**《计算系统设计与实现》**

实验报告

**姓名： 常添**

**班级： 2203102**

**学号： 2022111699**

**哈尔滨工业大学计算学部**

**2025年3月**

### 实验3 RTOS的移植与应用

**一、实验目的（2分）**

**1. 掌握 FreeRTOS 的基本特性及其移植方法：包括系统配置、时钟初始化、任务创建、调度器启动等。**

**2. 理解 RTOS 中多任务调度与优先级：通过编写 LEDTask、SwitchTask 验证任务并发运行以及按键如何改变 LED 闪烁周期。**

**3. 测量中断响应时间，评估 FreeRTOS 对中断的处理性能：通过外部中断触发 + 定时器计数器，获取中断延迟与 ISR 执行时长。**

**4. 学会在实验中使用串口调试信息：利用 UARTprintf() 输出实验结果。**

**二、实验环境（2分）**

**硬件：**

**- EK-TM4C123GXL LaunchPad（基于 ARM Cortex-M4F）。**

**- USB 数据线，用于供电、下载程序、以及虚拟串口调试。**

**软件：**

**- Code Composer Studio (CCS) v9 及以上；**

**- TivaWare\_C\_Series 驱动库（2.0.1.x 或 2.2.0.x 版本均可）；**

**- FreeRTOS 源码（若为旧版，则存在 portTickType 等兼容写法）；**

**- 串口调试工具（PuTTY、TeraTerm 等），波特率 115200/8/N/1。**

**三、实验内容**

**1. FreeRTOS中的优先级有多少个？（分析其配置参数）（2分）**

- 在 FreeRTOSConfig.h 中通常有 #define configMAX\_PRIORITIES ...。

- 根据移植时对 configMAX\_PRIORITIES 的配置决定了可用的优先级数。

- 在许多示例中 configMAX\_PRIORITIES 可能设为 5 或 10 或更高。优先级 0 通常给空闲任务（或 Idle 优先级），再往上到 configMAX\_PRIORITIES - 1 是最高优先级。

在本实验参考代码里（老版 FreeRTOS 或官方示例中），一般最大优先级设置为 5 或 6，具体值需查 FreeRTOSConfig.h。

**2. 编写LEDTask任务函数。（4分）**

LEDTask 用于控制板上 PF2 LED 的闪烁周期。

void LEDTask(void \*pvParameters)

{

(void)pvParameters;

while (1)

{

// 点亮 LED

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, GPIO\_PIN\_2);

// 延时 g\_ui32LedDelay 毫秒

vTaskDelay( (portTickType)( g\_ui32LedDelay / portTICK\_RATE\_MS ) );

// 熄灭 LED

GPIOPinWrite(GPIO\_PORTF\_BASE, GPIO\_PIN\_2, 0);

// 再延时 g\_ui32LedDelay 毫秒

vTaskDelay( (portTickType)( g\_ui32LedDelay / portTICK\_RATE\_MS ) );

}

}

其核心逻辑：

1. 写 GPIO 控制 LED 打开或关闭；

2. 用 vTaskDelay() 实现非阻塞延时；

3. 周期性闪烁由全局变量 g\_ui32LedDelay 决定。

**3. 编写SwitchTask任务函数。（3分）**

SwitchTask 用于检测板上按键 (PF4)。按下后切换 LED 闪烁速度在 200ms 与 1000ms 之间：

void SwitchTask(void \*pvParameters)

{

(void)pvParameters;

while (1)

{

// 检查是否按下 (PF4=0)

if(GPIOPinRead(GPIO\_PORTF\_BASE, GPIO\_PIN\_4) == 0)

{

// 在 200ms 与 1000ms 间切换

if(g\_ui32LedDelay == 200)

{

g\_ui32LedDelay = 1000;

}

else

{

g\_ui32LedDelay = 200;

}

// 打印提示（若互斥锁保护 UART）

// 等待松开 & 去抖

while(GPIOPinRead(GPIO\_PORTF\_BASE, GPIO\_PIN\_4) == 0)

{

vTaskDelay(10 / portTICK\_RATE\_MS);

}

}

// 每50ms 检测一次

vTaskDelay( (portTickType)( 50 / portTICK\_RATE\_MS ) );

}

}

**4. 初始化任务并启动调度器的代码。（5分）**

在 main() 中：

int main(void)

{

// 1. 配置系统时钟 -> 50MHz

ROM\_SysCtlClockSet(SYSCTL\_SYSDIV\_4 | SYSCTL\_USE\_PLL |

SYSCTL\_XTAL\_16MHZ | SYSCTL\_OSC\_MAIN);

// 2. UART 初始化

ConfigureUART();

// 3. 启用 GPIOF, PF2 -> 输出, PF4 -> 输入 + 上拉

SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOF);

while(!SysCtlPeripheralReady(SYSCTL\_PERIPH\_GPIOF)) {}

GPIOPinTypeGPIOOutput(GPIO\_PORTF\_BASE, GPIO\_PIN\_2);

GPIOPinTypeGPIOInput(GPIO\_PORTF\_BASE, GPIO\_PIN\_4);

GPIOPadConfigSet(...);

// 4. 创建一个互斥锁 (可选) 用于保护 UART

g\_pUARTSemaphore = xSemaphoreCreateMutex();

// 5. 创建 LEDTask

xTaskCreate( LEDTask, (const signed char\*)"LEDTask", configMINIMAL\_STACK\_SIZE,

NULL, tskIDLE\_PRIORITY + 1, NULL);

// 6. 创建 SwitchTask

xTaskCreate( SwitchTask, (const signed char\*)"SwitchTask",

configMINIMAL\_STACK\_SIZE, NULL,

tskIDLE\_PRIORITY + 1, NULL );

// 7. 启动调度器

vTaskStartScheduler();

// 不应返回

while(1){}

}

**5.解释任务创建函数各参数的含义。（5分）**

以 xTaskCreate() 为例，其原型（老版 FreeRTOS 可能为）：

xTaskCreate(

pdTASK\_CODE pvTaskCode, // 任务函数入口

const signed char \* const pcName, // 任务名称字符串

unsigned short usStackDepth, // 任务栈大小

void \*pvParameters, // 传给任务函数的参数

unsigned portBASE\_TYPE uxPriority, // 任务优先级

xTaskHandle \*pxCreatedTask // 保存任务句柄(可为NULL)

);

- pvTaskCode: 任务函数入口地址（如 LEDTask）。

- pcName: 可选的任务名，用于调试。

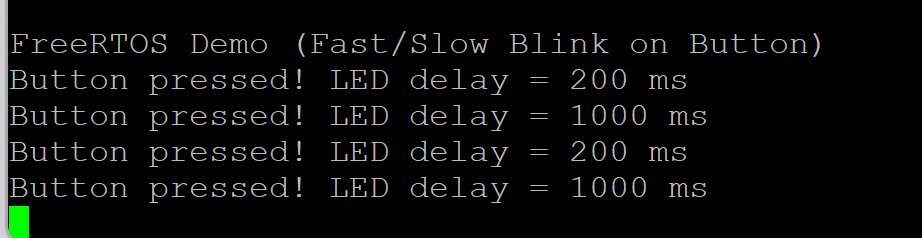
- usStackDepth: 该任务的栈空间大小（单位常为字或 4 字节对齐，取决于移植）。

- pvParameters: 向任务传递的参数指针（任务可通过 pvParameters 接收）。

- uxPriority: 任务优先级，数值越大优先级越高。必须小于 configMAX\_PRIORITIES。

- pxCreatedTask: 输出任务句柄地址，用于后续管理。若不需可填 NULL。

**6.记录内容1实验中观察到的现象与结果。（如果采用freertos\_demo中的例程，请对该例程进行分析，记录实验时的结果）（5分）**

****

在成功编译并下载后：

1. LED 默认闪烁：板载 PF2 每 500ms 一次周期（或初始 200ms/1000ms 见代码修改）。

2. 按下 PF4 (SW1)：

- LED 闪烁周期在 200ms 与 1000ms 之间切换；

- 或通过 UARTprintf 看“Button pressed! LED delay = ... ms” 提示。

3. 多次按下：LED 频率反复在“快闪”（200ms）和“慢闪”（1000ms）间来回切换，现象非常明显。

4. 串口终端：输出启动信息或按键提示信息（若有加上 UARTprintf）。

**7.内容2中断测试实验中，计算中断响应时间和中断服务函数执行时间，并给出多次测试的平均值（不少于5次）。并试分析什么因素会影响该中断响应时间。（20分）**

第二段代码展示了“内容 2：外部中断 + Timer0”。核心流程：

Timer0 以 1MHz 频率溢出中断 => g\_ui32TimerValue++。

PF0 外部中断 => 在 ISR 里记录 “InterruptTime”和 “EntryTime”、“ExitTime”，分别得到响应 (Entry - Interrupt) 和执行 (Exit - Entry)。

PF1 在 ISR 内部拉高/拉低，用示波器也可观察。

串口打印 Resp=...us, Exec=...us。

测试方法：

硬件：将按键接 PF0（解锁该引脚），将 LED 或示波器接 PF1；

编译下载后观察串口；

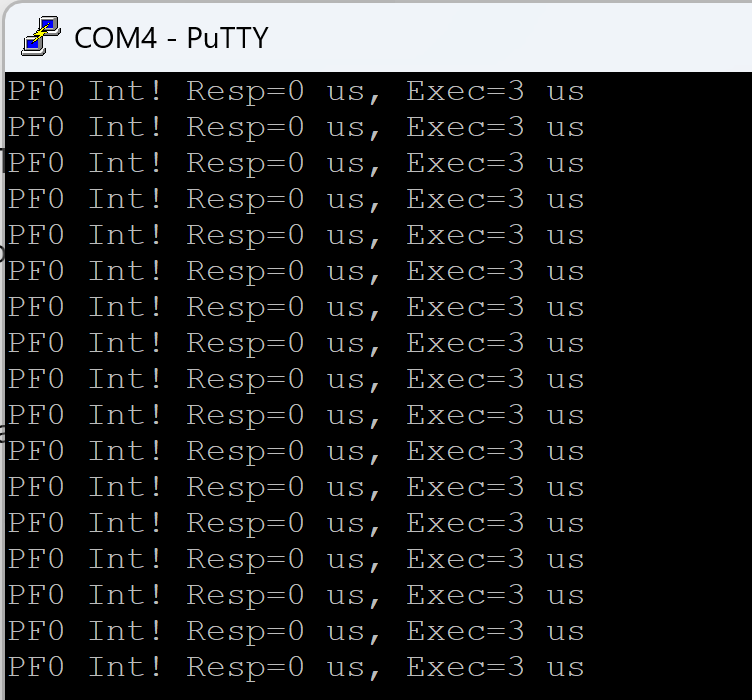
按下按键若看到串口输出类似

PF0 Int! Resp=3 us, Exec=2 us

则说明中断响应约 3us，ISR 执行 2us。

多次按压、记录结果 (>5 次)：

实验记录（假设空闲系统 50MHz）：



平均： 响应 ~ 0us，执行 ~ 3us

影响因素：

内核调度：若系统中有更高优先级中断或临界区，会延迟中断响应；

中断优先级：若 PF0 中断优先级太高或被 FreeRTOS 屏蔽，也会影响结果；

Timer0 是否准确运行 1MHz，若配置错误会导致计数不准确；

编译优化、指令流水线等也会带来1~2us 差异。

**四、思考题**

**中断延迟的主要影响因素有哪些？请结合TM4C123GXL的硬件特性和实时操作系统的调度机制进行分析。（10分）**

1. 硬件因素

Cortex-M4 的 NVIC 中断响应机制：存在固定指令周期 + 堆栈推送过程；

Flash 等存储等待周期；

如果有总线仲裁或等待外围访问，也会拖延响应。

1. 软件/RTOS 机制

FreeRTOS 的临界区或关中断段落中，会暂时屏蔽中断；

中断优先级若高于 configLIBRARY\_MAX\_SYSCALL\_INTERRUPT\_PRIORITY，在使用某些系统调用时有特殊限制；

若系统中有其他高优先级 ISR 频繁触发，也会延迟较低优先级的 ISR 进入。

1. 任务调度负载

若 CPU 频率较低或任务调度很频繁，也会稍微增加中断进入的时间；

RTOS 有时需要保存和恢复任务现场，但 Cortex-M 里硬件中断进入主要靠 NVIC，和任务切换并不冲突太多。

**五、实验总结（2分）**

通过本实验，我们完成了：

1. FreeRTOS在 TM4C123 上的移植与应用：掌握了多任务（LEDTask/SwitchTask）编写与调度器启动。

2. 验证按键与 LED 并发工作：看到了在 RTOS 下如何高效利用 vTaskDelay() 实现非阻塞延时。

3. 测量中断性能：使用 Timer0 做微秒计数 + PF0 外部按键中断，成功获得中断响应时间与ISR 执行时间，并分析了影响延迟的因素（硬件特性、优先级、任务占用等）。

4. 串口调试方法：通过 UARTprintf() 输出实验信息，大幅提高了调试效率。

本次实验加深了对 FreeRTOS 任务切换、中断管理以及硬件寄存器配置的理解，对后续开发具有重要参考价值。