

SK4100 软件源程序

---

```
#include "Drv_HT1623.h"
```

```
/*
*****
* 函数原型: static void delay(uint16_t time)
* 功    能: us 延时
* 输    入: time : 时间
* 参    数: uint16_t time
* 调    用: 内部调用
*****
*/
static void delay(uint16_t time)
{
    unsigned char a;
    for(a = 100; a > 0; a--);
}

/*
*****
* 函数原型: static void write_mode(unsigned char MODE)
* 功    能: 写入模式,数据 or 命令
* 输    入: MODE : 数据 or 命令
* 参    数: unsigned char MODE
* 调    用: 内部调用
*****
*/
static void write_mode(unsigned char MODE)
{
    delay(10);
    Clr_1625_Wr;//RW = 0;
    delay(10);
    Set_1625_Dat;//DA = 1;
    Set_1625_Wr;//RW = 1;
    delay(10);
    Clr_1625_Wr;//RW = 0;
    delay(10);
    Clr_1625_Dat;
    delay(10);//DA = 0;
    Set_1625_Wr;//RW = 1;
    delay(10);
    Clr_1625_Wr;//RW = 0;
    delay(10);
    if (0 == MODE)
    {
        Clr_1625_Dat;//DA = 0;
    }
    else
    {
        Set_1625_Dat;//DA = 1;
    }
}
```

```

    delay(10);
    Set_1625_Wr;//RW = 1;
    delay(10);
}

/*
*****
* 函数原型: static void write_command(unsigned char Cbyte)
* 功    能: LCD 命令写入函数
* 输    入: Cbyte: 控制命令字
* 参    数: unsigned char Cbyte
* 调    用: 内部调用
*****
*/
static void write_command(unsigned char Cbyte)
{
    unsigned char i = 0;
    for (i = 0; i < 8; i++)
    {
        Clr_1625_Wr;
        //Delay_us(10);
        if ((Cbyte >> (7 - i)) & 0x01)
        {
            Set_1625_Dat;
        }
        else
        {
            Clr_1625_Dat;
        }
        delay(10);
        Set_1625_Wr;
        delay(10);
    }
    Clr_1625_Wr;
    delay(10);
    Clr_1625_Dat;
    Set_1625_Wr;
    delay(10);
}

/*
*****
* 函数原型: static void write_address(unsigned char Abyte)
* 功    能: LCD 地址写入函数
* 输    入: Abyte: 地址
* 参    数: unsigned char Abyte
* 调    用: 内部调用
*****
*/
static void write_address(unsigned char Abyte)

```

---

```

{
    unsigned char i = 0;
    Abyte = Abyte << 1;
    for (i = 0; i < 6; i++)
    {
        Clr_1625_Wr;
        if ((Abyte >> (6 - i)) & 0x01)
        {
            Set_1625_Dat;
        }
        else
        {
            Clr_1625_Dat;
        }
        delay(10);
        Set_1625_Wr;
        delay(10);
    }
}

/*
*****
* 函数原型: static void write_data_8bit(unsigned char Dbyte)
* 功    能: LCD 8bit 数据写入函数
* 输    入: Dbyte: 数据
* 参    数: unsigned char Dbyte
* 调    用: 内部调用
*****
*/
static void write_data_8bit(unsigned char Dbyte)
{
    int i = 0;
    for (i = 0; i < 8; i++)
    {
        Clr_1625_Wr;
        if ((Dbyte >> (7 - i)) & 0x01)
        {
            Set_1625_Dat;
        }
        else
        {
            Clr_1625_Dat;
        }
        delay(10);
        Set_1625_Wr;
        delay(10);
    }
}

/*

```

## SK4100 软件 V1.0

```

*****
* 函数原型: void write_data_4bit(unsigned char Dbyte)
* 功    能: LCD 4bit 数据写入函数
* 输    入: Dbyte: 数据
* 参    数: unsigned char Dbyte
* 调    用: 内部调用
*****
*/
void write_data_4bit(unsigned char Dbyte)
{
    int i = 0;
    for (i = 0; i < 4; i++)
    {
        Clr_1625_Wr;
        if ((Dbyte >> (3 - i)) & 0x01)
        {
            Set_1625_Dat;
        }
        else
        {
            Clr_1625_Dat;
        }
        delay(10);
        Set_1625_Wr;
        delay(10);
    }
}

/*
*****
* 函数原型: void Lcd_Init(void)
* 功    能: LCD 初始化, 对 lcd 自身做初始化设置
*****
*/
void Lcd_Init(void)
{
    Set_1625-Cs;
    Set_1625-Wr;
    Set_1625-Dat;
    delay(500);
    Clr_1625-Cs;//CS = 0;
    delay(10);
    write_mode(0);//命令模式
    write_command(0x01);//Enable System
    write_command(0x03);//Enable Bias
    write_command(0x04);//Disable Timer
    write_command(0x05);//Disable WDT
    write_command(0x08);//Tone OFF
    write_command(0x18);//on-chip RC 震荡
    write_command(0x29);//1/4Duty 1/3Bias

```

---

```

    write_command(0x80);//Disable IRQ
    write_command(0x40);//Tone Frequency 4kHz
    write_command(0xE3);//Normal Mode
    Set_1625_Cs;//CS = 1;
}

/*
*****
* 函数原型: void Lcd_Clr(void)
* 功    能: LCD 清屏函数
*****
*/
void Lcd_Clr(void)
{
    Write_Addr_Dat_N(0x0, 0x00, 60);
}

/*
*****
* 函数原型: void Lcd_All(void)
* 功    能: LCD 全显示函数
*****
*/
void Lcd_All(void)
{
    Write_Addr_Dat_N(0x0, 0xff, 60);
}

/*
*****
* 函数原型: void Write_Addr_Dat_N(unsigned char _addr, unsigned char _dat, unsigned char
n)
* 功    能: 屏幕显示
* 输    入: _addr: 地址  char _dat: 数据  n: 个数
* 参    数: unsigned char _addr, unsigned char _dat, unsigned char n
*****
*/
void Write_Addr_Dat_N(unsigned char _addr, unsigned char _dat, unsigned char n)
{
    unsigned char i = 0;
    Clr_1625_Cs;//CS = 0;
    delay(10);
    write_mode(1);
    write_address(_addr);
    for (i = 0; i < n; i++)
    {
        write_data_8bit(_dat);
    }
    Set_1625_Cs;//CS = 1;
}

```

```

#include "Drv_EC11A.h"

/*****全局变量声明*****/
uint8_t Work_Option;//选择工位号
uint8_t SetMode_Option;//选择设置模式
uint8_t EC11A_Knob1,EC11A_Knob2;//在旋动旋钮时
uint8_t Run_Status;//系统状态
uint8_t Work_All;//工位设置相同

/*****局部变量声明*****/
uint8_t EC11A_Flag;//进入中断延时标志
uint8_t Key1_Press,Key2_Press;//按下按钮
uint16_t KEY1_Count,KEY2_Count;//记录 KEY1,KEY2 按下的时间

/*
*****
* 函数原型: void EC11A_FlagCheak(uint16_t dT)
* 功 能: 检测延时检测延时-2ms
* 输 入: dT : 周期
* 参 数: uint16_t dT
*****
*/
void EC11A_FlagCheak(uint16_t dT)
{
    static uint16_t T;
    T += dT;//周期加加
    if(T % 4 == 0)//计时 4ms
    {
        EC11A_Flag = 1;//进入中断
        T = 0;//计时清零
    }
}

/*
*****
* 函数原型: void Check_Knob(void)
* 功 能: 检测旋钮状态-500ms
*****
*/
void Check_Knob(void)
{
    if(EC11A_Knob1)//旋钮被转动
    {
        EC11A_Knob1--;//1S 倒计时
        if(EC11A_Knob1 == 0)//旋钮被转动
        {
            Twinkle_Time1 = 2000;//闪烁时间
        }
    }
    if(EC11A_Knob2)//旋钮被转动

```

```

    {
        EC11A_Knob2--; //1S 倒计时
        if(EC11A_Knob2 == 0) //旋钮被转动
        {
            Twinkle_Time2 = 2000; //闪烁时间
        }
    }
}

/*
*****
* 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)
{
    static uint32_t KEY_Cnt;
    static uint32_t Next_Cnt;
    uint32_t Speed_Key; //旋钮转速
    if(EC11A_Flag == 1) //进入中断
    {
        /*****右旋钮*****/
        if(GPIO_Pin == KEY2B_Pin) //右边旋钮触发
        {
            if(sys.Motor_Stop) return;

            if((HAL_GPIO_ReadPin(KEY2A_GPIO_Port, KEY2A_Pin) == 0) && (HAL_GPIO_ReadPin(KEY2B_GPIO_Port, KEY2B_Pin) == 0)) //如果向左旋转
            {
                Set_Speed += 5;
                Set_Speed = (Set_Speed > 200) ? 200 : Set_Speed; //速度不超过 200 转
            }
            else
            if((HAL_GPIO_ReadPin(KEY2A_GPIO_Port, KEY2A_Pin) == 1) && (HAL_GPIO_ReadPin(KEY2B_GPIO_Port, KEY2B_Pin) == 0)) //如果向右旋转
            {
                Set_Speed -= 5;
                Set_Speed = (Set_Speed < 40) ? 40 : Set_Speed; //速度设置小于 40 转时清零
            }
            EC11A_Knob1 = 1;
        }

        /*****左旋转*****/
        if(GPIO_Pin == KEY1B_Pin) //左边旋钮触发
        {
            if(sys.Motor_Stop) return;
            KEY_Cnt++;
            if(KEY_Cnt > 3)
            {

```



```

uint32_t First= HAL_GetTick();
Speed_Key=120000/(First-Next_Cnt);
Next_Cnt=First;
KEY_Cnt=0;
}

if((HAL_GPIO_ReadPin(KEY1A_GPIO_Port,KEY1A_Pin)==0)&&(HAL_GPIO_ReadPin(
KEY1B_GPIO_Port,KEY1B_Pin)==0))//如果向左旋转
{
    if(Speed_Key >1500)
        Set_Time += 600;
    else
        Set_Time += 60;
    Set_Time = (Set_Time > 86400) ? 86400 : Set_Time;//时间最多设定 23 小时
59 分钟
}
else
if((HAL_GPIO_ReadPin(KEY1A_GPIO_Port,KEY1A_Pin)==1)&&(HAL_GPIO_ReadPin(KEY
1B_GPIO_Port,KEY1B_Pin)==0))//如果向右旋转
{
    if(Speed_Key >1500)
        Set_Time -= 600;
    else
        Set_Time -= 60;
    Set_Time = (Set_Time < 60) ? 0 : Set_Time;//时间小于 1 分钟不设定
}
Time_State = (Set_Time < 60) ? 0 : 1;//判断是否设置了时间
EC11A_Knob2 = 1;
}
EC11A_Flag = 0;//关闭中断
}

/*****左边按钮中断*****/
if(GPIO_Pin ==KEY1_Pin)
{
    Key1_Press = 1;//按下标志被置一
}

/*****右边按钮中断*****/
if(GPIO_Pin ==KEY2_Pin)
{
    Key2_Press = 1;//按下标志被置一
}
}

/*
*****
* 函数原型: void Check_KeyState(void)
* 功    能: 按键检测
*****

```

```

*/
void Check_KeyState(void)
{
    /*******KEY1*****/
    if(Key1_Press == 1)//按钮被按下
    {
        if(HAL_GPIO_ReadPin(KEY1_GPIO_Port,KEY1_Pin)==0)//如果 KEY1 按下
            KEY1_Count++;//按下时间++
        if(HAL_GPIO_ReadPin(KEY1_GPIO_Port,KEY1_Pin)==1)//如果 KEY1 抬起
        {
            if(KEY1_Count < 200)//短按
            {
                if(sys.Run_Status == 0)
                {
                    Speed_ADDMode = 0;
                    sys.Run_Status = 1;
                    SpeedSet_Flag = 1;
                    TimeSet_Flag = 1;
                    Speed_Val.SumError = 0x5161;
                    Twinkle_Time1 = 0;
                    EC11A_Knob1 = 0;
                    Twinkle_Time2 = 0;
                    EC11A_Knob2 = 0;
                }
                else
                {
                    sys.Motor_Stop = 1;//检测电机
                    Speed_ADDMode = 2;//进入减速模式下
                }
                Beep_Time = 0.1;
            }
            else//等于 200 时再清零
            {
                KEY1_Count = 0;//按钮计数清零
                Key1_Press = 0;//按钮状态为抬起
            }
            KEY1_Count = 0;//按钮计数清零
            Key1_Press = 0;//按钮状态为抬起
        }
        if(KEY1_Count > 200 && KEY1_Count < 400)//长按
        {
            KEY1_Count = 400;//按钮计数等于 200，这样就不会在抬起再进入单击了
        }
    }
    /*******KEY2*****/
    if(Key2_Press == 1)//按钮被按下
    {
        if(HAL_GPIO_ReadPin(KEY2_GPIO_Port,KEY2_Pin)==0)//如果 KEY2 按下
            KEY2_Count++;//按下时间++
    }
}

```

---

```

if(HAL_GPIO_ReadPin(KEY2_GPIO_Port,KEY2_Pin)==1)//如果 KEY2 抬起
{
    if(KEY2_Count < 200)//短按
    {
        if(sys.Run_Status == 0)
        {
            Speed_ADDMode = 0;
            sys.Run_Status = 1;
            SpeedSet_Flag = 1;
            TimeSet_Flag = 1;
            Speed_Val.SumError = 0x5161;
            Twinkle_Time1 =0;
            EC11A_Knob1=0;
            Twinkle_Time2 =0;
            EC11A_Knob2=0;
        }
        else
        {
            sys.Motor_Stop = 1;//检测电机
            Speed_ADDMode = 2;//进入减速模式下
        }
        Beep_Time = 0.1;
        KEY2_Count = 0;//按钮计数清零
        Key2_Press = 0;//按钮状态为抬起
    }
    else//等于 200 时再清零
    {
        KEY2_Count = 0;//按钮计数清零
        Key2_Press = 0;//按钮状态为抬起
    }
}
if(KEY2_Count > 200 && KEY2_Count < 400)//长按
{
    KEY2_Count = 400;//按钮计数等于 200，这样就不会在抬起再进入单击了
}
}

#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
* 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 0.3 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

```

## SK4100 软件 V1.0

---

```
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 "Show.h"
```

```
/******全局变量声明*****/
```

```
uint16_t Twinkle_Time1,Twinkle_Time2;//闪烁时间
```

```
/******局部变量声明*****/
```

```
uint8_t Run_Flag;//运行闪烁图标
```

```
uint8_t Speed_ShowFlag,Time_ShowFlag;//速度、时间显示的标志位 0:常亮 1: 熄灭
```

```
uint8_t DOWNL_Tab[]={0x77,0x24,0x5d,0x6d,0x2e,0x6b,0x7b,0x25,0x7f,0x6f};
```

```
uint8_t UP_Tab[]={0xee,0x24,0xba,0xb6,0x74,0xd6,0xde,0xa4,0xfe,0xf6};
```

```
uint8_t DOWNR_Tab[]={0xee,0x48,0xba,0xda,0x5c,0xd6,0xf6,0x4a,0xfe,0xde};
```

```
/*
```

```
*****
```

```
* 函数原型: void Check_ShowFlag(uint16_t dT)
```

```
* 功 能: 闪烁检测
```

```
* 输 入: dT:执行周期
```

```
* 参 数: uint16_t dT
```

```
*****
```

```
*/
```

```
void Check_ShowFlag(uint16_t dT)
```

```
{
```

```
    if(sys.Run_Status)//运行时
```

```
        Run_Flag = ~Run_Flag;//运行时闪烁
```

```
    if(Twinkle_Time1 && EC11A_Knob1==0)//闪烁和没有操作旋钮时
```

```
{
```

```
    Twinkle_Time1 -= dT;//闪烁计时
```

```
    Speed_ShowFlag = ~Speed_ShowFlag;//速度闪烁
```

```
    if(Twinkle_Time1 == 0)//如果闪烁结束
```

```
{
```

```
    SpeedSet_Flag = 1;
```

```
    Beep_Time = 0.1;
```

---

```

    }
}
else
    Speed_ShowFlag = 0;//速度闪烁清零

if(Twinkle_Time2 && EC11A_Knob2==0)//闪烁和没有操作旋钮时
{
    Twinkle_Time2 -= dT;//闪烁计时
    Time_ShowFlag = ~Time_ShowFlag;//时间闪烁
    if(Twinkle_Time2 == 0)//如果闪烁结束
    {
        TimeSet_Flag = 1;
        Beep_Time = 0.1;
    }
}
else
    Time_ShowFlag = 0;//时间闪烁清零
}

/*
*****
* 函数原型: void LCD_Light(short LCD_Status)
* 功    能: 打开和关闭背光显示
* 输    入: LCD_Status: 1、打开背光 0: 关闭背光
* 参    数: short LCD_Status
*****
*/
void LCD_Light(short LCD_Status)
{
    switch(LCD_Status)
    {
        case 0 :
            __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0);//不输出 pwm
            break ;
        case 1 :
            __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 45);//不输出 pwm
            break ;
    }
}

/*
*****
* 函数原型: void Display_Set_Time(uint32_t dis_set_time)
* 功    能: 写最左边的设定时间
* 输    入: dis_set_time 左边最下面的时间
* 参    数: uint32_t dis_set_time
*****
*/
void Display_Set_Time(uint32_t dis_set_time)
{

```

```

uint8_t seg1,seg2,seg3,seg4;
seg1=0;seg2=0;seg3=0;seg4=0;;
uint8_t SH,H,SM,M;//时间的单位取值

SH=dis_set_time/3600/10;//计算十位单位的小时数
H=dis_set_time/3600%10;//计算个位单位的小时数
SM=dis_set_time%3600/60/10;//计算十分位单位的分钟数
M=dis_set_time%3600/60%10;//计算个分位单位的分钟数

seg1 = DOWNL_Tab[SH];//时间十位单位的小时数
seg2 = DOWNL_Tab[H];//时间个位单位的小时数
seg3 = DOWNL_Tab[SM];//时间十分位单位的分钟数
seg4 = DOWNL_Tab[M];//时间个分位单位的分钟数

seg2 = seg2|0x80;//设定时间冒号（下面）
// seg3 = seg3|0x80;//设定时间小数点（下面）
seg4 = seg4|0x80;//设定时间 min 文本显示

if(Time_State == 0)//没有设置时间,显示 “--:--”
{
    seg1=(seg1&0x80)|0x08;
    seg2=(seg2&0x80)|0x08;
    seg3=(seg3&0x80)|0x08;
    seg4=(seg4&0x00)|0x08;//0x00 将单位"min"去掉
}

if(Time_ShowFlag && EC11A_Knob2 == 0 && Time_State == 1)//时间到和旋钮无操作时
{
    seg1&=0x00;
    seg2&=0x00;
    seg3&=0x00;
    seg4&=0x80;
}

Write_Addr_Dat_N(0, seg1,1);
Write_Addr_Dat_N(2, seg2,1);
Write_Addr_Dat_N(4, seg3,1);
Write_Addr_Dat_N(6, seg4,1);
}

/*
*****
* 函数原型: void Display_Rel_Time(uint32_t dis_rel_time)
* 功 能: 写最左边的设定时间
* 输 入: dis_rel_time 左边最下面的时间
* 参 数: uint32_t dis_rel_time
*****
*/
void Display_Rel_Time(uint32_t dis_rel_time)
{

```



```

uint8_t seg1,seg2,seg3,seg4;
seg1=0;seg2=0;seg3=0;seg4=0;;
uint8_t SH,H,SM,M;//时间的单位取值

SH=dis_rel_time/3600/10;//计算十位单位的小时数
H=dis_rel_time/3600%10;//计算个位单位的小时数
SM=dis_rel_time%3600/60/10;//计算十分位单位的分钟数
M=dis_rel_time%3600/60%10;//计算个分位单位的分钟数

seg1 = UP_Tab[SH];//时间十位单位的小时数
seg2 = UP_Tab[H];//时间个位单位的小时数
seg3 = UP_Tab[SM];//时间十分位单位的分钟数
seg4 = UP_Tab[M];//时间个分位单位的分钟数

// seg1 = seg1|0x01;//温度符号 “℃” 图标（下面）
seg2 = seg2|0x01;//实际时间冒号（上面）
// seg3 = seg3|0x01;//实际时间小数点（上面）
seg4 = seg4|0x01;//时间图标（上面）

if(Time_State == 0)//没有设置时间,显示 “--:--”
{
    seg1=(seg1&0x01)|0x10;
    seg2=(seg2&0x01)|0x10;
    seg3=(seg3&0x01)|0x10;
    seg4=(seg4&0x01)|0x10;
}
if(sys.Run_Status == 1 && Run_Flag > 1 && DownTime_Over == 0 && Time_State ==
1)//运行时
{
    seg2&=0xFE;//实际时间冒号（上面）消失
    seg4&=0xFE;//时间图标（上面）消失
}
Write_Addr_Dat_N(54, seg1,1);
Write_Addr_Dat_N(52, seg2,1);
Write_Addr_Dat_N(50, seg3,1);
Write_Addr_Dat_N(48, seg4,1);
}

/*
*****
* 函数原型: void Display_SetSpeed(uint16_t dis_set_speed)
* 功 能: 写最右边的设定速度
* 输 入: dis_set_speed 右边最下面的速度
* 参 数: uint16_t dis_set_speed
*****
*/
void Display_Set_Speed(uint16_t dis_set_speed)
{
    uint8_t seg1,seg2,seg3,seg4;
    seg1=0;seg2=0;seg3=0;seg4=0;;

```

```

//设定速度
if(dis_set_speed>999)
{
    seg1=DOWNR_Tab[dis_set_speed/1000];
    seg2=DOWNR_Tab[dis_set_speed/100%10];
    seg3=DOWNR_Tab[dis_set_speed/10%10];
    seg4=DOWNR_Tab[dis_set_speed%10];
}
else if(dis_set_speed >99)
{
    seg1=DOWNR_Tab[0];
    seg2=DOWNR_Tab[dis_set_speed/100];
    seg3=DOWNR_Tab[dis_set_speed/10%10];
    seg4=DOWNR_Tab[dis_set_speed%10];
}
else if(dis_set_speed >9)
{
    seg1=DOWNR_Tab[0];
    seg2=DOWNR_Tab[0];
    seg3=DOWNR_Tab[dis_set_speed/10];
    seg4=DOWNR_Tab[dis_set_speed%10];
}
else
{
    seg1=DOWNR_Tab[0];
    seg2=DOWNR_Tab[0];
    seg3=DOWNR_Tab[0];
    seg4=DOWNR_Tab[dis_set_speed%10];
}

if(Speed_ShowFlag && EC11A_Knob1 == 0)//时间到和旋钮无操作时
{
    seg1&=0x00;
    seg2&=0x00;
    seg3&=0x00;
    seg4&=0x01;
}
seg4 = seg4|0x01;//rpm 文本显示

Write_Addr_Dat_N(26, seg1,1);
Write_Addr_Dat_N(28, seg2,1);
Write_Addr_Dat_N(30, seg3,1);
Write_Addr_Dat_N(32, seg4,1);
}

/*
*****
* 函数原型: void Display_RelSpeed(uint16_t dis_rel_speed)
* 功 能: 写最右边的实际速度

```

---

```

* 输入:  dis_rel_speed  右边最上面的速度
* 参数:  uint16_t dis_rel_speed
*****
*/
void Display_Rel_Speed(uint16_t dis_rel_speed)
{
    uint8_t seg1,seg2,seg3,seg4;
    seg1=0;seg2=0;seg3=0;seg4=0;;

    //设定速度
    if(dis_rel_speed>999)
    {
        seg1=UP_Tab[dis_rel_speed/1000];
        seg2=UP_Tab[dis_rel_speed/100%10];
        seg3=UP_Tab[dis_rel_speed/10%10];
        seg4=UP_Tab[dis_rel_speed%10];
    }
    else if(dis_rel_speed >99)
    {
        seg1=UP_Tab[0];
        seg2=UP_Tab[dis_rel_speed/100];
        seg3=UP_Tab[dis_rel_speed/10%10];
        seg4=UP_Tab[dis_rel_speed%10];
    }
    else if(dis_rel_speed >9)
    {
        seg1=UP_Tab[0];
        seg2=UP_Tab[0];
        seg3=UP_Tab[dis_rel_speed/10];
        seg4=UP_Tab[dis_rel_speed%10];
    }
    else
    {
        seg1=UP_Tab[0];
        seg2=UP_Tab[0];
        seg3=UP_Tab[0];
        seg4=UP_Tab[dis_rel_speed%10];
    }
    seg1 = seg1|0x01;//圆图标的尾巴

    if(!Speed_CwIcn)
    {
        seg2 = seg2|0x01;//顺时针的三角形
        seg3 = seg3|0x00;//逆时针的三角形
    }
    else
    {
        seg2 = seg2|0x00;//顺时针的三角形
        seg3 = seg3|0x01;//逆时针的三角形
    }
}

```

```

// seg4 = seg4|0x01;//加热图标
// if(sys.Run_Status == 1 && Run_Flag > 1 && Set_Speed > 0 && Rel_Speed)//运行时
if(sys.Run_Status == 1 && Run_Flag > 1)//运行时
{
    seg1&=0xfe;//圆图标的尾巴消失
    seg2&=0xfe;//顺时针的三角形消失
    seg3&=0xfe;//逆时针的三角形消失
}
Write_Addr_Dat_N(46, seg1,1);
Write_Addr_Dat_N(44, seg2,1);
Write_Addr_Dat_N(42, seg3,1);
Write_Addr_Dat_N(40, seg4,1);
}

/*
*****
* 函数原型: void Deal_Speed(void)
* 功 能: 速度显示处理
*****
*/
void Deal_Speed(void)
{
    /*****SpeedL1_ADD_Mode*****/
    if(sys.Run_Status == 1)//启动的情况下
    {
        if(Speed_ADDMode == 0)//在电机控制中, 速度未处理
        {
            if(Ctrl_Speed >= Display_RelSpeed)//控制速度大于实际速度
            {
                Speed_New = 0;//现在的速度清零
                Speed_Last = 0;//之前的速度清零
                Speed_ADDMode = 1;//进入加速模式下
            }
            else if(Ctrl_Speed < Display_RelSpeed)//控制速度小于实际速度
            {
                Speed_New = 0;//现在的速度清零
                Speed_Last = Display_RelSpeed;//之前的速度等于当前显示速度
                Speed_ADDMode = 2;//进入减速模式下
            }
        }
        if(Speed_ADDMode == 1)//在进入加速模式下
        {
            if(sys.Motor_Stop == 0)
            {
                if((Rel_Speed) >= Ctrl_Speed)//实际速度大于等于控制速度
                {
                    Speed_ADDMode = 3;//进入稳定模式
                    return;
                }
            }
        }
    }
}

```

```

    }
}
Speed_New = (Rel_Speed); //记录当前速度
if(Speed_New > Speed_Last) //当前速度大于上一次速度
    Display_RelSpeed = Speed_New; //显示当前速度
else //当前速度小于上一次速度
{
    Display_RelSpeed = Speed_Last; //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
    Speed_New = Speed_Last; //将上一次速度赋值给当前速度
}
Speed_Last = Speed_New; //将当前速度保存
}
else if(Speed_ADDMode == 2) //速度下降模式下
{
    if(sys.Motor_Stop == 0)
    {
        if((Rel_Speed) <= Ctrl_Speed) //实际速度小于等于控制速度
        {
            Speed_ADDMode = 3; //稳定模式
            return;
        }
    }
    Speed_New = (Rel_Speed); //记录当前速度
    if(Speed_New < Speed_Last) //当前速度小于上一次速度
        Display_RelSpeed = Speed_New; //显示当前速度
    else //当前速度大于上一次速度
    {
        Display_RelSpeed = Speed_Last; //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
        Speed_New = Speed_Last; //将上一次速度赋值给当前速度
    }
    Speed_Last = Speed_New; //将当前速度保存
}
}
else if(Speed_ADDMode == 3) //速度稳定模式下
{
    Display_RelSpeed = Ctrl_Speed; //显示控制速度
// if(Ctrl_Speed > 400)
// {
//     if(Rel_Speed-Ctrl_Speed>110)
//     {
//         Speed_Tz = 1;
//         Speed_ADDMode = 2;
//         sys.Motor_Stop = 1;
//     }
// }
// else if(Ctrl_Speed <= 400 && Ctrl_Speed > 230)
// {
//     if(Rel_Speed-Ctrl_Speed>40)

```

---

```

//      {
//          Speed_Tz = 1;
//          Speed_ADDMode = 2;
//          sys.Motor_Stop = 1;
//      }
//  }
    }
else
{
    Speed_New = 0; // 现在的速度清零
    Speed_Last = 0; // 之前的速度清零
    Speed_ADDMode = 0;
    Display_RelSpeed = 0;
}
}

/*
*****
* 函数原型: void Show_Display(void)
* 功    能: 显示屏幕内容
*****
*/
void Show_Display(void)
{
    Deal_Speed();
    Display_SetSpeed = Set_Speed;
    Display_SetTime = Set_Time; // 显示设定时间
    Display_CtrlTime = Ctrl_Time + 59; // 显示实际时间

    Display_Set_Time(Display_SetTime);
    Display_Rel_Time(Display_CtrlTime);
    Display_Set_Speed(Display_SetSpeed);
    Display_Rel_Speed(Display_RelSpeed);
}
#include "SetVal.h"

/*****全局变量声明*****/
uint8_t SpeedSet_Flag, TimeSet_Flag; // 检测速度和时间设置标志位

/*
*****
* 函数原型: void Check_Set(void)
* 功    能: 检测设置
*****
*/
void Check_Set(void)
{
    if(Twinkle_Time1 == 0 && SpeedSet_Flag)

```

---

```

    {
        Ctrl_Speed = Set_Speed;
        Speed = Set_Speed;
        if(Speed_ADDMode != 0)//假如工位只有在启动并且设置了速度的情况下不等于 0,
        不在未处理模式下
            Speed_ADDMode = 0;//进入未处理, 判断加速还是减速
        Param.Speed = Set_Speed;//转速
        Save_Param_En = 1;
        SpeedSet_Flag = 0;
    }
    if(Twinkle_Time2 == 0 && TimeSet_Flag)
    {
        Ctrl_Time = Set_Time;
        Time = Set_Time;
        Param.Time = Set_Time;//时间
        Save_Param_En = 1;
        TimeSet_Flag = 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 TIM1CH1_CAPTURE_STA = 0;//捕获周期数
uint32_t TIM1CH1_CAPTURE_VAL;//捕获计数值
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_IC_Start_IT(&htim1, TIM_CHANNEL_3);//motor 输入捕获
    HAL_TIM_Base_Start_IT(&htim1);
}

/*
*****
* 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_Cnt++;//每 50ms 进入

```

---

```

    if(Speed_Cnt >= 10)//0.5s 发现没出发输入捕获
    {
        Rel_Speed = 0;//将速度清零
        Speed_Cnt = 0;//计数清零
    }
}

/*
*****
* 函数原型: void TIM1_Poll(void)
* 功    能: 检测状态
*****
*/
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
        TIM1CH1_CAPTURE_STA=0;//捕获的周期清零
    }
}

/*
*****
* 函数原型: void Check_Speed(void)
* 功    能: 检测速度是否停止
*****
*/
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Instance == TIM1)
    {
        if(P_Status)//捕获周期计数状态:开启
        {
            TIM1CH1_CAPTURE_STA++;//捕获周期数++
        }
    }
}

/*
*****
* 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
        TIM1CH1_CAPTURE_VAL = HAL_TIM_ReadCapturedValue(&htim1,
TIM_CHANNEL_3);//获取当前捕获计数值
        Capture = TIM1CH1_CAPTURE_STA;//定时器的周期值
        Capture *= 50;//一个周期 50us (48000000/12/200 = 0.00005us)
        Capture += TIM1CH1_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 / 5;//rpm
        if((rel - Rel_Speed < 300 && rel - Rel_Speed > 0) || (Rel_Speed - rel < 300 &&
Rel_Speed - rel > 0))
            Rel_Speed = rel;
        P_Status = 0;//捕获周期计数状态:0 关闭
        Speed_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
    {
        TIM1CH1_CAPTURE_STA = 0;//清除周期计数
        TIM1CH1_CAPTURE_VAL = 0;//清楚捕获寄存器
        CAPTURE_First = 1;//已捕获第一个上升沿
        CAPTURE_Status = 0;//捕获结束
        P_Status = 1;//捕获周期计数
    }
}
}
#include "PID.h"

/*****结构体*****/
PID_val_t Speed_Val;//pid 数据结构
PID_arg_t Speed_Arg;//pid 数据系数

/*
*****
* 函数原型: void PID_Init(void)
* 功 能: pid 系数初始化
*****
*/
void PID_Init(void)
{
    Speed_Arg.Kp= 0.0014;
    Speed_Arg.Ki= 0.0012;
    Speed_Arg.Kd= 0.004;
}

```

```

/*
*****
* 函数原型: void PID_Speed(
    uint16_t Expect, //期望值 (设定值)
    uint16_t Feedback, //反馈值 (实际值)
    PID_arg_t *pid_arg, //PID 参数结构体
    PID_val_t *pid_val) //PID 数据结构体
* 功 能: PID 控制
* 输 入: Expect, //期望值 (设定值)
    Feedback, //反馈值 (实际值)
    PID_arg_t *pid_arg, //PID 参数结构体
    PID_arg_t *pid_arg, //PID 参数结构体
*****
*/
void PID_Speed(
    uint16_t Expect, //期望值 (设定值)
    uint16_t Feedback, //反馈值 (实际值)
    PID_arg_t *pid_arg, //PID 参数结构体
    PID_val_t *pid_val) //PID 数据结构体

{
    pid_val->Error = Expect - Feedback; //当前误差
    if(pid_val->Error > 75)
        pid_val->Error = 75;
    if(pid_val->Error < -200)
    {
        pid_val->Error = -200;
    }
    pid_val->SumError = pid_val->Error + pid_val->SumError; //误差和
    pid_val->D_Error = pid_val->Error - pid_val->LastError; //误差偏差
    pid_val->LastError = pid_val->Error; //保存上一次误差
    pid_val->Out =
pid_arg->Kp*pid_val->Error+pid_arg->Ki*pid_val->SumError+pid_arg->Kd*pid_val->D_Error;
    if(pid_val->Out < 1)
        pid_val->Out = 1;
    if(pid_val->Out > 1 & pid_val->Out < 190)
        pid_val->Out = pid_val->Out;
}
#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.type = Type;
    Param.Time = 0;//时间
    Param.Speed = 100;//转速
    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.type != Type)
    {
        Param_Reset();
        Set_Time = Param.Time;//时间
        Set_Speed = Param.Speed;//转速
        Save_Param_En = 1;
    }
    else
    {
        Set_Time = Param.Time;//时间
        Set_Speed = Param.Speed;//转速
    }

    //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 "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();
}

```

---

```

static void Loop_2ms(void)//2ms 执行一次
{
    EC11A_FlagCheak(2);//旋钮检测延时
}

static void Loop_10ms(void)//10ms 执行一次
{
    Check_KeyState();//按键检测
}

static void Loop_50ms(void)//50ms 执行一次
{
    Check_Set();//检测设置
    Motor_Ctrl();//电机控制
}

static void Loop_100ms(void)//100ms 执行一次
{
    Cheak_TimeDown(100);//时间倒计时检测
    Buzzer_Status(0.1f);//蜂鸣器检测
    Param_Save_Overtime(0.1f);//保存标志位置 1， 0.5s 后保存
    Check_MotorStop(0.1f);//检测电机是否停止
}

static void Loop_500ms(void)//500ms 执行一次
{
    Check_ShowFlag(500);//屏幕闪烁检测
    Check_Knob();//旋钮旋动检测
}

static void Loop_1S(void)//1S 执行一次
{
    Check_Speed(1.0f);//检测速度是否停止
}

void Sys_Loop(void)
{
    if(T_cnt_2ms >= 2) {
        Loop_2ms();
        T_cnt_2ms = 0;
    }
    if(T_cnt_10ms >= 10) {
        Loop_10ms();
        T_cnt_10ms = 0;
    }
    if(T_cnt_50ms >= 50) {
        Loop_50ms();
        T_cnt_50ms = 0;
    }
    if(T_cnt_100ms >= 100) {

```

```

        Loop_100ms();
        T_cnt_100ms = 0;
    }
    if(T_cnt_500ms >= 500) {
        Loop_500ms();
        T_cnt_500ms = 0;
    }
    if(T_cnt_1S >= 1000) {
        Loop_1S();
        T_cnt_1S = 0;
    }
}
#include "Ctrl_Motor.h"

/*
*****
* 函数原型: void Motor_Ctrl(void)
* 功    能: 电机控制
*****
*/
void Motor_Ctrl(void)
{
    if(sys.Run_Status == 1)//启动
    {
        /*****Speed_L1*****/
        if(Ctrl_Speed && ((DownTime_Over == 0)|| (Ctrl_Time)))//速度大于 0 和定时器没有
        结束
        {
            if(sys.Motor_Stop)
            {
                PID_Speed(0,Rel_Speed,&Speed_Arg,&Speed_Val);//电机 PID 控制
                PWM = Speed_Val.Out;//pid 输出
            }
            else
            {
                PID_Speed((Ctrl_Speed),Rel_Speed,&Speed_Arg,&Speed_Val);//电机 PID 控
                制
                PWM = Speed_Val.Out;//pid 输出
            }
        }
        else
        {
            sys.Motor_Stop = 1;//检测电机
            PID_Speed(0,Rel_Speed,&Speed_Arg,&Speed_Val);//电机 PID 控制
            PWM = Speed_Val.Out;//pid 输出
        }
    }
    else
    {
        Speed_Val.SumError = 0;
    }
}

```

---

```

        Speed_Val.Out = 0;
        PWM = 0;//pwm 不输出
    }
}

/*
*****
* 函数原型: void Check_MotorStop(float dT)
* 功    能: 检测电机是否停止, 停止后开盖
*****
*/
void Check_MotorStop(float dT)
{
    static float T;
    if(sys.Motor_Stop)
    {
        if(Speed_Cw)
        {
            if(Rel_Speed <= 100)
            {
                T += dT;
                if(T > 1.0f)
                {
                    if(!Speed_CwIcn)
                    {
                        HAL_GPIO_WritePin(CW_GPIO_Port,          CW_Pin,
GPIO_PIN_SET);//逆时针
                        Speed_ADDMode = 0;
                        Speed_CwIcn = 1;
                    }
                    else
                    {
                        HAL_GPIO_WritePin(CW_GPIO_Port,          CW_Pin,
GPIO_PIN_RESET);//顺时针
                        Speed_ADDMode = 0;
                        Speed_CwIcn = 0;
                    }
                    sys.Motor_Stop = 0;
                    Speed_Cw = 0;
                    Speed_Tz = 0;
                    T = 0;
                }
            }
        }
        // else if(Speed_Tz)
        // {
        //     if(Rel_Speed <= 100)
        //     {
        //         T += dT;
        //         if(T > 1.0f)

```

---

```

//          {
//          Speed_ADDMode = 0;
//          sys.Motor_Stop = 0;
//          Speed_Tz = 0;
//          Speed_Cw = 0;
//          T = 0;
//          }
//      }
//  }
    else if(Rel_Speed <= 100)
    {
        T += dT;
        if(T > 1.0f)
        {
            PWM = 0;
            SpeedSet_Flag = TimeSet_Flag = 1;//设置变量
            sys.Run_Status = 0;//关闭
            sys.Motor_Stop = 0;//电机已经停止
            Speed_Cw = 0;
            Speed_Tz = 0;
            T = 0;
        }
    }
}

/*
*****
* 函数原型: void Motor_Init(void)
* 功    能: 电机初始化
*****
*/
void Motor_Init(void)
{
    HAL_GPIO_WritePin(MO_GPIO_Port, MO_Pin, GPIO_PIN_SET);//使能电机
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);//motor PWM 输出
}

#include "Ctrl_DownTime.h"

/*
*****
* 函数原型: void Cheak_TimeDown(uint16_t dT)
* 功    能: 时间倒计时检测
* 输    入: dT:执行周期
* 参    数: uint16_t dT
*****
*/
void Cheak_TimeDown(uint16_t dT)
{

```



---

```

static uint16_t T;
if(DownTime_Over==1)//倒计时结束
{
    DownTime_Over=0;//将时间标志清零
    Speed_ADDMode = 2;//进入减速模式下
    Beep_Flash = 5;//蜂鸣器响 5 下
}
if(sys.Run_Status && sys.Motor_Stop == 0)//启动系统
{
    T += dT;
    if(T == 1000)//1S
    {
        if(Time_State && DownTime_Over == 0 && Rel_Speed)//如果实际时间显示和
        倒计时没有结束的标志还在
        {
            Ctrl_Time--;//控制时间--
            if(Ctrl_Time == 0)
                DownTime_Over= 1;//time1 倒计时结束
        }
        T = 0;//周期清零
    }
}
else
{
    DownTime_Over=0;//将时间标志清零
}
}
#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;

    /*****背光源亮度控制*****/
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);

    /*****电机初始化*****/
    Motor_Init();
    PID_Init();
    /*****编码器初始化*****/
    Encoder_Init();

```

---

```

/*****初始化 lcd 屏幕*****/
Lcd_Init();//初始化
LCD_Light(LCD_ON);//打卡背光
Lcd_All();//显示全部内容
HAL_Delay(1000);//延时 1S
Lcd_Clr();//清屏

/*****系统参数初始化*****/
Param_Read();
Speed = Ctrl_Speed = Set_Speed;
Time = Ctrl_Time = Set_Time;
Time_State = (Set_Time < 60) ? 0 : 1;//判断是否设置了时间
Beep_Time = 0.1;

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

_sys_sys;//系统初始化检测

int Ctrl_Speed;//控制速度（期望值）
int Set_Speed;//设置速度
int Rel_Speed;//实际速度
int Display_SetSpeed;//用于显示设置速度
int Display_RelSpeed;//用于显示实际速度
int Speed;//临时存储速度
uint8_t Speed_Cnt;//输入捕获进入的次数
int Speed_New;//用于速度显示处理更新
int Speed_Last;//用于速度显示处理存储
int Speed_ADDMode;//用于判断速度时上升还是下降
uint8_t Speed_CwIcn;//电机转动方向
uint8_t Speed_Cw;//电机转动方向
uint8_t Speed_Tz;//转子跳子

int32_t Ctrl_Time;//控制时间（期望值）
int32_t Set_Time;//设置时间
int32_t Display_SetTime;//显示时间
int32_t Display_CtrlTime;//显示时间
int32_t Time;//临时存储值
uint8_t Time_State;//时间的状态
uint8_t DownTime_Over;//时间倒计时结束

```