

doi: 10.7690/bgzdh.2015.04.018

## 内存对齐机制对移植 $\mu\text{cos-ii}$ 的影响

陈大鹏, 王 伟

(中国兵器工业第五八研究所军品部, 四川 绵阳 621000)

**摘要:** 为提高  $\mu\text{cos-ii}$  移植代码的稳定性和可靠性, 避免在实际使用过程中可能出现的问题, 针对 DSP TMS320F28335 在进行 32 位数据存取时, 要求偶地址对齐这一特殊情况, 通过仔细分析中断过程中堆栈指针的变化, 对  $\mu\text{cos-ii}$  的任务栈空间结构做出了相应的调整, 并基于这种调整给出了移植函数及应用程序编写的指导性说明, 同时设计了一个用于测试移植代码的实验。结果表明: 文中所提的移植方法是可行的, 能够使  $\mu\text{cos-ii}$  达到任务调度的目的。

**关键词:** TMS320F28335; 内存对齐;  $\mu\text{cos-ii}$ ; 栈空间结构

**中图分类号:** TP311.1 **文献标志码:** A

## Effect of Memory Alignment Mechanism for Transplant $\mu\text{cos-ii}$

Chen Dapeng, Wang Wei

(Department of Military Products, No. 58 Research Institute of China Ordnance Industry, Mianyang 621000, China)

**Abstract:** In order to improve the stability and reliability of the codes for transplant  $\mu\text{cos-ii}$  and avoid possible problems in the using process, this paper analyzed the changes of stack pointer in the interrupt process carefully by contraposing to the fact that DSP TMS320F28335 expects memory wrappers or peripheral-interface logic to align any 32-bit read or write to an even address, made some adjustments of the task stack structure of  $\mu\text{cos-ii}$  and gave a guidance of coding for transplant functions and applications based on these adjustments. At last, an experiment was designed to test the transplant codes. The result shows that this method is feasible and can achieve the task switch purpose of  $\mu\text{cos-ii}$ .

**Keywords:** TMS320F28335; memory alignment;  $\mu\text{cos-ii}$ ; stack structure

### 0 引言

$\mu\text{cos-ii}$  是 Jean J. Labrosse 先生设计的可以媲美很多商用实时系统的开源微内核, 其代码编写非常规范, 很容易理解, 并且通过了严格的测试, 在工程实际中得到了广泛的应用<sup>[1]</sup>。

TMS320F28335 (以下简称“F28335”)是德州仪器推出的专用于控制系统设计的 32 位浮点 DSP, 工作频率最高可达 150 MHz, 因其片内资源丰富, 同时又有良好的集成开发环境 CCS (code composer studio), 所以被广泛地用于电机控制、电力设备控制以及工业控制等领域<sup>[2]</sup>。

基于上述 2 个原因, 近年来已有学者开始研究如何在 F28335 上移植并使用  $\mu\text{cos-ii}$ <sup>[3]</sup>。然而, 这些文章中都没有考虑到 F28335 在进行 32 位数存取时要求偶地址对齐这一关键性问题, 同时在很多细节上也存在不足之处, 导致相应的移植不够完善, 存在一定的安全隐患。因此, 笔者认真研究了这些被忽略的环节, 使整个移植代码更加稳定可靠。

### 1 F28335 的内存对齐机制

F28335 内部存储空间是 16 位。由于其特殊的

架构特点, 在进行 32 位数存取时, 要求偶地址对齐, 即这个数的低 16 位必须存储在偶地址上。如果地址生成逻辑产生了一个奇地址, 那么 CPU 必须在前一个偶地址上开始读写, 然而这种对齐机制并不影响地址生成逻辑所产生的地址<sup>[4]</sup>。

图 1 所示为一个 32 位数在入栈时, 由于堆栈指针 SP 奇偶性的不同, 对操作结果的影响。

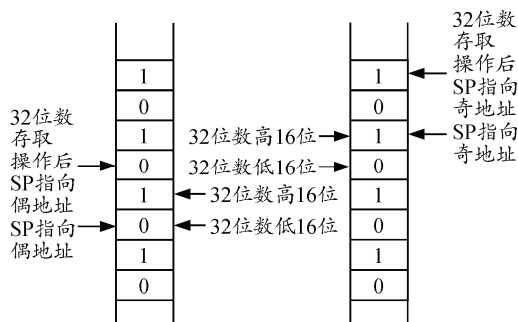


图 1 堆栈指针 SP 的奇偶性对 32 位数存取的影响

### 2 内存对齐机制对移植的影响

#### 2.1 内存对齐机制对任务栈结构的要求

如第 1 节所述, F28335 这种特殊的内存对齐机制对  $\mu\text{cos-ii}$  的任务栈结构提出了新的要求, 必须对

收稿日期: 2014-11-06; 修回日期: 2014-12-13

作者简介: 陈大鹏(1989—), 男, 安徽人, 在读硕士, 从事信息融合以及目标信息处理技术研究。

其进行合理的设计。

1) 栈底指针以及栈顶指针。

$\mu\text{cos-ii}$  的任务堆栈通过函数 OSTaskStkInit() 进行初始化<sup>[1]</sup>, 传递给 OSTaskStkInit() 的参数有任务的入口地址、任务的堆栈起始地址以及任务所需的参数(一个空类型指针), OSTaskStkInit() 的返回值是一个指向栈顶的指针。

由于一个任务中可能会产生 32 位数的存取操作(例如函数调用时, 传递 32 位的参数), 因此要保证这个任务的堆栈起始地址为偶地址, 若为奇地址, 如图 1 在传递 32 位参数时有可能破坏堆栈中原有的数据, 这样会出现意想不到的结果。故为了不在应用程序里增加调整堆栈指针 SP 奇偶性的代码, 在给每个任务分配堆栈空间时, 传递给 OSTaskStkInit() 的栈底指针, 也就是任务的堆栈起始地址应是偶地址。基于同样的原因, 中断服务程序在保存了所有寄存器后, 堆栈指针也应该为偶地址, 即 OSTaskStkInit() 返回的栈顶指针必须是偶地址。

$\mu\text{cos-ii}$  一般是用数组的形式来为每个任务分配栈空间, 可用编译指示宏命令 #pragma DATA\_SECTION() 来保证数组的首地址为偶数。

2) 栈内容以及栈内容初始值。

为了达到任务调度的目的,  $\mu\text{cos-ii}$  要求在任务切换时任务堆栈要保存 CPU 的所有寄存器。F28335 在中断时 CPU 会自动压入一部分寄存器入栈, 这些寄存器都是 32 位, 由于堆栈指针 SP 指向的总是下一个可以使用的栈空间, 因此为了不破坏原堆栈中的数据, SP 值将自动加 1 后再进行压栈操作, 出栈情况正好相反, 在恢复寄存器后 SP 值将自动减 1<sup>[4]</sup>, 也就是说中断时 CPU 自动压栈导致堆栈指针 SP 要增加 15, 中断返回导致堆栈指针 SP 要减少 15。

如果按前文所述方法正确地初始化了栈底指针(偶地址), 那么在保存了 CPU 中断时自动保存的一部分寄存器后, 堆栈指针指向的应该是奇地址, 这时如果不做处理的话, 在保存了其他的寄存器后, 返回值也将是奇地址, 即栈顶指针是奇地址, 这不符合预期的分析, 解决此问题的一个可行方法是将 SP 加 1 后再保存其他寄存器。

在初始化寄存器时, 要特别考虑 ST1、IER 以及 DBGIER 的初值情况, 不合适的初始值将会导致移植的失败。

状态寄存器 ST1 包含了程序运行的重要信息, 根据文献[4], 可将 ST1 初始化为 0x8A08。中断使能寄存器 IER 一般设为 0x3FFF, 即允许全部中断,

也可以根据实际情况设置为其他值。DBGIER 称为调试中断使能寄存器, 仿真时, 如果想要一些中断在 CPU 暂停时也能响应, 应该设置相应的位为 1, 这里初始化为 0x0000, 即没有时间临界的中断。

此外, 通过研究 CCS 里函数调用过程发现, 如果传递给函数的参数是指针类型, 那么, 这个参数不是通过堆栈来传递, 而是通过寄存器 AR4 传递; 因此, 如果用 CCS 编译  $\mu\text{cos-ii}$  的话, 传递给任务的参数(一个空类型指针)一定要放在寄存器 AR4 中, 其中 AR4 放置参数低位, AR4H 放置参数高位。

一个初始化好的任务堆栈应该具有如图 2 所示的结构。

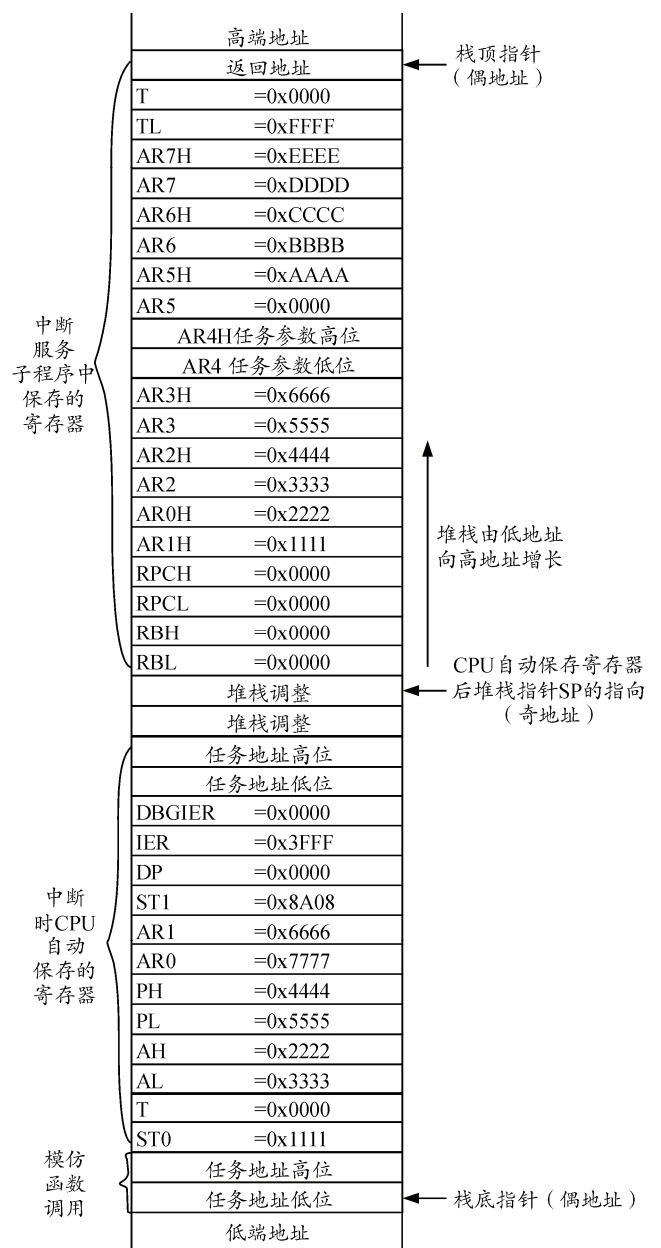


图 2 初始化后的任务堆栈结构

### 2.2 移植函数的编写

移植函数的编写要严格按照图 2 来进行, 使用如下的方法可确保任务堆栈的统一性:

1) 如果已把任务的堆栈起始地址初始化为偶数, 那么 CCS 的编译机制就会确保此任务在运行过程中堆栈指针 SP 总是偶数, 但在中断时由于 F28335 的硬件机制, 堆栈指针 SP 将变为奇数, 所以移植函数在保存 CPU 寄存器时, 应该加入 ASP 堆栈调整指令, 把堆栈指针 SP 调整为偶数。

2) 除 OSStartHighRdy()外, 其余函数在执行中断返回指令 IRET 前, 应加上 NASP 堆栈调整指令, 把堆栈指针 SP 调整为奇数。

3) 由于是开启  $\mu\text{cos-ii}$  的多任务, OSStartHighRdy()不会返回, 没必要保存 CPU 寄存器, 也就没有 ASP 指令, 所以不能用 NASP 指令调整堆栈, 其应该在执行中断返回指令 IRET 前直接把堆栈指针 SP 减 1。

### 2.3 浮点寄存器的保存

F28335 的浮点寄存器共有 10 个, 每个都是 32 位<sup>[5]</sup>。一个任务如果需要浮点运算, 那么在任务切换时必须保存 CPU 的浮点寄存器, 以免这些寄存器的内容被新任务破坏。

文献[6]详细地介绍了  $\mu\text{cos-ii}$  是如何保存 CPU 浮点寄存器的, 这里不再赘述, 仅对以下几点进行说明:

1) 并不是每个任务都需要进行浮点运算, 并且浮点寄存器的保存和恢复需消耗一定的 CPU 时间, 所以为保证应用程序的实时性, 浮点寄存器没有和通用寄存器那样, 在每次任务切换时都保存和恢复。

2) 通常是定义一个二维数组来保存浮点寄存器, 然而所有浮点寄存器的存取都是 32 位, 因此要保证这个二维数组的首地址是偶数。可用上文定义任务栈空间的方法来处理。

3) 可能有多个任务需要保存浮点寄存器, 因此用 OSMemCreate()函数所分配的内存块中每个分区必须包含偶数个存取单元, 根据 F28335 的浮点寄存器个数, 推荐为每个分区分配 40 个字节。

4) 应该用汇编语言编写 OSFPSave() 和 OSFPRestore()这 2 个函数, 传递给这 2 个函数的参数是指向分配给各自任务的用于保存浮点寄存器的存储空间的指针, 正如前文分析的一样, 此参数必须用寄存器 AR4 来传递。

### 2.4 其他注意事项

1) F28335 堆栈指针 SP 的寻址范围是 0x0000 0000-0x0000 FFFF, 因此在给任务分配栈空间时不能超过此范围。

2) 在 CMD 文件中合理地分配各种变量的存储空间, 提高存储空间的利用率, 以免发生存储空间不足的情况。

3) 中断服务程序一开始就应该按照图 2 所示的结构来保存 CPU 所有寄存器, 注意如果用 C 语言来编写, 千万不要加关键词 interrupt, 以免编译器对代码进行优化, 自动加入一部分汇编语言代码, 来保存中断函数里用到的寄存器, 这会破坏任务栈结构, 使多任务不能运行。

## 3 实验验证

该实验的软件平台是 TI 公司的集成开发环境 CCS3.3, 硬件平台是一块 F28335 的核心板, 笔者编写了 3 个任务, 其中一个任务为优先级最高的启动任务, 用来启动另外 2 个任务以及时钟节拍, 其余 2 个任务只做延迟处理以及浮点运算, 其中的延迟功能用  $\mu\text{cos-ii}$  提供的系统函数 OSTimeDly()来实现, 通过在程序的相应处打断点, 并不断地运行程序, 来判断  $\mu\text{cos-ii}$  是否能正常地调度任务。如果能调度则证明  $\mu\text{cos-ii}$  的移植是成功的。

测试的工作流程如图 3 所示, 其中的断点 1 和断点 2 是在 CCS 里设置的软断点。

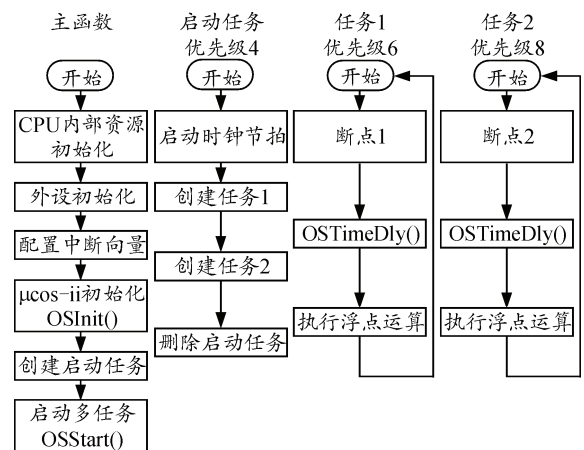


图 3  $\mu\text{cos-ii}$  移植测试流程

测试的具体步骤如下:

- 1) 主机通过仿真器连接到目标板。
- 2) 下载编写好的程序, 并在相应处打上断点。
- 3) 调出 CPU Registers 和 Watch Window 窗口, 在 Watch Window 窗口里监测 CPU 的浮点寄存器。