2015年全国大学生电子设计竞赛



风板控制装置（I题）

【高职高专组】

2015年8月15日

摘 要

本系统以MSP430单片机为控制核心，通过PID算法，实现了对风板的控制。系统主要由电源模块、角度测量模块、电机驱动模块、显示模块、键盘模块和声光模块等构成。通过PID反馈输出PWM来改变直流风机风力大小，使风板转角根据需求变化。加速度传感器MPU6050检测风板位置的变化，并将风板角度在LCD液晶器上显示，同时单片机对采集的数据进行分析，实时调整PWM输出，通过驱动芯片L298N控制风机风速，使风板达到稳定的状态，并带有相应的声光提醒功能,使系统人性化，经过反复测试，达到了设计要求。

关键词：MSP430;PID;PWM;直流风机;MPU6050

目 录

1 系统方案

本系统主要由主控模块、角度检测模块、电机驱动模块、显示模块、电源模块组成，下面分别论证这几个模块的选择。

1.1主控模块的论证与选择

方案一：采用传统8位的51单片机作为该系统的控制核心。经典51单片机具有价格低廉，使用简单等特点，但其存在外设I/O端口较少，运算速度低，功能单一，不稳定等缺点。

方案二：采用 TI 公司所生产的 MSP430F149 单片机为主控制芯片，运算速度快，超低功耗，有非常丰富的片内资源，性价比高。

综合比较以上两个方案，选择TI公司生产的430芯片，在低功耗方面有显著的优势，处理数据快，且其片内资源丰富，满足系统设计需求。

1.2角度检测模块方案论证

方案一：采用模拟三轴加速度计MMA7260, MMA7260QT是检测物件运动和方向的传感器，它根据物件运动和方向改变输出信号的电压值。通过A/D转换器读取输出信号，检测其运动和方向。

方案二：采用mpu6050传感器可准确追踪快速与慢速动作，并且可调整感测范围，可快速、直接将检测信号给控制器 。

控制帆板角度是个快速处理的过程，方案一还需采集电路对AD进行采集转化为数字量，综合考虑选择方案二。

1.3电机驱动模块

本设计的主要目的在于控制风机的转速，因此电机驱动模块是必不可少，其方案有以下两种。

方案一：采用大功率晶体管组合电路构成驱动电路，这种方法结构简单，成本低、易实现，但由于在驱动电路中采用了大量的晶体管相互连接，使得电路复杂、抗干扰能力差、可靠性下降，我们知道在实际的生产实践过程中可靠性是一个非常重要的方面。因此此中方案不宜采用。

方案二：采用专用的电机驱动芯片，例如L298N、L297N等电机驱动芯片，由于它内部已经考虑到了电路的抗干扰能力，安全、可靠行，所以我们在应用时只需考虑到芯片的硬件连接、驱动能力等问题就可以了，所以此种方案的电路设计简单、抗干扰能力强、可靠性好。设计者不需要对硬件电路设计考虑很多，可将重点放在算法实现和软件设计中，大大的提高了工作效率。

基于上述理论分析和实际情况，电机驱动模块选用方案二。

1.4 显示模块方案

方案一：选用常见的数码管显示，成本低，只能显示简单的字符和数字。显示位数较多时，轮番扫描占用CPU时间。

方案二：选用12864显示屏做显示。12864的显示为128x64，显示面积大，数字和汉字显示容易实现，程序要求不是很高，更加方便。

方案三：用彩屏做显示。彩屏显示效果好，但成本高，功耗大，编程设计相对繁琐。

由于系统显示信息量较多，对比所述方案，选择12864作为系统显示器。

1.5电源模块方案

电源是任何系统能否运行的能量来源，本系统中电源模块为主控制器、电机驱动、角度检测模块等提供电源。

方案一：通过电阻分压的形式将整流后的电压分别降为控制芯片和电机运行所需的电压，此种方案原理和硬件电路连接都比较简单，但对能量的损耗大，在实际应用系统同一般不宜采用。

方案二：通过固定芯片对整流后的电压进行降压、稳压处理（如7812、7805等），此种方案可靠性、安全性高，对能源的利用率高，并且电路简单容易实现。

根据系统的具体要求，采用方案二作为系统的供电模块。

2.系统理论分析与计算

2.1角度确定

风板运动过程中需要实时检测角度的变化，通过计算加速度传感器传回的数据，可以测得风板的角度，加速度与角度存在如下关系：



2.2风速控制

风速的快慢直接决定了系统风板角度的大小。通过pid调节,单片机输出PWM波形，可对风板进行快速、准确的调整。

通过不断调整P(比例)、I(积分)、D（微分）值，系统的稳定性得到明显的提高，响应时间也加快了。由各个参数的控制规律可知，比例P使反应变快，微分D使反应提前，积分I使反应滞后。在一定范围内，P，D值越大，调节的效果越好。

3.电路与程序设计

3.1硬件电路设计

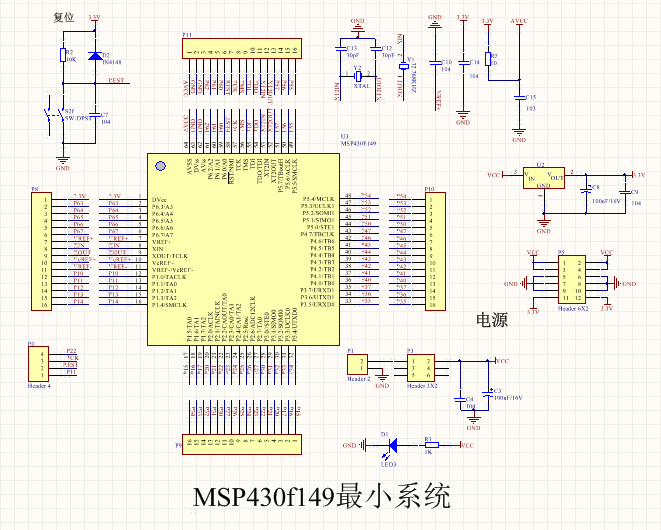
系统总体框图如图3.1所示。



图 3.1系统总体框图

3.1.1单片机最小系统设计

MSP430F149单片机，其最小系统包括电源电路、复位电路、时钟电路，具体电路设计如图3.2所示。

图3.2 最小系统原理图

3.1.2 角度测量设计

本系统中要实时监控风板的角度，系统采用mpu6050传感器,通过计算可迅速得出测量的角度，从而反馈给单片机进行相应操作。

3.1.3 显示模块设计

本系统采用LCD12864作为显示，模块电路图如图 3.3所示。

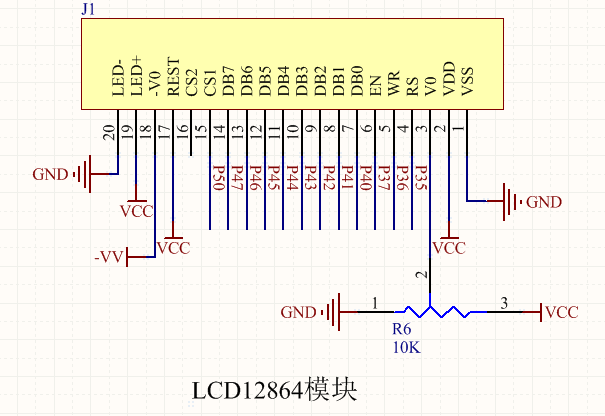


图 3.3 LCD12864电路图

3.1.4直流风机驱动

电机驱动芯片L298N内部包含4通道逻辑驱动电路。是一种二相和四相电机的专用驱动器。L298可驱动2个电机，OUT1、OUT2和OUT3、OUT4之间分别接2个电动机。5、7、10、12脚接输入控制电平，控制电机的正反转，ENA，ENB接控制使能端，控制电机的停转。利用单片机产生PWM信号接到ENA，ENB端子，对电机的转速进行调节。电机驱动电路如图3.4所示。

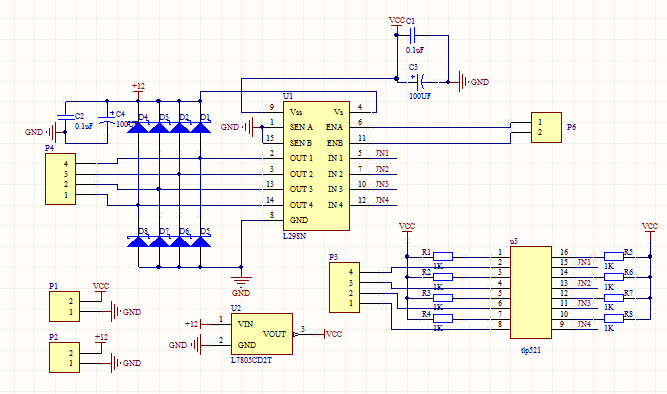


图3.4电机驱动

3.1.5供电系统电路

在电子电路及设备中，一般都需要稳定的直流电源供电。小功率的稳压电源的组成如图3.5所示，它由电源变压器、整流电路、滤波电路和稳压电路四部分组成。直流稳压电源电路如图3.6所示。

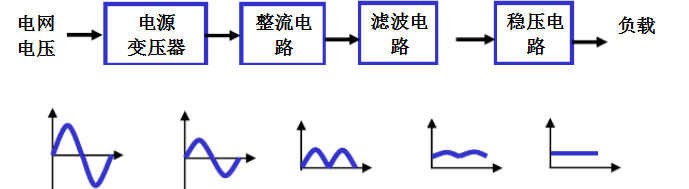


图 3.5直流稳压电路工作原理

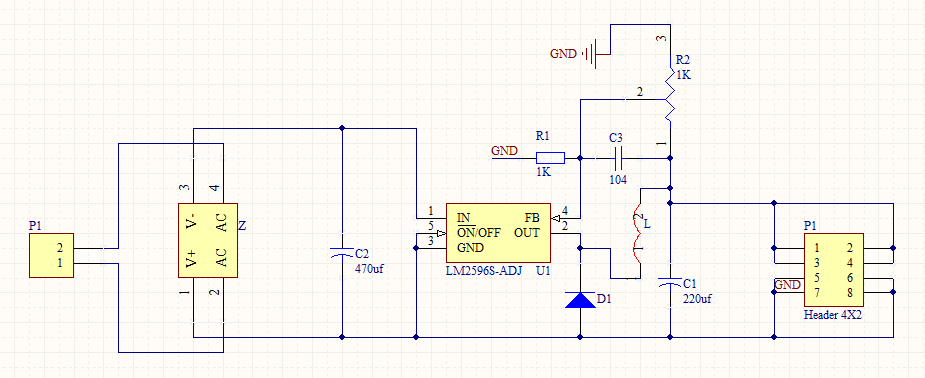


图 3.6 LM2596可调电源模块

3.2程序设计

3.2.1程序功能描述

根据设计要求，软件部分主要实现风机转速控制以及声光报警与液晶显示。

（1）风机转速控制部分：在键盘按下设定风板的角度后，风板15s内处于指定位置并稳定5秒以上，上下波动不超过5度，根据传感器测出的距离通过PID算法调整风机转速来调整风板的位置。

（2）键盘设置部分：在键盘按下后，风机作出相应转速，风板达到指定位置。

（3）液晶显示部分：液晶器显示风板位置及维持时间。

3.2.2程序设计思路

系统程序主要由角度检测部分，PID调节部分和显示部分组成。设定需求角度，通过获取测量角度值来反馈给单片机，单片机做出相应的PWM调节,从而达到需求的角度。同时液晶将对实时采集的角度和按键值信息进行显示。

3.2.3程序流程图

程序总体流程图如图3.7所示，PWM控制流程图如图3.8所示，12864液晶显示电路流程图如图3.9所示。

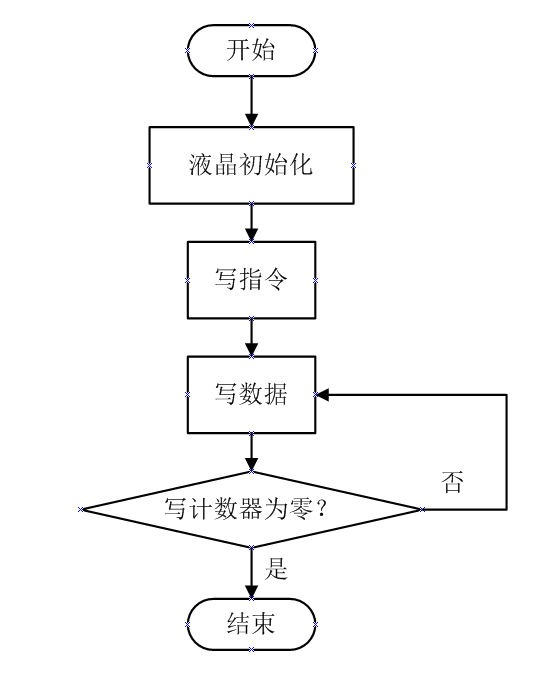
  

图 3.7 程序主流程图 图3.8 PWM控制电路 图3.9 12864液晶程序流程图

4系统测试

4.1测试仪器

高精度的数字毫伏表、模拟示波器、数字示波器、数字万用表、指针式万用表、量角器、秒表。

4.2测试方案

1、硬件测试

对各个模块进行测试，测试通过后使用。

2、软件仿真测试

对程序的错误和不能正确实现的部分进行调节和改正。

3、硬件软件联调

对整体功能的实现进行进一步调节。

4.3 测试结果及分析

4.3.1测试结果

a. 风板实际角度与角度传感器角度比较

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 实测角 | 40 | 60 | 80 | 100 | 120 | 130 | 150 |
| 角度传感器值 | 40 | 59 | 81 | 101 | 120 | 131 | 150 |

b.风板角度与pwm关系如表4.2所示。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 不带砝码 | 测试次数 | 设定角度 | 测试结果 | 测试误差 | 完成时间 |
| 1 | 45 |  |  |  |
| 2 | 65 |  |  |  |
| 3 | 85 |  |  |  |
| 4 | 90 |  |  |  |
| 5 | 115 |  |  |  |
| 6 | 135 |  |  |  |
| 带砝码 | 7 | 45 |  |  |  |
| 8 | 65 |  |  |  |
| 9 | 85 |  |  |  |
| 10 | 90 |  |  |  |
| 11 | 115 |  |  |  |
| 12 | 135 |  |  |  |

4.3.2测试分析与结论

根据上述测试数据，随着PWM的增加，风机的风速逐渐加大，风板设定的角度能维持3~5s，由此可以得出以下结论：

1、风机的转速可通过PWM调控且成正比关系。

2、风板实际角度与传感器检测的角度误差1度，当稳定时，角度误差小于5度波动，符合系统要求。

综上所述，本系统达到设计要求。

5 设计总结

不说我们在电赛中学到的新知识是多么有价值，也不用说它拓宽了多少我们的眼界，只是说它让我们的能力得到提高就已足以成为我们努力付出的回报。

在这四天三夜的奋战中，遇到过很多困难，搭载硬件时，由于对量角器安装角度有误差，导致软件校准时存在较大误差，经过仔细排查，不断改变方案，最终解决了问题。软件调试时，PID参量设置成了最头疼的问题，过大调整系统波动较大，过小调整风板很难达到预设角度。功夫不负有心人，两天的幸苦没有白费，当风机稳定的转动到达预设角度时，喜悦之情难以言表。理论与现实总是用通过实践联系起来的，我们在这次比赛中不再是局限于课本或是参考资料中的理论知识，而是把所有的实验都按部就班的做过，并通过实验结果对所理解的知识进行了加强巩固，更是对原来的理解偏差进行了改正，使我们对平时所学的课程更加透彻。当然，我们自己动手焊接了不少硬件电路，用到了大量的基础知识，还把实习中学到的焊接技术，对以前的知识进行了一次整合。再来，它增强了我们的沟通能力。合作精神是一个团队成立成长的根本所在，我们组的队员从第一个合作项目起就十分的团结，大家各有分工，共同讨论，为我们日后的工作能力打下了基础。

这次大赛给予了我们很多，通过这次比赛大家一起交流学习，互帮互助，增强了合作意识。老师们的帮助让我们感觉老师不再仅仅是一个指导者，更像一个家长带领我们成长，让我们跟老师之间的关系更加的紧密。最后，我们衷心感谢我们的指导老师，也祝愿此次大赛圆满成功！

附录1:电路原理图

附录2：源程序

#include <msp430x14x.h>

#include "PID.h"

#define P\_DATA 0.5

#define I\_DATA 0

#define D\_DATA 0

typedef struct PID

{

int SetPoint; //设定目标 Desired Value

long SumError; //误差累计

double Proportion; //比例常数 Proportional Const

double Integral; //积分常数 Integral Const

double Derivative; //微分常数 Derivative Const

int LastError; //Error[-1]

int PrevError; //Error[-2]

}PID;

static PID sPID;

static PID \*sptr = &sPID;

void IncPIDInit(int target)

{

sptr->SumError = 0; //误差的累加

sptr->LastError = 0; //Error[-1]

sptr->PrevError = 0; //Error[-2]

sptr->Proportion =P\_DATA; //比例常数 Proportional Const

sptr->Integral =I\_DATA; //积分常数 Integral Const

sptr->Derivative =D\_DATA; //微分常数 Derivative Const

sptr->SetPoint =target; //目标

}

//增量式 PID 控制设计

void PID\_dat(unsigned int target)

{

sptr->SetPoint =target; //目标

}

int IncPIDCalc(int NextPoint)

{

register int iError, iIncpid;

//当前误差

iError = sptr->SetPoint - NextPoint;

//增量计算

iIncpid = sptr->Proportion \* iError //E[k] 项

- sptr->Integral \* sptr->LastError //E[k－1] 项

+ sptr->Derivative \* sptr->PrevError; //E[k－2] 项

//存储误差，用于下次计算

sptr->PrevError = sptr->LastError;

sptr->LastError = iError;

//返回增量值

return(iIncpid);

}

//位置式 PID 控制设计

unsigned int LocPIDCalc(int NextPoint)

{

register int iError,dError;

iError = sptr->SetPoint - NextPoint; //偏差

sptr->SumError += iError; //积分

dError = iError - sptr->LastError; //微分

sptr->LastError = iError;

return(sptr->Proportion \* iError //比例项

+ sptr->Integral \* sptr->SumError //积分项

+ sptr->Derivative \* dError); //微分项

}

#include <msp430x14x.h>

#include"IIC.h"

#include"lcd12864.h"

#include <math.h>

#include <stdio.h>

int cunt=0;

uchar dis[5]; //显示数字(-511至512)的字符数组

int dis\_data;

float Tvae()

{

int data\_xyz[3];

float Roll,Pitch,zz,Q,T,K;

data\_xyz[0]=GetData(ACCEL\_XOUT\_H); //合成数据

data\_xyz[1]=GetData(ACCEL\_YOUT\_H);

data\_xyz[2]=GetData(ACCEL\_ZOUT\_H);

Q=(float)data\_xyz[0]\*3.9; //分别是加速度X,Y,Z的原始数据，10位的

T=(float)data\_xyz[1]\*3.9;

K=(float)data\_xyz[2]\*3.9;

Roll=(float)((atan2(K,T)\*180/3.14159265)-90); //X轴角度值

return ( -Roll);

}

void lcd\_printf(uchar \*s,int temp\_data)

{

if(temp\_data<0)

{

temp\_data=-temp\_data;

\*s='-';

}

else \*s=' ';

\*++s =temp\_data/100+'0';

temp\_data=temp\_data%100; //取余运算

\*++s =temp\_data/10+'0';

temp\_data=temp\_data%10; //取余运算

\*++s =temp\_data+'0';

\*++s='\0';

}

void InitMPU6050()

{

//Single\_WriteI2C(PWR\_MGMT\_1, 0x80); //复位

Single\_WriteI2C(PWR\_MGMT\_1, 0x00); //解除休眠状态

Single\_WriteI2C(SMPLRT\_DIV, 0x07); //陀螺仪采样率，典型值：0x07(125Hz)

Single\_WriteI2C(CONFIG, 0x06); //低通滤波频率，典型值：0x06(5Hz)

Single\_WriteI2C(GYRO\_CONFIG, 0x18); //陀螺仪自检及测量范围，典型值：0x18(不自检，2000deg/s)

Single\_WriteI2C(ACCEL\_CONFIG, 0x01);//加速计自检,测量范围及高通滤波频率,典型值：0x01(不自检，2G，5Hz)

}

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

//合成数据

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

int GetData(uchar REG\_Address)

{

char H,L;

H=Single\_ReadI2C(REG\_Address);

L=Single\_ReadI2C(REG\_Address+1);

return (H<<8)+L; //合成数据

}

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

//在12864上显示10位数据

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

int Display10BitData(int value)

{

value/=64; //转换为10位数据

lcd\_printf(dis, value); //转换数据显示

//csj(y,x,dis);

return value;

}

void I2C\_Start()

{

SDA1; //拉高数据线

SCL1; //拉高时钟线

delayus(5); //延时

SDA0; //产生下降沿

delayus(5); //延时

SCL0; //拉低时钟线

}

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

//I2C停止信号

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

void I2C\_Stop()

{

SDA0; //拉低数据线

SCL1; //拉高时钟线

delayus(5); //延时

SDA1; //产生上升沿

delayus(5); //延时

}

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

//I2C发送应答信号

//入口参数:ack (0:ACK 1:NAK)

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

void I2C\_SendACK(uchar ack)

{

SDAOUT;

if(ack) SDA1;

else SDA0;

// SDA = ack; //写应答信号

SCL1; //拉高时钟线

delayus(5); //延时

SCL0; //拉低时钟线

delayus(5); //延时

}

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

//I2C接收应答信号

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

uchar I2C\_RecvACK()

{

uchar cy;

SCL1; //拉高时钟线

SDAIN;

delayus(5); //延时

if(SDADATA)

{

cy=1;

}

else

{

cy=0;

}

// cy = SDA; //读应答信号

SCL0; //拉低时钟线

delayus(5); //延时

SDAOUT;

return cy;

}

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

//向I2C总线发送一个字节数据

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

void I2C\_SendByte(uchar dat)

{

uchar i;

for (i=0; i<8; i++) //8位计数器

{

if((dat<<i)&0x80)

{

SDA1;

}

else

{

SDA0;

}

// SDA = cy; //送数据口

SCL1; //拉高时钟线

delayus(5); //延时

SCL0; //拉低时钟线

delayus(5); //延时

}

I2C\_RecvACK();

}

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

//从I2C总线接收一个字节数据

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

uchar I2C\_RecvByte()

{

uchar i;

uchar dat = 0,cy;

SDA1; //使能内部上拉,准备读取数据,

SDAIN;

for (i=0; i<8; i++) //8位计数器

{

dat <<= 1;

SCL1; //拉高时钟线

delayus(5); //延时

if(SDADATA)

{

cy=1;

}

else

{

cy=0;

}

dat |= cy; //读数据

SCL0; //拉低时钟线

delayus(5); //延时

}

SDAOUT;

return dat;

}

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

//向I2C设备写入一个字节数据

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

void Single\_WriteI2C(uchar REG\_Address,uchar REG\_data)

{

I2C\_Start(); //起始信号

I2C\_SendByte(SlaveAddress); //发送设备地址+写信号

I2C\_SendByte(REG\_Address); //内部寄存器地址，

I2C\_SendByte(REG\_data); //内部寄存器数据，

I2C\_Stop(); //发送停止信号

}

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

//从I2C设备读取一个字节数据

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

uchar Single\_ReadI2C(uchar REG\_Address)

{

uchar REG\_data;

I2C\_Start(); //起始信号

I2C\_SendByte(SlaveAddress); //发送设备地址+写信号

I2C\_SendByte(REG\_Address); //发送存储单元地址，从0开始

I2C\_Start(); //起始信号

I2C\_SendByte(SlaveAddress+1); //发送设备地址+读信号

REG\_data=I2C\_RecvByte(); //读出寄存器数据

I2C\_SendACK(1); //接收应答信号

I2C\_Stop(); //停止信号

return REG\_data;

}