Proteus 单片机仿真中的 μC/OS－II 移植※

■桂林电子科技大学　徐模辉

摘　要　针对 51 系列单片机的特点,介绍嵌入式实时操作系统内核 μC/OS - II 在 51 系列单片机上的移植环境和条件;详细阐述在 Keil uVision3 开发环境下的移植步骤和移植过程;充分利用 51 系列单片机自身的硬件资源,结合外扩部分其他硬件,用 Proteus 软件成功进行了系统移植的硬件仿真;分析移植测试实验情况,验证 μC/OS - II 的实时性、稳定性和安全性。

关键词　Proteus　嵌入式实时操作系统　μC/OS - II　移植　单片机仿真

引　言

由于技术的进步,单片机系统硬件的规模越来越大,功能越来越强,为了对整个系统及其所操作的部件、装置等资源进行统一协调、指挥和有效控制,在嵌入式系统中运用操作系统是非常必要的。而操作系统是一个通用的程序,要在自己的嵌入式系统中应用操作系统,必须根据所用 CPU 的不同来进行移植。本文将具体论述嵌入式实时多任务操作系统 μC/OS - II 在 51 系列单片机上的移植,并介绍在 Proteus 软件下成功进行的系统仿真。

1　μC/OS - II 介绍

μC/OS - II 是由美国人 Jean J. Labrosse 编写的一个实时多任务操作系统。其在 2000 年得到了美国联邦航空管理局对用于商用飞机的符合 RTCA DO - 178B 标准的认证,证明 μC/OS - II 具有足够的稳定性和安全性[1]。

μC/OS - II 的移植对 CPU 芯片的要求[2]:①CPU 支持中断,并能产生定时中断;②CPU 支持一定容量的硬件堆栈;③CPU 有将堆栈指针和其他寄存器读出和存储到堆栈或内存的指令;④编译器能产生可重入代码;⑤用 C 语言就可以打开、关闭中断。本设计采用 51 系列单片机,开发环境采用 Keil 公司的 Keil uVision3,基本满足上述要求。最后系统在 Proteus 6.9SP4 软件下仿真。

2　Proteus 仿真软件介绍

Proteus 软件是来自英国 Labcenter electronics 公司的 EDA 工具软件。Proteus 软件除了具有与其他 EDA 工具一样的原理布图、PCB 自动或人工布线及电路仿真的功能外,其突破性的功能是,它的电路仿真是互动的;针对微处理器的应用,还可以直接在基于原理图的虚拟原型上编程,并实现软件源码级的实时调试;如有显示及输出,配合系统配置的虚拟仪器(如示波器、逻辑分析仪等),还能看到运行后输入/输出的效果。因此 Proteus 建立了完备的电子设计开发环境。

Proteus 产品系列还包含了革命性的 VSM 技术,用户可以对基于微控制器的设计连同所有的周围电子器件一起仿真;甚至可以实时采用诸如 LED/LCD、键盘、RS232 终端等动态外设模型来对设计进行交互仿真。最新版支持非常丰富的仿真元件(共 7 000 多种),还有很多第三方模型,如 MMC 卡、以太网卡、ATA 硬盘、麦克风等。

3　μC/OS－II 移植过程

μC/OS - II 的绝大多数代码都是用 C 语言编写的。一般情况下,这部分代码不需要修改就可以使用,因此它的移植工作主要与 4 个文件相关:汇编文件(OS_CPU_A.ASM)、处理器相关 C 语言文件(OS_CPU.H、OS_CPU_C.C)和配置文件(OS_CFG.H)。

3.1　开发环境设置

由于 8051 单片机片内 ROM 和 RAM 资源有限,需扩展片外 ROM 和 RAM;在 Keil uVision3 编译时,需对其进行一些设计,以符合实际设计电路的要求。

①内存模式选择大模式(Large: variables in XDATA)使用外部 RAM,16 位间接寻址。在 Off-chip Xdata memory 的 Ram 框中 Start 设置为 0x0080,Size 为 0x1F40(外扩 RAM 大小为 32 KB)程序存储器也选择大模式(Large: 64 KB program),并选中 Use On-chip ROM (0x0

～0xFFF)表示使用片上 ROM。在 Off-chip Code memory 的 Eprom 框中 Start 设置为 0x1000,Size 为 0x1F40(外扩 ROM 大小为 32 KB)。

②定义 C51 优化级别。利用其最高级别优化:"9: Common Block Subroutines"。采用此优化后的代码运行速度也许会变慢,但此种优化能够显著提高代码密度。这对于片内 ROM 小的 51 系列单片机是非常有帮助的。

3.2 改写文件 OS_CPU.H

(1) 堆栈的增长方向

51 单片机的堆栈地址是从低向高地址增长(由下往上)的,所以将堆栈增长方向的常数 OS_STK_GROWTH 定义为 0,即

```
# define OS_STK_GROWTH 0
```

(2) 定义临界段的宏

设置临界区的两个宏分别直接使用 51 单片机的开中断和关中断指令来实现。

```
# define OS_ENTER_CRITICAL()   EA = 0
# define OS_EXIT_CRITICAL()    EA = 1
```

(3) 定义任务切换宏

因为 MCS-51 没有软中断指令,所以用程序调用代替。两者的堆栈格式相同,RETI 指令复位中断系统,RET 则没有。实践表明,对于 MCS-51,用子程序调用入栈,用中断返回指令 RETI 出栈是没有问题的;反之,中断入栈 RET 出栈则不行。

```
# define   OS_TASK_SW()   OSCtxSw()
```

(4) 定义数据类型

由于 bit 类型为 C51 特有,不能用在结构体里,所以 BOOLEAN 要定义成 unsigned char 类型。其他数据类型按照 C51 中的数据类型定义就可以了。另外,"pdata"、"data"在 μC/OS 中用作一些函数的形参,但它同时又是 Keil C51 的关键字,会导致编译错误,因此可以把"pdata"改成"ppdata","data"改成"ddata"。

3.3 改写文件 OS_CPU_C.C

在文件 OS_CPU_C.C 中,主要应该改写任务堆栈初始化函数 OSTaskStkInit()。由于要用单片机上的定时器为系统设置时钟中断,因而还需添加对 51 系列单片机定时器的初始化程序。

(1) 改写任务堆栈初始化函数

TCB 结构体中 OSTCBStkPtr 总是指向用户堆栈最低地址。该地址空间内存放用户堆栈长度,其上空间存放系统堆栈映像,即用户堆栈空间大小 = 系统堆栈空间大小 + 1。SP 总是先加 1 再存数据,因此,SP 初始时指向系统堆栈起始地址(OSStack)减 1 处(OSStkStart)。很明显系统堆栈存储空间大小等于 SP - OSStkStart。用户堆栈初始化时从下向上依次保存:用户堆栈长度(15)、PCL、PCH、PSW、ACC、B、DPL、DPH、R0、R1、R2、R3、R4、R5、R6、R7。不保存 SP,任务切换时根据用户堆栈长度计算得出。OSTaskStkInit 函数总是返回用户栈最低地址。

(2) 系统时钟初始化

操作系统 tick 时钟使用了 51 单片机的 T0 定时器。它的初始化代码如下:

```
void InitTimer0(void) reentrant{
    TMOD = TMOD &0xF0;
    TMOD = TMOD| 0x01;
                //模式 1(16 位定时器),仅受 TR0 控制
    TH0 = 0x70; //定义 Tick = 50 次/s(即 0.02 s/次)
    TL0 = 0x00; //OS_CPU_A.ASM 和 OS_TICKS_PER_SEC
    ET0 = 1;    //允许 T0 中断
    TR0 = 1;
}
```

3.4 改写文件 OS_CPU_A.ASM

OS_CPU_A.ASM 的改写是移植的难点,它需要用户主要编写 4 个汇编语言函数:OSStartHighRdy()、OSCtxSw()、OSIntCtxSw()、OSTickISR()。由于 OSTickISR()用中断定时器子程序来代替,因此实际上只需编写 3 个函数:初始任务调度函数 OSStartHighRdy()、任务级任务切换函数 OSCtxSw()和中断级任务切换函数 OSIntCtxSw()。

3.5 改写文件 OS_CFG.H

这个文件主要是对系统进行裁剪,对相关的配置常量进行相应的设置。其示意代码如下:

```
# define MaxStkSize       100
# define OS_MAX_EVENTS    2
        //事件控制块的最大数量,最小应为 2
# define OS_MAX_MEM_PART   2
        //内存控制块的最大数量,最小应为 2
# define OS_MAX_QS 2
        //消息队列最大数量,最小应为 2
# define OS_MAX_TASKS   11       //任务最大数
# define OS_LOWEST_PRIO 12       //最低优先级
# define OS_TASK_IDLE_STK_SIZE MaxStkSize
        //堆栈容量
# define OS_TASK_STAT_EN   0
        //是否使用统计任务,1 是 0 否
# define OS_TASK_STAT_STK_SIZE MaxStkSize
```

```
        //统计任务堆栈容量
# define OS_CPU_HOOKS_EN 1
        //是否实现钩子函数,1 是 0 否
# define OS_MBOX_EN 0
        //是否使用消息队列,1 是 0 否
# define OS_MEM_EN 0
        //是否使用内存管理函数,1 是 0 否
# define OS_Q_EN    0
        //是否使用消息队列函数,1 是 0 否
# define OS_SEM_EN 0
        //是否使用型号量管理函数,1 是 0 否
# define OS_TASK_CHANGE_PRIO_EN    0
        //是否实验改变任务优先级函数,1 是 0 否
# define OS_TASK_CREATE_EN 1
        //是否使用 OSTaskCreate()函数
# define OS_TASK_CREATE_EXT_EN 0
        //是否使用 OSTaskCreateExt()函数
# define OS_TASK_DEL_EN    0
        //是否使用删除任务函数
# define OS_TASK_SUSPEND_EN    0
        //是否使用任务挂起和唤醒函数
# define OS_TICKS_PER_SEC      50
        //调用 OSTimeTick()函数的次数
```

3.6 编写应用程序

这里主要创建 3 个任务,任务 1: 串口发送“The first task is working: P1 = 0xAA”出去,同时单片机 P1 口输出 0xAA,编号为奇数的 LED 点亮,偶数的 LED 不亮。该任务的延时时间为 1 s。在首次启动单片机时,任务 1 先会发送 1 行“**********”,接着发 1 行“* Hello! The world. *”,再发 1 行“**********”。任务 2:串口发送“The second task is woring: P1 = 0x00”,同时单片机 P1 口输出 0x00,所有 LED 都点亮。该任务延时时间 3 s。任务 3: 串口发送“The third task is woring: P1 = 0xff”,同时单片机 P1 口输出 0xff,所有 LED 都不亮。该任务延时时间 6 s。在调度这些任务前,系统首先执行初始化函数 OSInit()、时钟初始化函数 InitTimer0(),接着对单片机串口进行初始化。

4 仿真验证

4.1 Keil uVision3 调试

在 Keil uVision3 中可以对单片机本身进行一些单片机内部资源的模拟调试。整个程序经 Keil uVision3(版本 8.06)编译后,代码量为 7.3 KB,可直接在 Keil uVision3 上仿真运行,结果如图 1 所示。任务 1 每秒显示 1 次,任务 2 每 3 s 显示 1 次,任务 3 每 6 s 显示 1 次。从显示结果看,显然是正确的。

4.2 Proteus 仿真

Proteus 不仅可以对单片机内部资源传真而且可以对其他外围硬件进行仿真,其仿真结果一般都与硬件实际效果基本一致。首先启动 Proteus(使用版本为 6.9SP4)软件,新建一个设计并添加相应元件,连接电路如图 2 所示。单片机选用 AT89C52,晶振频率选用 22.118 4 MHz。由于单片机内部的 ROM 和 RAM 有限,对于移植 μC/OS-II 还不足以满足要求。所以外扩了 1 片 32 KB 的 RAM 和 ROM。设置好各元件的参数后,将 Keil 编译好的 HEX 文件添加到单片机中就可以启动仿真了。设置好串口终端的波特率。运行仿真,串口显示的数据和 Keil uVision3 仿真结果一样,同时 P1 口所接的 LED 不断地闪烁。说明系统已经运行,且结果和我们预期的一样,系统移植成功。

5 问题探讨

5.1 可重入性问题

由于 μC/OS-II 是一个可抢占式内核,因此系统中的绝大多数函数都应该是可重入的。而在 Keil C51 编译器中,在函数定义时的默认值都是不可重入的,因此需要在系统中的每一个函数的声明以及定义处都加上“large reentrant”的修饰符,以保证函数的可重入性[3]。为了函数重入,形参和局部变量必须保存在堆栈里,由于 51 系列单片机硬件堆栈太小,Keil 将根据内存模式在相应内存空间仿真堆栈(生长方向由上向下,与硬件栈相反)。对于大模式编译,函数返回地址保存在硬件堆栈里,形参和局部变量放在仿真堆栈中,栈指针为 ?C_XBP,XBPSTACK = 1 时,起始值在 startup.a51 中初始化为 FFFFH + 1。仿真堆栈效率是非常低的,但为了重入操作必须使用。Keil C51 可以混合使用 3 种仿真堆栈(大、中、小模式),为了提高效率,针对 51 系列单片机可以使用大模式编译。

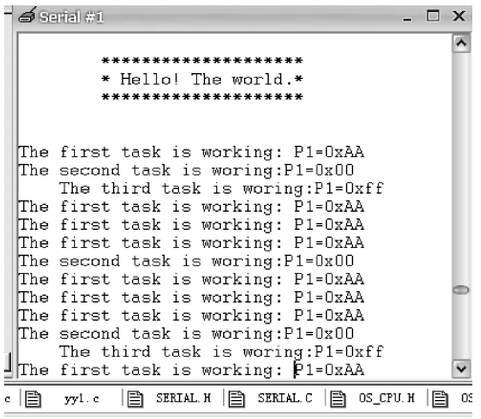

图 1 Keil uVision3 仿真结果

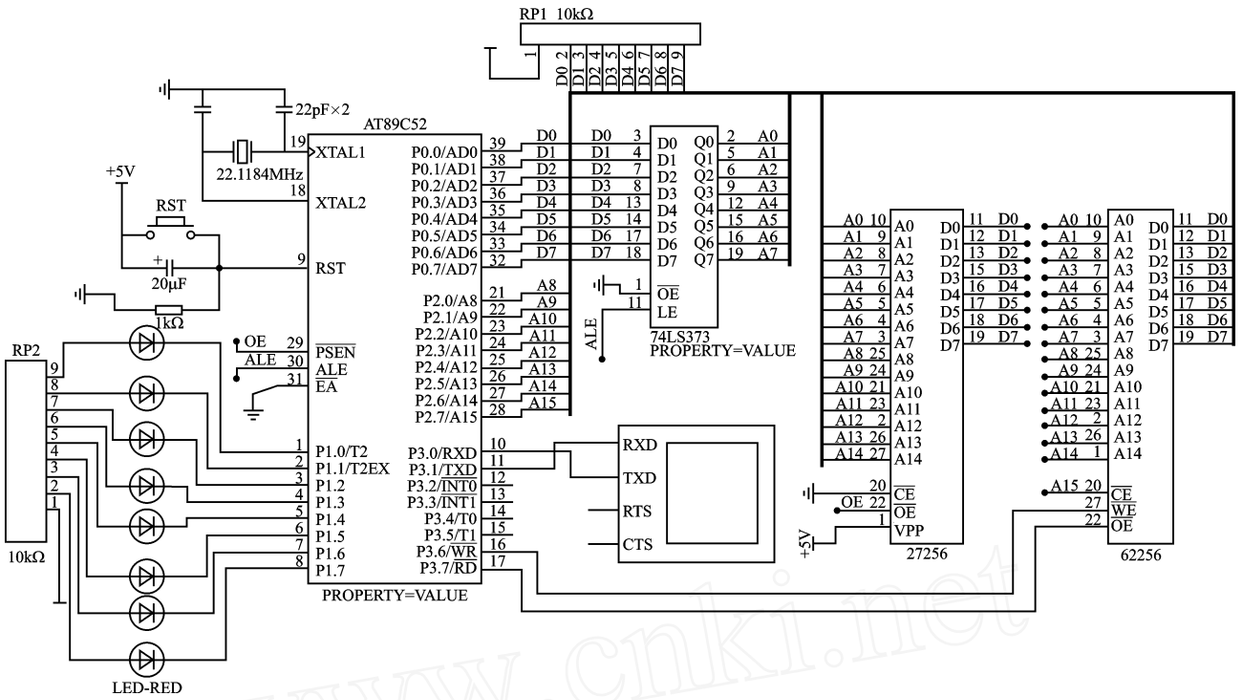

图 2　Proteus 仿真电路

5.2　堆栈长度的确定

堆栈总是越长越好,但现实是不允许的。用户堆栈空间的大小是可以精确计算出来的。用户堆栈空间 = 硬件堆栈空间 + 仿真堆栈空间。对 μC/OS - II 来说,不同任务使用不同空间效果会更好。但为了在 51 系列单片机上有效实现任务重入,针对 51 系列单片机使用统一的固定大小的堆栈空间更好。综合考虑,硬件堆栈空间大小定成 64 字节。仿真堆栈大小取决于形参和局部变量的类型及数量,可以精确算出。因为所有用户栈使用相同空间大小,所以取占用空间最大的任务函数的空间大小为仿真堆栈空间大小。这样用户堆栈空间大小就唯一确定了。由于 51 系列单片机的 SP 只有 8 位,无法在 64 KB 空间中自由移动,只好采用拷贝全部硬件堆栈内容的办法。这样就大大占用了 CPU,使效率降低。切换频率决定了 CPU 的耗费,频率越高耗费越大。在耗费无法避免的情况下,可采取一些措施来提高效率:

- ret 和 reti 混用减少代码;
- IE、SP 不入出栈,通过另外方式解决;
- 用 IDATA 关键字声明在汇编中用到的全局变量,变 DPTR 操作为 Ri 操作;
- 设计堆栈结构,简化算法;
- 让串口输入/输出工作在系统态,不占用任务 TCB 和优先级,增加弹性缓冲区,减少等待。

临界区保护略 ——编者注。

结　语

μC/OS - II 在中小型工业领域中有良好的应用前景,采用 μC/OS - II 系统也更能对系统进行实时控制。本文通过在单片机的数据存储器中建立工作堆栈、仿真堆栈、任务堆栈及其附加段,实现了 μC/OS - II 嵌入式实时操作系统向 51 系列单片机的移植。采用 Proteus 软件对硬件系统进行仿真可以保证实际硬件电路设计的准确性,且可以缩短产品的开发周期。本文介绍了用 Proteus 软件对 μC/OS - II 嵌入式实时操作系统向 51 系列单片机的移植进行的成功仿真,并详细分析了系统移植过程中应注意的一些问题。此移植的成功为 μC/OS - II 嵌入式实时操作系统的复杂应用,提供了基本的条件。

编者注:本文为期刊缩略版,全文见本刊网站 www.mesnet.com.cn。

参考文献

[1] 任哲. 嵌入式实时操作系统 μC/OS - II 原理及应用[M]. 北京:北京航空航天大学出版社,2005.

[2] Labrosse. Jean J. 嵌入式实时操作系统 μC/OS - II[M]. 邵贝贝,译. 北京:北京航空航天大学出版社,2003.

[3] 董余红,阎俊武. μC/OS - II 在 51 系列单片机上的移植[J]. 导弹与航天运载技术,2004(6):39 - 40.

徐模辉(硕士研究生),主要研究方向为微电子学与固体电子学。

(收稿日期:2007-06-19)