目录

图目录

表目录

摘要

随着现代科技的不断进步，各个领域对测速系统的应用越来越多，同时也对测速精度的要求越来越高。从测速仪器是否与转轴接触又可分为接触式，非接触式，目前常用的是非接触测量，采用传感器为检测元件，读取到的信号通过微处理器来进行计算显示。

目前，常见的单片机有 51 系列和 MSP430 系列等。51 系列应用广泛、功能完备，但抗干扰能力不强；与上述相比，MSP430 系列具有功能强大、集成度高、抗干扰能力强、成本低、市场流通性大、能进入低功耗模式运行，保证产品的时间长、损耗低、精度高、稳定性佳等特点。

综上所述，本文将基于 MSP430G2553 单片机进行非接触式转速测量仪的设计。

关键字：转速测量；单片机；MSP430；光电传感器

Abstract

With the continuous progress of modern science and technology, the application of speed measurement system in various fields is increasing, and the requirement of speed measurement accuracy is also becoming higher and higher.

Whether the speed measuring instrument is in contact with the rotating shaft can be divided into contact type and non-contact type. Currently, non-contact measurement is commonly used. The sensor is used as the detection element, and the signal read is calculated and displayed by the microprocessor.

At present, the common SCM has 51 series and MSP430 series.Series 51 is widely used and has complete functions, but its anti-interference ability is not strong.Compared with the above, MSP430 series has powerful functions, high integration, strong anti-interference ability, low cost, large market liquidity, can enter the low-power mode of operation, to ensure the product's long time, low loss, high precision, good stability and other characteristics.

To sum up, this paper will be based on MSP430G2553 microcontroller for the design of non-contact speed measuring instrument.

Keywords: speed measurement; single chip microcomputer; MSP430; photoelectric sensor

非接触式转速测量表

1 设计题目

非接触式转速测量表。

2 设计目的

运用单片机原理及其应用等课程知识,根据题目要求进行软硬件系统的设计和调试，从而加深对传感器的理解，把学过的比较零碎的知识系统化，比较系统地学习开发单片机应用系统的基本步骤和基本方法，使应用知识能力、设计能力、调试能力以及报告撰写能力等有一定的提高。

3 设计内容及要求

设计一个用光电反射式传感器检测的非接触式转速测量表，检测范围 60-3000 转/分，精度±5%，显示分辨率 0.1 转/分。

4 系统总体结构

此次非接触式转速测量表包括：光电反射式传感器，显示模块，控制模块。

系统上电，光电反射式传感器的电源由 MSP430G2553 开发板上的 3.3V 电源提供。单片机的 1 个 IO 口捕获光电反射式传感器输出的方波周期，最后借助软件编程将转速显示在 128 段液晶显示屏上。系统总设计框图如图 4-1 下：

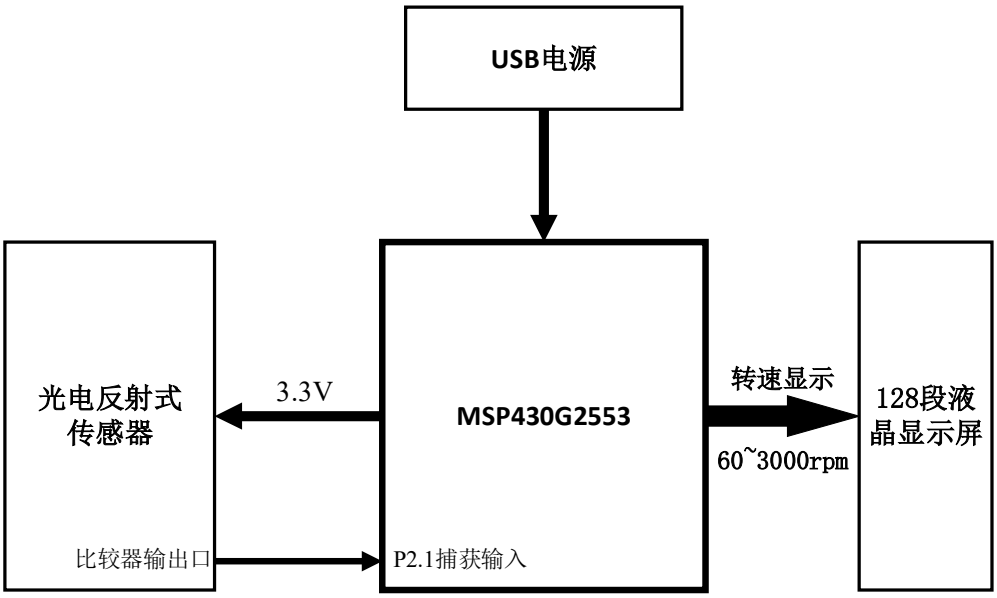

图 4-1 系统框图

5 硬件设计

5.1 元件选取

(一) MSP430G2553

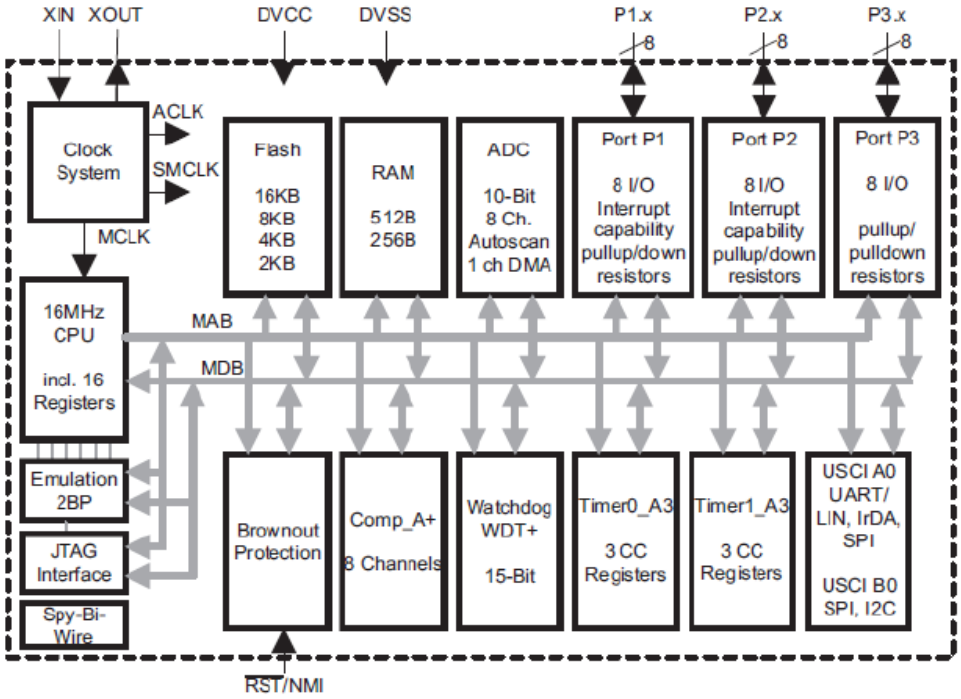

图 5-1 MSP430G2x53 微控制器结构

如图 5-1 所示，此次课程设计主要用到了 MSP430G2553 内部的定时器功能。

MSP430 单片机中 Timer_A 定时器就是一种辅助功能强大的定时器，具备捕获和 PWM 输出等极其有用功能。

MSP430x2xx 系列单片机的 Timer_A 模块的整体构造如图所示，包括 1 个 16 位定时器（Timer Block）和 3 个捕获比较模块（CCRx）。

1）16 位定时器的最大定时值 65535，当前计数值被存放在 TAR 寄存器中。

2）CCRx 的捕获模块 Caputre 由 1 个输入 IO 口（CCIx）控制，输入上升沿或下降沿均能触发比较模块动作，捕获发生后的瞬间 TAR 值被存入 TACCRx 寄存器。

3）CCRx 的比较模块 Comparator 控制 1 个输出 IO 口（TAx）去生成各种脉冲波形。当 TAR 计数值与预存入 TACCRx 寄存器的值相等时，比较模块动作，以某种预设规则控制 IO 电平，生成波形。

4）由于捕获模块 Caputre 和比较模块 Comparator 共用了 TACCRx 寄存

器，捕获 Capture 的功能是写 TACCRx，而比较 Comparator 的功能是读 TACCRx 模块，所以捕获和比较不能同时使用。

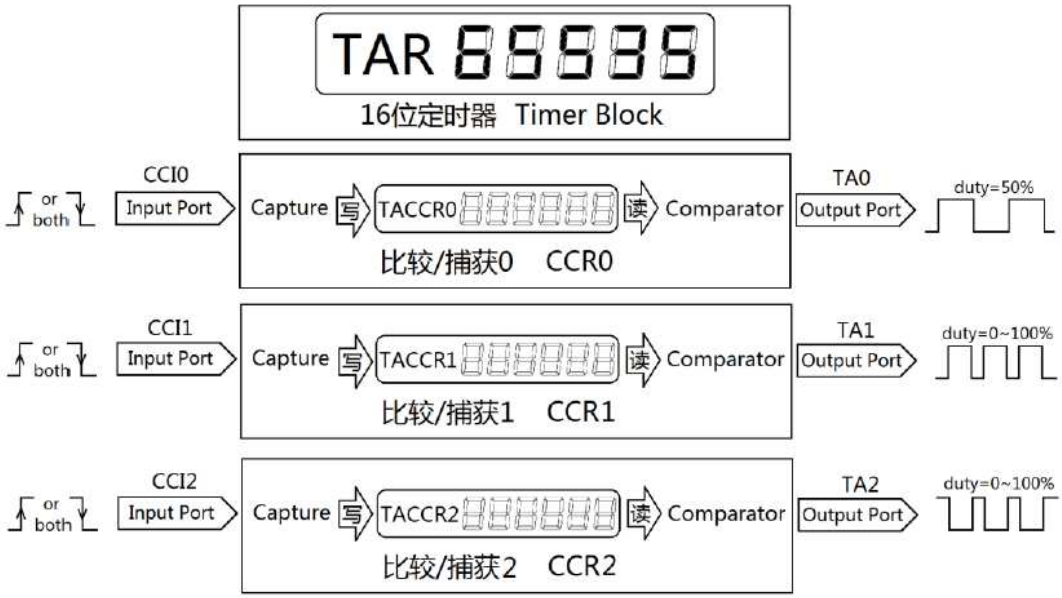

图 5-2 Timer_功能框图

捕获模块

将 CAP 设置为 1，CCRx 工作于捕获模式。主定时器一般设置为连续计数模式，当 CCRx 检测到 CCIx（某带捕获功能的 IO 口）的电平边沿时，瞬间读取 TAR 寄存器的值并写入 TACCRx 。CCRx 可以选择检测上升沿或下降沿，或者都检测。CCRx 用于测定信号脉宽时，只需要分别记录信号上升沿时刻和下降沿时刻，两时刻相减就是脉宽；而测量频率时，连续记录两次上升沿时刻，相减就是周期。

比较模块

当 CAP=0 时，CCRx 工作于比较模式。CCR0 在比较模式中，将用于设定定时器的周期，所以我们暂时当 CCR0“牺牲”了，只讨论 CCR1 和 CCR2 的工作情况。当 CCR1/2 发现 TAR 的值与 TACCR0 或它们自己的 TACCRx 相等时，便会自动改变输出 IO 口 TAx 的输出电平，从而生成波形。改变的规则由 OUTMODx 寄存器决定，共有 8 种规则。

这 8 种规则配合主定时器 TAR 的 3 种模式（连续计数、增计数、增减计数），可以无需 CPU 干预生成各种波形。

本次课题，我将 P2.1 复用为捕获输入引脚，用来捕获光电反射式传感器输出的方波周期。

(二) ST188

ST188 为反射式红外光电传感器。采用高发射功率红外光电二极管和高灵敏度光电晶体管组成。检测距离可调整，采用非接触检测方式。

图 5-4 是光电二极管的外形图，由发射二极管和接收管组成。A、K 是红外发射二极管的正负极，C、E 是接收管的正负极。因此只要 A 极接高电平、K 极接低电平，红外发射管就能发出红外线。可以在传感器加上外围电路来检测接收管的信号，进而确定是否接收到反射回来的红外线。

图 5-5 图 5-6 为 ST188 的极限参数和电气特性。

图 5-3 ST188 实物图

图 5-4 ST188 外形图

极限参数 (Ta=25℃)

项目		符号	数值	单位	
输入	正向电流	IF	50	mA	流过发光二极管的电流（A_K 电流）
	反向电压	Vr	6	V	反向加在 A_K 间的电压
	耗散功率	P	75	mW	
输出	集-射电压	Vceo	25	V	最大集射电压
	射-集电压	Veco	6	V	最大射集电压
	集电极功耗	Pc	50	mW	
工作温度		Topr	-20∽65	℃	实用的环境温度
储存温度		Tstg	-30∽75	℃	

图 5-5 ST188 极限参数

项　　目		符号	测试条件		最小	典型	最大	单位
输入	正向压降	V_F	I_F=20mA		-	1.25	1.5	V
	反向电流	I_R	V_R=3V		-	-	10	μA
输出	集电极暗电流	I_{CEO}	Vce=20V		-	-	1	μA
	集电极亮电流	IL	Vce=15V	H1	0.30	-	-	mA
				H2	0.40	-	-	mA
			I_F=8 mA	H3	0.50	-	-	mA
	饱和压降	V_{CE}	I_F=8mA　I_C=0.5 mA		-	-	0.4	V

图 5-6 ST188 电气特性

(三) LM393

LM393 是双电压比较器集成电路。

输出负载电阻能衔接在可允许电源电压范围内的任何电源电压上，不受 Vcc 端电压值的限制.此输出能作为一个简单的对地 SPS 开路(当不用负载电阻没被运用)，输出部分的陷电流被可能得到的驱动和器件的 β 值所限制.当达到极限电流(16mA)时，输出晶体管将退出而且输出电压将很快上升。

以下是 LM393 的实物图与内部结构透视图。

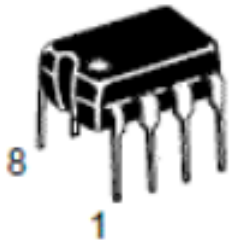

图 5-8 LM393 实物图

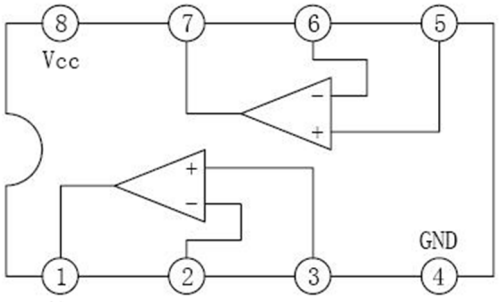

图 5-7 LM393 内部透视图

此次课题的光电反射式传感器电路就借助了 LM393 内部的 1 个电压比较器实现方波输出。

(四) 128 段 LCD 液晶屏

MSP430G2553 通过 I2C 协议 SCL、SDA，对应为 P1.6 和 P1.7 去控制扩展版上的 TCA6416A 芯片输出 4 个信号 CS、WR、RD、DATA，对应的引脚分别为 P1.4、P1.6、P1.5、P1.7，控制 LCD 驱动芯片 HT1621，来实现 128 段 LCD 的显示。下图 5-9、图 5-10 为 LCD 的控制流程和 LCD 原理图。

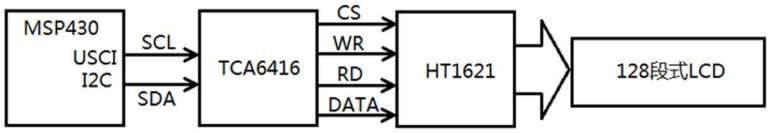

图 5-9 LCD 控制流程

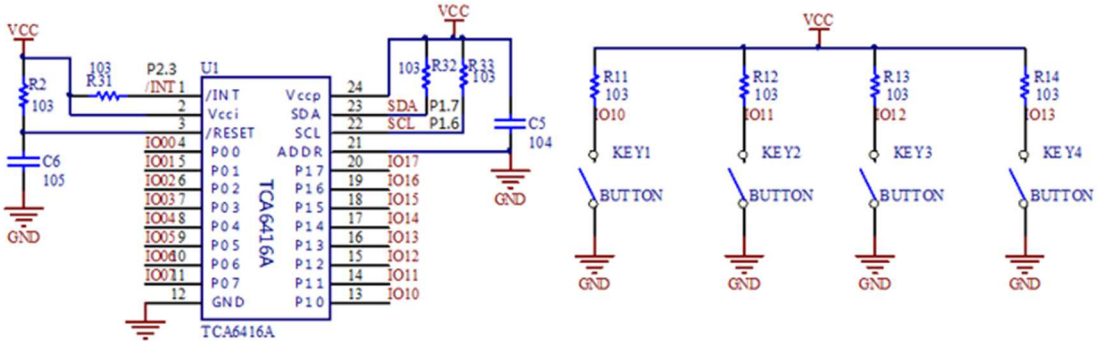

图 5-11 基于 I2C 的 IO 扩展

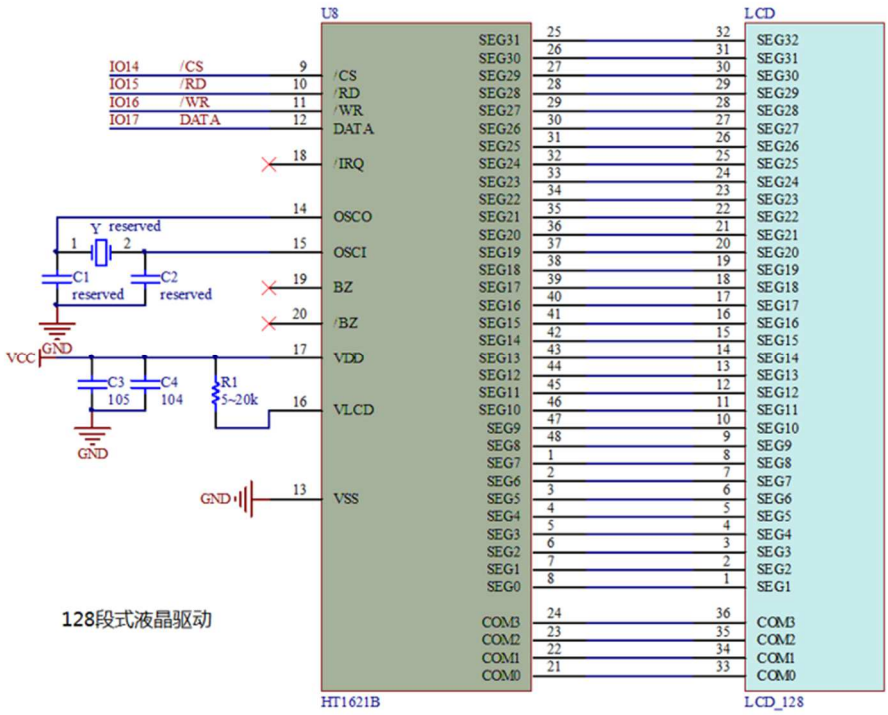

图 5-10 LCD 驱动模块

5.2 电路设计

(一) 光电传感器模块

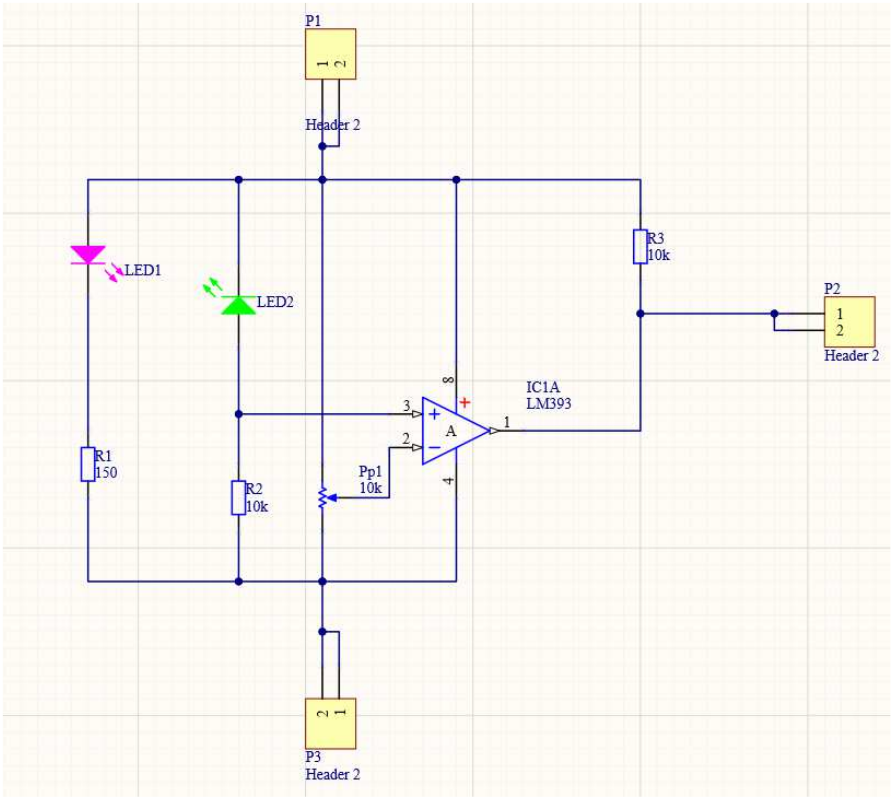

图 5-12 光电传感器模块

如图 5-12 所示，LED1 为红外发射管，LED2 为红外接收管。工作原理为当红外接收管 LED2 接收到红外线时，LED2 导通，根据接收到的光强不同，LM393 同相输入端电压也会不一样，调节反向输入端的滑动变阻器可以人为规定比较电压。

当前方有障碍物时，LED2 接收到反射的红外线，LED2 导通，同向输入端的电压大于反向输入端的电压，LM393 的 1 引脚输出高电平。相反当前方没有障碍物时，LED2 截至，同向输入端的电压小于反向输入端的电压，LM393 的 1 引脚输出低电平。在测量转速时，前方扇叶或物体的周期遮挡和不遮挡将会使得比较器 LM393 输出周期固定的方波，将该输出端交给单片机的 IO 引脚捕获即可。

6 软件设计

6.1 中断流程图

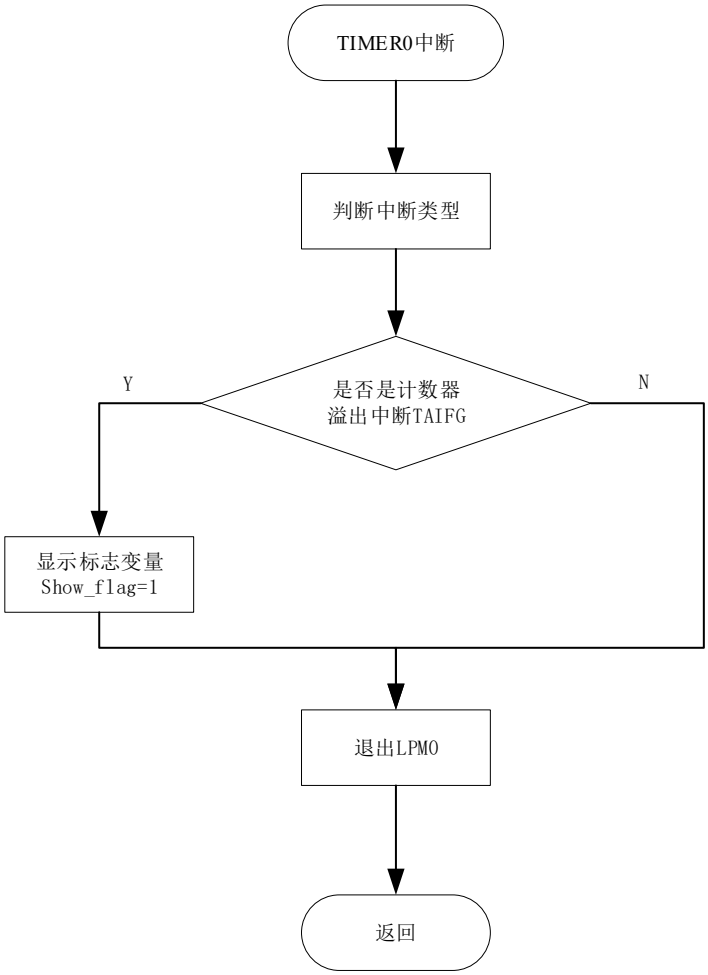

图 6-1 TIMER0 中断服务函数

如上**错误!未找到引用源。**所示，为定时器 TIMER0 的中断流程图，当发生定时器 TIMER0 中断时，进入中断服务函数判断引发的是哪种类型的中断，该定时器用到的是 TAIFG 溢出中断向量。一旦引发 TAIFG 溢出中断，则将转速显示标志变量 show_flag=1。由于 TIMER0 定时器选的时钟源为 32768Hz，计数模式为连续模式，所以转速大约每 2 秒显示一次。

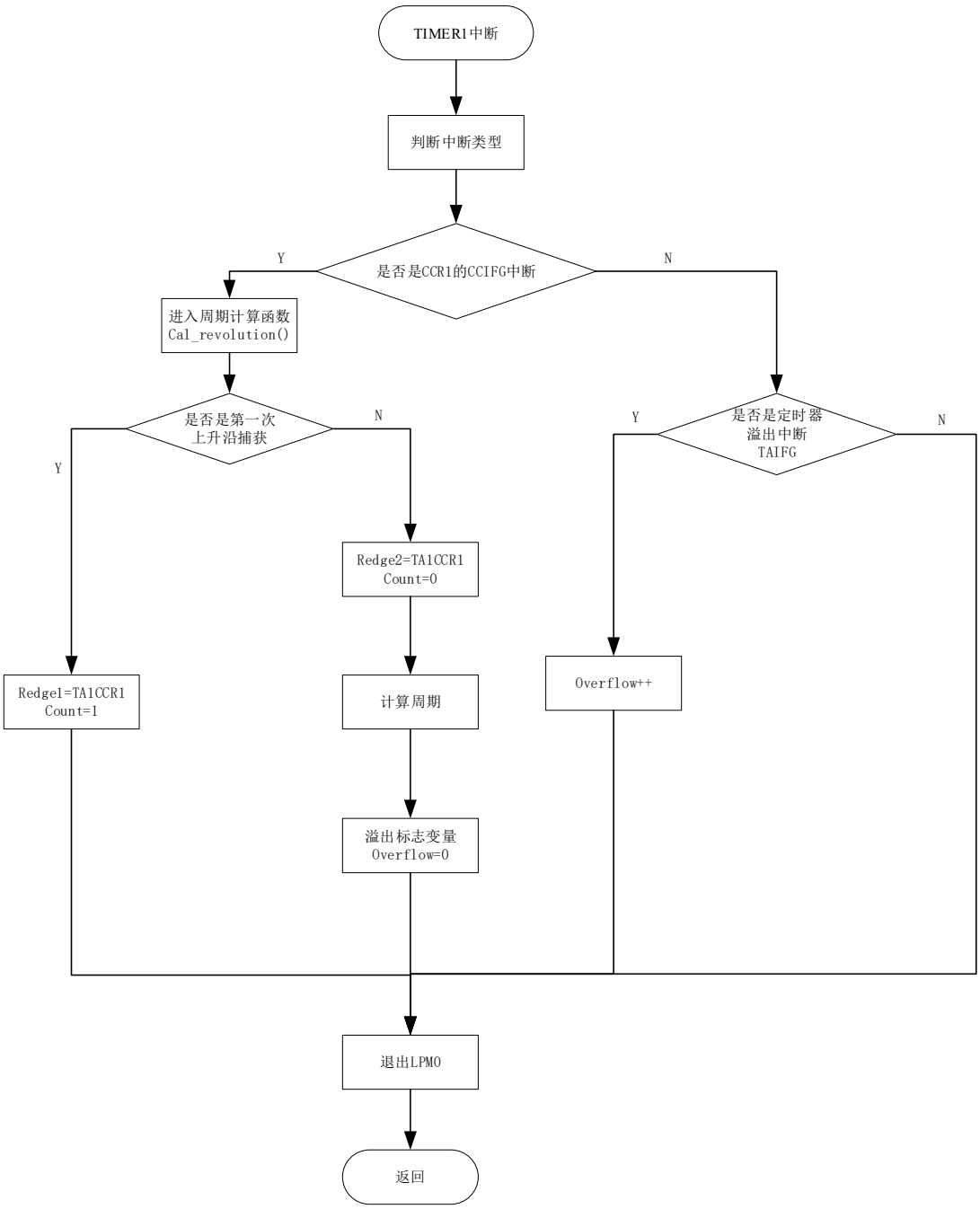

图 6-2 TIMER1 中断服务函数

如图 6-2 所示，为 TIMER1 的中断服务函数流程图，当发生 TIMER1 中断时，判断引发的是捕获中断 CCIFG 还是溢出中断 TAIFG。如果是 CCIFG 中断，并且是第一次上升沿捕获，读取 TA1CCR1 的值，赋值给变量 REdge1，并且将上升沿捕获次数变量 count=1，退出 LPM0，返回主函数；如果不是第一次上升沿捕获，读取 TA1CCR1 的值，赋值给变量 FEdge2，并且将上升沿捕获次数变量 count=0，计算方波的周期，将计数溢出标志变量 overflow=0，退出 LPM0，返回

主函数。

一旦发生定时器溢出中断 TAIFG，将溢出标志 overflow 累加，退出低功耗模式 LPM0，返回主函数。

6.2 主函数流程图

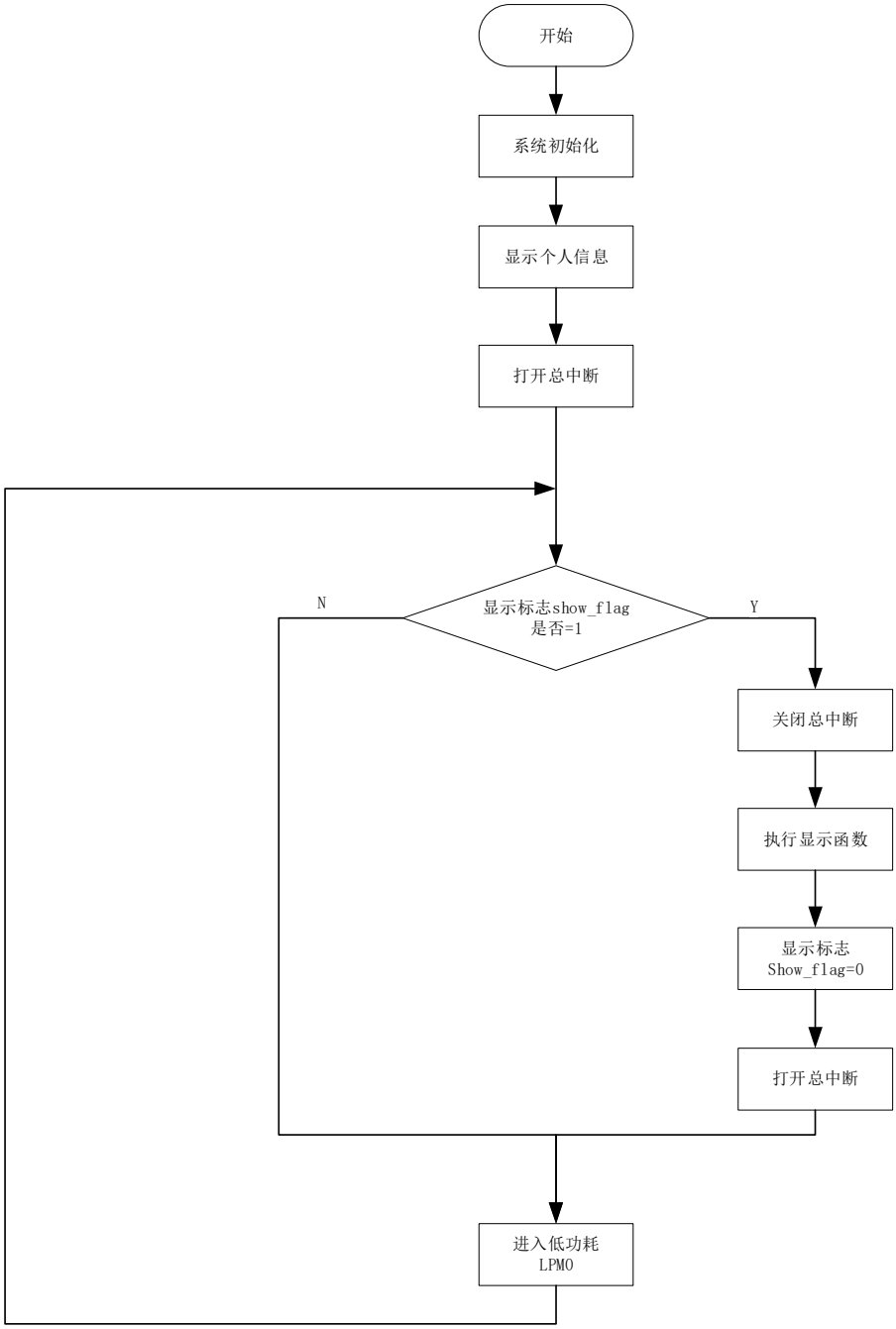

图 6-3 主函数流程图

如图 6-3 所示，为此次主程序的流程图，系统上电，系统程序进入初始化，显示个人信息，打开总中断，进入死循环。开始，判断显示标志 show_flag=1，如果 show_flag=1，那么关闭总中断，执行显示函数，将将转速显示标志变量 show_flag=0，关闭总中断，进入低功耗等待中断函数退出低功耗模式 LPM0；如果 show_flag=0，那么直接进入低功耗模式 LPM0，等待中断函数退出低功耗模式 LPM0。

6.3 显示函数流程图

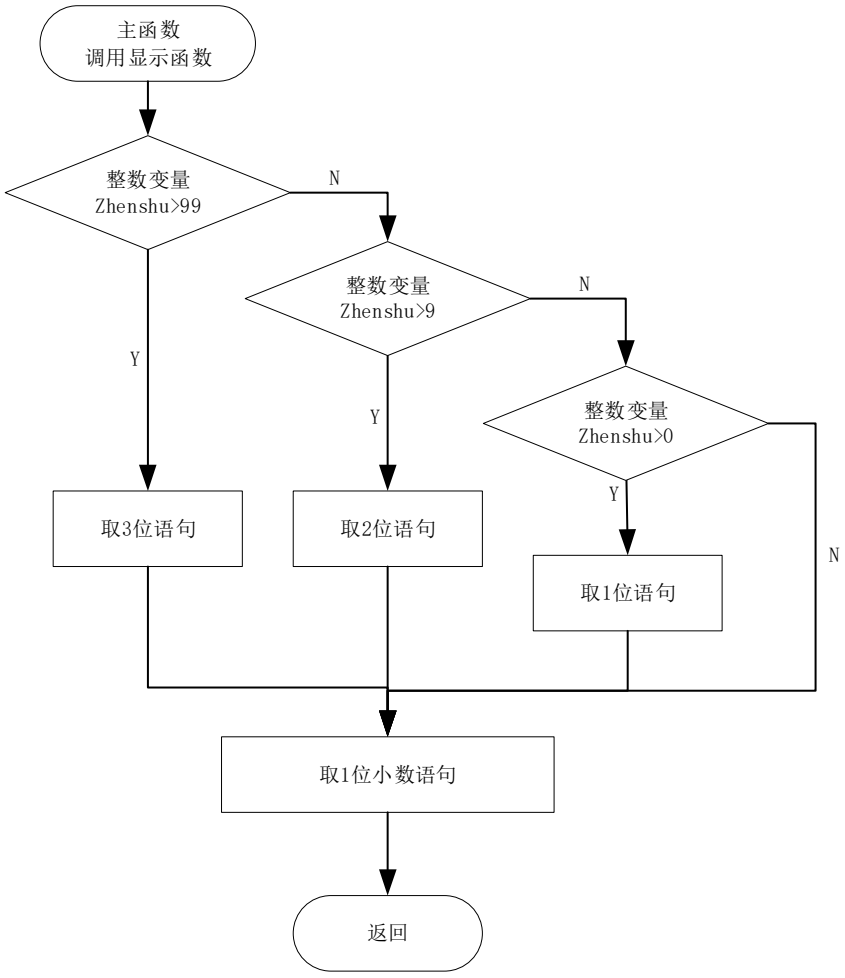

图 6-4 显示函数流程图

主函数调用显示函数之前，已经对转速的整数部分和小数部分完成处理，将整数部分保存在变量 zhenshu 中，将小数部分保存在变量 xiaoshu 里。调用显示函数时，首先判断整数部分的转速区间，然后再执行不同的取位函数，最后将整数和小数一同显示在 128 段液晶显示屏上。

7 硬件调试

7.1 转速测试

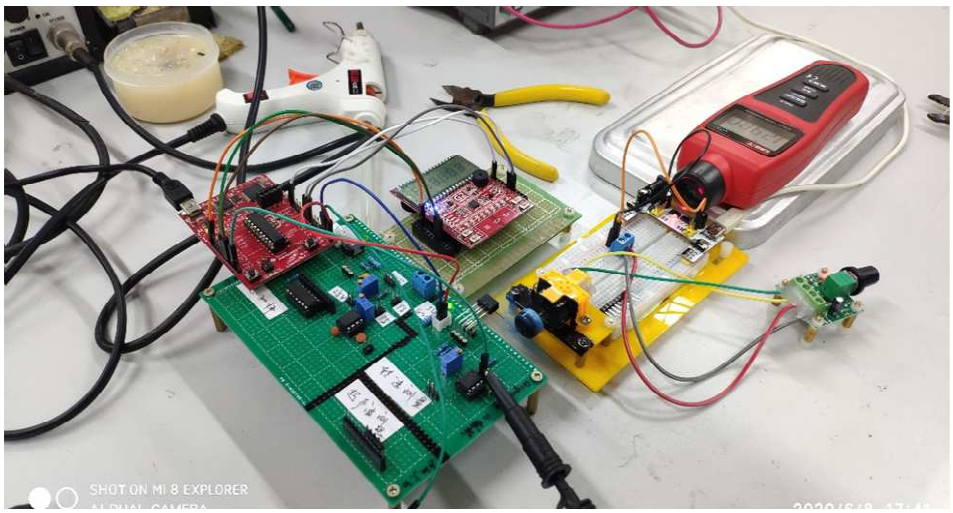

图 7-1 系统上电

如图 7-1 所示，系统上电

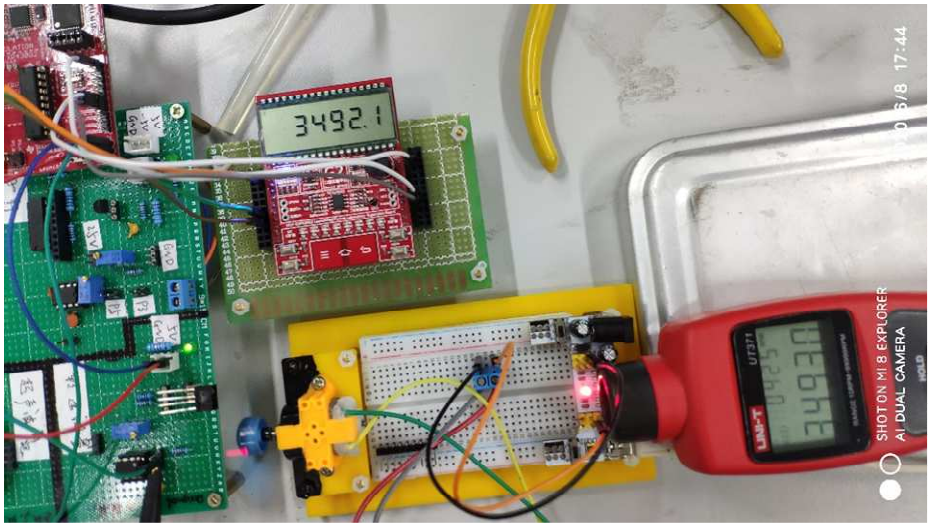

图 7-2 转速测试 1

如图 7-2 所示，为 3492rpm 测试，LCD 显示稳定，比较准确。

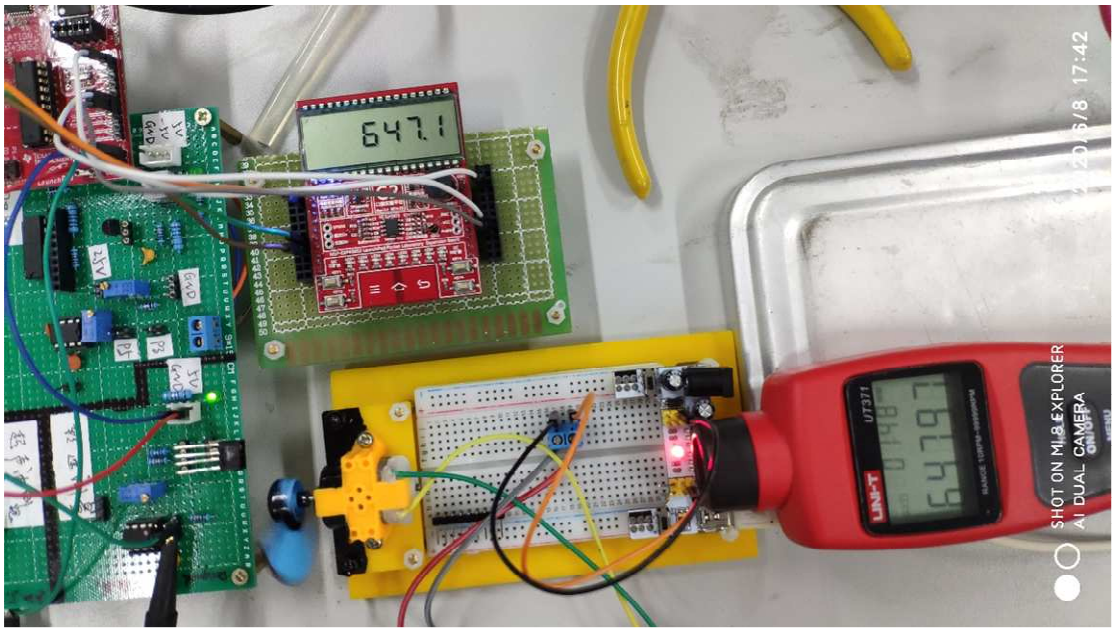

图 7-3 转速测试 2

如图 7-3 所示，为 647rpm，LCD 显示稳定，比较准确。

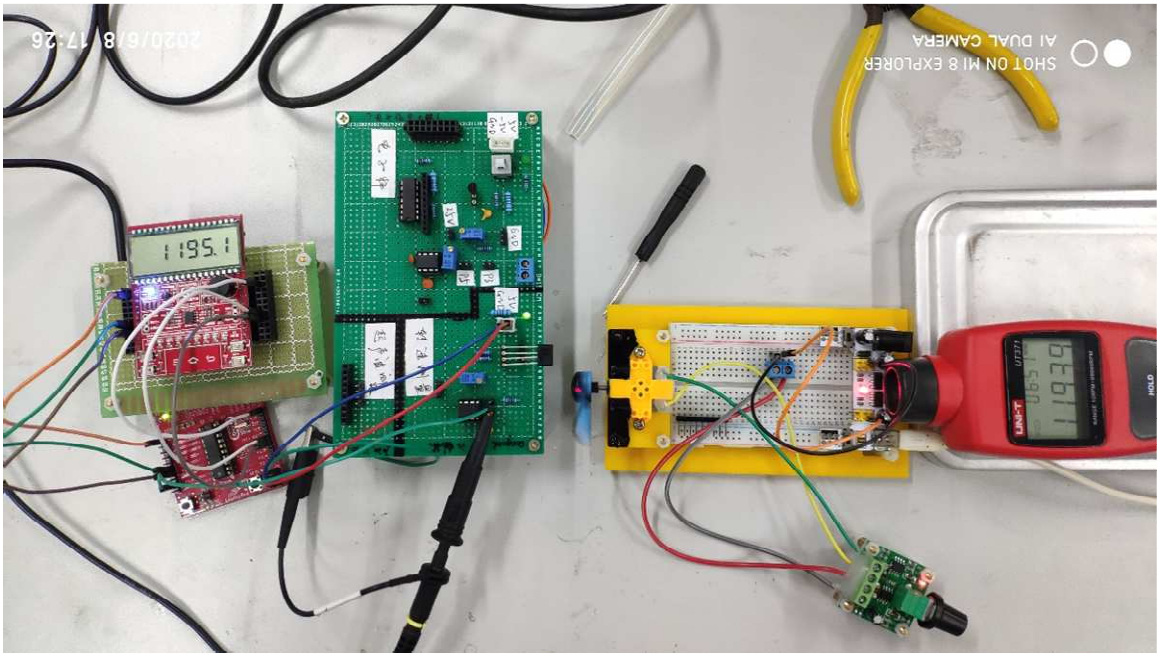

图 7-4 转速测试 3

如图 7-4 所示，为 1195rpm，LCD 显示稳定，比较准确。

7.2 数据记录

表格 7-1 调试数据记录

仪表显示 单位 rpm	LCD 显示 单位 rpm	误差
648.0	647.1	-0.13%
1193.9	1195.1	0.10%
1268.0	1269.2	0.09%
1318.0	1323.4	0.41%
1322.1	1322.1	0.00%
1510.0	1511.5	0.10%
1827.4	1823.8	-0.20%
1872.0	1870.6	-0.07%
1991.1	1989.9	-0.06%
2303.1	2302.2	-0.04%
2324.6	2323.9	-0.03%
2351.9	2351.7	-0.01%
2407.1	2409.4	0.10%
2686.0	2685.0	-0.04%
3163.9	3160.9	-0.09%
3510.8	3505.6	-0.15%
3817.9	3817.6	-0.01%
3828.5	3825.6	-0.08%
3837.6	3825.0	-0.33%
4096.0	4095.4	-0.01%
4561.6	4560.4	-0.03%
4727.1	4726.1	-0.02%
4747.3	4748.9	0.03%
4788.8	4795.3	0.14%
4991.7	4990.0	-0.03%
5093.4	5091.5	-0.04%
5491.8	5491.1	-0.01%
5649.6	5641.6	-0.14%
5694.5	5665.9	-0.50%
5738.5	5732.0	-0.11%

实验测量数据如上表所示，可以观察到误差都很小，1%以下。就没有考虑拟合的处理。

8 设计小结

此次的非接触式转速测量表让我对光电传感器有了些许了解。除了翻阅光电的技术手册外，此次的波形捕获，我又一次加深了对 MSP430G2553 的认识。波形捕获的关键之一在于中断函数的编写，将 IO 引脚初始化为波形捕获输入后，一旦发生沿的跳变，就会引发 CCIFG 中断。此次的课题我用到了 TIMER1 的 CCR1 作为波形的捕获寄存器，TIMER0 作为定时计数器的功能。另外，周期的计算也是波形捕获的重中之重，需要考虑变量的类型、运算溢出、计数溢出等一系列问题。

此次的传感器的课程设计，感谢李老师和季老师的指导与帮助，不管是在硬件电路设计上还是程序设计上他们都给了我特别大的启发。

9 参考文献

[1]毛敏.电机转速测量系统设计[J].山东工业技术,2017(10):189-190.

附件 1 源程序代码

```
#include <msp430G2553.h>
#include "LCD_128.h"
#include "TCA6416A.h"
#include "HT1621.h"

int zhengshu = 0, xiaoshu = 0,show_flag=0;
char overflow_times = 0;

float zhuansu = 0;
void LCD_Init();
void gpio_init();
void timer0_init();
void time1_stop();
void timer1_init();
void Revolution_Display();
void display_my_info();
void cal_revolution();
int main(void)
{
    WDTCTL = WDTPW + WDTHOLD;
    TCA6416A_Init();
    LCD_Init();
    gpio_init();
    timer0_init();
    timer1_init();
    display_my_info();
    _EINT();
    while (1)
```

```
        {
            if(show_flag==1)
            {
              _DINT();
              Revolution_Display();
              show_flag=0;
              _EINT();
            }
            LPM0;
        }
}


void gpio_init()
{
    P2DIR &= ~BIT1; //P2.1 设置为输入
    P2SEL |= BIT1;   //使用 P2.1 复用功能
}

void display_my_info()
{
    LCD_DisplaySeg(4);
    LCD_DisplaySeg(9);
    LCD_DisplaySeg(10);//显示 J
    LCD_DisplayDigit(8, 2);
    LCD_ClearSeg(12);
    LCD_ClearSeg(19);//显示 H
    LCD_DisplayDigit(0, 3);
    LCD_ClearSeg(25);
    LCD_ClearSeg(26);
```

```
    LCD_ClearSeg(27);//显示 L
    LCD_DisplayDigit(1, 4);
    LCD_DisplayDigit(0, 5);
    LCD_DisplayDigit(7, 6);
    HT1621_Reflash(LCD_Buffer);
    __delay_cycles(1000000);
    LCD_Clear();
    LCD_DisplayDigit(0, 5);
    LCD_DisplayDigit(0, 6);
    LCD_DisplaySeg(_LCD_DOT4);
    HT1621_Reflash(LCD_Buffer);
}

void cal_revolution()
{
    static int REdge1 = 0, REdge2 = 0,Period=0;
    static char Count = 0;
    const float f=32768;
    if (!Count)
    {
        REdge1 = TA1CCR1;
        Count=1;
    }
    else
    {
        REdge2 = TA1CCR1;
       // time1_stop();
        Count = 0;
        Period=REdge2 - REdge1;
        zhuansu =(f/(65536.0*overflow_times+Period))*60;
```

```
            zhengshu = (int)zhuansu; //zhengshu 为显示整数，在显示程序中调用
            xiaoshu = (zhuansu - zhengshu) * 10;
            overflow_times = 0;
            //TA1R=0;
        }
    }

    void timer0_init()
    {
        TA0CTL = TASSEL_1 + MC_2 + TACLR + TAIE;                    //
ACLK, Continous up,start,interrupt enable
    }

    void timer1_init()
    {
        TA1CCTL1 = CAP + CM_1 + CCIE + SCS + CCIS_0;
        TA1CTL |= TASSEL_1 + MC_2 + TACLR + TAIE; // SMCLK, Cont Mode;
start timer
    }


    #pragma vector = TIMER0_A1_VECTOR
    __interrupt void TIMER0_A0(void)
    {
     switch(TA0IV)
     {
          case  TA0IV_NONE:     break;          // Vector  0:  No interrupt
          case  TA0IV_TACCR1: break;          // Vector  2:  TACCR1 CCIFG
          case  TA0IV_TACCR2: break;          // Vector  4:  TACCR2 CCIFG
          case  TA0IV_6: break;               // Vector  6:  Reserved CCIFG
```

```
        case    TA0IV_8: break;                 // Vector    8:    Reserved CCIFG
        case    TA0IV_TAIFG:                    // Vector 10:    TAIFG
                show_flag=1;                    //-----启用中断服务函数-----
               // TA0CTL&=~TAIFG;
                break;                          // Vector 10:    TAIFG
        default:    break;
    }
    LPM0_EXIT;
}

#pragma vector = TIMER1_A1_VECTOR
__interrupt void TIMER1_A1_ISR(void)
{
    switch (TA1IV) //AIV 的值是在 0--10 内的偶数时才会执行 switch 函数内的语句
    {
    case TA1IV_NONE:
        break;              // Vector    0:    No interrupt
    case TA1IV_TACCR1: // Vector    2:    TACCR1 CCIFG
        cal_revolution();
        //__bic_SR_register_on_exit(LPM0_bits); // Exit LPM0 on return to main
        break;
    case TA1IV_TACCR2:
        break; // Vector    4:    TACCR2 CCIFG
    case TA1IV_6:
        break; // Vector    6:    ON_Perioderved CCIFG
    case TA1IV_8:
        break; // Vector    8:    ON_Perioderved CCIFG
    case TA1IV_TAIFG:
```

```
            overflow_times += 1;
        //    TA1CTL &= ~TAIFG;
            // __bic_SR_register_on_exit(LPM0_bits + GIE);
            break;
        default:
            break;
        }
        LPM0_EXIT;
    }

    void time1_stop()
    {
        TA1CTL |= MC_0 + TACLR; // stop

    }

    void LCD_Init()
    {
        HT1621_init();
        //相关硬件的初始化，其中 I2C 模块的初始化由 TCA6416A 初始化函数在内部完成了， LCD_128 库函数由 HT1621 初始化函数在内部引用了
        //-----显示固定不变的 LCD 段-----
        LCD_DisplayDigit(5,2);
        LCD_DisplayDigit(8,3);
        LCD_ClearSeg(20);
        LCD_ClearSeg(25);
        LCD_DisplayDigit(6,4);
        LCD_ClearSeg(53);
        LCD_DisplayDigit(6,5);
        LCD_ClearSeg(61);
```

```
    LCD_DisplayDigit(0,6);
    HT1621_Reflash(LCD_Buffer);
    __delay_cycles(1000000);
}

void Revolution_Display()
{

    LCD_DisplayDigit(LCD_DIGIT_CLEAR, 2);
    LCD_DisplayDigit(LCD_DIGIT_CLEAR, 3);
    LCD_DisplayDigit(LCD_DIGIT_CLEAR, 4);
    //-----根据 ON_Period 拆分并显示数字-----
    if (zhengshu > 999) //1000~9999（4 位）
    {
        LCD_DisplayDigit(zhengshu / 1000, 2);
        LCD_DisplayDigit((zhengshu / 100) % 10, 3);
        LCD_DisplayDigit((zhengshu / 10) % 10, 4);
        LCD_DisplayDigit(zhengshu % 10, 5);
        LCD_DisplayDigit(xiaoshu, 6);
    }
    else if (zhengshu > 99) //100~999（3 位）
    {
        LCD_DisplayDigit(zhengshu / 100, 3);
        LCD_DisplayDigit((zhengshu / 10) % 10, 4);
        LCD_DisplayDigit(zhengshu % 10, 5);
        LCD_DisplayDigit(xiaoshu, 6);
    }
    else if (zhengshu > 9) //（2 位）
    {
        LCD_DisplayDigit(zhengshu / 10, 4);
```

```
            LCD_DisplayDigit(zhengshu % 10, 5);
            LCD_DisplayDigit(xiaoshu, 6);
        }
        else
        {
            LCD_DisplayDigit(0, 5);
            LCD_DisplayDigit(0, 6);
        }
        HT1621_Reflash(LCD_Buffer); //-----更新缓存，真正显示-----
    }
```