简易转速控制系统设计

摘 要

以51单片机为控制核心，通过单片机I/O口输出PWM波，发送到光耦电路，控制外接+12v电压的导通与截止，从而间接控制风扇的转速，在对风扇转速进行调速的过程同时，采用霍尔元件对风扇的转速进行实时采集，并发送到单片机。使用PID算法修正实际转速，最后在LED显示屏上实时显示风扇的实际转速与设定转速。

本次设计由AT89C52单片机以及风扇、矩阵键盘、LED和霍尔传感器构成的转速闭环控制系统。

关键词 风扇；PWM；PID算法调速；AT89C52；

**目 录**

[摘 要 I](#_Toc77177537)

[1. 设计概述 1](#_Toc77177538)

[1.1设计内容 1](#_Toc77177539)

[1.2设计要求 1](#_Toc77177540)

[1.3设计难点 1](#_Toc77177541)

[2. 设计方案 2](#_Toc77177542)

[2.1系统方案 2](#_Toc77177543)

[2.2系统构成 2](#_Toc77177544)

[2.3电路工作原理 3](#_Toc77177545)

[2.3方案选择 4](#_Toc77177546)

[3. 设计实现 5](#_Toc77177547)

[3.1硬件电路设计 5](#_Toc77177548)

[3.1.1单片机整个控制模块 5](#_Toc77177549)

[3.1.2光耦电路设计 7](#_Toc77177550)

[3.1.3霍尔传感器 8](#_Toc77177551)

[3.1.4LED二极管 9](#_Toc77177552)

[3.1.5独立键盘控制模块 10](#_Toc77177553)

[3.2软件设计 11](#_Toc77177554)

[3.2.1总程序设计 11](#_Toc77177555)

[3.2.2PID函数调节模块 12](#_Toc77177556)

[3.2.3按键扫描模块 12](#_Toc77177557)

[3.2.4数码管显示模块 12](#_Toc77177558)

[4. 设计验证 13](#_Toc77177559)

[4.1验证总结 13](#_Toc77177560)

[4.2调试 14](#_Toc77177561)

[4.3完成部分 17](#_Toc77177562)

[4.4未完成部分 17](#_Toc77177563)

[5. 总结 18](#_Toc77177564)

[参考文献 19](#_Toc77177565)

[附录 20](#_Toc77177566)

# 设计概述

1.1设计内容

①能以数字的形式显示转速，并能通过按键给定转速，转速单位为：转/分钟;

②12V直流电机采用电脑机箱12V风扇代替；

③控制要求，从200转/分钟加速到400转/分钟，控制超调量≤120转，由给定阶跃变化开始道转速控制在400±80转的时间≤20秒，稳态误差≤10转；

③画出控制系统结构图。

1.2设计要求

1. 确定系统设计方案；
2. 进行系统的硬件设计；
3. 完成必要得到参数计算与元器件选择；
4. 完成应用程序设计；
5. 应用程序的调试。

1.3设计难点

①PID算法参数的设定，需要计算和不断的更改才能找到最佳参数；

②霍尔元件获取脉冲信号计数并且未在网络上找到相关配套原理图。

③光耦电路低电平控制。

④转速设置为r/min时很难找到合适的参数进行修改误差。

# 设计方案

2.1系统方案

针对本次课题的设计任务，分析得到：本次课程设计以AT89C52单片机为核心，以按键指令为输入端，控制风扇的转速。在设计中，采用改变占空比对风扇转速进行实时控制。

2.2系统构成

一个单片机应用系统的硬件电路设计应包含有两个部分内容：第一是系统扩展，即当单片机内部的功能单元，如ROM、RAM、I／O口、定时／计数器、中断系统等容量不能满足应用系统要求时，必须在片外进行扩展，选择适当的芯片，设计相应的电路。第二是系统配置，即按照系统功能要求配置外围设备，如键盘、显示器、打印机、D/A、A/D转换器等，并设计相应的接口电路。因此，系统的扩展和配置应遵循下列原则：

（1）尽可能选择典型电路，并符合单片机的常规用法。

（2）系统的扩展与外围设备配置应满足系统功能的要求，并留有适当的余量，以便进行二次开发。

（3）硬件结构应与应用软件方案统一考虑，软件能实现的硬件功能尽可能用软件来实现，但需注意的是软件实现占用CPU的时间。面且，响应时间比硬件长。

（4）单片机外接电路较多时，应考虑其驱动能力，减少芯片功耗，降低总线负载。

用单片机控制电路使之工作在占空比可调的开关状态，精确调整风扇转速。这种电路由于工作在管子的饱和截止模式下，效率非常高；PWM调速电路保证了可以简单地实现转速和方向的控制；电子开关的速度很快，稳定性也极佳，是一种广泛采用的PWM调速技术。我们采用了定频调宽方式，因为采用这种方式，风扇在运转时比较稳定；并且在采用单片机产生PWM脉冲的软件实现上比较方便。且对于风扇，采用软件延时所产生的定时误差在允许范围。其结构图如图 2‑1所示：

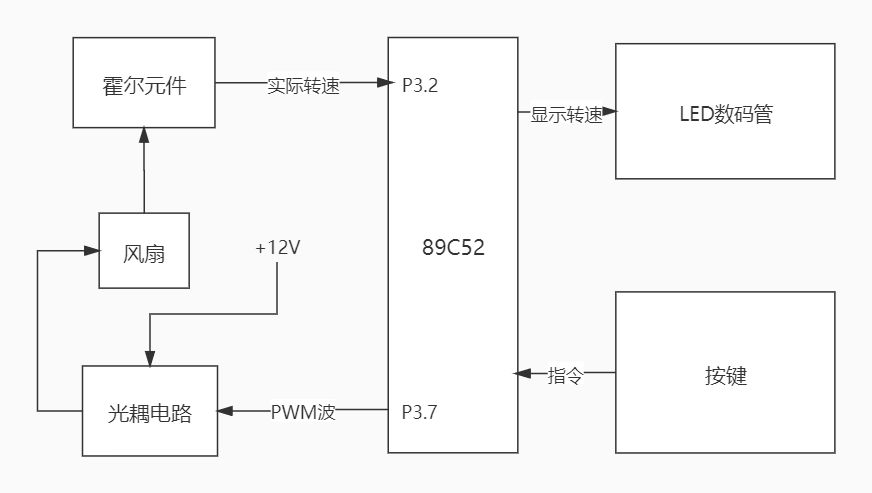


图 2‑1结构图

2.3电路工作原理

PWM是通过控制固定电压的直流电源开关频率，从而改变负载两端的电压，进而达到控制要求的一种电压调整方法。PWM可以应用在许多方面，如电机调速、温度控制、压力控制等。在PWM 驱动控制的调整系统中，按一个固定的频率来接通和断开电源，并根据需要改变一个周期内“接通”和“断开”时间的长短。通过改变直流电机电枢上电压的“占空比”来改变平均电压的大小，从而控制风扇的转速。

因此，PWM又被称为“开关驱动装置”。在脉冲作用下，当电机通电时，速度增加;电机断电时，速度逐渐减少。只要按一定规律，改变通、断电的时间，即可让电机转速得到控制。当我们改变占空比时，可以得到不同的电机平均速度，从而达到调速的目的。

在此设计中，因为光耦电路是使用低电平控制，即低电平所占比例越大，速度越快，反之，速度越慢。当实际速度大于设定速度是，低电平的占空比会减少，反之，占空比增加，如图 2‑2所示：

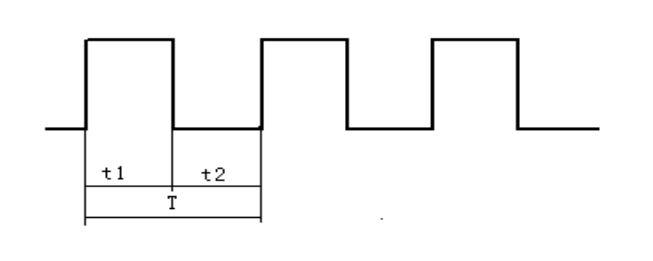


图 2‑2占空比方波

设光耦电路输入端始终接高电平时，电机转速为0，当始终接低电平时，风扇转速为最大，设占空比为，则电机的平均速度为，其中a指的是电机的平均速度;是指电机在全通电时的最大速度;是指占空比。

由上面的公式可见，当我们改变占空比时，就可以得到不同的电机平均速度,从而达到调速的目的。严格来说，平均速度与占空比D并非严格的线性关系，但是在一般的应用中，我们可以将其近似的看成是线性关系。

2.3方案选择

方案一：采用电阻网络或数字电位器调整风扇的分压，从而达到调速的目的。但是电阻网络智能实现有级调速，而数字电阻的元器件价格比较昂贵。更主要的问题在于一般风扇的电阻很小，但电流很大，分压不仅会降低效率，而且实现起来很困难。

方案二：采用继电器对风扇的开或关进行控制，通过开关的切换对电机的速度进行调整。这个方案的优点是电路较为简单，缺点是继电器的响应时间慢、机械结构容易损坏、寿命较短、可靠性不高。

方案三：采用PID算法控制PWM波形，外接光耦芯片4n35驱动电路，后使用霍尔元件进行感应测速。测速稳定且外围电路简单等优点。

综合各方面因素，采用了方案三。

# 设计实现

3.1硬件电路设计

（1）单片机控制模块

（2）光耦电路模块

（3）霍尔传感器模块

（4）数码管显示模块

3.1.1单片机整个控制模块

系统主要采用AT89C52单片机为主控芯片（芯片原理图如图 3‑1所示）

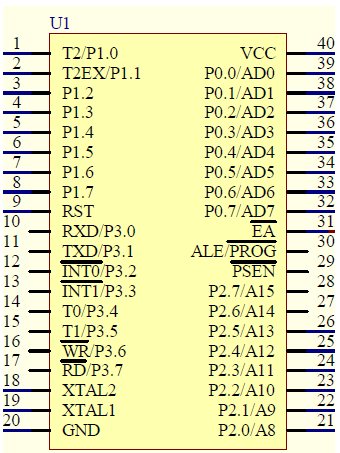


图 3‑1 AT89C52主控芯片

在单片机P2口读取按键指令，在P3^7输出可控的PWM方波，在P3^2(外部中断0)引脚输入霍尔元件的计数然后控制风扇的转速，总体原理图如图 3‑2所示。

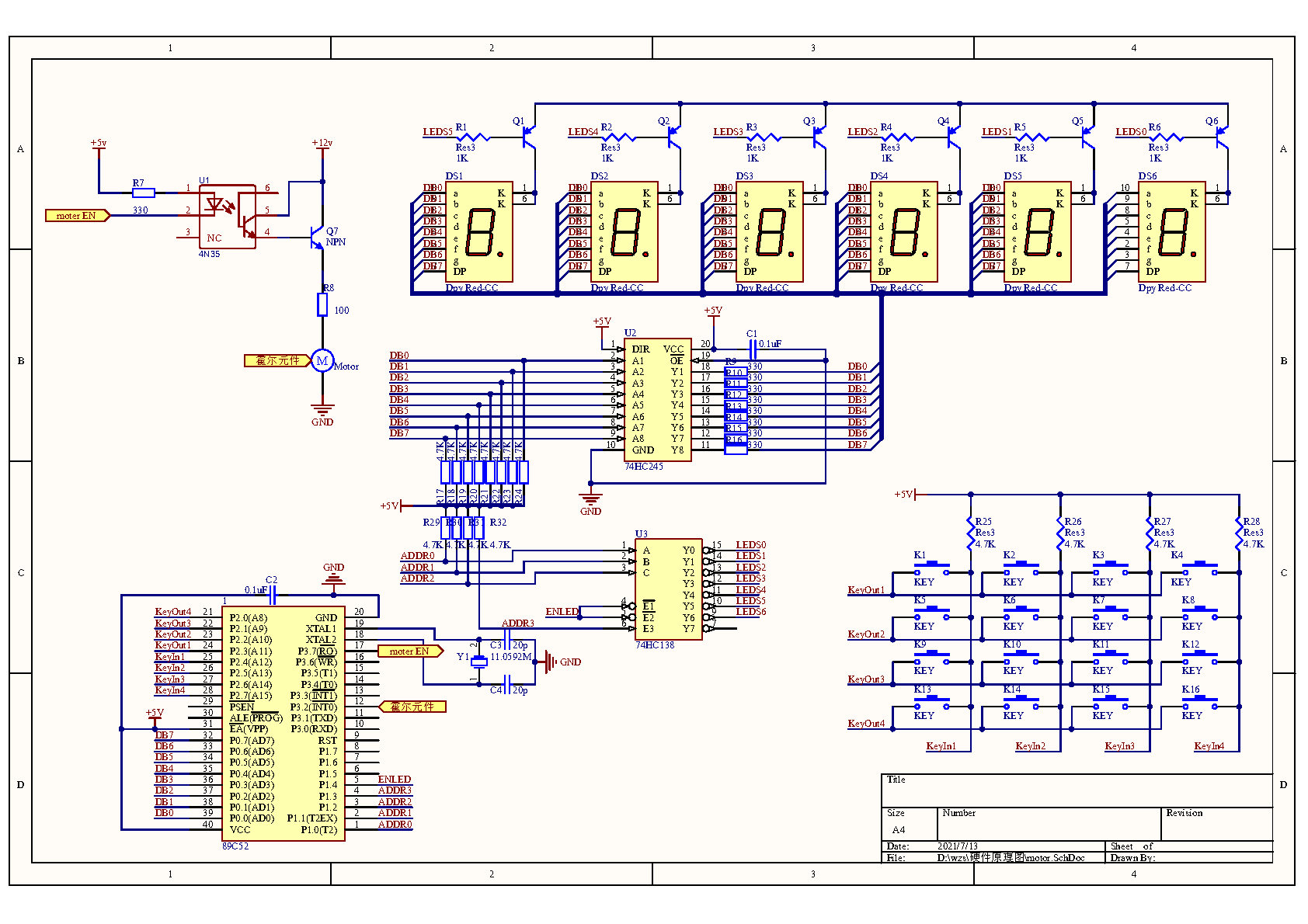


图 3‑2总体原理图

电源电路设计：

风扇驱动系统中，供电电源为系统的正常工作提供能源，本系统主要是采用笔记本电脑USB的+5V直流电源给单片机供电，外接+12V电源给光耦电路供电。

3.1.2光耦电路设计

工作原理：光电耦合器主要由三部分组成：光的发射、光的接收及信号放大。 光的发射部分主要由发光器件构成，发光器件一般都是发光二极管，发光二极管加上正向电压时，能将电能转化为光能而发光，发光二极管可以用直流、交流、脉冲等电源驱动，但发光二极管在使用时必须加正向电压。光的接收部分主要由光敏器件构成，光敏器件一般都是光敏晶体管， 光敏晶体管是利用PN结在施加反向电压时，在光线照射下反向电阻由大变小的原理来工作的。

光耦（其芯片型号为4n35，）的基本作用，是将输入、输出侧电路进行有效的电气上的隔离；能以光形式传输信号；有较好的抗干扰效果；输出侧电路能在一定程度上得以避免强电压的引入和冲击。光耦是一种转换器件，通过电到光再到电的转换过程。

通过这种转换就可以把前级和后级隔离开来，起到保护的作用。其芯片封装如图3-4所示：

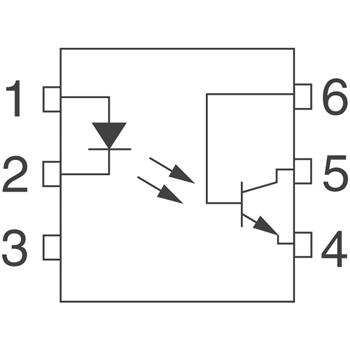


图 3‑4光耦芯片原理图

制作光耦一般把[发光二极管](https://www.eefocus.com/baike/481479)和[光敏三极管](https://www.eefocus.com/baike/480316)封装在一起，发光二极管为输入端，光敏三极管为输出端。

常使用的电路有电气绝缘，防止后级电路的高压干扰或者传入前级的弱电，破坏弱电内的器件的稳定性。

风扇背面的参数为12V，0.13A，即计算出采用12V电压，电阻采用100Ω，即电流最大为0.12A，在误差范围之内。

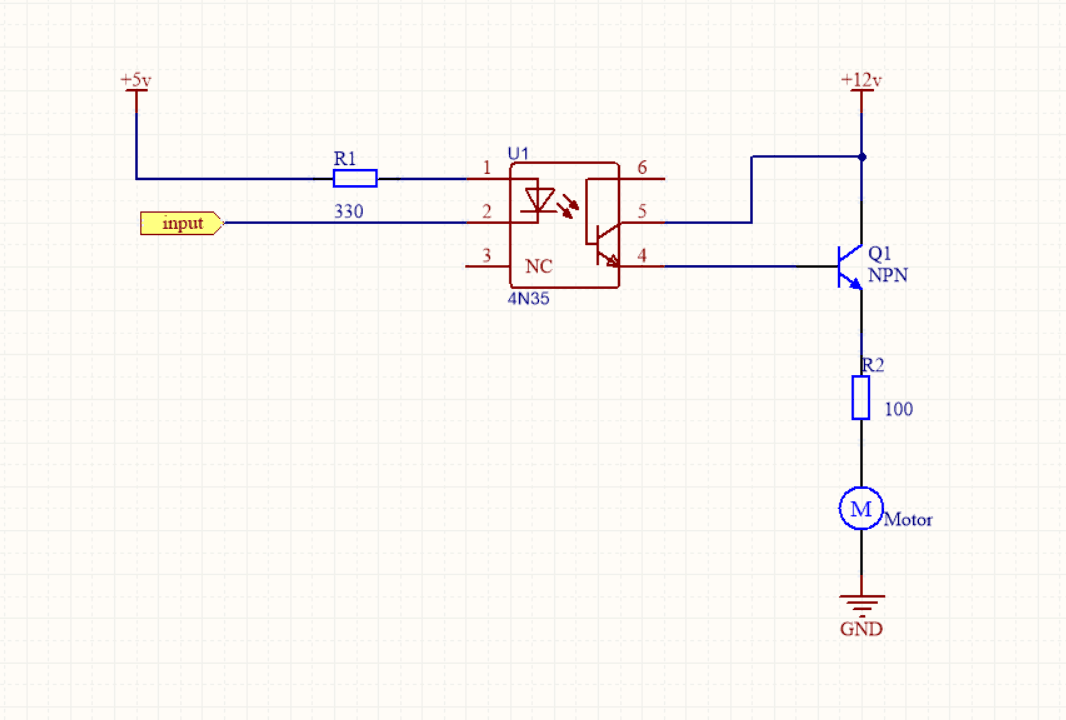
驱动电路也是常用的场合，微小的电流驱动发光二极管，然后使感光三极管收到微弱光照放大成较大的电压信号来驱动相关器件。设计的光耦电路原理图如图 3‑5所示：

图 3‑5光耦电路原理图

3.1.3霍尔传感器

霍尔效应：在一块半导体薄片上，其长度为l，宽度为b，厚度为d,当它被置于磁感应强度为B的磁场中，如果在它相对的两边通以控制电流I，且磁场方向与电流方向正交，则在半导体另外两边将产生一个大小与控制电流I和磁感应强度B乘积成正比的电势UH，即UH=KHIB，其中kH为霍尔元件的灵敏度。该电势称为霍尔电势，半导体薄片就是霍尔元件。

工作原理：霍尔开关集成电路中的信号放大器将霍尔元件产生的幅值随磁场强度变化的霍尔电压UH放大后再经信号变换器、驱动器进行整形、放大后输出幅值相等、频率变化的方波信号。信号输出端每输出一个周期的方波，代表转过了一个齿。单位时间内输出的脉冲数N，因此可求出单位时间内的速度V＝NT。

霍尔元件相关电路的原理为，磁铁每经过1次，在单片机引脚输入1次脉冲信号，因为贴了4个磁铁以稳定整体的重量，让风扇转的更加平稳，在代码中每检测到4次脉冲计数一次。

霍尔传感器用于测量电机转速时，一般是霍尔传感器固定安装，而在电机的旋转部位安装一个导磁性好的磁钢，旋转过程中，磁钢每接近霍尔传感器一次，霍尔传感器认为电机旋转了一圈，以此计算电机转速。霍尔传感器的电路原理图如图 3‑6所示：

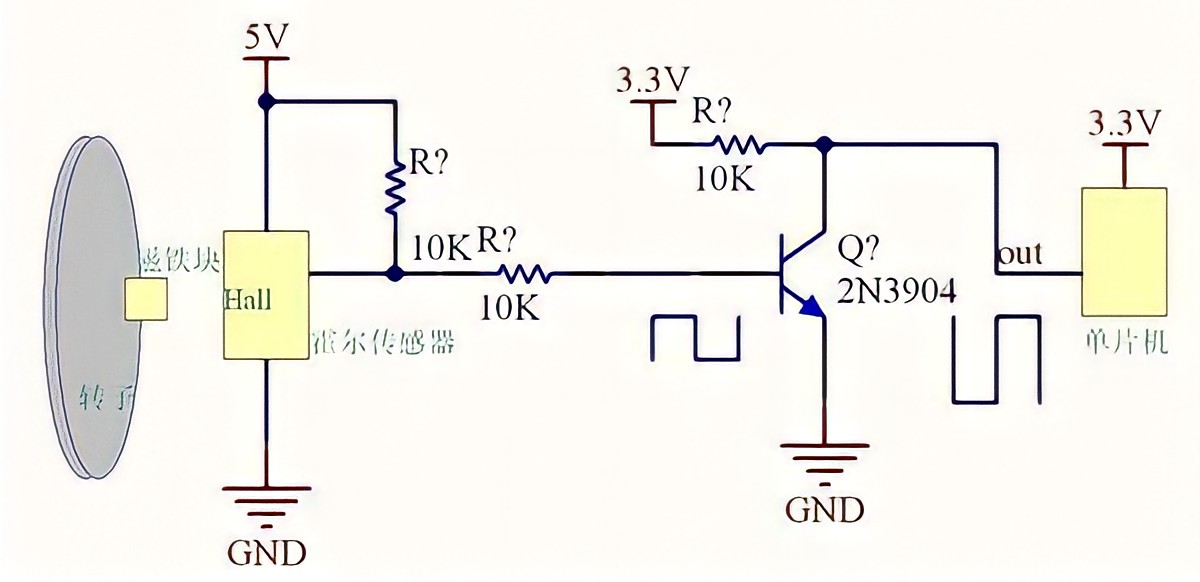


图 3‑6霍尔传感器电路原理图

3.1.4LED二极管

AT89C52采用的为共阳极数码管，一个数码管有八段：a,b,c,d,e,f,g,h即由八个发光二极管组成；发光二极管导通的方向是一定的（导通电压一般取为1.7V），这八个发光二极管的公共端有两种：可以分别接+5V（即为共阳极数码管）或接地（即为共阴极数码管）。

共阳极： 位选为高电平（即1）选中数码管, 各段选为低电平（即0接地时）选中各数码段。

共阴极： 位选为低电平（即0）选中数码管, 各段选为高电平（即1接+5V时）选中各数码段。

1位数码显示数字其实就是这8个发光二极管，哪个亮哪个不亮，从而达到显示数字的效果，亮和不亮无非就是0或1。以共阴极为例。

数码管的段依次从低到高如图中的a,b,c,d,e,f,g,h(h是小数点),如果要显示数字1，即b,c亮，其他都不亮，对应的二进制数字为:00000110,转换为16进制为0x06。即段码为0X06时，这个数码管就能显示数字1了。以此类推就可以得出1-9的断码了。共阳极的原理也是如此，对共阴极的段码取反即可。

8位数码管可以理解为8个1位数码管组成的，这就涉及到位码，你可以理解为位置，像第一位数码管，第二位数码管这样。以共阴极为例。位选为低电平（即0）时选中该数码管。

比如你有1个8八位数码管，你只想让第1、2、3、位数码管显示亮

起来，那么即选中1，2，3，其他均不选，二进制数字为11111000，转换为16进制为0XF8，只要设置位码为0xF8即可达到这个效果了，其他也是一样的，共阳极取反即可。

其原理图如图 3‑7所示：

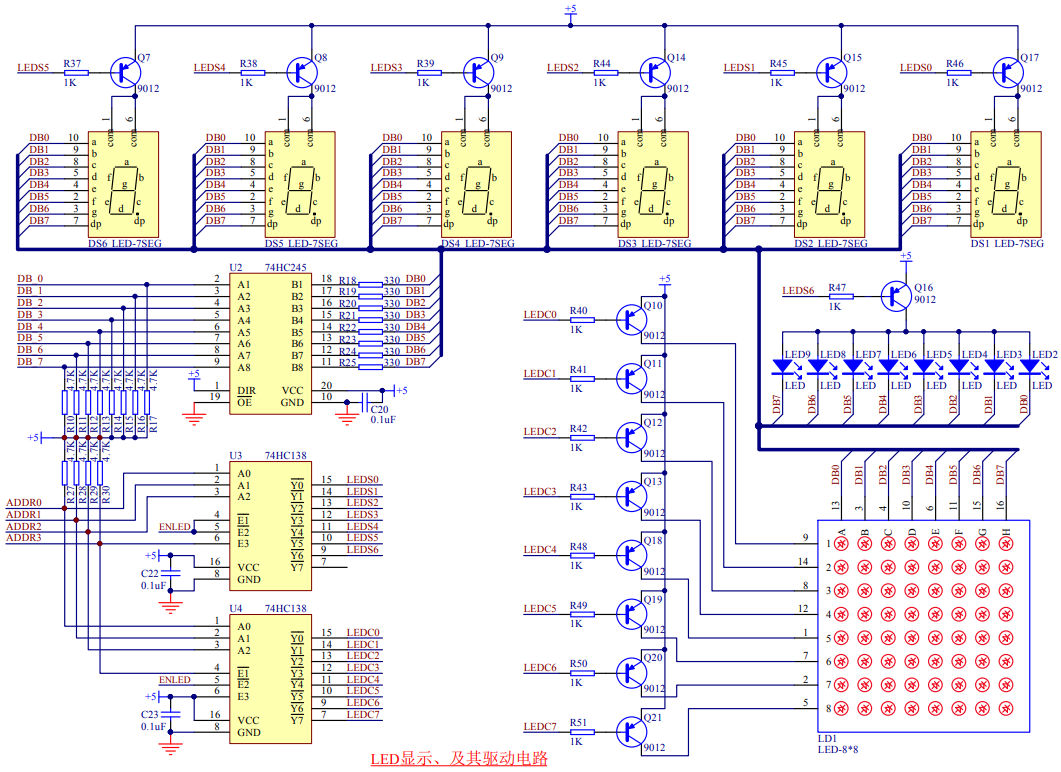


图 3‑7数码管电路原理图

前三个数码管显示实际测试速度(r/min)，后三个数码管显示规定速度(r/s)。

3.1.5独立键盘控制模块

键盘的功能及分类：

键盘是一种最常用的输入设备,它是一组按键的集合，从功能上可分为数字键和功能键两种，作用是输入数据与命令，查询和控制系统的工作状态，实现简单的人机对话。

键盘的分类

(a)键盘按照接口原理可分为编码键盘与非编码键盘两类。这两类键盘的主要区别是识别键符及给出相应键码的方法。

编码键盘主要是用硬件来实现对键的识别;

非编码键盘主要是由软件来实现键盘的定义与识别。

(b)键盘按照其结构可分为独立式键盘与矩阵式键盘两类。

独立式键盘主要用于按键较少的场合,矩阵式键盘主要用于按键较多的场合，也称行列式键盘

其原理图如图 3‑8所示：

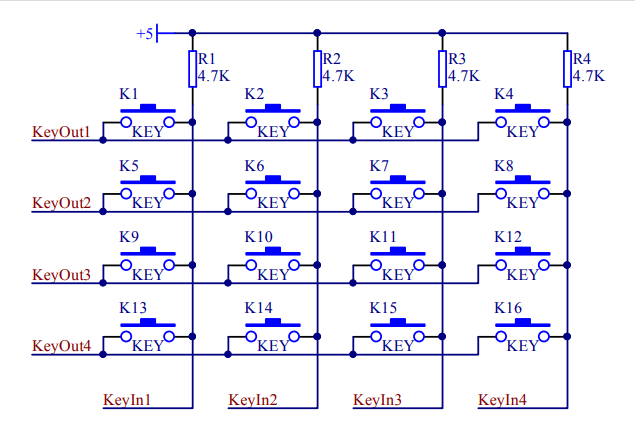


图 3‑8按键电路原理图

独立式键盘：

独立式键盘的按键相互独立，每个按键接一根I/O口线，一根I/O口线上的按键工作状态不会影响其它I/O口线的工作状态。因此，通过检测I/O口线的电平状态，即可判断键盘上哪个键被按下。

3.2软件设计

3.2.1总程序设计

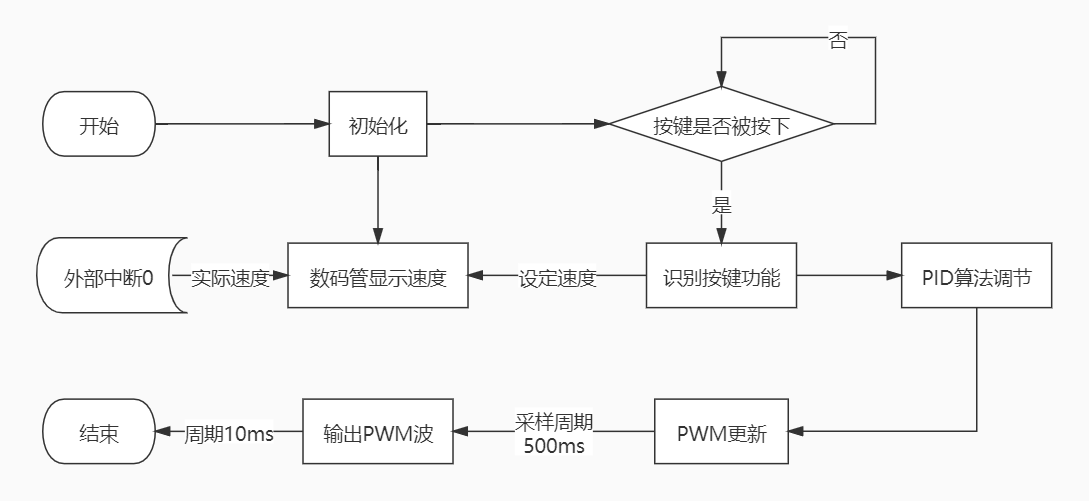


图 3‑9程序框架图

本课题我们所应用的是AT89C52单片进行一个简易的转速控制程序，我们所要实现的功能是经过PID算法调节PWM输出PWM波形，并且同时在按键的控制下，会在数码管显示设定的速度，外部中断0是霍尔元件的输入，其实际转速就是通过霍尔元件所感应，程序框架图如图 3‑9所示。

首先对风扇端口定义选择其输出引脚、按键端口定义及数码管端口定义，而后进行全局变量定义。

3.2.2PID函数调节模块

该模块我们定义参数tar\_speed目标速度和real\_speed实际速度及返回值pwm为PWM波的占空比。

PID算法我们要设计相关参数kp、ki和kd，还需要速度偏差（error），上一次速度偏差（last\_error\_1）和上上一次速度偏差（last\_error\_2）。此时便需要利用这些参数进行计算pwm的输出增量，知道pwm的输出增量，我们还需对输出占空比进行限幅。

3.2.3按键扫描模块

按键的布局如图 3‑10所示。

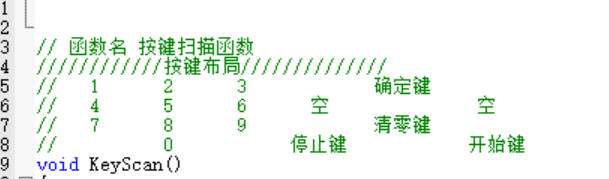


图 3‑10按键布局图

该部分首要任务是判定到底是哪个按键按下，然后实现对应功能。第一行第一个键按下为数字1，第一行第二个键按下为数字2，第一行第三个键按下为数字3，第一行第四个键按下为确定键，第二行第一个按键为数字4，第二行第二个按键为数字5，第二行第三个按键为数字6，第三行第一个按键为数字7，第三行第二个按键为数字8，第三行第三个按键为数字9，第三行第四个按键为停止键，第三行第五个按键为开始键。由单片机通过PID算法来控制电机转速接近设定值。从而通过0~9的数字键来直间接控制电机转速。

3.2.4数码管显示模块

在AT89C52单片机中有6个数码管，我们利用3—8译码法对其选定，前三个数码管为实际转速，后三个数码管显示理论转速。

# 设计验证

4.1验证总结

本次设计通过使用keil软件进行程序的编写，编译完成后生成.hex文件，然后使用STC单片机下载器下载到单片机开发板内。原理图是通过Altium Designer来绘制的。

硬件电路：光耦电路、霍尔元件感应芯片、单片机开发板电路；

使用仪器:直流电压发生器如图 4‑1所示、转速仪如图 4‑2所示；



图 4‑1直流电压发生器



图 4‑2转速仪

4.2调试

通过按键改变每秒几转的显示，然后其每分钟多少转在前三个数码管显示。

* 1. 240转/分钟 4转/秒如下图 4‑3所示;
  2. 240转/分钟PWM占空比方波如下图 4‑3所示。

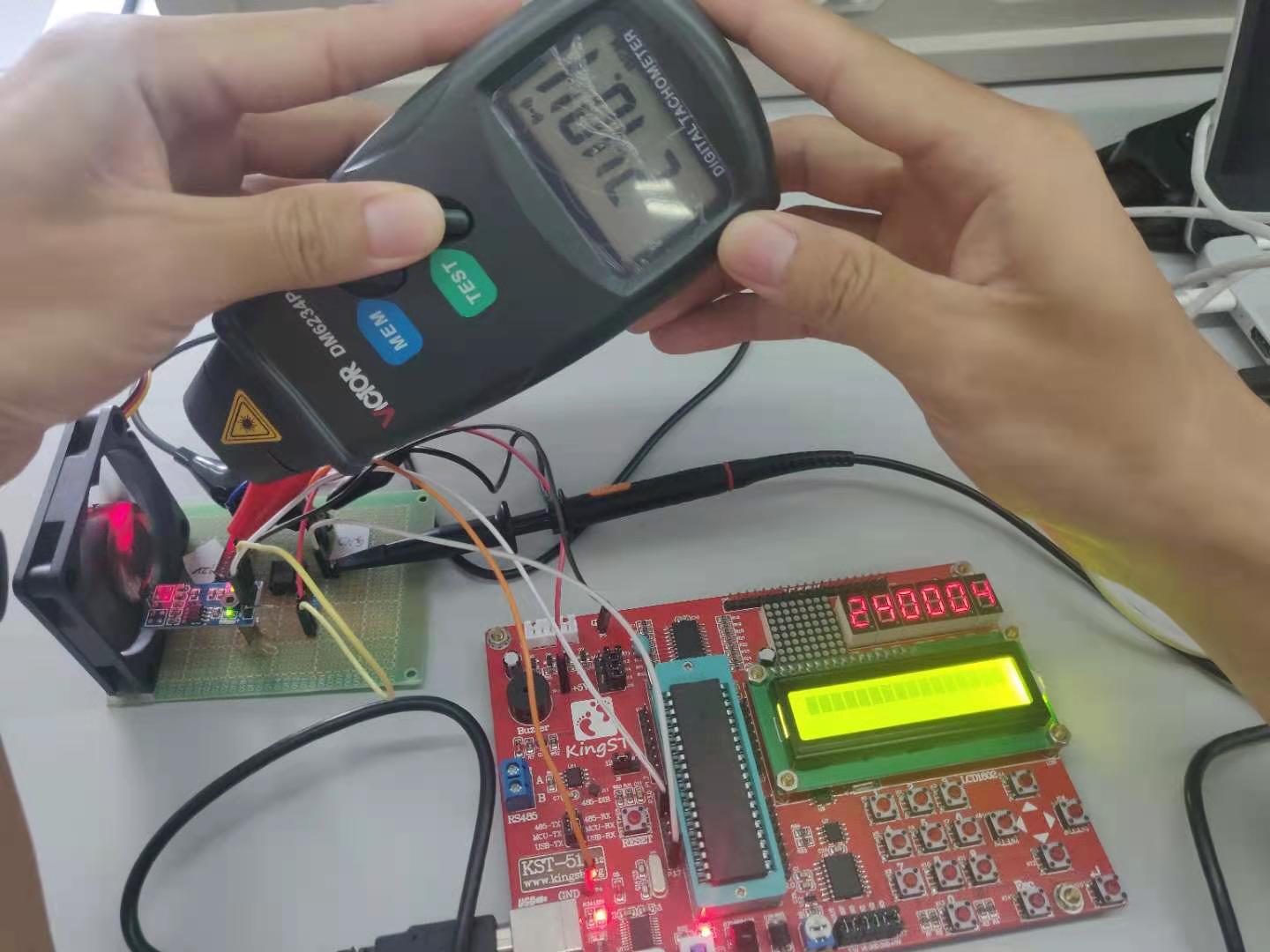


图 4‑3（a） 240转/分钟 4转/秒

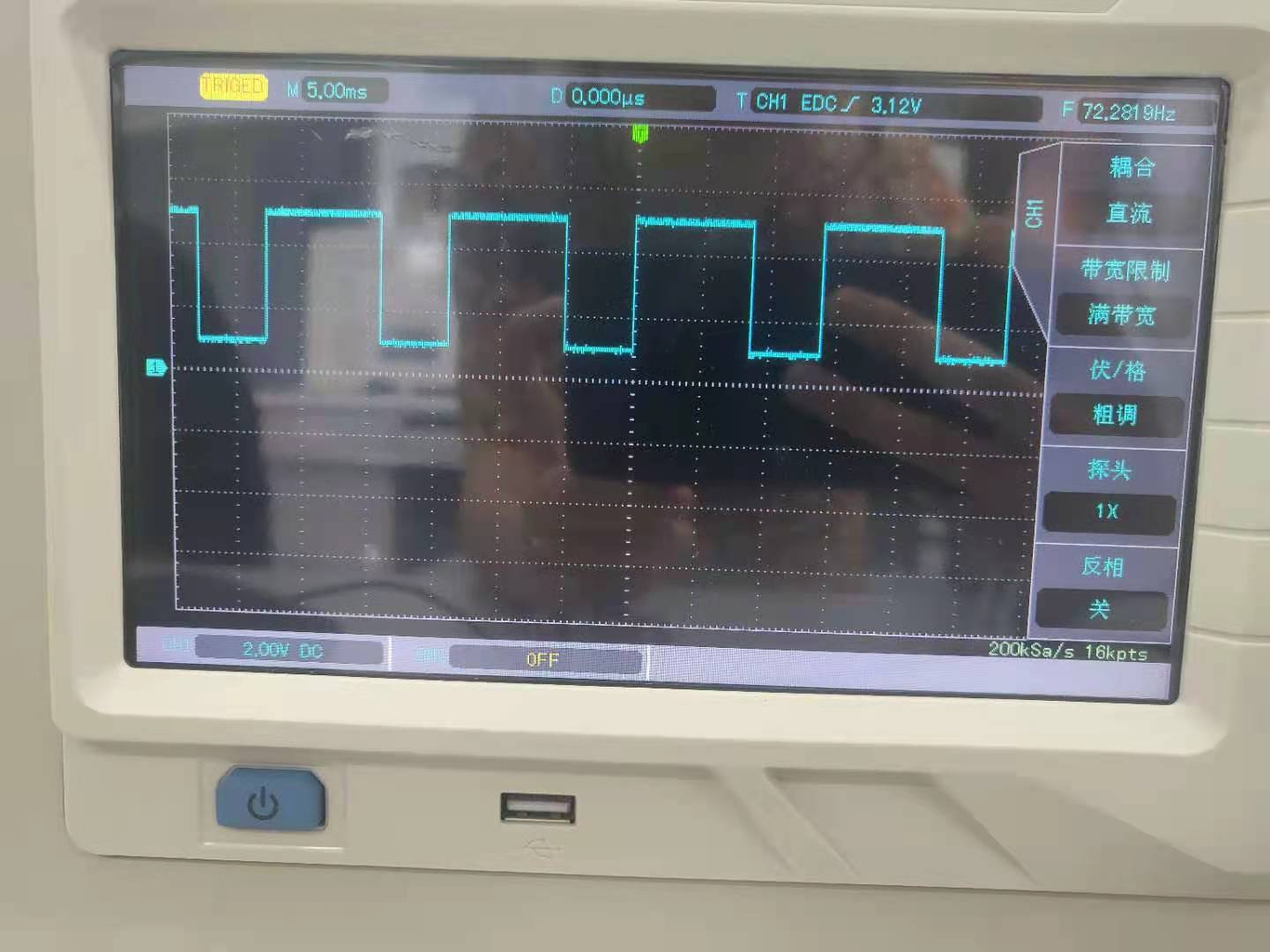


图 4‑3（b）240转/分钟PWM占空比方波

1. 360转/分钟 6转/秒如下图 4‑4所示；
2. 360转/分钟PWM占空比方波如下图4-2（b）所示

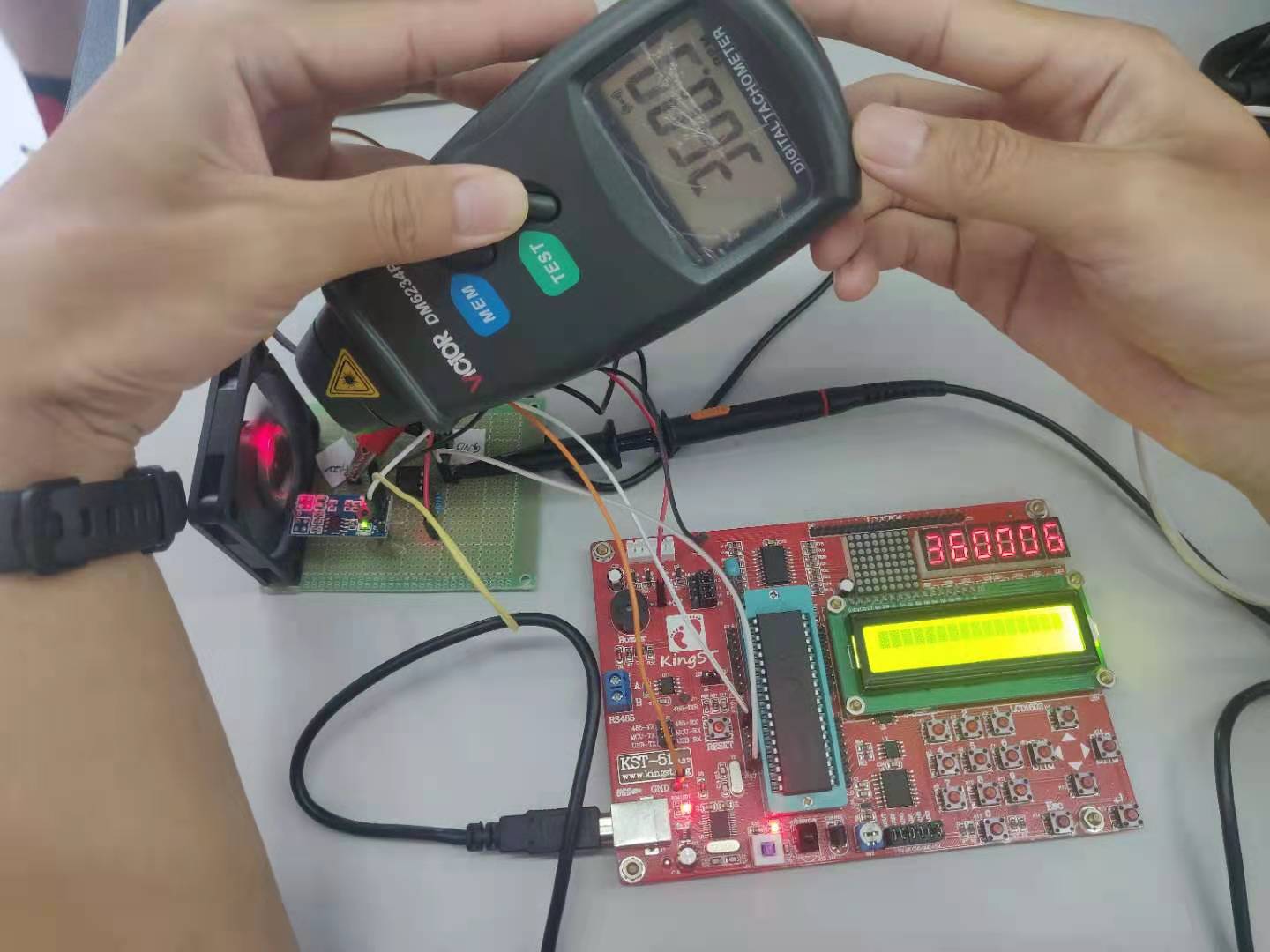


图 4‑4（a）360转/分钟 6转/秒

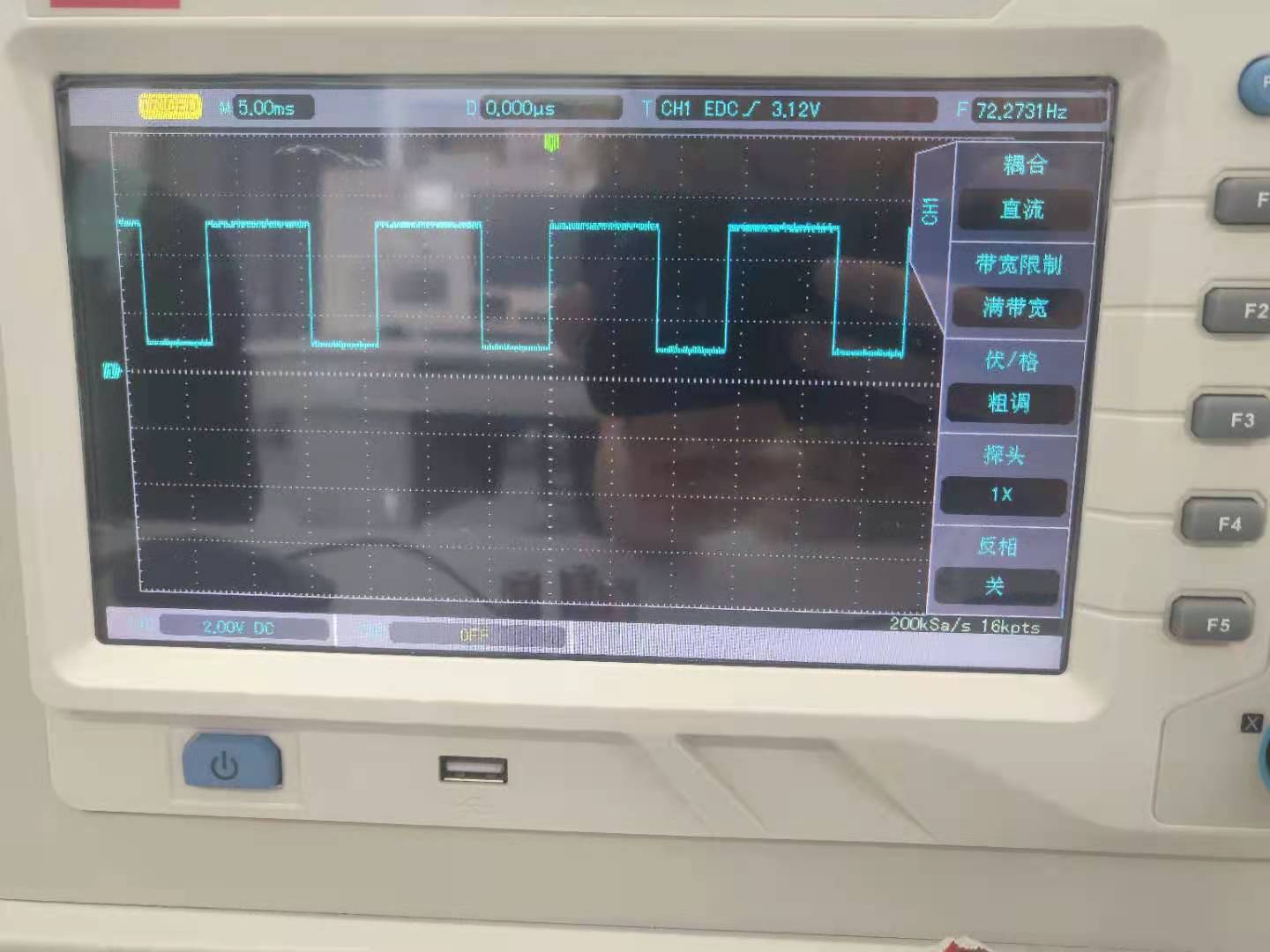


图 4‑4（b）360转/分钟PWM占空比方波

1. 480转/分钟 8转/秒如下图图 4‑5所示；
2. 480转/分钟PWM占空比方波如下图 4‑5所示



图 4‑5（a）480转/分钟 8转/秒

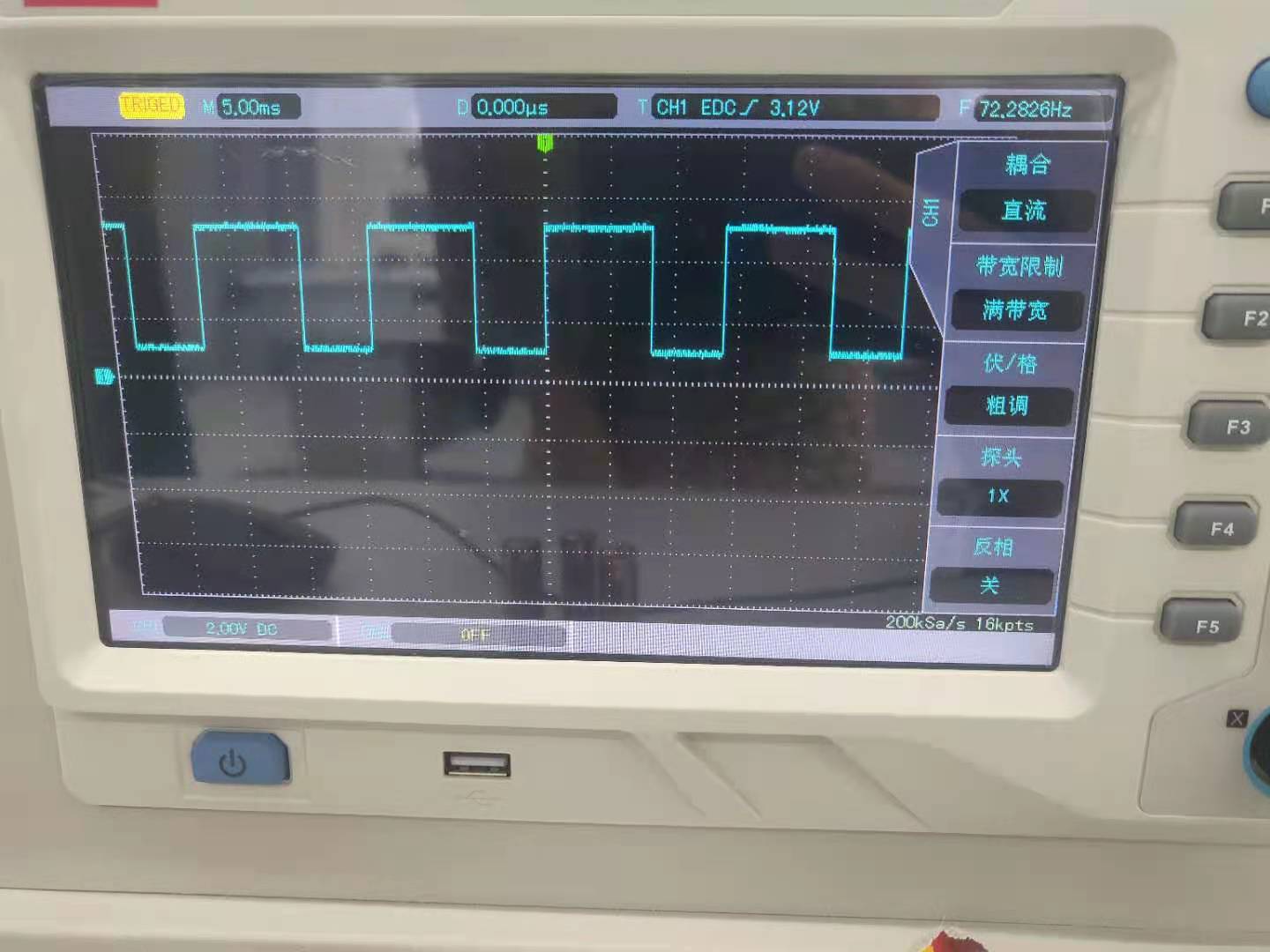


图 4‑5（b）480转/分钟PWM占空比方波

4.3完成部分

根据课题要求，我们完成了相应转速的设置，并在开发板的数码管的右边3个显示设定转速（r/s）,同时在霍尔元件获取实际转速，同时显示在开发板数码管的左边3位，显示实际转速（r/min）。

使用PID算法，对Kp,Ki,Kd三个参数进行计算，使实际转速与设定转速相接近，以消除误差。

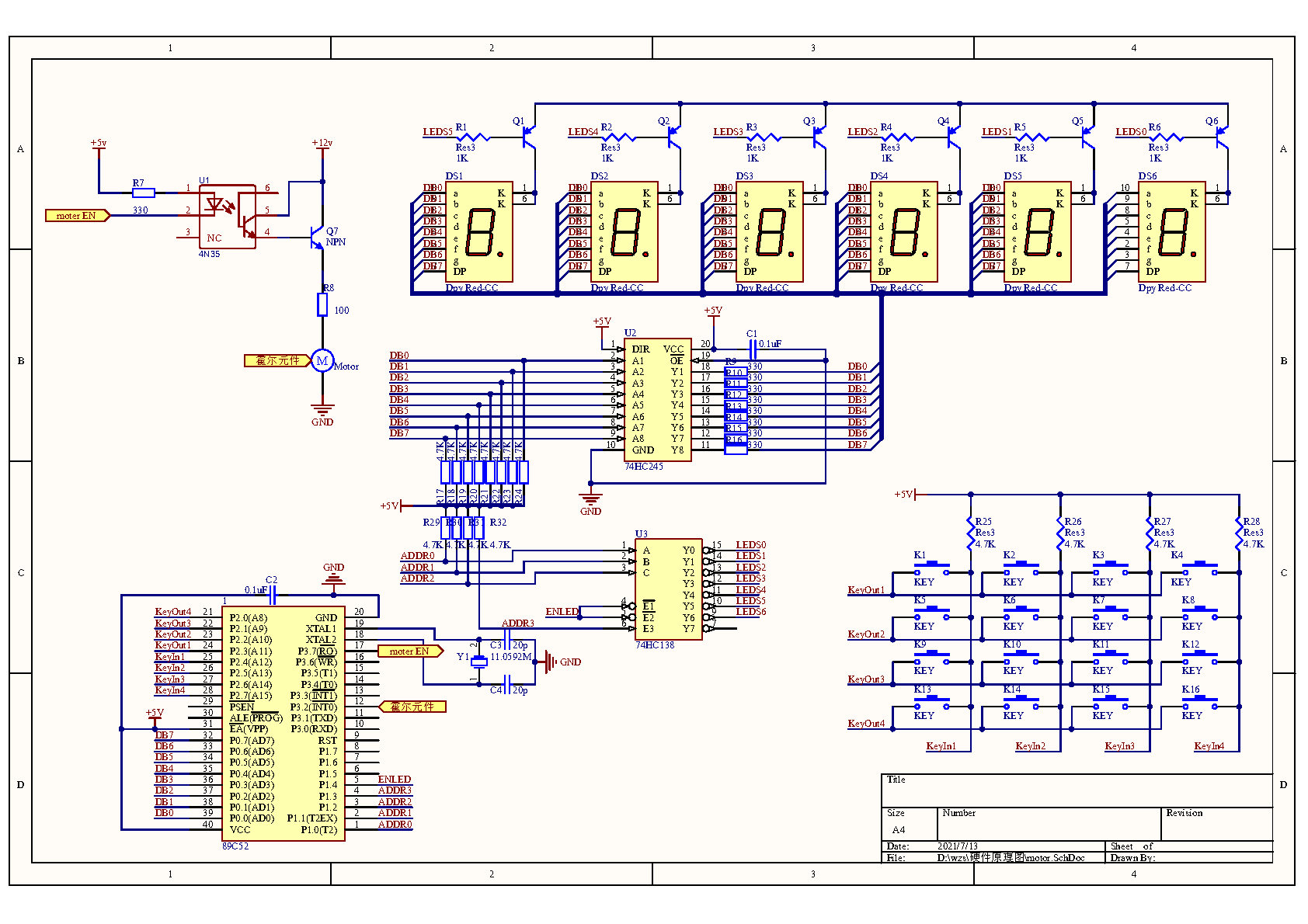
风扇转速从240r/min上升到360r/min，再上升到900转每分钟，误差范围在±60r/min，当240转-420转时误差较小，约±30r/min，在360r/min时几乎不存在误差。

4.4未完成部分

代码实现部分为r/s，未实现r/min更加精确的计算，代码中计算实现误差为1转，但因为单位是r/s，采用直接乘60，放大到r/min，代码中1r/s的误差即被放大为了±60r/min，是个不小的误差。未能解决此问题

附录

电路图



主程序源代码

#include "reg52.h"

#define uchar unsigned char

#define uint unsigned int

// 风扇端口定义,PWM输出端—>焊接的输入

sbit motor\_EN = P3^7; //pwm波输出端口

//手把手-数码管—38译码器

sbit ADDR0 = P1^0; // 数码管位选端口定义P1，段选端口:P2

sbit ADDR1 = P1^1;

sbit ADDR2 = P1^2;

sbit ADDR3 = P1^3;

sbit ENLED = P1^4; //使能LED

//按键矩阵-手把手

sbit KEY\_IN\_1 = P2^4;

sbit KEY\_IN\_2 = P2^5;

sbit KEY\_IN\_3 = P2^6;

sbit KEY\_IN\_4 = P2^7;

sbit KEY\_OUT\_1 = P2^3;

sbit KEY\_OUT\_2 = P2^2;

sbit KEY\_OUT\_3 = P2^1;

sbit KEY\_OUT\_4 = P2^0;

unsigned char code LedChar[] = { //数码管显示字符转换表

0xC0, 0xF9, 0xA4, 0xB0, 0x99, 0x92, 0x82, 0xF8,

0x80, 0x90, 0x88, 0x83, 0xC6, 0xA1, 0x86, 0x8E

};

unsigned char KeySta[4][4] = { //全部矩阵按键的当前状态

{1, 1, 1, 1}, {1, 1, 1, 1}, {1, 1, 1, 1}, {1, 1, 1, 1}

};

unsigned char backup[4][4] = { //按键值备份，保存前一次的值

{1, 1, 1, 1}, {1, 1, 1, 1}, {1, 1, 1, 1}, {1, 1, 1, 1}

};

/\*\*\*\*\*全局变量定义\*\*\*\*\*/

uint speed\_counter = 0; // 编码器输出计数器

uint speed = 0; // 实际速度

uint set\_speed = 10; // 设定速度

uint target\_speed = 10; // 目标速度

uint PWM = 0; // PWM波占空比 PWM %

int pwm\_reg = 0;

uchar motor\_begin = 0; // 电机开始转动标志位

uchar PID\_begin = 0; // PID计算标志位

// PID相关参数 P I D 参数,2.0,0.6,0.4

float Kp = 2.0, //Kp是加快系统响应速度，提高系统的调节精度

Ki = 1.0, //Ki用于消除稳态误差；

Kd = 0.4; //Kd改善系统的稳态性能

int error = 0; // 速度偏差

int last\_error\_1 = 0; // 上一次速度偏差

int last\_error\_2 = 0; // 上上一次速度偏差

/\*\*\*\*\*函数声明\*\*\*\*\*/

void display(uchar i, uchar number); // 显示函数

void delay(); // 延时函数

void KeyScan(); // 按键扫描函数

int PID(uint tar\_speed, uint real\_speed); // PID调节函数

// 函数名 主函数

void main(){

P0 = 0x00;

P1 = 0xf0;

TMOD = 0x01; //设置 T0 为模式 1

TH0 = 0xFC; //为 T0 赋初值 0xFC67，定时 1ms

TL0 = 0x67;

ET0 = 1; //使能 T0 中断

TR0 = 1; //启动 T0

EA = 1; //打开总中断

IT0 = 1; // 设置外部中断0触发方式, 0为低电平触发 1为下降沿触发

EX0 = 1; // 打开外部中断0

while(1){

KeyScan(); // 按键扫描

// 显示实际速度

display(1,speed\*60 / 100); delay();//r/min

display(2,speed\*60 % 100 / 10); delay();

display(3,speed\*60 % 10); delay();

// 显示设定速度

display(4,set\_speed / 100); delay();//r/s

display(5,set\_speed % 100 / 10);delay();

display(6,set\_speed % 10); delay();

// PWM更新

if(PID\_begin == 1){

PWM = PID(target\_speed, speed);

PID\_begin = 0;

}

}

}

// 函数名 PID调节函数

// 参 数 tar\_speed 目标速度

// real\_speed 实际速度

// 返回值 pwm PWM波占空比

int PID(uint tar\_speed, uint real\_speed){

static int pwm\_INC = 0, np = 0, ni = 0, nd = 0;

// 计算pwm输出增量

error = tar\_speed - real\_speed;

np = Kp \* (error - last\_error\_1);

ni = Ki \* error;

nd = Kd \* (error - 2 \* last\_error\_1 + last\_error\_2);

pwm\_INC = np + ni + nd;

pwm\_reg = pwm\_reg + pwm\_INC;

last\_error\_2 = last\_error\_1;

last\_error\_1 = error;

// 输出限幅

if(pwm\_reg < 0)

pwm\_reg = 0;

if(pwm\_reg > 100)

pwm\_reg = 100;

pwm\_INC = 0;

return pwm\_reg;

}

// 函数名 定时器0中断服务函数

void TR0\_isr() interrupt 1{

static uint i = 0;

static uint speed\_control = 0;

TH0 = 0xFF; // 定时器重装载值，定时100us

TL0 = 0x9C;

i++;

speed\_control++; //

// 霍尔元件采样周期 500ms-5000

if(i == 5000){

speed = speed\_counter/2; //4个磁铁，半秒---\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if(speed>15){ //限制速度

speed = 15;

}

speed\_counter = 0;

i = 0;

if(motor\_begin == 1) // PID计算一次

PID\_begin = 1;

}

// PWM波周期 10ms 频率 100Hz

if(speed\_control <= PWM)

motor\_EN = 0;

if(speed\_control > PWM)

motor\_EN = 1;

if(speed\_control >= 100)

speed\_control = 0;

}

// 函数名 外部中断0服务函数------P3^2（外部中断0的引脚所在位置）

void INT0\_isr() interrupt 0 {

speed\_counter++; //霍尔传感器输出P3.2

}

// 函数名 延时1ms函数

void delay(){

uchar i, j;

for(i = 0; i < 10; i++)

for(j = 0; j < 33; j++) ;

}

// 函数名 按键扫描函数

////////////按键布局//////////////

// 1 2 3 确定键

// 4 5 6 空 空

// 7 8 9 清零键

// 0 停止键 开始键

void KeyScan(){

static uchar ten\_flag = 0;

P2 = 0x0f; //P2初始化

EA = 1; //使能总中断

ENLED = 0; //选择数码管 DS1 进行显示

ADDR3 = 1;

ADDR2 = 0;

ADDR1 = 0;

ADDR0 = 0;

if (P2 != 0x0f)

{

switch(P2)

{

//第一行按键按下

case 0x07:

P2 = 0xf0;

switch(P2)

{

// 第一行第一个按键 数字1

case 0xe0 :

if(ten\_flag == 0)

{

set\_speed = 1;

ten\_flag = 1;

}

else

set\_speed = set\_speed \* 10 + 1 ;

break;

// 第一行第二个按键 数字2

case 0xd0 :

if(ten\_flag == 0)

{

set\_speed = 2;

ten\_flag = 1;

}

else

set\_speed = set\_speed \* 10 + 2;

break;

// 第一行第三个按键 数字3

case 0xb0 :

if(ten\_flag == 0)

{

set\_speed = 3;

ten\_flag = 1;

}

else

set\_speed = set\_speed \* 10 + 3;

break;

// 第一行第四个按键 确定键

case 0x70 :

target\_speed = set\_speed;ten\_flag=0; break;

} while(P2 != 0xf0); break;

//第二行按键按下

case 0x0b:

P2 = 0xf0;

switch(P2)

{

// 第二行第一个按键 数字4

case 0xe0 :

if(ten\_flag == 0)

{

set\_speed = 4;

ten\_flag = 1;

}

else

set\_speed = set\_speed \* 10 + 4;

break;

// 第二行第二个按键 数字5

case 0xd0 :

if(ten\_flag == 0)

{

set\_speed = 5;

ten\_flag = 1;

}

else

set\_speed = set\_speed \* 10 + 5;

break;

// 第二行第三个按键 数字6

case 0xb0 :

if(ten\_flag == 0)

{

set\_speed = 6;

ten\_flag = 1;

}

else

set\_speed = set\_speed \* 10 + 6;

break;

}while(P2 != 0xf0); break;

//第三行按键按下

case 0x0d:

P2 = 0xf0;

switch(P2){

// 第三行第一个按键 数字7

case 0xe0 :

if(ten\_flag == 0)

{

set\_speed = 7;

ten\_flag = 1;

}

else

set\_speed = set\_speed \* 10 +7;

break;

// 第三行第二个按键 数字8

case 0xd0 :

if(ten\_flag == 0)

{

set\_speed = 8;

ten\_flag = 1;

}

else

set\_speed = set\_speed \* 10 + 8;

break;

// 第三行第三个按键 数字9

case 0xb0 :

if(ten\_flag == 0)

{

set\_speed = 9;

ten\_flag = 1;

}

else

set\_speed = set\_speed \* 10 + 9;

break;

// 第三行第四个按键 清零键

case 0x70 : set\_speed = 0;ten\_flag=0;break;

}while(P2 != 0xf0); break;

//第四行按键按下

case 0x0e:

P2 = 0xf0;

switch(P2){

// 第四行第一个按键 数字0

case 0xe0 :

if(ten\_flag == 0)

{

set\_speed = 0;

ten\_flag = 0;

}

else

set\_speed = set\_speed \* 10;

break;

// 第四行第二个按键 停止键

case 0xd0 :

motor\_begin = 0; PWM = 0; pwm\_reg = 0;

break;

// 第四行第三个按键 开始键

case 0xb0 :

motor\_begin = 1;

break;

} while(P2 != 0xf0);break;

}

}

}

// 函数名 数码管显示函数

// 参 数 i : 选择第i个数码管显示 范围:1-6

// number : 显示数字number 范围:0-9

// 返回值 无

void display(uchar i, uchar number){ //3-8译码器选择哪个数码管点亮

switch (i){

case 1 : ADDR0 = 1; ADDR1 = 0;ADDR2 = 1; break;

case 2 : ADDR0 = 0; ADDR1 = 0;ADDR2 = 1; break;

case 3 : ADDR0 = 1; ADDR1 = 1;ADDR2 = 0; break;

case 4 : ADDR0 = 0; ADDR1 = 1;ADDR2 = 0; break;

case 5 : ADDR0 = 1; ADDR1 = 0;ADDR2 = 0; break;

case 6 : ADDR0 = 0; ADDR1 = 0;ADDR2 = 0; break;

}

P0 = LedChar[number];

}

//-------------按键扫描定时中断--手把手

void InterruptTimer0() interrupt 3{ //定时器3

unsigned char i;

static unsigned char keyout = 0; //矩阵按键扫描输出索引

static unsigned char keybuf[4][4] = { //矩阵按键扫描缓冲区

{0xFF, 0xFF, 0xFF, 0xFF}, {0xFF, 0xFF, 0xFF, 0xFF},

{0xFF, 0xFF, 0xFF, 0xFF}, {0xFF, 0xFF, 0xFF, 0xFF}

};

TH1 = 0xFC; //重新加载初值

TL1 = 0x67;

//将一行的 4 个按键值移入缓冲区

keybuf[keyout][0] = (keybuf[keyout][0] << 1) | KEY\_IN\_1;

keybuf[keyout][1] = (keybuf[keyout][1] << 1) | KEY\_IN\_2;

keybuf[keyout][2] = (keybuf[keyout][2] << 1) | KEY\_IN\_3;

keybuf[keyout][3] = (keybuf[keyout][3] << 1) | KEY\_IN\_4;

//消抖后更新按键状态

for (i=0; i<4; i++) //每行 4 个按键，所以循环 4 次

{

if ((keybuf[keyout][i] & 0x0F) == 0x00){ //连续 4 次扫描值为 0，即 4\*4ms 内都是按下状态时，可认为按键已稳定的按下

KeySta[keyout][i] = 0;

}

else if ((keybuf[keyout][i] & 0x0F) == 0x0F){ //连续 4 次扫描值为 1，即 4\*4ms 内都是弹起状态时，可认为按键已稳定的弹起

KeySta[keyout][i] = 1;

}

}

//执行下一次的扫描输出

keyout++; //输出索引递增

keyout = keyout & 0x03; //索引值加到 4 即归零

switch (keyout) //根据索引，释放当前输出引脚，拉低下次的输出引脚

{

case 0: KEY\_OUT\_4 = 1; KEY\_OUT\_1 = 0; break;

case 1: KEY\_OUT\_1 = 1; KEY\_OUT\_2 = 0; break;

case 2: KEY\_OUT\_2 = 1; KEY\_OUT\_3 = 0; break;

case 3: KEY\_OUT\_3 = 1; KEY\_OUT\_4 = 0; break;

default: break;

}

}

教师评语