Main.c

```
#include "stm32f10x.h"
#include "sys.h"
#include "delay.h"
#include "math.h"
#include "usart1.h"
#include "adc1.h"
#include "hcsr04.h"
#include "relay.h"
#include "stdio.h"
extern __IO uint16_t ADCConvertedValuex;
extern __IO uint16_t ADCConvertedValuey;
extern __IO uint16_t ADCConvertedValuez;
#define Max_roll 290 //??
#define Min_roll 260 //??
int main(void)
{
	float length, voltagex,voltagey,voltagez;//???
	uint16_t adc_roll;
	delay_init();
	NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2); //?? NVIC ???? 2
	ADC_Configure();
	RELAY_Configure();
	USART1_Configure();//USART? ? ?
	Hcsr04Init();
	delay_ms(1000);// ?????????
	while(1)
	{
		length =Hcsr04GetLength();
		voltagex =(float)ADCConvertedValuex*(3.3/4096)-1.70;//AD???????
		voltagey =(float)ADCConvertedValuey*(3.3/4096)-1.70;//AD???????
		voltagez =(float)ADCConvertedValuez*(3.3/4096)-1.61;//AD???????
		adc_roll = (uint16_t)((atan2(voltagez,-voltagey)*57.2957796)+180);
		delay_ms(2000);
		printf("???:%.3f??\r\n\r\n",length);
		printf("X?????:%.2fV\r\n",voltagex);//?????,????
		printf("y?????:%.2fV\r\n",voltagey);//?????,????
		printf("Z?????:%.2fV\r\n",voltagez);//?????,????
		printf("??????:%dV\r\n",adc_roll);//
		if(length >=10.0)
		{
			RELAY1_On();
		}
		else if(length <=6.0)
```

```
            {
                RELAY1_Off();
            }
            if((adc_roll > Min_roll) && (adc_roll < Max_roll))//??????????,???
            {
                RELAY2_Off();                                    //
            }
            else
            {
                RELAY2_On();
            }
ADC_SoftwareStartInjectedConvCmd(ADC1, ENABLE);
      }
}
```

Usart.c

```
#include "usart1.h"
#include "stdio.h"
uint8_t USART1_RX_Buffer[USART_RX_MAX] = { 0 }; //定义 1.USART1 接收缓存
uint8_t USART1_RX_Index = 0; //定义 2.USART1 接收数组下标
uint8_t USART1_RX_OverFlag = 0; //定义 3.USART1 接收完成标志位
/**
  * @简介：将 C 库中 printf 重定向到 USART
  * @参数: ch-待发送字符，f-指定文件
  * @返回值：ch
  */
int fputc(int ch, FILE *f)
{
    USART_SendData(USART1, (u8) ch);
    while(!(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == SET))
    {
    }
    return ch;
}

void USART1_Configure(void)
{
     /* 定义 GPIO 初始化结构体 */
     GPIO_InitTypeDef GPIO_InitStructure;
     /* 定义 USART 初始化结构体 */
     USART_InitTypeDef USART_InitStructure;
```

```
    NVIC_InitTypeDef NVIC_InitStructure;
    /* 打开 GPIOA、AFIO 和 USART1 时钟 */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA | RCC_APB2Periph_AFIO |
RCC_APB2Periph_USART1, ENABLE);
    /* 配置 PA9(USART_Tx)为开漏输出，IO 速度 50MHz */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
    /* 完成配置 */
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    /* 配置 PA10(USART1_Rx)为浮空输入 */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    /* 完成配置 */
    GPIO_Init(GPIOA, &GPIO_InitStructure);
    /* 配置 USART 波特率、数据位、停止位、奇偶校验、硬件流控制和模式 */
    USART_InitStructure.USART_BaudRate = 4800;//波特率 4800
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;//8 数据位
    USART_InitStructure.USART_StopBits = USART_StopBits_1;//1 停止位
    USART_InitStructure.USART_Parity = USART_Parity_No;//无奇偶校验
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;//
无硬件流控制
    USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;//接收和发送模
式
    /* 完成配置 */
    USART_Init(USART1, &USART_InitStructure);
    /* 使能 USART1 */
    USART_Cmd(USART1, ENABLE);
    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE); //开启接收 RXNE 中断
    USART_ITConfig(USART1, USART_IT_IDLE, ENABLE); //开启接收 IDLE 中断
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn; //USART1 中断通道
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0; //抢占优先级 1
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1; //子优先级 3
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //IRQ 通道使能
    NVIC_Init(&NVIC_InitStructure); //配置生效
}

void USART1_IRQHandler(void)
{
    uint8_t Res;
    /* 如果发生了接收中断 */
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
    {
        //Res = USART1->DR; //寄存器方式读取数据
```

```
        Res = USART_ReceiveData(USART1); //库函数方式读取接收到的 1 个字节
        if(USART1_RX_Index >= USART_RX_MAX)
        USART1_RX_Index = 0; //防止下标越界
        USART1_RX_Buffer[USART1_RX_Index++] = Res;
        /* 清除接收中断标志位(注:也可以省略，读 DR 自动清除) */
        USART_ClearFlag(USART1, USART_FLAG_RXNE);
    }
    if(USART_GetITStatus(USART1, USART_IT_IDLE) != RESET)
    {
        USART1_RX_OverFlag = 1; //接收完成标志位置 1
        USART_ClearFlag(USART1, USART_FLAG_IDLE);
        USART_ITConfig(USART1, USART_IT_IDLE, DISABLE); //关闭接收 IDLE 中断
    }
}
```

Usart.h

```
#ifndef __USART1_H
#define __USART1_H
#include "stm32f10x.h"
#define USART_RX_MAX 255 //定义最大接收字节数为 255
extern uint8_t USART1_RX_Buffer[USART_RX_MAX]; //定义 1.USART1 接收缓存
extern uint8_t USART1_RX_Index; //定义 2.USART1 接收数组下标
extern uint8_t USART1_RX_OverFlag; //定义 3.USART1 接收完成标志位
void USART1_Configure(void);
#endif
```

Relay.c

```
#include "relay.h"
void RELAY_Configure(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOD|RCC_APB2Periph_GPIOC,ENABLE);
    GPIO_InitStructure.GPIO_Pin =GPIO_Pin_2;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;
    GPIO_Init(GPIOD,&GPIO_InitStructure);
```

```
    GPIO_InitStructure.GPIO_Pin =GPIO_Pin_12;
    GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode=GPIO_Mode_Out_PP;
    GPIO_Init(GPIOC,&GPIO_InitStructure);
    GPIO_ResetBits(GPIOD,GPIO_Pin_2);
    GPIO_ResetBits(GPIOC,GPIO_Pin_12);
}

void RELAY1_On(void)
{
    GPIO_SetBits(GPIOC,GPIO_Pin_12);
}

void RELAY1_Off(void)
{
    GPIO_ResetBits(GPIOC,GPIO_Pin_12);
}

void RELAY2_On(void)
{
    GPIO_SetBits(GPIOD,GPIO_Pin_2);
}

void RELAY2_Off(void)
{
    GPIO_ResetBits(GPIOD,GPIO_Pin_2);
}
```

Relay.h

```
#ifndef _RELAY_H
#define _RELAY_H
#include "stm32f10x.h"
#include "delay.h"
void RELAY_Configure(void);//LED 引脚初始化
void RELAY1_On(void);
void RELAY1_Off(void);
void RELAY2_On(void);
void RELAY2_Off(void);
#endif
```

Hcsr04.c

```
#include "hcsr04.h"

#define HCSR04_PORT    GPIOA
#define HCSR04_CLK     RCC_APB2Periph_GPIOA
#define HCSR04_TRIG     GPIO_Pin_5
#define HCSR04_ECHO    GPIO_Pin_6
#define TRIG_Send       PAout(5)
#define ECHO_Reci       PAin(6)
u16 msHcCount =0;
void Hcsr04Init(void)
{
    GPIO_InitTypeDef    GPIO_InitStructure;
    NVIC_InitTypeDef    NVIC_InitStructure;
    TIM_TimeBaseInitTypeDef    TIM_TimeBasestructure;
    RCC_APB2PeriphClockCmd(HCSR04_CLK,ENABLE);
    //IO 口初始化
    GPIO_InitStructure.GPIO_Pin = HCSR04_TRIG;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_OD;//设置 I0 口输出模式为开漏输出
    GPIO_Init(HCSR04_PORT,&GPIO_InitStructure);
    GPIO_ResetBits(HCSR04_PORT,HCSR04_TRIG);
    //IO 口初始化
    GPIO_InitStructure.GPIO_Pin =HCSR04_ECHO;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING; //设置 I0 口输入模式浮空输
    GPIO_Init(HCSR04_PORT, &GPIO_InitStructure);
    GPIO_ResetBits(HCSR04_PORT,HCSR04_ECHO);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3,ENABLE); //使能定时器时钟
    TIM_TimeBasestructure.TIM_Period =(999);//设置在下一个更新事件装入活动的自动重
    TIM_TimeBasestructure.TIM_Prescaler=(72-1); //设置用来作为 TIM3 时钟频率除数的预
    TIM_TimeBasestructure.TIM_ClockDivision=TIM_CKD_DIV1;//设置时钟分割
    TIM_TimeBasestructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM3, &TIM_TimeBasestructure);
    TIM_ClearFlag(TIM3, TIM_FLAG_Update);
    TIM_ITConfig(TIM3,TIM_IT_Update,ENABLE);//使能指定的 TIM3 中断，打开更新中断
    //中断优先级 NVIC 设置
    NVIC_InitStructure.NVIC_IRQChannel=TIM3_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority =2;//设置抢占优先级
    NVIC_InitStructure.NVIC_IRQChannelSubPriority =0;//设置从优先级
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //IRQ 通道使能
    NVIC_Init(&NVIC_InitStructure);//初始化 NVIC 寄存器
```

```
    TIM_Cmd(TIM3,DISABLE);
}
void OpenTimer()
{
    TIM_SetCounter(TIM3,0);//设置 TIM3 计数寄存器的值
    msHcCount =0;
    TIM_Cmd(TIM3,ENABLE);//使能定时器 TIM3
}
void closeTimer()
{
    TIM_Cmd(TIM3,DISABLE);//停止使能定时器 TIM3
}
//定时器 3 中断服务程序
void TIM3_IRQHandler(void)
{
    if(TIM_GetITStatus(TIM3,TIM_IT_Update)!= RESET)//检查 TIM3 的更新中断是否发生
    {
        TIM_ClearITPendingBit(TIM3,TIM_IT_Update );//清除 TIM3 更新中断标志
        msHcCount++;
    }
}
u32 GetEchoTimer(void)
{
    u32   t=0;
    t = msHcCount*1000;//将时间转化为微秒!
    t +=   TIM_GetCounter(TIM3);//获取当前计数器的值
    TIM3->CNT =0;//清零计数器
    delay_ms(50);
    return t;
}
float Hcsr04GetLength(void)
{
    u32   t = 0;
    int16_t   i = 0;
    float   lengthTemp = 0;
    float   sum = 0;
    while(i!=5)
    {
        TRIG_Send=0;//IO 口 PA5 输出一个低电平
        while(ECHO_Reci == 0);
        OpenTimer();//开始计时
        i=i+1;
        while(ECHO_Reci == 1);
        closeTimer();//结束计时
```

```
            TRIG_Send = 1;//IOLIPA5 输出一个高电平
            t = GetEchoTimer();
            lengthTemp=((float)t*0.017);//计算单次物体距离
            sum = lengthTemp + sum;
            delay_ms(10);
        }
        lengthTemp = sum/5.0;//求五次距离平均值
        return lengthTemp;
}
```

Hcsr04.h

```
#ifndef    __HCSR04_H
#define    __HCSR04_H
#include "sys.h"
#include "delay.h"
float Hcsr04GetLength(void);
void Hcsr04Init(void);
#endif
```

Adc1.c

```
#include "adc1.h"
__IO uint16_t ADCConvertedValuex;
__IO uint16_t ADCConvertedValuey;
__IO uint16_t ADCConvertedValuez;
void ADC_Configure(void)
{
    /* 定义 GPIO 和 ADC 初始化结构体 */
    GPIO_InitTypeDef GPIO_InitStructure;
    ADC_InitTypeDef ADC_InitStructure;
    /* 使能时钟，并配置 PB0、PB1 为模拟输入 */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOC
|RCC_APB2Periph_AFIO | RCC_APB2Periph_ADC1, ENABLE);
    NVIC_InitTypeDef NVIC_InitStructure;
    NVIC_InitStructure.NVIC_IRQChannel = ADC1_2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
```

```
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_1;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOB, &GPIO_InitStructure);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
    RCC_ADCCLKConfig(RCC_PCLK2_Div6);
    /* 设置 ADC 工作模式：独立、扫描、连续、不使用外部触发、数据右对齐、1 个转换 */
    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;//独立
    ADC_InitStructure.ADC_ScanConvMode = ENABLE;//扫描
    ADC_InitStructure.ADC_ContinuousConvMode =DISABLE;//不连续
    //ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;//无外部触发
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;//数据右对齐
    //ADC_InitStructure.ADC_NbrOfChannel = 1;//3 个转换
    /* 完成配置 */
    ADC_Init(ADC1, &ADC_InitStructure);
    ADC_InjectedSequencerLengthConfig(ADC1, 3);
    /* 配置 ADC1 转换通道，PB0 对应通道 8 */
    ADC_InjectedChannelConfig(ADC1, ADC_Channel_8, 1, ADC_SampleTime_55Cycles5);
    /* 配置 ADC1 转换通道，PB1 对应通道 9 */
    ADC_InjectedChannelConfig(ADC1, ADC_Channel_9, 2, ADC_SampleTime_55Cycles5);
    /* 配置 ADC1 转换通道，PC0 对应通道 10 */
    ADC_InjectedChannelConfig(ADC1, ADC_Channel_10, 3, ADC_SampleTime_55Cycles5);
    ADC_ExternalTrigInjectedConvConfig(ADC1, ADC_ExternalTrigInjecConv_None);
    /* 使能 ADC1 对应的 DMA */
    //ADC_DMACmd(ADC1, ENABLE);
//  ADC_SetInjectedOffset(ADC1, ADC_InjectedChannel_1, 0x100);
//  ADC_SetInjectedOffset(ADC1, ADC_InjectedChannel_2, 0x100);
//  ADC_SetInjectedOffset(ADC1, ADC_InjectedChannel_3, 0x100);
    ADC_ITConfig(ADC1, ADC_IT_JEOC, ENABLE);
    /* 使能 ADC1 */
    ADC_Cmd(ADC1, ENABLE);
    /* 复位 ADC1 的校准寄存器 */
    ADC_ResetCalibration(ADC1);
    /*等待 ADC 校准寄存器复位完成*/
    while(ADC_GetResetCalibrationStatus(ADC1));
    /* 开始校准 ADC */
    ADC_StartCalibration(ADC1);
    /* 等待校准完成*/
    while(ADC_GetCalibrationStatus(ADC1));
    /* 软件方式触发 ADC */
```

```
    ADC_SoftwareStartInjectedConvCmd(ADC1, ENABLE);
}

void ADC1_2_IRQHandler(void)
{
    if(ADC_GetITStatus(ADC1,ADC_IT_JEOC))
    {
        ADCConvertedValuex=ADC_GetInjectedConversionValue(ADC1,
ADC_InjectedChannel_1);
        ADCConvertedValuey=ADC_GetInjectedConversionValue(ADC1,
ADC_InjectedChannel_2);
        ADCConvertedValuez=ADC_GetInjectedConversionValue(ADC1,
ADC_InjectedChannel_3);
    }
    ADC_ClearITPendingBit(ADC1, ADC_IT_JEOC);
}
```

Adc1.h

```
#ifndef __ADC1_H
#define __ADC1_H
#include "stm32f10x.h"
void ADC_Configure(void);
#endif
```

Delay.c

```
#include "delay.h"

static u8  fac_us=0;                          //us 延时倍乘数
static u16 fac_ms=0;                          //ms 延时倍乘数,在 ucos 下,代表每个节
拍的 ms 数


#if SYSTEM_SUPPORT_OS                         //如果 SYSTEM_SUPPORT_OS 定义了,
说明要支持 OS 了(不限于 UCOS).
//当 delay_us/delay_ms 需要支持 OS 的时候需要三个与 OS 相关的宏定义和函数来支持
//首先是 3 个宏定义:
//    delay_osrunning:用于表示 OS 当前是否正在运行,以决定是否可以使用相关函数
//delay_ostickspersec:用于表示 OS 设定的时钟节拍,delay_init 将根据这个参数来初始哈
systick
```

```
// delay_osintnesting:用于表示 OS 中断嵌套级别,因为中断里面不可以调度,delay_ms 使用该参数来决定如何运行
//然后是 3 个函数:
//    delay_osschedlock:用于锁定 OS 任务调度,禁止调度
//delay_osschedunlock:用于解锁 OS 任务调度,重新开启调度
//       delay_ostimedly:用于 OS 延时,可以引起任务调度.

//本例程仅作 UCOSII 和 UCOSIII 的支持,其他 OS,请自行参考着移植
//支持 UCOSII
#ifdef      OS_CRITICAL_METHOD                          //OS_CRITICAL_METHOD 定义了,说明要支持 UCOSII
#define delay_osrunning             OSRunning           //OS 是否运行标记,0,不运行;1,在运行
#define delay_ostickspersec   OS_TICKS_PER_SEC//OS 时钟节拍,即每秒调度次数
#define delay_osintnesting    OSIntNesting          //中断嵌套级别,即中断嵌套次数
#endif

//支持 UCOSIII
#ifdef      CPU_CFG_CRITICAL_METHOD                       //CPU_CFG_CRITICAL_METHOD 定义了,说明要支持 UCOSIII
#define delay_osrunning             OSRunning           //OS 是否运行标记,0,不运行;1,在运行
#define delay_ostickspersec   OSCfg_TickRate_Hz //OS 时钟节拍,即每秒调度次数
#define delay_osintnesting    OSIntNestingCtr           //中断嵌套级别,即中断嵌套次数
#endif


//us 级延时时,关闭任务调度(防止打断 us 级延迟)
void delay_osschedlock(void)
{
#ifdef CPU_CFG_CRITICAL_METHOD                          //使用 UCOSIII
    OS_ERR err;
    OSSchedLock(&err);                                  //UCOSIII 的方式,禁止调度，防止打断 us 延时
#else                                                   //否则 UCOSII
    OSSchedLock();                                      //UCOSII 的方式,禁止调度，防止打断 us 延时
#endif
}

//us 级延时时,恢复任务调度
void delay_osschedunlock(void)
{
#ifdef CPU_CFG_CRITICAL_METHOD                          //使用 UCOSIII
```

```
    OS_ERR err;
    OSSchedUnlock(&err);                    //UCOSIII 的方式,恢复调度
#else                                       //否则 UCOSII
    OSSchedUnlock();                        //UCOSII 的方式,恢复调度
#endif
}

//调用 OS 自带的延时函数延时
//ticks:延时的节拍数
void delay_ostimedly(u32 ticks)
{
#ifdef CPU_CFG_CRITICAL_METHOD
    OS_ERR err;
    OSTimeDly(ticks,OS_OPT_TIME_PERIODIC,&err); //UCOSIII 延时采用周期模式
#else
    OSTimeDly(ticks);                       //UCOSII 延时
#endif
}

//systick 中断服务函数,使用 ucos 时用到
void SysTick_Handler(void)
{
    if(delay_osrunning==1)                  //OS 开始跑了,才执行正常的调度处理
    {
        OSIntEnter();                       //进入中断
        OSTimeTick();                       //调用 ucos 的时钟服务程序
        OSIntExit();                        //触发任务切换软中断
    }
}
#endif


//初始化延迟函数
//当使用 OS 的时候,此函数会初始化 OS 的时钟节拍
//SYSTICK 的时钟固定为 HCLK 时钟的 1/8
//SYSCLK:系统时钟
void delay_init()
{
#if SYSTEM_SUPPORT_OS                       //如果需要支持 OS.
    u32 reload;
#endif
    SysTick_CLKSourceConfig(SysTick_CLKSource_HCLK_Div8); //选择外部时钟    HCLK/8
    fac_us=SystemCoreClock/8000000;         //为系统时钟的 1/8
#if SYSTEM_SUPPORT_OS                       //如果需要支持 OS.
```

```
    reload=SystemCoreClock/8000000;                 //每秒钟的计数次数 单位为 M
    reload*=1000000/delay_ostickspersec;    //根据 delay_ostickspersec 设定溢出时间
                                                    //reload 为 24 位寄存器,最大值:16777216,在 72M 下,约合 1.86s 左右
    fac_ms=1000/delay_ostickspersec;            //代表 OS 可以延时的最少单位

    SysTick->CTRL|=SysTick_CTRL_TICKINT_Msk;    //开启 SYSTICK 中断
    SysTick->LOAD=reload;                       // 每 1/delay_ostickspersec 秒中断一次

    SysTick->CTRL|=SysTick_CTRL_ENABLE_Msk;     //开启 SYSTICK

#else
    fac_ms=(u16)fac_us*1000;                    //非 OS 下,代表每个 ms 需要的 systick 时钟数
#endif
}

#if SYSTEM_SUPPORT_OS                           //如果需要支持 OS.
//延时 nus
//nus 为要延时的 us 数.
void delay_us(u32 nus)
{
    u32 ticks;
    u32 told,tnow,tcnt=0;
    u32 reload=SysTick->LOAD;                   //LOAD 的值
    ticks=nus*fac_us;                           //需要的节拍数
    tcnt=0;
    delay_osschedlock();                        //阻止 OS 调度，防止打断 us 延时
    told=SysTick->VAL;                          //刚进入时的计数器值
    while(1)
    {
        tnow=SysTick->VAL;
        if(tnow!=told)
        {
            if(tnow<told)tcnt+=told-tnow;       //这里注意一下 SYSTICK 是一个递减的计数器就可以了.
            else tcnt+=reload-tnow+told;
            told=tnow;
            if(tcnt>=ticks)break;               //时间超过/等于要延迟的时间,则退出.
        }
    };
    delay_osschedunlock();                      //恢复 OS 调度

}
```

```
//延时 nms
//nms:要延时的 ms 数
void delay_ms(u16 nms)
{
    if(delay_osrunning&&delay_osintnesting==0)     //如果 OS 已经在跑了,并且不是在中断里面(中断里面不能任务调度)
    {
        if(nms>=fac_ms)                            //延时的时间大于 OS 的最少时间周期
        {
            delay_ostimedly(nms/fac_ms);           //OS 延时
        }
        nms%=fac_ms;                               //OS 已经无法提供这么小的延时了,采用普通方式延时
    }
    delay_us((u32)(nms*1000));                     //普通方式延时
}
#else //不用 OS 时
//延时 nus
//nus 为要延时的 us 数.
void delay_us(u32 nus)
{
    u32 temp;
    SysTick->LOAD=nus*fac_us;                      //时间加载
    SysTick->VAL=0x00;                             //清空计数器
    SysTick->CTRL|=SysTick_CTRL_ENABLE_Msk ;       //开始倒数
    do
    {
        temp=SysTick->CTRL;
    }while((temp&0x01)&&!(temp&(1<<16)));          //等待时间到达
    SysTick->CTRL&=~SysTick_CTRL_ENABLE_Msk;       //关闭计数器
    SysTick->VAL =0X00;                             //清空计数器
}
//延时 nms
//注意 nms 的范围
//SysTick->LOAD 为 24 位寄存器,所以,最大延时为:
//nms<=0xffffff*8*1000/SYSCLK
//SYSCLK 单位为 Hz,nms 单位为 ms
//对 72M 条件下,nms<=1864
void delay_ms(u16 nms)
{
    u32 temp;
    SysTick->LOAD=(u32)nms*fac_ms;                 //时间加载(SysTick->LOAD 为 24bit)
    SysTick->VAL =0x00;                            //清空计数器
```

```
    SysTick->CTRL|=SysTick_CTRL_ENABLE_Msk ;      //开始倒数
    do
    {
        temp=SysTick->CTRL;
    }while((temp&0x01)&&!(temp&(1<<16)));         //等待时间到达
    SysTick->CTRL&=~SysTick_CTRL_ENABLE_Msk;      //关闭计数器
    SysTick->VAL =0X00;                           //清空计数器
}
#endif
```

Delay.h

```
#ifndef __DELAY_H
#define __DELAY_H
#include "sys.h"

void delay_init(void);
void delay_ms(u16 nms);
void delay_us(u32 nus);

#endif
```