如何设计复杂的多任务程序

田开坤

湖北师范学院物电学院电工电子中心 435002 tkaikun@163.com

我们在入门阶段，一般面对的设计都是单一的简单的任务，流程图可以如图 1 所示，通常会用踏步循环延时来满足任务需要。

面对多任务，稍微复杂的程序设计，沿用图 1 的思想，我们会做出如图 2 所示的程序，在大循环体中不断增加任务，通常还要用延时来满足特定任务节拍，这种程序设计思想它有明显的不足，主要是各个任务之间相互影响，增加新的任何之后，以前很好的运行的任务有可能不正常，例如数码管动态扫描，本来显示效果很好的驱动函数，在增加新的任务后出现闪烁，显示效果变差了。

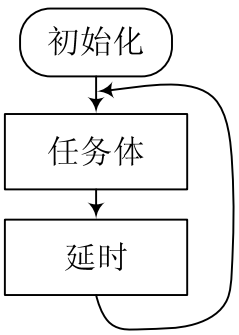

图1 单一任务简单流程图

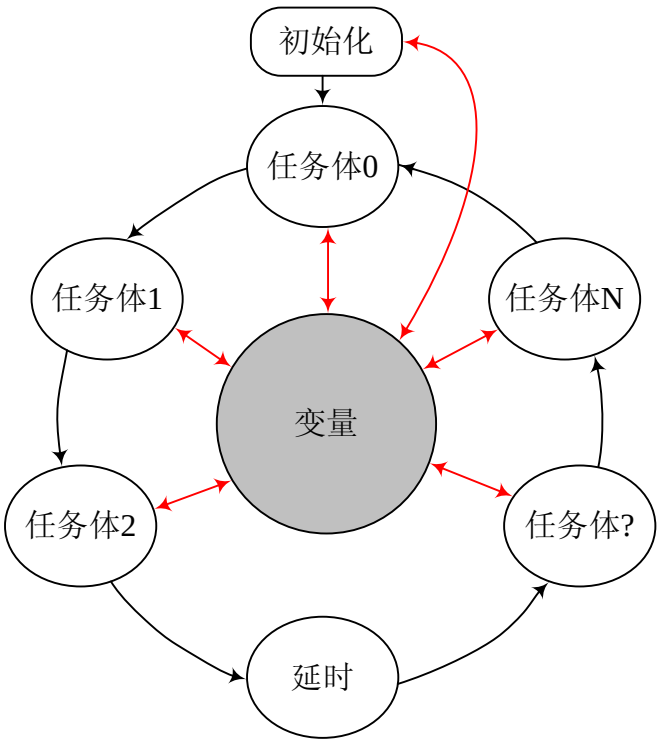

图2 多任务简单流程图

很明显，初学者在设计程序时，需要从程序构架思想上下功夫，在做了大量基本模块练习之后，需要总结提炼自己的程序设计思路（程序架构思想）。

首先我们来理解“任务”，所谓任务，就是需要 CPU 周期“关照”的事件，绝大多数任务不需要 CPU 一直“关照”，例如启动 ADC 的启动读取。甚至有些任务“害怕”CPU 一直“关照”例如 LCD 的刷新，因为 LCD 是显示给人看的，并不需要高速刷新，即便是显示的

内容在高速变化，也不需要高速刷新，道理是一样的。这样看来，让 CPU 做简单任务一定很浪费，事实也是如此，绝大多数简单任务，CPU 都是在“空转”（循环踏步延时）。对任务总结还可以知道，很多任务需要 CPU 不断“关照”，其实这种“不断”也是有极限的，比如数码管动态扫描，能够做到 40Hz 就可以了，又如键盘扫描，能够做到 20Hz（经验值），基本上也就不会丢有效按键键值了，再如 LCD 刷新，我觉得做到 10Hz 就可以了，等等。看来，绝大多数任务都是工作在低速频度。而我们的 CPU 一旦运行起来，速度又很快，CPU 本身就是靠很快的速度执行很简单的指令来胜任复杂的任务（逻辑）的。如果有办法把“快”的 CPU 分成多个慢的 CPU，然后给不同的任务分配不同速度的 CPU，这种设想是不是很好呢！确实很好，下面就看如何将“快”的 CPU 划分成多个“慢”的 CPU。

根据这种想法，我们需要合理分配 CPU 资源来“关照”不同的任务，最好能够根据任务本身合理占用 CPU 资源，首先看如图 3 所示的流程图，各个任务流程独立，各任务通过全局变量来交互信息，在流程中有一个重要的模块“任务切换”，就是任务切换模块实现 CPU 合理分配，这个任务切换模块是怎么实现的呢？

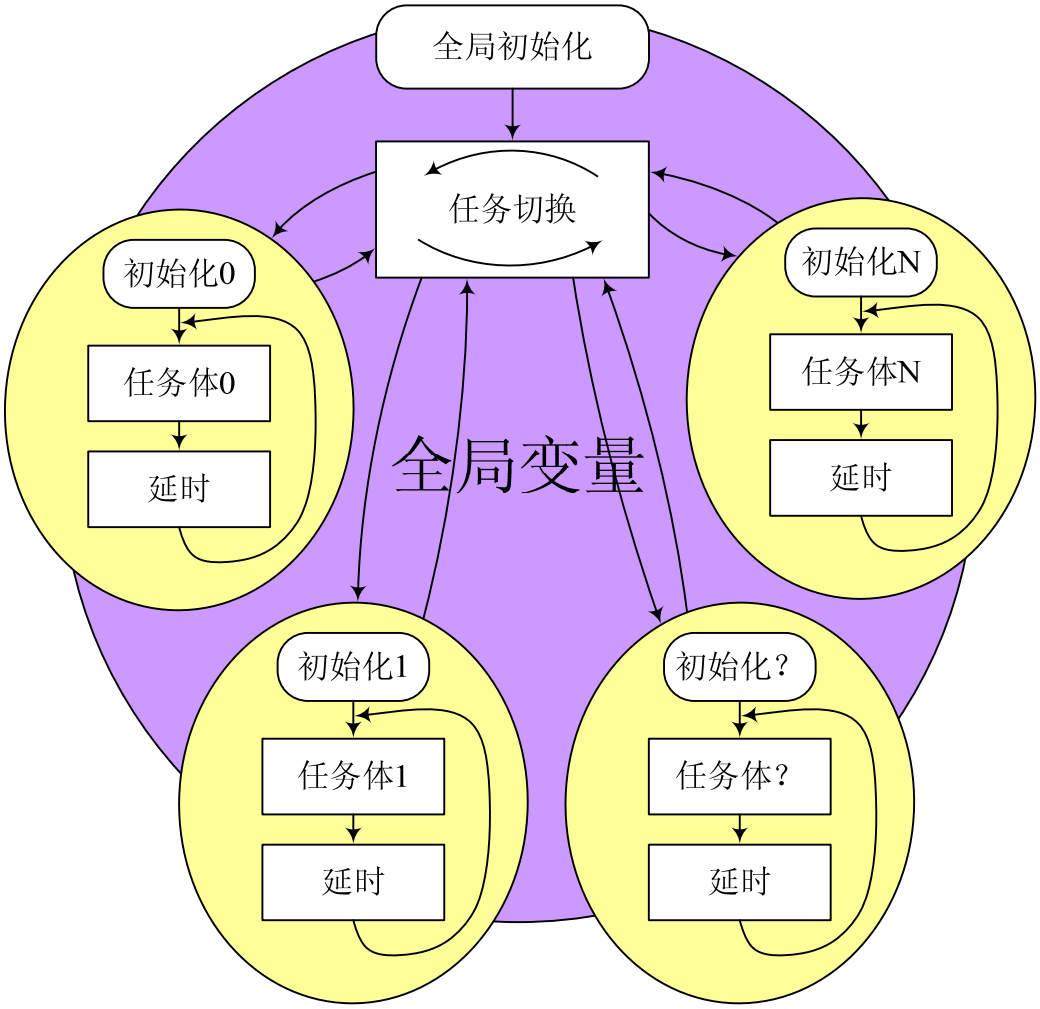

图3 多任务复杂流程图

首先需要理解，CPU 一旦运行起来，就无法停止（硬件支持时钟停止的不在这里讨论），

谁能够控制一批脱缰的马呢？对了，有中断，中断能够让 CPU 回到特定的位置，设想，能不能用一个定时中断，周期性的将 CPU 这匹运行着的脱缰的马召唤回来，重新给它安排特定的任务，事实上，任务切换就是这样实现的。

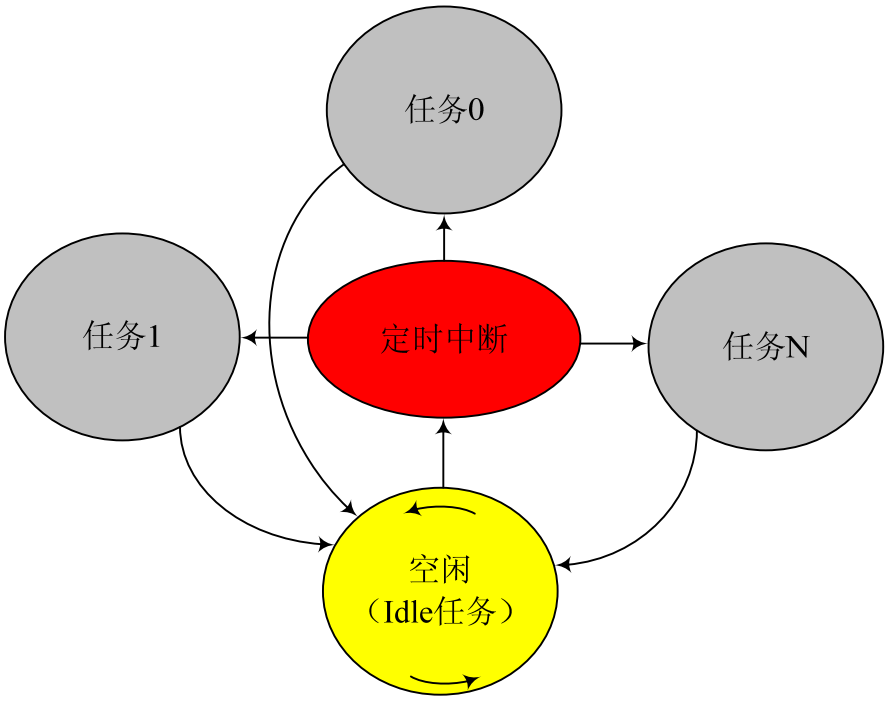

图 A

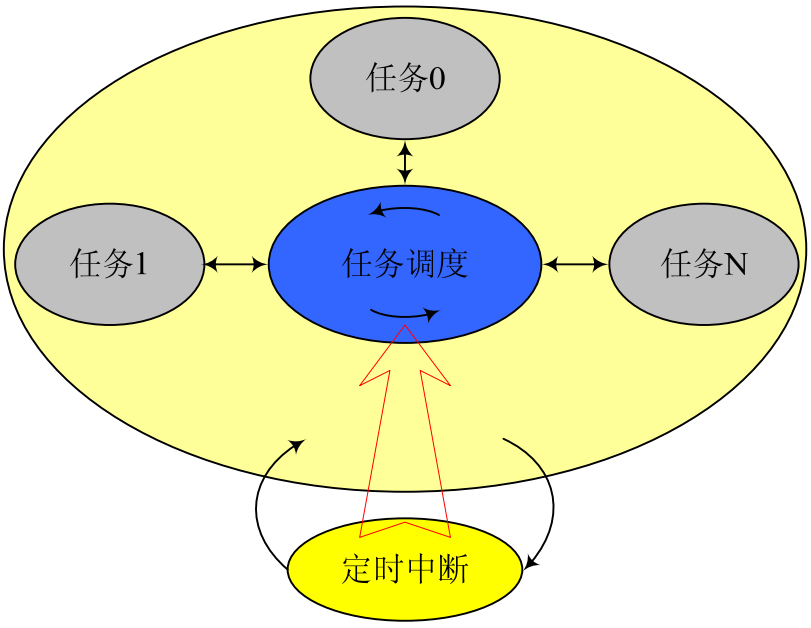

图 B

图 4 定时中断实现任务切换

如图 4A 所示，CPU 在空闲任务循环等待，定时中断将 CPU 周期性唤回，根据任务设计了不同的响应频度，满足条件的任务将获得 CPU 资源，CPU 为不同任务“关照”完成后，再次返回空闲任务，如此周而复始，对于各个任务而言，好像各自拥有一个独立的 CPU，各自

独立运行。用这种思想构建的程序框架，最大的好处是任务很容易裁剪，系统能够做得很复杂。

在充分考虑单片机中断特性（在哪里中断就返回到哪里）后，实际可行的任务切换如图 4B 所示，定时中断可能发生在任务调度，随机任务执行的任何时候，图中最大的框框所示，不管中断在何时发生，它都会正常返回，定时中断所产生的影响只在任务调度模块起作用，即依次让不同的任务按不同的节拍就绪。任务调度会按一定的优先级执行就绪任务。

总结不同的任务需要 CPU 关照的频度，选择最快的那个频度来设定定时器中断的节拍，一般选择 200Hz，或者 100Hz 都可以。另外再给每个任务设定一个节拍控制计数器 C，也就是定时器每中断多少次后执行任务一次。例如取定时中断节拍为 200Hz，给任务设定的 C=10，则任务执行频度为 200/10=20Hz，如果是数码管扫描，按 40Hz 不闪烁规律，则任务节拍控制计数器 C=5 即可。在程序设计中，C 代表着任务运行的节拍控制参数，我们习惯用 delay 来描述，不同的任务用 task0，task1……来描述。

明天继续写如何用代码实现！2009-6-29

下面我们来用代码实现以上多任务程序设计思想。

首先是任务切换

```
while(1)
{
    if(task_delay[0]==0)   task0();     //task0 就绪，
    if(task_delay[1]==0)   task1();     //task1 就绪，
    ……
}
```

很显然，执行任务的条件是任务延时量 task_delay=0，那么任务延时量谁来控制呢？定时器啊！定时器中断对任务延时量减一直到归零，标志任务就绪。当没有任务就绪时，任务切换本身就是一个 Idle 任务。

```
void timer0(void) interrupt 1
{
    if(task_delay[0]) task_delay[0]--;
    if(task_delay[1]) task_delay[1]--;
    ……
}
```

例如 timer0 的中断节拍为 200Hz，task0_delay 初值为 10，则 task0()执行频度为 200/10=20Hz。

有了以上基础，我们来设计一个简单多任务程序，进一步深入理解这种程序设计思想。任务要求：用单片机不同 IO 脚输出 1Hz，5Hz，10Hz，20Hz 方波信号，这个程序很短，将直接给出。

```
#include "reg51.h"
#define TIME_PER_SEC 200      //定义任务时钟频率，200Hz
#define CLOCK 22118400        //定义时钟晶振，单位 Hz
```

```
#define MAX_TASK 4                //定义任务数量

extern void task0(void);   //任务声明
extern void task1(void);
extern void task2(void);
extern void task3(void);

sbit f1Hz    = P1^0;   //端口定义
sbit f5Hz    = P1^1;
sbit f10Hz = P1^2;
sbit f20Hz = P1^3;

unsigned char task_delay[4];  //任务延时变量定义

//定时器 0 初始化
void timer0_init(void)
{
    unsigned char i;
    for(i=0;i<MAX_TASK;i++) task_delay[i]=0;    //任务延时量清零
    TMOD = (TMOD & 0XF0) | 0X01;                //定时器 0 工作在模式 1，16Bit 定时器模
式
    TH0 = 255-CLOCK/TIME_PER_SEC/12/256;
    TL0 = 255-CLOCK/TIME_PER_SEC/12%256;
    TR0 =1;
    ET0 =1;                                     //开启定时器和中断
}

//  系统 OS 定时中断服务
void timer0(void) interrupt 1
{
    unsigned char i;
    TH0 = 255-CLOCK/TIME_PER_SEC/12/256;
    TL0 = 255-CLOCK/TIME_PER_SEC/12%256;
    for(i=0;i<MAX_TASK;i++) if(task_delay[i]) task_delay[i]--;
     //每节拍对任务延时变量减 1 ， 减至  0  后，任务就绪。
}

/*main 主函数*/
void main(void)
{
    timer0_init();
    EA=1;//开总中断
    while(1)
    {
```

```
            if(task_delay[0]==0) {task0(); task_delay[0] = TIME_PER_SEC/ 2;}
            //要产生 1hz 信号，翻转周期就是 2Hz，以下同
            if(task_delay[1]==0) {task1(); task_delay[1] = TIME_PER_SEC/10;}
            //要产生 5hz 信号，翻转周期就是 10Hz，以下同
            if(task_delay[2]==0) {task2(); task_delay[2] = TIME_PER_SEC/20;}
            if(task_delay[3]==0) {task3(); task_delay[3] = TIME_PER_SEC/40;}
        }
    }

    void task0(void)
    {
        f1Hz = !f1Hz;
    }

    void task1(void)
    {
        f5Hz = !f5Hz;
    }

    void task2(void)
    {
        f10Hz = !f10Hz;
    }

    void task3(void)
    {
        f20Hz = !f20Hz;
    }
```

仿真效果如图 5 所示。

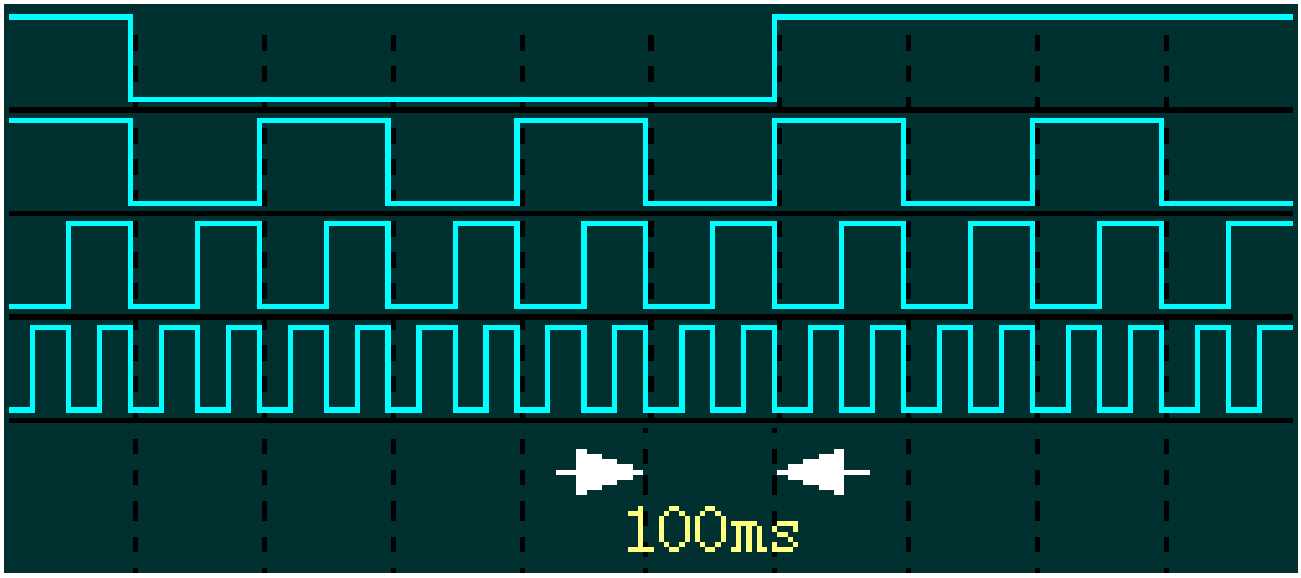

图 5 仿真波形图

同样的程序，同学们可以考虑用图 2 所示的思想设计，看看容易不容易，如果你的程序实现了相同的功能，如果我改变要求，改变信号的频率，你的程序容易修改吗？

要进一步完善这种程序设计思想，有几个问题还需要考虑：

对任务本身有什么要求？

不同任务之间有没有优先级？（不同的事情总有个轻重缓急吧!）

任务间如何延时？

……

为了回答这些问题，下面我们来分析 CPU 的运行情况。

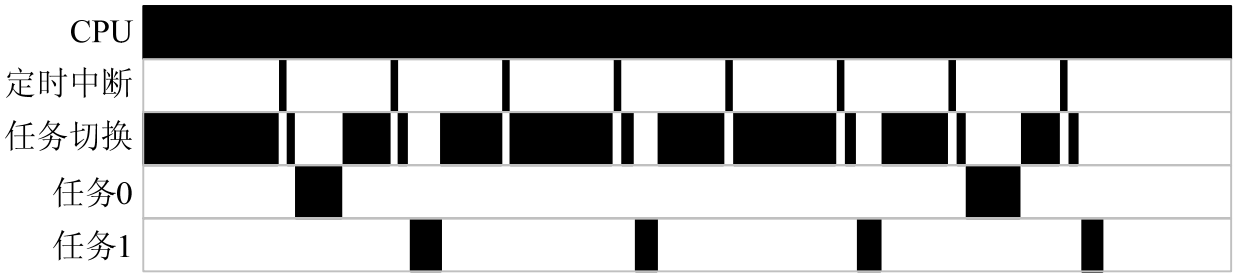

图 6 CPU 运行情况示意图

CPU 运行情况如图 6 所示，黑色区域表示 CPU 进程，系统启动后，CPU 将无休止的运行，CPU 资源将如何分配呢？程序首先进入“任务切换”进程，如果当前没有任务就绪，就在任务切换进程循环（也可以理解为空闲进程），定时中断将 CPU 当前进程打断，在定时中断进程可能让某些任务就绪，中断返回任务切换进程，很快会进入就绪任务 0，CPU“关照”完任务 0，再次回到任务切换进程，如果还有其它任务就绪，还会再次进入其它任务，没有任务就循环等待，定时中断会不断让新的任务就绪，CPU 也会不断进入任务“关照”。这样不同的任务就会获得不同的 CPU 资源，每一个任务都像是拥有一个独立的 CPU 为之服务。

从这种进程切换我们可以看出，在定时中断和任务切换过程中，额外的占用了一些 CPU 资源，这就是定时中断频度不宜太快，否则将大大降低 CPU 的有效资源率，当然太慢也不行。另外就是 CPU 每次关照任务的时间不能太长，如果超过一个中断周期，就会影响到其它任务的实时性。所谓的实时性就是按定时中断设定的节拍，准时得到 CPU 关照。这样，每一个子任务就必须简单，每次“关照”时间最好不要超过定时中断节拍周期（5ms 或 10ms，初学者要对 ms 有一个概念，机器周期为 us 级的单片机，1ms 可以执行上千条指令，对于像数码管扫描，键盘扫描，LCD 显示等常规任务都是绰绰有余的，只是遇到大型计算，数据排序就显得短了）

关于任务优先级的问题：一个复杂系统，多个任务之间总有“轻重缓急”之区别，那些需要严格实时的任务通常用中断实现，中断能够保证第一时间相应，我们这里讨论的不是那种实时概念，是指在最大允许时差内能够得到 CPU“关照”，例如键盘扫描，为了保证较好的操作效果，快的/慢的/长的/短的（不同人按键不一样）都能够正确识别，这就要保证足够的扫描速度，这种扫描速度对不同的按键最好均等，如果我们按 50Hz 来设计，那么就要保证键盘扫描速度在任何情况下都能够做到 50Hz 扫描频度，不会因为某个新任务的开启而被

破坏，如果确实有新的任务有可能破坏这个 50Hz 扫描频度，我们就应该在优先级安排上让键盘扫描优先级高于那个可能影响键盘扫描的任务。这里体现的就是当同时多个任务就绪时，最先执行哪个的问题，任务调度时要优先执行级别高的任务。

关于“长”任务的问题：有些任务虽然很独立，但完成一次任务执行需要很长时间，例如 DS18B20，从复位初始化到读回温度值，最长接近 1s，这主要是 DS18B20 温度传感器完成一次温度转换需要 500 到 750ms，这个时间对 CPU 而言，简直是太长了，就像一件事情需要我们人等待 10 年一样，显然这样的任务是其它任务所耽搁不起的。像类似 DS18B20 这样的器件（不少 ADC 也是这样），怎么设计任务体解决“长”的问题。进一步研究这些器件发现，真正需要 CPU“关照”它们的时间并不长，关键是等待结果要很长时间。解决的办法就是把类似的器件驱动分成多个段：初始化段、启动段、读结果段，而在需要花长时间等待时间段，不要 CPU 关照，允许 CPU 去关照其它任务。

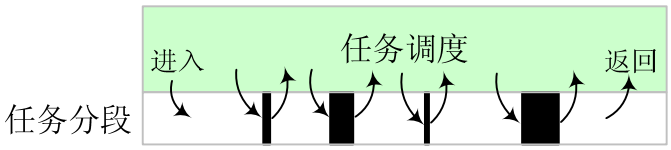

图 7 任务分段

任务分段如图 7 所示，将一个任务分成若干段，确保每段需要 CPU 关照时长小于定时器中断节拍长，这样 CPU 在处理这些长任务时，就不会影响到其它任务的执行。

Easy51RTOS

正是基于以上程序设计思想，总结完善后提出一种耗费资源特别少并且不使用堆栈的多线程操作系统，这个操作系统以纯 C 语言实现，无硬件依赖性，需要单片机的资源极少。起名为 Easy51RTOS，特别适合初学者学习使用。有任务优先级，通过技巧可以任务间延时，缺点是高优先级任务不具有抢占功能，一个具有抢占功能的操作系统，一定要涉及到现场保护与恢复，需要更多的 RAM 资源，涉及到堆栈知识，文件系统将很复杂，初学者学习难度大。

为了便于初学者学习，将代码文件压缩至 4 个文件。

Easy51RTOS.Uv2	Keil 工程文件，KEIL 用户很熟悉的
main.c	main 函数和用户任务 task 函数文件
os_c.c	Easy51RTOS 相关函数文件
os_cfg.h	Easy51RTOS 相关配置参数头文件

文件解读如下：

os_cfg.h

```
#include "reg51.h"
#define TIME_PER_SEC 200      //定义任务时钟频率，200Hz
#define CLOCK 22118400        //定义时钟晶振，单位 Hz
#define MAX_TASK 4            //定义任务数量
```

```
//函数变量声明，在需要用以下函数或变量的文件中包含此头文件即可
extern void task0(void);
extern void task1(void);
extern void task2(void);
extern void task3(void);
extern unsigned char task_delay[MAX_TASK];
extern void run(void (*ptask)());
extern void os_timer0_init(void);
```

os_c.c

```
#include "os_cfg.h"
unsigned char task_delay[MAX_TASK];   //定义任务延时量变量
//定时器 0 初始化
void os_timer0_init(void)
{
    unsigned char i;
    for(i=0;i<MAX_TASK;i++) task_delay[i]=0;
    TMOD = (TMOD & 0XF0) | 0X01;    //定时器 0 工作在模式 1，16Bit 定时器模式
    TH0 = 255-CLOCK/TIME_PER_SEC/12/256;
    //CRY_OSC,TIME_PER_SEC 在 os_cfg.h 中定义
    TL0 = 255-CLOCK/TIME_PER_SEC/12%256;
    TR0 =1;
    ET0 =1;                                    //开启定时器和中断
}

//  系统 OS 定时中断服务
void os_timer0(void) interrupt 1
{
    unsigned char i;
    TH0 = 255-CLOCK/TIME_PER_SEC/12/256;
    TL0 = 255-CLOCK/TIME_PER_SEC/12%256;
    for(i=0;i<MAX_TASK;i++) if(task_delay[i]) task_delay[i]--;
    //每节拍对任务延时变量减 1 ，减至 0 后，任务就绪。
}

//指向函数的指针函数
void run(void (*ptask)())
{
   (*ptask)();
}
```

main.c

```
#include "os_cfg.h"
```

```
#define TASK_DELAY0 TIME_PER_SEC/1          //任务执行频度为 1Hz
#define TASK_DELAY1 TIME_PER_SEC/2          //任务执行频度为 2Hz
#define TASK_DELAY2 TIME_PER_SEC/10         //任务执行频度为 10Hz
#define TASK_DELAY3 TIME_PER_SEC/20         //任务执行频度为 20Hz
void (* code task[])() = {task0,task1,task2,task3};   //获得任务 PC 指针

sbit LED0 = P1^0;    //演示用 LED 接口定义
sbit LED1 = P1^1;
sbit LED2 = P1^2;
sbit LED3 = P1^3;

/*main 主函数*/
void main(void)
{
    unsigned char i;
    os_timer0_init();       //节拍发生器定时器初始化
    EA = 1;                 //开总中断

    while(1)
    {
        for(i=0;i<MAX_TASK;i++)
            if (task_delay[i]==0) {run(task[i]); break;} //就绪任务调度
    }    //上一行 break 有特殊作用，详细解释见后文
}

void task0(void) //任务 0
{
    LED0 = !LED0;
    task_delay[0] = TASK_DELAY0;
}

void task1(void) //任务 1
{
    LED1 = !LED1;
    task_delay[1] = TASK_DELAY1;
}

void task2(void) //任务 2
{
    LED2 = !LED2;
    task_delay[2] = TASK_DELAY2;
}

void task3(void) //任务内分段设计
```

```
{
    static unsigned char state=0;   //定义静态局部变量
    switch (state)
    {
        case 0:
            LED3 = !LED3;
            state = 1;
            task_delay[3] = TASK_DELAY3;
            break;

        case 1:
            LED3 = !LED3;
            state = 2;
            task_delay[3] = TASK_DELAY3*2;
            break;

        case 2:
            LED3 = !LED3;
            state = 0;
            task_delay[3] = TASK_DELAY3*4;
            break;

        default:
            state = 0;
            task_delay[3] = TASK_DELAY3;
            break;
    }
}
```

仿真图如图 8 所示

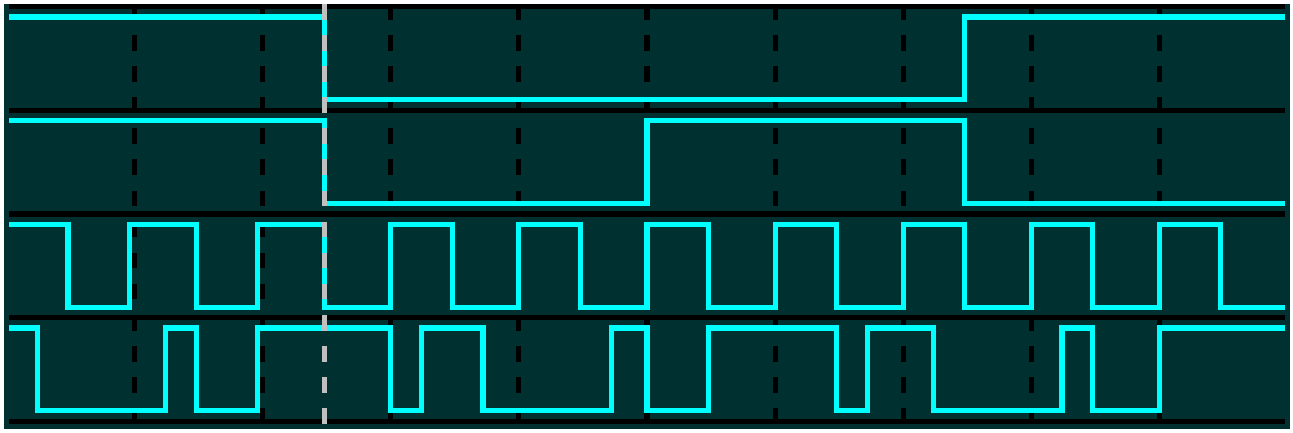

图 8 仿真波形图

主程序巧妙实现优先级设定：

```
    for(i=0;i<MAX_TASK;i++)
```

```
            if (task_delay[i]==0) {run(task[i]); break;} //就绪任务调度
```

这里的 break 将跳出 for 循环，使得每次重新任务调度总是从 task0 开始，就意味着优先级高的任务就绪会先执行。这样 task0 具有最高优先级，task1、task2、task3 优先级依次降低。

特别是 void task3(void)用 switch(state)状态机实现了任务分段，这也是任务内系统延时的一种方法。

田开坤 2009-6-30

后记：希望同学们参考我的这种程序设计思想，完成以下系统软件设计。

数控直流稳压电源：系统结构框图如图 9 所示（并非原理框图），整合这些资源，完成软硬件设计。

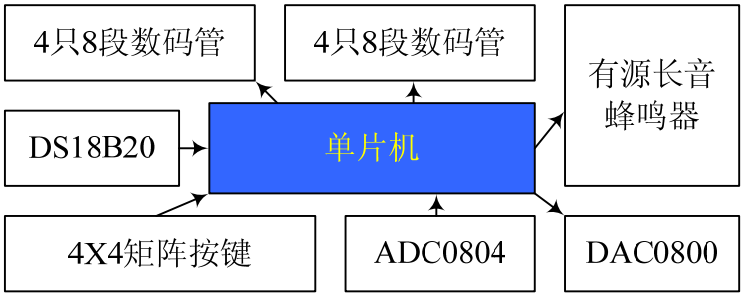

图 9 数控稳压电源结构框图

要求如下：

1．8 只数码管只能动态扫描方式驱动，4 位显示电压设定值，4 位显示实际测量值；

2．DS18B20 用来测量关键器件温度，超过 70 度，关闭输出并报警：蜂鸣器发声提示；

3．矩阵键盘要用动态扫描方式驱动，每次按键有蜂鸣器发声提示；

4．ADC+DAC 实现闭环控制（AD 测量值等于设定值，不等则修正 DA 输出）；

5．使用具有 EEPROM 的单片机，能够存储当前设定状态；

6．尽量减少硬件成本，尽量用复杂的软件代替硬件（例如数码管不允许用专用芯片）。