doi: 10.7690/bgzdh.2016.07.017

μC/OS-Ⅲ在 STM32F103RC 上的移植

唐小平
(绵阳市维博电子有限责任公司传感器技术部，四川 绵阳 621000)

摘要：针对在 STM32F103RC 处理器组建的系统上移植 μC/OS-Ⅲ时需要一定移植条件的问题，介绍一种移植 μC/OS-Ⅲ的方法，Cortex-M3 内核处理器为 μC/OS-Ⅲ移植带来的便利性以及 μC/OS-Ⅲ移植工作中涉及到的 3 方面内容，重点说明 μC/CPU 移植文件编写与修改的细节，并详述基于 STM32F103RC 产品平台下的 μC/OS-Ⅲ成功移植案例。实践结果表明：该方法具有可操作性，能够实现 μC/OS-Ⅲ在 STM32F103RC 上的移植。

关键词：μC/OS-Ⅲ；实时操作系统；任务调度；任务堆栈；STM32F103RC；Cortex-M3
中图分类号：TP311.54 **文献标志码**：A

Transplantation of μC/OS-Ⅲ on STM32F103RC

Tang Xiaoping
(*Department of Sensor Technology*, *Mianyang Weibo Electronics Co*., *Ltd.*, *Mianyang* 621000, *China*)

Abstract: Aiming at the problem that certain conditions are needed when transplants μC/OS-Ⅲ on STM32F103RC, this paper introduces a method of transplanting C/OS- III. This paper introduces the method for transplanting μC/OS-Ⅲ. The paper introduces convenience which bring by Cortex-M3 kernel and three aspects of content about μC/OS-Ⅲ transplantation, focuses on introducing the details of file edition and revision of the μC/OS-Ⅲ transplantation, and described the successful transplantation of μC/OS-Ⅲ on STM32F103RC. The results show that this method has the maneuverability, and it can achieve the transplantation of μC/OS-Ⅲ on STM32F103RC

Keywords: μC/OS-Ⅲ; real-time operating system (RTOS); task schedule; task stack; STM32F103RC; Cortex-M3

0 引言

在单片机嵌入式领域使用实时操作系统 RTOS，是现在系统设计的需要。μC/OS-Ⅲ是源代码公开的商用嵌入式实时操作系统内核，由著名的 μC/OS-Ⅱ发展而来。μC/OS-Ⅲ针对以 ARM Cortex 为代表的新一代 CPU，面向带有可用于优先级查表的硬件指令的 32 位 CPU 嵌入式应用。μC/OS-Ⅲ允许利用这类高端 CPU 的特殊硬件指令来实现高效的任务调度算法，而无需使用 μC/OS-Ⅱ的软件任务调度算法，且 μC/OS-Ⅲ支持时间片轮转调度算法。从核心任务调度算法的改变来看，μC/OS-Ⅲ已经是一个全新的嵌入式 RTOS 内核。工业调查一致显示操作系统 μC/OS-Ⅲ是嵌入式领域中的最为流行的操作系统之一，为数以百万计的嵌入式系统开发者提供有效的解决方案。

1 移植条件

1.1 处理器要求

1) 处理器有可用的 ANSI C 编译器，能生成可重入代码。

2) 处理器支持中断，并且可以产生定时中断。

3) 处理器可以开关中断。

4) 处理器支持能够容纳足够多的数据(数千字节)的硬件堆栈。

5) 处理器有将堆栈指针和其他 CPU 寄存器读出并存储到堆栈或内存中的指令。

6) 处理器有足够的 RAM 空间用来存储 μC/OS-Ⅲ的变量、数据结构体和内部任务堆栈。

1.2 编译器要求

1) 编译器必须包括汇编器、C 编译器和链接/定位器。2) 编译器应支持 32 位数据类型。对一些高速 32 位处理器，编译器还应支持 64 位数据类型。

根据上述要求，笔者选用 STM32F103RC 为处理器，选用 MDK460 为编译器。

2 STM32F103RC 处理器简介[1]

STM32F103RC 单片机基于专为要求高性能、低成本、低功耗的嵌入式应用专门设计的 ARM Cortex-M3 内核，CPU 时钟频率 72 MHz，1.25DMips/MHz，2×12 位 ADC，256 k Falsh，48 K RAM，12 个独立的可配置的 DMA 通道。

3 Cortex-M3 内核为移植带来便利性[2]

Cortex-M3 内核的一个关键部件是嵌套向量中断控制器(NVIC)，为控制器提供一个标准的中断架

收稿日期：2016-03-25；修回日期：2016-05-03
作者简介：唐小平(1976—)，男，四川人，学士，从事智能电量隔离传感器的设计与开发。

构和异常中断处理，NVIC 自动保存了半数以上的 CPU 寄存器，当中断退出时，又恢复它们，使中断处理更加有效率。

Cortex-M3 内核的控制器支持 2 种操作模式：线程模式(Thread Mode)和处理模式(Handle Mode)，2 种方式都可以配置自己的堆栈(进程堆栈与主堆栈)，任务执行在线程模式时使用进程堆栈，中断执行在处理模式时使用主堆栈。当一个异常发生的时候，任务的上下文自动保护到进程堆栈里面，于是，处理器进入处理器模式，把主堆栈激活，从异常返回，任务上下文恢复，重新回到线程模式。

NVIC 的一个异常是 PendSV(可悬起的系统调用)。PendSV 的典型使用场合是在上下文切换时(在不同任务之间切换)。PendSV 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。为实现这个机制，需要把 PendSV 编程为最低优先级的异常[3]。

Cortex-M3 内核的控制器包括一个 24 位的向下计数器，具备自动加载和完成后中断的功能，称为 SysTick。SysTick 的设计是作为 RTOS 的系统时钟节拍中断。

μC/OS-Ⅲ使用PendSV向量完成上下文切换，使用SysTick向量完成系统时钟节拍中断。

这些特性给实时内核 μC/OS-Ⅲ提供了更大的便利性。

4　移植涉及的工作[4]

移植工作涉及 3 方面的内容：CPU、操作系统和板极代码。板极代码也就是板极支持包(BSP)，对于 μC/OS-Ⅲ来说，很少用到。

移植包括 μC/OS-Ⅲ移植和 μC/CPU 移植文件。

4.1　μC/OS-Ⅲ移植文件

移植需要编写 3 个文件：os_cpu.h、os_cpu_c.c、os_cpu_a.asm。

4.1.1　os_cpu.h

os_cpu.h 包含处理器和应用相关的常量、宏及类型定义，μC/OS-Ⅲ函数原型声明。

```
#define NVIC_INT_CTR *((CPU_REG32 *)0xE00ED04
```

宏定义中断控制与状态寄存器

```
#define NVIC_PENDSVSET 0x10000000
#define OS_TASK_SW() NVIC_INT_CTRL = NVIC_ PENDSVSET
#define OSIntCtxSw() NVIC_INT_CTRL = NVIC_ PENDSVSET
```

在 Cortex-M3 中，任务级的上下文切换和中断级的上下文切换都通过 PendSV 异常来实现。

4.1.2　os_cpu_c.c

μC/OS-Ⅲ移植需要定义以下函数：

```
OSIdleTaskHook()
OSInitHook()
OSStatTaskHook()
OSTaskCreateHook()
OSTaskDelHook()
OSTaskReturnHook()
OSTaskStkInit()
OSTaskSwHook()
OSTimeTickHook()
OS_CPU_SysTickHandler()
OS_CPU_SysTickInit()
```

其中，重点关注这 3 个函数的编写

```
OSTaskStkInit()
OS_CPU_SysTickHandler()
OS_CPU_SysTickInit()
```

后 2 个函数一个是系统时钟中断与系统时钟初始化。下面重点介绍 OSTaskStkInit()函数。

4.1.3　OSTaskStkInit()

该函数初始化正在创建任务的堆栈帧。当 μC/OS-Ⅲ创建一个任务时，使任务的堆栈看起来好像刚发生中断，模拟中断将任务现场保存到任务堆栈中。OSTaskStkInit()函数返回前的堆栈框架结构如图 1 所示。创建任务时 OSTaskCreate()函数会将新的栈顶(p_stk)保存到创建任务的任务控制块 OS_TCB 中。

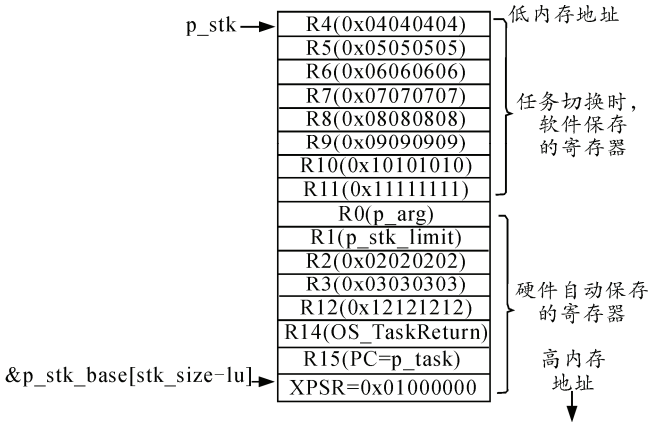

图 1　OSTaskStkInit()函数创建的堆栈框架

4.1.4　os_cpu_a.asm

OS_CPU_A.ASM 包含特定处理器相关的 4 个函数代码，以汇编语言实现：OSStartHighRdy()、

OS_TASK_SW() 和 OSIntCtxSw()，OS_CPU_PendSVHandler()。注意这4个汇编函数的改写。

OSStart()函数会调用本函数来开始多任务管理。在调用之前，OSStart()函数会首先找出先前已经创建的任务中优先级最高的就绪态任务(OSTCBHighRdyPtr 会指向该任务的任务控制块OS_TCB)。图2是指针OSTCBHighRdy->StkPtr指向的堆栈结构。

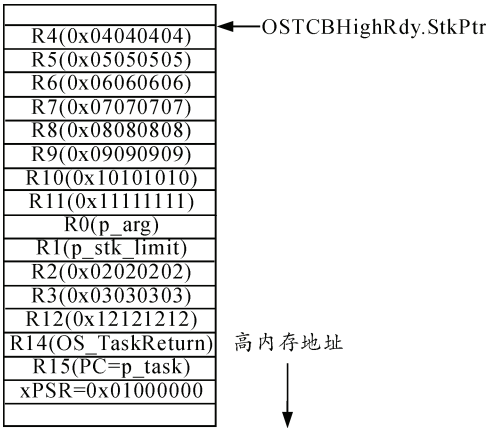

图2 指针OSTCBHighRdy->StkPtr指向的栈结构

OS_TASK_SW()被OSSched()调用，OSSched()执行任务级的上下文切换。OSIntCtxSw()被OSIntExit()调用，在中断服务ISR完成后，执行上下文切换。这些函数只是触发PendSV中断处理程序，并不执行真正的上下文切换。

OS_CPU_PendSVHandler()用于在任务中，或者在完成中断服务ISR后，执行上下文切换。OS_CPU_PendSVHandler()被OSStartHighRdy()、OS_TASK_SW()和OSIntCtxSw()调用。

4.2 μC/CPU移植文件[5]

μC/CPU包含CPU相关的功能封装文件以及编译器相关的数据类型定义。

移植需要修改的文件：cpu_cfg.h、cpu.h、cpu_a.asm。

4.2.1 cpu_cfg.h

cpu_cfg.h包含一个实际项目的μC/CPU配置模板。cpu_cfg.h决定是否使能中断禁止时间测量，CPU是否支持用汇编语言实现前导零计数指令或者用C模拟。

4.2.2 cpu.h

cpu.h包含处理器和应用相关的常量、宏定义及类型声明。μC/OS-Ⅲ不使用标准的C数据类型，而是声明更直观的数据类型以便于移植。cpu.h还包含一些函数原型声明。

4.2.3 cpu_a.asm

cpu_a.asm包含了一些是用汇编语言编写的函数，可用来开中断和关中断、计算前导零，以及其他一些只能用汇编语言编写与CPU相关的函数。

μC/OS-Ⅲ特别重要的3个函数：CPU_SR_Save()，CPU_SR_Restore()，CPU_CntLeadZeros()。

5 μC/OS-Ⅲ移植后的应用实例[6]

实例：用STM32F103RC内部12位AD实现1路直流电压与1路直流电流的采样与计算(ADC1实现电压采样，ADC2实现电流采样)，并用LED灯以0.5 s时间间隔闪烁指示CPU正常工作状态，通过RS485通信口实现参数传递。系统中断与系统任务划分情况见表1、表2。

表1 系统中断

中断	功能描述
Timer2	用于启动AD DMA操作，AD的采样频率3.2 kHz
DMA1	ADC1，ADC2数据读取，发送数据采样完成标志
SysTick	μC/OS-Ⅲ系统时钟节拍中断
PendSV	在这个中断里实现系统任务切换
USART1	实现RS485通讯数据的接收与发送

表2 系统任务

任务序号	功能描述
1	LED灯闪烁指示任务
2	等待采样数据完成标志，完成参数计算分析
3	实现通讯任务处理

6 系统任务函数示例

```
static OS_TCB Task1TCB,Task2TCB, Task3TCB;
static
CPU_STKTask1Stk[128],Task2Stk[128],Task3Stk[128];
static  void  Task1(void* p_arg);
static  void  Task2(void* p_arg);
static  void  Task3(void* p_arg);
int main(void)
{  OS_ERR      err;
   STM32Init();
   OSInit(&err);
   OSFlagCreate ((OS_FLAG_GRP *)&myflag,
                 (CPU_CHAR *)"myflag_1",
                 (OS_FLAGS )0,
                 (OS_ERR *)&err);
   OSTaskCreate((OS_TCB   *)&Task1TCB,
                 (CPU_CHAR *)"Task1 Start",
                 (OS_TASK_PTR )Task1,
                 (void *)0,
                 (OS_PRIO )5,
                 (CPU_STK *)&Task1Stk[0],
```

```
                    (CPU_STK_SIZE)12,
                    (CPU_STK_SIZE)128,
                    (OS_MSG_QTY )0,
                    (OS_TICK )0,
                    (void *)0,
    (OS_OPT)(OS_OPT_TASK_STK_CHK|
    OS_OPT_TASK_STK_CLR),
                (OS_ERR      *)&err);
    ……
                  OSStart(&err);   }
static    void   Task1(void *p_arg)
{OS_ERR err;
while(1)
{    OSFlagPend((OS_FLAG_GRP*)&myflag,
    (OS_FLAGS ) 0x04,
    (OS_TICK ) 0,
    (OS_OPT) OS_OPT_PEND_FLAG_SET_ANY +
    OS_OPT_PEND_FLAG_CONSUME+
    OS_OPT_PEND_BLOCKING,
    (CPU_TS *)&ts,
     (OS_ERR *)&err);
     ComputeParam();
    }}
```

（上接第 56 页）

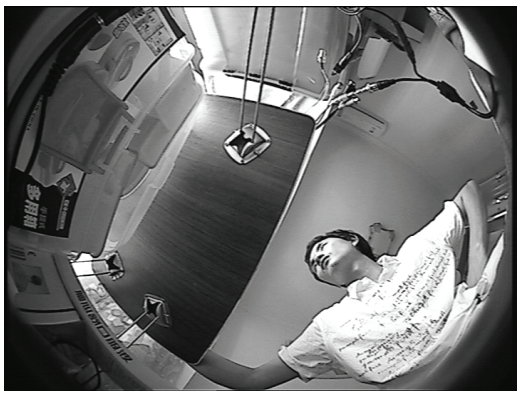

图 4　校正前的鱼眼图像

图 5　校正后的鱼眼图像

从校正效果图可看出：未校正前，左边的盒子边缘已经严重变形(图 4 左下侧)，校正后的盒子边缘则恢复正常(图 5 左下侧)，校正效果显著。通过对比专用的畸变测试图，测得校正后的鱼眼图像的相对畸变不大于 5%，能满足一般观测应用要求。

6　结束语

笔者所设计的鱼眼校正方法，能进行全视场的校正，校正效果能满足要求，实时性强。鱼眼镜头一般接近或等于 180°，甚至 230°[5]，同时还有 4 种类型。文中虽然只针对 180°的等距投影鱼眼做出校正实例，但实际上只要知道镜头参数或测量了相关畸变数据，所有的鱼眼镜头所拍摄的视频图像都可采用文中所用的 Chebyshev 插值方法得到其多项式校正函数，然后采用 FPGA 或其他器件进行校正。

参考文献：

[1] 王永仲. 现代军用光学技术[M]. 北京: 科学出版社, 2003: 387.

[2] 陈晃明, 陈向颖. 鱼眼镜头光学设计[J]. 北京理工大学学报, 1989, 9(3): 35-42.

[3] Uwe Meyer-Baese. 数字信号处理的 FPGA 实现[M]. 刘凌, 胡永生, 译. 北京: 清华大学出版社, 2003: 56.

[4] 丁学恭, 吴晓苏. 切比雪夫插值用于等幅电流矢量微步距控制[J]. 电气传动, 2007, 37(6): 45-47.

[5] 席志红, 孙丽华, 孙昭光. 基于鱼眼镜头的全视觉图像[J]. 应用科技, 2007, 34(12): 8-11.

7　结论

得益于 μC/OS-Ⅲ在设计时对可移植性的充分考虑，按照前面介绍的方法，μC/OS-Ⅲ的移植相对来说是比较容易的。笔者在基于 STM32F103RC 的产品平台下，成功实现了 μC/OS-Ⅲ在 STM32F103RC 上的移植。实践结果表明：该方法具有可操作性，能够实现 μC/OS-Ⅲ在 STM32F103RC 上的移植。

参考文献：

[1] ST 公司. STM32F10xxx 用户守则[S]. 瑞士: ST 公司, 2010: 1-21.

[2] Joseph Yiu. Cortex-M3 权威指南[M]. 宋岩, 译. 北京: 北京航天航空大学出版社, 2009: 83-107.

[3] 张乾, 梁森, 韦利明. 嵌入式共固化耐高温阻尼复合材料结构的模态分析试验研究[J]. 四川兵工学报, 2015, 36(2): 76-79.

[3] Labrosse Jean J. 嵌入式实时操作系统[M]. 北京: 北京航空航天大学出版社, 2012: 14-316.

[5] Labrosse Jean J. 嵌入式实时操作系统 μC/OS-Ⅲ应用开发[M]. 北京: 北京航空航天大学出版社, 2012: 7-136.

[6] ST 公司. STM32F10xx 数据手册[S]. ST 公司, 2012: 1-89.