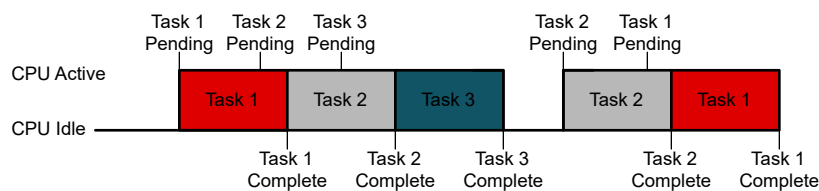


### 1 Description

This subsystem example shows how to implement a simple, non-preemptive, run-to-completion (RTC) scheduler in MSPM0. The example includes both the scheduler, and simple task header and source files, which demonstrate the minimum requirements for building tasks for this kind of scheduling implementation. In a system, use of an RTC scheduler is most appropriate when there are multiple tasks which need to be completed by the system, that can be triggered in any order, and the actual execution time or order of these tasks is not critical.



**Figure 1-1. Run-to-Completion Scheduler**

### 2 Required Peripherals

The task scheduler subsystem is generic and appropriate for any device in the MSPM0 portfolio. [Table 2-1](#) lists the peripherals used in the example tasks, but these are not required to make use of the scheduler portion of the example.

**Table 2-1. Required Peripherals**

| Subblock Functionality  | Peripheral Use | Notes  |
|-------------------------|----------------|--|
| DAC8 (Optional)         | (1 ×) COMP     | Shown as COMP_0_INST in code                 |
| Buffer (Optional)       | (1 ×) OPA      | Shown as OPA_0_INST in code                  |
| Timer (Optional)        | (1 ×) TIMG     | Shown as TIMER_0_INST in code                |
| LED Output (Optional)   | (1 ×) GPIO     | Shown as GPIO_LEDS_USER_LED_1 in code        |
| Switch Input (Optional) | (1 ×) GPIO     | Shown as GPIO_SWITCHES_USER_SWITCH_1 in code |

### 3 Compatible Devices

Based on the requirements shown in [Table 2-1](#), the example code is compatible with the devices shown in [Table 3-1](#).

**Table 3-1. Compatible Devices**

| Compatible Devices                       | EVM                           |
|--|-------------------------------|
| MSPM0Lx                                  | <a href="#">LP-MSPM0L1306</a> |
| MSPM0Gx                                  | <a href="#">LP-MSPM0G3507</a> |
| MSPM0Cx (without use of DAC8 and Buffer) | <a href="#">LP-MSPM0C1104</a> |

## 4 Design Steps

Complete the following to implement the simple scheduler application:

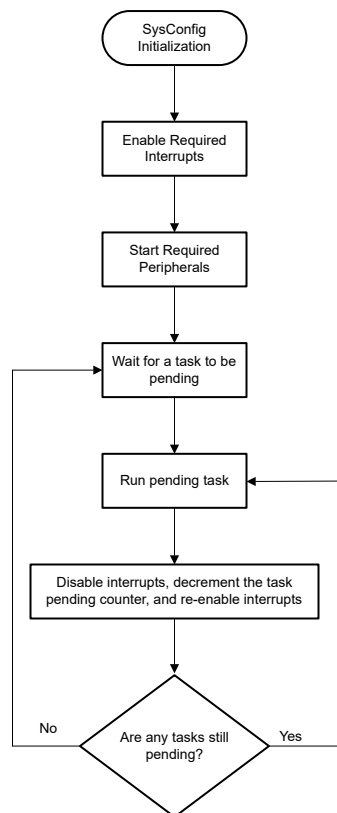
1. Either start with the example subsystem project, or add the scheduler source and header files to the existing project.
2. The scheduler function is constructed to act as the main software loop for the application. After initialization, add a call to the scheduler function as shown in [Section 7](#).
3. For each task to be performed in the system, create a function to get, set, and reset the pending flag for the appropriate task. Also create the actual function to be run when the scheduler attempts execution. The DAC8Driver and SwitchDriver source and header files provide simple examples of how this can be done.
4. Add the appropriate Interrupt Request (IRQ) handlers to enable the pending tasks based on the required hardware events. The IRQ handlers set the pending task flags, and increment the pending task counter. These values are checked by the scheduler when the device is woken from sleep by a system interrupt.

## 5 Design Considerations

When integrating tasks into the task scheduler subsystem, consider the following:

1. If multiple interrupts or tasks are queued at the same time, the main scheduler loop services the tasks in the order the tasks appear in gTasksList. This can be considered a simple priority, although still not preemptive.
2. All tasks are interrupt-driven in this architecture, meaning that the appropriate IRQ handler must set the pending flag associated with the task to be run. If only a single operation of an event makes sense in the system, only increment the gTasksPendingCounter if the flag was not already set. If multiple occurrences of an event need to be queued at the same time, use an integer value for the pending flag, rather than strictly a true or false Boolean value.

## 6 Software Flow Chart



**Figure 6-1. Application Software Flow Chart**

## 7 Application Code

### 7.1 Scheduler Code

The scheduler code is stored in the `modules/scheduler/scheduler.c` file, and includes a list of all function pointers that the scheduler needs to access in `gTasksList`. Each task can provide a function for getting and resetting the ready flag or pending flag, and a pointer to the task to be run.

Within the scheduler loop, the `gTasksPendingCounter` value keeps track of how many tasks are pending. As the loop cycles through each pending task flag, when the loop finds one that is pending, the scheduler loop decrements this counter. After all tasks are cleared, the device enters low power mode via a call to `__WFI`.

```
#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();
        }
    }
}
```

### 7.2 Main Application Code

The initialization of the device for operation of the scheduler and tasks is handled within the main application source code file, `task_scheduler.c`. The call to `SYSCFG_DL_init` configures the hardware peripherals needed in the example code, then interrupts are enabled, and the `TIMER_0_INST` counter is started. After that, the code enters the scheduler loop.

Within the required IRQ handlers, the appropriate flags are set during interrupt to tell the scheduler a task is pending.

```
#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;
    }
}
}
```

## 8 Additional Resources

- Texas Instruments, [Download the MSPM0 SDK](#)
- Texas Instruments, [Learn more about SysConfig](#)
- Texas Instruments, [MSPM0C LaunchPad™](#)
- Texas Instruments, [MSPM0L LaunchPad™](#)
- Texas Instruments, [MSPM0G LaunchPad™](#)
- Texas Instruments, [MSPM0 Academy](#)

## 9 E2E

See TI's [E2E™](#) support forums to view discussions and post new threads to get technical support for utilizing MSPM0 devices in designs.

## 10 Trademarks

LaunchPad™ and E2E™ are trademarks of Texas Instruments.  
All trademarks are the property of their respective owners.

## IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, regulatory or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to [TI's Terms of Sale](#) or other applicable terms available either on [ti.com](https://www.ti.com) or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

TI objects to and rejects any additional or different terms you may have proposed.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2024, Texas Instruments Incorporated