光伏发电跟踪器
项目报告

指导老师：	祁才君、张伟
学　　院：	电气工程学院
项目成员：	陈捷润 3140100861 张建佳 3140104497

二〇〇六年七月

目录

摘要：

由于太阳位置随时间而变化，使得光伏发电系统的光伏电池光照强度不稳定，不能充分利用太阳能资源。光伏发电跟踪装置可以使光伏电池始终跟踪太阳的运动轨迹，从而提高发电效率达 35%左右，具有很高的实用价值。本项目以 MSP430F5438A 单片机为核心控制单元，以 PID 控制为核心算法，协调控制外围设备，包括模拟光源、模拟光伏发电板、光敏检测电路、ADS1118 串口 A/D 转换电路、液晶显示与键盘电路、DRV8412 全桥 PWM 驱动电路和直流电机。

关键词： MSP430、PID 控制、太阳跟踪、AD 转换

Abstract：

Photovoltaic power generation system can't perform its full potential due to the unstable light intensity fallen on the photovoltaic cells as the sun position changes with time. As the practical solution to the problem, photovoltaic power tracking device keeps real-time tracking of the sun, largely improving the power generation efficiency by 35 percent. Based on the core control unit,MSP430F5438A, and the core PID algorithm, this integrated project coordinate a set of peripheral devices, including simulating light source, simulation of photovoltaic panels, light detection circuit, A/D conversion circuit based on ADS1118, lcd displaying module, keyboard, full bridge PWM drive circuit and DC motor.

Keyword: MSP430, PID control, light tracking, A/D conversion

一、项目方案分析

光伏发电跟踪器由光敏检测电路、ADS1118 串口 A/D 转换电路、单片机（MSP430F5438）电路、液晶显示与键盘电路、DRV8412 全桥 PWM 驱动电路以及模拟太阳能板等几部分组成，通过光敏检测电路的感应可实现对光源的自动跟踪。整个系统的结构示意图如下：

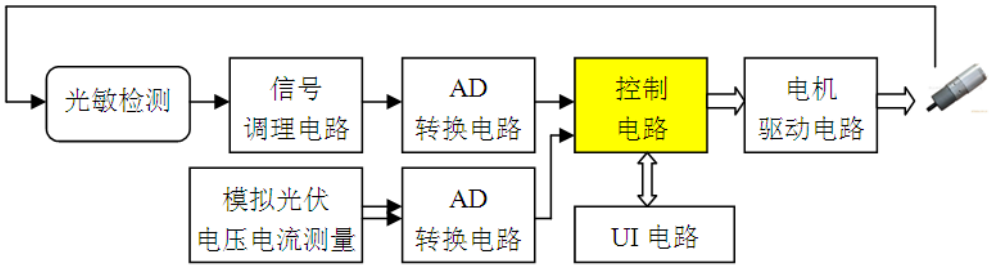


图 1 模拟光伏发电跟踪器系统结构示意图

自动跟踪光伏发电模型装置如图 2 所示，装置由模拟光源、直流减速电机、光敏检测电路和模拟太阳能板组成。模拟光源为 5 个 LED 灯（ LED1-LED5），通过装置面板上的 5 个开关 SW1-SW5 控制，通过顺次切换开关来模拟太阳光的移动。模拟光源的电源由接线端子上的+12V 和 GND 两端子提供。直流减速电机位于装置面板的下方，用来调节光伏板（模拟太阳能板）的偏转方向，直流减速电机的驱动电源由接线端子上的 Motor+和 Motor-两端子接入，电源为正时电机正转，电源为负时电机反转。

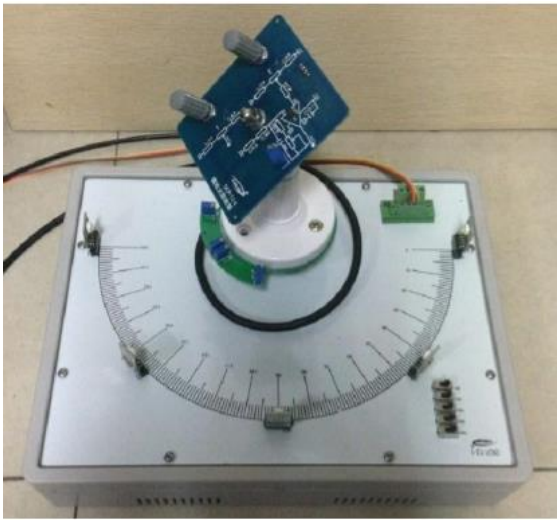


图 2 自动跟踪光伏发电装置

光敏检测电路位于电机上方的圆形线路板上，其原理图如图 3 所示。

光敏电阻位于线路板的下方，其中 R4、R5 并列朝前，R1 和 R2 各偏转 45° 布置，为了减小背景光照的影响，测光电路采用差分电路的形式，R1、R4 和 R2、R5 各构成一个差分电路。模拟太阳能板安装在白色支架的上方，其原理图如图 4 所示。电位器 RW1（已给定标称值为 1KΩ 电位器）用于模拟光伏电池的输出电压，要求可调范围 0～2V 左右（对应于实际的光伏电池的电压范围是 0～100V），电位器 RW2（已给定标称值为 470Ω 电位器）用于模拟光伏电池的输出电流，RW2 的可调电压范围要求 0～0.25V 左右，运放的输出范围要求 0～2V （对应于实际的光伏电池的电流范围是 0～2A）。

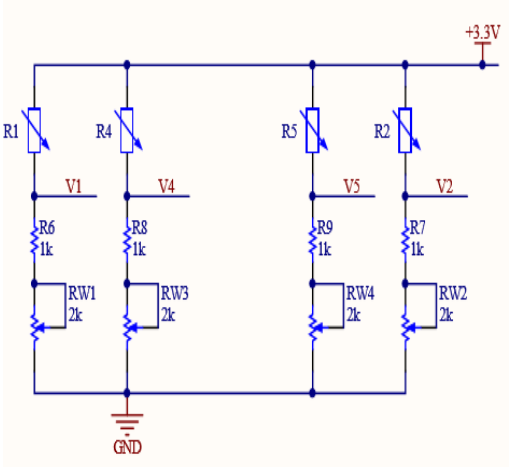


图 3 光敏检测电路原理图

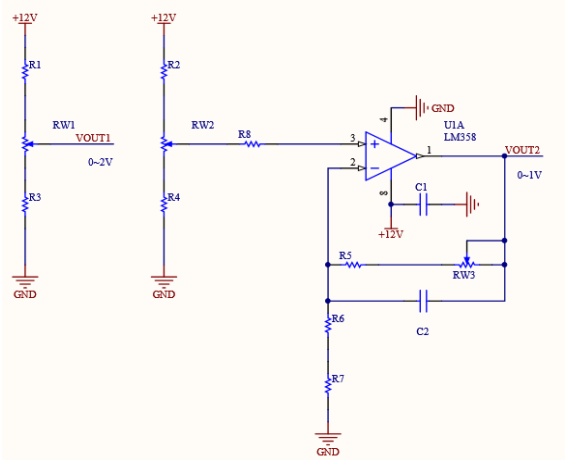


图 4 模拟太阳能板原理图

光敏检测的 AD 转换由片外 16 位 ADC 芯片 ADS1118 实现，模拟太阳能板的 AD 转换则由片内的 AD 转换模块实现。直流电机驱动模块由控制芯片产生 PWM 波形，控制电机驱动芯片 drv8412 实现。

二、项目具体要求

光伏发电跟踪器要求实现对自动跟踪光伏发电装置的控制。具体要求如下：

通过光伏发电跟踪器前面板的按键设置相关参数，可以实现光伏电池瞬时功率测量，发电量测量，太阳光模拟跟踪功能。按下模式选择按键，可以切换发电模式和跟踪模式。开机时，默认状态为发电模式，按下模式选择按键，进入跟踪模式；再次按下该按键，进入发电

模式。

发电模式下，要求液晶显示屏上实时显示系统时钟、瞬时功率和发电量，并且能够利用键盘对时钟参数进行实时修改，修改时有闪烁的光标提示，如图 5 所示。

跟踪模式下，发电装置能够根据光源位置的变化进行跟踪，同时液晶显示屏上面实时显示跟踪时间如图 6 所示。

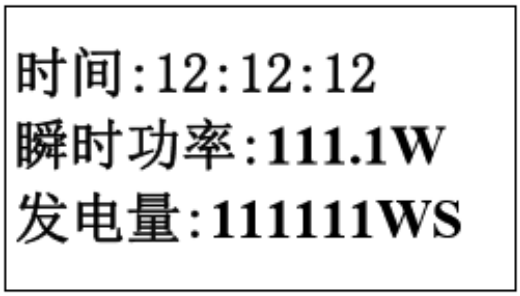


图 5 发电模式显示信息示意图

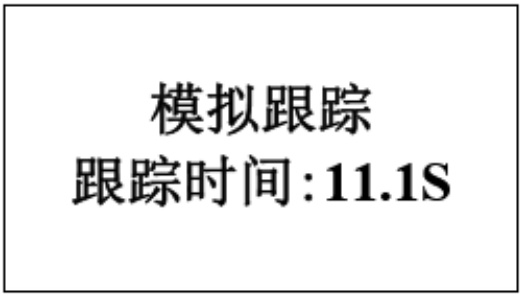
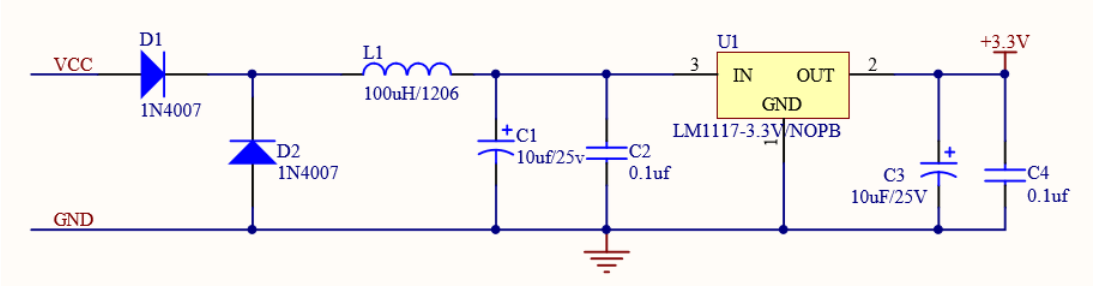


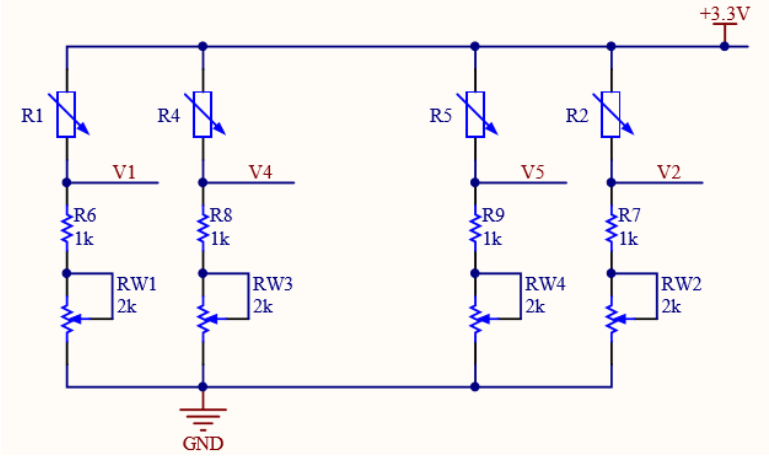
图 6 跟踪模式显示信息示意图

三、硬件电路分析

1. 光电检测电路

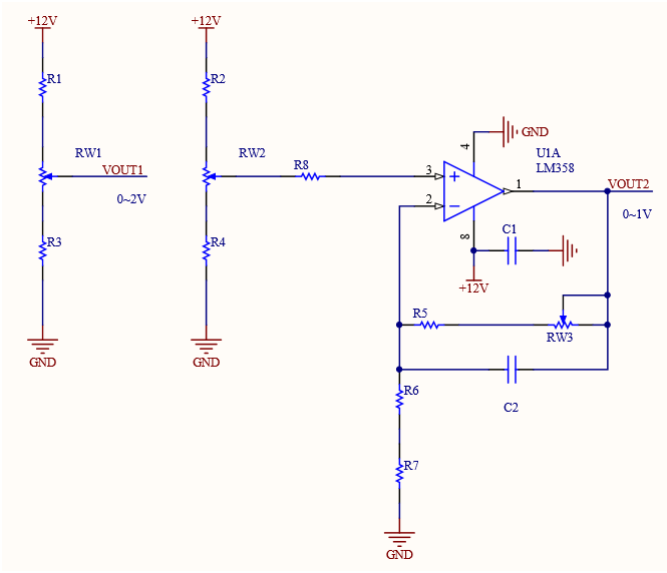


上图为光电检测电路的电源部分，将输入的 12V 电压转换为工作电压 3.3V。二极管 D1、D2 起电路保护作用，L1、C1、C2 改善输入电压纹波，LM1117 模块实现 3.3V 的降压，C3、C4 改善输出电压纹波。



上图为光强信号检测部分，R1、R2、R4、R5 为四个光敏电阻，光强的改变会改变电压 V1、V2、V4、V5 的大小。四个电压信号经差分放大电路后送入 AD 转换，再将数字信号送入单片机。RW1-RW4 用于调试光强检测效果。

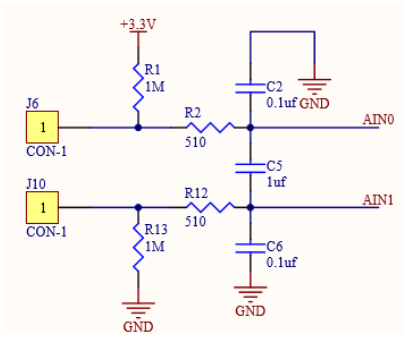
2. 模拟太阳能板

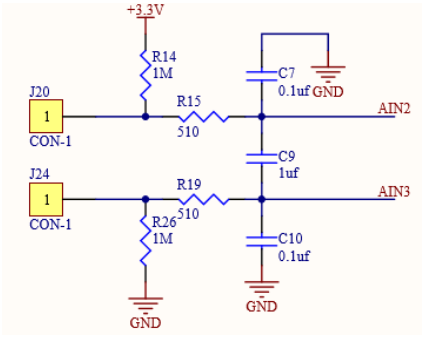


模拟太阳能板安装在白色支架的上方，原理图如上图所示。电位器 RW1 用于模拟光伏电池的输出电压，可调范围为 0-2V(对应光伏电池的电压范围是 0-100V)，电位器 RW2 用于模拟光伏电池的输出电流，RW2 的可调电压范围为 0-0.25V 左右，运放的输出范围为 0-2V(对应实际的光伏电池的电流范围)。VOUT1 和 VOUT2 直接送入单片机的八路 AD 输入通道 JA 口。

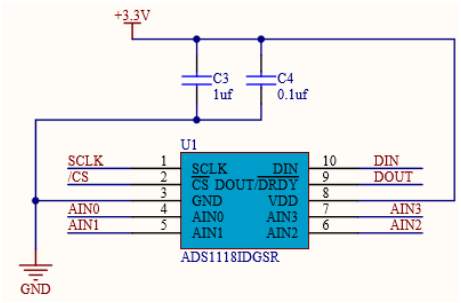
3. DCP-208 串口 AD 转换电路

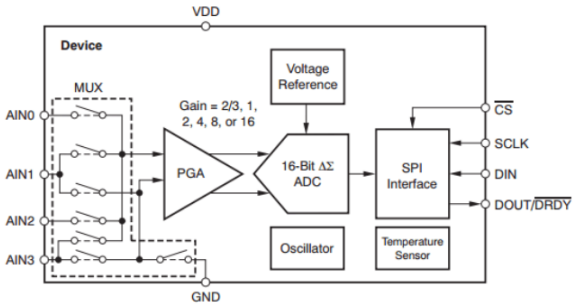
此模块主要由模拟信号输入、AD1118 及外围电路、偏置电路、数字信号输出电路组成，AD1118 功能脚及数字输出由 PMOD 口引出。





上图为偏置电路，将光电检测电路的四个电压信号偏置后送入 AD1118 芯片。





如上图为 AD1118 芯片的内部电路和外围接口。AD1118 输入端由一个复用器组成，具有四个单端信号输入端或两个差分输入端，分为为 AIN0-AIN4。SCLK 为时钟输入，CS/为片选信号，DOUT/DRDY 为串行数据输出，DIN 为串行数据输入，这四个信号端通过接线排 PMOD 送入单片机。

4. DCP-218H 桥 PWM 输出电路

控制信号输入：JG、JF

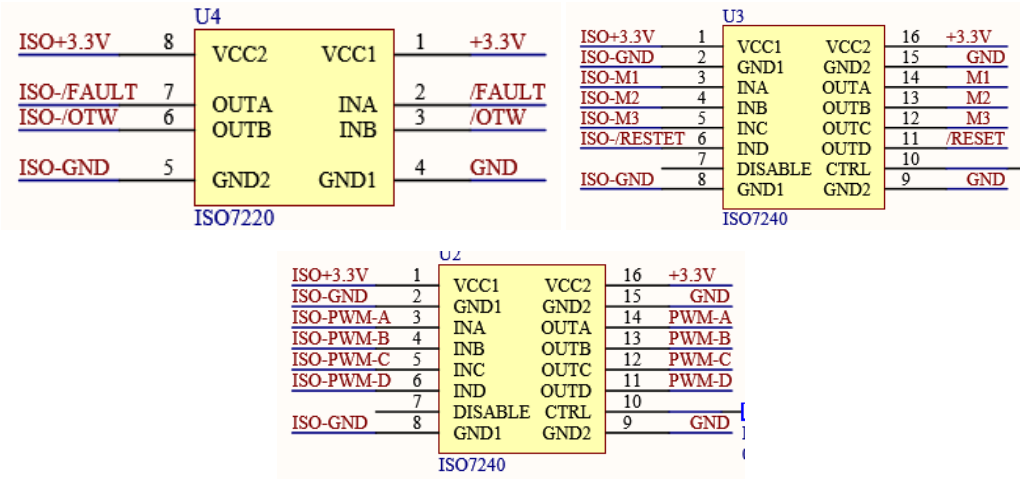
输出：OUTA、OUTB、OUTC、OUTD

实现功能：通过单片机相应的引脚输出控制 PWM 的产生情况，从而控制电机的转动情况。

工作原理：

单片机的相应控制引脚通过接线排 JG、JF 引出，连接到 ISO7240（ISO7220）的输入侧，作为其输入信号，ISO7240 模块能实现控制侧电路与高压负载侧电路的隔离，其输出信号接到 drv8412 的相应引脚，控制其工作状态。其中改变 M1、M2、M3 的值可以实现对 PWM 工作模式的选择，改变 PWMA、PWMB、PWMC、PWMD 的值可实现输出 OUTA、OUTB、OUTC、OUTD 的值，进而实现控制电机的正转、反转等状态。

工作所需的电压 PVDD、GVDD 以及数字信号逻辑电压则由三端稳压模块 LM317 和 LM1117-3.3V 提供，这样能实现高压负载侧和数字控制电路侧的完全隔离。



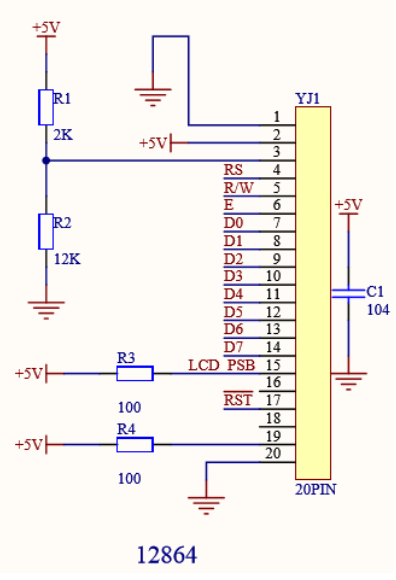
5. 液晶显示和键盘电路

控制信号输入：JC、JD、JE（PMOD 接口）

输出：无

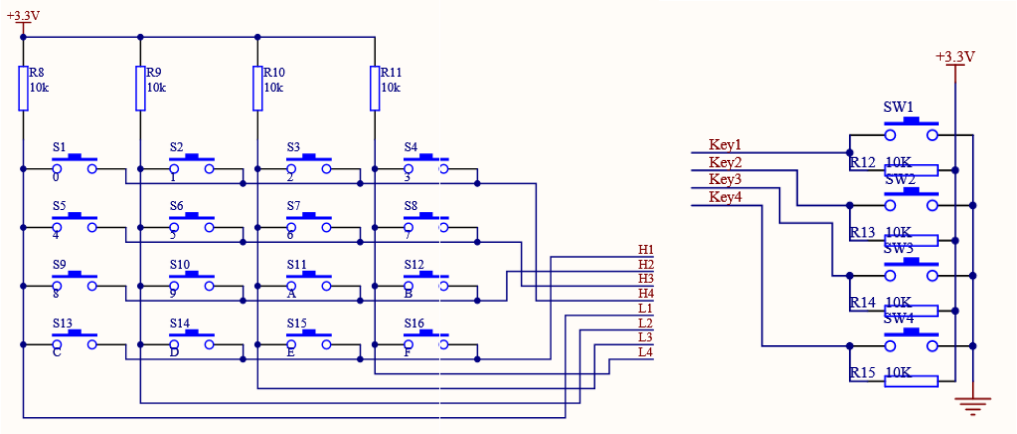
- 液晶显示电路如图：

  D0-D7 为 8 位并行输入数据，单片机将需要显示的数据通过相应的引脚以并行数据 D0-D7 的形式输出至此。RS 指示输入的 8 位数据是指令还是需要显示的数据，R/W 则是读写模式控制信号，E 是使能信号，RST 是复位端口。由此根据 LCD 的工作原理，通过单片机对相应的输出引脚进行合适的赋值就能够使 LCD 上面显示相应的内容。



- 键盘电路

  如图，键盘的硬件电路采用矩阵形式，键盘对应的行和列都与单片机相应的输入输出口连接，在进行键盘扫描的时候可以采用整行整列扫描的方法：先将行对应的 IO 口设为下拉电阻输出状态，列对应的 IO 口设为输入状态。对四列的输入进行检测，若某列出现逻辑 0，则该列有按键被按下；然后将列对应的 IO 口设为下拉电阻输出状态，行对应的 IO 口设为输入状态。对四行的输入进行检测，若某行出现逻辑 0，则该行有按键被按下，这样进行一次完整的扫描就能够确定摁键的位置，然后进行去抖处理，可以进行摁键中断的处理。



额外扩展的四个按键则可以直接以查询的方式进行扫描，对应的引脚设为输入状态，没有按键摁下的时候，为逻辑 1，有按键摁

四、软件程序分析

由于整个系统有很清晰的结构，所以实现整个系统的功能时可以分模块编写相应的驱动程序，然后再进行各模块的整合和整个系统的控制调试。

1. LCD 显示驱动实现

本项目中 LCD 液晶显示需用的是 12864 型号，在编写之前阅读其用户手册，理解其内部显示的原理和驱动方法，以及相应寄存器的使用方法，然后编写程序进行驱动控制。Lcd 显示里面写命令和写数据是重复利用的部分，所以利用函数实现，这样只需调用，初始化部分也用函数实现。由于用到相应的输入输出口比较多，因此每个模块应该进行相应的定义，这样方便程序的编写和阅读。

定义部分：

```
    #define LCD_DataIn    P8DIR=0x00    //数据口方向设置为输入
    #define LCD_DataOut   P8DIR=0xff    //数据口方向设置为输出
    #define LCD2MCU_Data  P8IN
    #define MCU2LCD_Data  P8OUT
    #define LCD_CMDOut    P3DIR|=(BIT0+BIT4+BIT5)     //P3 口的相应三位设置为
输出
    #define LCD_RS_L          P3OUT&=~BIT0          //RS = P3.0
    #define LCD_RS_H          P3OUT|=BIT0
    #define LCD_RW_L          P3OUT&=~BIT5           //RW = P3.5
    #define LCD_RW_H          P3OUT|=BIT5
    #define LCD_EN_L          P3OUT&=~BIT4           //EN = P3.4
    #define LCD_EN_H          P3OUT|=BIT4
```

```
/*****************************************
函数名称：Ini_Lcd
功    能：初始化液晶模块
参    数：无
返回值  ：无
*****************************************/
```

```
    void Ini_Lcd(void)
    {
        LCD_CMDOut;    //液晶控制端口设置为输出
        Write_Cmd(0x30);   //基本指令集
        Delay_1ms();
        Write_Cmd(0x02);   // 地址归位
        Delay_1ms();
        Write_Cmd(0x0c);   //整体显示打开,游标关闭
        Delay_1ms();
        Write_Cmd(0x01);   //清除显示
        Delay_1ms();
        Write_Cmd(0x06);   //游标右移
        Delay_1ms();
        Write_Cmd(0x80);   //设定显示的起始地址
    }
```

```
/*****************************************
函数名称：Write_Cmd
功    能：向液晶中写控制命令
参    数：cmd--控制命令
返回值  ：无
*****************************************/
```

```
    void Write_Cmd(uchar cmd)
    {
        uchar lcdtemp = 0;
        LCD_RS_L;           //输入命令格式
        LCD_RW_H;           //从芯片读数据格式
        LCD_DataIn;         //单片机设为接收数据状态
        do {                      //判忙
            LCD_EN_H;           //使能芯片
            _NOP();
            lcdtemp = LCD2MCU_Data;  //单片机接收数据
            LCD_EN_L;           //使能端无效
        }
        while(lcdtemp & 0x80);
        LCD_DataOut;       //单片机设为输出数据状态
        LCD_RW_L;          //向芯片写数据格式
        MCU2LCD_Data = cmd;   //单片机输出数据
        LCD_EN_H;            //使能端有效
        _NOP();
        LCD_EN_L;            //使能端无效
    }
```

```
/*****************************************
函数名称：Write_Data
```

功　　能：向液晶中写显示数据
参　　数：dat--显示数据
返回值　：无
***/

```
void  Write_Data(uchar dat)
{
    uchar lcdtemp = 0;
    LCD_RS_L;               //输入命令格式
    LCD_RW_H;                //读数据格式
    LCD_DataIn;               //单片机设为接收数据状态
    do                        //判忙
    {
        LCD_EN_H;           //芯片使能端有效
        _NOP();
        lcdtemp = LCD2MCU_Data;  //读数据
        LCD_EN_L;              //芯片使能端无效
    }
    while(lcdtemp & 0x80);
    LCD_DataOut;         //单片机输出数据
    LCD_RS_H;            //数据格式
    LCD_RW_L;            //向芯片写数据格式
    MCU2LCD_Data = dat;     //写数据
    LCD_EN_H;             //使能端有效
    _NOP();
    LCD_EN_L;              //使能端无效
}
```

2. 键盘扫描实现

本项目中用到的键盘硬件电路连接采用矩阵式点阵的方式，驱动程序的基本思路就是设置好相应的 IO 口的输入输出状态（行输出，列输入），然后逐行输出低电平，检测列输入的数据，若有低电平则进行延时去抖后确定是否有键按下。
相应代码如下：

```
#define key_H_OUT     P7DIR|=0xF0   //键盘四行设为输出状态
#define key_L_IN      P2DIR=0x00   //键盘四列设为输入状态
```

```
unsigned int  key_event(void)
{
    unsigned int key_num;    //当前扫描得到的按键的位置
static unsigned int key_past;    //局部静态变量存放上次按键的位置
    unsigned int key;             //扫描得到的按键值
key_H_OUT;                //键盘四行设为输出状态
key_L_IN;                 //键盘四列设为输入状态
key_num=0;                //初始化扫描键码值
```

然后逐行输出低电平，再进行逐列扫描，只以第一行为例：

```
    P7OUT=0x7F;                 //第一行输出低电平
    if(0xE0==(P2IN&0xF0))       //判断第一列是否有按键按下
    {
        _delay_cycles(1000);          //延时 1ms，去抖动
        if(0xE0==(P2IN&0xF0)){
        key_num=1;
        }}
    if(0xD0==(P2IN&0xF0))        //判断第二列是否有按键按下
        {
        _delay_cycles(1000);
        if(0xD0==(P2IN&0xF0)){
            key_num=2;
        }}
    if(0xB0==(P2IN&0xF0))    //判断第三列是否有按键按下
        {
        _delay_cycles(1000);
        if(0xB0==(P2IN&0xF0)){
            key_num=3;
        }}
    if(0x70==(P2IN&0xF0))         //判断第四列是否有按键按下
        {
        _delay_cycles(1000);
        if(0x70==(P2IN&0xF0)){
            key_num=4;
        }}
```

扫描完之后需要对键码进行判断：

```
    //判断扫描得到摁键的情况
          if(key_num!=key_past)      //如果扫描得到的按键和上次不相同，则有一个
新的按键摁下
          {
           key_past=key_num;
          Key=key_num;
               }
          else{
              key=0;               //如果是同一次按键，则 keycode 值无效
           } }
          return key;
}
```

3. 片外 ADC 芯片 ADS1118 驱动实现

外置 AD 转换模块主要由差分信号输入放大部分、A/D 芯片 ADS1118 和数字信号到核心板 SPI 接口组成。编写程序实现 ADS1118 的单次转换，并将数据通过 SPI 口传到核心板。主函数中调用到的函数包括初始化函数和 AD 动作函数（内部引用 SPI 通讯函数、AD 转换开关

控制函数、AD 转换结果读取函数）。

//ADS1118 初始化函数

```
void ADS1118A_Init(void){
    P1OUT |= 0x02;                          // Set P1.1 for CS
    P1DIR |= 0x02;                          // Set P1.1 to output direction
    P3SEL |= 0x80;                          // P3.7 option select
    P5SEL |= 0x30;                          // P5.4,5 option select
    UCB1CTL1 |= UCSWRST;                    // **Put state machine in reset**
    UCB1CTL0 |= UCMST+UCSYNC+UCMSB;         // 3-pin, 8-bit SPI master
                                            // Clock polarity high, MSB
    UCB1CTL1 |= UCSSEL_2;                   // SMCLK
    UCB1BR0 = 0x05;                         // /2
    UCB1BR1 = 0;                            //
    UCB1CTL1 &= ~UCSWRST;                   // **Initialize USCI state machine**
    __delay_cycles(100);                    // Wait for slave to initialize
}
```

//ADS1118 和核心板间通信函数

```
signed int WriteSPI(unsigned int config, unsigned char mode){
    signed int msb;
    unsigned int temp;                      //volatile signed int dummy;

    temp = config;
    if (mode==1) temp = 0;                  //temp = (config | 0x8000)&(0xfffD);

    while(!(UCB1IFG&UCTXIFG));
    UCB1TXBUF = (temp >> 8 );               // Write MSB of Config
    while(!(UCB1IFG&UCRXIFG));
    msb = UCB1RXBUF;                        // Read MSB of Data

    while(!(UCB1IFG&UCTXIFG));
    UCB1TXBUF = (temp & 0xff);              // Write LSB of Config
    while(!(UCB1IFG&UCRXIFG));
    msb = (msb << 8) | UCB1RXBUF;           // Read LSB of Data

    while(!(UCB1IFG&UCTXIFG));
    UCB1TXBUF = (temp >> 8 );               // Write MSB of Config
    while(!(UCB1IFG&UCRXIFG));
    dummy = UCB1RXBUF;                      // Read MSB of Config

    while(!(UCB1IFG&UCTXIFG));
    UCB1TXBUF= (temp & 0xff);               // Write LSB of Config
    while(!(UCB1IFG&UCRXIFG));
    dummy = (dummy <<8) | UCB1RXBUF;        // Read LSB of Config
```

```
    __delay_cycles(100);
    return msb;
}
```

//ADS 开关控制函数

```
void ADS_Config(signed int temp_config_value){
    signed int Config_Value;
    Config_Value = temp_config_value;
    //Config_Value = 0x8583;                    // Initial Config Register
    // ADS1118 configuration AIN0/AIN1, FS=+/-2.048, DR=128sps, PULLUP on DOUT
    P1OUT &=~ 0x02;                             // Set CS low
    __delay_cycles(100);                        // Wait for slave to initialize
    WriteSPI(Config_Value,0);                   // Write configuration to ADS1118
    __delay_cycles(100);                        // Wait for slave to initialize
    P1OUT |= 0x02;                              // Set CS high
}
```

//AD 转换结果读取函数

```
int ADS_Read(void){
    unsigned int Data, Config_Value;
    //Config_Value = 0x058B;
    Config_Value = 0;
    // ADS1118 configuration AIN0/AIN1, FS=+/-2.048, DR=128sps, PULLUP on DOUT
    P1OUT &=~ 0x02;                             // Set CS low
    Data = WriteSPI(Config_Value,1);            // Read data from ADS1118
    P1OUT |= 0x02;                              // Set CS high
    return Data;
}
```

//ADS1118 动作函数

```
float ADS1118A_Act(){
    volatile int ADC_Result;

    ADS_Config(0xB583);                          //config ch2
    ADC_Result = ADS_Read();                    // Read data from ch2,the last time result
    Voltage_ch1 = ADC_Result*1.0/32768*2.048;
    __delay_cycles(10000);                      //need to wait time until end of convert

    ADS_Config(0x8583);                          //config ch1
    ADC_Result = ADS_Read();                    // Read data from ch1,the last time result
    Voltage_ch2 = ADC_Result*1.0/32768*2.048;          __delay_cycles(10000);
    return (Voltage_ch1-Voltage_ch2);
}
```

4. 片内 AD 转换驱动实现

为达到更高的 AD 转换精度，我们选用单片机内置 12 位 AD 转换模块。转换公式为 $N_{ADC} = 4095 \times \frac{Vin - V_{R-}}{V_{R+} - V_{R-}}$，这里我们另 VR+=Vref（2.5V），VR-=AVss（GND）。程序编写涉及转换时钟

源选择、输入通道选择、参考电压选择、采用保持时间选择、转换模式选择、转换开启个停止等。以下为具体的初始化程序和单次 ADC12 采样工作函数。

// ADC12 采用初始化函数

```
void ADC_Init(){
    ADC12CTL0 &= ~ADC12ENC;                          //关闭采样使能
    P6SEL |= BIT0 + BIT1;                            //选择 P6.0,P6.1 为模拟信号的输入端
//开启 ADC12，自动循环采样模式，设置采样保持时间为 16 个机器周期
    ADC12CTL0 |= ADC12ON + ADC12MSC + ADC12SHT0_2;
//使能电压管理，选择参考电压源，内部参考电压选择 2.5V，打开内部参考电压
    ADC12CTL0 |= REFMSTR + ADC12SREF_1 + ADC12REF2_5V + ADC12REFON;
    ADC12CTL1 |= ADC12SHP + ADC12CONSEQ_1;   //使用采样定时器，单次顺序采样
    ADC12MCTL0 |= ADC12INCH_0;                       //选择采样通道 A0
    ADC12MCTL1 |= ADC12INCH_1+ADC12EOS;      //选择采样通道 A1，设置采样顺序结点
    ADC12IE = 0x02;                                  //使能 ADC12IFG.1
    ADC12CTL0 |= ADC12ENC;                           //使能 AD 转换//ADC12 模块初始化函数
```

//ADC 中断函数，当两个通道相继采样完成后悔触发中断

```
#pragma vector=ADC12_VECTOR
__interrupt void ADC12ISR (void) {
    Vout1 = ADC12MEM0;                               //读取两个通道 AD 转换结果
    Vout2 = ADC12MEM1;
    Pre_Power = (1.0*Vout1*Vout2)/(25*625);          //将 AD 转换结果计算为发电功率
}
```

//开始一次内部 AD 转换语句: ADC12CTL0 |= ADC12SC;

5. PWM 波产生与控制程序实现

我们采用 MSP430F5438A 的 16 位定时器 B 来生成 PWM 波。定时器模式设为增计数模式；时钟选用 32kHz，不分频；波形周期由 TTCCR0 设置，为 100 个机器周期；波形由 TB0.3 和 TB0.4 口输出；OUTMODE 选择模式 7。初始化程序和电机动作执行程序分别如下：

//初始化电机

```
void Motor_Init(){
    P4DIR |= 0X18;                          //选择 TB0.3&TB.4 口输出
    P4SEL |= 0X18;
    P7DIR |= BIT3;                          //低电平复位 H 桥 PWM 波 dvr8412 模块
    P7OUT &=~BIT3;
    P9DIR = 0x0E;                           //  设置 M0\M1\M2 为第一种模式
    P9OUT &=~0x0E;
    P7DIR &=~BIT2;                          //  错误保护标志/FAULT
    P2DIR &=~BIT1;                          //  过热保护标志位/OTW
    P7OUT |=BIT3;                           //将/RESET 拉高防止重复复位
    TBCCR0 = 100;                           //设定周期为 100 个机器周期
    TBCTL = TBSSEL_1 + MC_1 + TBCLR;        //选用 32kHz，不分频，增计数
    TBCCTL3 |= OUTMOD_7;                    //选择 TB0.3 为模式 7
    TBCCTL4 |= OUTMOD_7;                    //选择 TB0.4 为模式 7
    return;
}
```

//电机正转、反转或停止

```
void Motor_Act(float Dir){          //Dir 为光电检测模块两差模通道 ch1 和 ch2 的插值转化后的结果
    if(Dir==0){                     //若返回值 0，电机停止运转
        TBCCR3 = 0;    TBCCR4 = 0;
    }
    else if(Dir>0){                 //若返回值大于 0，电机正转
        if(Dir>70)
            Dir=70;
        TBCCR3 = 10+Dir;   TBCCR4 = 0;
    }
    else {                          //若返回值小于 0，电机反转
        if(Dir<-70)
            Dir=-70;
        TBCCR3 = 0;    TBCCR4 = 10-Dir;
    }
    return;
}
```

6. 光伏发电跟踪系统实现

系统主函数由初始化、各模块动作响应和有 0.01s（最初开环控制为 0.1s，因 PID 控制调试需要改为 0.01s）的定时中断三部分组成。

```
//主函数
void main(void) {
    WDTCTL = WDTPW + WDTHOLD;       //关闭看门狗
    Motor_Init();                   //初始化电机
    Ini_Lcd();                      //初始化 cry12864 显示模块
    ADC_Init();                     //初始化 Internal_ADC
    ADS1118A_Init();                //初始化 ADS1118A 模块
    TA0CTL = TASSEL_1 + MC_1;       //ACLK UP_MODE CLEAR
    TA0CCTL0 = CCIE;                //使能 CCR0 中断
    TA0CCR0 = 320;                  //定时 0.01s
    _enable_interrupts();           //使能总中断
    while(1){
        Display();                  //LCD 屏幕显示
        key_change();               //键盘扫描
        ADC12CTL0 |= ADC12SC;       //开始一次内部 AD 转换;
        if(Mode){                   //判断处于发电模式还是跟踪模式
            temp=ADS1118A_Act();    //读取光电检测结果并通过 ADS1118 转化
            if(temp>=-0.03&&temp<=0.03)   //当光电检测结果小于偏差容限时，电机停止
                Motor_Act(0);
            else if(temp>=0.25)     //PD 调节阀门，光电检测结果和目标值相差太多，电机全速运行
                Motor_Act(80);
            else if(temp<=-0.25)
```

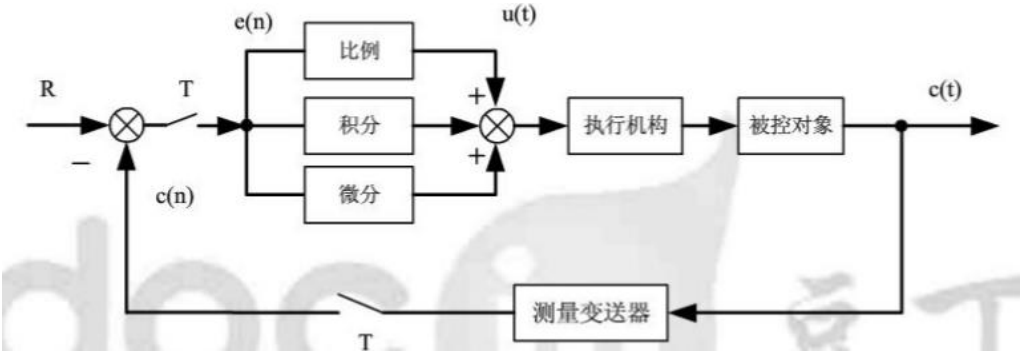
```
                Motor_Act(-80);
            else
                Motor_Act(Motor_Dir);       //当光电检测结果接近目标值时启动 PD 控制
        }
    }
}
```

//0.01s 定时中断

```
#pragma vector=TIMER0_A0_VECTOR
__interrupt void TIMER0_A0(void){
    clock_change();                 //改变时钟显示值、并更改发电功率、发电总量显示值
}
```

五、PID 闭环调试过程

为达到对电机跟踪快速、精准和稳定的控制，我们以 PID 控制原理对开环程序系统进行改进，根据被控变量（光电检测两差模通道的差值）实时值与目标值之间的偏差以及偏差的变化率等简单参数，通过工程方法对比例系数 Kp、积分时间 Ti、微分时间 Td 三个参数进行调整。



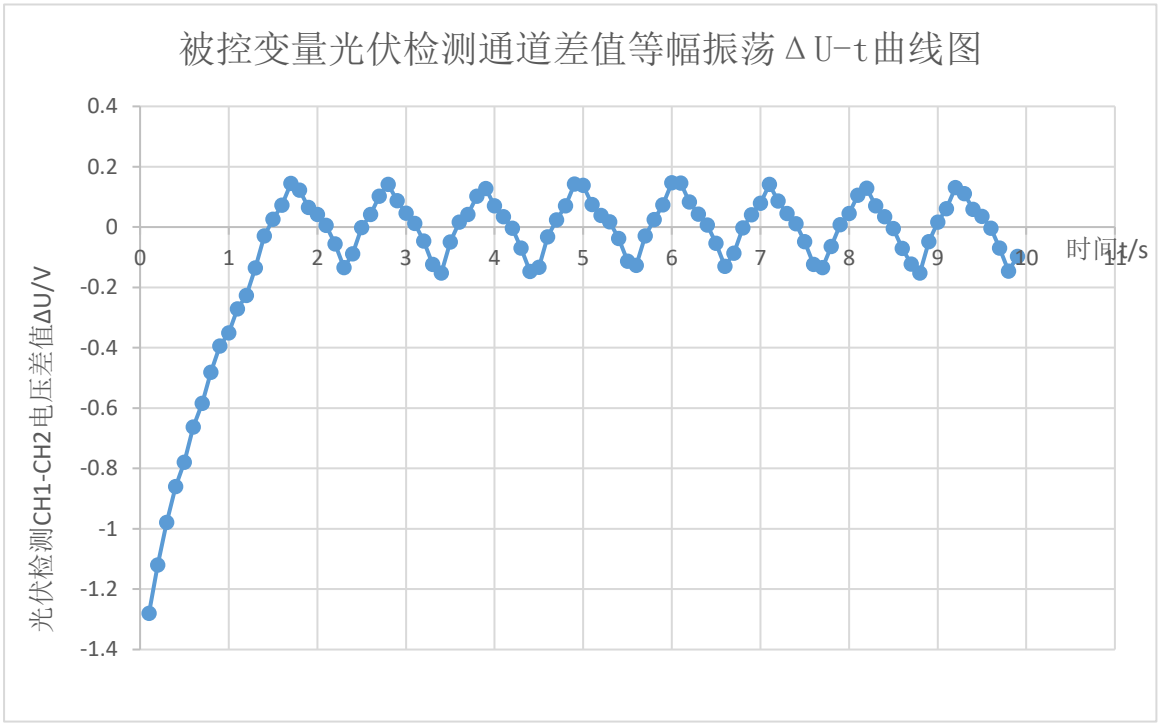
PID 控制原理图

PID 控制算法式：

$$u(n) = K_P \left\{ e(n) + \frac{T}{T_I} \sum_{i=0}^{n} e(i) + \frac{T_D}{T} [e(n) - e(n-1)] \right\} + u_0$$

为测得比例系数 Kp、积分时间 Ti、微分时间 Td，我们采用临界比例度法，在系统闭环的情况下，只保留比例环节，给系统施加一个阶跃输入（即突然开启一盏灯），并开辟一段数组空间以 0.1s 的间隔记录被控变量△U 的变化情况，若△U 的过渡过程无振荡或呈衰减振荡，则继续增大 Kp 值；若△U 的过渡过程呈发散振荡，则应减小 Kp 值，直到调至某一 Kp 值，过渡过程出现等幅振荡为止。

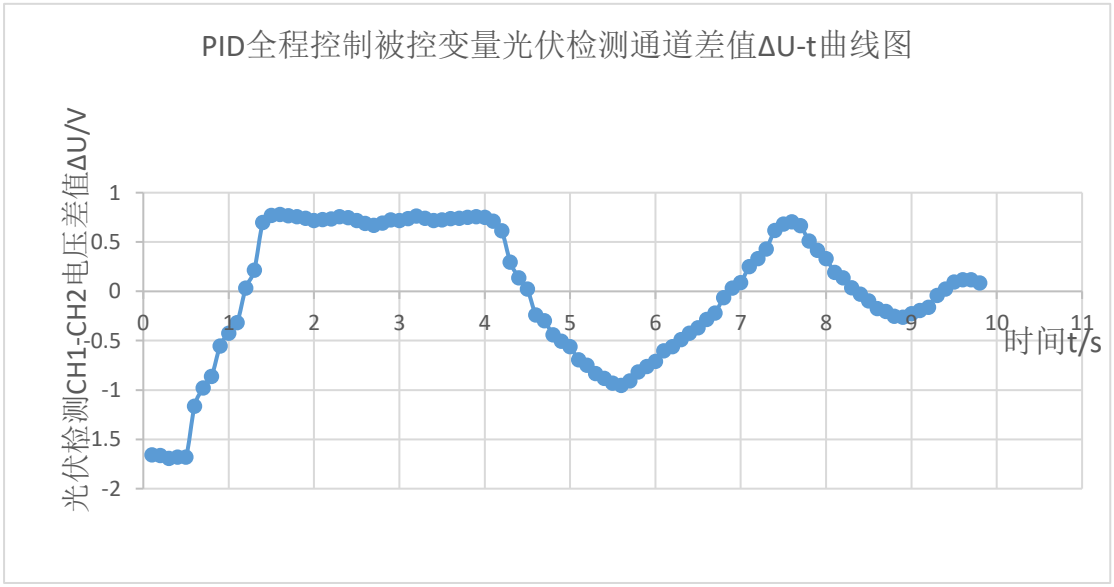
实验中我们设置 Kp 初值为 43，得到一条较理想的等幅振荡曲线：



根据曲线临界周期 Tc/s 得到 Kp、Tt 和 Td 值如下:

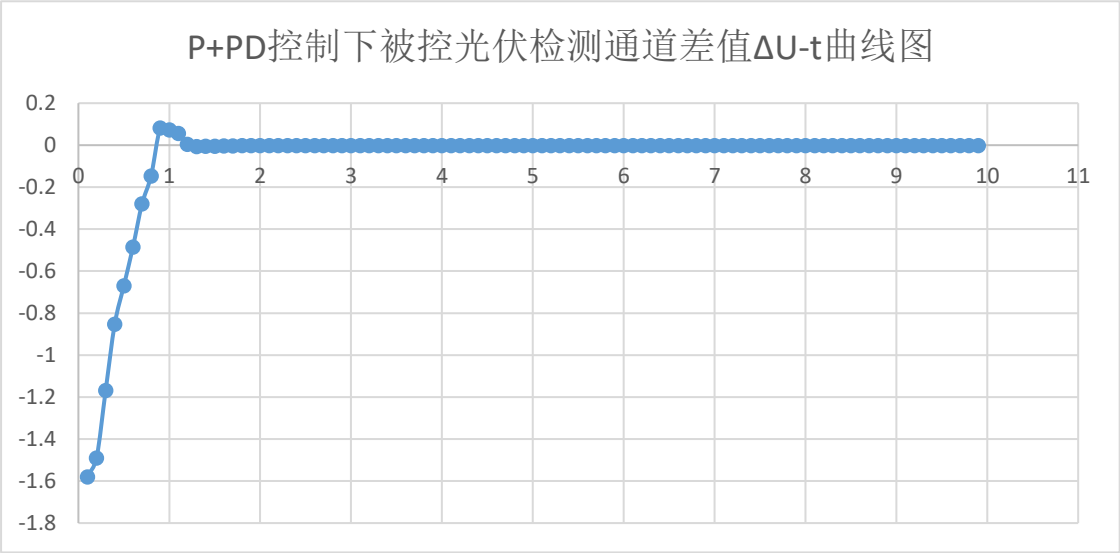
临界周期 Tc/s	Kp=0.6Kc	Ti=0.5Tc(s)	Td=0.12Tc(s)
1.06666667	25.8	0.533333333	0.128

根据以上参数值，更改程序，得到 PID 控制初步效果曲线如下：



可知，此时虽然电机振荡衰减速度很快，但首次振荡过调幅度很大，分析原因是由于积分环节造成的过调。于是我们增大积分比例 Ti，测试后曲线仍后较大过调量。于是我们去除积分环节，更改程序再次调试，肉眼观察电机转动，发现虽然此时电机过调振荡幅度很小，但电机转动速度不快。综合考虑后我们决定结合电机全速运转和 PD 调节，让电机最初达到较快速度，当被控变量接近目标值再加入 PD 调节。

在电机速度既快、过调量也较小的情况下，反复调节比例系数 Kp、微分系数 Td、PD 控制阀门和光伏检测通道差值ΔU 误差容限，使得控制电机不出现振荡或振荡幅度很小且衰减速度很快。最终被控变量△U-t 曲线如下：



六、项目中遇到的问题及解决办法

我们在软件编写过程遇到的问题不少，但基本是小问题，算法没有出现致命性错误，因而下面我们只选取几个问题作为代表。

1. **问题：**“整行整列”法进行键盘扫描失败

   原因分析：之前单片机课程中键盘扫描是利用“整行整列”法——先确定按键所在的列，在确定按键所在的行，进行的，但是这次实验中，采用这种方法需要在一次扫描中连续改变相应 IO 口的输入输出状态。进行单步调试时发现在程序中将 IO 口从输出状态改为输入状态后，相应的 PxIN 的内容不变，导致键盘扫描结果出错。

   解决办法：将扫描方法改为“逐行逐列”法，行对应的 IO 口都设定为输出状态，列对应的 IO 都设定为输入状态，然后逐行输出低电平，逐列检测输入值，从而确定按键位置。

2. **问题：**发电总量一旦超过 10000WS,程序就跑飞

   原因分析：Lcd 模块程序中存在下面一段代码：

```
while(Power_temp>=beishu){
    beishu *= 10;
}
```

   可知当 Power_temp 大于 10000 时，程序要跳出此循环就必须至少为 100000，而 MSP430 为 16 位处理器，int 数据类型的变量最大只能为 65535，这时程序就跑飞了。

   解决办法：将 beishu 的数据类型改为 long int。

3. **问题：**发电模式下倒计时有误，例如 01：30：21--01：30：10--01：30：19，即出现提前借位。

   原因分析：倒计时时钟的运行原理为，根据上一秒时钟各位数值判断下一秒时钟各位应有数值。例如上一秒时刻为 01：30：21，下一秒应为 01：30：20，即更改秒低位，如果将秒高位判断语句放在秒低位更改语句后，则此时秒高位处理判断语句中的秒低位为更改后的数值而不是上一秒时刻的数值。

   解决办法：调整时钟各位处理语句的顺序，或设置变量储存上一秒时刻各位的数值。

4. **问题：**在系统开环正常运行的基础上，加上闭环 P 比例调节，更改程序后程序跑飞

   原因分析：开辟用于记录被控变量光伏检测通道差值 ΔU 的数组长度太长，超出 MSP430 的 RAM 空间。

   解决办法：增大记录时间间隔为 0.1s，数组长度设为 100。

七、结束语

在整个项目中，从整个系统结构原理解剖、模块程序编写、控制系统初步建立、PID 控制调节到最后的实验报告撰写，我和队友既有明确分工，也能密切配合，大大提高了整个项目的完成效率。例如，在编写模块程序时，我和队友明确将初始化函数和执行机构动作函数分离，做好子函数的独立封装等。和其他组比较，我们**花时最少，PID 控制效果最好，收获也颇丰**。

该综合项目提高了我们的独立自主设计和调试软硬件综合系统的能力，更宝贵的是给我们日后进行进一步的学习或做竞赛积累了经验和灵感。例如该项目 PID 控制参数的确定过程只能通过不断更改程序、烧写程序、观察效果来实现，但如果我们做一些系统的拓展，加入能通过键盘输入更改 PID 控制的参数的功能，能极大提高调试速度。

最后，衷心感谢祁老师和张老师对我们的引导和帮助，感谢学院对爱迪生班学生的特殊培养！