HAL库编程基础

由于HA库的代码较为冗长，对于本资料所提供的历年真题代码，其编程格式如下：（main.c）

> 头文件放置位置

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "adc.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "i2c_hal.h"
#include "lcd.h"
#include "mint.h"
#include "stdio.h"
#include "string.h"
```

系统自动生成

用户自己添加

> 变量定义位置

```
/* USER CODE BEGIN PV */
extern keys key[4];

bool page=0,flag1=0,flag2=0,flag3=0;

char text[30],rxdat,text1[20];
extern int dtime,dtime1,dtime2,dtime3;

uint8_t Height=0,Level=4,Level_1=4,TH[3] = {10,20,30},Line=0;
double adc_val=0;

/* USER CODE END PV */
```

> 函数声明位置

```
/* USER CODE BEGIN PFP */
void LCD_Proc(void);
void KEY_Proc(void);
/* USER CODE END PFP */
```

> 函数定义位置

```
/* USER CODE BEGIN 4 */
/**************************************显示模块************************************/
void LCD_Proc(void)
{
  if(page == 0)
  {
    flag1 = 1;
    LCD_DisplayStringLine(Line1,(u8 *)"    Liquid Level    ");
    sprintf(text,"  Threshold1:%dcm   ",Height);
    LCD_DisplayStringLine(Line4,(u8 *)text);
    sprintf(text,"     ADC:%.2fV      ",adc_val);
    LCD_DisplayStringLine(Line6,(u8 *)text);
    sprintf(text,"       Level:%d      ",Level);
    LCD_DisplayStringLine(Line8,(u8 *)text);
  }
  else if(page == 1)
  {
    //此页面为设置页面，认为不工作，进行相应操作
    flag1 = 0;
    dtime1 = 1;
    LED_Proc(1,0);
    //page1显示部分
    LCD_DisplayStringLine(Line1,(u8 *)"   Paramter Setup   ");
    sprintf(text,"  Threshold1:%dcm   ",TH[0]);
    if(Line == 0) LCD_SetTextColor(Green);
    LCD_DisplayStringLine(Line4,(u8 *)text);
    LCD_SetTextColor(Black);

    sprintf(text,"  Threshold1:%dcm   ",TH[1]);
    if(Line == 1) LCD_SetTextColor(Green);
    LCD_DisplayStringLine(Line6,(u8 *)text);
    LCD_SetTextColor(Black);

    sprintf(text,"  Threshold1:%dcm   ",TH[2]);
```

省赛基础部分

省赛主要考察开发板上的各种外设：;LED、ADC、定时器中断、输入捕获、PWM输出、USART、EEPROM、LCD、按键、(DCA、可编程电阻从未考过)，要求我们在熟练掌握各种外设的基础上进行项目开发。嵌入式不重算法但是重逻辑。外设与逻辑是每年的考点，因此在省赛部分我们应当

1.重点熟悉各种外设的基础配置，牢记代码

2.赛前两到三周，进行真题训练，快速掌握一些外设应用的技巧以及比赛中常用的逻辑技巧。

3.每次刷完题目一定！一定！一定！要好好检查是否正真的完成了赛题要求，很多人觉得自己明明都做出来了，但是却拿了个省二、省三的原因就在这里。（没有标准答案，因此可以找别人做的进行对照）

嵌入式的省赛并不难，做好以上3点，在比赛时程序设计题满足所有要求，省一就有了，如果想要名次靠前，那么就需要补一补客观题，这个是一大难点。

一、LED部分

> 简单粗暴，但是不能做到独立控制

```
void LED_Proc(unsigned char pin)
{
    HAL_GPIO_Write(GPIOC,GPIO_PIN_All,GPIO_PIN_SET);
    HAL_GPIO_Write(GPIOC,PIN << 2,GPIO_PIN_RESET);
    HAL_GPIO_Write(GPIOC,GPIO_PIN_2,GPIO_PIN_SET);
    HAL_GPIO_Write(GPIOD,GPIO_PIN_2,GPIO_PIN_RESET);
}
```

> 实现每个LED的单独控制（强烈建议）

```
void LED_Proc(int LED_Num,int Status)
{
    static unsigned char LED_status = 0xFF;//静态局部变量，只会初始化一次。
    if(LED_Num > 8 || LED_Num < 1) return;
    if(Status == 0) LED_status |= (0x01 << (LED_Num-1));//灭灯
    else LED_status &= ~(0x01 << (LED_Num-1));  //亮灯

    GPIOC -> ODR = (LED_status << 8);
    HAL_GPIO_Write(GPIOC,GPIO_PIN_2,GPIO_PIN_SET);
    HAL_GPIO_Write(GPIOD,GPIO_PIN_2,GPIO_PIN_RESET);
}
```

二、ADC部分

> 选择引脚，配置为ADCx模式

```
HAL_ADCEx_Calibration_Start(&hadc2,ADC_SINGLE_ENDED);
double getADC(ADC_HandleTypeDef *pin)
{
    double adc;
    HAL_ADC_Start(pin);
    adc = HAL_ADC_GetValue(pin);
    return adc*3.3/4096;
}
```

三、DAC部分

> 选择引脚PA4、PA5，配置为DAC_OUT
>
> 在DAC选项中配置DAC_OUT1、DAC_OUT2模式为Connected to external pin only
>
> 在程序中利用函数进行设置输出电压

```
void DAC1_OUT1_Set_Vol(float vol) //设置PA4的输出电压
{
    uint16_t temp;
    temp = (4096*vol/3.3f);
    HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_1,DAC_ALIGN_12B_R,temp);
    HAL_DAC_Start(&hdac1, DAC_CHANNEL_1);//启动DAC1 通道1输出
}

void DAC1_OUT2_Set_Vol(float vol) //设置PA5的输出电压
{
    uint16_t temp;
    temp = (4096*vol/3.3f);
    HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_2,DAC_ALIGN_12B_R,temp);
    HAL_DAC_Start(&hdac1, DAC_CHANNEL_2);//启动DAC1 通道2输出
}
```

四、定时器部分

1、定时器中断

> 选择一个可用的定时器TIMX
>
> 使能TIMX，设置分频系数(Prescaler)、重装载值(Counter Period)
>
> 开启NVIC中断
>
> 进入程序，开启定时器中断：HAL_TIM_Base_Start_IT(&htimx);
>
> 编写中断回调函数。

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Instance == TIMx)
    {
        //需要处理的操作
    }
}
```

1.1TIM短按键实现

> 开启中断后，定义结构体：

```
#include "stdbool.h"
typedef struct
{
    bool key_sta;
    bool single_sta;
    uint8_t judge_sta;
}keys;
```

> 具体思路为：
>
> 第一次进入中断，若被按下，标志一次
>
> 第二次进入如果仍然被按下则认为真的被按下了，再标志一次，并将按下标志置一
>
> 第三次及以上进入，需要按键松开才使标志变为最初状态

```
struct keys key[4] = {0,0,0};
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Instance == TIM3)
    {
        key[0].key_sta = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_0);
        key[1].key_sta = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_1);
        key[2].key_sta = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_2);
        key[3].key_sta = HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0);

        for(int i=0;i<4;i++)
        {
            switch(key[i].judge_sta)
            {
                case 0:
                {
                    if(key[i].key_sta == 0) key[i].judge_sta = 1;
                }
                break;
                case 1:
                {
                    if(key[i].key_sta == 0)
                    {
                        key[i].judge_sta = 2;
                        key[i].single_sta = 1;
                    }
                    else key[i].judge_sta = 0;
                }
                break;
                case 2:
                {
                    if(key[i].key_sta == 1) key[i].judge_sta = 0;
                }
                break;
            }
        }
    }
}
```

1.2TIM长按键实现

> 长按键在短按键的基础上加入时长的判断，首先定义结构体如下：

```
#include "stdbool.h"
typedef struct
{
    bool key_sta;
    bool short_sta;
    bool long_sta;
    int time_flag;
    uint8_t judge_sta;
}keys;
```

> 长按键的具体思路为：
>
> judge==0：判断按键是否被按下，是：judge==1，否：judge==0
>
> judge==1：判断按键是否真的被按下，是：judge==2，否：judge==0
>
> judge==2：开始计时，如果没有超过一定时间按键被松开，则判定为短按键judge==0，short==1，time ==0，否则judge==3
>
> judge==3，如果需要按下后长按键一直被触发则反复计时，否则只执行一次。如果计时值达到且按键没有被松开，则置long为1，time为0，其余不变，若被松开则置judge为0。。。。。

```c
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Instance == TIMx)
    {
        key[0].key_sta = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_0);
        key[1].key_sta = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_1);
        key[2].key_sta = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_2);
        key[3].key_sta = HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0);

        for(int i=0;i<4;i++)
        {
            switch(key[i].judge_sta)
            {
                case 0:
                {
                    if(key[i].key_sta == 0) key[i].judge_sta = 1;
                }
                break;
                case 1:
                {
                    if(key[i].key_sta == 0) key[i].judge_sta = 2;
                    else key[i].judge_sta = 0;
                }
                break;
                case 2:
                {
                    key[i].time_sta ++;
                    if(key[i].time_sta < 78 && key[i].key_sta == 1)
                    {
                        key[i].short_sta = 1;
                        key[i].judge_sta = 0;
                        key[i].time_sta = 0;
                    }
                    else if(key[i].time_sta >= 78) key[i].judge_sta = 3;
                }
                break;
                case 3:
                {
                    if(key[i].key_sta == 0)
                    {
                        key[i].time_sta++;
                        if(key[i].time_sta >= 15)
                        {
                            key[i].long_sta = 1;
                            key[i].time_sta = 0;
                        }
                    }
                    else
                    {
                        key[i].judge_sta = 0;
```

```
                        key[i].time_sta = 0;
                    }
                }
                break;
            }
        }
    }
}
```

2、定时器PWM输出

> 点击引脚，选择TIMx通道
>
> 使能TIMX，配置TIMX模式为：PWM Generation CHx
>
> 设置分频系数(Prescaler)、重装载值(Counter Period)、占空比(Pulse)
>
> 进入程序，初始化定时器，设置初始占空比。

```
HAL_TIM_PWM_Start(&htim16,TIM_CHANNEL_1);         //初始化TIM16PWM输出
HAL_TIM_PWM_Start(&htim17,TIM_CHANNEL_1);         //初始化TIM17PWM输出
__HAL_TIM_SetCompare(&htim16,TIM_CHANNEL_1,10);   //初始化TIM16初始占空比
__HAL_TIM_SetCompare(&htim17,TIM_CHANNEL_1,10);   //初始化TIM17初始占空比
```

3、定时器输入捕获

> 点击引脚，选择TIMx通道（选择CH1的）
>
> 使能TIMX，配置TIMX模式为：Input Capture direct mode
>
> 设置分频系数为（80-1），重装载值默认(不能太小，参数太差会导致精度下降)，使能NVIC中断。
>
> 初始化定时器

```
HAL_TIM_IC_Start_IT(&htim2,TIM_CHANNEL_1);
HAL_TIM_IC_Start_IT(&htim2,TIM_CHANNEL_2);//测占空比开启，设为间接模式
```

> 进入程序,设置中断回调函数：

```
unsigned int ccrl_val=0;
unsigned int frq=0;
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Instance == TIM2)//测量频率
    {
        ccrl_val = HAL_TIM_ReadCapturedValue(htim,TIM_CHANNEL_1)
        __HAL_TIM_SetCounter(htim,0);
        frq = (80000000 / 80) / ccrl_val;
        HAL_TIM_IC_Start(htim,TIM_CHANNEL_1);
    }
}
```

> 若要测量占空比，则需要打开通道2的间接捕获：Input Capture indirect mode

```
double ccrl_vala=0,ccrl_valb=0,duty2=0;
unsigned int frq=0;
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Instance==TIM2) //测量占空比以及频率
    {
        if(htim->Channel==HAL_TIM_ACTIVE_CHANNEL_1)//中断消息来源 选择直接输入的通道
        {
            ccr1_val2a=HAL_TIM_ReadCapturedValue(htim,TIM_CHANNEL_1);
            ccr1_val2b=HAL_TIM_ReadCapturedValue(htim,TIM_CHANNEL_2);
            __HAL_TIM_SetCounter(htim,0);
            frq2=(80000000/80)/ccr1_val2a;
            duty2=(ccr1_val2b/ccr1_val2a)*100;
            HAL_TIM_IC_Start(htim,TIM_CHANNEL_1);
            HAL_TIM_IC_Start(htim,TIM_CHANNEL_2);
        }
    }
}
```

五、eeprom部分（I2C）

基础函数编写

> 找到官方提供的i2c底层驱动
>
> CUBMX中初始化PB6、PB7为GPIO_OUTPUT
>
> 利用提供的函数封装eeprom的读写程序
>
> 写操作联系芯片，

```
unsigned char eeprom_read(unsigned char addr)
{
    unsigned char dat;
    I2CStart();
    I2CSendByte(0xa0);
    I2CWaitAck();
    I2CSendByte(addr);
    I2CWaitAck();
    I2CStop();

    I2CStart();
    I2CSendByte(0xa1);
    I2CWaitAck();
    dat = I2CReceiveByte();
    I2CWaitAck();//I2CSendNotAck
    I2CStop();
    return dat;
}
```

> 再编写eeprom_write()

```
void eeprom_write(unsigned char addr,unsigned char dat)
{
    I2CStart();
    I2CSendByte(0xa0);
    I2CWaitAck();
    I2CSendByte(addr);
    I2CWaitAck();
    I2CSendByte(dat);
    I2CWaitAck();
    I2CStop();
}
```

> 需要注意的是eeprom每个地址只能存8为，而一个unsigned int类型的数据为16位，所以需要分开存储，读取的时候同理。

```
unsigned int dat;
unsigned char dath,datl;
dath = dat >> 8;
datl = dat & 0xff;
errprom_write(1,dath);
HAL_Delay(10);
eeprom_write(2,datl);
```

六、USART部分

1.普通收发

基础配置

> CUBMX中使能PA9、PA10为USART_TX、USART_RX模式
>
> NVIC中开启中断，更新程序。

发送部分

```
char temp[20];
sprintf(temp,"happy\r\n");
HAL_UART_Transmit(&huart1,(uint8_t *)temp,strlen(temp),50);
```

接收部分

> 在main.c中开启串口接收中断：HAL_UART_Receive_IT(&huart1,&rxdat,1);
>
> 编写接收中断回调函数。

```
char rxdata[30];
uint8_t rxdat;
uint8_t rx_p;
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
{
    rxdata[rx_p++] = rxdat;
    HAL_UART_Receive_IT(&huart1,&rxdat,1);
}
```

2.DMA+串口数据收发

> 发送数据

```
HAL_UART_Transmit_DMA(&huart1,(uint8_t *)"Hello!",sizeof("Hello!"));
```

> 接收数据(不定长)
>
> 首先在main.c中进行初始化

```
//main中初始化
HAL_UART_Receive_DMA(&huart1,rxdata,100);//开启DMA中断接收
__HAL_UART_ENABLE_IT(&huart1,UART_IT_IDLE);//开启串口闲时中断
```

> 随后找到stm32g4xx_it.c，在中断服务函数USART1_IRQHandler中进行编写

```
void USART1_IRQHandler(void)
{
  /* USER CODE BEGIN USART1_IRQn 0 */

  /* USER CODE END USART1_IRQn 0 */
  HAL_UART_IRQHandler(&huart1);
  /* USER CODE BEGIN USART1_IRQn 1 */
  if((__HAL_UART_GET_FLAG(&huart1,UART_FLAG_IDLE) != RESET))//判断接收是否结束
    {
       __HAL_UART_CLEAR_IDLEFLAG(&huart1);//清楚标志位
       HAL_UART_DMAStop(&huart1);
       //__HAL_DMA_GET_COUNTER(&hdma_usart1_rx);//获取DMA接收计数值
       //100-temp; //计算数据长度,100为DMA接收长度
       //在这里写你想要做的事情

       //最后开启接收中断
       HAL_UART_Receive_DMA(&huart1,rx_buffer,100);//开启DMA
    }
  /* USER CODE END USART1_IRQn 1 */
}
```

七、可编程电阻

> 读操作，联系芯片，读为1。发送地址后，读数据。

```
uint8_t MCP4017_Read(void)
{
    uint8_t dat;
    I2CStart();
    I2CSendByte(0x5f);
    I2CWaitAck();

    dat = I2CReceiveByte();
    I2CSendNoAck();
    I2CStop();
    return dat;
}
```

> 写操作，联系芯片，写为0。发送地址后，写数据

```
void MCP4017_Write(uint8_t dat)
{
    I2CStart();
    I2CSendByte(0x5e);
    I2CWaitAck();

    I2CSendByte(dat);
    I2CWaitAck();
    I2CStop();
}
```

附1：Unix时间

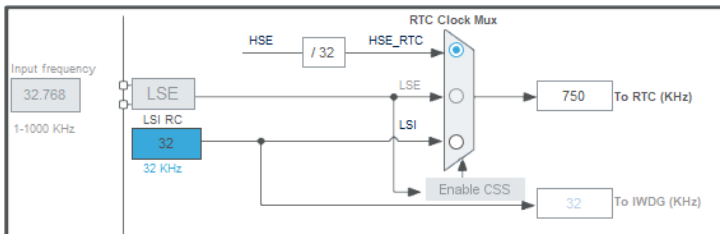
```
#include "time.h"

struct tm tm_info;
long time;

tm_info.tm_year = year - 1900;
tm_info.tm_mon = month - 1;
tm_info.tm_mday = day;
tm_info.tm_hour = hours;
tm_info.tm_min = minter;
tm_info.tm_sec = second;
tm_info.tm_isdst = -1; //夏令时

time = mktime(&tm_info); //得到以秒计的Unix时间。
```

附2：STM32片内RTC的使用

1.CUBMX初始化

> 点开RTC前三选项如下：

> 时钟页面配置如下：



回到RTC，依次配置如下：

▾ General	
Hour Format	Hourformat 24
Asynchronous Predivider value	125-1
Synchronous Predivider value	6000-1
▾ Calendar Time	
Data Format	BCD data format
Hours	0
Minutes	0
Seconds	0
Day Light Saving: value of hour adjustment	Daylightsaving None
Store Operation	Storeoperation Reset
Alarm Mask Date Week day	Enable
Alarm Mask Hours	Enable
Alarm Mask Minutes	Enable
Alarm Mask Seconds	Disable
Alarm Sub Second Mask	All Alarm SS fields are masked.
Alarm Date Week Day Sel	Date
Alarm Date	1

> 程序内首先初始化：

```
extern RTC_AlarmTypeDef sAlarm;
RTC_TimeTypeDef GetTime;
RTC_DateTypeDef GetDate;
//读取数据，顺序不能变化否则无法读取。
HAL_RTC_GetTime(&hrtc,&GetTime,RTC_FORMAT_BIN);
HAL_RTC_GetDate(&hrtc,&GetDate,RTC_FORMAT_BIN);
//如需要开启秒中断，则加入下面语句，否则不需要。
sAlarm.AlarmTime.Seconds = GetTime.Seconds + 1;
HAL_RTC_SetAlarm_IT(&hrtc,&sAlarm,RTC_FORMAT_BIN);
```

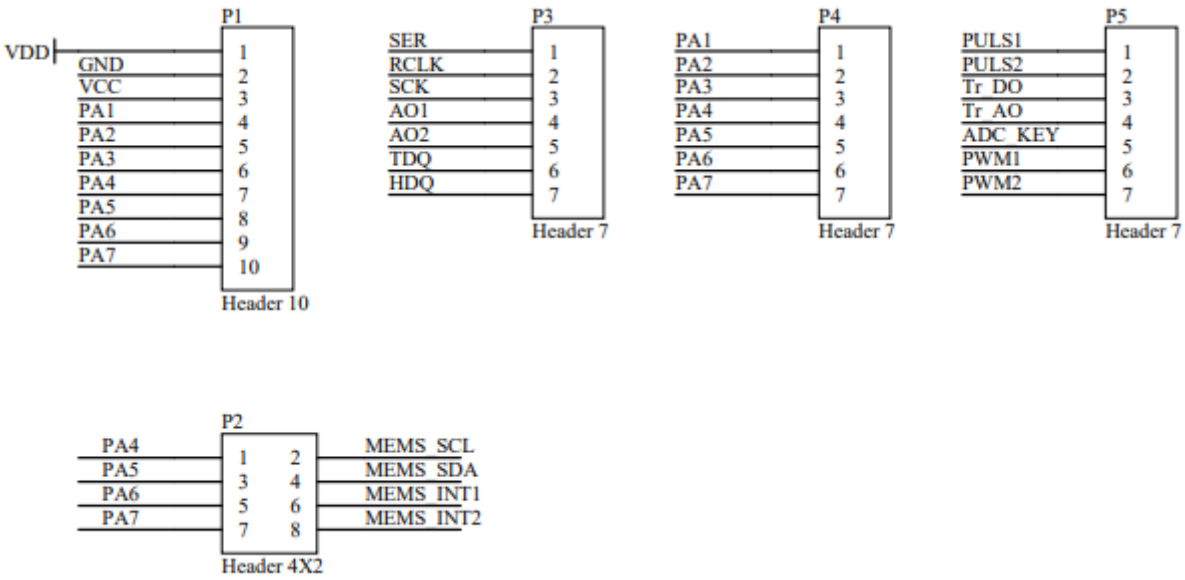
> 若开启中断，中断配置如下：(函数可在头文件：stm32g4xx_hal_rtc.h中找到)
>
> 每秒进入一次；若需改变则改变配置。见配置图2。

```c
void HAL_RTC_AlarmAEventCallback(RTC_HandleTypeDef *hrtc)
{
    HAL_RTC_GetTime(hrtc,&GetTime,RTC_FORMAT_BIN);
    HAL_RTC_GetDate(hrtc,&GetDate,RTC_FORMAT_BIN);
    //Show Time

    sAlarm.AlarmTime.Seconds = GetTime.Seconds + 1;
    HAL_RTC_SetAlarm_IT(hrtc,&sAlarm,RTC_FORMAT_BIN);
}
```

国赛部分（拓展板部分）

国赛在开发板的基础上加入了拓展版，拓展版包含较多的资源，因此国赛的题量一般较大，如果在省赛时你不能在3小时内完赛那么你就需要在这段时间里加快速度了，蓝桥杯每年的难度是在不断上升，相信24年也不例外。（23年开始国赛也在分赛区比赛，部分赛区由于第一次举办未提供必要的设备：示波器、万用表，大家比赛时一定要向赛点确认是否会提供，否则问题很大）

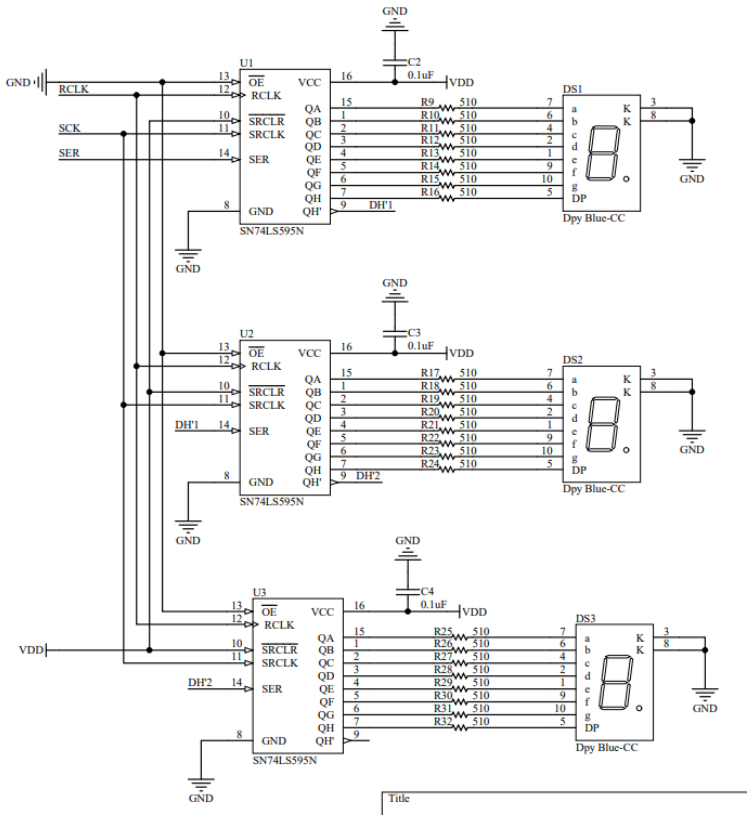
> 拓展版资源包括四路PWM输出、两路方波输出以及下面将要讲解的部分，其接口如下：



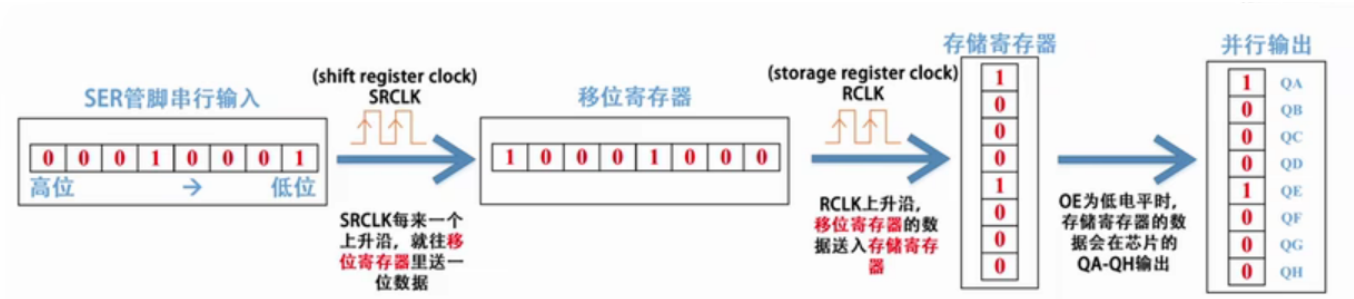
一、数码管控制

1.原理介绍

> 原理图如下：



> SN74LS595N为移位寄存器，SRCLK每次由低电平变为高电平时QA-QH全体向右移一位，QH的值移至QH'，而SER的值移动到QA。RCLK每次由低变高，移位寄存器的值就转移到存储寄存器，当OE为低时输出。



2.整数显示

> 据此可以得到，我们要在数码管山显示数字则只需要将需要输出的数字转化为段码，再利用以上特性进行传输即可达到显示效果。

```
/*
**PA1:SER：数据输入端
**PA2:RCLK：上升沿移位寄存器的值转移到存储寄存器
**PA3:SRCLK：上升沿移位寄存器向右移位
*/
unsigned char num[11] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07,
                         0x7f, 0x6f, 0x00};
void LED_Disp(int data)
{
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_2,GPIO_PIN_RESET);
    int tem = 0;
    for(int i=0;i<3;i++)
    {
        tem = num[data%10];
        if(data == 0 && i>0) tem = num[10];
        data/=10;
        for(int j=0;j<8;j++)
        {
            HAL_GPIO_WritePin(GPIOA,GPIO_PIN_3,GPIO_PIN_RESET);
            if(tem & 0x80) HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1,GPIO_PIN_SET);
            else HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1,GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOA,GPIO_PIN_3,GPIO_PIN_SET);
            tem = tem << 1;
        }
    }
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_2,GPIO_PIN_SET);
}
```

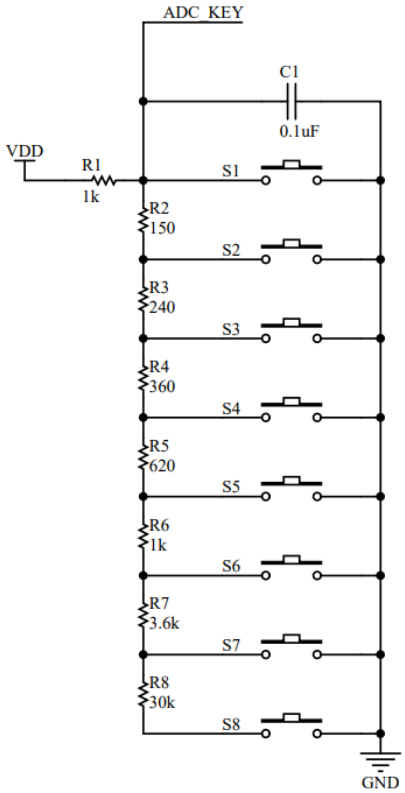
3.小数显示

> 小数与整数不同之处仅为小数点的亮与灭。当确定了小数的位数时，可以借助点亮了小数点的段码进行替换即可达到显示小数的效果。

```
/*
**PA1:SER：数据输入端
**PA2:RCLK：上升沿一位寄存器的值转移到存储寄存器
**PA3:SRCLK：上升沿移位寄存器向右移位
*/
unsigned char num[11] = {0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07,
                         0x7f, 0x6f, 0x00};//纯段码
unsigned char d_num[11] = {0xbf, 0x86, 0xdb, 0xcf, 0xe6, 0xed, 0xfd, 0x87,
                         0xff, 0xef, 0x80};//带小数点段码
/*
*d为小数的位数
*data为小数乘10^d后得到的整数。最多显示3位
*如：要显示3.30，则data = 330，d = 2。
*如：要显示33.0，则data = 330，d = 1。
*/
void LED_Disp_d(int data,int d)
{
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_2,GPIO_PIN_RESET);
    int tem = 0;
    for(int i=0;i<3;i++)
    {
        if(i == d) tem = d_num[data%10];
        else tem = num[data%10];
//        if(data == 0 && i>0) tem = num[10];
        data/=10;
        for(int j=0;j<8;j++)
        {
            HAL_GPIO_WritePin(GPIOA,GPIO_PIN_3,GPIO_PIN_RESET);
            if(tem & 0x80) HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1,GPIO_PIN_SET);
            else HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1,GPIO_PIN_RESET);
            HAL_GPIO_WritePin(GPIOA,GPIO_PIN_3,GPIO_PIN_SET);
            tem = tem << 1;
        }
    }
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_2,GPIO_PIN_SET);
}
```

二、ADC按键

> 原理图如下：



可见，ADCKEY实质上就是利用不同按钮按下时的电压值的大小来判断，按下的按键。

因而仅需要判断电压所在区间即可知道按下的为哪一个按键。

S1按下：V = 4095*0/1000=0

S2按下：V = 4095*[1-1000/(150+1000)]=534

S3按下：V = 4095*[1-1000/(150+1000+240)]=1148

S4按下：V = 4095*[1-1000/(150+1000+240+360)]=1755

S5按下：V = 4095*[1-1000/(150+1000+240+360+620)]=2367

S6按下：V = 4095*[1-1000/(150+1000+240+360+620+1000)]=2879

S7按下：V = 4095*[1-1000/(150+1000+240+360+620+1000+3600)]=3507

S8按下：V = 4095*[1-1000/(150+1000+240+360+620+1000+3600+30000)]=3984

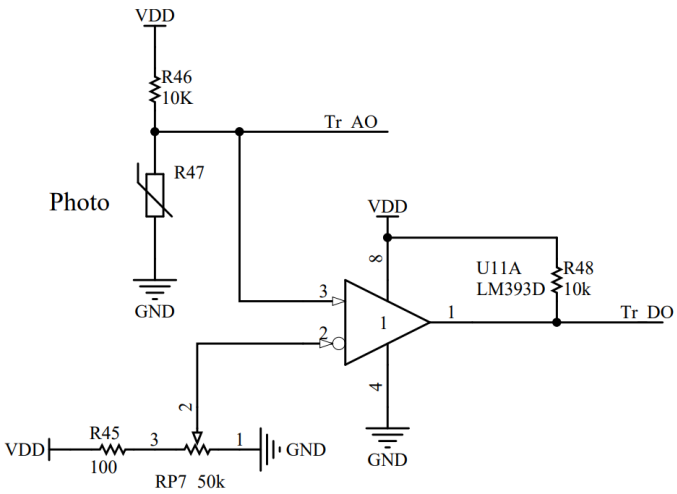
无按钮按下：V=4095;

有程序如下：

```c
/*
*读取ADC的函数不再重复，此处仅给出判断语句。
*/
uint16_t ADC_Key_Scan(void)
{
    if((int)val[0] < 10) return 1;
    if((int)val[0] < 800) return 2;
    if((int)val[0] < 1600) return 3;
    if((int)val[0] < 2000) return 4;
    if((int)val[0] < 2700) return 5;
    if((int)val[0] < 3300) return 6;
    if((int)val[0] < 3700) return 7;
    if((int)val[0] < 4000) return 8;
    return 0;
}
```

三、光敏电阻

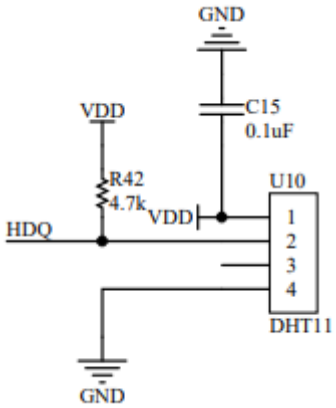
> 原理图如下：



> 光敏电阻相当于一个“电位器”，光照强度就是旋钮。
>
> 其后接入一个比较器，当V光>VRP7时输出高电平，反之输出低电平。
>
> 本质上还是考察ADC的运用，下面介绍多通道采集配置步骤：
>
> 首先初始化多个引脚，然后点击ADC，将Number of Conversion的值改为你所开启的通道值
>
> 修改rank，即扫描通道的排序，同时设置扫描速度，需要设置的慢一点，否则不能正常运行。
>
> 最后编写程序如下：（这里开启了两个通道）

```
double val[2] = {0};
void GetADC_Val(ADC_HandleTypeDef* adc)
{
    HAL_ADC_Start(adc);
    val[0] = HAL_ADC_GetValue(adc)*3.3/4095;
    HAL_ADC_Start(adc);
    val[1] = HAL_ADC_GetValue(adc)*3.3/4095;
}
```

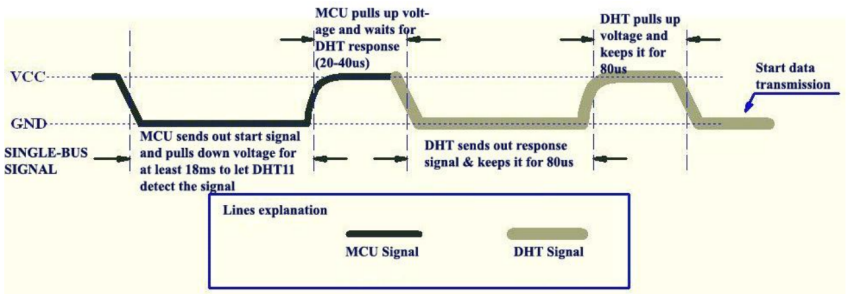
四、DHT11温湿度传感器



时序图如下：

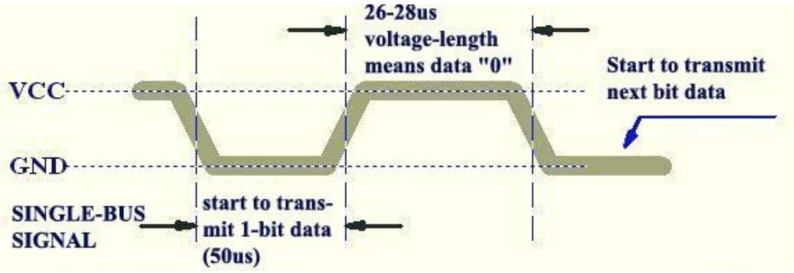
> 首先主机将其拉低18ms，然后释放20-40us，就会进入数据接收模式：

Data Single-bus free status is at high voltage level. When the communication between MCU and DHT11 begins, the programme of MCU will set Data Single-bus voltage level from high to low and this process must take at least 18ms to ensure DHT's detection of MCU's signal, then MCU will pull up voltage and wait 20-40us for DHT's response.



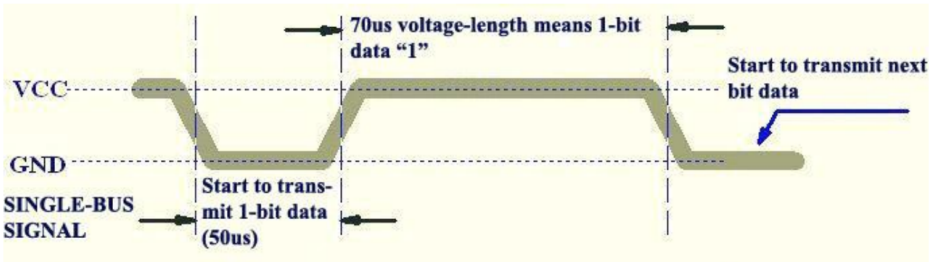
读0

> 在接收到主机的请求后，DHT开始发送数据，若为0。首先将电平拉低50us，而后释放拉高26-28us，再拉低，此时的电平高低即为数据的1、0。接着开始下一个数据的传输



读1

> DHT首先拉低电平，而后持续一个较长的高电平：70us。则为发送数据1。



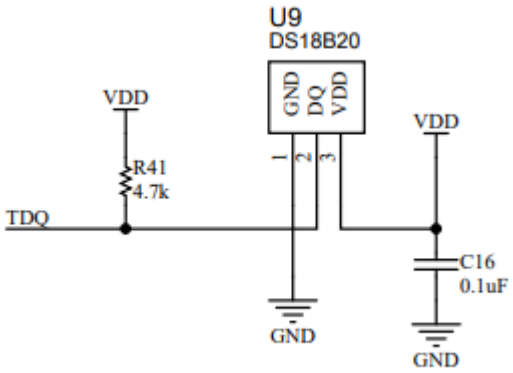
> 因此数据的读取可以在拉高后的28-70us内读，为低则是0，否则是1。
>
> 程序就像这样，可以按图加以理解：
>
> （比赛时会直接给出读取的函数，不用自己写，但是还是理解一下的好！）

```
unsigned int dht11_read(void)
{
    int i;
    long long val;
    int timeout;

    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7,GPIO_PIN_RESET);
    delay_us(18000);  //pulldown  for 18ms
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7,GPIO_PIN_SET);
    delay_us( 20 );    //pullup for 30us

    mode_input();

        //等待DHT11拉高，80us
    timeout = 5000;
    while( (! HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_7)) && (timeout > 0) ) timeout--;
//wait HIGH

    //等待DHT11拉低，80us
    timeout = 5000;
    while( HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_7) && (timeout > 0) ) timeout-- ;
//wait LOW

    #define CHECK_TIME 28

    for(i=0;i<40;i++)
    {
        timeout = 5000;
        while( (! HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_7)) && (timeout > 0) ) timeout--;
//wait HIGH

        delay_us(CHECK_TIME);
        if ( HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_7) )
        {
            val=(val<<1)+1;
        }
        else
        {
            val<<=1;
        }

        timeout = 5000;
        while( HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_7) && (timeout > 0) ) timeout-- ;
//wait LOW
    }

    mode_output();
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7,GPIO_PIN_SET);

    if (((val>>32)+(val>>24)+(val>>16)+(val>>8) -val ) & 0xff  ) return 0;
```

```
    else return val>>8;

}
```

五、DS18B20温度传感器

> 原理图如下：

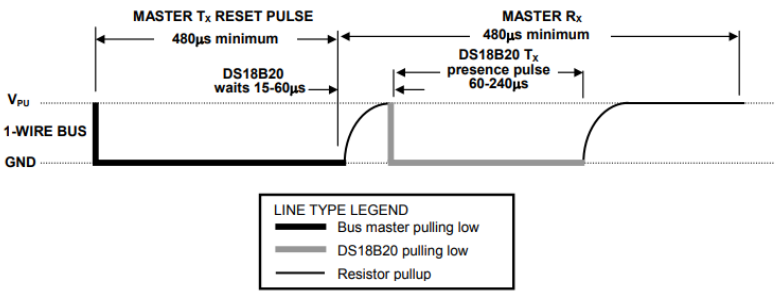


> 下面简要介绍DS18B20的使用方法：

1.复位操作

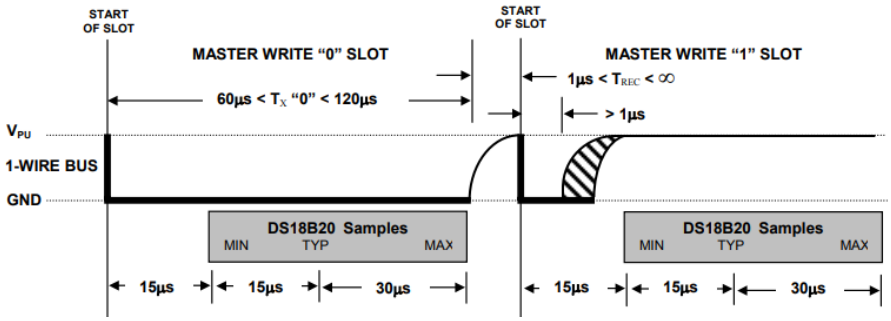
> 与DS18B20的所有通信都从一个初始化序列开始，该序列由来自master的复位脉冲和来自DS18B20的存在脉冲组成。当DS18B20发送在场脉冲响应复位时，它是在向主控表明它在总线上，准备操作。

INITIALIZATION TIMING Figure 13



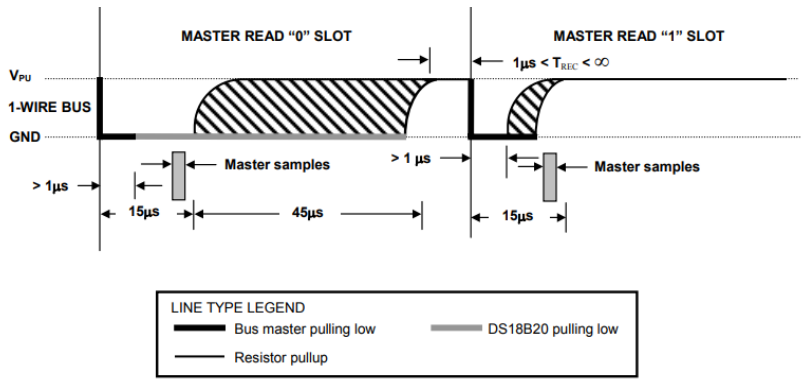
2.写操作

> 有两种写时间槽:“写1时间槽”和“写0时间槽”。总线主用一个写1时隙给DS18B20写逻辑1，用一个写0时隙给DS18B20写逻辑0。所有写时隙的持续时间必须最少为60us，每个写时隙之间的恢复时间最少为1us。两种类型的写时隙都是由master将单总线拉低发起的
>
> 要生成写1时隙，在拉低单线总线后，总线主必须在15ms内释放单线总线。当总线释放时，5k的上拉电阻会将总线拉高。要生成写0时隙，在拉低1线总线后，总线主必须在时隙的持续时间内(至少60us)继续保持总线低电平。
>
> DS18B20在master发起写时隙后持续15us到60us的窗口期间对单线总线进行采样。如果总线在采样窗口期间是高电平，一个1被写入DS18B20。如果线路低，则向DS18B20写入一个0。



3.读操作

> DS18B20只能在master发出读取时隙时向master发送数据。因此，master发出读片[BEh]或读电源[B4h]命令后，必须立即生成读时隙，这样DS18B20才能提供所请求的数据。此外，master在发出Convert T [44h]或Recall E2[B8h]命令后，可以生成读时隙。
>
> 读操作的时序图如下：

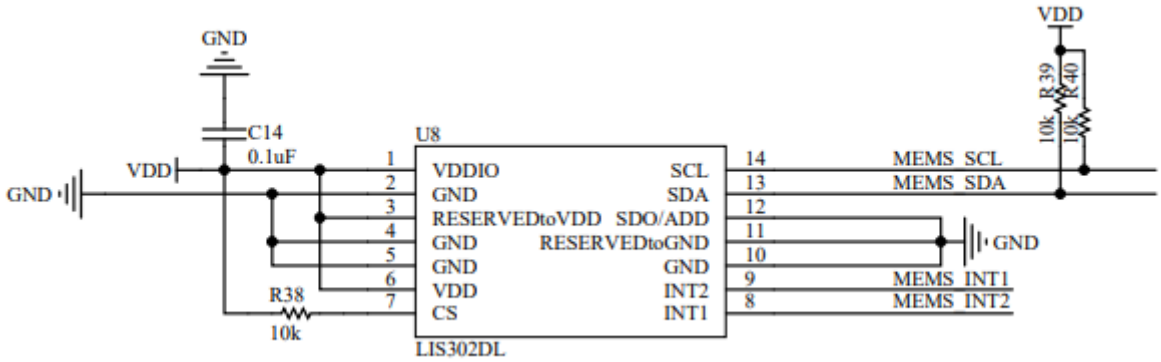


4.读温度

> 如果看到了这里，那么前面的不懂也行，就按照下面的步骤使用提供的函数进行读取即可。就像学习eeprom一样。
>
> DS18B20的典型温度读取过程, DS18B20的典型温度读取过程为:
>
> 复位→发SKIP ROM命令（OXCC)→发开始转换命令(0X44)→延时→复位→发送SKIPROM命令(OXCC)→发读存储器命令（OXBE)→连续读出两个字节数据(即温度)→结束。
>
> 最后，如果你有一个底层的驱动（23年比赛时给的代码和这是一样的，需要自己看看给的函数叫啥。），如果一样那么你就可以像下面这样写一个读取温度的函数：

```
double Read_temperature(void)
{
    uint8_t h,l;
    uint16_t tem;
    double Tem;

    ow_reset();
    ow_byte_wr(OW_SKIP_ROM);
    ow_byte_wr(DS18B20_CONVERT);

    ow_reset();
    ow_byte_wr(OW_SKIP_ROM);
    ow_byte_wr(DS18B20_READ);

    l = ow_byte_rd();
    h = ow_byte_rd();
    tem = (h<<8) + l;
    if((tem & 0xf800) == 0xf800)
    {
        tem = (~tem) + 1;
        Tem = tem * -0.0625;
    }
    else Tem = tem * 0.0625;
    return Tem;
}
```

六、LIS302DL三轴加速度传感器



> 与eeprom类似，利用i2c通讯
>
> 使用前修改提供的i2c底层驱动

1.读写程序

```
unsigned char LIS_read(unsigned char add)
{
    unsigned char data;
    I2CStart();
    I2CSendByte(0x38);
    I2CWaitAck();
    I2CSendByte(add);
    I2CWaitAck();
    I2CStop();

    I2CStart();
    I2CSendByte(0x39);
    I2CWaitAck();
    data = I2CReceiveByte();
    I2CSendNotAck();
    I2CStop();
    return data;
}
```

> 写

```
void LIS_write(unsigned char add,unsigned char data)
{
    I2CStart();
    I2CSendByte(0x38);
    I2CWaitAck();
    I2CSendByte(add);
    I2CWaitAck();
    I2CSendByte(data);
    I2CWaitAck();
    I2CStop();
}
```

> SDO接地，故0x38为写，0x39为读

Command	SAD[6:1]	SAD[0] = SDO	R/W	SAD+R/W
Read	001110	0	1	00111001 (39h)
Write	001110	0	0	00111000 (38h)
Read	001110	1	1	00111011 (3Bh)
Write	001110	1	0	00111010 (3Ah)

2.0x0f寄存器

> 0x0f为设备标识寄存器，此寄存器包含用于LIS302DL的设备标识符设置为3B。

WHO_AM_I (0Fh)

Table 17. Register

0	0	1	1	1	0	1	1

Device identification register. This register contains the device identifier that for LIS302DL is set to 3Bh.

> 可在初始化时进行校验：

```
while(!LIS_read(0x0f))
{
    LCD_DisplayStringLine(Line5, (uint8_t *)"ERROR               ");
}
snprintf((char*)text,20,"OK!!!,%X               ",LIS_read(0x0f));
```

3.0x20寄存器

> 同时，初始化时应设置0x20寄存器为0x47，激活设备、开启x、y、z轴测量。

CTRL_REG1 (20h)

Table 18. Register

DR	PD	FS	STP	STM	Zen	Yen	Xen

Table 19. Register description

DR	Data rate selection. Default value: 0 (0: 100 Hz output data rate; 1: 400 Hz output data rate)
PD	Power Down Control. Default value: 0 (0: power down mode; 1: active mode)
FS	Full Scale selection. Default value: 0 (refer to *Table 3* for typical full scale value)
STP, STM	Self Test Enable. Default value: 0 (0: normal mode; 1: self test P, M enabled)
Zen	Z axis enable. Default value: 1 (0: Z axis disabled; 1: Z axis enabled)
Yen	Y axis enable. Default value: 1 (0: Y axis disabled; 1: Y axis enabled)
Xen	X axis enable. Default value: 1 (0: X axis disabled; 1: X axis enabled)

```
LIS_write(0x20,0x47);
```

4.0x27寄存器

> 0x27的第4位为数据监测为，有新的数据时为1

Table 26. Register

ZXYOR	ZOR	YOR	XOR	ZYXDA	ZDA	YDA	XDA

Table 27. Register description

ZYXOR	X, Y and Z axis data overrun. Default value: 0 (0: no overrun has occurred; 1: new data has over written the previous one before it was read)
ZOR	Z axis data overrun. Default value: 0 (0: no overrun has occurred; 1: a new data for the Z-axis has overwritten the previous one)
YOR	Y axis data overrun. Default value: 0 (0: no overrun has occurred; 1: a new data for the Y-axis has overwritten the previous one)
XOR	X axis data overrun. Default value: 0 (0: no overrun has occurred; 1: a new data for the X-axis has overwritten the previous one)
ZYXDA	X, Y and Z axis new data available. Default value: 0 (0: a new set of data is not yet available; 1: a new set of data is available)
ZDA	Z axis new data available. Default value: 0 (0: a new data for the Z-axis is not yet available; 1: a new data for the Z-axis is available)
YDA	Y axis new data available. Default value: 0 (0: a new data for the Y-axis is not yet available; 1: a new data for the Y-axis is available)
XDA	X axis new data available. Default value: 0 (0: a new data for the X-axis is not yet available; 1: a new data for the X-axis is available)

> 在新数据到来时进行读取,并进行处理

```
if(LIS_read(0x27) & 0x08)
{
    x = LIS_read(0x29) * 18.0 / 100;
    y = LIS_read(0x2B) * 18.0 / 100;
    z = LIS_read(0x2D) * 18.0 / 100;
}
```

5.数据存储寄存器

OUT_X (29h)

Table 28. Register

XD7	XD6	XD5	XD4	XD3	XD2	XD1	XD0

OUT_Y (2Bh)

Table 29. Register description

YD7	YD6	YD5	YD4	YD3	YD2	YD1	YD0

OUT_Z (2Dh)

Table 30. Register

ZD7	ZD6	ZD5	ZD4	ZD3	ZD2	ZD1	ZD0

第十届国赛真题复盘

1.使用到的模块

（1）ADC：两路ADC电压检测

（2）PWM输入捕获：检测拓展版上脉冲的占空比

```
void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{
    if(htim->Instance == TIM3)
    {
        if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2)
        {
            val_a = HAL_TIM_ReadCapturedValue(&htim3,TIM_CHANNEL_1);
            val_b = HAL_TIM_ReadCapturedValue(&htim3,TIM_CHANNEL_2);
            __HAL_TIM_SetCounter(&htim3,0);
            duty = (unsigned char)(val_a/val_b*100);
            HAL_TIM_IC_Start(&htim3,TIM_CHANNEL_1);
            HAL_TIM_IC_Start(&htim3,TIM_CHANNEL_2);
        }
    }
}
```

(3) 按键：短按键+长按键

(4) UART+DMA：不定长数据接收

> 首先初始化空闲中断以及DMA接收

```
HAL_UART_Receive_DMA(&huart1,(u8 *)text,strlen(text));
__HAL_UART_ENABLE_IT(&huart1,UART_IT_IDLE);
```

> 然后再中断服务函数内进行判断

```
void USART1_IRQHandler(void)
{
    /* USER CODE BEGIN USART1_IRQn 0 */

    /* USER CODE END USART1_IRQn 0 */
    HAL_UART_IRQHandler(&huart1);
    /* USER CODE BEGIN USART1_IRQn 1 */
    if(__HAL_UART_GET_FLAG(&huart1,UART_FLAG_IDLE) != RESET)
    {
        __HAL_UART_CLEAR_IDLEFLAG(&huart1);
        HAL_UART_DMAStop(&huart1);
        unsigned char num;
        num = 100 -     __HAL_DMA_GET_COUNTER(&hdma_usart1_rx);
        //你想要做的事情

        HAL_UART_Receive_DMA(&huart1,(u8 *)text,strlen(text));
    }
    /* USER CODE END USART1_IRQn 1 */
}
```

(5) LED：互不干扰的LED闪烁、亮灭

(6) eeprom：16位数据的存储

(7) DS18B20：温度数据的读取

> 复位->发送skip-->发送convetr-->复位-->发送skip-->发送读取指令-->连续读两个（第一个是低位，第二个是高位-->正负判断-->输出）

```c
double Read_temperature(void)
{
    uint8_t h,l;
    uint16_t tem;
    double Tem;

    ow_reset();
    ow_byte_wr(OW_SKIP_ROM);
    ow_byte_wr(DS18B20_CONVERT);

    ow_reset();
    ow_byte_wr(OW_SKIP_ROM);
    ow_byte_wr(DS18B20_READ);

    l = ow_byte_rd();
    h = ow_byte_rd();
    tem = (h<<8) + l;
    if((tem & 0xf800) == 0xf800)
    {
        tem = (~tem) + 1;
        Tem = tem * -0.0625;
    }
    else Tem = tem * 0.0625;
    return Tem;
}
```

(8) 数码管：参数显示

> 三位数码管从低位开始传入，但是每个位的八段需要从高位开始传输（见原理图）
>
> 下面给出示例

```
void LED_DISP(uint16_t data)
{
    unsigned char tep;
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_2,GPIO_PIN_RESET);
    for(int i=0;i<3;i++)
    {
        if(data == 0 && i>0) tep = num[10];
        else tep = num[data%10];
        data /= 10;

        for(int j=0;j<8;j++)
        {
            HAL_GPIO_WritePin(GPIOA,GPIO_PIN_3,GPIO_PIN_RESET);
            if(tep & 0x80) HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1,GPIO_PIN_SET);
            else HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1,GPIO_PIN_RESET);
            tep = tep << 1;
            HAL_GPIO_WritePin(GPIOA,GPIO_PIN_3,GPIO_PIN_SET);
        }
    }
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_2,GPIO_PIN_SET);
}
```

2.使用到的技巧

（1）定时闪烁、发送数据

> 共有四个地方使用到了这一部分技巧，给出模板如下：

```
bool flag;
unsigned char time;
//在判断处确定flag的值，并且对time归0
if(flag == 1)
{
    time++;
    if(time >= 20)
    {
        time = 0;
        //你要做的事
    }
}
```

（2）高亮行

> 这一部分属于常考的内容，需要知道，LCD屏幕的数据显示，背景与前景就相当于画笔，你换了颜色他就画不同的颜色，画完这一行再换回去就实现了刚画的那一行和其它行的颜色不一样的效果了；

```
if(line == 0) LCD_SetBackColor(Yellow);
sprintf(text,"    T:%d              ",T);
LCD_DisplayStringLine(Line2,(u8 *)text);
LCD_SetBackColor(White);

if(line == 1) LCD_SetBackColor(Yellow);
sprintf(text,"    X:AO%d            ",AO+1);
LCD_DisplayStringLine(Line3,(u8 *)text);
LCD_SetBackColor(White);
```

第十二届国赛真题复盘

1.使用到的模块

（1）频率、占空比测量

（2）光敏电阻ADC测量

（3）LED控制

（4）DMA+串口不定长数据接收

2.使用到的技巧

（1）冒泡法排序

> K为数据的个数

```
for(int i=0;i<k-1;i++)
{
    for(int j=0;j<k-i-1;j++)
    {
        if(sort[j]>sort[j+1])
        {
            c = sort[j];
            sort[j] = sort[j+1];
            sort[j+1] = c;
        }
    }
}
```

（2）单通道测量占空比

> 思路就是：每次进入中断都改变一次中断触发方式，三次后就表明一个周期的测量完成了，关闭测量，在外部进行计算，计算完成后再开启中断，进行下一次测量。
>
> __HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_1,触发方式：初始化里面找）；
>
> HAL_TIM_IC_Stop(htim,TIM_CHANNEL_1);

```
if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_1)
{
    switch(tim17_flag)
    {
        case 0:
        {
            val17[0] = HAL_TIM_ReadCapturedValue(htim,TIM_CHANNEL_1);
            tim17_flag=1;

__HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_1,TIM_INPUTCHANNELPOLARITY_FALLING);
        }
        break;
        case 1:
        {
            val17[1] = HAL_TIM_ReadCapturedValue(htim,TIM_CHANNEL_1);
            tim17_flag=2;

__HAL_TIM_SET_CAPTUREPOLARITY(htim,TIM_CHANNEL_1,TIM_INPUTCHANNELPOLARITY_RISING);
        }
        break;
        case 2:
        {
            val17[2] = HAL_TIM_ReadCapturedValue(htim,TIM_CHANNEL_1);
            tim17_flag=3;
            __HAL_TIM_SetCounter(htim,0);
            HAL_TIM_IC_Stop(htim,TIM_CHANNEL_1);
        }
        break;
    }
}
```

> 计算并开启中断

```
b[0] = ((val17[1]-val17[0])/(val17[2]-val17[0])*100 - 10) * 9 / 8;
b[0] = b[0] > 90?90:b[0];
b[0] = b[0] < 0?0:b[0];
__HAL_TIM_SET_CAPTUREPOLARITY(&htim17,TIM_CHANNEL_1,TIM_INPUTCHANNELPOLARITY_RISING);
HAL_TIM_IC_Start_IT(&htim17,TIM_CHANNEL_1);
tim17_flag = 0;//标志位
```

第十三届国赛真题复盘

1.使用到的模块

（1）长按键

```
void keyy(void)
{
    key[0].key_sta = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_0);
    key[1].key_sta = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_1);
    key[2].key_sta = HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_2);
    key[3].key_sta = HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_0);
    for(int i=0;i<4;i++)
    {
        switch(key[i].judge_sta)
        {
            case 0:
            {
                if(key[i].key_sta == 0) key[i].judge_sta = 1;
            }
            break;
            case 1:
            {
                if(key[i].key_sta == 0) key[i].judge_sta = 2;
                else key[i].judge_sta = 0;
            }
            break;
            case 2:
            {
                key[i].time_sta++;
                if(key[i].key_sta == 1 && key[i].time_sta < 100)
                {
                    key[i].single_sta = 1;
                    key[i].time_sta = 0;
                    key[i].judge_sta = 0;
                }
                else if(key[i].key_sta == 0 && key[i].time_sta >= 100)
key[i].judge_sta = 3;
            }
            break;
            case 3:
            {
                if(key[i].time_sta >= 100)
                {
                    key[i].time_sta = 0;
                    key[i].long_sta = 1;
                }
                if(key[i].key_sta == 1) key[i].judge_sta = 0;
            }
            break;
        }
    }
}
```

（2）不定长数据接收

> 首先是初始化
>
> 然后转到中断服务函数内编写

```
//初始化代码
HAL_UART_Receive_DMA(&huart1,(uint8_t *)rxdata,100);
__HAL_UART_ENABLE_IT(&huart1,UART_IT_IDLE);
//中断服务函数内代码
if(__HAL_UART_GET_FLAG(&huart1,UART_FLAG_IDLE) != RESET)
{
    __HAL_UART_CLEAR_IDLEFLAG(&huart1);
    HAL_UART_DMAStop(&huart1);
    unsigned char num=0;
    num = 100-__HAL_DMA_GET_COUNTER(&hdma_usart1_rx);
    //你要做的事
    HAL_UART_Receive_DMA(&huart1,(uint8_t *)rxdata,100);
}
```

（3）频率检测

（4）ADC电压测量

（5）eeprom读写

（6）LED控制

（7）PWM输出

2.使用到的技巧

（1）LED闪烁

```
//老方法
unsigned char flag,flagtime,ledflag;
if(flag == 1)
{
    flagtime++;
    if(flagtime >= 10)
    {
        //你要做的操作
        flagtime = 0;
    }
}
```

（2）LCD底层驱动的理解

> 本次比赛中要求翻转LCD显示的方向，需要了解下面几个函数
>
> void LCD_DisplayStringLine(u8 Line, u8 *ptr),输入为行和要打印的字符串的首地址

```
void LCD_DisplayStringLine(u8 Line, u8 *ptr)
{
    if(lcd_sta == 0)
    {
        u32 i = 0;
        u16 refcolumn = 319;//319: 列地址
        while ((*ptr != 0) && (i < 20))     //    20: 打印字符数
        {
            LCD_DisplayChar(Line, refcolumn, *ptr);
            refcolumn -= 16;
            ptr++;
            i++;
        }
    }
    else
    {
        Line = 216 - Line;
        u32 i = 0;
        u16 refcolumn = 0;//319;
        while ((*ptr != 0) && (i < 20))     //    20
        {
            LCD_DisplayChar(Line, refcolumn, *ptr);
            refcolumn += 16;
            ptr++;
            i++;
        }
    }
}
```

> 函数LCD_DrawChar(u8 Xpos, u16 Ypos, uc16 *c)用于绘制单个字符
>
> 输入参数为行、列以及字符地址

```
void LCD_DrawChar(u8 Xpos, u16 Ypos, uc16 *c)
{
    u32 index = 0, i = 0;
    u8 Xaddress = 0;

    Xaddress = Xpos;
    LCD_SetCursor(Xaddress, Ypos);
    if(lcd_sta == 0)
    {
        for(index = 0; index < 24; index++)
        {
            LCD_WriteRAM_Prepare(); /* Prepare to write GRAM */
            for(i = 0; i < 16; i++)
            {
                if((c[index] & (1 << i)) == 0x00)
                {
                    LCD_WriteRAM(BackColor);
                }
                else
                {
                    LCD_WriteRAM(TextColor);
                }
            }
            Xaddress++;
            LCD_SetCursor(Xaddress, Ypos);
        }
    }
    else
    {
        for(index = 24; index >= 1; index--)
        {
            LCD_WriteRAM_Prepare(); /* Prepare to write GRAM */
            for(i = 16; i >= 1; i--)
            {
                if((c[index-1] & (1 << (i-1))) == 0x00)
                {
                    LCD_WriteRAM(BackColor);
                }
                else
                {
                    LCD_WriteRAM(TextColor);
                }
            }
            Xaddress++;
            LCD_SetCursor(Xaddress, Ypos);
        }
    }
}
```

> 思考：如何旋转显示?