μC/OS-II在Cortex-M3内核上的移植及优化①

孙顺远　秦会斌　崔佳冬　丁红斌

(杭州电子科技大学 新型电子器件与应用研究所 浙江 杭州 310018)

摘　要： 描述了源代码公开的实时操作系统μC/OS-II 在以 Cortex-M3 内核的 ARM 处理器 STM32-F103ZET6 上的移植过程。介绍了 ARM 的 Cortex-M3 内核的基本概念,及移植过程,并对移植过程中的任务堆栈做出优化，最后通过建立简单任务,来验证移植的正确性以及系统在 STM32F103ZET6 上运行的可行性和稳定性。

关键词： μC/OS-II；Cortex-M3；ARM；移植；堆栈；优化

Porting μC/OS-II to Cortex-M3 and Optimization

SUN Shun-Yuan, QIN Hui-Bin, CUI Jia-Dong, DING Hong-Bin

(School of Electronics Information,Hangzhou Dianzi University, Hangzhou 310018, China)

Abstract: This paper introduces the process of porting an open source real time operating system μC/OS-II to the ARM processor STM32F103ZET6. Several basic concepts of the Cortex-M3 and how to optimize the stack of task are briefly introduced. By creating some simple tasks, the grafting process is proved to be successful and the system running on STM32F103ZET6 is proved to be feasible and stable.

Keywords: μC/OS-II; Cortex-M3; ARM; porting; stack; optimization

随着嵌入式技术的广泛应用，电子产品设计人员根据产品的可靠性和实时性的要求，在软件开发过程中越来越倾向于基于嵌入式操作系统的开发模式。μc/os-II 是一种基于抢占式实时多任务调度的简单高效、源代码公开的实时嵌入式操作系统，具有良好的可扩展性和可移植性[1]。意法半导体公司(ST)2008 年正式在市场上推出的以 Cortex-M3 为内核的 STM32F103ZET6，其库文件与μc/os-II 的联合使用，对于产品减少开发周期和降低成本有重要意义。本文主要介绍一下μC/OS-II 在处理器 STM32F103ZET6 上的移植及其堆栈优化。

1　软硬件开发环境及处理器介绍

1.1 软硬件开发环境介绍

本文移植过程中使用的开发平台是万利公司推出的评估板 EK-STM3210E，微处理器是意法半导体公司(ST)的 STM32F103ZET6，软件环境是由 ARM 公司 2009 最新推出的 RealView MDK3.40 开发套件，这款开发套件集成了 C 语言编译器、宏编译、链接/定位以及 HEX 文件产生器，界面友好，在调试程序和软件仿真方面表现良好。

1.2 关于处理器的简单介绍

这款 ARM 微控制器 STM32F103ZET6 以 Cortex-M3 为内核，支持硬件除法，单周期 32 位乘法，集 32 位 RISC 处理器、低功耗、高性能模拟技术、DMA 以及灵活的静态存储器控制器(FSMC)等丰富的片上外设于一体，最高时钟频率可达 72MHz,指令速度可接近 80MIPS[2]，常见的有以下应用：

(1) 工业领域应用：变频器、扫描仪和工控网络等。

(2) 建筑和安防应用：警报系统、可视电话和 HVAC。

① 基金项目:国家自然科学基金(60601022)

收稿时间:2009-07-20;收到修改稿时间:2009-09-04

(3) 低功耗应用：血糖测试仪、电表和电池供电应用。

(4) 家电应用：电机控制和应用控制。

(5) 消费类产品：PC 外设、游戏机、数码相机和 GPS。

2 μC/OS-II的移植过程

本文中移植的μC/OS-II 的版本为 2.86，所有的移植过程都是该版本的源代码。

2.1 μC/OS-II 内核移植介绍

要使μC/OS-II 正常运行，处理器必须满足一下要求[3]：

(1) 处理器的 C 编译器能产生可重入型代码。

(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(频率通常在 10 至 100HZ 之间)

(4) 处理器能支持容纳数据存储硬件堆栈

(5) 处理器包含将堆栈指针和其他 CPU 寄存器的内容读出并存储到堆栈或内存的指令。

根据上述要求如图 1 所示,进行μC/OS-II 的移植,主要对 OS_CPU.H,OS_CPU_C.C，OS_CPU_A.ASM 三个文件进行修改[4]。

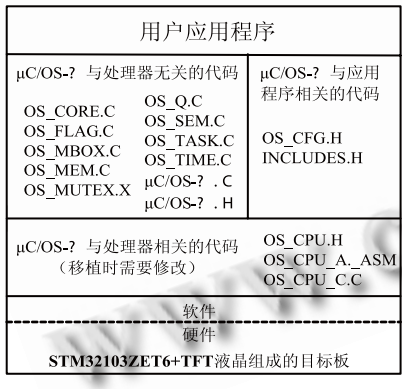

图 1 μC/OS-II 体系结构

2.1.1 OS_CPU.H 文件

头文件 OS_CPU.H 包含μC/OS-II 所需要的常量、宏和自定义类型，首先是根据使用的 MDK 编译器的相关说明文档得到编译器所支持的基本数据类型，然后据此对 OS_CPU.H 相关数据进行重新定义，这里强调指出重定义 OS-STK 处理器堆栈为 32 位，OS-CPU-SR 处理器状态寄存器字长尾 32 位。代码如下：

```
typedef    unsigned long    OS-STK；
typedef    unsigned long    OS-CPU-SR；
```

其次是跟处理器相关部分代码包括临界区访问处理、处理器堆栈增长方向及任务切换宏定义。这里需要说明的以 Cortex-M3 内核微处理器的堆栈的增长方式是从内存高地址向低地址方向递减的，即“向下生长的满栈”[5]代码如下：

```
#define    OS-STK-GROWTH    1；
```

2.1.2 OS_CPU_C.C 文件

OS_CPU_C.C 文件中有 10 个 C 语言函数，这些函数中唯一必须需要修改的就是 OSTaskStkInit，此函数的作用就是把任务堆栈初始化成好像刚发生过中断一样。Cortex-M3 内核微处理器响应中断时硬件自动保存现场：依次把 xPSR,PC,LR,R12 以及 R3–R0 由硬件自动压入适当的堆栈中，如果当响应异常时，当前的代码正在使用 PSP，则压入 PSP，即使用线程堆栈；否则压入 MSP，使用主堆栈。一旦进入了服务例程，就将一直使用主堆栈。所以这里就不用考虑，系统具体用的那个堆栈了，并且初始化时需要注意的是 xPSR,PC,LR 的初值问题，部分代码如下：

```
stk=ptos;
*(stk)=(INT32U)0x01000000L;
*(--stk)=(INT32U)task;
*(--stk)=(INT32U)0xFFFFFFFEL;
```

2.1.3 汇编 OS_CPU_A.ASM 文件

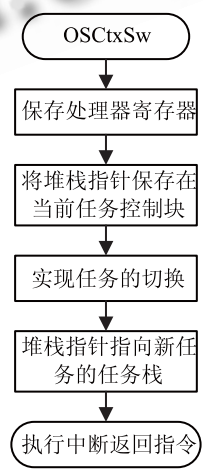

图 2 OSStartHighRdy 框图

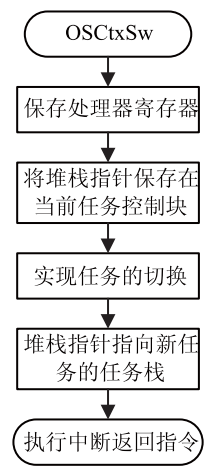

图 3 OSCtxSw 框图

OS_CPU_A.ASM 文件中包括需要修改的 3 个汇

编函数分别为 OSStartHighRdy、OSCtxSw、OSIntCtxSw.这里面还应该包括一个 OSTickISR 函数，不过这个函数可以用 C 语言实现。在三个汇编函数在修改过程中需要特别注意的就是在任务级的任务切换的时候，注意要保存的的非自动入栈的寄存器，它们分别是 R4-R11,并且要注意的是在弹出栈的时候要按照先进后出的概念。其中 OSStartHighRdy、OSCtxSw 程序的框图分别如图 2、图 3 所示。

2.1.4 OSTickISR 函数的 C 语言编写

μC/OS-Ⅱ要求用户提供一个周期性时钟源，来实现时间的延时和超时功能，时钟节拍该每秒钟发生 10~100 次。时钟节拍可以用定时器中断实现，也可以放在任务中实现。STM32F103ZET6 中有一个专用的系统时钟节拍定时器 SysTick,利用这个定时器给系统产生一个 10ms 一次的时钟节拍中断，定时器 SysTick 的初始化和中断程序框图如图 4 所示及部分代码如下：

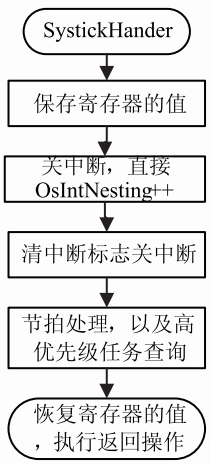

图 4　OSTickISR 框图

(1) 初始化程序部分代码：

```
#define OS_TICKS_PER_SEC                100
cnts=rcc_clocks.HCLK_Frequency/OS_TICKS_PER_SEC;
SysTick_SetReload(cnts);
```

(2) SysTick 中断处理程序部分代码：void SysTickHandler(void)

```
{
 OS_CPU_SR  cpu_sr;
 OS_ENTER_CRITICAL();
 OSIntNesting++;
 OS_EXIT_CRITICAL();
 OSTimeTick();
 OSIntExit();
}
```

在这个 OSTickISR 函数的移植过程中，一定要注意时钟节拍中断服务的使能位置，常见的错误就是在调用 OSInit()和 OSStart()之间允许时钟节拍中断，因为如果在中间允许节拍中断，μC/OS-Ⅱ开始执行第一个任务前，时钟节拍中断服务程序就会执行，而此时的μC/OS-Ⅱ处于未知状态，应用程序自然就会崩溃。本移植过程中，将初始化以及开时钟节拍中断服务程序放在第一个任务之中。

3　系统的优化及性能测试分析

3.1 系统的堆栈优化

在μC/OS-II 中，每个任务都定义了一个独立的堆栈空间，这个堆栈空间用来存放任务的相关信息，其中关于中断嵌套时需要保存的上下文的堆栈空间使用量随着中断嵌套的深度而不断增加，一般办法就是定义一个充分大的栈空间，使之不会溢出。这样就会导致栈空间的浪费。本移植过程就是为中断嵌套单独定义一个中断嵌套栈。在发生第 1 次中断时，中断服务程序将栈空间切换到中断嵌套栈，这样，以后发生的嵌套中断就一直使用这个栈空间。在中断返回到第 1 次中断时，即 OSIntNesting 为 1 时，中断服务程序再从中断嵌套栈切换回任务栈。很明显这样每个任务的堆栈通过察看反汇编代码分配合适的大小，而不是充分大，减少不必要内存的浪费。

3.2 系统的性能测试及分析

测试移植的系统是否正常工作。首先不加任何应用代码来测试移植好的μC/OS-Ⅱ。同时需要注意配置 OS-CFG.H，即将需要的功能模块包含到操作系统中去，测试步骤如下：

确保 MDK 编译器和连接器的正常工作;

① 验证 OSTaskStkInit( )和 OSStartHighRdy( )函数的正确运行;

② 验证 OSCtxSw( )函数;

③ 验证 OSIntCtxSw( )和 OSTickIsr( )函数

验证完内核正常运行后，在系统中创建了三个任务，一个任务是液晶 LCD 的显示刷新任务，另一个任

务是按键 KEY1 和 KEY2 以及 Joystick 按键的任务，另一个任务是 LED 灯根据 AD 采样电压值来决定闪烁时间的任务，最后达到的效果通过串口表现出来,同时也可以观察到开发板上面的 LED 也在不停的闪烁，由此可以看出移植代码成功,内核正常运转，如图 5 所示。

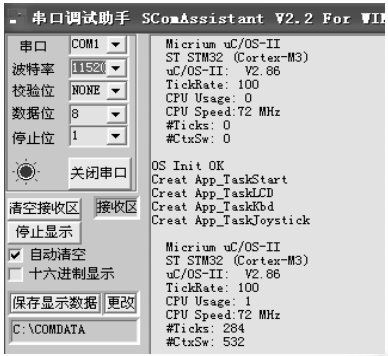

图 5 移植系统运行效果图

4 结语

μC/OS-II 作为一个优秀的实时操作系统，已经被移植到各种体系结构的微处理器上。本设计实现了在 STM32F103ZET6 上的成功移植，并且系统运行稳定。可以在本系统移植工作基础上进行二次开发，充分利用 Cortex-M3 为内核的 STM32 系列单片机的性价比高、功耗低、开发周期短等特点，在工业控制领域、建筑和安防应用、低功耗医疗电子、家电应用以及消费品产品应用上发挥更显著的作用。

参考文献

1 陈是知.μC/OS-II 内核分析、移植与驱动程序开发.北京:人民邮电出版社, 2007. 127－164.

2 王永虹.ARM Cortex-M3 微控制器原理与实践.北京:北京航空航天大学出版社, 2008. 1－45.

3 任哲.嵌入式实时操作系统 μC/OS-II 原理及应用.北京:北京航空航天大学出版社, 2005. 13－23.

4 Jean J. Labrosse.嵌入式实时操作系统 μC/OS-II.北京:北京航空航天大学出版社, 2003. 26－36.

5 Yiu J.The Definitive Guide to the ARM Cortex- M3. 2007. 175－179.