μ C/OS-II 在 stm32 上的移植

Porting μ C/OS-II to STM32

西安工程大学电子信息学院 肖 磊 张 娜
Xiao Lei Zhang Na

摘 要：该论文描述了实时操作系统μ C/OS-II 在以Cortex-M3内核的STM32F103VBT6上的移植过程。介绍了Cortex—M3内核的基本概念及移植过程。简要的说明了判断μ C/OS-II内核是否正常运行的方法。最后通过建立简单任务，来验证移植的正确性以及系统在STM32F103VBT6上运行的可行性和稳定性。

关键词：μ C/OS-II Cortex-M3 移植

Abstract: This paper introduces the process of porting an real time operating system μ C/OS-II to the processor STM32F103VBT6. Several basic concepts of the Cortex-M3. The method whether the kernel runs in normal is introduced in brief. 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.

Key words: μ C/OS-II Conex-M3 porting

【中图分类号】TP311 【文献标识码】B 文章编号 1606-5123(2011)06-0063-04

1 引言

随着计算机技术的进步，在工业生产及人们的日常生活中有许多设备和装置中都内置了计算机系统，使得这些设备或装置具有很高的自动化性能和某种程度的智能性，从而极大地满足了人们生产和生活的需要。也带来了嵌入式操作系统迅速发展。μ C/OS-II内核作为一种代码公开的嵌入式实时操作系统的内核非常有特色，在规模不大的代码内实现了抢占式任务调度和多任务间通信等功能。要使μ C/OS-II正常运行，处理器必须满足以下要求：

(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。

2 处理器的简介

微控制器STM32F103VBT6以Cortex～M3为内核，Cortex～M3是首款基于ARMv7-M体系结构的32位RISC处理器，具有低功耗、少门数、短中断延迟、低调试成本等优点。它是专门为在微控制系统、汽车车身系统、工业控制系统和无线网络等对功耗和成本敏感的嵌入式应用领域实现高系统性能而设计的，它大大简化了编程的复杂性，集高性能、低功耗、低成本于一体。

3 软件硬开发环境

本移植过程使用的软件环境是RealView MDK开发套件，此产品是ARM公司最新推出的针对各种嵌入式处理器的软件开发工具，该开发套件功能强大，包括了μ Vision4集成开

© 1994-2011 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

发环境和RealView编译器。使用的硬件平台是深圳英蓓特公司推出的全功能评估板STM103V100，其上所采用的处理器是ST意法半导体公司生产的32位哈佛结构ARM处理器STM32F103VBT6，该处理器内置ARM公司最新的Cortex—M3核，并且具有非常丰富的片上资源。界面友好，在调试程序和软件仿真方面表现良好。

4 移植过程

本文中移植的μ C/OS-II的版本为2.90，所有的移植过程都是该版本的源代码。图1描述了μ C/OS-II在stm32上移植的体系结构。

在移植过程中，用户需要注意的就是与处理器相关的代码。这部分包括头文件OS_CPU.H、汇编文件OS_CPU_A.ASM、C代码文件OS_CPU_C.C。

4.1 OS_CPU.H：基本配置和定义

(1)数据类型

为了便于可移植性，程序中没有直接使用C语言中一些数据类型的定义，因为它们与处理器类型有关，隐含不移植性。可程序中自己定义了一套数据类型。

(2)允许和禁止中断宏

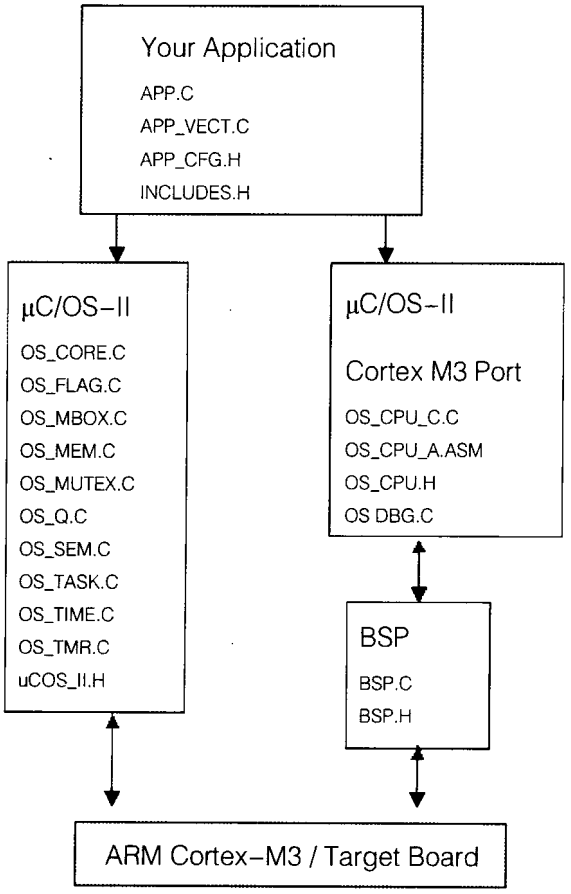

图1 μ C/OS-II 在 stm32 上移植的体系结构

μ C/OS-II需要先禁止中断，再访问代码临界区，访问完毕后，重新允许中断。以免临界段代码被多任务或中断服务程序的破坏。

在STM32处理器上实现的代码如下：

```
#define OS_CRITICAL_METHOD  3
#define OS_ENTER_CRITICAL() {cpu_sr = OS_CPU_SR_Save();}
#define OS_EXIT_CRITICAL() {OS_CPU_SR_Restore(cpu_sr);}
```

注意：如果应用程序使用这些宏，必须定义cpu_sr，并赋初值为0。

```
OS_CPU_SR  cpu_sr = 0;
```

(3)栈的生长方向

如果栈的生长方向从上往下，置OS_STK_GROWTH为1，否则置OS_STK_GROWTH为0。

(4)OS_TASK_SW()宏

从低优先级任务切换到高优先级任务时被调用。如果处理器支持软中断，使用软中断将中断向量指向OSCtxSW()函数。或者直接调用OSCtxSW()函数。

4.2 OS_CPU_A.ASM：移植与处理器相关的函数

(1)OSStartHighRdy()

OSStartHighRdy()函数在OSStart()中被调用，负责从最高优先级任务的TCB控制块中获得任务的栈指针SP，将CPU现场恢复。OSStartHighRdy()涉及将处理器寄存器保存到栈的操作。

(2)OSCtxSw()

当任务放弃CPU的使用权，OS_TASK_SW()宏调用OSCtxSw()函数。OSCtxSw()函数执行任务优先级切换，在STM32中所有的上下文切换都是由PendSV handler决定的。OSCtxSw()触发PendSV handler，并且返回到调用函数。PendSV handler不会马上运行，因为OS_TASK_SW()是在中断禁止的时候调用。PendSV handler只有当中断允许时才能运行。OSCtxSw()函数先将当前任务的CPU现场保存到该任务的栈中，然后获得最高优先级任务的栈指针，并从该栈中恢复任务的CPU现场，使之继续执行，完成一次任务切换。OS_TASK_SW()由OS_Sched()函数调用。

```
OSCtxSw
        LDR     R4, =NVIC_INT_CTRL      ; trigger the PendSV exception
        LDR     R5, =NVIC_PENDSVSET
        STR     R5, [R4]
        BX      LR
```

(3)OSIntCtxSw()

© 1994-2011 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

当中断服务程序结束时，OSIntExit()调用OSIntCtxSw()，以此来判断是否有比中断任务更重要的任务要执行。这样做的目的是尽快地让高优先级的任务得到响应，保证系统的实时性能。OSIntCtxSw()与OSCtxSw()都是用于任务切换的函数，其区别是，OSIntCtxSw()无需再保存CPU寄存器，因为在调用OSIntCtxSw()前已发生了中断，将默认的CPU寄存器保存到了被中断的任务栈中。

```
OSCtxSw
        LDR     R4, =NVIC_INT_CTRL      ; trigger the PendSV exception
        LDR     R5, =NVIC_PENDSVSET
        STR     R5, [R4]
        BX      LR
```

(4)OSPendSV()

OSPendSV()在μ C/OS-II中处理所有上下文切换，见图2。STM32在发生异常时会自动保存半数寄存器，因此OSPendSV handler只需保存R4-R11和修改栈指针即可。使用PendSV异常意味着上下文保存和恢复是使用一个相同的方法，无论它是引发的某项任务或由于发生中断或异常。注意，必须将指针指向异常向量表位置14(OSPendSV)。必须把OSPendSV编程为最低优先级的异常，因为OSPendSV异常会自动延迟上下文切换的请求，直到其他的ISR都完成了处理后才放行。

(5)OSTickISR()

μ C/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链，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。

```
void OS_TickISR_Handler (void)
{
 OS_CPU_SR cpu_sr;
 OS_ENTER_CRITICAL(); /* Tell uC/OS-II that we are starting an ISR */
 OSIntNesting++;
 OS_EXIT_CRITICAL();
 OSTimeTick();/*Call uC/OS-II's OSTimeTick()  */
 OSIntExit();/* Tell uC/OS-II that we are leaving the ISR  */
}
```

在0STickISR函数的移植过程中，应确定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，μ C/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μ C/OS-II处于未知状态，应用程序自然就会崩溃。

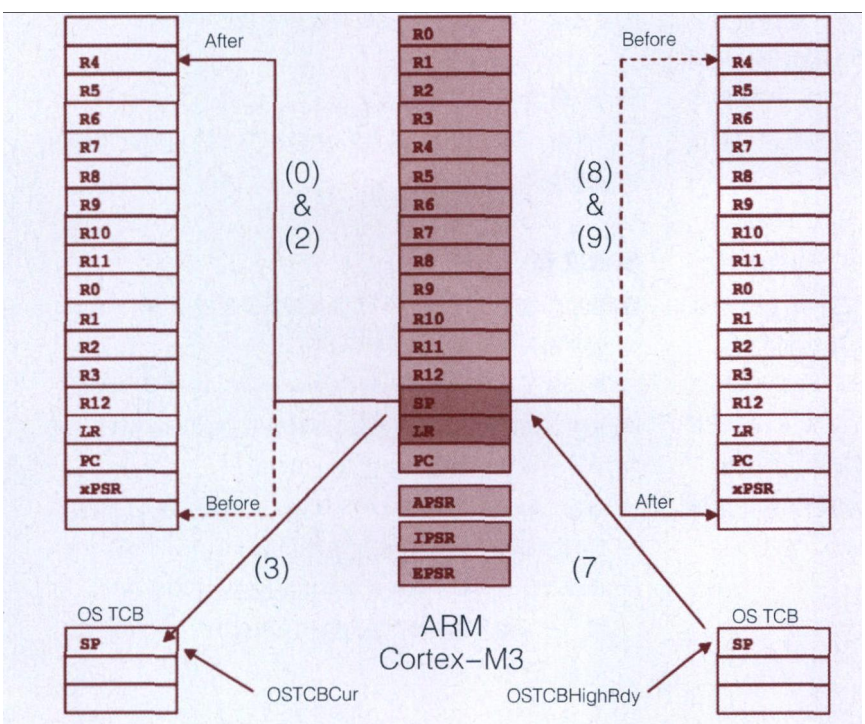

图2 Cortex-M3上下文切换

4.3 OS_CPU_C.C

ucos ii的移植要求编写10个简单的C函数：

```
OSInitHookBegin()
OSInitHookEnd()
OSTaskCreateHook()
OSTaskDelHook()
OSTaskIdleHook()
OSTaskStatHook()
OSTaskStkInit()
OSTaskSwHook()
OSTCBInitHook()
OSTimeTickHook()
```

这些函数中，唯一必须移植的是任务栈初始化函数OSTaskStkInit()。这个函数在任务创建时被调用，负责初始化任务的栈结构并返回新栈的指针stk。栈初始化工作结束后，返回新的栈顶指针。其他9个函数必须声明，但并不一定要包含任何代码。

```
OS_STK *OSTaskStkInit (void (*task)(void *pd), void *p_arg, OS_STK *ptos, INT16U opt)
{
   OS_STK *stk;
   (void)opt;      /* 'opt' is not used, prevent warning */
   stk= ptos;      /* Load stack pointer          */
            /* Registers stacked as if
```


© 1994-2011 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

```
saved on exception   */
        *(stk)=(INT32U)0x01000000L;    /*xPSR            */
        *(--stk)=(INT32U)task;         /*Entry   Point   */
        *(--stk)=(INT32U)0xFFFFFFFEL;  /*R14(LR)         */
        *(--stk)=(INT32U)0x12121212L;  /*R12             */
        *(--stk)=(INT32U)0x03030303L;  /*R3              */
        *(--stk)=(INT32U)0x02020202L;  /*R2              */
        *(--stk)=(INT32U)0x01010101L;  /*R1              */
        *(--stk)=(INT32U)p_arg;        /*R0:argument     */
          /* Remaining registers saved on process stack   */
        *(--stk)=(INT32U)0x11111111L;  /*R11             */
        *(--stk)=(INT32U)0x10101010L;  /*R10             */
        *(--stk)=(INT32U)0x09090909L;  /*R9              */
        *(--stk)=(INT32U)0x08080808L;  /*R8              */
        *(--stk)=(INT32U)0x07070707L;  /*R7              */
        *(--stk) =(INT32U)0x06060606L; /* R6             */
        *(--stk)=(INT32U)0x05050505L;  /*R5              */
        *(--stk)=(INT32U)0x04040404L;  /*R4              */
      return (stk);
    }
```

5 测试移植代码

做完μ C/OS-II的移植后，就应该验证移植的μ C/OS-II是否能正常工作。首先不应该加如何代码来验证移植好的μ C/OS-II，也就是验证内核自身的运转状况。这样做出于以下两个原因：(1)避免将事情复杂化。(2)如果有部分代码不能正常工作，就可以明白是移植本身的问题，而不是应用代码产生的问题。如果将两个基本的任务和节拍中断运行起来，那么以后添加应用任务就很简单了。

需要验证的移植代码有OSStartHighRdy()，OSTaskStkInit()，OSCtxSw()，OSTickISR()，OSIntCtxSw()。确保RealView MDK编译器和连接器的正常工作。接下来我们简单的来说明验证OSStartHighRdy()，OSTaskStkInit()。

为了进行验证，需要修改OS_CPU_C.C，OS_CFG..H，以及main()。在OS_CFG..H文件，设置OS_TASK_STAT_EN为0，以禁止统计任务。因为没有添加应用程序，所有唯一的任务是μ C/OS-II的空闲任务：OS_TaskIdle()。接下来单步执行，直到μ C/OS-II切换到OS_TaskIdle()。

部分代码如下：

```
void main(void)
{
OSInit();
/*关LED*/
OSStart();
}
void OSTaskIdleHook(void)
{
if(LED is ON)
关LED;
}else{
开LED;
}
}
```

将代码下载到目标系统中进行验证。如果LED闪烁不正常，则说明OSStartHighRdy()，OSTaskStkInit()不能正常工作。否则说明OSStartHighRdy()，OSTaskStkInit()能正常工作。通过类似的方法也可以验证OSCtxSw()，OSTickISR()，OSIntCtxSw()。

如果以上函数都能验证成功，我们认为内核自身的运转正常。接下来，我们就可以根据自己的所需来编写应用程序了。

6 结束语

本设计实现了在STM32F103VBT6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系在微控制系统、汽车车身系统、工业控制系统和无线网络等对功耗和成本敏感的嵌入式应用领域实现高系统性能。

作者简介

肖磊　男　硕士研究生，主要研究方向为控制工程与控制理论。

参考文献

[1] 李宁．基于MDK的STM32处理器开发应用[M]．北京：北京航天航空大学出版社，2009．

[2] Joseph Yiu．The Definitive Guide to me ARM Cortex-M3．

[3] 任哲．嵌入式实时操作系统μ C/OS-II原理及应用[M]．北京：北京航空航天大学出版社，2006．

[4] 陈瑶．Cortex-M3+μ C/OS-II嵌入式系统开发入门与应用[M]．北京：人民邮电出版社，2010．

[5] Jean J.Labrosse．嵌入式实时操作系统μ C/OS-II[M]．北京：北京航空航天大学出版社，2007．

© 1994-2011 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net