

MC3100 源程序

```

/*****
*
*???:?????
*?:
*?:2013/09/07
*?:
*?:
*****/

/
#include "ht1623.h"
/*****
*
* ?    ? : delay(uint i)
* ?    ? : 5us??
* ?    ? :
* ? ? ? : ?
*
* ????:
* ??    ??    ??    ???????
* -----
*****/

/
void delay(uint16_t time)
{
    unsigned char a;
    for(a=100;a>0;a--);
}

void write_mode(unsigned char MODE)    //写入模式,数据 or 命令
{
    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);
}

/*
*   LCD 命令写入函数
*   入口:cbyte ,控制命令字
*   出口:void
*/
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);
}

/*
*   LCD 地址写入函数

```

```
* 入口:cbyte,地址
* 出口:void
*/
void write_address(unsigned char Abyte)
{
    unsigned char i = 0;
    Abyte = Abyte << 1;

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

/*
* LCD 数据写入函数
* 入口:Dbyte,数据
* 出口:void
*/
void write_data_8bit(unsigned char Dbyte)
{
    int i = 0;

    for (i = 0; i < 8; i++)
    {
        Clr_1625_Wr;
        //Delay_us(10);
        if ((Dbyte >> (7 - i)) & 0x01)
        {
            Set_1625_Dat;
        }
        else
        {
            Clr_1625_Dat;
        }
        delay(10);
        Set_1625_Wr;
        delay(10);
    }
}
```

```

    }
}

void write_data_4bit(unsigned char Dbyte)
{
    int i = 0;

    for (i = 0; i < 4; i++)
    {
        Clr_1625_Wr;
        //Delay_us(10);
        if ((Dbyte >> (3 - i)) & 0x01)
        {
            Set_1625_Dat;
        }
        else
        {
            Clr_1625_Dat;
        }
        delay(10);
        Set_1625_Wr;
        delay(10);
    }
}

////////////////////////////////////接口函数
/*
*   LCD 初始化，对 lcd 自身做初始化设置
*   入口:void
*   出口:void
*/
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;
}

/*
 * LCD 清屏函数
 * 入口:void
 * 出口:void
 */
void lcd_clr(void)
{
    write_addr_dat_n(0x0, 0x00, 50);
}

/*
 * LCD 全显示函数
 * 入口:void
 * 出口:void
 */
void lcd_all(void)
{
    write_addr_dat_n(0x0, 0xFF, 60);
}

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 "key.h"
#include "user.h"

extern uint16_t Rpm, Time_SUM;
uint16_t Scan_Status, KEY_Flag, RUN_Status;
uint16_t cur, Set_Flag, Set_Count, Key_Count, Key1_Count;
extern uint8_t Set_Flag1, Set_Flag2;
extern uint8_t Time_Status;

```

```

extern uint8_t Sys_Mode;
extern uint8_t Point_Flag;
uint16_t MAX_RPM;
uint8_t KEY1_Pin_ON;
extern uint16_t PWM;
extern uint16_t BEEP_Count,BEEP_Close;
uint16_t full_rpm;
uint16_t full_Convert_Set;
extern uint8_t rpm_flag;
uint16_t save_time,save_rpm;
void stop(void);
/*****
*
* 名 称: Key_Scan(GPIO_TypeDef* GPIOx,uint16_t GPIO_Pin)
* 功 能: 按键扫描
* 参 数: PIO_TypeDef* GPIOx,uint16_t GPIO_Pin
* 返 回: KEY_ON/KEY_OFF
*
* 修改历史:
* 改动原因:
* -----
*****/
/
uint8_t Key_Scan(GPIO_TypeDef* GPIOx,uint16_t GPIO_Pin)
{

    if(HAL_GPIO_ReadPin (GPIOx,GPIO_Pin) == 0 )
    {

        BEEP();
        BEEP_Close=200;
        if(KEY_Flag==0)
        {
            KEY_Flag=1;
            return KEY_ON;
        }
        uint32_t cur_time = HAL_GetTick();
        static uint32_t start_time = 0;
        if(start_time == 0)
            start_time = cur_time;

        if(cur_time - start_time < cur)
            return KEY_OFF;

        if(HAL_GPIO_ReadPin (GPIOx,GPIO_Pin) == 0)
        {

```

```

        Scan_Status++;
        if(Scan_Status>3)
            cur=8;
            start_time = cur_time;
            return KEY_ON;
    }

}
else
{
    if((HAL_GPIO_ReadPin (GPIOB,KEY2_Pin)
==1)&&(HAL_GPIO_ReadPin (GPIOB,KEY3_Pin) ==1 ) )
    {
        if(HAL_GPIO_ReadPin (GPIOB,KEY1_Pin) ==1)
        {
            KEY1_Pin_ON=0;
        }
        Scan_Status=0;
        cur=400;
        return KEY_OFF;
    }
}
return KEY_OFF;
}

/*****
*
* 名 称: Key_Handle(void)
* 功 能: 按键处理
* 参 数: PIO_TypeDef* GPIOx,uint16_t GPIO_Pin
* 返 回值:
*
* 修改历史:
* 改动原因:
* -----
*****/
/

void Key_Handle(void)
{
    if(( Key_Scan(GPIOB,KEY1_Pin) == KEY_ON))
    {
        // BEEP();
        Set_Flag++;
        if(Set_Flag ==1)
        {
            Set_Flag1=1;

```

```

        Set_Flag2=0;
    }
    else if(Set_Flag ==2)
    {
        Set_Flag2 =1;
        Set_Flag1=0;
    }

//Set_Flag1=0;
//Set_Flag2=0;
Set_Count=10;
if(Set_Flag>2)
{
    Set_Flag=1;
    Set_Flag1=1;
    Set_Flag2=0;
}
KEY1_Pin_ON++;

if(KEY1_Pin_ON>3)
{
    HAL_GPIO_WritePin(BEEP_GPIO_Port,    BEEP_Pin,
GPIO_PIN_SET);

    BEEP_Count=680;
    BEEP_Close=200;
    if(Sys_Mode==Sys_RPM)
    {
        Sys_Mode=Sys_RCF;
        if(Rpm>7000)
        Rpm=7000;
        Point_Flag=0;
        Set_Flag=0;
    }
    else
    {
        Sys_Mode=Sys_RPM;
        Point_Flag=0;
        Set_Flag=0;
    }
    KEY1_Pin_ON=0;

}
Key1_Count=5000;

}

if ( (Key_Scan(GPIOB,KEY2_Pin) == KEY_ON))//加键
{

```

```

if(Set_Flag==1)
{
    if(Sys_Mode==Sys_RPM)//rpm 模式
        MAX_RPM=7000;
    else if(Sys_Mode==Sys_RCF)//rcf 模式
        MAX_RPM=7000;
    Rpm=Rpm+500;
    if(Rpm>MAX_RPM)
        Rpm=MAX_RPM;

}
else if(Set_Flag==2)
{
    if(Time_Status ==0)
        Time_SUM +=1;
    else
        Time_SUM +=60;
    if(Time_SUM>5940)
        Time_SUM=5940;
}
if(RUN_Status==Sys_STOP)
    Set_Count=10;//按键设置计时
else
    Set_Count=3;
Key_Count=3;//按键加减计时

//PWM=Rpm/30;
switch(Rpm)
{
    case 1000: full_Convert_Set=6;
    break;
    case 1500: full_Convert_Set=7;
    break;
    case 2000: full_Convert_Set=9;
    break;
    case 2500: full_Convert_Set=14;
    break;
    case 3000: full_Convert_Set=17;
    break;
    case 3500: full_Convert_Set=21;
    break;
    case 4000: full_Convert_Set=27;
    break;
    case 4500: full_Convert_Set=33;
    break;
    case 5000: full_Convert_Set=40;
    break;
    case 5500: full_Convert_Set=48;
    break;
}

```

```

        case 6000: full_Convert_Set=54;
        break;
        case 6500: full_Convert_Set=60;
        break;
        case 7000:full_Convert_Set=70;
        break;
        default: full_Convert_Set=0;

    }

}

if ( (Key_Scan(GPIOB,KEY3_Pin) == KEY_ON))//减键
{

    if(Set_Flag==1)
    {
        Rpm=Rpm-500;
        if(Rpm<1000)
            Rpm=1000;
    }
    else if(Set_Flag==2)
    {
        if(Time_SUM<61)
        {
            if(Time_SUM>0)
                Time_SUM -=1;
        }
        else
            Time_SUM -=60;
        if(Time_SUM<10)
            Time_SUM=10;
    }
    if(RUN_Status==Sys_STOP)
        Set_Count=10;//按键设置计时
    else
        Set_Count=3;
    Key_Count=3;//按键加减计时
    //PWM=Rpm/30;
    switch(Rpm)
    {
        case 1000: full_Convert_Set=6;
        break;
        case 1500: full_Convert_Set=7;
        break;
        case 2000: full_Convert_Set=9;
        break;
        case 2500: full_Convert_Set=14;
        break;
    }
}

```

```

        case 3000: full_Convert_Set=17;
        break;
        case 3500: full_Convert_Set=21;
        break;
        case 4000: full_Convert_Set=27;
        break;
        case 4500: full_Convert_Set=33;
        break;
        case 5000: full_Convert_Set=40;
        break;
        case 5500: full_Convert_Set=48;
        break;
        case 6000: full_Convert_Set=54;
        break;
        case 6500: full_Convert_Set=60;
        break;
        case 7000:full_Convert_Set=70;
        break;
        default: full_Convert_Set=0;

    }

}

// if ( (Key_Scan(GPIOB,KEY_T_Pin) == KEY_ON))
// {
//     if(Sys_Mode==Sys_RPM)
//     {
//         RUN_Status =Sys_RUN;
//     }
// }
//
// if ( (Key_Scan(GPIOB,KEY4_Pin) == KEY_ON))
// {
//     if(RUN_Status ==Sys_RUN)
//     {
//         RUN_Status =Sys_Down;
//         rpm_flag=1;
//         //stop();
//     }
//     else
//     {
//         //if(Sys_Mode==Sys_RCF)
//         RUN_Status =Sys_RUN;
//         save_time=Time_SUM;
//         save_rpm=Rpm;
//     }
// }

```

```

        Set_Flag=0;
        full_rpm=Rpm;
        switch(Rpm)
        {
            case 1000: full_Convert_Set=6;
            break;
            case 1500: full_Convert_Set=7;
            break;
            case 2000: full_Convert_Set=9;
            break;
            case 2500: full_Convert_Set=14;
            break;
            case 3000: full_Convert_Set=17;
            break;
            case 3500: full_Convert_Set=21;
            break;
            case 4000: full_Convert_Set=27;
            break;
            case 4500: full_Convert_Set=33;
            break;
            case 5000: full_Convert_Set=40;
            break;
            case 5500: full_Convert_Set=48;
            break;
            case 6000: full_Convert_Set=54;
            break;
            case 6500: full_Convert_Set=60;
            break;
            case 7000:full_Convert_Set=70;
            break;
            default: full_Convert_Set=0;

        }

    }

    Set_Flag1=0;
    Set_Flag2=0;
}

}

void stop(void)
{
    RUN_Status =Sys_STOP;
    // Time_SUM=30;
    // if(Sys_Mode==Sys_RPM)//点动模式
    // Rpm=3000;
    // else if(Sys_Mode==Sys_RCF)//连续模式
    // Rpm=1500;

```

```

}
#include "user.h"
#include "ht1623.h"
#include "tim.h"
uint8_t Sys_Mode;//系统运行模式
uint8_t Cover_Status;

extern uint8_t Time_Status;
extern uint16_t Rpm,Time_SUM,Key_Count;
extern uint16_t cur,KEY_Flag;
extern uint16_t RUN_Status;
uint8_t Point_Flag;
uint16_t BEEP_Count,BEEP_Close;
void BEEP(void)
{
    if(BEEP_Close==0)
    {
        HAL_GPIO_WritePin(BEEP_GPIO_Port, BEEP_Pin, GPIO_PIN_SET);

        BEEP_Count=1000;
    }
}

void Sys_Init(void)
{
    HAL_GPIO_WritePin(BEEP_GPIO_Port, BEEP_Pin, GPIO_PIN_RESET);
    __HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_2,50);
    HAL_TIM_PWM_Stop(&htim1, TIM_CHANNEL_1);
    Sys_Mode=Sys_RPM;
    RUN_Status=Sys_STOP;
    Point_Flag=0;
    Time_Status =0;
    KEY_Flag=0;
    Key_Count=0;
    Time_SUM=300;
    Rpm=7000;
    cur=400;
    BEEP_Close=0;
    lcd_all();
    //HAL_Delay (1000);
    BEEP();

    lcd_clr();
    lcd_init();
}
/**

```

```

* File Name           : TIM.c
* Description          : This file provides code for the configuration
*                       of the TIM instances.

*****

* @attention
*
* <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
* All rights reserved.</center></h2>
*
* This software component is licensed by ST under BSD 3-Clause license,
* the "License"; You may not use this file except in compliance with the
* License. You may obtain a copy of the License at:
*
*                       opensource.org/licenses/BSD-3-Clause
*

*****

*/

/* Includes -----*/
#include "tim.h"

/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

TIM_HandleTypeDef htim1;

/* TIM1 init function */
void MX_TIM1_Init(void)
{
    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
    TIM_MasterConfigTypeDef sMasterConfig = {0};
    TIM_OC_InitTypeDef sConfigOC = {0};
    TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

    htim1.Instance = TIM1;
    htim1.Init.Prescaler = 32-1;
    htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
    htim1.Init.Period = 100-1;
    htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
    htim1.Init.RepetitionCounter = 0;
    htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
    if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
    {
        Error_Handler();
    }
    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
    if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
    {

```

```

    Error_Handler();
}
if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
{
    Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
{
    Error_Handler();
}
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 0;
sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) !=
HAL_OK)
{
    Error_Handler();
}
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) !=
HAL_OK)
{
    Error_Handler();
}
sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
sBreakDeadTimeConfig.DeadTime = 0;
sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
sBreakDeadTimeConfigAutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
{
    Error_Handler();
}
HAL_TIM_MspPostInit(&htim1);

}

void HAL_TIM_Base_MspInit(TIM_HandleTypeDef* tim_baseHandle)
{
    if(tim_baseHandle->Instance==TIM1)
    {
        /* USER CODE BEGIN TIM1_MspInit 0 */

```

```

/* USER CODE END TIM1_MspInit 0 */
/* TIM1 clock enable */
__HAL_RCC_TIM1_CLK_ENABLE();

/* TIM1 interrupt Init */
HAL_NVIC_SetPriority(TIM1_BRK_UP_TRG_COM_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);
HAL_NVIC_SetPriority(TIM1_CC_IRQn, 0, 0);
HAL_NVIC_EnableIRQ(TIM1_CC_IRQn);
/* USER CODE BEGIN TIM1_MspInit 1 */

/* USER CODE END TIM1_MspInit 1 */
}
}
void HAL_TIM_MspPostInit(TIM_HandleTypeDef* timHandle)
{

    GPIO_InitTypeDef GPIO_InitStruct = {0};
    if(timHandle->Instance==TIM1)
    {
        /* USER CODE BEGIN TIM1_MspPostInit 0 */

        /* USER CODE END TIM1_MspPostInit 0 */

        __HAL_RCC_GPIOA_CLK_ENABLE();
        /**TIM1 GPIO Configuration
        PA8      -----> TIM1_CH1
        PA9      -----> TIM1_CH2
        */
        GPIO_InitStruct.Pin = GPIO_PIN_8|GPIO_PIN_9;
        GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
        GPIO_InitStruct.Pull = GPIO_NOPULL;
        GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
        GPIO_InitStruct.Alternate = GPIO_AF2_TIM1;
        HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

        /* USER CODE BEGIN TIM1_MspPostInit 1 */

        /* USER CODE END TIM1_MspPostInit 1 */
    }

}

void HAL_TIM_Base_MspDeInit(TIM_HandleTypeDef* tim_baseHandle)
{

    if(tim_baseHandle->Instance==TIM1)
    {
        /* USER CODE BEGIN TIM1_MspDeInit 0 */

```

```

/* USER CODE END TIM1_MspDeInit 0 */
/* Peripheral clock disable */
__HAL_RCC_TIM1_CLK_DISABLE();

/* TIM1 interrupt Deinit */
HAL_NVIC_DisableIRQ(TIM1_BRK_UP_TRG_COM_IRQn);
HAL_NVIC_DisableIRQ(TIM1_CC_IRQn);
/* USER CODE BEGIN TIM1_MspDeInit 1 */

/* USER CODE END TIM1_MspDeInit 1 */
}
}

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */

/***** (C) COPYRIGHT STMicroelectronics *****/
/* USER CODE BEGIN Header */
/**

*****
* @file           : main.c
* @brief          : Main program body
*****
* @attention
*
* <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
* All rights reserved.</center></h2>
*
* This software component is licensed by ST under BSD 3-Clause license,
* the "License"; You may not use this file except in compliance with the
* License. You may obtain a copy of the License at:
*
*                 opensource.org/licenses/BSD-3-Clause
*
*****
*/
/* USER CODE END Header */

/* Includes -----*/
#include "main.h"
#include "tim.h"
#include "gpio.h"

/* Private includes -----*/
/* USER CODE BEGIN Includes */
#include "ht1623.h"

```

```

#include "lcd.h"
#include "user.h"
#include "key.h"
/* USER CODE END Includes */

/* Private typedef -----*/
/* USER CODE BEGIN PTD */
extern uint16_t Rpm,Time_SUM,RUN_Status;
extern uint8_t Set_Flag1,Set_Flag2,rpm_flag;
/* USER CODE END PTD */

/* Private define -----*/
/* USER CODE BEGIN PD */
void PWM_RPM_Convert(void);
/* USER CODE END PD */

/* Private macro -----*/
/* USER CODE BEGIN PM */
#define speedx100 1
/* USER CODE END PM */

/* Private variables -----*/

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */
extern  uint8_t cnt;
extern  uint16_t BEEP_Count,BEEP_Close;
/* USER CODE END PFP */

/* Private user code -----*/
/* USER CODE BEGIN 0 */
uint32_t WriteFlashData = 0x12345678;
uint32_t addr = 0x0807E000;
uint16_t  Rpm_Cnt,PWM;
extern uint16_t Set_Flag,Set_Count,Key_Count,Key1_Count;
extern uint8_t Cover_Status;
uint16_t Convert_Set;
/* USER CODE END 0 */

/**
 * @brief The application entry point.
 * @retval int
 */
int main(void)

```

```

{
    /* USER CODE BEGIN 1 */

    /* USER CODE END 1 */

    /* MCU Configuration-----*/

    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
    HAL_Init();

    /* USER CODE BEGIN Init */

    /* USER CODE END Init */

    /* Configure the system clock */
    SystemClock_Config();

    /* USER CODE BEGIN SysInit */

    /* USER CODE END SysInit */

    /* Initialize all configured peripherals */
    MX_GPIO_Init();
    MX_TIM1_Init();
    /* USER CODE BEGIN 2 */
    HAL_TIM_Base_Start_IT(&htim1);

    Sys_Init();
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
    // PWM=0;
    rpm_flag=1;
    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
    while (1)
    {
        //HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
        Cover_Status=HAL_GPIO_ReadPin (KEY_UP_GPIO_Port,KEY_UP_Pin);
        if(RUN_Status ==Sys_RUN)
        {
            PWM=Rpm/100;

            if(Cover_Status==0)
            {
                PWM=0;
                if(RUN_Status ==Sys_RUN)
                {

```

```

        BEEP();
        BEEP_Close=200;
    }

    Convert_Set=0;
    RUN_Status =Sys_STOP;
}

uint8_t Set_PWM;
if(Convert_Set>40)
    Set_PWM=Convert_Set;
else
    Set_PWM=Convert_Set;
//PWM_RPM_Convert();
__HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_1,Set_PWM);//pwm
0—71 speed 0-7000rpm
}
else
    __HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_1,0);//pwm 0—400
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */
    LCD_Display();
    Key_Handle();
}
/* USER CODE END 3 */
}

/**
 * @brief System Clock Configuration
 * @retval None
 */
void SystemClock_Config(void)
{
    RCC_OscInitTypeDef RCC_OscInitStruct = {0};
    RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

    /** Initializes the CPU, AHB and APB busses clocks
     */
    RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
    RCC_OscInitStruct.HSEState = RCC_HSE_ON;
    RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
    RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
    RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL4;
    RCC_OscInitStruct.PLL.PREDIV = RCC_PREDIV_DIV1;
    if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
    {
        Error_Handler();
    }
    /** Initializes the CPU, AHB and APB busses clocks
     */

```

```

RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                               |RCC_CLOCKTYPE_PCLK1;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;

if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
{
    Error_Handler();
}
}

/* USER CODE BEGIN 4 */
uint32_t next,Speed_Rel;
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    //测速
    if(GPIO_Pin ==FG_Pin)
    {
        Rpm_Cnt++;
        if(Rpm_Cnt>3)
        {
            uint32_t first = HAL_GetTick();
            if((first-next)>0)
                Speed_Rel=120000/(first-next);
            //Rpm=Speed_Rel;
            next=first;
            Rpm_Cnt=0;
        }
    }
}

uint32_t ms10;
uint32_t ms;
extern uint16_t Rpm,Time_SUM;
extern uint16_t save_time,save_rpm;
extern uint8_t Point_Flag;
extern uint8_t Sys_Mode;
extern uint16_t BEEP_Count,BEEP_Close;
uint16_t Half_Sec;
extern uint16_t full_rpm;
extern
    uint16_t full_Convert_Set;
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Instance == TIM1)
    {
        ms10++;
        if(Key1_Count)

```

```
Key1_Count--;

if(BEEP_Close)
    BEEP_Close--;

if(BEEP_Count)
    BEEP_Count--;
if(BEEP_Count==0)
    HAL_GPIO_WritePin(BEEP_GPIO_Port, BEEP_Pin, GPIO_PIN_RESET);
ms++;

if((ms%1000)==0)
{
    if( RUN_Status ==Sys_Down)
        if(Convert_Set>0)
        {
            Convert_Set--;
            if(Convert_Set==0)
            {
                RUN_Status =Sys_STOP;

                BEEP();
                Rpm=save_rpm;
                Time_SUM=save_time;
            }
        }
}

if(ms>2000)
{
    ms=0;

    if( RUN_Status==Sys_RUN)
    {
        if(full_Convert_Set>Convert_Set)
        {
            Convert_Set++;
        }
        else if(full_Convert_Set<Convert_Set)
        {
            Convert_Set--;
        }
    }
}
```

```

    }

    if(ms10>5000)
    {

        Half_Sec++;

        if(Half_Sec>1)
        {
            if(RUN_Status ==Sys_RUN)
            {

                if(Time_SUM>0)
                {
                    if(Set_Flag<2)
                    Time_SUM--;
                }
            }
            if(Time_SUM==0)
            {
                //NVIC_SystemReset();
                RUN_Status =Sys_Down;
                if(Convert_Set==0)
                {
                    BEEP();
                    Rpm=save_rpm;
                    Time_SUM=save_time;
                }
                //NVIC_SystemReset();

            }

            Half_Sec=0;
        }
        //if(Sys_Mode==Sys_RPM)
        //Point_Flag=~Point_Flag;

        //设置位置闪烁
        if(Set_Flag)
        {
            if(Set_Count)
            Set_Count--;
            else
            {
                Set_Flag1=0;
                Set_Flag2=0;
                Set_Flag=0;
            }
        }
    }

```



```
        if(Set_Flag==1)
        {
            if(Set_Flag1)
                Set_Flag1=0;
            else
                Set_Flag1=1;
            //Set_Flag1=~Set_Flag1;
        }
        else if(Set_Flag==2)
        {
            if(Set_Flag2)
                Set_Flag2=0;
            else
                Set_Flag2=1;
        }
        //Set_Flag2=~Set_Flag2;

        if(Key_Count)
            Key_Count--;
    }

    if(RUN_Status ==Sys_RUN)
    {
        if(rpm_flag)
            rpm_flag=0;
        else
            rpm_flag=1;
    }

    ms10=0;

}

//10ms//0.1ms
}
}

void PWM_RPM_Convert(void)
{
    switch(Rpm)
    {
        case 1000: Convert_Set=6;
        break;
    }
}
```

```

        case 1500: Convert_Set=7;
        break;
        case 2000: Convert_Set=9;
        break;
        case 2500: Convert_Set=12;
        break;
        case 3000: Convert_Set=17;
        break;
        case 3500: Convert_Set=21;
        break;
        case 4000: Convert_Set=27;
        break;
        case 4500: Convert_Set=33;
        break;
        case 5000: Convert_Set=40;
        break;
        case 5500: Convert_Set=48;
        break;
        case 6000: Convert_Set=54;
        break;
        case 6500: Convert_Set=60;
        break;
        case 7000: Convert_Set=70;
        break;
        default: Convert_Set=0;

    }

}

/* USER CODE END 4 */

/**
 * @brief This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return state */

    /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT
/**
 * @brief Reports the name of the source file and the source line number
 * where the assert_param error has occurred.
 * @param file: pointer to the source file name

```

```

    * @param   line: assert_param error line source number
    * @retval None
    */
void assert_failed(char *file, uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line number,
       tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
    /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/***** (C) COPYRIGHT STMicroelectronics *****/
#include "lcd.h"
#include "user.h"
void write_addr_dat_n(unsigned char _addr, unsigned char _dat, unsigned char n);

uint8_t LCD_ADD[]={0x5f,0x06,0x3d,0x2f,0x66,0x6b,0x7b,0x0e,0x7f,0x6f};
uint8_t Time_Status;
uint8_t Rpm_B,Rpm_S,Rpm_G,time_1,time_2;
uint16_t Rpm,Time_SUM,Dis;
uint16_t rel_rpm;
extern uint16_t Set_Flag,Key_Count,Key1_Count;
extern uint8_t Sys_Mode;
uint8_t Set_Flag1,Set_Flag2,point_add,rpm_flag;
extern uint8_t Point_Flag;
extern uint8_t KEY1_Pin_ON;
extern uint8_t Cover_Status;
float Rcf,R,S,rel_rcf;
extern uint8_t Convert_Set;
extern uint16_t RUN_Status;

void LCD_Display()
{
    //Rpm=234;
    //Time_SUM=3600;
    R=Rpm;
    Rcf=72*(R/1000)*(R/1000);
    if(Cover_Status==1)
    {
        if(Convert_Set>60)
            rel_rpm=Convert_Set*50+3500;
        else if(Convert_Set>54)
        {
            rel_rpm=Convert_Set*80+1700;
        }
        else if(Convert_Set>50)
            rel_rpm=Convert_Set*80+1680;
        else if(Convert_Set>40)
            rel_rpm=Convert_Set*64+2428;
    }
}

```

```

        else if(Convert_Set>33)
            rel_rpm=Convert_Set*77+1920;
    else if(Convert_Set>30)
        rel_rpm=Convert_Set*77+1960;
        else if(Convert_Set>21)
            rel_rpm=Convert_Set*78+1894;
        else if(Convert_Set>20)
            rel_rpm=Convert_Set*78+1862;
        else if(Convert_Set>15)
            rel_rpm=Convert_Set*49+2167;
        else if(Convert_Set>10)
            rel_rpm=Convert_Set*75+1450;
        else if(Convert_Set==9)
            rel_rpm=2000;
        else if(Convert_Set==8)
            rel_rpm=1800;
        else if(Convert_Set==7)
            rel_rpm=1500;
        else if(Convert_Set==6)
            rel_rpm=1000;
        else if(Convert_Set>0)
            rel_rpm=Convert_Set*140;

    S=rel_rpm;
    rel_rcf=72*(S/1000)*(S/1000);

}

if(Sys_Mode==Sys_RPM)
{
    if((RUN_Status==Sys_STOP)|| (Set_Flag))
    {
        Dis=Rpm;
    }
    else

        Dis=rel_rpm;

}
else
    Dis=Rcf;

if(Dis<10) {Rpm_B=0;Rpm_S=0;Rpm_G=0;}
else if (Dis<100) {Rpm_B=0;Rpm_S=0;Rpm_G=Dis/10;}
else if (Dis<1000) {Rpm_B=0;Rpm_S=Dis/100;Rpm_G=Dis/10%10;}
else if (Dis<10000) {Rpm_B=Dis/1000;Rpm_S=Dis/100%10;Rpm_G=Dis/10%10;}

```

```

//更新转速

if(Time_SUM<60) Time_Status =0;
else Time_Status =1;
if(Time_Status ==0) {time_1=Time_SUM/10;time_2=Time_SUM%10;if(Time_SUM<10)
time_1=0;}
else if(Time_Status ==1)
{time_1=Time_SUM/60/10;time_2=Time_SUM/60%10;if(Time_SUM<10) time_1=0;}
//更新时间

```

```

if(Set_Flag1)
{
    if((Key_Count==0)&&(Key1_Count==0))
    {
        if(Sys_Mode==Sys_RCF)

            write_addr_dat_n(0x00, 0x80, 1);
        else
            write_addr_dat_n(0x00, 0, 1);

        if(Cover_Status==0)
            write_addr_dat_n(0x02, 0x80, 1);
        else
            write_addr_dat_n(0x02, 0x00, 1);

        if(Cover_Status==1)
            write_addr_dat_n(0x04, 0x80, 1);
        else
            write_addr_dat_n(0x04, 0, 1);
    }
    else
    {
        if(Sys_Mode==Sys_RCF)
        {
            if(rpm_flag)
                write_addr_dat_n(0x00, LCD_ADD[Rpm_B]|0x80, 1);
            else
                write_addr_dat_n(0x00, LCD_ADD[Rpm_B]&0x7f, 1);
        }
        else
            write_addr_dat_n(0x00, LCD_ADD[Rpm_B], 1);

        if(Cover_Status==0)
            write_addr_dat_n(0x02, LCD_ADD[Rpm_S]|0x80, 1);
        else
            write_addr_dat_n(0x02, LCD_ADD[Rpm_S], 1);
    }
}

```

```

        if(Cover_Status==1)
            write_addr_dat_n(0x04, LCD_ADD[Rpm_G]|0x80, 1);
        else
            write_addr_dat_n(0x04, LCD_ADD[Rpm_G], 1);
    }
}
else
{
    if(Sys_Mode==Sys_RCF)
    {
        if(rpm_flag)
            write_addr_dat_n(0x00, LCD_ADD[Rpm_B]|0x80, 1);
        else
            write_addr_dat_n(0x00, LCD_ADD[Rpm_B]&0x7f, 1);
    }
    else
        write_addr_dat_n(0x00, LCD_ADD[Rpm_B], 1);

    if(Cover_Status==0)
        write_addr_dat_n(0x02, LCD_ADD[Rpm_S]|0x80, 1);
    else
        write_addr_dat_n(0x02, LCD_ADD[Rpm_S], 1);

    if(Cover_Status==1)
        write_addr_dat_n(0x04, LCD_ADD[Rpm_G]|0x80, 1);
    else
        write_addr_dat_n(0x04, LCD_ADD[Rpm_G], 1);
}

if(Set_Flag2)
{
    if((Key_Count==0)&&(Key1_Count==0))
    {
        if(Sys_Mode==Sys_RPM)
            write_addr_dat_n(0x06,0|0x80, 1);
        else
            write_addr_dat_n(0x06,0, 1);

        if(Time_Status ==0)
        {
            if(Point_Flag==1)
                write_addr_dat_n(0x08, 0x01&0x7f, 1);
            else
                write_addr_dat_n(0x08, 0x01|0x80, 1);
        }
        else
        {
            if(Point_Flag==1)
                write_addr_dat_n(0x08, 0x08&0x7f, 1);
            else

```

```

        write_addr_dat_n(0x08, 0x08|0x80, 1);
    }
    write_addr_dat_n(0x0a, (0&0x0f)<<4, 1);
}
else
{
    if(Sys_Mode==Sys_RPM)
        write_addr_dat_n(0x06, LCD_ADD[time_1]|0x80, 1);
    else
        write_addr_dat_n(0x06, LCD_ADD[time_1], 1);

    if(Time_Status ==0)
    {
        if(Point_Flag==1)
            write_addr_dat_n(0x08,
((LCD_ADD[time_2]&0xf0)|0x01)&0x7f, 1);
        else
            write_addr_dat_n(0x08,
((LCD_ADD[time_2]&0xf0)|0x01)|0x80, 1);
    }
    else
    {
        if(Point_Flag==1)
            write_addr_dat_n(0x08, ((LCD_ADD[time_2]&0xf0)|0x08)&0x7f,
1);
        else
            write_addr_dat_n(0x08, (LCD_ADD[time_2]&0xf0)|0x88, 1);
    }

    write_addr_dat_n(0x0a, (LCD_ADD[time_2]&0x0f)<<4, 1);
}

}
else
{
    if(Sys_Mode==Sys_RPM)
    {
        if(rpm_flag)
            write_addr_dat_n(0x06, LCD_ADD[time_1]|0x80, 1);
        else
            write_addr_dat_n(0x06, LCD_ADD[time_1]&0x7f, 1);
    }
    else
    {
        write_addr_dat_n(0x06, LCD_ADD[time_1], 1);
    }

    if(Time_Status ==0)

```

```
        {
            if(Point_Flag==1)
                write_addr_dat_n(0x08, ((LCD_ADD[time_2]&0xf0)|0x01)&0x7f, 1);
            else
                write_addr_dat_n(0x08, ((LCD_ADD[time_2]&0xf0)|0x01)|0x80, 1);

        }
    else
    {
        if(Point_Flag==1)
            write_addr_dat_n(0x08, ((LCD_ADD[time_2]&0xf0)|0x08)&0x7f, 1);
        else
            write_addr_dat_n(0x08, ((LCD_ADD[time_2]&0xf0)|0x08)|0x80, 1);

    }

    write_addr_dat_n(0x0a, (LCD_ADD[time_2]&0x0f)<<4, 1);
}

if(Set_Flag2)
{
    if((Key_Count==0)&&(Key1_Count==0))
        point_add=0;
    else
        point_add=LCD_ADD[time_2];
}
else
    point_add=LCD_ADD[time_2];
}
```