValue = s_7number[num];
//  numValue = num;
    TM1650_Start();
    TM1650_Send_Byte(indexAddr);
    TM1650_Wait_Ack();
    TM1650_Send_Byte(numValue);
    TM1650_Wait_Ack();
    TM1650_Stop();
}

/**
 * @brief TM1650 显示函数
 *
 * @param brightness  亮度 1 到 8 宏定义
 */
void TM1650_SetDisplay(uint8_t brightness)
{
    TM1650_SendCommand(0x48,brightness*16 + 1*4 + 1);
}

/**
 * @brief TM1650 显示位数值函数
 *
 * @param Num  数值
 */
void DisplayNumber_4BitDig(uint16_t Num)
{   uint16_t Numb;
    Numb = Num+10000;

//  TM1650_SendDigData(3,0x01);
    TM1650_SendDigData(3,Numb/1000%10);
    TM1650_SendDigData(2,Num/100%10);
    TM1650_SendDigData(1,Num/10%10);
    TM1650_SendDigData(4,Num%10);
}

/**
 * @brief TM1650 驱动初始化
 *
 */
```

```
void DrvTM1650_Init(void)
{
	TM1650_SCL_H;
	TM1650_SDA_H;
	HAL_Delay(20);//这个 20ms 延时不加开机点亮不了
	TM1650_SetDisplay(brighting_1);
}

#include "Drv_Uart.h"

#if 1

/**
  * @brief  重定向 c 库函数 printf 到 DEBUG_USARTx
  *
  */
int fputc(int ch, FILE *f)
{
	HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 0xffff);
	return ch;
}

/**
  * @brief  重定向 c 库函数 getchar,scanf 到 DEBUG_USARTx
  *
  */
int fgetc(FILE *f)
{
	uint8_t ch = 0;
	HAL_UART_Receive(&huart1, &ch, 1, 0xffff);
	return ch;
}
#endif

/**
  * @brief  串口初始化
  *
  */
void Drv_Uart_Init(void)
{
	HAL_UART_Receive_IT(&huart1, (uint8_t *)&aRxBuffer1, 1);
}

char RxBuffer1[RXBUFFERSIZE];//接收数据
uint8_t aRxBuffer1;//接收中断缓冲
uint8_t Uart1_Rx_Cnt = 0;//接收缓冲计数
/**
  * @brief  串口 1 中断回调函数，要在 mian 中先调用一次接收中断函数
  *
  * @param huart  串口号
```

```
 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    UNUSED(huart);
    /*********USART1**********/
    if(huart->Instance == USART1)//判断是由哪个串口触发的中断
    {
        HAL_UART_Receive_IT(&huart1, (uint8_t *)&aRxBuffer1, 1);//重新使能串口 1 接收
中断
    }
}

/**
 * @brief 打印一个字节
 *
 * @param huart 串口号
 * @param DataToSend 一个字节
 */
void Uart_Put_Char(UART_HandleTypeDef *huart, uint8_t DataToSend)
{
    HAL_UART_Transmit(huart, &DataToSend, 1, 0xFFFF);
}

/**
 * @brief 打印一串字符串
 *
 * @param huart 串口号
 * @param Str 一串字符串
 */
void Uart_Put_String(UART_HandleTypeDef *huart, uint8_t *Str)
{
    uint8_t *p;
    p = Str;
    while(*p != '\0')
    {
        Uart_Put_Char(huart, *p);
        p++;
    }
}

/**
 * @brief 打印一个数组
 *
 * @param huart 串口号
 * @param DataToSend 数组
 * @param data_num 数组长度
 */
void Uart_Put_Buf(UART_HandleTypeDef *huart, uint8_t *DataToSend , uint8_t data_num)
{
    while(data_num)
```

```
    {
        Uart_Put_Char(huart, *DataToSend);
        DataToSend++;
        data_num--;
    }
}
```