分析 uC/OS-II 在 MSP430 单片机芯片上实现 RTOS 的问题

单片机作为嵌入式信息产品的一个重要应用方面，其使用、设计面临着全新的挑战。一方面，人们对嵌入式产品的要求越来越高，稳定可靠、功能丰富、物美价廉的信息产品将成为人们的首选。另一方面，随着微电子工艺水平的发展，单片机处理器的能力不断提高，从最初的 8 位单片机到 16 位，进而 32 位单片机，功能越来越强大，执行速度越来越快，集成度、精确度也越来越高，应用领域进一步拓宽。可以说，单片机芯片的性能已经能够满足现代人们对嵌入式信息产品的更高要求。为了能将二者有效地结合起来，嵌入式 RTOS 的软件设计方法也取代了以前的前后台(超循环)设计方法，越来越受到重视和应用。

正如分时操作系统中 Linux 的出现打破了 Windows 一统天下的局面一样，由美国 Jean J.Labrosse 先生设计和编写的 uC/OS-II(Micro C OS 2)的出现也给国内的 RTOS 应用者带来了惊喜。uC/OS-II 的最大优点与 Linux 相同，即其源代码全部公开，使人们在应用它的同时能清楚地了解内部的实现细节，并且能够根据自己的需求进行移植和修改。特别重要的是 uC/OS-II 经过 8 年的发展，已经成功地在多个行业得到应用，保证了实用性和可靠性。它的出现改变了以前人们在使用 RTOS 时的态度，减少了经济上的顾虑，对于国内 RTOS 的研究、推广、应用将起到重要的推动作用。uC/OS-II 采用微内核设计，使用 C 语言编写，追求灵活性，可配置、可裁剪、可扩充、移植性强。需要强调的是 uC/OS-II 严格采用优先级抢占式调度方案。在创建任务时，根据任务的重要性给每个任务分配不同的优先级。任务调度时，先执行高优先级的任务，然后按照优先级由高到低执行任务。如果在某个任务执行中，激发了一个优先级更高的任务，那么在该任务执行结束后，将由任务调度器调度去执行所激发的高优先级任务，而不是顺序执行。

下面就 uC/OS-II 在 TI 公司生产的 MSP430F148 芯片上的移植和应用来探讨在单片机上实现 RTOS 可能遇到的一些问题。

1 MSP430 系列单片机简介

MSP430 系列单片机是由 TI 公司开发的 16 位单片机。其突出特点是超低功耗，非常适合于各种功率要求低的场合。有多个系列和型号，分别由一些基本功能模块按不同的应用目标组合而成。典型应用是流量计、智能仪表、医疗设备和保安系统等方面。由于其较高的性能价格比，应用已日趋广泛。

MSP430F148 是 TI 新近推出的 MSP430F14x/13x 系列单片机中的一款。相对 MSP430 系列的其它芯片，主要特点如下：

超低功耗。由于内置了功耗极低的快速闪存，因此，MSP430F14x/13x 系列在待机模式下所消耗的电能还少于电池未使用时的自然损耗。在正常的工作状态下，如果工作电压为 2.2 V，其典型消耗电流仅为 250uA/MIPS，而待机模式下工作电流降至仅 1uA 以下。

执行速度快。MSP430F13x/14x 系列的工作电压范围为 1.8~3.6 V，性能可达 8 MIPS。

存储容量大。MSP430F148 片内内置了 48 KB Flash ROM 和 2 KB RAM。RAM 空间是 MSP430 系列中最大的，基本符合运行 RTOS 的需要。

高性能 A/D。包含了 1 个具有 8 个外部通道的 12 位高性能 A/D 转换器。利用芯片内置的自动扫描功能，A/D 转换器可以不需要中央处理器的协助而独立工作。

集成度高。该器件还包括 1 个独立的看门狗、2 个脉宽调制定时器(PWM)、1 个比较器、2 个 USART 口以及 48 个输入/输出引脚等部件。

在线支持强。MSP430F13x/14x 系列均可由 MSP-FET430P140 闪速仿真工具(FET)提供支持。该 FET 是一种完整的集成开发环境，包括源代码级调试器、仿真器、汇编/连接器、C 编译器、2 种*估芯片、目标板、JTAG 接口以及编程单元等。

由以上介绍可以看出，MSP430F148 属于一种中低端的单片机，只具备运行 RTOS 的基本条件，所以在它上面运行 RTOS 所遇见的一些问题，对于一般的单片机而言是具有代表性的。

2 中断堆栈的结构设计

在 uC/OS-II 中，任务切换分为任务级切换和中断级切换。其中任务级切换是通过发软中断指令或依靠处理器执行陷阱指令来完成的。软中断指令会强制将一些处理器寄存器保存到当前任务的堆栈中，并执行任务调度。其目的是使处于就绪态的任务的堆栈结构看起来就像刚发生过中断并将全部寄存器保存在堆栈的情形一样。如 MCS-51 以及 x86 芯片都有类似的指令，但问题出在有一些单片机芯片中没有软中断指令，并且在发生中断时保存寄存器的情况根据单片机芯片和所使用的编译器的不同而有很大区别。

MSP430F148 中就没有软中断指令，所使用的 IAR 编译器在发生中断时也不保存所有的寄存器，而是只保存几个在中断中使用到的寄存器。所有这些都是不符合 uC/OS-II 的移植条件的。我们的解决方法是根据具体情况来自己定义一个中断结构，不论是在任务级调度还是中断发生或调度以及任务堆栈的初始化时，都要按照这个结构来执行。代价是必须对所编写的中断程序的汇编代码进行人工修改，使之符合这个中断结构。

为设计一个符合要求的中断堆栈结构，首先必须清楚所使用的单片机在发生中断时执行了哪些操作，即向堆栈中保存了哪些寄存器以及它们的顺序。当 MSP430 单片机发生中断时，只进行 2 条基本操作，先将 SR(状态寄存器)压入堆栈中保存，然后将中断发生时要执行的下一条指令的 PC 值压入堆栈保存。其次，要清楚所使用的 C 编译器在编译 C 语言编写的中断程序时，进行了哪些默认的操作。通过对所使用的 IAR V2.13 编译器编译产生的汇编程序进行分析，可以发现，

除了以上的 2 条基本操作以外，在中断程序的开头，还自动保存了 R12~R15 四个寄存器，余下的 R4~R11 八个寄存器中只保存在中断程序中用到的个别寄存器，而不是全部保存。但在 RTOS 中必须保存所有的寄存器，这样才能正确保存该任务的状态。通过以上分析，我们定义了 MSP430 运行 uC/OS-II 时的中断堆栈结构，如图 1 所示。

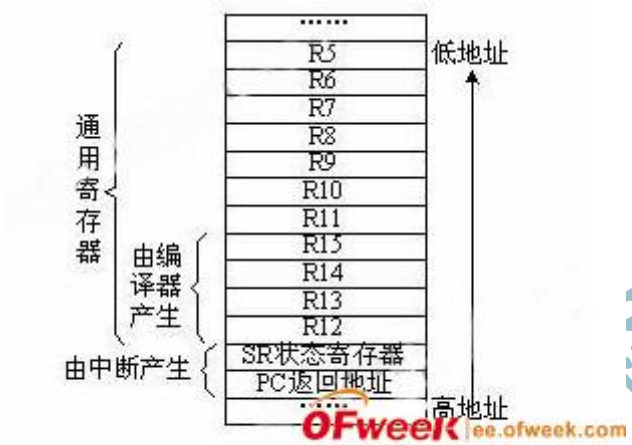

图1 MSP430F148的自定义中断堆栈结构

3 如何保证单片机的低功耗特性

单片机在嵌入式方面的应用都非常强调单片机的超低功耗特性。MSP430 系列的特点也在于此。如果由于运行 RTOS 而破坏了单片机的低功耗特性是得不偿失的。一般的单片机都规定了几种不同功耗的工作模式，可以根据具体的需求来选择。不同工作模式是通过读写 1 个或 1 组寄存器来控制 CPU、时钟、晶振以及外围设备的运行来实现的。

MSP430 系列单片机有 6 种不同的工作模式，都是通过对状态寄存器 SR 的读写来实现的。在 RTOS 中，由于每个任务都可以分别保存自己的状态，包括状态寄存器，所以在实现低功耗工作模式时更加灵活方便。首先，在设计每个任务时，都可以根据任务的具体要求定义它的工作模式。其次，在整个系统设计中，设计一个最低优先级的任务，其作用就是使系统进入特定的低功耗工作模式。这样，在其它任务都运行完毕后，系统会调用这个任务使整个系统进入低功耗工作模式。当其它任务又恢复运行时(如延时结束)，会自动进入其特定的工作状态，以达到降低功耗的目的。

4 如何减少 RTOS 在运行中占用的 RAM 空间

影响 RTOS 在单片机上应用的主要原因是由于在单片机上运行 RTOS 需要占用一定的系统资源，如系统时钟、RAM、FLASH 或 ROM 等，从而减少了应用系统对资源的利用。特别是对 RAM 的占用。一般而言，单片机上的内部 RAM 数量都很少(如 MSP430F148 是整个 MSP430 中 RAM 最多的，也只有 2 KB)，虽然可以通过外部扩展来增加 RAM 数量，但这样不仅增加了设计的难度和产品成本，而且有时还

使系统应用无法进行扩展。所以，最好的方法是能够充分利用单片机的内部 RAM 来运行 RTOS。

通过分析 uC/OS-II 对 RAM 的使用情况可知，占用 RAM 空间最多的原因，是由于在设计 uC/OS-II 时，要给每个任务都分配一个单独的任务堆栈。特别在单片机的硬件设计没有将中断堆栈与任务堆栈分开时，计算任务堆栈的大小时不仅要计算任务中变量和函数嵌套所使用的 RAM 大小，还必须计算该任务在运行时发生中断和中断嵌套所需要的 RAM 空间的大小。由于每一个任务均需预留中断和中断嵌套所需要的 RAM 空间的大小，所以使得大量 RAM 空间被浪费。最直接的解决方法就是利用软件来将任务堆栈和中断堆栈分离，使得在计算任务堆栈的大小时，只需计算任务本身所需的 RAM 空间大小，从而提高了 RAM 的使用效率，增加了更多的应用任务。

所谓将中断堆栈与任务堆栈分离，就是在内存中专门开辟出一块区域作为中断堆栈使用，任何一个任务运行时发生中断都只使用它。设计的原则：一是要尽量将中断任务与普通任务分开;二是模拟的中断堆栈指针必须一直保持在中断堆栈的顶部，即中断时中断堆栈指针要时刻保持与 SP 的同步变化。

为了达到这个目的，单片机芯片必须具备以下 2 个条件。

首先，单片机芯片必须有一个通用寄存器和相应的指令能够模仿堆栈指针 SP 的功能，即能实现软堆栈。在 MSP430 系列单片机中有以下指令可以仿真 SP 的功能(把 R4 作为中断堆栈指针使用)：

MOV @R4+，SP ;将 R4 所指向地址中的内容存入 SP;中，同时 R4 中内容加 2

MOV SP，0(R4) ;将 SP 中的内容存入 R4 所指向的地址中

MOV @R4+，PC ;将 R4 所指向地址中的内容存入 PC;中，同时 R4 中内容加 2

其次，作为模拟中断堆栈指针的寄存器 R4，必须在中断之外的任何情况下不被使用。因为，此时的 R4 必须时刻保持在中断堆栈的顶部，如果改变它的值，就会改变中断堆栈的结构。一般这个要求是由所使用的编译器来保证的，在我们所使用的 IAR 编译器中，有一个选项可以避免使用 R4 和/或 R5。

具体设计时，我们在 uC/OS-II 每个任务的 TCB(任务控制块)结构中加入以下几项：

TSP--任务堆栈指针。发生中断后，指向该任务的任务堆栈的顶部。

ISP--中断堆栈指针。如果在中断中发生任务切换，指向该任务在中断堆栈所保存状态的顶部。

FromInt 标志--是否来自中断标志。用来判断该任务的状态是保存在中断堆栈中(为 1)，还是保存在任务堆栈中(为 0)。

下面假设一个普通任务 1 在执行过程发生中断，对它在中断执行过程中可能出现的几种情况进行分析。

(1)在普通任务 1 运行时引发中断，在中断中没有激活更高优先级的任务，而是正常结束中断，继续执行任务 1，如图 2 所示。

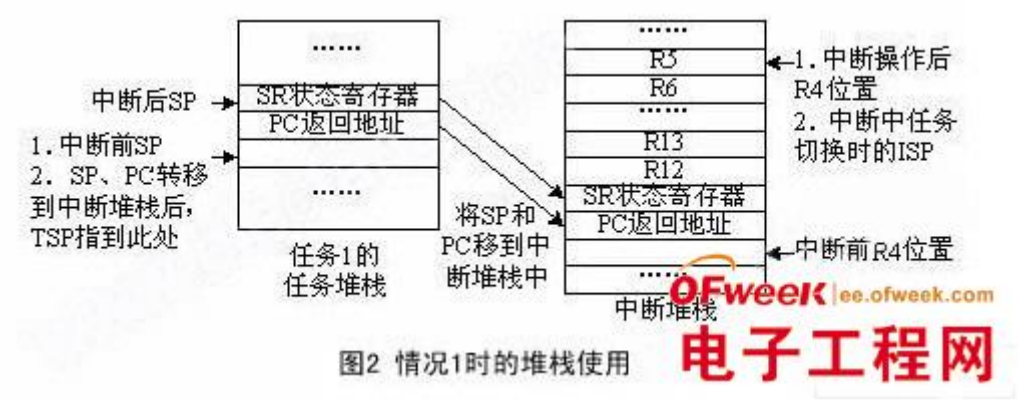

图2 情况1时的堆栈使用

开始中断：将在中断发生时保存在当前任务堆栈的 SR 和 PC 移到中断堆栈中保存，同时 SP 回到中断前的位置并将它保存到该任务 TCB 中的 TSP 中(这是为了在中断结束后，保持任务堆栈的连续性)，然后将 SP 指到目前中断堆栈的顶部，按照自定义堆栈结构的顺序依次将所有寄存器都保存到中断堆栈中。保存的过程中 R4 必须与 SP 保持同步变化，同时将 FromInt 标志置 1。

退出中断：由于没有激活更高优先级的任务，所以在中断任务完成后，将按正常的顺序退出中断，即将保存在中断堆栈中的寄存器推出堆栈，将 FromInt 标志置 0，SP 重新指向该任务的任务堆栈中，最后，将 PC 指针指向中断前的返回地址，继续程序运行。

(2)在普通任务 1 运行时引发中断，在中断中激活更高优先级的任务 2。中断结束时由任务调度器调度去执行更高优先级的任务 2，没有返回普通任务 1。

当执行任务 2 时，任务调度器会将任务 2 保存在自己任务堆栈中的状态恢复并执行任务 2。执行完后，如果没有激活更高优先级的任务，那么按照优先级高低的原则，调度器将调度执行任务 1。通过判断任务 1 的 TCB 中的 FromInt 标志，可以知道任务 1 的状态是保存在任务堆栈中还是中断堆栈中，从而可以将其状态恢复，继续运行。

(3)在普通任务 1 运行时引发中断，在中断中激活更高优先级的任务 2，执行任务 2 时又发生中断。

由于 uC/OS-II 是严格按照优先级抢占式原则进行任务调度的，所以将任务状态保存在中断堆栈顶部的任务的优先级一定比状态保存在它下面的任务的优先级高。在执行时，是由中断堆栈的顶部向底部顺序执行。在这种假设中，一定先执行任务 2，然后执行任务 1，如图 3 所示。

(4)在普通任务 1 运行时引发中断，在中断中激活更高优先级的任务 2。在执行任务 2 时又发生中断，在中断过程中任务 2 由于等待信号量而被挂起。

这种情况在系统最初设计时已经被禁止，在中断中不允许使用信号量将中断挂起。

(5) 在普通任务 1 运行时引发中断，在中断中激活更高优先级的任务 2。在执行任务 2 时又发生中断，中断中激活更高优先级的任务 3。中断结束时由任务调度器调度去执行更高优先级的任务 3。

这种情况与讨论的情况 2 是一样的。

(6)高优先级任务 2 被更高优先级的任务 3 中止，在任务 3 运行完后，任务调度器将直接调度执行任务 1(按照优先级调度)。

由于各个任务的 ISP 和 TSP 在任务切换前都已经保存在该任务的 TCB 中，任务 1 的堆栈指针和 R4 可以回到该任务在其任务堆栈和中断堆栈的正确的位置。

任务 2 被中止包括两种情况。一是任务 2 被别的任务删除，此时任务 2 在中断堆栈中占用的空间会自动释放。二是任务 2 被别的任务挂起，此时应在将程序挂起的函数 TaskSuspend()中添加一段代码，将其保存在中断堆栈中的状态移到自己的任务堆栈中，同时将其 TCB 中的 FromInt 标志设为 0。这样，在任务 2 解除挂起后，会去任务堆栈中恢复其状态。

(7)中断中发生中断嵌套。

发生中断嵌套时，要按照中断嵌套的机制进行处理。首先，在中断嵌套中，不允许进行任务调度，这样，即使在中断嵌套中激发了更高优先级的任务，也必须等到最后中断退出前才进行调度执行。这一点是由 uC/OS-II 系统设计保证的。其次，保存寄存器和函数调用所占用的 RAM 字节全部在中断嵌套中。在退出中断嵌套时，不必将 TCB 中的 FromInt 标志复位。

5 程序设计流程

(1)中断程序结构和设计流程，如图 4 所示。

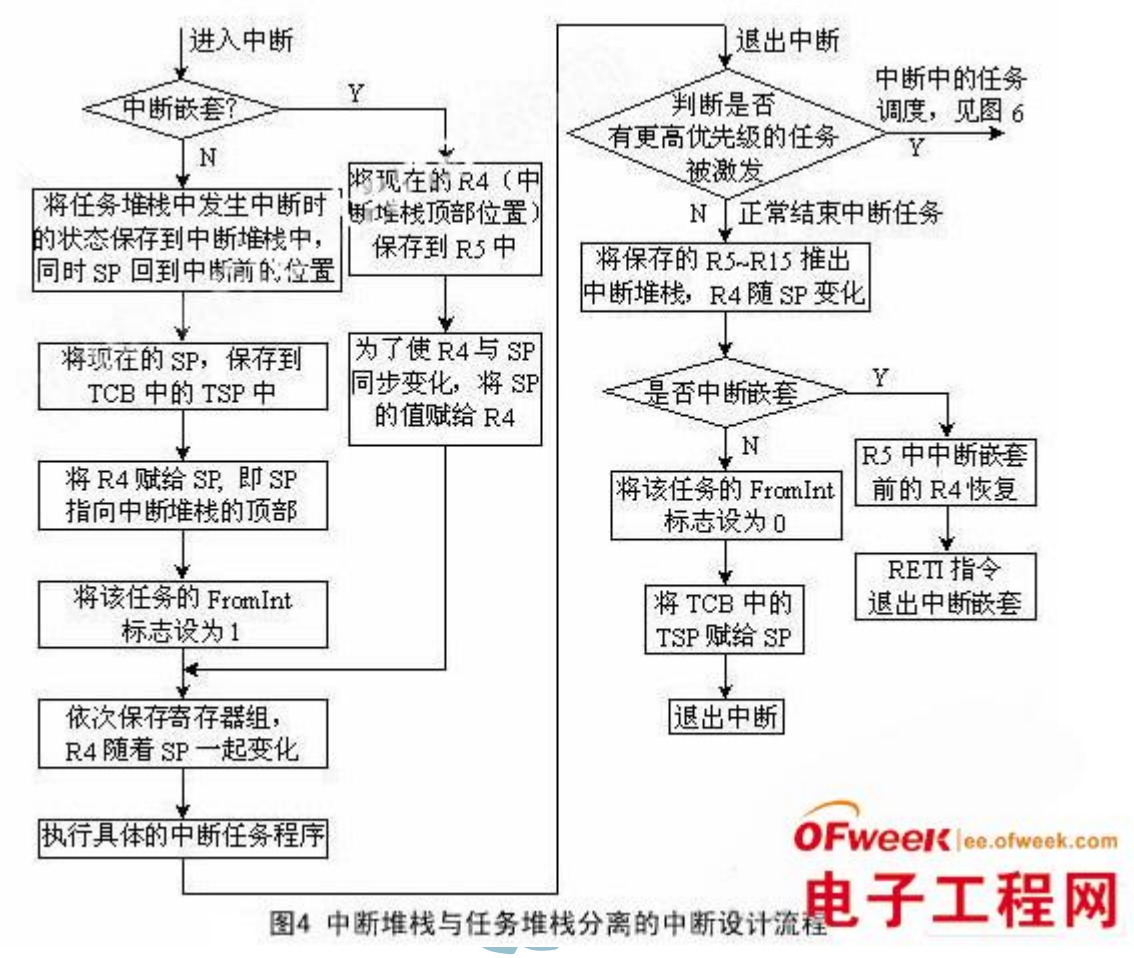

图4 中断堆栈与任务堆栈分离的中断设计流程

(2)普通任务间的任务切换程序流程，如图 5 所示。

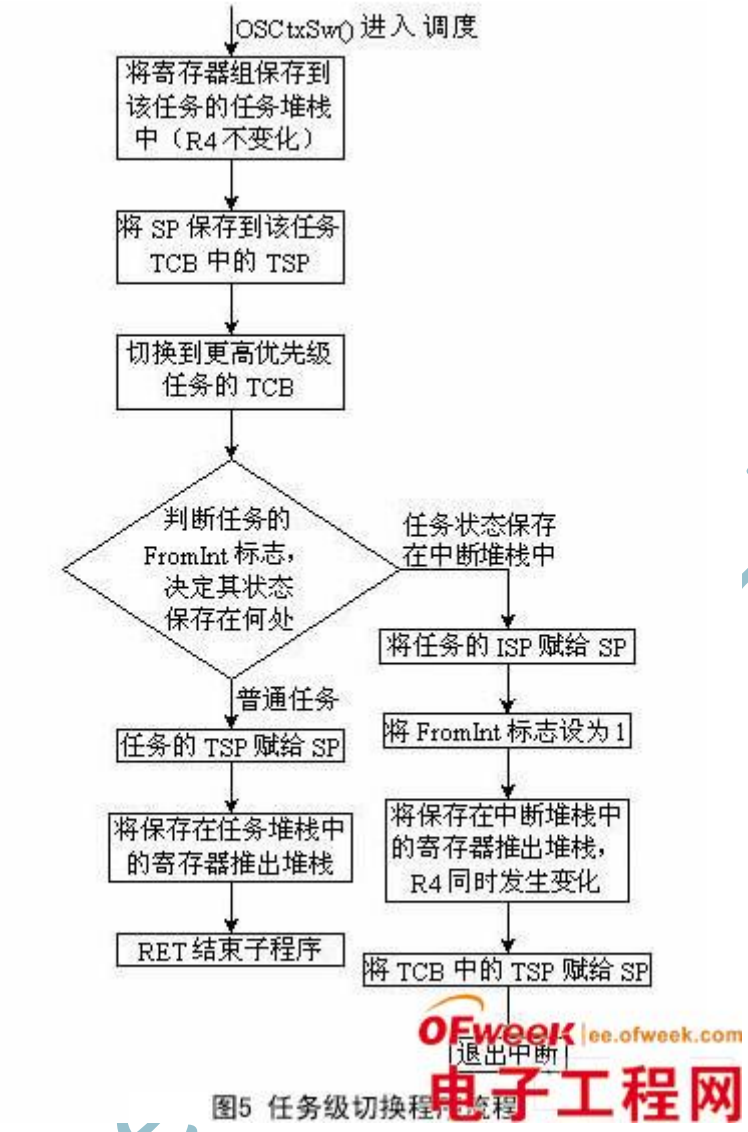

图5 任务级切换程序流程

(3)在中断中任务切换程序流程，如图 6 所示。

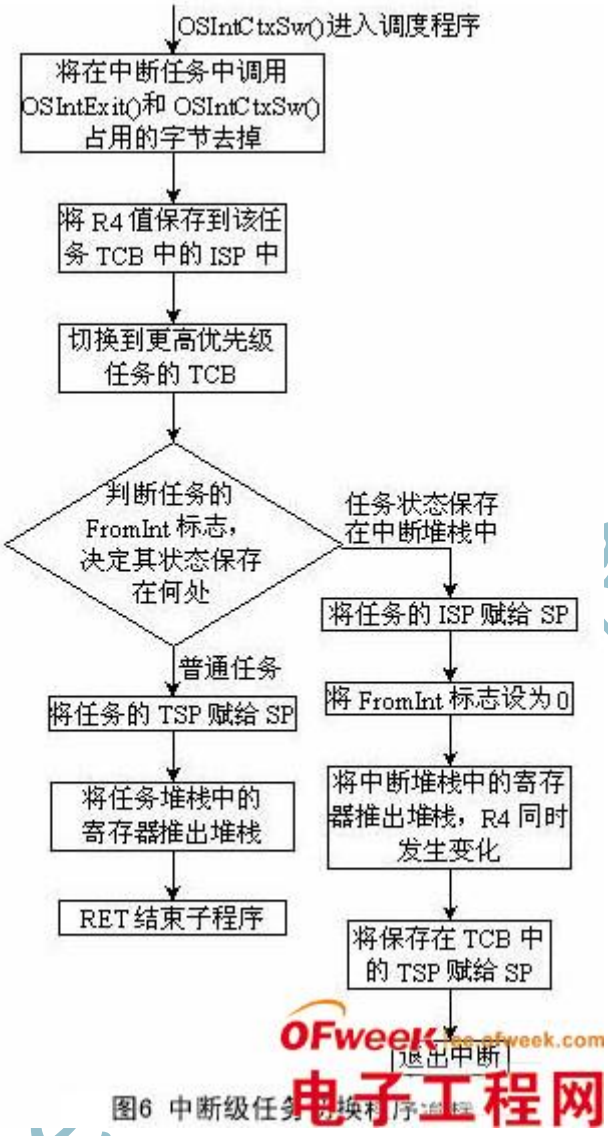

图6 中断级任务切换程序流程

我们已经在 MSP430F148 上成功运行了 uC/OS-II，在 RAM 只有 2 KB 的情况下，能够运行 16 个任务，可以满足一些复杂的应用需求，大大扩展了 MSP430F148 的应用范围，并且提高了应用系统的实时性。为了验证实际效果，在此基础上，我们将几个常用的家庭仪表--水表、暖气表、热水表集成在一起，运行效果良好，达到设计要求。