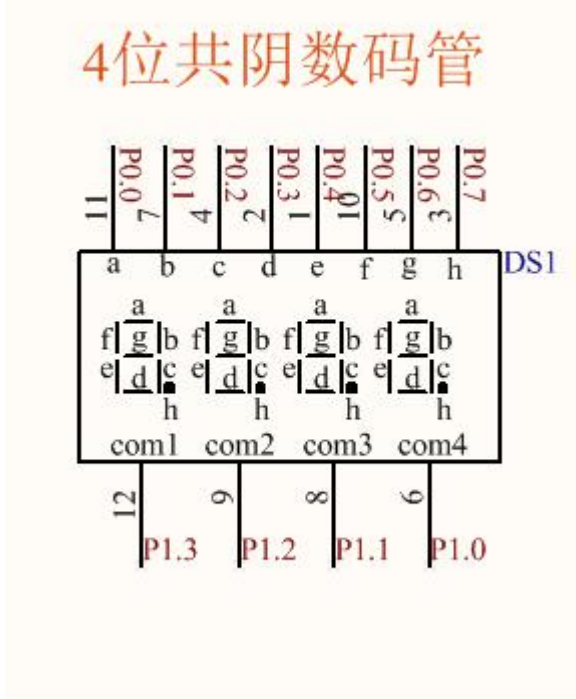
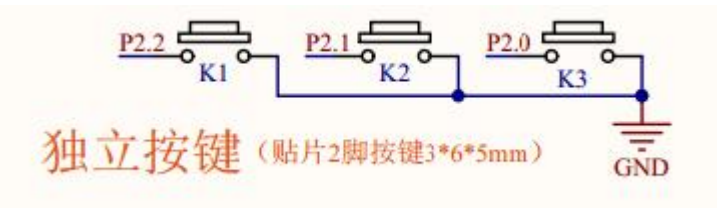


第一百一十五节： 按键控制数码管的秒表。

【115.1 按键控制数码管的秒表。】



上图 115.1.1 数码管



上图 115.1.2 独立按键

本节通过一个秒表的小项目，让大家学会以下 4 个知识点：

- （1）上层的界面显示框架几乎都要用到更新变量，更新变量包括整屏更新和局部更新，本节只用到整屏更新。更新变量是用全局变量在函数之间传递信息。作用是，当有某个需要显示的数据发生改变的时候，就要给更新变量置 1，让显示函数重新更新一次显示，确保最新的数据能及时显示出来，平时没有数据更新改变的时候不用频繁更新显示避免占用 CPU 过多的时间。
- （2）凡是需要显示数字的地方，都必须涉及如何把一个数据按“个十百千万...”的位逐个提取出来的算法。这个算法比较简单，主要用“求余”和“求商”这两个运算语句就可以随心所欲的把数据位提取出来。除此之外，还要学会如何用 if 语句判断数据的范围，来把高位尚未用到的某个数码管屏蔽，让该位数码管只显示一个“不显示”的数据（避免直接显示一个 0）。
- （3）我常把单片机程序简化成 4 个代表：按键（人机输入），数码管（人机界面），跑马灯（应用程序），串口（通信）。本节的“应用程序”不是跑马灯，而是秒表。不管是跑马灯，还是秒表，都要用到一个总启动 Gu8RunStart 和一个总运行步骤 Gu8RunStep。建议大家，总启动 Gu8RunStart 和总运行步骤

Gu8RunStep 应该成双成对的出现（这样关断响应更及时，并且结构更紧凑，漏洞更少），比如，凡是总启动 Gu8RunStart 发生改变的时候，总运行步骤 Gu8RunStep 都复位归零一下。

（4）一个硬件的定时器中断，可以衍生出 N 个软件定时器，之前跟大家介绍的是“递减式”的软件定时器，而且实际应用中，“递减式”的软件定时器也是用得最多。本节因为项目的需要，需要用到的是“累加式”的软件定时器。不管是哪种软件定时器，大家都要注意定时器变量在定义时所用到的数据类型，这个数据类型决定了定时时间的长度，比如在 51 单片机中，unsigned int 的范围是 0 到 65535，最大一次性定时 65.535 秒。而 unsigned long 的范围是 0 到 4294967295，最大一次性定时 4294967.295 秒。本节秒表的时间超过 65.535 秒，因此需要用到 unsigned long 类型的定时器变量。

本节秒表程序的功能：K1 按键是复位按键，每按一次，秒表都停止并且重新归零。K2 按键是启动和暂停按键，当秒表处于复位后停止的状态时按一次则开始启动，当秒表处于正在工作的状态时按一次则处于暂停状态，当秒表处于暂停的状态时按一次则继续处于工作的状态。本节 4 位数数码管，显示的时间是带 2 位小数点的，能显示的时间范围是：0.00 秒到 99.99 秒。代码如下：

```
#include "REG52.H"

#define KEY_FILTER_TIME  25

#define SCAN_TIME  1

void T0_time();
void SystemInitial(void) ;
void Delay(unsigned long u32DelayTime) ;
void PeripheralInitial(void) ;

void KeyScan(void);
void KeyTask(void);

void DisplayScan(void); //底层显示的驱动函数
void DisplayTask(void); //上层显示的任务函数

void RunTask(void); //秒表的应用程序

sbit KEY_INPUT1=P2^2;
sbit KEY_INPUT2=P2^1;

sbit P1_0=P1^0;
sbit P1_1=P1^1;
sbit P1_2=P1^2;
sbit P1_3=P1^3;

//数码管转换表
code unsigned char Cu8DigTable[]=
{
0x3f, //0      序号 0
```

```

0x06, //1      序号 1
0x5b, //2      序号 2
0x4f, //3      序号 3
0x66, //4      序号 4
0x6d, //5      序号 5
0x7d, //6      序号 6
0x07, //7      序号 7
0x7f, //8      序号 8
0x6f, //9      序号 9
0x00, //不显示 序号 10
};

//数码管底层驱动扫描的软件定时器
volatile unsigned char vGu8ScanTimerFlag=0;
volatile unsigned int vGu16ScanTimerCnt=0;

//秒表的软件定时器，注意，这里是 unsigned long 类型，范围是 0 到 4294967295 毫秒
volatile unsigned char vGu8StopWatchTimerFlag=0;
volatile unsigned long vGu32StopWatchTimerCnt=0;

//数码管上层每 10ms 就定时刷新一次显示的软件定时器。用于及时更新显示秒表当前的实时数值
volatile unsigned char vGu8UpdateTimerFlag=0;
volatile unsigned int vGu16UpdateTimerCnt=0;

unsigned char Gu8RunStart=0; //应用程序的总启动
unsigned char Gu8RunStep=0; //应用程序的总运行步骤。建议跟 vGu8RunStart 成双成对出现
unsigned char Gu8RunStatus=0; //当前秒表的状态。0 代表停止，1 代表正在工作中，2 代表暂停

unsigned char Gu8WdUpdate=1; //开机默认整屏更新一次显示。此变量在显示框架中是非常重要的变量

volatile unsigned char vGu8Display_Righ_4=10; //开机默认最高位数码管显示一个“不显示”数据
volatile unsigned char vGu8Display_Righ_3=0;
volatile unsigned char vGu8Display_Righ_2=0;
volatile unsigned char vGu8Display_Righ_1=0;

volatile unsigned char vGu8Display_Righ_Dot_4=0;
volatile unsigned char vGu8Display_Righ_Dot_3=1; //开机默认保留显示 2 个小数点
volatile unsigned char vGu8Display_Righ_Dot_2=0;
volatile unsigned char vGu8Display_Righ_Dot_1=0;

volatile unsigned char vGu8KeySec=0;

void main()

```

```

{
    SystemInitial();
    Delay(10000);
    PeripheralInitial();
    while(1)
    {
        KeyTask();      //按键的任务函数
        DisplayTask();  //数码管显示的上层任务函数
        RunTask();      //秒表的应用程序
    }
}

void KeyTask(void)      //按键的任务函数
{
    if(0==vGu8KeySec)
    {
        return;
    }

    switch(vGu8KeySec)
    {
        case 1:        //复位按键

            Gu8RunStatus=0; //秒表返回停止的状态

            Gu8RunStart=0; //秒表停止
            Gu8RunStep=0; //总运行步骤归零。建议跟 vGu8RunStart 成双成对出现

            vGu8StopWatchTimerFlag=0;
            vGu32StopWatchTimerCnt=0; //秒表的软件定时器清零

            Gu8WdUpdate=1; //整屏更新一次显示

            vGu8KeySec=0;
            break;

        case 2:        //启动与暂停的按键
            if(0==Gu8RunStatus) //在停止状态下
            {
                Gu8RunStatus=1; //秒表处于工作状态

                vGu8StopWatchTimerFlag=0;
                vGu32StopWatchTimerCnt=0;
                vGu8StopWatchTimerFlag=1; //启动秒表的软件定时器
            }
        }
    }
}

```

```

        Gu8RunStart=1;    //秒表总开关启动
        Gu8RunStep=0;    //总运行步骤归零。建议跟 vGu8RunStart 成双成对出现
    }
    else if(1==Gu8RunStatus) //在工作状态下
    {
        Gu8RunStatus=2; //秒表处于暂停状态
    }
    else //在暂停状态下
    {
        Gu8RunStatus=1; //秒表处于工作状态
    }

    Gu8WdUpdate=1; //整屏更新一次显示，确保在暂停的时候能显示到最新的数据

    vGu8KeySec=0;
    break;
}
}

void DisplayTask(void) //数码管显示的上层任务函数
{
    //需要借用的中间变量，用来拆分数数据位
    static unsigned char Su8Temp_4, Su8Temp_3, Su8Temp_2, Su8Temp_1; //需要借用的中间变量

    /* 注释一：
    * 此处为什么要多加 4 个中间过渡变量 Su8Temp_X? 是因为 vGu32StopWatchTimerCnt 分解数据的时候
    * 需要进行除法和求余数的运算，就会用到好多条指令，就会耗掉一点时间，类似延时了一会。我们
    * 的定时器每隔一段时间都会产生中断，然后在中断里驱动数码管显示，当 vGu32StopWatchTimerCnt
    * 还没完全分解出 4 位有效数据时，这个时候来的定时中断，就有可能导致显示的数据瞬间产生不完整，
    * 影响显示效果。因此，为了把需要显示的数据过渡最快，所以采取了先分解，再过渡显示的方法。
    */

    if(1==Gu8WdUpdate) //如果需要整屏更新
    {
        Gu8WdUpdate=0; //及时清零，只更新一次显示即可，避免一直进来更新显示

        //先分解数据

        //Su8Temp_4 提取“十秒”位。
        Su8Temp_4=vGu32StopWatchTimerCnt/10000; //实际精度是 0.0001 秒,但显示精度是 0.01 秒
        Su8Temp_4=Su8Temp_4%10;

```

```

//Su8Temp_3 提取“个秒”位。
Su8Temp_3=vGu32StopWatchTimerCnt/1000; //实际精度是 0.0001 秒,但显示精度是 0.01 秒
Su8Temp_3=Su8Temp_3%10;

//Su8Temp_2 提取“百毫秒”位。
Su8Temp_2=vGu32StopWatchTimerCnt/100; //实际精度是 0.0001 秒,但显示精度是 0.01 秒
Su8Temp_2=Su8Temp_2%10;

//Su8Temp_1 提取“十毫秒”位。
Su8Temp_1=vGu32StopWatchTimerCnt/10; //实际精度是 0.0001 秒,但显示精度是 0.01 秒
Su8Temp_1=Su8Temp_1%10;

//判断数据范围,来决定最高位数码管是否需要显示。
if(vGu32StopWatchTimerCnt<10000) //10.000 秒。实际 4 位数码管最大只能显示 99.99 秒
{
    Su8Temp_4=10; //在数码管转换表里,10 代表一个“不显示”的数据
}

//上面先分解数据之后,再过渡需要显示的数据到底层驱动变量里,让过渡的时间越短越好
vGu8Display_Righ_4=Su8Temp_4; //过渡需要显示的数据到底层驱动变量
vGu8Display_Righ_3=Su8Temp_3;
vGu8Display_Righ_2=Su8Temp_2;
vGu8Display_Righ_1=Su8Temp_1;

vGu8Display_Righ_Dot_4=0;
vGu8Display_Righ_Dot_3=1; //保留显示 2 位小数点
vGu8Display_Righ_Dot_2=0;
vGu8Display_Righ_Dot_1=0;

}
}

void RunTask(void) //秒表的应用程序
{
    if(0==Gu8RunStart)
    {
        return; // 如果秒表处于停止状态,则直接退出当前函数,不执行该函数以下的其它代码
    }

    switch(Gu8RunStep)
    {
        case 0: //在这个步骤里,主要用来初始化一些参数

```

```

        vGu8UpdateTimerFlag=0;
        vGu16UpdateTimerCnt=10; //每 10ms 更新显示一次当前秒表的时间
        vGu8UpdateTimerFlag=1;

        Gu8RunStep=1; //跳转到每 10ms 更新显示一次的步骤里
        break;

    case 1: //每 10ms 更新一次显示，确保实时显示秒表当前的时间
        if(0==vGu16UpdateTimerCnt) //每 10ms 更新显示一次当前秒表的时间
        {
            vGu8UpdateTimerFlag=0;
            vGu16UpdateTimerCnt=10; //重置定时器，为下一个 10ms 更新做准备
            vGu8UpdateTimerFlag=1;

            Gu8WdUpdate=1; //整屏更新一次显示当前秒表的时间
        }
        break;
    }
}

void KeyScan(void) //按键底层的驱动扫描函数，放在定时中断函数里
{
    static unsigned char Su8KeyLock1;
    static unsigned int Su16KeyCnt1;
    static unsigned char Su8KeyLock2;
    static unsigned int Su16KeyCnt2;

    if(0!=KEY_INPUT1)
    {
        Su8KeyLock1=0;
        Su16KeyCnt1=0;
    }
    else if(0==Su8KeyLock1)
    {
        Su16KeyCnt1++;
        if(Su16KeyCnt1>=KEY_FILTER_TIME)
        {
            Su8KeyLock1=1;
            vGu8KeySec=1;
        }
    }
}

```

```

    if(0!=KEY_INPUT2)
    {
        Su8KeyLock2=0;
        Su16KeyCnt2=0;
    }
    else if(0==Su8KeyLock2)
    {
        Su16KeyCnt2++;
        if(Su16KeyCnt2>=KEY_FILTER_TIME)
        {
            Su8KeyLock2=1;
            vGu8KeySec=2;
        }
    }
}

```

void DisplayScan(void)     //数码管底层的驱动扫描函数，放在定时中断函数里

```

{
    static unsigned char Su8GetCode;
    static unsigned char Su8ScanStep=1;

    if(0==vGu16ScanTimerCnt)
    {

        P0=0x00;
        P1_0=1;
        P1_1=1;
        P1_2=1;
        P1_3=1;

        switch(Su8ScanStep)
        {
            case 1:
                Su8GetCode=Cu8DigTable[vGu8Display_Righ_1];

                if(1==vGu8Display_Righ_Dot_1)
                {
                    Su8GetCode=Su8GetCode|0x80;
                }
                P0=Su8GetCode;
                P1_0=0;
                P1_1=1;

```



```

        P1_2=1;
        P1_3=1;
        break;

    case 2:
        Su8GetCode=Cu8DigTable[vGu8Display_Righ_2];
        if(1==vGu8Display_Righ_Dot_2)
        {
            Su8GetCode=Su8GetCode|0x80;
        }
        P0=Su8GetCode;
        P1_0=1;
        P1_1=0;
        P1_2=1;
        P1_3=1;
        break;

    case 3:
        Su8GetCode=Cu8DigTable[vGu8Display_Righ_3];
        if(1==vGu8Display_Righ_Dot_3)
        {
            Su8GetCode=Su8GetCode|0x80;
        }
        P0=Su8GetCode;
        P1_0=1;
        P1_1=1;
        P1_2=0;
        P1_3=1;
        break;

    case 4:
        Su8GetCode=Cu8DigTable[vGu8Display_Righ_4];
        if(1==vGu8Display_Righ_Dot_4)
        {
            Su8GetCode=Su8GetCode|0x80;
        }
        P0=Su8GetCode;
        P1_0=1;
        P1_1=1;
        P1_2=1;
        P1_3=0;
        break;
}

```

```

        Su8ScanStep++;
        if (Su8ScanStep>4)
        {
            Su8ScanStep=1;
        }

        vGu8ScanTimerFlag=0;
        vGu16ScanTimerCnt=SCAN_TIME;
        vGu8ScanTimerFlag=1;
    }
}

void T0_time() interrupt 1
{
    KeyScan();        //按键底层的驱动扫描函数
    DisplayScan();    //数码管底层的驱动扫描函数

    if (1==vGu8ScanTimerFlag&&vGu16ScanTimerCnt>0)
    {
        vGu16ScanTimerCnt--; //递减式的软件定时器
    }

    //每 10ms 就定时更新一次显示的软件定时器
    if (1==vGu8UpdateTimerFlag&&vGu16UpdateTimerCnt>0)
    {
        vGu16UpdateTimerCnt--; //递减式的软件定时器
    }

    //秒表实际走的时间的软件定时器，注意，这里是“累加式”的软件定时器
    if (1==vGu8StopWatchTimerFlag&&1==Gu8RunStatus) //秒表处于工作的状态
    {
        vGu32StopWatchTimerCnt++; //累加式的软件定时器
    }

    TH0=0xfc;
    TL0=0x66;
}

void SystemInitial(void)
{
    P0=0x00;

```

```
P1_0=1;
P1_1=1;
P1_2=1;
P1_3=1;

TMOD=0x01;
TH0=0xfc;
TL0=0x66;
EA=1;
ET0=1;
TR0=1;
}

void Delay(unsigned long u32DelayTime)
{
    for(;u32DelayTime>0;u32DelayTime--);
}

void PeripheralInitial(void)
{
}
}
```