MS1200 软件源程序

```
#include "Drv_LedDisplay.h"

/*
*********************************************************************
 * 函数原型：void UCdata_Display(uint8_t uc)
 * 功      能：UC 数据判断控制引脚
*********************************************************************
*/
void UCdata_Display(uint8_t uc)
{
       for(uint8_t i=0; i<7; i++)
      {
           if((uc<<i) & 0x80)
           {
                switch(i)
                {
                     case 0:  HAL_GPIO_WritePin(A_GPIO_Port, A_Pin, GPIO_PIN_RESET);break;
                     case 1:  HAL_GPIO_WritePin(B_GPIO_Port, B_Pin, GPIO_PIN_RESET);break;
                     case 2:  HAL_GPIO_WritePin(C_GPIO_Port, C_Pin, GPIO_PIN_RESET);break;
                     case 3:  HAL_GPIO_WritePin(D_GPIO_Port, D_Pin, GPIO_PIN_RESET);break;
                     case 4:  HAL_GPIO_WritePin(E_GPIO_Port, E_Pin, GPIO_PIN_RESET);break;
                     case 5:  HAL_GPIO_WritePin(F_GPIO_Port, F_Pin, GPIO_PIN_RESET);break;
                     case 6:  HAL_GPIO_WritePin(G_GPIO_Port, G_Pin, GPIO_PIN_RESET);break;
                }
           }
           else
           {
                switch(i)
                {
                     case 0: HAL_GPIO_WritePin(A_GPIO_Port, A_Pin, GPIO_PIN_SET);break;
                     case 1: HAL_GPIO_WritePin(B_GPIO_Port, B_Pin, GPIO_PIN_SET);break;
                     case 2: HAL_GPIO_WritePin(C_GPIO_Port, C_Pin, GPIO_PIN_SET);break;
                     case 3: HAL_GPIO_WritePin(D_GPIO_Port, D_Pin, GPIO_PIN_SET);break;
                     case 4: HAL_GPIO_WritePin(E_GPIO_Port, E_Pin, GPIO_PIN_SET);break;
                     case 5: HAL_GPIO_WritePin(F_GPIO_Port, F_Pin, GPIO_PIN_SET);break;
                     case 6: HAL_GPIO_WritePin(G_GPIO_Port, G_Pin, GPIO_PIN_SET);break;
                }
           }
      }
}

/*
```

```
**********************************************************************
 * 函数原型：void DIGdata_Display(uint8_t DIG)
 * 功      能：DIG 数据判断控制引脚
**********************************************************************
*/
void DIGdata_Display(uint8_t DIG)
{
    for(uint8_t i=0; i<4; i++)
    {
        if((DIG<<i) & 0x80)
        {
            switch(i)
            {
                case 0: HAL_GPIO_WritePin(DIG1_GPIO_Port, DIG1_Pin, GPIO_PIN_SET);break;
                case 1: HAL_GPIO_WritePin(DIG2_GPIO_Port, DIG2_Pin, GPIO_PIN_SET);break;
                case 2: HAL_GPIO_WritePin(DIG3_GPIO_Port, DIG3_Pin, GPIO_PIN_SET);break;
                case 3: HAL_GPIO_WritePin(DIG4_GPIO_Port, DIG4_Pin, GPIO_PIN_SET);break;
            }
        }
        else
        {
            switch(i)
            {
                case 0: HAL_GPIO_WritePin(DIG1_GPIO_Port, DIG1_Pin, GPIO_PIN_RESET);break;
                case 1: HAL_GPIO_WritePin(DIG2_GPIO_Port, DIG2_Pin, GPIO_PIN_RESET);break;
                case 2: HAL_GPIO_WritePin(DIG3_GPIO_Port, DIG3_Pin, GPIO_PIN_RESET);break;
                case 3: HAL_GPIO_WritePin(DIG4_GPIO_Port, DIG4_Pin, GPIO_PIN_RESET);break;
            }
        }
    }
}

/*
**********************************************************************
 * 函数原型：void DIGdata_Set(void)
 * 功      能：1-4DIG 引脚全部拉高
**********************************************************************
*/
void DIGdata_Set(void)
{
    HAL_GPIO_WritePin(DIG1_GPIO_Port, DIG1_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(DIG2_GPIO_Port, DIG2_Pin, GPIO_PIN_RESET);
```

```
    HAL_GPIO_WritePin(DIG3_GPIO_Port, DIG3_Pin, GPIO_PIN_RESET);
    HAL_GPIO_WritePin(DIG4_GPIO_Port, DIG4_Pin, GPIO_PIN_RESET);
}
#include "Drv_EC11A.h"

/**********结构体***********/
_EC11A_ EC11A[1];//旋钮参数

/*
*********************************************************************
 * 函数原型：void EC11A_Init(void)
 * 功    能：EC11A 初始化定时器
*********************************************************************
*/
void EC11A_Init(void)
{
    EC11A[0].EXTI_Pin = KEY1A_Pin;//EC11A 旋钮中断引脚
    EC11A[0].EC11A_Pin = KEY1B_Pin;//EC11A 旋钮输入引脚
    EC11A[0].EC11A_GPIO = KEY1B_GPIO_Port;//EC11A 旋钮输入 GPIO 端口

    EC11A[0].Key_Pin = KEY1_Pin;//EC11A 按键输入引脚
    EC11A[0].Key_GPIO = KEY1_GPIO_Port;//EC11A 按键输入 GPIO 端口

    EC11A[0].Tim = &EC11A_Tim_1;//定时器选择
    EC11A[0].EC11A_Fast = EC11A_Fast_1;//判断旋转速度阈值
}

/*
*********************************************************************
 * 函数原型：void EC11A_Speed(float dT)
 * 功    能：EC11A 旋钮速度计算
*********************************************************************
*/
void EC11A_Speed(float dT)
{
    EC11A[0].EC11A_Speed = EC11A[0].EC11A_Cnt*60/20;//一秒检测一次。转一圈 20 个反
馈，一分钟的速度
    EC11A[0].EC11A_Cnt = 0;//将检测到的计数清零
}

/*
*********************************************************************
 * 函数原型：void Check_Knob(float dT)
 * 功    能：检测旋钮状态-500ms
*********************************************************************
*/
void Check_Knob(float dT)
{
    if(!EC11A[0].EC11A_Knob)
        return;
```

```
    if(EC11A[0].EC11A_Knob)//旋钮被转动
        EC11A[0].EC11A_Knob -=dT;//倒计时
    if(!EC11A[0].EC11A_Knob)
    {
        Speed.Ctrl_Speed = Speed.Set_Speed;
        Param.Speed = Speed.Set_Speed;
        Save_Param_En = 1;
        if(sys.Run_Status)
            Beep_Time = 0.1;//蜂鸣器响 0.1S
    }
}

/*
**********************************************************************
 * 函数原型：void EC11AKey_Scan(float dT)
 * 功      能：EC11A 按键扫描
**********************************************************************
*/
void EC11AKey_Scan(float dT)
{
    /*************************************MENU                                键
*************************************/
    if(HAL_GPIO_ReadPin(EC11A[0].Key_GPIO,EC11A[0].Key_Pin) == KEY_DOWN)//按下
按键
    {
        if(EC11A[0].LongPress == 0)//没有长按过
        {
            EC11A[0].Key_Cnt += dT;//按下时间++
            EC11A[0].Key_Flag = 1;//按键按下标志置一
        }
    }
    if(EC11A[0].Key_Flag == 1)//按键被按下
    {
        if(HAL_GPIO_ReadPin(EC11A[0].Key_GPIO,EC11A[0].Key_Pin)  ==  KEY_UP)//抬
起按键
        {
            if(EC11A[0].Key_Cnt > 0.1 && EC11A[0].Key_Cnt < 1.5)//小于 1.5S 是单击
            {
                if(!sys.Run_Status)
                {
                    sys.Run_Status = 1;//启动系统
                    Speed_Val.Integral = 25;
                }
                else
                {
                    sys.Run_Status = 0;//启动系统
                }
                Beep_Time = 0.1;//蜂鸣器响 0.1S
            }
            EC11A[0].Key_Flag = 0;//按键事件结束，等待下一次按下
```

```
            EC11A[0].LongPress = 0;//长按标志清零
            EC11A[0].Key_Cnt = 0;//按钮计数清零
        }
        if(EC11A[0].Key_Cnt > 1.5 && EC11A[0].Key_Cnt < 3)//按键时间大于 1.5S 小于 3S
表示长按
        {
            if(EC11A[0].LongPress == 0)//如果没有一直一直长按着
            {

                EC11A[0].LongPress = 1;//长按标志置一
            }
        }
    }
}

/*
**********************************************************************
 * 函数原型：void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
 * 功　　能：外部中断
**********************************************************************
*/
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    UNUSED(GPIO_Pin);

    if(GPIO_Pin == EC11A[0].EXTI_Pin)//A 上升沿触发外部中断
    {
        HAL_TIM_Base_Start_IT(EC11A[0].Tim);//开始定时器
        while(EC11A[0].TIM_Cnt <= 2)//定时器一个周期 1ms，计时 2ms 内看看 A 有没有电
跳变
        {
            if(GPIO_Pin == EC11A[0].EXTI_Pin)//在 2ms 内，检测到电平变化
            {
                HAL_TIM_Base_Stop_IT(EC11A[0].Tim);//停止定时器
                EC11A[0].TIM_Cnt = 0;//清除 TIM 计数
                EC11A[0].EC11A_Cnt++;//旋钮计数
                EC11A[0].EC11A_Knob = 2;//在旋转旋钮时
                if(HAL_GPIO_ReadPin(EC11A[0].EC11A_GPIO,EC11A[0].EC11A_Pin) ==
0)//加
                {
                    if(EC11A[0].EC11A_Speed < EC11A[0].EC11A_Fast)//如果慢慢旋转
                        Speed.Set_Speed += 10;
                    else
                        Speed.Set_Speed += 30;
                    if(Speed.Set_Speed > Speed_MAX)
                        Speed.Set_Speed = Speed_MAX;
                    break;
                }
                else if(HAL_GPIO_ReadPin(EC11A[0].EC11A_GPIO,EC11A[0].EC11A_Pin)
== 1)//减
```

```
                    {
                        if(EC11A[0].EC11A_Speed < EC11A[0].EC11A_Fast)//如果慢慢旋转
                            Speed.Set_Speed -= 10;
                        else
                            Speed.Set_Speed -= 30;
                        if(Speed.Set_Speed <= Speed_MIN)
                            Speed.Set_Speed = Speed_MIN;
                        break;
                    }
                    break;
                }
            }
            HAL_TIM_Base_Stop_IT(EC11A[0].Tim);//停止定时器
            EC11A[0].TIM_Cnt = 0;//清除 TIM 计数
        }
}

/*
**********************************************************************
 * 函数原型：void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
 * 功      能：定时器计数中断
**********************************************************************
*/
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Instance == EC11A_Tim_1.Instance)
    {
        EC11A[0].TIM_Cnt++;
    }
    if(htim->Instance == TIM1)
    {
        if(P_Status)//捕获周期计数状态:开启
        {
            TIM1CH3_CAPTURE_STA++;//捕获周期数++
        }
    }
}
#include "Drv_Beep.h"

/**********全局变量**********/
float Beep_Time;//蜂鸣器响的时间
float Beep_Flash;//蜂鸣器响的次数

/*
**********************************************************************
 * 函数原型：  void Buzzer_Status(float dT)
 * 功      能：  蜂鸣器的状态检测
 * 输      入:  dT:执行周期-50ms
 * 参      数：  uint16_t dT
**********************************************************************
```

```
*/
void Buzzer_Status(float dT)
{
    static float BT;
    if(Beep_Time <= 0 && Beep_Flash <= 0)//蜂鸣器的时间小于等于 0 时
    {
        Beep_OFF;//关闭蜂鸣器
        return;
    }
    if(Beep_Time)
    {
        Beep_ON;//打开蜂鸣器
        Beep_Time -= dT;//蜂鸣器响的时间--
    }
    if(Beep_Flash)
    {
        BT =   BT + dT;//周期++
        if(BT < 0.2)//如果小于 0.2s 时
        {
            Beep_ON;//蜂鸣器响
        }
        else if(BT >= 0.2 && BT < 0.3)//在 0.2 和 0.3s 之间时
        {
            Beep_OFF;//关闭蜂鸣器
        }
        else if(BT >= 0.3)//大于等于 0.2s 时
        {
            Beep_Flash--;//次数--
            BT = 0;//周期清零
        }
    }
}
#include "Drv_Flash.h"

/*********************用法*********************/
//Flash_Write((uint8_t *)(&Param),sizeof(Param));
//Flash_Read((uint8_t *)(&Param),sizeof(Param));
/*
*********************************************************************
 * 函数原型：uint8_t Flash_Write(uint8_t *addr, uint16_t len)
 * 功    能：写入 Flash
 * 输    入：addr 需要写入结构体的地址，len 结构体长度
 * 输    出：写入是否成功
 * 参    数：uint8_t *addr, uint16_t len
*********************************************************************
*/
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;
}

```
/*
**********************************************************************
 * 函数原型：uint8_t Flash_Read(uint8_t *addr, uint16_t len)
 * 功　　能：读取 Flash
 * 输　　入：addr 需要写入结构体的地址，len 结构体长度
 * 输　　出：读取是否成功
 * 参　　数：uint8_t *addr, uint16_t len
**********************************************************************
*/
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_Motor.h"

/*
**********************************************************************
 * 函数原型：void Motor_Init(void)
 * 功      能：电机初始化
**********************************************************************
*/
void Motor_Init(void)
{
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);//开启 tim1 通道一
    HAL_GPIO_WritePin(MO_GPIO_Port, MO_Pin, GPIO_PIN_RESET);//电机刹车打开
}
#include "Show.h"

/**********全局变量声明******/
uint8_t Display_Flag;//用于显示刷新

/**********局部变量声明******/
uint8_t Speed_ShowFlag;//速度显示的标志位 0:常亮 1：熄灭
uint8_t DIG;//DIG 的数据
uint8_t SPEED_Tab[] = {0xFC, 0x60, 0xDA, 0xF2, 0x66, 0xB6, 0xBE, 0xE0, 0xFE, 0xF6};//显示 0-9
uint8_t Cnt;//显示位数，用于刷新

/*
**********************************************************************
 * 函数原型：void Display_SpeedShow(int16_t speed)
 * 功      能：显示速度
 * 输      入: speed：要显示的速度
 * 参      数：int16_t speed
**********************************************************************
*/
void Display_Speed(int16_t speed)
{
```

```
    uint8_t Val;//用于百十个取出来的数字
    if(Cnt == 1)
    {
        /***********L1 千位***********/
        if(speed > 999)//大于 999 时
        {
            Val=speed/1000;//取出千位的数字
            UCdata_Display(SPEED_Tab[Val]);
//            if(Speed_ShowFlag >= 1 && EC11A_Knob == 0)//闪烁速度
//                DIGdata_Display(0x00);
//            else
                DIGdata_Display(0x80);
        }
        else//小于 999 时
        {
            DIGdata_Display(0x00);
        }
    }
    else if(Cnt == 2)
    {
        /***********L1 百位***********/
        if(speed > 99)//大于 99 时
        {
            Val=speed/100;//取出百位的数字
            if(speed > 999)//加入大于 999 时
                Val=Val%10;//取出百位的数字
            UCdata_Display(SPEED_Tab[Val]);
//            if(Speed_ShowFlag >= 1 && EC11A_Knob == 0)
//                DIGdata_Display(0x00);
//            else
                DIGdata_Display(0x40);
        }
        else
        {
            DIGdata_Display(0x00);//不显示
        }

    }
    else if(Cnt == 3)
    {
        /***********L1 十位***********/
        if(speed > 9)//大于 9 时
        {
            Val=speed/10;//取出十位的数字
            if(speed > 99)//大于 99 时
                Val=Val%10;//取出十位的数字
            UCdata_Display(SPEED_Tab[Val]);
//            if(Speed_ShowFlag >= 1 && EC11A_Knob == 0)
//                DIGdata_Display(0x00);
//            else
```

```
                    DIGdata_Display(0x20);
            }
            else
            {
                DIGdata_Display(0x00);//不显示
            }
        }
        else if(Cnt == 4)
        {
            /***********L1 个位***********/
            Val=speed%10;//取出个位
            UCdata_Display(SPEED_Tab[Val]);
//              if(Speed_ShowFlag >= 1 && EC11A_Knob == 0)
//                  DIGdata_Display(0x00);
//              else
                DIGdata_Display(0x10);
        }
        else if(Cnt == 5)//刷新
        {
            DIGdata_Set();
            Cnt = 0;
        }
}
/*
*********************************************************************
 * 函数原型：  void Show_Display(void)
 * 功      能：  显示屏幕内容
*********************************************************************
*/
void Show_Display(void)
{
        if(Display_Flag == 1)
        {
            Cnt++;
            if(!sys.Run_Status)
            {
                Speed.Display_Speed = Speed.Set_Speed;
            }
            else
            {
                if(EC11A[0].EC11A_Knob > 0)
                {
                    Speed.Display_Speed = Speed.Set_Speed;
                }
                else
                {
                    Speed.Display_Speed = Speed.Rel_Speed;
        //          Deal_Speed();
                }
            }
```

```
            Display_Speed(Speed.Display_Speed);
            Display_Flag = 0;
        }
}
#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;

/*
*********************************************************************
 * 函数原型：void Param_Reset(void)
 * 功      能：初始化硬件中的参数
*********************************************************************
*/
void Param_Reset(void)
{
    Param.Flash_Check_Start = FLASH_CHECK_START;

    Param.Speed = 2000;//转速 2000

    Param.Flash_Check_End   = FLASH_CHECK_END;
}

/*
*********************************************************************
 * 函数原型：  void Param_Save(void)
 * 功      能：  保存硬件中的参数
*********************************************************************
*/
void Param_Save(void)
{
    Flash_Write((uint8_t *)(&Param),sizeof(Param));
}

/*
*********************************************************************
 * 函数原型：void Param_Read(void)
 * 功      能：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_Speed = Param.Speed;//将 Flash 中的速度赋值
        EC11A[0].EC11A_Knob = 1;
        Save_Param_En = 1;
    }
    else
    {
        Speed.Set_Speed = Param.Speed;//将 Flash 中的速度赋值
        EC11A[0].EC11A_Knob = 1;
    }

    //保存参数
    if(Save_Param_En)
    {
        Save_Param_En = 0;
        Param_Save();
    }
}

/*
**********************************************************************
 * 函数原型：void Param_Save_Overtime(float dT)
 * 功      能：保存标志位置 1，0.5s 后保存
**********************************************************************
*/
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 "Speed.h"

/**********全局变量声明******/
uint8_t CAPTURE_Status = 0;//捕获状态
uint8_t CAPTURE_First = 1;//捕获第一个高电平
uint16_t TIM1CH3_CAPTURE_STA = 0;//捕获周期数
```

```
uint32_t TIM1CH3_CAPTURE_VAL;//捕获计数值
uint32_t TIM1CH3_CAPTURE_VAL1;//第一次进入中断时的数值
uint32_t P_Status = 1;//捕获周期计数状态    1 开启  0 关闭
float frq;//周期频率值

/*
**********************************************************************
 * 函数原型：void Encoder_Init(void)
 * 功      能：编码器初始化
**********************************************************************
*/
void Encoder_Init(void)
{
    HAL_TIM_Base_Start_IT(&htim1);//开启定时器 1
    HAL_TIM_IC_Start_IT(&htim1, TIM_CHANNEL_3);//开启 time1 通道 3 输入捕获
}

/*
**********************************************************************
 * 函数原型：void Check_Speed(float dT)
 * 功      能：检测速度是否停止-0.05s
**********************************************************************
*/
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_Speed = 0;//将速度清零
        Speed.Stop_Cnt = 0;//计数清零
    }
}

/*
**********************************************************************
 * 函数原型：void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
 * 功      能：输入捕获回调函数
**********************************************************************
*/
uint32_t Capture,rel;//输入捕获数和计算后的速度
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if(CAPTURE_Status == 0)
    {
        if(CAPTURE_First)//捕获到第一个上升沿
        {
            CAPTURE_Status = 1;//捕获中
            CAPTURE_First = 0;//清除捕获第一个上升沿标志
            TIM1CH3_CAPTURE_VAL      =      HAL_TIM_ReadCapturedValue(&htim1,
TIM_CHANNEL_3);//获取当前捕获计数值
```

```
                TIM1CH3_CAPTURE_VAL += TIM1CH3_CAPTURE_VAL1;
                Capture = TIM1CH3_CAPTURE_STA;//定时器的周期值
                Capture *= 100;//一个周期 1000us（48000000/24/200 = 0.0001s)
                Capture += TIM1CH3_CAPTURE_VAL;//输入不获到的微秒数加上之前周期的
时间
                frq = 1.0 / (((float)Capture) / 1000000.0);//频率计算，用 1S/（周期/1000000.0) ;
（周期/1000000.0)为转化单位为 S
                rel = 60 * frq / 2;//rpm
                Speed.Rel_Speed = rel;
                P_Status = 0;//捕获周期计数状态:0 关闭
                Speed.Stop_Cnt = 0;//检测速度
                __HAL_TIM_SET_COUNTER(&htim1, 0);//清楚输入捕获的值
                __HAL_TIM_DISABLE(&htim1);//关闭定时器一计数
            }
            else
            {
                TIM1CH3_CAPTURE_STA = 0;//清除周期计数
                TIM1CH3_CAPTURE_VAL = 0;//清楚捕获寄存器
                CAPTURE_First = 1;//已捕获第一个上升沿
                TIM1CH3_CAPTURE_VAL1      =      HAL_TIM_ReadCapturedValue(&htim1,
TIM_CHANNEL_3);//获取当前捕获计数值
                CAPTURE_Status = 0;//捕获结束
                P_Status = 1;//捕获周期计数
            }
        }
}

/*
**********************************************************************
 * 函数原型：void TIM1_Poll(void)
 * 功      能：TIM1 轮训状态切换
**********************************************************************
*/
void TIM1_Poll(void)
{
    if(CAPTURE_Status)
    {
        __HAL_TIM_ENABLE(&htim1);//开启计数定时器一计数
        CAPTURE_Status=0;//捕获状态为 0
        TIM1CH3_CAPTURE_STA=0;//捕获的周期清零
    }
}
#include "PID.h"

/*
**********************************************************************
 * 函数原型：void AltPID_Calculation(float dT, float Expect, float Freedback, _PID_Arg_ *
PID_Arg, _PID_Val_ * PID_Val, float Error_Lim, float Integral_Lim)
 * 功      能：微分先行 PID 计算
 * 输      入：dT：周期（单位：秒）
```

```
                Expect：期望值（设定值）
                Freedback：反馈值
                _PID_Arg_ * PID_Arg：PID 参数结构体
                _PID_Val_ * PID_Val：PID 数据结构体
                Error_Lim：误差限幅
                Integral_Lim：积分误差限幅
 * 参      数：float dT, float Expect, float Freedback, _PID_Arg_ * PID_Arg, _PID_Val_ * PID_Val, float Error_Lim, float Integral_Lim
**********************************************************************
*/
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
}

/*
**********************************************************************
 * 函数原型：void IncPID_Calculation(float dT,float Expect,float Freedback,_PID_Arg_ * PID_Arg,_PID_Val_ * PID_Val,float Integral_Lim)
 * 功      能：增量式 PID 计算
 * 输      入：dT：周期（单位：秒）
                Expect：期望值（设定值）
                Freedback：反馈值
                _PID_Arg_ * PID_Arg：PID 参数结构体
                _PID_Val_ * PID_Val：PID 数据结构体
 * 参      数：float dT,float Expect,float Freedback,_PID_Arg_ * PID_Arg,_PID_Val_ * PID_Val
**********************************************************************
*/
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 "Ctrl_Scheduler.h"

uint16_t   T_cnt_2ms=0,
           T_cnt_10ms=0,
           T_cnt_50ms=0,
           T_cnt_100ms=0,
           T_cnt_500ms=0,
           T_cnt_1S=0;

void Loop_Check(void)
{
    T_cnt_2ms++;
    T_cnt_10ms++;
    T_cnt_50ms++;
    T_cnt_100ms++;
    T_cnt_500ms++;
    T_cnt_1S++;

    Sys_Loop();
}
uint16_t val = 0;
static void Loop_2ms(float dT)//2ms 执行一次
{
    Display_Flag = 1;

}

static void Loop_10ms(float dT)//10ms 执行一次
{
    EC11AKey_Scan(dT);//EC11A 按键扫描
    Param_Save_Overtime(dT);//保存标志位置 1，0.5s 后保存
}

static void Loop_50ms(float dT)//50ms 执行一次
{
    Motor_Ctrl(dT);//电机控制
    Buzzer_Status(dT);//蜂鸣器检测
}
```

```
static void Loop_100ms(float dT)//100ms 执行一次
{
//    PWM = val;
}

static void Loop_500ms(float dT)//500ms 执行一次
{
    Check_Knob(dT);//检测旋钮状态
    Check_Speed(dT);//检测速度是否停止
}

static void Loop_1S(float dT)//1S 执行一次
{
    EC11A_Speed(dT);//EC11A 旋钮速度计算
}

void Sys_Loop(void)
{
    if(T_cnt_2ms >= 1) {
        Loop_2ms(0.001f);
        T_cnt_2ms = 0;
    }
    if(T_cnt_10ms >= 10) {
        Loop_10ms(0.01f);
        T_cnt_10ms = 0;
    }
    if(T_cnt_50ms >= 50) {
        Loop_50ms(0.05f);
        T_cnt_50ms = 0;
    }
    if(T_cnt_100ms >= 100) {
        Loop_100ms(0.1f);
        T_cnt_100ms = 0;
    }
    if(T_cnt_500ms >= 500) {
        Loop_500ms(0.5f);
        T_cnt_500ms = 0;
    }
    if(T_cnt_1S >= 1000) {
        Loop_1S(1.0f);
        T_cnt_1S = 0;
    }
}
#include "Ctrl_Motor.h"

/**********结构体*************/
_PID_Arg_ Speed_Arg;
_PID_Val_ Speed_Val;
```

```
/*
**********************************************************************
 * 函数原型：void Motor_PID(void)
 * 功      能：电机控制 PID 系数
**********************************************************************
*/
void Motor_PID(void)
{
    Speed_Arg.Kp = 40 * 0.001f;
    Speed_Arg.Ki = 60 * 0.001f;
    Speed_Arg.Kd = 0 * 0.001f;
}

/*
**********************************************************************
 * 函数原型：void Motor_Ctrl(float dT)
 * 功      能：电机控制
**********************************************************************
*/
void Motor_Ctrl(float dT)
{
//  static float T = 0;
    if(sys.Run_Status)//启动
    {
        HAL_GPIO_WritePin(MO_GPIO_Port, MO_Pin, GPIO_PIN_SET);//电机刹车打开
        /**********Speed_L1**********/
        if(Speed.Ctrl_Speed)//速度大于 0 和定时器没有结束
        {

    AltPID_Calculation(dT,Speed.Ctrl_Speed,Speed.Rel_Speed,&Speed_Arg,&Speed_Val,150,1
50);//电机 PID 控制
            if(Speed_Val.Out<0)
                Speed_Val.Out = 0;
            PWM = Speed_Val.Out;//pid 输出
        }
        else
        {
            PWM = 0;//pid 输出
        }
    }
    else
    {
        if(Speed_Val.Out)
            Speed_Val.Out -= 1;
        if(Speed_Val.Out<0)
        {
            HAL_GPIO_WritePin(MO_GPIO_Port, MO_Pin, GPIO_PIN_RESET);//电机刹车
打开
            Speed_Val.Out = 0;
        }
```

```
            PWM = Speed_Val.Out;//pwm 不输出
        }
}
#include "System_Init.h"

/*
*********************************************************************
 * 函数原型：  void System_Init(void)
 * 功      能：  系统功能初始化
*********************************************************************
*/
void System_Init(void)
{
        /**********系统初始化成功**********/
        sys.Init_ok = 0;

        /**********EC11A 初始化定时器*******/
        EC11A_Init();

        /**********参数初始化**************/
        Param_Read();

        /**********电机初始化**************/
        Motor_Init();

        /**********编码器初始化************/
        Encoder_Init();

        /**********PID 系数初始化***********/
        Motor_PID();

        /**********开机蜂鸣器响************/
//      Beep_Time = 0.1;//蜂鸣器响 0.1S

        /**********系统初始化成功**********/
        sys.Init_ok = 1;
}
#include "Structs.h"

_sys_ sys;//系统初始化检测
_Speed_ Speed;//速度参数
#ifndef __Structs_H__
#define __Structs_H__

#include "stm32f0xx_hal.h"

#define FLASH_CHECK_START 0xAA
#define FLASH_CHECK_END 0xBB

#define Speed_MIN 200//转速最小 200
```

```
#define Speed_MAX 2000//转速最大 2000

typedef struct
{
    uint8_t Init_ok;//系统初始化是否完成，完成为 1
    uint8_t Run_Status;//系统状态
}_sys_;
extern _sys_ sys;//系统初始化检测

typedef struct
{
    uint8_t Speed_ADDMode;//显示处理的模式
    int16_t Set_Speed;//设置速度
    int16_t Ctrl_Speed;//控制速度（期望值）
    int16_t Rel_Speed;//实际速度
    int16_t Display_Speed;//用于显示速度
    int16_t Speed_New;//当前的最大最小速度
    int16_t Speed_Last;//之前的速度
    float Stop_Cnt;//检测电机是否停止
}_Speed_;
extern _Speed_ Speed;//速度参数


#endif
#ifndef _INCLUDE_H_
#define _INCLUDE_H_

/***************系统头文件****************/
#include "System_Init.h"
#include "Structs.h"
#include "stdlib.h"
#include "string.h"
#include "math.h"
#include "tim.h"
#include "gpio.h"
#include <stdio.h>

/***************硬件头文件****************/
#include "Drv_LedDisplay.h"
#include "Drv_EC11A.h"
#include "Drv_Beep.h"
#include "Drv_Flash.h"
#include "Drv_Motor.h"

/***************应用头文件****************/
#include "Show.h"
#include "Param.h"
#include "Speed.h"
#include "PID.h"
```

```
/***************底层头文件***************/
#include "Ctrl_Scheduler.h"
#include "Ctrl_Motor.h"

#endif
#ifndef __DRV_EC11A_H__
#define __DRV_EC11A_H__

#include "include.h"

/**********宏定义**********/
#define EC11A_Tim_1 htim6//旋钮 1 的定时器
#define EC11A_Fast_1 40//旋钮快转的值

//用于识别按键按下的状态识别
#define PRESS RESET//低电平识别

#define KEY_DOWN PRESS/*键按下*/
#define KEY_UP   !PRESS/*键弹起*/

/**********结构体**********/
typedef struct
{
    uint16_t EXTI_Pin;//EC11A 旋钮中断引脚
    uint16_t EC11A_Pin;//EC11A 旋钮输入引脚
    GPIO_TypeDef* EC11A_GPIO;//EC11A 旋钮输入 GPIO 端口
    uint16_t Key_Pin;//EC11A 按键输入引脚
    GPIO_TypeDef* Key_GPIO;//EC11A 按键输入 GPIO 端口
    TIM_HandleTypeDef *Tim;//选用的定时器
    uint8_t EC11A_Fast;////旋钮快转的值

    float EC11A_Speed;//旋转旋钮时的速度
    uint16_t EC11A_Cnt;//旋动时的，一圈是 20 个
    uint8_t EC11A_Knob;//在旋动旋钮时
    uint8_t TIM_Cnt;//定时器计数

    float Key_Cnt;//按下时间
    uint8_t Key_Flag;//按键按下标志
    uint8_t LongPress;//按键长按标志

}_EC11A_;
extern _EC11A_ EC11A[1];//旋钮参数

/**********全局变量声明******/
/**********局部变量声明******/
/**********全局函数**********/
void EC11A_Init(void);//EC11A 初始化定时器
void EC11A_Speed(float dT);//EC11A 旋钮速度计算
void EC11AKey_Scan(float dT);//EC11A 按键扫描
void Check_Knob(float dT);//检测旋钮状态
```