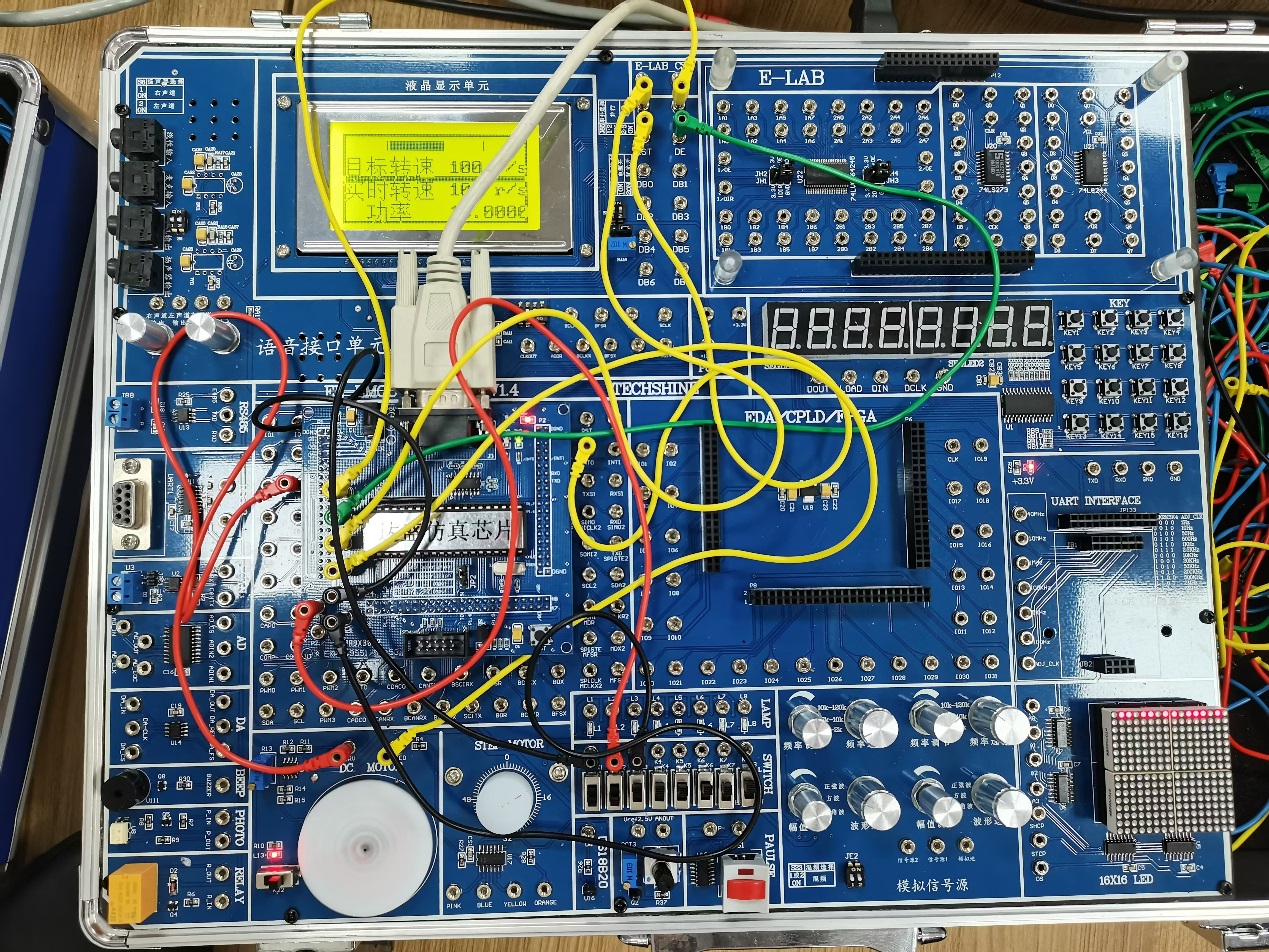
**实验报告七 直流电机脉宽调制调速**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. 电路图

如图进行电路链接。



1. 程序分析

1、流程图的形式给出程序设计的思路，

图示

描述已自动生成

2、代码及必要注释。

#include <reg51.h>

#include <intrins.h>

#include <stdio.h>

#include "st7920.h"

#include "tlc2543.h"

#include "delay.h"

#include "math.h"

#define MAX\_N 30000 // 最大累加值, 即最大占空比

#define MAX\_SPEED 180 // 最大转速, 单位 r/s, 测试得到

#define TICK\_TIME\_US 100 // 每个tick周期 = 100us

#define TICK\_PER\_MS (1000/TICK\_TIME\_US) // 每毫秒的tick数

#define TIMER\_PRESET\_VALUE TICK\_TIME\_US // 定时器的预设值, 设为此值 = 100us 时，定时器每tick中断一次

int targetSpeed; // 目标转速

int motorSpeed; // 实时转速

unsigned int motorN = 0; // 越大转越快

sbit motorDrive = P0^5; // 电机驱动引脚

sbit key1 = P2^4;

sbit key2 = P2^5;

sbit key3 = P2^6;

// 绘制边框

void drawBorder(void) {

unsigned char t, y;

LCD\_startGraphic(); // 进入绘图模式

// Draw top border

for (y = 0; y < 2; y++) { // 宽度为2, 0-1

LCD\_set\_graphic\_address(0, y); //设置绘图模式的起始地址 (重要；以实际显示屏左上角为原点的坐标系)

for (t = 0; t < 16; t++) { // 一次写两个字节

LCD\_write\_data(0xFF); // Fill entire row

LCD\_write\_data(0xFF); // Fill entire row

// 写入两个字节，系统内部 x 地址自动加 1

}

}

// Draw bottom border

for (y = 62; y < 64; y++) { // 宽度为2, 62-63

LCD\_set\_graphic\_address(0, y);

for (t = 0; t < 16; t++) {

LCD\_write\_data(0xFF); // Fill entire row

LCD\_write\_data(0xFF); // Fill entire row

}

}

// Draw left border

for (y = 2; y < 62; y++) { // 高度为60, 2-61

LCD\_set\_graphic\_address(0, y);

LCD\_write\_data(0xC0); // Fill first column

//LCD\_write\_data(0x00); // A half byte for extra column

}

// Draw right border

for (y = 2; y < 62; y++) {

LCD\_set\_graphic\_address(14 \* 8, y);

LCD\_write\_data(0x00); // A half byte for the second last position

LCD\_write\_data(0x03); // Fill last column

}

for (y =30;y<=33;y++) {

LCD\_set\_graphic\_address(0, y);

LCD\_write\_data(0x3F);

LCD\_write\_data(0xFF);

for (t = 16/16; t<=96/16;t++)

{

LCD\_write\_data(0xFF);

LCD\_write\_data(0xFF);

}

LCD\_set\_graphic\_address(112, y);

LCD\_write\_data(0xFF);

LCD\_write\_data(0xFC);

}

LCD\_endGraphic();

}

// 绘制电机速度条

// 位置为 x = 10~109, y = 5~12

void displayMotorSpeedBar(int currentSpeed, int maxSpeed) {

unsigned char t, y;

int barLength = (64 \* currentSpeed) / maxSpeed;

int barHeight = 8; // Define the height of the bar

LCD\_startGraphic();

// clear bar first

for (y = 5; y < 4 + barHeight; y++){ // Positioning vertically in the screen

LCD\_set\_graphic\_address(16, y); // Start after the left border

for (t = 0; t < 96 / 8; t++) { // Calculating how many full bytes

LCD\_write\_data(0x00); // Fill byte

}

}

// draw starter and ender

for (y = 5; y < 4 + barHeight; y++){ // Positioning vertically in the screen

LCD\_set\_graphic\_address(16, y); // Start after the left border

LCD\_write\_data(0x00);

LCD\_write\_data(0x02);

LCD\_set\_graphic\_address(96, y);

LCD\_write\_data(0x40);

LCD\_write\_data(0x00);

}

// x = 10~109, y = 5~12

for (y = 5; y < 4 + barHeight; y++) { // Positioning vertically in the screen

LCD\_set\_graphic\_address(32, y); // Start after the left border

for (t = 0; t < barLength / 8; t++) { // Calculating how many full bytes

LCD\_write\_data(0xFF); // Fill byte

}

LCD\_write\_data(0xFF << (8 - (barLength % 8))); // Fill remaining part of byte if any portion left

}

LCD\_endGraphic();

}

void printScreen(void) {

char printText[16];

// Draw border

drawBorder();

displayMotorSpeedBar(motorSpeed, MAX\_SPEED);

//LCD\_clear();

sprintf(printText, "目标转速 %3d r/s", targetSpeed);

LCD\_write\_string(1 << 8, printText);

sprintf(printText, "实时转速 %3d r/s", motorSpeed);

LCD\_write\_string(2 << 8, printText);

sprintf(printText, " 功率 %4f% ", (motorN / (MAX\_N/100.0)));

LCD\_write\_string(3 << 8, printText);

// Display motor speed as bar

//LCD\_clear();

}

// PWM

// 每个tick周期触发一次

// 使用累加进位法，输出0/1，平滑控制电机驱动

void motorControl(void)

{

// motorS 是累加器

// motorN 是更新的 每个周期内高电平维持的时间长度

// MAX\_N 是最大累加值

static unsigned int motorS = 0; // 累加器

motorS += motorN;

if(motorS > MAX\_N)

{

motorS -= MAX\_N;

motorDrive = 0;

}

else

{

motorDrive = 1;

}

}

float integral = 0, last\_error = 0;

const float Kp = 0.010, Ki = 0.007, Kd = 0.007;

// PID算法更新函数

void updateN(void)

{

// 计算当前误差

float error, derivative, output;

error = targetSpeed - motorSpeed;

// 计算积分和微分

integral += error;

derivative = error - last\_error;

last\_error = error;

// 计算输出

output = Kp \* error + Ki \* integral + Kd \* derivative;

// 限制输出范围

if(output > 1) output = 1;

if(output < 0) output = 0;

// 更新 motorN, 即每个周期内高电平维持的时间长度。

motorN = (unsigned int) (MAX\_N \* output);

}

// 读取目标转速, 通过ADC读取

void readTargetSpeed(void)

{

unsigned int value;

int newSpeed;

value = read2543(0);

value = value > 4000 ? 4000 : value;

newSpeed = value / 4000.0 \* MAX\_SPEED;

if(newSpeed - targetSpeed > 1 || targetSpeed - newSpeed > 1)

{

targetSpeed = newSpeed;

integral = 0;

last\_error = 0;

}

}

// 定时器中断服务函数

// 每个tick周期 = 100us = 0.1ms

// 功能：计算时间; 控制电机驱动输出

unsigned long int ticks = 0;

void timerISR(void) interrupt 1

{

ticks += 1; // 更新ticks

motorControl(); // 累加进位法控制电机驱动输出

TH0 = (65536 - TIMER\_PRESET\_VALUE) / 256; // 设置定时器的高字节

TL0 = (65536 - TIMER\_PRESET\_VALUE) % 256; // 设置定时器的低字节

}

// 外部中断服务函数

// 每次电机转一圈，触发一次

unsigned int motorRound;

void motorISR(void) interrupt 0 using 0

{

motorRound += 1;

}

// 更新电机速度

// 每0.5s更新一次

void updateMotorSpeed(void)

{

static unsigned long int lastTick; // 上次更新时的ticks值

float interval = (ticks - lastTick) \* TICK\_TIME\_US / 1000000.0; // 计算两次更新的时间间隔

if(interval >= 0.5) // 大于0.5s时更新

{

lastTick = ticks;

motorSpeed = motorRound / interval;

motorRound = 0;

}

}

void wait\_ms(unsigned int ms)

{

unsigned long int now = ticks;

while(ticks - now < ms \* TICK\_PER\_MS);

}

void readTargetSpeed\_ByKey(void)

{

int newSpeed=0;

// fill in this part to set a newSpeed

// ...

// ...

if (key1) newSpeed += 50;

if (key2) newSpeed += 50;

if (key3) newSpeed += 50;

if (newSpeed - targetSpeed > 1 || targetSpeed - newSpeed > 1)

{

targetSpeed = newSpeed;

integral = 0;

last\_error = 0;

}

}

int main()

{

unsigned char i;

int speed\_back;

// 设置定时器的工作模式和预设值

// 定时器0中断触发间隔为 100us = 0.1ms

TMOD = 0x01; // 定时器0工作在模式1

TH0 = (65536 - TIMER\_PRESET\_VALUE) / 256; // 设置定时器的高字节

TL0 = (65536 - TIMER\_PRESET\_VALUE) % 256; // 设置定时器的低字节

TR0 = 1; // 开启定时器并使能定时器中断

ET0 = 1;

IT0 = 1; // 外部中断0为低电平触发

EX0 = 1; // 开EX0中断

EA = 1;

LCD\_init();

LCD\_clear();

//LCD\_write\_string(0 << 8,"实验7&8 PID电机");

LCD\_write\_string(1 << 8,"目标转速 000 r/s");

LCD\_write\_string(2 << 8,"实时转速 000 r/s");

//LCD\_write\_string(3 << 8," 运行中 ");

//drawLCD();

// 定时器中断时，更新电机速度

// 外部中断时，更新电机圈数 (用于计算速度)

// 主循环中，更新目标转速，更新电机控制信号，更新屏幕显示

while(1)

{

readTargetSpeed\_ByKey();

//readTargetSpeed(); // 读取目标转速 (通过ADC读取)

updateMotorSpeed(); // 更新电机速度 (通过外部中断，计算电机转速)

updateN(); // 更新占空比 (通过PID算法，更新motorN)

printScreen();

wait\_ms(500);

}

return 0;

}

1. 思考题（见PPT）

1. 闭环控制相比于开环控制的优点有哪些？

**改善系统响应性能：** 闭环控制可提升系统精度。反馈可以校正由于负载变化、外部扰动、或系统参数漂移引起的误差。

**增强稳定性：** 当系统参数发生变化时，闭环系统可以自动适应这些变化，维持系统稳定运行。

**减少外部扰动影响：** 闭环控制系统可以减少外界干扰，比如电源波动、温度变化对系统造成的影响。

**自适应与自调节：** 闭环系统可以自我调整以达到最佳工作状态，例如PID控制系统可以根据反馈调整控制量，以达到预期输出。

**容错性：** 闭环系统能够感知并校正由于内部故障而产生的偏差。

**实现自动控制：** 闭环控制系统不需要人工干预即可实现预设的控制任务。

2. 说明实验原理中累加进位法的正确性。

通过控制累加值 ，我们可以调整占空比。设定累加器初始值为0，每个时钟周期累加同一个值 ，当累加器值达到 时产生进位。

一个PWM周期由以下两部分组成：

1. 累加值从 增加到 的时间，这段时间内输出高电平。
2. 累加器溢出后直至再次达到 的时间，这段时间内输出低电平。

由于累加器是在溢出时清零，不管累加器当前值如何，它都将在 周期后再次溢出。

因此，如果累加步长 是 的一个分数，累加器将以稳定的频率溢出，生成稳定的PWM波形，其中高电平时间占整个周期的 。

通过动态更改 的值，可以在不更改定时器的基础频率的情况下动态改变PWM信号的高电平持续时间，这即是调整PWM占空比。

3. 如果电机转速不是很慢的情况下，也可以通过测量转动一圈的时间来测速，讨论这种方法的实现框架。

通过测量电机转动一圈的时间来计算转速的方法，关键在于精确测量时间间隔。实现框架大致如下：

**中断服务：**使用传感器产生的脉冲触发中断。在中断服务例程中，记录下两个连续脉冲的时间戳。

**时间测量：**利用定时器来提供时间戳，用以测量两个脉冲间的时间差。

**计算速度：**通过时间差计算转速（转速 = 1 / 测量时间）。

**数据平滑：**为了得到平滑的速度读数，可以实施滤波算法。

4. 查阅ST7920的高级功能，分析这些功能在实际中的应用场景。

ST7920是一款常用于液晶显示器(LCD)的控制芯片，支持多种显示模式，包括文字和图形。ST7920的一些高级功能及其应用场景包括：

**文字和图形混合显示：** 在界面中同时展示文字和图形（如符号、简单形状等）。适用于需要同时显示数据和图标的情况，比如工业仪表的用户界面。

**内部字符ROM和RAM：** ST7920内有字符生成的ROM，可以直接显示常用字符和符号，而RAM区可存储自定义字符。这在需要显示专有信息符号或非标准文字符号的应用中非常有用。

**二级图形缓冲区：** 支持多层图形处理，这在复杂的图形显示中有用，比如游戏界面、高级仪表盘等。

1. 问题分析
   * + 1. 转速测量中，若采用测量转动时间得到直流电机转速，在部分实验器材上，由于机械故障可能导致转速不稳定。若采用算法进行数据平滑，运算时间过长且编程复杂。故最终仍然采用计数的方式得到转速，进行PID控制。最终实验结果稳定、转速控制精准，但是系统反应时间稍长。