Mian.c

#include "led.h"

#include "delay.h"

#include "sys.h"

#include "usart.h"

#include "usart2.h"

#include "esp8266.h"

#include "timer.h"

#include "string.h"

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

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

extern \_Bool Timer\_Flag ; //时间到 标准位

extern \_Bool Control\_Flag ; //时间控制 标准位

extern unsigned short Time; //定时时间

int main(void)

{

unsigned char status=1; //服务是否被连接状态 0：已连接 1：未连接

\_Bool netWork = 0; //服务器端口监听状态 1：正常 0：异常

unsigned char send[20];

unsigned char \*dataPtr = NULL; //ESP8266设备收到服务器下发的数据 缓存地址

delay\_init(); //延时函数初始化

NVIC\_PriorityGroupConfig(NVIC\_PriorityGroup\_2);//设置中断优先级分组为组2：2位抢占优先级，2位响应优先级

uart\_init(115200); //串口一初始化为115200

TIM2\_Int\_Init(4999,7199); //10Khz的计数频率，计数到5000为500ms

LED\_Init(); //初始化与控制设备连接的硬件接口

delay\_ms(1000);

NET\_IO\_Init(); //串口二初始化

while(1)

{

if(netWork==1) //设备已经就绪

{

LED0 = 0;

if(Timer\_Flag == 1) //5S到了

{

if(status==0) //已连接

{

sprintf((char \*)send,"HeartOK %hd\r\n",Time); //打包心跳数据 TCP连接中通常需要隔断时间发一条数据用于维持心跳 ，不然会被服务器踢出

status = Esp8266\_Net\_SendData((unsigned char \*)send,strlen((char \*)send)); //发送数据到APP

}

else

{

status = Esp8266\_WaitConnect(); //获取连接状态

}

Timer\_Flag = 0;

}

dataPtr = Esp8266\_GetIPD(10); //获取ESP8266下发的数据 dataPtr已经是处理之后的数据了

if(dataPtr!=NULL)

{

if(strstr((char \*)dataPtr,"Q1") != NULL)

{ JD0 = 0; ;}

else if(strstr((char \*)dataPtr,"G1") != NULL)

{ JD0 = 1;}

else if(strstr((char \*)dataPtr,"Q2") != NULL)

{ JD1 = 0;}

else if(strstr((char \*)dataPtr,"G2") != NULL)

{ JD1 = 1;}

else if(strstr((char \*)dataPtr,"Q3") != NULL)

{ JD2 = 0; ;}

else if(strstr((char \*)dataPtr,"G3") != NULL)

{ JD2 = 1;}

else if(strstr((char \*)dataPtr,"Q4") != NULL)

{ JD3 = 0;}

else if(strstr((char \*)dataPtr,"G4") != NULL)

{ JD3 = 1;}

else if(strstr((char \*)dataPtr,"OFF") != NULL)

{ JD0 = 1;JD1 = 1;JD2 = 1;JD3 = 1;}

else if(strstr((char \*)dataPtr,"ON") != NULL)

{ JD0 = 0;JD1 = 0;JD2 = 0;JD3 = 0;}

else if(strstr((char \*)dataPtr,"T") != NULL) //判断Txx:xx

{

char \*TimePtr = NULL;

TimePtr=strchr((char \*)dataPtr,'T'); //查找字符串'T'中首次出现字符ptrIPD的位置

TimePtr++; //地址加一

if(TimePtr[2]==':') //如果格式正确

{

Time=((TimePtr[0]-0x30)\*10+TimePtr[1]-0x30)\*60+(TimePtr[3]-0x30)\*10+TimePtr[4]-0x30; //将字符转换为整数

if(Time>0)

Control\_Flag=1; //倒计时标志位打开？

}

}

}

}

else

{

LED0 = 1;

if(!Esp8266\_Init()) //ESP8266初始化配置

{

netWork = 1; //初始化成功则 幅值为1 表示ESP8266初始化成功 才进入下一步TCP通讯

}

}

}

}

Led.c

#include "led.h"

// IO初始化

void LED\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB|RCC\_APB2Periph\_GPIOC, ENABLE); //使能PA端口时钟

GPIO\_InitStructure.GPIO\_Pin =GPIO\_Pin\_13; //PA4 -- 蜂鸣器

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_Out\_PP; //推挽输出

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz; //IO口速度为50MHz

GPIO\_Init(GPIOC, &GPIO\_InitStructure); //根据设定参数初始化GPIOPA4

GPIO\_SetBits(GPIOC,GPIO\_Pin\_13); //PA4 输出高

GPIO\_InitStructure.GPIO\_Pin =GPIO\_Pin\_5|GPIO\_Pin\_6|GPIO\_Pin\_7|GPIO\_Pin\_8; //P5-8

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_Out\_PP; //推挽输出

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz; //IO口速度为50MHz

GPIO\_Init(GPIOB, &GPIO\_InitStructure); //根据设定参数初始化GPIOPA4

GPIO\_SetBits(GPIOB,GPIO\_Pin\_5|GPIO\_Pin\_6|GPIO\_Pin\_7|GPIO\_Pin\_8); //PA4 输出高

}

Usart2.c

#include "led.h"

// IO初始化

void LED\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStructure;

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOB|RCC\_APB2Periph\_GPIOC, ENABLE); //使能PA端口时钟

GPIO\_InitStructure.GPIO\_Pin =GPIO\_Pin\_13; //PA4 -- 蜂鸣器

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_Out\_PP; //推挽输出

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz; //IO口速度为50MHz

GPIO\_Init(GPIOC, &GPIO\_InitStructure); //根据设定参数初始化GPIOPA4

GPIO\_SetBits(GPIOC,GPIO\_Pin\_13); //PA4 输出高

GPIO\_InitStructure.GPIO\_Pin =GPIO\_Pin\_5|GPIO\_Pin\_6|GPIO\_Pin\_7|GPIO\_Pin\_8; //P5-8

GPIO\_InitStructure.GPIO\_Mode = GPIO\_Mode\_Out\_PP; //推挽输出

GPIO\_InitStructure.GPIO\_Speed = GPIO\_Speed\_50MHz; //IO口速度为50MHz

GPIO\_Init(GPIOB, &GPIO\_InitStructure); //根据设定参数初始化GPIOPA4

GPIO\_SetBits(GPIOB,GPIO\_Pin\_5|GPIO\_Pin\_6|GPIO\_Pin\_7|GPIO\_Pin\_8); //PA4 输出高

}

Esp8266.c

#include "usart2.h"

#include "esp8266.h"

#include "delay.h"

#include "string.h"

#include "stdio.h"

NET\_DEVICE\_INFO netDeviceInfo = {"", "", 0, 0, 1, 0, 0};

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

\* 函数: \_Bool Esp8266\_Init(void)

\* 描述: 网络设备初始化

\* 参数: nono.

\* 返回: 0成功 1失败

\* 备注:

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

\_Bool Esp8266\_Init(void)

{

switch(netDeviceInfo.initStep)

{

case 0:

NET\_IO\_ClearRecive();

NET\_IO\_Send((unsigned char \*)"AT+GMR\r\n", strlen("AT+GMR\r\n"));

delay\_ms(150);

printf( "\r\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\n%s\r\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\r\n", netIOInfo.buf); //打印版本信息

NET\_IO\_ClearRecive(); //清除串口

netDeviceInfo.initStep=1;

break;

case 1:

printf( "AP Tips: AT\r\n");

if(!NET\_DEVICE\_SendCmd("AT\r\n","OK",50)) //测试启动

netDeviceInfo.initStep=2;

break;

case 2:

printf( "AP Tips: AT+RST\r\n");

if(!NET\_DEVICE\_SendCmd("AT+RST\r\n","OK",50)) //重启模块

netDeviceInfo.initStep=3;

delay\_ms(500);

break;

case 3:

printf( "AP Tips: AT+CWMODE=2\r\n");

if(!NET\_DEVICE\_SendCmd((char \*)"AT+CWMODE=2\r\n","OK",50)) //设置为AP模式

netDeviceInfo.initStep=4;

break;

case 4:

printf( "AP Tips: AT+CIPMUX=1\r\n");

if(!NET\_DEVICE\_SendCmd((char \*)"AT+CIPMUX=1\r\n","OK",50)) //启动多连接

netDeviceInfo.initStep=5;

break;

case 5:

printf( "AP Tips: AT+CIPSERVER=1,888\r\n");

if(!NET\_DEVICE\_SendCmd("AT+CIPSERVER=1,888\r\n","OK",50)) //开启服务器 地址为192.168.4.1 端口888

netDeviceInfo.initStep=6;

break;

}

if(netDeviceInfo.initStep == 6)

return 0;

else

return 1;

}

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

\* 函数: \_Bool Esp8266\_Net\_SendData(unsigned char \*string,unsigned char Len)

\* 描述: 使网络设备发送数据到平台

\* 参数: data：需要发送的数据

\* len：数据长度

\* 返回: 0成功 1失败

\* 备注:

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

\_Bool Esp8266\_Net\_SendData(unsigned char \*string,unsigned char Len)

{

unsigned char cmd[40];

memset(cmd,0,sizeof(cmd));

sprintf((char\*)cmd,"AT+CIPSEND=0,%d\r\n",Len);

if(NET\_DEVICE\_SendCmd((char\*)cmd,">",50)) return 1;

if(NET\_DEVICE\_SendCmd((char\*)string,"SEND OK",50)) return 1;

return 0;

}

/\*

unsigned char Esp8266\_WaitConnect(void)

{

unsigned char status=0;

unsigned char timeOut=100;

NET\_IO\_Send((unsigned char \*)"AT+CIPSTATUS\r\n", strlen("AT+CIPSTATUS\r\n"));

while(--timeOut)

{

if(NET\_IO\_WaitRecive()==0)

{

if(strstr((const char \*)netIOInfo.buf, "STATUS:2")) //获得IP

{

status=2;

printf("ESP8266 Got IP\r\n");

}

else if(strstr((const char \*)netIOInfo.buf, "STATUS:3")) //建立连接

{

status = 0;

printf("ESP8266 Connect OK\r\n");

}

else if(strstr((const char \*)netIOInfo.buf, "STATUS:4")) //失去连接

{

status = 1;

printf("ESP8266 Lost Connect\r\n");

}

else if(strstr((const char \*)netIOInfo.buf, "STATUS:5")) //物理掉线

{

status = 3;

printf("ESP8266 Lost\r\n"); //设备丢失

}

break;

}

delay\_ms(2);

}

if(timeOut == 0)

{

status = 3;

printf("ESP8266 TimeOut\r\n");

}

return status;

}

\*/

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

\* 函数: unsigned char Esp8266\_WaitConnect(void)

\* 描述: 检查网络设备连接状态 等待 模块被连接 获取模块状态

\* 参数: none.

\* 返回: 返回状态

\* 备注: 如果有cilent连接 会返回+CIPSTATUS: IP和端口

模块返回如下：

AT+CIPSTATUS

STATUS:5

+CIPSTATUS:0,"TCP","192.168.4.2",33743,888,1

OK

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

unsigned char Esp8266\_WaitConnect(void)

{

unsigned char status=0;

unsigned char timeOut=100;

NET\_IO\_Send((unsigned char \*)"AT+CIPSTATUS\r\n", strlen("AT+CIPSTATUS\r\n"));

while(--timeOut)

{

if(NET\_IO\_WaitRecive()==0)

{

if(strstr((const char \*)netIOInfo.buf, "+CIPSTATUS:")) //如果有cilent连接 会返回+CIPSTATUS: IP和端口

{

status = 0; //已连接

}

else

status = 3; //未连接

break;

}

delay\_ms(2);

}

if(timeOut == 0) //无响应

{

status = 2;

}

return status;

}

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

\* 函数: unsigned char \*Esp8266\_GetIPD(unsigned short TimeOut)

\* 描述: 处理ESP8266发来的数据

\* 参数: none.

\* 返回: 返回接受到的数据

\* 备注: 接受数据 "+IPD,ID,x:yyyyyy" ID用户ID x代表数据长度，yyy是数据内容

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

unsigned char \*Esp8266\_GetIPD(unsigned short TimeOut)

{

char \*ptrIPD = NULL;

do

{

if(NET\_IO\_WaitRecive()==0)//返回OK

{

ptrIPD = strstr((char \*)netIOInfo.buf,"IPD");//搜索“IPD”头

if(ptrIPD == NULL) //如果没找到，可能是IPD头的延迟，还是需要等待一会，但不会超过设定的时间

{

//UsartPrintf(USART\_DEBUG, "\"IPD\" not found\r\n");

}

else

{

ptrIPD=strchr(ptrIPD,':');//查找字符串':'中首次出现字符ptrIPD的位置。

if(ptrIPD != NULL)

{

ptrIPD++;

return (unsigned char \*)(ptrIPD);

}

else

return NULL;

}

}

delay\_ms(5);

}while(TimeOut--);

return NULL;

}

Timer.c

#include "timer.h"

#include "led.h"

#include "string.h"

#include "delay.h"

\_Bool Timer\_Flag = 1; //时间到 标准位

\_Bool Control\_Flag = 0 ; //时间控制 标准位

unsigned short Time = 0; //定时时间

void TIM2\_Int\_Init(u16 arr,u16 psc)

{

TIM\_TimeBaseInitTypeDef TIM\_TimeBaseStructure;

NVIC\_InitTypeDef NVIC\_InitStructure;

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_TIM2, ENABLE); //时钟使能

//定时器TIM3初始化

TIM\_TimeBaseStructure.TIM\_Period = arr; //设置在下一个更新事件装入活动的自动重装载寄存器周期的值

TIM\_TimeBaseStructure.TIM\_Prescaler =psc; //设置用来作为TIMx时钟频率除数的预分频值

TIM\_TimeBaseStructure.TIM\_ClockDivision = TIM\_CKD\_DIV1; //设置时钟分割:TDTS = Tck\_tim

TIM\_TimeBaseStructure.TIM\_CounterMode = TIM\_CounterMode\_Up; //TIM向上计数模式

TIM\_TimeBaseInit(TIM2, &TIM\_TimeBaseStructure); //根据指定的参数初始化TIMx的时间基数单位

TIM\_ITConfig(TIM2,TIM\_IT\_Update,ENABLE ); //使能指定的TIM3中断,允许更新中断

//中断优先级NVIC设置

NVIC\_InitStructure.NVIC\_IRQChannel = TIM2\_IRQn; //TIM3中断

NVIC\_InitStructure.NVIC\_IRQChannelPreemptionPriority = 1; //先占优先级0级

NVIC\_InitStructure.NVIC\_IRQChannelSubPriority = 3; //从优先级3级

NVIC\_InitStructure.NVIC\_IRQChannelCmd = ENABLE; //IRQ通道被使能

NVIC\_Init(&NVIC\_InitStructure); //初始化NVIC寄存器

TIM\_Cmd(TIM2, ENABLE); //使能TIMx

}

//定时器2中断服务程序

void TIM2\_IRQHandler(void) //TIM2中断 500ms中断

{

static unsigned char Tt = 0;

static unsigned int Cnt = 0;

if (TIM\_GetITStatus(TIM2, TIM\_IT\_Update) != RESET) //检查TIM3更新中断发生与否

{

TIM\_ClearITPendingBit(TIM2, TIM\_IT\_Update ); //清除TIMx更新中断标志

Cnt++;

if(Cnt>=10) //500ms\*10 = 5S

{

Cnt = 0;

Timer\_Flag = 1; //赋予标志位

}

if(Control\_Flag==1) //倒计时标志位打开？

{

Tt++;

if(Tt>=2) //1S到

{

Tt = 0;

if(Time>0) //如果时间数据大于0

Time--;

if(Time==0) //倒计时完成了

{

Control\_Flag = 0; //清除标志位

JD0 = 1; JD1 = 1;JD2 = 1; JD3 = 1; //继电器全关

}

}

}

}

}