μC/OS-Ⅱ在AT89S52上的移植

王莹莹,李 斌
(重庆大学 自动化学院,重庆 400044)

摘 要:通讯和网络技术的迅速发展推动嵌入式系统的发展进程,嵌入式系统凭借其高度的可靠性、低成本、小体积、低功耗的特点占据了广阔的市场。在嵌入式系统中使用实时操作系统,可以提高系统的稳定性、可靠性和实时性,因此为单片机移植操作系统是单片机开发的必要工作。简要介绍了实时操作系统μC/OS-Ⅱ的结构,详细分析其在AT89S52处理器上的移植过程。指出了移植中出现的问题,并对移植后的系统进行了测试,从而构成了一个实时多任务的开发平台。

关键词:实时操作系统;μC/OS-Ⅱ;AT89S52;移植
中图分类号:TP316 **文献标识码**:A **文章编号**:1673-629X(2007)11-0244-03

Porting of RTOS μC/OS-Ⅱ to AT89S52 MCU

WANG Ying-ying, LI Bin
(Automatization College, Chongqing University, Chongqing 400044, China)

Abstract: Development of communication and network's technology promoted process of embedded system, it occupied a larger market rely on its high dependability, low cost, little body and low expenditure. An embedded system using RTOS has higher reliability, higher stability and higher speed, so porting RTOS to MCU is a necessary work of development MCU. Introduces the architecture of RTOS μC/OS-Ⅱ briefly as well as gives a detailed analysis on porting μC/OS-Ⅱ to AT89S52 MCU. It points out the problem in porting, and tests the ported operating system, so the real-time kernel development platform is founded.

Key words: RTOS; μC/OS-Ⅱ; AT89S52; porting

0 引 言

嵌入式系统是以应用为中心,以计算机技术为基础,软硬件可剪裁,适应应用系统对功能、可靠性、成本及功耗严格要求的专用计算机系统[1]。随着科技的发展,嵌入式系统已经在生产生活的各个方面得到广泛的应用。研究嵌入式系统,一个必不可少的基础性工作就是实现嵌入式实时操作系统在目标处理器上的移植。目前高性能的51单片机应用仍非常广泛,为其移植实时操作系统是有一定的现实意义的。文中研究了将源代码公开的实时操作系统μC/OS-Ⅱ移植到AT89S52上的方法及实现。

1 μC/OS-Ⅱ的体系结构

μC/OS-Ⅱ是一个完整的、可移植、固化、剪裁的占先式实时多任务内核。它采用ANSI的C语言编写,包含一小部分汇编语言代码,使之可供不同架构的微处理器使用[2]。μC/OS-Ⅱ是完全可剥夺型的实时内核,总是运行就绪条件下优先级最高的任务。μC/OS-Ⅱ可以管理64个任务,提供了任务管理的各种函数和4种同步对象:信号量、邮箱、消息队列和事件标志组。μC/OS-Ⅱ把连续的大块内存按分区来管理,消除了多次分配与释放内存所引起的内存碎片。其体系结构如图1所示。

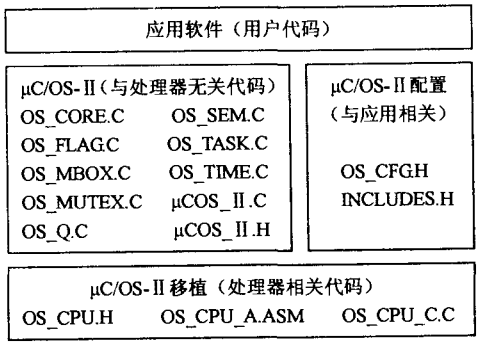

图1 μC/OS-Ⅱ体系结构

1)μC/OS-Ⅱ核心代码:包括10个C程序和1个头文件,主要实现了系统调度、任务管理、内存管理、信号量、消息邮箱和消息队列等系统功能。此部分代码与处理器无关。

收稿日期:2007-01-06
作者简介:王莹莹(1982-),女,辽宁人,硕士研究生,研究方向为单片机与嵌入式系统;李 斌,副教授,硕士生导师,主要从事导航制导方面的研究。

2)μC/OS－Ⅱ配置代码:包括 2 个头文件,用于剪裁和配置 μC/OS－Ⅱ。此部分代码与用户实际应用相关。

3)μC/OS－Ⅱ移植代码:包括 1 个汇编文件、1 个 C 程序文件和 1 个头文件,这是移植 μC/OS－Ⅱ所需要的代码。此部分代码与处理器相关。

2　μC/OS－Ⅱ的移植过程

2.1　移植条件分析

AT89S52[3]支持定时中断,并支持外扩 64kB 的最大数据存储器空间,满足了移植条件:

①处理器支持中断,并且能产生定时中断;

②处理器支持能够容纳一定量数据的硬件堆栈;

③处理器有将堆栈和其它 CPU 寄存器的内容读出,并保存到堆栈或内存中去的指令。

Keil μVision2[4]集成开发环境通过构造仿真函数来解决函数的可重入问题;可使用 C 语言对 EA 赋值实现对中断的开关,满足了移植条件:

a.处理器的 C 编译器能产生可重入型代码;

b.用 C 语言就可以开/关中断。

2.2　移植环境设置

使用 Keil μVision2 集成开发环境,在配置工程时,将存储模式设置成:Large:variables in XDATA;将程序储存器大小设置成:Large:64kB program,并选中 Use On－chip ROM (0x0－0x01FFF);根据实际的扩展电路设置外部数据存储器的大小和起始地址。

2.3　μC/OS－Ⅱ移植代码的实现

2.3.1　OS_CPU .H 的移植

OS_CPU.H 包括了用 # define 语句定义的、与处理器相关的常数、宏以及数据类型。

AT89S52 处理器系统硬件堆栈的单元宽度为一个字节,所以数据类型中的 OS_STK 需要做修改,代码如下:

```
typedef unsigned char OS_STK
```

临界区保护使用方式 1,代码如下:

```
# define OS_CRITICAL_METHOD 1
# if OS_CRITICAL_METHOD= =1
# define OS_ENTER_CRITICAL() EA=0
# define OS_EXIT_CRITICAL() EA=1
# endif
```

AT89S52 的硬件堆栈是从下向上增长的,故堆栈增长方向定义为向上递增,代码如下:

```
# define OS_STK_GROWTH 0
```

AT89S52 不提供软中断指令,任务级任务切换函数通过程序调用实现,代码如下:

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

2.3.2　OS_CPU_C .C 的移植

移植要求用户根据需要修改、实现该文件中的 6 个 C 语言函数(OSTaskStkInit();OSTaskCreateHook();OSTaskDelHook();OSTaskSwHook();OSTaskIdleHook();OSTaskStatHook();OSTimeTickHook())。这些函数中,5 个系统 Hook 函数可以为空函数,也可以根据用户自己的需要编写相应的操作代码,但必须声明。任务栈结构初始化函数 OSTaskStkInit()必须根据移植时统一定义的任务栈结构进行初始化[5]。

OSTaskStkInit()函数由 OSTaskCreate()或 OSTaskCreateExt()调用,用于系统创建用户任务时,建立并初始化任务堆栈。函数将所需的寄存器入栈,返回新堆栈的栈顶指针,并将它保存在该任务的任务控制块 OS_TCB 中,最终使初始化后的堆栈跟刚发生过一次中断一样。默认情况下 μVision2 的 C 编译器对指针型参数使用工作寄存器 R1、R2 和 R3 进行传递。R1 中存放指针参数的低地址;R2 中存放指针参数的高地址;R3 中存放指针指向的存储空间类型。堆栈初始化时除了这三个寄存器的值有意义外,其他的只需要调整堆栈指针保留空间即可。

任务堆栈的第一个字节存放的是任务堆栈的初始有效长度,它表示在之后的任务切换中,从任务堆栈中拷贝到系统硬件堆栈中的数据长度。在任务堆栈中还必须保存该任务仿真堆栈指针?C_XBP 的值。

2.3.3　OS_CPU_A.ASM 的移植

移植要求用户在该文件中实现 4 个汇编语言函数:运行优先级最高的就绪任务函数 OSStartHighRdy();任务级的任务切换函数 OSCtxSw();中断级的任务切换函数 OSIntCtxSw()和时钟节拍中断服务函数 OSTickISR()。

1)OSTaskStkInit()的实现。

OSStartHighRdy()函数是在 OSStart()多任务函数启动之后,负责从最高优先级任务的 TCB 控制块中获得该任务的堆栈指针,并将该任务堆栈中的寄存器值恢复到处理器的寄存器中,然后调用中断返回指令。

获取了将要恢复运行的任务的堆栈指针和堆栈长度后,通过字节拷贝的方式将位于 xdata 中的任务堆栈内容复制到位于 idata 中的硬件堆栈中,同时需要恢复硬件堆栈指针 SP 和仿真堆栈指针? C_XBP;最后从硬件堆栈中恢复所有寄存器的值到处理器寄存器。

2)OSCtxSw()和 OSIntCtxSw()的实现。

OSCtxSw()和 OSIntCtxSw()实现的功能都是任务切换,即保存当前任务的所有寄存器到任务堆栈,然后将新任务的任务堆栈中的寄存器恢复到处理器寄存

器中。不同的是 OSIntCtxSw()是在 ISR 中被调用,所有寄存器都被正确地保存到了被中断任务的堆栈之中,所以在 OSIntCtxSw()中无需再保存 CPU 寄存器。但是正因为 OSIntCtxSw()是在 ISR 中被调用,且是通过 OSIntExit()调用的,两级调用都存在对返回地址的入栈操作,这就使此时的硬件堆栈的内容比起运行 OSCtxSw()实现任务切换有所不同,如图 2 所示。而 μC/OS-Ⅱ要求以任何方式挂起的任务都必须具有相同的堆栈结构,所以必须对 SP 进行处理,使系统硬件堆栈结构跟所有被挂起的任务堆栈结构一样。由图 2 可以看出,只需将 SP 减 4 即可。

3)OSTickISR()的实现。

利用 AT89S52 的定时器 0 产生时钟节拍的中断源,它通过调用 OSTimeTick()函数,为操作系统提供周期性的时钟源,以实现任务时间的延迟和超时功能。

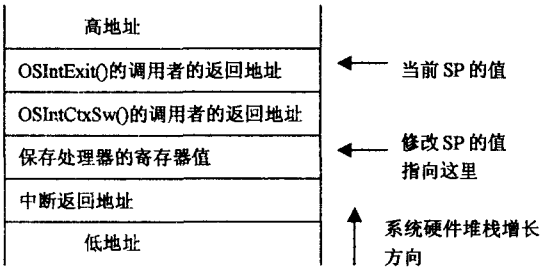

图 2 在 ISR 执行过程中的系统硬件堆栈内容

3 移植中解决的问题

3.1 函数的可重入

由于 AT89S52 的堆栈太小,为了实现函数的可重入,形式参数和局部变量的值都保存在 μVision2 的仿真堆栈中(生长方向由上向下,与硬件堆栈相反)[6]。仿真堆栈位于系统的外部数据存储区(xdata),栈指针为?C_XBP。该堆栈的设置在文件 STRATUP .A51 中。KEIL C51 在不修改 STRATUP.A51 文件的情况下,缺省使用 64k 的外部 RAM,它把 0000H~FFFFH 全部仿真为可读写的 RAM,而用户的硬件系统可能没有用到那么大的 RAM 空间,或者将一些地址空间映射给了别的设备。所以根据系统需要,修改 STRATUP.A51 文件,并将其加入项目编译,代码如下:

```
XBPSTACK EQU 1
XBPSTACKTOP EQU 0x7FFFF+1
```

3.2 核心代码的修改

μC/OS-Ⅱ的核心文件本来在移植过程中是不需要修改的,但其中一些文件使用的变量名如:pdata、data 是 μVision2 扩展的 C 语言关键字,这样会导致编译错误,需要修改。可将 pdata 改成 ppdata、将 data 改成 ddata。μVision2 的 C 编译器要求使用关键字 reentrant 来产生可重入代码,所以必须在每个 C 函数的定义和声明中添加关键字 reentrant。

4 移植结果测试

创建 2 个任务 A、B 优先级分别为 2、3。测试程序代码如下:

```
OSTaskCreate(Task A,(void *)0,&TaskStartStkA,2);
OSTaskCreate(Task B,(void *)0,&TaskStartStkB,3);
  void TaskA(void *ddata) reentrant {
  ddata=ddata;
  for(;;)
  { printf("AA\n");
    OSTimeDly(60*OS_TiCKS_PER_SEC);
    }
  }
  void TaskB(void *ddata) reentrant
  {
  ddata=ddata;
  for(;;)
  {printf("BB\n");
    OSTimeDly(30*OS_TiCKS_PER_SEC);
  }
}
```

多任务调度开始后,通过串口接受的数据为:AA BB AA BB…。高优先级的任务 Task A 能首先被调度运行,说明 OSTaskStkInit()和 OSTaskStkInit()是正确的。任务 TaskA()和 TaskB()由时钟节拍驱动而被周期地调用,说明 OSCtxSw(),OSIntCtxSw()和 OSTickISR()正确,因此表明移植是成功的。

参考文献:

[1] 田 泽.嵌入式系统开发与应用[M].北京:北京航空航天大学出版社,2005:544-572.

[2] Labrosse J 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,2003:283-307.

[3] 张毅刚,彭喜元.新编 MCS-51 单片机应用设计[M].哈尔滨:哈尔滨工业大学出版社,2003:29-82.

[4] 徐爱均,彭秀华.Keil Cx51 V7.0 单片机高级语言编程与 μVision2 应用实践[M].北京:电子工业出版社,2004:435-605.

[5] 孙 静.嵌入式实时操作系统 μC/OS-Ⅱ的移植[J].电工技术,2005(2):2-3.

[6] 黄 力.实时操作系统的移植与扩展性的研究[J].现代电子技术,2005(14):2-3.