Subsystem Design

任务调度器

1 说明

该子系统示例展示了如何在 MSPM0 中实现简单的非抢先式运行至完成 (RTC) 调度器。该示例包括调度器文件以及简单任务头文件和源文件，这些文件演示了为此类调度实现构建任务的最低要求。在一个系统中，当系统需要完成多个任务，这些任务可以按任何顺序触发，并且这些任务的实际执行时间或顺序并不重要时，最适合使用 RTC 调度器。

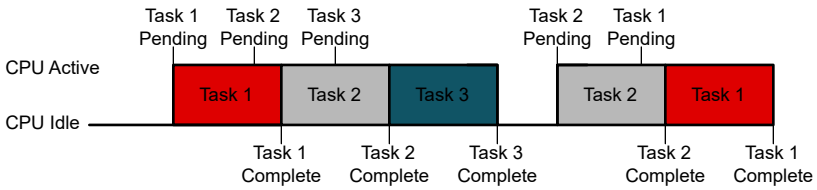

图 1-1. 运行至完成调度器

2 所需外设

任务调度器子系统是通用的，适用于 MSPM0 产品系列中的任何器件。表 2-1 列出了示例任务中使用的外设，但使用示例中的调度器部分不需要这些外设。

表 2-1. 所需外设

子块功能	外设使用	注释
DAC8（可选）	（1 个）COMP	在代码中显示为 COMP_0_INST
缓冲器（可选）	（1 个）OPA	在代码中显示为 OPA_0_INST
计时器（可选）	（1 个）TIMG	在代码中显示为 TIMER_0_INST
LED 输出（可选）	（1 个）GPIO	在代码中显示为 GPIO_LEDS_USER_LED_1
开关输入（可选）	（1 个）GPIO	在代码中显示为 GPIO_SWITCHES_USER_SWITCH_1

3 设计步骤

完成以下操作以实现简单的调度器应用：

1. 从示例子系统工程开始，或者将 scheduler 源文件和头文件添加到现有工程中。
2. 构建的 scheduler 函数用作应用程序的主软件循环。初始化后，添加对 scheduler 函数的调用，如节 6 所示。
3. 对于要在系统中执行的每个任务，创建一个函数来获取、设置和复位相应任务的挂起标志。此外，还创建在调度器尝试执行时要运行的实际函数。DAC8Driver 和 SwitchDriver 源文件和头文件提供了如何实现该操作的简单示例。
4. 添加相应的中断请求 (IRQ) 处理程序，以根据所需的硬件事件启用挂起的任务。IRQ 处理程序设置挂起任务标志，并使挂起任务计数器递增。当器件被系统中断从睡眠状态唤醒时，scheduler 会检查这些值。

4 设计注意事项

在将任务集成到任务调度器子系统中时，请考虑以下注意事项：

1. 如果多个中断或任务同时排队，则主调度器循环会按照任务在 gTasksList 中出现的顺序来处理任务。这可以被视为简单的优先级，但仍然不抢先。
2. 在该架构中，所有任务均由中断驱动，这意味着相应的 IRQ 处理程序必须设置与要运行的任务相关联的挂起标志。如果系统中只有一个事件的操作有意义，那么只有在尚未设置标志的情况下才使 gTasksPendingCounter 递增。如果需要同时对某个事件的多次发生进行排队，请对挂起标志使用整数值，而不是严格使用 true 或 false 布尔值。

5 软件流程图

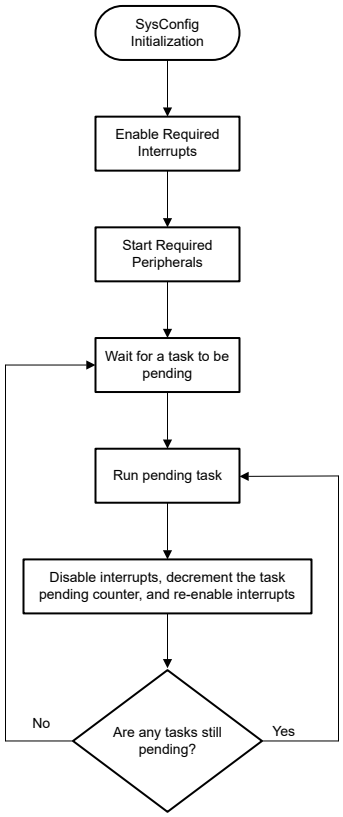

图 5-1. 应用软件流程图

6 应用代码

6.1 调度器代码

调度器代码存储在 modules/scheduler/scheduler.c 文件中，包括调度器需要在 gTasksList 中访问的所有函数指针的列表。每个任务可提供一个用于获取和复位就绪标志或挂起标志的函数，以及一个指向要运行的任务的指针。

在调度器循环中，gTasksPendingCounter 值会跟踪有多少任务处于挂起状态。当循环遍历每个挂起任务标志时，如果循环发现一个挂起的任务，则调度器循环就会使该计数器递减。清除所有任务后，器件会通过调用 *__WFI* 进入低功耗模式。

```
#include "scheduler.h"
#define NUM_OF_TASKS 2 /* Update to match required number of tasks */
volatile extern int16_t gTasksPendingCounter;

/*
  * Update gTasksList to include function pointers to the
  * potential tasks you want to run. See DAC8Driver and
  * switchDriver code and header files for examples.
  *
  */
static struct task gTasksList[NUM_OF_TASKS] =
    {
      { .getRdyFlag = getSwitchFlag,    .resetFlag = resetSwitchFlag,   .taskRun = runSwitchTask },
      { .getRdyFlag = getDACFlag,       .resetFlag = resetDACFlag,      .taskRun = runDACTask },
/*    {.getRdyFlag = , .resetFlag = , .taskRun = }, */
    };


void scheduler() {

    /* Iterate through all tasks and run them as necessary */
    while(1) {

        /*
         * Iterate through tasks list until all tasks are completed.
         * Checking gTasksPendingCounter prevents us from going to
         * sleep in the case where a task was triggered after we
         * checked its ready flag, but before we went to sleep.
         */
        while(gTasksPendingCounter > 0)
        {
            for(uint16_t i=0; i < NUM_OF_TASKS; i++)
            {
                /* Check if current task is ready */
                if(gTasksList[i].getRdyFlag())
                {
                    /* Execute current task */
                    gTasksList[i].taskRun();
                    /* Reset ready for for current task */
                    gTasksList[i].resetFlag();
                     /* Disable interrupts during read, modify, write. */
                    __disable_irq();
                    /* Decrement pending tasks counter */
                    (gTasksPendingCounter)--;
                    /* Re-enable interrupts */
                    __enable_irq();
                }
            }
        }
        /* Sleep after all pending tasks are completed */
        __WFI();
    }
}
```

6.2 主应用程序代码

用于调度器和任务运行的器件初始化在主应用程序源代码文件 task_scheduler.c 中进行处理。对 SYSCFG_DL_init 的调用会配置示例代码中所需的硬件外设，然后会启用中断，并启动 TIMER_0_INST 计数器。之后，代码进入调度器循环。

在所需的 IRQ 处理程序内，会在中断期间设置相应的标志以告知调度器任务处于挂起状态。

```
#include "ti_msp_dl_config.h"
#include "modules/scheduler/scheduler.h"

/* Counter for the number of tasks pending */
volatile int16_t gTasksPendingCounter = 0;

int main(void)
{
    SYSCFG_DL_init();

    /* Enable IRQs */
    NVIC_EnableIRQ(GPIO_SWITCHES_INT_IRQN);
    NVIC_EnableIRQ(TIMER_0_INST_INT_IRQN);

     /* Start timer to update DAC8 output */
    DL_TimerG_startCounter(TIMER_0_INST);

    /* Enter Task Scheduler */
    scheduler();
}

/* Interrupt Handler for S2 (PB21) button press, toggles LED */
void GROUP1_IRQHandler(void)
{
    switch (DL_Interrupt_getPendingGroup(DL_INTERRUPT_GROUP_1)) {
    /* S2 (PB21) has been pressed execute PB21 task */
        case GPIO_SWITCHES_INT_IIDX:
            /* Increment counter if ready flag is not already set. */
            gTasksPendingCounter += !getSwitchFlag();
            setSwitchFlag();
            break;
    }
}

/* Interrupt Handler for TIMG0 zero condition, updates DAC8 value */
void TIMER_0_INST_IRQHandler(void)
{
    switch (DL_TimerG_getPendingInterrupt(TIMER_0_INST)) {
        case DL_TIMER_IIDX_ZERO:
            /* Increment counter if ready flag is not already set. */
            gTasksPendingCounter += !getDACFlag();
            setDACFlag();
            break;
        default:
            break;
    }
}
```

7 其他资源

- 德州仪器 (TI)，下载 MSPM0 SDK
- 德州仪器 (TI)，详细了解 SysConfig
- 德州仪器 (TI)，MSPM0C LaunchPad™
- 德州仪器 (TI)，MSPM0L LaunchPad™
- 德州仪器 (TI)，MSPM0G LaunchPad™
- 德州仪器 (TI)，MSPM0 Academy

8 E2E

请访问 TI 的 E2E™ 支持论坛来查看讨论并发布新主题，以获得在设计中使用 MSPM0 器件的技术支持。

9 修订历史记录

注：以前版本的页码可能与当前版本的页码不同

Changes from Revision * (August 2024) to Revision A (August 2025)	Page
• 删除了“兼容器件”部分	2

10 商标

LaunchPad™ and E2E™ are trademarks of Texas Instruments.
所有商标均为其各自所有者的财产。

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。 严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265