AN12881

使用FreeRTOS控制电机

Rev.0 - 2020年5月

应用笔记

作者： NXP半导体

URL: https://www.nxp.com/docs/en/application-note/AN12881.pdf

目录

1 介绍

本应用笔记主要描述在电机控制应用程序中使用FreeRTOS。它主要是作为AN12235的附录[1]，描述了MCSPTE1AK144开发工具包的设计[2]，但它可以作为一些通用电机控制应用的一般指南。

由于非操作系统电机控制应用程序通常是中断驱动的(如[1]中所述)，应用程序任务函数的执行通常与与驱动电机的PWM信号同步的周期性中断有关，例如, ADC中断。然后，应用程序任务直接在中断服务程序例程和while无限循环中执行。

应用程序的复杂性随着应用程序中使用的任务函数的数量而增加，这可能导致执行应用程序任务的中断次数增加和中断服务程序(ISR)运行的持续时间延长，或者在主函数中对状态变量进行过于复杂和耗时的轮询。

FreeRTOS提供了强大的工具来管理应用程序任务，而不需要使用传统的外设中断服务函数，其中就包括优先级抢占机制。

2 电机控制应用任务

一个典型的电机控制应用程序可分为以下任务：

- 外设配置
- 用户输入检测
- 应用故障检测
- MOSFET 预驱动器故障检测
- 反馈量获取
- 电机控制状态机
- LED指示
- 外部通信处理（例如，CAN、LIN、FreeMASTER）
- FreeMASTER

2.1 基于非操作系统的应用程序

在基于非操作系统的电机控制应用程序中，任务正态分布在主函数和ADC或PWM中断服务例行程之间。其分布情况如下表所示。

表1.基于非操作系统的应用程序任务函数分布

应用程序任务	执行区域
外设配置	main()

续表见下一页。

表1.基于非操作系统的应用程序任务函数分布（续）

应用程序任务	执行区域
MOSFET 预驱动器故障检测	main()无限循环
外部通信处理	
用户输入检测	ADC中断服务程序
反馈量获取	
故障检测	
电机控制状态机	
LED指示	
FreeMASTER记录仪	

下图显示了在基于100μs执行一次电流闭环的非操作系统的电机控制应用程序中main()函数执行的典型时间和ADC中断执行的典型时间之间的关系。每第n个 ADCISR的执行时间更长，因为速度闭环也会被处理。

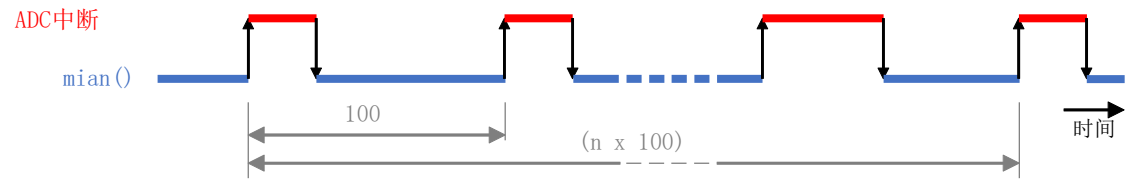

图1. 非操作系统应用程序任务定时

由于精确电机控制需要对 ADC 中断的快速反应，时序关键任务在 ISR 中执行，而不是在 main() 函数无限循环中轮询各自的外设寄存器标志。

2.2 基于FreeRTOS的应用程序

为了尽量减少在ISR中花费的时间，FreeRTOS支持中断延迟处理。下面的示例将使用应用程序来延迟中断处理，因为FreeRTOS允许控制延迟处理任务的优先级，可以最大程度减少ISR与延时处理任务之间的延迟[3].

下表总结了应用程序各个任务在FreeRTOS任务中的分布情况。

表2.基于FreeRTOS的应用程序任务函数分布

应用程序任务	FreeRTOS任务/CPU中断	FreeRTOS任务优先级[1]
外设配置	Init任务	4
MOSFET预驱动器故障检测	主任务	2
外部通信处理		
反馈量获取	ADC中断服务程序	-

表2.基于FreeRTOS的应用程序任务函数分布（续）

应用程序任务	FreeRTOS任务/CPU中断	FreeRTOS任务优先级[1]
故障检测		
电机控制状态机	状态机任务	7=(configMAX_PRIORITIES-1)
FreeMASTER记录仪		
用户输入检测	RTOS定时任务（软件定时器回调）	3=configTIMER_TASK_PRIORITY
LED指示		
1.最高FreeRTOS任务优先级由FreeRTOSConfig.h中的configMAX_PRIORITIES值定义。		

基于FreeRTOS的应用程序任务的时间如下图所示。

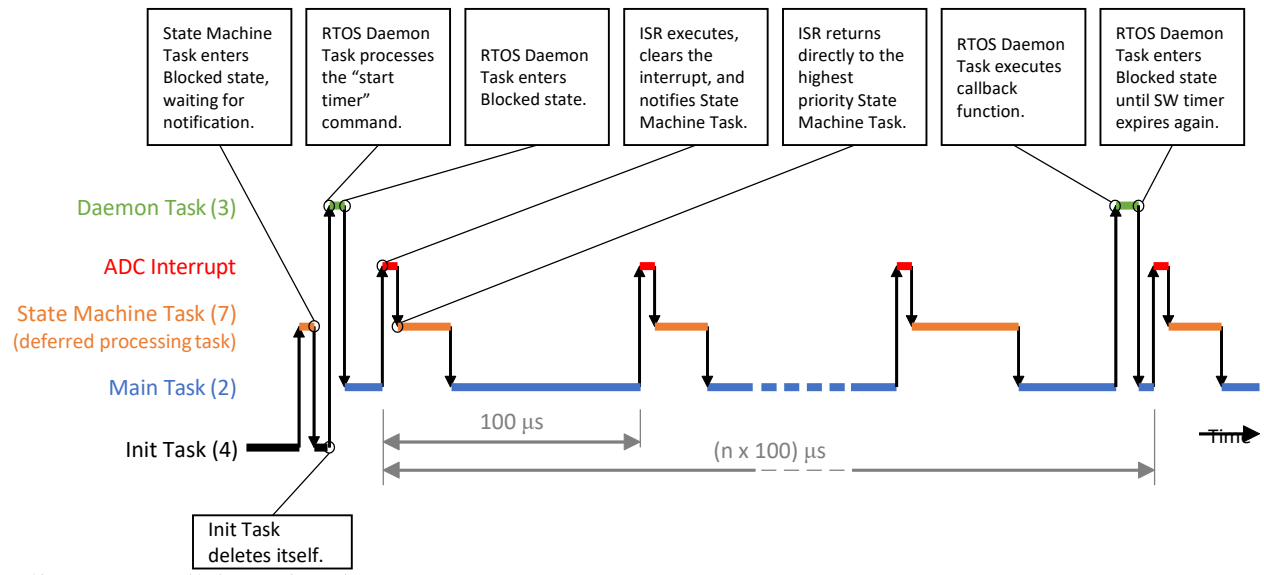

图2. 基于Free-RTOS的应用程序计时

2.2.1 创建FreeRTOS任务

xTaskCreat()API函数可以创建FreeRTOS任务。该函数的原型如下：

```
BaseType_t xTaskCreate
(
    TaskFunction_t pxTaskCode,
    const char *const pcName,
    const configSTACK_DEPTH_TYPE usStackDepth,
    void *const pvParameters,
    UBaseType_t uxPriority,
    TaskHandle_t *const pxCreatedTask
)
```

下表列出了xTaskCreate()函数的输入参数。

表3.xTaskCreate()输入参数

xTaskCreate()输入参数	描述
pxTaskCode	任务执行函数
pcName	任务名称。包括NULL终止符在内的任务名称的长度受到FreeRTOSConfig.h中定义的configMAX_TASK_NAME_LEN值的限制
usStackDepth	任务堆栈的大小为堆栈可以保存的单词数（例如，Cortex-M4上的words(4个字节)数）。
pvParameters	任务函数输入参数地址（void*）。分配给该参数的值将被传递到实现该任务的函数中。
uxPriority	任务优先级，在 0（最低优先级）到 (configMAX_PRIORITIES - 1)（最高优先级）的范围内。
pxCreatedTask	任务句柄的指针。

2.2.2 输入任务

Init任务（初始化任务）是在FreeRTOS调度程序启动之前在main()函数中创建的。

- 示例1：初始化任务创建

```
/* Create Init Task */
ret = xTaskCreate(pmsm_init,"pmsm_init_tsk",256u,NULL,4u,NULL);

/* Start FreeRTOS scheduler */
vTaskStartScheduler();
```

初始化任务用于执行外设配置、创建主任务和状态机任务，并初始化软件定时器。一旦初始任务创建了主任务，主任务将进入“就绪”状态，但由于与运行状态下的初始任务相比优先级更低，因此主任务仍然处于“就绪”状态。

- 示例2：主任务创建

```
/* Main Task function prototype */
void pmsm_main(void* pvParameters);

BaseType_t ret;

/* Create Main Task for fault checking and external communication processing */
ret = xTaskCreate(pmsm_main, "pmsm_main_tsk", 128u, NULL, 2u, NULL);
```

一旦初始化任务创建了状态机任务，状态机任务将立即优先于初始化任务，因为状态机任务具有更高的优先级。状态机任务开始获取信号量，在调用ulTaskNotifyTake()函数后，它进入阻塞状态，将控制释放回初始任务。状态机任务将在状态机任务节中进一步描述.

- 示例3：状态机任务的创建

```
/* State Machine Task function prototype */
void pmsm_state(void* pvParameters);

TaskHandle_t  PmsmStateHandle;
```

```
BaseType_t ret;

/* Create State Machine Task (ADC ISR will defer to) */
ret = xTaskCreate(pmsm_state, "pmsm_state_tsk", 128u, NULL, (configMAX_PRIORITIES - 1),
      &PmsmStateHandle);
```

软件定时器由xTimerCreate()创建，并由xTimerStart()函数启动，向定时器命令队列发送“开始定时器”命令，从而让定时器任务从“已阻塞”状态过渡到“准备就绪”状态。守护进程任务将在RTOS守护进程任务节中进一步描述.

- 示例4：FreeRTOS软件定时器初始化示例

```
/* Software timer callback function prototype */
void PeriodicalTimerCbk(TimerHandle_t xTimer);

TimerHandle_t PeriodicalTimer;
BaseType_t ret;

/* Create 1ms periodical software timer with PeriodicalTimerCbk() callback function */
PeriodicalTimer = xTimerCreate("PeriodicalTimer", pdMS_TO_TICKS(1), pdTRUE, (void *)0,
                  PeriodicalTimerCbk);

/* Start software timer with no block time specified */
ret = xTimerStart(PeriodicalTimer, 0);
```

一旦创建了所有必要的任务和软件定时器，初始化任务将通过调用vTaskDelete(NULL)API函数来删除自己。

2.2.3 RTOS守护进程任务

RTOS 守护进程任务是一个标准任务，当任务调度器通过 vTaskStartScheduler() 函数调用启动时，由 FreeRTOS 自动创建。创建守护进程任务时，优先级为 configTIMER_TASK_PRIORITY，栈大小为 FreeRTOSConfig.h 中定义的 configTIMER_TASK_STACK_DEPTH。这些参数对于所有创建的软件定时器都是通用的。 守护进程任务用于为使用定时器命令队列从调用任务发送到守护进程任务的软件定时器命令提供服务。在这个应用中，软件定时器周期性地执行一个回调函数，负责用户输入检测和 LED 控制。

如图 2 所示，软件定时器由初始化任务创建和启用。由于守护进程任务的优先级低于初始化任务，一旦 初始化任务通过 vTaskDelete(NULL) 函数调用删除自身，它就会处理定时器命令队列中的“start timer”命令，允许守护进程任务从 就绪过渡到运行状态。处理完“启动定时器”命令后，守护进程任务进入阻塞状态。

一旦软件定时器到期，守护程序任务再次进入就绪状态。如果当前处于Ready状态的任务是最高优先级的任务，并且没有更高优先级的任务处于运行状态，则进入运行状态并执行PeriodicTimerCbk()回调函数。否则，它必须等待更高优先级的任务进入阻塞状态。

注

软件定时器是一种合适的 FreeRTOS 函数去使用最少的系统资源来触发周期性事件。 但是，在使用多个软件定时器时，所有配置的定时器都依赖于具有自身优先级的 RTOS 守护进程任务。 如果应用程序需要单独的优先级，建议使用单独的 OS 任务，尽管最终资源成本更高。

2.2.4 ADC中断

状态变量采集在ADC ISR中执行，电机控制应用中的故障检测也必须在ISR中执行，以最小化任何潜在的故障反应时间。然后，ADC ISR直接发送通知给最高优先级的状态机任务，并将控制权释放给FreeRTOS调度器。

为了在状态机任务中直接紧跟着 ISR，应用程序中使用了任务通知函数。 vTaskNotifyGiveFromISR() API 函数直接向任务发送通知，增加其通知值，从而将其通知状态设置为挂起（如果它尚未处于挂起状态）。 有关任务通知代码示例，请参见示例 4。

一旦 vTaskNotifyGiveFromISR() 退出，它将 xHigherPriorityTaskWoken 设置为 pdTRUE，让后续的 portYIELD_FROM_ISR() 调用知道执行上下文切换。

- 示例5：状态机任务通知

```
BaseType_t xHigherPriorityTaskWoken;

/* Defer to State Machine Task: */
/* Initialize xHigherPriorityTaskWoken variable */
xHigherPriorityTaskWoken = pdFALSE;
/* Send notification to State Machine Task, update xHigherPriorityTaskWoken variable value */
vTaskNotifyGiveFromISR(PmsmStateHandle, &xHigherPriorityTaskWoken);
/* Perform context switch based on xHigherPriorityTaskWoken variable value, to ensure that the
interrupt returns directly to the highest priority task */
portYIELD_FROM_ISR(xHigherPriorityTaskWoken);
```

要成功通知状态机任务，当使用xTaskCreate()API函数创建状态机任务时，必须带入任务句柄参数(请参见示例3中的PmsmStateHandle变量)。

2.2.5 状态机任务

状态机任务被初始化任务创建并进入运行状态后，在ulTaskNotifyTake()函数调用期间进入阻塞状态，等待通知，因为任务通知值为零。 一旦 ADC ISR 向状态机任务发送通知，任务通知值就会递增，导致状态机任务进入就绪状态。 ISR 之后直接执行状态机任务，该任务是当前处于就绪状态的最高优先级任务。

由于任务通知值现在不等于零，ulTaskNotifyTake() 将通知值归零并退出。 然后执行包括 FreeMASTER 记录器在内的状态机函数。

在任务中处理完状态机和FreeMASTER记录器后，状态机任务在后续的ulTaskNotifyTake()函数调用中切换回阻塞状态，等待接收下一个通知，因为通知值再次为零。

- 示例6：状态机任务函数

```
void pmsm_state(void* pvParameters)
{
     while(1)
     {
         /* Wait for notification from ADC ISR in blocked state indefinitely, zero task
         notification value on exit */
         ulTaskNotifyTake(pdTRUE, portMAX_DELAY);
         /* State machine function */
         StateMachine();
     }
}
```

ulTaskNotifyTake() 函数的第一个参数指定调用任务通知值应通过实现计数信号量递减 (pdFALSE)，或通过实现二进制信号量清零 (pdTRUE) [3]。

ulTaskNotifyTake() 函数的第二个参数指定调用任务应等待接收通知的时间。 示例 6 中的 portMAX_DELAY 参数值将其设置为无限期等待。

2.2.6 主任务

最低优先级的主任务在后台运行，通过中断引脚电平检测持续轮询 MOSFET 预驱动器故障。 这里不断调用外部通信处理函数，以保证对接收到的数据做出最快的响应。

在不需要连续的外部通信处理的情况下，可以不连续地轮询预驱动器故障，而是定时轮询，为CPU运行潜在的低优先级任务留出空间。这可以通过让主任务阻塞达到固定的 RTOS 滴答计数来实现，例如使用 vTaskDelayUntil() FreeRTOS 函数，或使用软件定时器（有关详细信息，请参阅 [3]）。

3 结论

为了尽量减少在ADC或PWM中断ISR中花费的时间，FreeRTOS提供了延迟中断处理。使用任务通知将ISR部分函数转移到操作系统任务中去执行，从而在ISR中花费最少的时间，可以减少应用程序中可能需要的其他中断的延迟。

在本应用笔记中描述的电机控制应用中使用的FreeRTOS软件函数仅为示例。在大多数情况下，FreeRTOS提供了实现所需函数的多种选项。

4 参考文献

1. 3-phase Sensorless PMSM Motor Control Kit with S32K144, AN12235, Rev. 1, 05/2020
2. MCSPTE1AK144: S32K144 Development Kit for BLDC and PMSM motor control
3. Mastering the FreeRTOS™ Real Time Kernel, 161204,

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/ SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer's applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: May 2020

Document identifier: AN12881

arm