doi: 10.3969/j.issn.1006-1576.2011.03.030

μC/OS-II 在某高炮火控系统中的应用

姚春[1]，许俐[2]
（1. 中国兵器工业第 58 研究所 军品部，四川 绵阳 621000；
2. 中国兵器工业第 58 研究所 信息中心，四川 绵阳 621000）

摘要：为提高系统的实时性和可靠性，在某高炮火控系统显控终端的开发中，将 μC/OS-II 内核移植到 C8051045 硬件平台上。分析 μC/OS-II 系统在 C8051045 上的应用，给出其系统测试步骤，并以 C8051045 单片机为应用平台进行 μC/OS-II 系统开发。该系统已成功应用于某高炮火控系统显控终端设计中，能满足多任务下的实时性要求，使整个系统更加稳定、可靠。

关键词：μC/OS-II；显控终端；C8051F045；实时性
中图分类号：TP311.54 **文献标志码**：A

μC/OS-II Application in Certain Type Flack Fire-Power Control System

Yao Chun[1], Xu Li[2]
(1. Dept. of Armament Products, No. 58 Research institute of China Ordnance Industries, Mianyang 621000, China;
2. Information Center, No. 58 Research institute of China Ordnance Industries, Mianyang 621000, China)

Abstract: In order to improve the real-time performance and reliability, in researching and developing display and control terminal of certain type flack fire-power control system, transplant μC/OS-II inner core into C8051045 hardware platform. Analyze the application of μC/OS-II system in C8051045, put forward the system testing steps, research and develop μC/OS-II system based on C8051045 single chip. The system is successfully used in display and control terminal design of certain type flack fire-power control system. It can satisfy the real-time requirements under multi-task, and make the system more stable and reliable.

Keywords: μC/OS-II; display and control terminal; C8051F047MCU; real-time performance

0 引言

某高炮火控系统显控终端是基于 C8051 硬件平台下开发的集成人机交互界面、键盘处理、实时通信和时钟管理等各功能模块的复杂部件。MCU 选用 C8051 硬件平台具有先天的可靠性和较大的成本优势，但如果单独在 C8051 上开发应用程序，在处理多任务调度时，会因程序流程复杂而导致难以保障其实时性、可靠性，故可对其植入嵌入式实时操作系统来管理硬件和软件资源[1]。

在我国军事应用中的实时控制领域，尤其是陆军武器装备嵌入式应用中，长期以来使用 Vxworks 等国外的商业内核，由于这些商业内核一般不公开内核源码，给系统安全带来了隐患。μC/OS-II 是一个免费的源码公开的嵌入式实时内核，结构小巧，核心代码只占用 8.3 K 字节，μC/OS-II 是可裁剪的，代码最少可达 2.7 K 字节，非常适合移植。因此，笔者将 μC/OS-II 系统移植到 C8051F045 单片机上，并运用到某高炮火控系统显控终端的设计中。

1 μC/OS-II 系统在 C8051F045 上的应用

要应用 μC/OS-II 系统，首先要在 C8051F045 上移植该系统；其次，在运行稳定正常的系统上进行开发。另外，还需要一个面向 C8051F045 的 C 编译器作为系统工作的环境。笔者使用 Keil 软件公司的 Keil uVision3 V8.09，支持 C 和汇编混合编程。

在 C8051F045 上移植 μC/OS-II 系统，移植的主要工作就是以 C8051F045 为处理器，修改与 C8051F045 相关的几个文件。

1.1 修改 INCLUDES.H 文件

对于 C8051F045，需要在 INCLUDES.H 中增加的头文件如下：

```
#include "os_cpu.h"
#include "os_cfg.h"
#include "μcos_ii.h"
```

1.2 修改 OS CPU.H 文件

为确保 C8051F045 系统在 KEIL 环境下正常运行，在 OS_CPU.H 中重新定义了一系列与 C8051F045 和 KEIL 编译器相关的数据结构、宏和常数。

要特别注意的是，C8051F045 单片机中堆栈是

收稿日期：2010-10-28；修回日期：2010-11-26
基金项目：总装备部型号项目（陆装科订部 407 号）
作者简介：姚春（1984—），男，重庆人，学士，助理工程师，从事火力控制领域研究。

按字节操作的，堆栈数据类型 OS_STK 声明为 8 位：

```
typedef unsigned char OS_STK; /*定义堆栈宽度为 8 位*/
typedef unsigned char OS_CPU_SR;
```

在 C8051F045 单片机中，堆栈是从低地址向高地址增长的，所以 OS_STK_GROWTH =0。

临界段的宏的设置利用 C8051F045 单片机的开中断和关中断指令来实现[2]：

```
#define OS_ENTER_CRITICAL() EA="0"
#define OS_EXIT CRITICAL()  EA="1"
```

OS_TASK_SW() 函数的定义：

```
#define Os_TASK_SW() OSCtxSw()
```

1.3 修改 OS_CPU_A.ASM 文件

用户需要编写 4 个函数的代码：OSSTartHihgRdv()、OSCtxSw()、OSIntCtxSw()和 OSTickISR()。另外，由于 C8051F045 在内部 RAM 上有所欠缺，需要利用外部程序存储空间作任务堆栈映像。函数代码实现方法如下：

1.3.1 编写 OSSTartHihgRdy()函数

对于将要恢复运行的就绪任务，获得其堆栈映像的最低地址，并计算出堆栈长度，然后，向系统堆栈复制数据、堆栈指针 SP 和堆栈映像指针?C_XBP，最后，利用中断返回。

1.3.2 编写 OSCtxSw()函数

先从当前任务的 TCB 控制块中获得当前任务堆栈长度和堆栈映像指针，然后，将系统堆栈的内容复制到任务堆栈映像，最后，获得将要恢复运行的就绪任务的 TCB，程序跳至 OSSTartHihgRdy()函数的入口，实现任务的切换。

1.3.3 编写 OSIntCtxSw()函数

OSIntCtxSw()函数代码大部分与 OSCtxSw()相同，不同之处在于：此处不需要再保存 C8051F045 寄存器；需要调整堆栈指针(SP=SP-4)，去掉在调用 OSIntExit()，OSIntCtxSw()中压入堆栈中的多余的内容，使堆栈中只包含任务的运行环境。

1.3.4 编写 OSTickISR()函数

OSTickISR()函数定位于51 单片机的定时器中断1上，它通过调用OSTime Tick ()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。

1.4 修改 OS_CPU_C.C 文件

编写 OSTaskStkInit()函数用来初始化堆栈。需要注意的是：在所用的 KEIL 编译环境中，任务参数是通过寄存器 R3、R2、R1 传递，而不是通过虚拟堆栈传递。如：

```
*stk++=(INT16U)dpdata&0xFF; //R1
*stk++=(INT16U)dpdata>>8; //R2
*stk++=0x03; //R3
```

2 系统测试

μC/OS-II 系统移植完成以后，需要检验系统是否能运行正常。通过参考其作者的测试方法，进行了 3 个步骤的测试：

验证 OSTaskStkInit()和 OSStartHighRdy()函数;

验证 OSCtxSw()函数;

验证 OSIntCtxSw()和 OSTickISR()函数。

测试结果证明，系统在 C8051F045 上运行正常。

3 相关问题

3.1 堆栈长度

从理论上讲，堆栈总是越长越好，但现实却不允许。μC/OS-II提供了一个OSTaskStkTest()函数[3]，专门用于测试所需任务堆栈的长度，只需稍加改动，也可被用来测试任务堆栈附加段的长度[4]。在堆栈长度不确定时，堆栈及其附加段空间应尽量设置大些。如果为附加段提供一个指针，可以提高系统查询效率；如果把附加段和仿真堆栈合并就更好。

3.2 函数嵌套

在任务调度过程中，必须限制函数嵌套的层数，否则系统很容易因堆栈溢出而崩溃。工作堆栈一般不会超过 80 个字节，如每层 15 个字节，嵌套不应超过 5 层。可设计一个全局变量来控制嵌套的层数。

3.3 中断开关时机

在多任务的调度过程中，如果对中断的开或关处理不当将会造成系统死机，为避免这种情况，要采取一些措施，如：不在任务建立前打开定时器开关，以免造成提前对任务进行调度。这个开关不是指 EA，而是指 ET，用 TR 也可以，因为在第一个任务建立前，EA 已经被打开过了。

3.4 存储覆盖问题

在μC/OS-II移植到单片机时，当仿真堆栈向下生长时，可能与存储器中的其它变量相互覆盖，还有可能造成其它数据结构和变量之间的覆盖。对编译系统来说，解决这一问题的最好办法是把部分失控的仿真堆栈安排在存储器的顶部，另外，利用操

作系统提供的存储器管理功能，也可解决这类问题。

4 C8051F045平台上的μC/OS-II系统开发

μC/OS-II系统稳定正常运行后，就可以依此为应用平台进行项目开发。在软件设计中，对于应用程序的任务的建立，根据μC/OS-II系统中建立任务的格式，确定任务的个数，设置好优先级。创建好任务后，在主函数外面分别列出各个任务函数，每个任务函数都是一个无限循环程序，在无限循环中调用实现某些功能的应用程序函数，然后按设计的需求设置挂起方式和挂起时间。将用汇编语言编写的中断服务子程序放在OS_CPU_C.C文件中，这些函数供主函数和任务调用。下面通过编写一个简单的测试应用程序来进行分析，测试应用程序实现了4个任务：Task1是每1 s 通过CAN发送显控终端输入的命令数据包，Task2是每4 s发送自检测数据包，Task3是处理CAN接收到的火炮参数信息，Task4是每2 s通过串口发送当前采集的火炮位置数据，波特率为9 600 kb/s。程序中设置了2个不同ID的message object用来发送CAN数据包，CAN总线接收采用中断方式，其优先级高于其他任务，为了保证系统的实时性，在中断程序中不处理数据，只是发送一个信号量，在Task3中处理CAN数据。发送串口数据采用的是查询方式，按字节发送。程序中设置4个任务的优先级依次为10、11、8、12，因此，Task3的优先级最高；Task1发送的CAN数据包的ID为0X001，Task2发送的CAN数据包的ID为0X002，因此，在CAN传输中Task1发送的数据包的优先级高，整个应用程序的结构如图1。

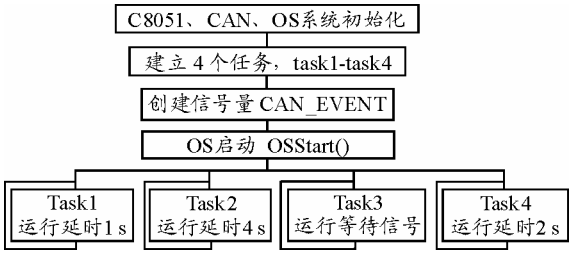

图1 各任务在μ-COSII中运行的结构图

在主程序中，首先初始化C8051F045和CAN，并进行系统初始化，调用OsInit ()函数，此函数中创建了一个空闲任务；然后调用API函数，创建4个任务（不包括空闲任务）；之后创建一个信号量CAN_EVENT，为中断与Task3通信所用，最后调用OSStart ()函数，OS系统开始运行优先级最高的任务。Task3的优先级最高，但在没收到CAN_EVENT之前，任务一直处于休眠状态，当CAN接收器收到数据包后，Task3进入就绪态，在中断返回时，进行任务切换，执行优先级最高的Task3。在Task3还未收到信号量之前，Task1、Task2和Task4根据时间延时和优先级的不同独立运行。从测试结果可见：当执行2次Task1后，Task4执行1次，当执行4次Task1后，Task2执行1次，这说明程序的任务调度和实时性都得到了很好的保证。

另外，为了提高代码效率，CAN中断服务程序是用汇编语言编写的：

```
CSEG AT 009BH;        CAN中断
LJMP Can ISR
RSEG ? PR? _? Can ISR? OS_CPU_A
Can ISR:
USING 0
CLREA;                关中断
PUSHALL
MOV R0, #LOW (OSIntNesting)
INC @R0;              置标志位,表明在执行中断程序
MOV R0, #LOW (CAN_EVENT)
MOV AR3, @R0
INC R0
MOV A, @R0
MOV R2,A
INC R0
MOV A, @R0
MOV R1,A
LCALL _?OSSemPost;    发送CANEVENT信号量
LCALL _?OSIntExit;    中断结束前进行任务切换
POPALL
RETI
```

5 结束语

笔者将μC/OS-II系统移植到C8051F045单片机上，并成功运用到某高炮火控系统显控终端的设计中，满足了在多任务下的实时性要求，使得整个系统更加稳定可靠，取得了预期效果。

参考文献：

[1] 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: 37-38.

[2] 楚育军. 利用实时内核开发嵌入式多任务程序[J]. 单片机与嵌入式系统应用, 2004, 17(4): 1-4.

[3] Jean J. Labrosse. μC/OS-II源码公开的实时嵌入式操作系统[M]. 邵贝贝, 等 译. 北京: 中国电力出版社, 2001: 29-30.

[4] 薛钧义. 微机控制系统及其应用[M]. 西安: 西安交通大学出版社, 2003: 168-171.

[5] 姚雾云, 徐德友. 某型自行火炮火控系统检测系统[J]. 兵工自动化, 2010, 29(12): 69-72.