

μC/OS-II 在 51 单片机上的移植

付伟林

(天津工业大学, 天津, 300160)

摘要: 研究嵌入式实时操作系统 μC/OS-II 在 51 单片机上的移植, 提出 μC/OS-II 任务的编写规范以及注意事项, 分析了它的移植技术, 并介绍了移植的详细步骤。

关键词: 嵌入式实时操作系统; μC/OS-II; 51 单片机; 内核; 移植

中图分类号: TP316 **文献标识码:** A **文章编号:** 1009-3044(2009)12-3265-02

Migration of μC/OS-II on 51 Single-Chip

FU Wei-lin

(Tianjin Polytechnic University, Tianjin 300160, China)

Abstract: Researches on the migration of embedded real-time operating system μC/OS-II on 51 single-chip, discusses the criterion and consideration of compiling μC/OS-II task.

Key words: embedded real time operating system; μC/OS-II; 51 single-chip; kernel; migration

1 引言

μC/OS-II 是一个开放式的内核, 它最主要的特点就是源码公开。它适用于小型控制系统, 具有执行效率高、占用空间小、实时性能优良和可扩展性强等特点。它是专为嵌入式系统设计的操作系统, 不论用于工程应用还是教学, 它都是非常优秀的。随着现代计算机技术的飞速发展, 嵌入式系统扮演了越来越重要的角色, 嵌入式系统设计日趋复杂。之所以选择将其移植到 51 单片机上, 是因为 51 单片机相对简单, 且成本较低, 相关资料齐全, 容易上手。

2 μC/OS-II 的简介

μC/OS-II 具有很强的可移植性, 可以广泛应用于各类 8 位、16 位、32 位微控制器或 DSP 中。它具有完全可剥夺型的实时内核, 其核心工作原理是让最高优先级的就绪任务处于运行状态, 它具有多任务的特点, 可以管理 64 个任务, 其中 56 个任务分配给用户, 另外它具有内核可裁减性、可确定性的特点, 并提供很多系统服务, 比如信号量、互斥信号量、事件标志、消息邮箱、消息队列、内存的分配和释放等。

3 μC/OS-II 的移植

3.1 μC/OS-II 移植条件

μC/OS-II 可以移植到很多不同类型单片机上。可以移植的单片机要求满足下列几个条件: 1) 处理器的 C 编译器能产生可重入代码; 2) 用 C 语言就可以打开和关闭中断; 3) 处理器支持中断, 并且能产生定时中断 (通常在 10 至 100Hz 之间); 4) 处理器支持能够容纳一定量的数据 (可能是几千字节) 的硬件堆栈; 5) 处理器有将堆栈指针和其他 CPU (8 位、16 位、32 位的处理器) 寄存器读出和存储到堆栈或内存 (RAM 或者 Flash) 中的指令。因为任务的切换是靠将 CPU 的寄存器的内容保存到要被切换出去的任务的堆栈中 (以便下次运行)。同时将要运行的任务的堆栈内容恢复到 CPU 寄存器中 (以便马上运行) 来实现。所以寄存器读出和存储到堆栈或内存中的指令不可少。51 单片机满足以上所有条件, 因此可以移植 μC/OS-II。

3.2 μC/OS-II 移植的步骤

简单来说, μC/OS-II 移植到 51 单片机上涉及到以下相关文件: 处理器相关 C 文件 (OS_CPU.H, OS_CPU_C.C)、汇编文件 (OS_CPU_A.ASM)。

移植的具体步骤如下:

1) 设置 OS_CPU.H 文件: 在 OS_CPU.H 文件里, 首先需要将 OS_STK 定义为 char 类型, 接着定义进入与退出临界区宏, 即将 OS_ENTER_CRITICAL() 定义为 EA=0, 将 OS_EXIT_CRITICAL() 定义为 EA=1。然后, 将 OS_STK_GROWTH 定义为 0, 最后将任务切换宏 OS_TASK_SW() 定义为 OSCtxSw(), 即完成了对 OS_CPU.H 文件的设置。

2) 设置 OS_CPU_C.C 文件: 移植 μC/OS-II 需要在 OS_CPU_C.C 中实现几个重要的函数, 其中 OSTaskStkInit() 用来初始化任务的堆栈。堆的大小可根据任务的实际情况自行确定。5 个系统钩子函数可直接定义它们为空函数。μC/OS-II 要求用户提供一个周期性的时钟源, 来实现时间的延时和超时功能。时钟节拍应该每秒发生 10-100 次。为了完成该任务, 本设计使用 51 的定时器 0 来提供时钟节拍, 因此 OS_CPU_C.C 文件中还包括定时器 0 的初始化函数。DisableINT()、EnableINT() 函数也在这个文件中实现。

```
void *OSTaskStkInit (void (*task)(void *pd), void *ppdata, void *ptos, INT16U opt) reentrant
{
    INT8 i;
    OS_STK *stk;
    ppdata = ppdata; // ppdata 未被用到保留此语句防止警告产生
```

```

opt = opt; //opt 没被用到保留此语句防止告警产生
*stk = (OS_STK *)ptos; //用户堆栈最低有效地址
*stk++ = 15; //用户堆栈长度
*stk++ = (INT16U)task & 0xFF; //任务地址低 8 位
*stk++ = (INT16U)task >> 8; //任务地址高 8 位
for(i=0;i<13;i++)
*stk ++ = 0x00; //PSW, ACC, B, DPL, DPH,R0—R7
return ((void *)ptos);
}

```

//初始化定时器 0

```
void InitTimer0(void) reentrant
```

```
{TMOD=10H;
```

```
T0=D8F0H;
```

```
TR0=1;}
```

```
void DisableINT()
```

```
{EA=0; //进入临界区关中断
```

```
INT_Count++; //全局变量关中断计数加 1}
```

```
void EnableINT()
```

```
{
```

```
EA=0; //进入临界区关中断
```

```
INT_Count- -; //全局变量关中断计数减 1
```

```
if(INT_Count==0)
```

```
EA=1; //开中断}
```

3)设置 OS_CPU_A.ASM 文件:

uC/OS-II 移植要求用户编写 4 个简单汇编语言函数:

- OSStartHighRdy //使就绪态任务中优先级最高的任务开始运行
- OSCtxSw //低优先级任务切换到高优先级任务
- OSTickISR //时钟节拍中断
- OSIntCtxSw //在 ISR 中执行任务切换功能

OSStartHighRdy 主要完成高优先级任务寄存器的恢复。必须恢复该任务在 CPU 使用权被剥夺时保留下来的全部寄存器的值,以便让这个高优先级任务能够继续运行。即把用户堆栈中的 PCL,PCH,PSW,ACC,B,DPL,DPH,R0,R1,R2,R3,R4,R5,R6,R7 全部拷贝到系统堆栈,然后系统堆栈再进行 POP 操作,将保存的这些值弹出到 CPU 的各个寄存器。使得此高优先级任务得到运行。其中部分关键代码如下:

```

MOVX A,@DPTR; //调整 DPTR,使其指向用户堆栈的最低地址
MOV R5,A; //R5=用户堆栈长度,因为堆栈长度放在用户堆栈最低处
MOV R0,#OSSktStart; //OSSktStart 为系统堆栈起始地址
restore_stack; //从用户堆栈到系统堆栈的 copy
INC DPTR
INC R0
MOVX A, @DPTR
MOV @R0, A
DJNZ R5, restore_stack
POPALL; //为自定义宏,将所有寄存器实现出栈

```

OSCtxSw 为任务切换函数,实现从低优先级任务到高优先级任务切换。要实现任务切换,一般包括两个过程:①保存当前低优先级任务的全部寄存器的值以及堆栈的长度值;②恢复高优先级任务以前在 CPU 使用权被剥夺时保存下来的全部寄存器的值,即前一个汇编程序 OSStartHighRdy 实现的功能。

其中部分关键代码如下:

以下代码完成第①步操作

```

PUSHALL; //用户定义的宏,将全部 CPU 寄存器保存到系统堆栈
MOV A, SP; //以下 3 句实现获得堆栈的长度
SUBB A, #OSSktStart
MOV R5,A; //R5 里保存堆栈长度,并将其保存在用户堆栈最底下
save_stack; //以下实现从系统堆栈到用户堆栈的拷贝
INC DPTR; //指向用户堆栈
INC R0; //指向系统堆栈
MOV A, @R0
MOVX @DPTR, A
DJNZ R5, save_stack;

```

而第②步代码和 OSStartHighRdy 代码一样,除此之外,将当前优先级最高任务的任务控制块指针 OSTCBHighRdy 赋值给当前任务控制块指针 OSTCBCur。

uC/OS-II 要求用户提供一个周期性的时钟源,来实现时间的延迟和超时功能。我们选用 8051 的 T0 定时器作为 tick 时钟。OSTickISR 函数为时钟节拍中断服务程序,当产生计时和定时中断时,执行此函数。此函数和其它中断服务子程序一样,都先要保存断点和保存现场,然后执行用户代码,最后对恢复保存的寄存器值。

其中关键代码:

```
CSEG AT 000BH; //0BH 为 0 号定时器 T0 的入口地址
```

(下转第 3274 页)

遍性,系统风险总量高的不代表某一个具体的资产的风险高。风险较大的原因有两种:一部分原因是由于某些系统资产面临的威胁发生可能性较大,而系统中存在相当多可被该威胁利用的安全漏洞或缺陷,如安全漏洞、密码猜测、拒绝服务等;另一种原因是面对范围较广,所有资产都存在该风险,导致整个系统风险值提高,如意外故障。

8 结束语

在一个全面的风险评估中,风险的所有重要因素都紧紧围绕着资产为中心,威胁、脆弱性以及风险都是针对资产而客观存在的。威胁利用资产自身的脆弱性使得安全事件的发生成为可能,从而形成了风险。这些安全事件一旦发生,将对资产甚至是整个系统都将造成一定的影响。因此资产的评估是风险评估的一个重要的步骤,它被确定和估价的准确性将影响着下一步所有因素的评估。

风险评估是根据资产所处的环境条件和资产以前遭受威胁损害的情况进行判断。确定威胁的主体和客体。最终获取了大量的威胁发生的频率或概率的第一手数据,以识别与选择适当和正确的风险控制策略,并采取相应的应付措施。

参考文献:

- [1] Cheswick,William R,Bellovin S M.Firewalls and Internet Security:Repelling the Wily Hacker[M].Addison-Wesley,1995.
- [2] 陈晓迪.信息化工程监理企业项目实施风险评价与对策研究[D].西安理工大学,2007.
- [3] 巫江,欧阳峰.企业信息化的风险管理探讨[J].企业经济,2006(2).

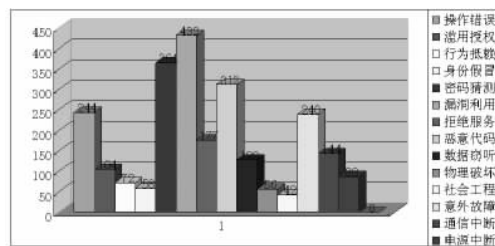


图2 风险分析表

(上接第3266页)

```

LJMP OSTickISR; //跳转到中断服务子程序
OSTickISR: PUSHALL; //保存所有寄存器的宏
CLR TR0; //后面4句为设置T0的工作方式以及赋值
MOV TH0,#70H; //定义Tick=50次/秒(即0.02秒/次)
MOV TL0,#00H; //OS_CPU_C.C和OS_TICKS_PER_SEC
SETB TR0
LCALL _? OSTimeTick; //调用系统Tick函数
LCALL _? OSIntExit; //系统函数,决定是否使得高优先级任务就绪
POPALL; //出栈
RETI

```

OSIntExit通过调用OSIntCtxSw,在ISR中执行任务切换功能。因为OSIntCtxSw是在ISR中被调用的,所以假定寄存器都被正确地保存了被中断的任务的堆栈之中。因此,OSIntCtxSw和上文中OSCtxSw的代码基本相同,差别仅仅在于开始之前少了一个POPALL操作。

4 结论

本文介绍了 $\mu c/OS-II$ 移植到51单片机上的全过程,对从事嵌入式系统移植工作的开发人员来说有一定的参考价值。作为一个优秀的实时操作系统 $\mu c/OS-II$ 已经被移植到各种体系结构的微处理器上,51系列处理器是8位处理器中最流行的,将 $\mu c/OS-II$ 移植到51平台上,能够使我们更深入地了解实时操作系统的构造,加快在51平台上的应用开发,并为更高层次上的扩展和改进打下基础。

参考文献:

- [1] 陈是知. $\mu c/OS-II$ 内核分析、移植与驱动程序开发[M].北京:人民邮电出版社,2007.
- [2] 罗蕾.嵌入式实时操作系统及应用开发[M].2版.北京:北京航空航天大学出版社,2007.
- [3] 于永.51单片机C语言常用模块与综合系统设计实例精讲[M].北京:电子工业出版社,2007.
- [4] Labrosse J.嵌入式实时操作系统 $\mu c/OS-II$ [M].邵贝贝,译.2版.北京:北京航空航天大学出版社,2003.
- [5] 胡汉才.单片机原理及系统设计[M].北京:清华大学出版社,2002.
- [6] 张云生.实时控制系统软件设计原理及应用[M].北京:国防工业出版社,1998.