#include "stm32f10x.h"

#include "1602.h"

#include "DS18B20.h"

#include "math.h"

int wendu=0,yw=0,hy=0;

typedef unsigned char u8\_t;

typedef signed char s8\_t;

typedef unsigned short u16\_t;

typedef signed short s16\_t;

typedef unsigned long u32\_t;

typedef signed long s32\_t;

//位带操作,实现51类似的GPIO控制功能

//具体实现思想,参考<<CM3权威指南>>第五章(87页~92页).

//IO口操作宏定义

#define BITBAND(addr, bitnum) ((addr & 0xF0000000)+0x2000000+((addr &0xFFFFF)<<5)+(bitnum<<2))

#define MEM\_ADDR(addr) \*((volatile unsigned long \*)(addr))

#define BIT\_ADDR(addr, bitnum) MEM\_ADDR(BITBAND(addr, bitnum))

//IO口地址映射

#define GPIOA\_ODR\_Addr (GPIOA\_BASE+12) //0x4001080C

#define GPIOB\_ODR\_Addr (GPIOB\_BASE+12) //0x40010C0C

#define GPIOC\_ODR\_Addr (GPIOC\_BASE+12) //0x4001100C

#define GPIOD\_ODR\_Addr (GPIOD\_BASE+12) //0x4001140C

#define GPIOE\_ODR\_Addr (GPIOE\_BASE+12) //0x4001180C

#define GPIOF\_ODR\_Addr (GPIOF\_BASE+12) //0x40011A0C

#define GPIOG\_ODR\_Addr (GPIOG\_BASE+12) //0x40011E0C

#define GPIOA\_IDR\_Addr (GPIOA\_BASE+8) //0x40010808

#define GPIOB\_IDR\_Addr (GPIOB\_BASE+8) //0x40010C08

#define GPIOC\_IDR\_Addr (GPIOC\_BASE+8) //0x40011008

#define GPIOD\_IDR\_Addr (GPIOD\_BASE+8) //0x40011408

#define GPIOE\_IDR\_Addr (GPIOE\_BASE+8) //0x40011808

#define GPIOF\_IDR\_Addr (GPIOF\_BASE+8) //0x40011A08

#define GPIOG\_IDR\_Addr (GPIOG\_BASE+8) //0x40011E08

//IO口操作,只对单一的IO口!

//确保n的值小于16!

#define PAout(n) BIT\_ADDR(GPIOA\_ODR\_Addr,n) //输出

#define PAin(n) BIT\_ADDR(GPIOA\_IDR\_Addr,n) //输入

#define PBout(n) BIT\_ADDR(GPIOB\_ODR\_Addr,n) //输出

#define PBin(n) BIT\_ADDR(GPIOB\_IDR\_Addr,n) //输入

#define PCout(n) BIT\_ADDR(GPIOC\_ODR\_Addr,n) //输出

#define PCin(n) BIT\_ADDR(GPIOC\_IDR\_Addr,n) //输入

#define PDout(n) BIT\_ADDR(GPIOD\_ODR\_Addr,n) //输出

#define PDin(n) BIT\_ADDR(GPIOD\_IDR\_Addr,n) //输入

#define PEout(n) BIT\_ADDR(GPIOE\_ODR\_Addr,n) //输出

#define PEin(n) BIT\_ADDR(GPIOE\_IDR\_Addr,n) //输入

#define PFout(n) BIT\_ADDR(GPIOF\_ODR\_Addr,n) //输出

#define PFin(n) BIT\_ADDR(GPIOF\_IDR\_Addr,n) //输入

#define PGout(n) BIT\_ADDR(GPIOG\_ODR\_Addr,n) //输出

#define PGin(n) BIT\_ADDR(GPIOG\_IDR\_Addr,n) //输入

//以下为汇编函数

void WFI\_SET(void); //执行WFI指令

void INTX\_DISABLE(void);//关闭所有中断

void INTX\_ENABLE(void); //开启所有中断

void MSR\_MSP(u32 addr); //设置堆栈地址

#define EnableINT() INTX\_ENABLE()

#define DisableINT() INTX\_DISABLE()

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

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

}

//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

}

char youren=0,kaisuo=0,baojing=0;

void GPIO\_Configuration(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA | RCC\_APB2Periph\_GPIOB|RCC\_APB2Periph\_GPIOC,ENABLE);

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB|RCC\_APB2Periph\_AFIO, ENABLE); //ENABLEPBCLK

GPIO\_PinRemapConfig(GPIO\_Remap\_SWJ\_JTAGDisable , ENABLE); //Disable jtag

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC1,ENABLE);

//DS18B20

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_2 ;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_Out\_OD;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

//配置 PA0 ADC采集模式

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_0;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AIN;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

//按键 红外

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_1;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_2MHz;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IPU;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

}

void USART1\_init(void)

{

USART\_InitTypeDef USART\_InitStructure;

GPIO\_InitTypeDef GPIO\_InitStructure;

NVIC\_InitTypeDef NVIC\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOA |RCC\_APB2Periph\_AFIO , ENABLE);

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_USART1,ENABLE);

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_9;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_AF\_PP;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

GPIO\_InitStructure.GPIO\_Pin = GPIO\_Pin\_10;

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz;

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_IN\_FLOATING;

GPIO\_Init(GPIOA, &GPIO\_InitStructure);

USART\_InitStructure.USART\_BaudRate = 9600;

USART\_InitStructure.USART\_WordLength = USART\_WordLength\_8b;

USART\_InitStructure.USART\_StopBits = USART\_StopBits\_1;

USART\_InitStructure.USART\_Parity = USART\_Parity\_No;

USART\_InitStructure.USART\_HardwareFlowControl = USART\_HardwareFlowControl\_None;

USART\_InitStructure.USART\_Mode = USART\_Mode\_Tx | USART\_Mode\_Rx;

//USART1

USART\_Init(USART1, &USART\_InitStructure);

USART\_Cmd(USART1, ENABLE);

USART\_ClearITPendingBit(USART1, USABLE);

USART1->DR = (0X55 & (uint16\_t)0x01FF);

while((USART1->SR&0X40)==0);

NVIC\_PriorityGroupConfig(NVIC\_PriorityGroup\_0);

NVIC\_InitStructure.NVIC\_IRQChannel = USART1\_IRQn;

NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority = 0;

NVIC\_InitStructure.NVIC\_IRQChannelSubPriority = 1;

NVIC\_InitStructure.NVIC\_IRQChannelCmd = ENABLE;

NVIC\_Init(&NVIC\_InitStructure);

}

void SendData(char dat)

{

USART1->DR = (dat& (uint16\_t)0x01FF);

while((USART1->SR&0X40)==0);

}

/\*----------------------------

发送字符串

----------------------------\*/

void SendString(char \*s)

{

while (\*s) //检测字符串结束标志

{

USART1->DR = (\*s++& (uint16\_t)0x01FF);

while((USART1->SR&0X40)==0);

}

}

int RX1\_Num=0;//保存接收的数据

int RX1\_JS=0,RX1\_KG=0;//保存接收的数据

int RX1\_OK=0;//1表明接收到数据 0表明没有收到数据

unsigned char RX1\_BUFF[200];

unsigned char TX1\_BUFF[200];

void RX1\_CHULI(void)

{

long int \*p;

int shumu=0;

int wz=0;

char i=0;

if(RX1\_OK==1)

{

RX1\_OK = 0;

shumu = RX1\_Num;

RX1\_Num = 0;

//开始处理接受到的数据

//判断接受到的数据长度

if(shumu==4)

{

//从机1发送

if(RX1\_BUFF[0]=='2' && RX1\_BUFF[1]=='2'&& RX1\_BUFF[2]=='2'&& RX1\_BUFF[3]=='2')

{

delay\_ms(150);

//通过LORA上传数据

SendData(0X02);

SendData(wendu);

SendData(yw);

SendData(hy);

SendData(0X55);

}

}

for(i=0;i<shumu;i++)

{

RX1\_BUFF[i]=0;

}

}

}

void chuli1(void)

{

if(RX1\_KG==1)

{

RX1\_JS

RX1\_KG = 0;

RX1\_JS = 0;

RX1\_OK = 1;

}

}

}

//串口1接收中断

void USART1\_IRQHandler(void)

{

if(USART\_GetITStatus(USART1, USART\_IT\_RXNE) != RESET)

{

USART\_ClearITPendingBit(USART1, USART\_IT\_RXNE);

//保存收到的数据

RX1\_BUFF[RX1\_Num++]=USART\_ReceiveData(USART1); //读取接收到的数据

if(RX1\_Num>=200)

{

RX1\_Num=0;

}

RX1\_KG = 1;

RX1\_JS = 0;

}

if(USART\_GetITStatus(USART1, USART\_IT\_PE) != RESET)

{

USART\_ClearITPendingBit(USART1, USART\_IT\_PE);

}

if (USART\_GetFlagStatus(USART1, USART\_IT\_LBD) != RESET)

{

USART\_ClearITPendingBit(USART1, USART\_IT\_LBD);

}

if(USART\_GetFlagStatus(USART1, USART\_FLAG\_ORE) != RESET)

{

USART\_ReceiveData(USART1);

USART\_ClearFlag(USART1, USART\_FLAG\_ORE);

}

if(USART\_GetFlagStatus(USART1, USART\_FLAG\_NE) != RESET)

{

USART\_ClearFlag(USART1, USART\_FLAG\_NE);

}

if(USART\_GetFlagStatus(USART1, USART\_FLAG\_FE) != RESET)

{

USART\_ClearFlag(USART1, USART\_FLAG\_FE);

}

if(USART\_GetFlagStatus(USART1, USART\_FLAG\_PE) != RESET)

{

USART\_ClearFlag(USART1, USART\_FLAG\_PE);

}

if (USART\_GetITStatus(USART1, USART\_IT\_TC) != RESET)

{

USART\_ClearITPendingBit(USART1, USART\_IT\_TC);

}

}

void UsartSendData1(unsigned char \*send\_buff,unsigned long int length)

{

unsigned long int i = 0;

delay\_ms(1);

for(i = 0;i < length;i ++)

{

USART1->DR = (send\_buff[i] & (uint16\_t)0x01FF);

while((USART1->SR&0X40)==0);

}

delay\_ms(1);

}

/\* ADC配置 \*/

void ADC\_Set\_0(void)

{

ADC\_InitTypeDef ADC\_InitStructure;//ADC结构体变量//注意在一个语句快内变量的声明要放在可执行语句的前面，否则出错，因此要放在ADC1\_GPIO\_Config();前面

ADC\_InitStructure.ADC\_Mode = ADC\_Mode\_Independent;//ADC1和ADC2工作在独立模式

ADC\_InitStructure.ADC\_ScanConvMode = DISABLE; //使能扫描

ADC\_InitStructure.ADC\_ContinuousConvMode = ENABLE;//ADC转换工作在连续模式

ADC\_InitStructure.ADC\_ExternalTrigConv = ADC\_ExternalTrigConv\_None;//由软件控制转换,不使用外部触发

ADC\_InitStructure.ADC\_DataAlign = ADC\_DataAlign\_Right;//转换数据右对齐

ADC\_InitStructure.ADC\_NbrOfChannel = 1;//转换通道为1

ADC\_Init(ADC1, &ADC\_InitStructure); //初始化ADC

ADC\_RegularChannelConfig(ADC1, ADC\_Channel\_0, 1, ADC\_SampleTime\_41Cycles5);

ADC\_Cmd(ADC1, ENABLE);//使能ADC1

// ADC\_ITConfig(ADC1, ADC\_IT\_EOC, ENABLE);

ADC\_SoftwareStartConvCmd(ADC1, ENABLE);

}

void ADC\_Set\_1(void)

{

ADC\_Cmd(ADC1, DISABLE);//使能ADC1

ADC\_ITConfig(ADC1, ADC\_IT\_EOC, DISABLE);

ADC\_SoftwareStartConvCmd(ADC1, DISABLE);

ADC\_InitStructure.ADC\_Mode = ADC\_Mode\_Independent;//ADC1和ADC2工作在独立模式

ADC\_InitStructure.ADC\_ScanConvMode = DISABLE; //使能扫描

ADC\_InitStructure.ADC\_ContinuousConvMode = ENABLE;//ADC转换工作在连续模式

ADC\_InitStructure.ADC\_ExternalTrigConv = ADC\_ExternalTrigConv\_None;//由软件控制转换,不使用外部触发

ADC\_InitStructure.ADC\_DataAlign = ADC\_DataAlign\_Right;//转换数据右对齐

ADC\_InitStructure.ADC\_NbrOfChannel = 1;//转换通道为1

ADC\_Init(ADC1, &ADC\_InitStructure); //初始化ADC

ADC\_RegularChannelConfig(ADC1, ADC\_Channel\_1, 1, ADC\_SampleTime\_28Cycles5);

ADC\_Cmd(ADC1, ENABLE);//使能ADC1

ADC\_SoftwareStartConvCmd(ADC1, ENABLE);

}

int main(void)

{

char ct=0;

float temp=0;

SystemInit();

SystemCoreClockUpdate();

delay\_init();

//引脚配置

GPIO\_Configuration();

//初始化LCD1602

Init1602();

//温度初始化

ds18b20\_init();

//配置串口

USART1\_init();

delay\_ms(1000);

while (1)

{

delay\_ms(10);

//采集并且计算一次烟雾

ADC\_Set\_0();

temp = ADC\_GetConversionValue(ADC1);

temp = 100.0\*(temp)/(4095.00)-50;

yw=temp;

if(yw<0)

{

yw = 0;

}

yw = yw\*2;

if(yw>99)

{

yw = 99;

}

//读取一次温度

wendu=ds18b20\_read();

if(wendu>99)

{

wendu=0;

}

//采集一次火焰状态

if(P

else

{

hy=0;

}

//处理LORA数据

chuli1();

RX1\_CHULI();

}

}

/\*\*

\* @brief 定时器3中断服务函数

\* @param TIMx: where x can be 1 to 17 to select the TIM peripheral

\* @param NewState: new state of the TIMx peripheral.

\* This parameter can be: ENABLE or DISABLE.

\* @retval None

\*/

#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(uint8\_t\* file, uint32\_t line)

{

/\* User can add his own implementation to report the file name and line number,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* Infinite loop \*/

while (1)

{}

}

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* (C) COPYRIGHT 2009 STMicroelectronics \*\*\*\*\*END OF FILE\*\*\*\*/

#include "ds18b20.h"

#define EnableINT()

#define DisableINT()

#define DS\_PORT GPIOA

#define DS\_DQIO GPIO\_Pin\_2

#define DS\_RCC\_PORT RCC\_APB2Periph\_GPIOA

#define DS\_PRECISION 0x7f //精度配置寄存器 1f=9位; 3f=10位; 5f=11位; 7f=12位;

#define DS\_AlarmTH 0x64

#define DS\_AlarmTL 0x8a

#define DS\_CONVERT\_TICK 1000

#define ResetDQ() GPIO\_ResetBits(DS\_PORT,DS\_DQIO)

#define SetDQ() GPIO\_SetBits(DS\_PORT,DS\_DQIO)

#define GetDQ() GPIO\_ReadInputDataBit(DS\_PORT,DS\_DQIO)

void Delay\_us(u32 Nus)

{

SysTick->LOAD=Nus\*9; //时间加载

SysTick->CTRL|=0x01; //开始倒数

while(!(SysTick->CTRL&(1<<16))); //等待时间到达

SysTick->CTRL=0X00000000; //关闭计数器

SysTick->VAL=0X00000000; //清空计数器

}

unsigned char ResetDS18B20(void)

{

unsigned char resport;

SetDQ();

Delay\_us(50);

ResetDQ();s

//resport = GetDQ();

while(GetDQ());

Delay\_us(500); //500us

SetDQ();

return resport;

}

void DS18B20WriteByte(unsigned char Dat)

{

unsigned char i;

for(i=8;i>0;i--)

{

ResetDQ(); //在15u内送数到数据线上，DS18B20在15-60u读数

Delay\_us(5); //5us

if(Dat & 0x01)

SetDQ();

else

ResetDQ();

Delay\_us(65); //65us

SetDQ();

Delay\_us(2); //连续两位间应大于1us

Dat >>= 1;

}

}

unsigned char DS18B20ReadByte(void)

{

unsigned char i,Dat;

SetDQ();

Delay\_us(5);

for(i=8;i>0;i--)

{

Dat >>= 1;

ResetDQ(); //从读时序开始到采样信号线必须在15u内，且采样尽量安排在15u的最后

Delay\_us(5); //5us

SetDQ();

Delay\_us(5); //5us

if(GetDQ())

Dat|=0x80;

else

Dat&=0x7f;

Delay\_us(65); //65us

SetDQ();

}

return Dat;

}

void ReadRom(unsigned char \*Read\_Addr)

{

unsigned char i;

DS18B20WriteByte(ReadROM);

for(i=8;i>0;i--)

{

\*Read\_Addr=DS18B20ReadByte();

Read\_Addr++;

}

}

void DS18B20Init(unsigned char Precision,unsigned char AlarmTH,unsigned char AlarmTL)

{

DisableINT();

ResetDS18B20();

DS18B20WriteByte(SkipROM);

DS18B20WriteByte(AlarmTH);

DS18B20WriteByte(Precision);

ResetDS18B20();

DS18B20WriteByte(SkipROM);

DS18B20WriteByte(CopyScratchpad);

EnableINT();

while(!GetDQ()); //等待复制完成 ///////////

}

void DS18B20StartConvert(void)

{

DisableINT();

ResetDS18B20();

DS18B20WriteByte(SkipROM);

DS18B20WriteByte(StartConvert);

EnableINT();

}

void DS18B20\_Configuration(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_APB2PeriphClockCmd(DS\_RCC\_PORT, ENABLE);

时钟速度

GPIO\_Init(DS\_PORT, &GPIO\_InitStructure);

}

void ds18b20\_init(void)

{

DS18B20\_Configuration();

DS18B20Init(DS\_PRECISION, DS\_AlarmTH, DS\_AlarmTL);

DS18B20StartConvert();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 名 ds18b20\_read()

\* 功 能：读取18x20的温度测量结果。注意要在DS18B20StartConvert()命令至少之后700ms以

后调用，才能读取准确的温度值。

\* 入口参数：无

\* 出口参数：温度数值

\* 范 例:返回12表示12

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

signed int ds18b20\_read(void)

{

unsigned char tempH,tempL;

float temp;

signed int result;

ResetDS18B20();

DS18B20WriteByte(SkipROM);

d);

tempL=DS18B20ReadByte();

tempH=DS18B20ReadByte();

ResetDS18B20();

DS18B20StartConvert();

//负

if(tempH & 0x80)

{

result = (tempH)\*256+tempL;

result = 0xffff-result;

result = result +

}

//正

else

{

temp = (tempH)\*256+tempL;

temp = temp\*625/100;

}

result = temp/100;

return (result);//计算温度，保留2位小数

}