实验10：  
高级并发 – 使用RTOS功能

北卡罗莱纳州立大学  
电子与计算机工程系

Alexander G. Dean博士  
2015年8月24日，版本1.0



**

*内容*

[概述 5](#_Toc462244405)

[设置 5](#_Toc462244406)

[RTOS和项目安装 5](#_Toc462244407)

[打开和配置项目 5](#_Toc462244408)

[下载代码 6](#_Toc462244409)

[评估起始程序 6](#_Toc462244410)

[使用互斥信号量实现OLED共用 6](#_Toc462244411)

[互斥量准备工作 6](#_Toc462244412)

[获取互斥量 7](#_Toc462244413)

[提供互斥量 7](#_Toc462244414)

[测试 7](#_Toc462244415)

[添加时钟任务 7](#_Toc462244416)

[任务准备工作 7](#_Toc462244417)

[测试 8](#_Toc462244418)

[更正任务周期 8](#_Toc462244419)

[使用二进制信号量实现任务触发 8](#_Toc462244420)

[信号量准备工作 8](#_Toc462244421)

[等待信号量 9](#_Toc462244422)

[发出信号量信号 9](#_Toc462244423)

[测试 9](#_Toc462244424)

[使用队列发送和接收串行数据 9](#_Toc462244425)

[UART准备工作 10](#_Toc462244426)

[队列准备工作 10](#_Toc462244427)

[任务级发送器代码 10](#_Toc462244428)

[发送器ISR代码 11](#_Toc462244429)

[任务级接收器代码 11](#_Toc462244430)

[接收器ISR代码 12](#_Toc462244431)

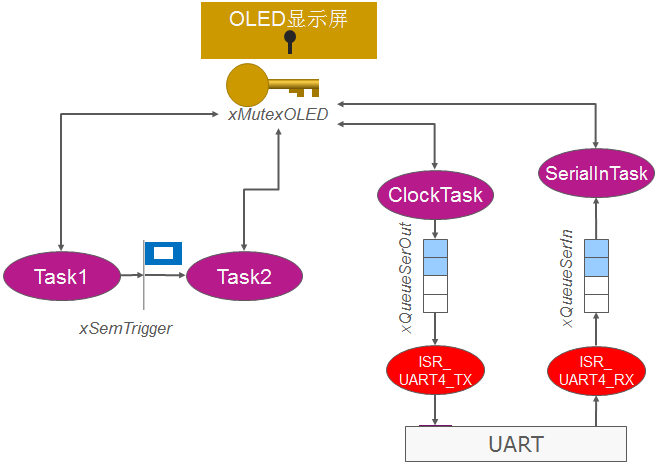
[测试 12](#_Toc462244432)

[完成 13](#_Toc462244433)

# 概述

在本实验练习中，您将通过添加使用RTOS功能进行任务控制和通信的代码来完成程序。本实验使用基本I/O屏蔽并在OLED显示屏上显示相关信息。

下图给出了您将在本实验中构建的系统软件架构。向程序中添加代码时，您将发现，回头查阅该图十分有用。



# 设置

## RTOS和项目安装

1. 如果尚未安装，请下载FreeRTOS并将其安装到您的PC中。可以通过此下载链接进行下载：[http://www.freertos.org](http://www.freertos.org/)。
2. 将归档Module10\_Adv\_Concurrency\_Code.zip解压到文件夹FreeRTOS/Demo/PIC32MZ\_MPLAB下的全新FreeRTOS安装中。这应创建三个目录。本实验使用目录Lab10\_Adv\_Concurrency中的项目。

## 打开和配置项目

1. 使用6引脚插座或电缆将PICkit 3调试器插入chipKIT板的JP1（ICSP）中。确保PICkit 3调试器上的白色三角指向JP1的引脚1（方形焊盘）。
2. 在chipKIT板上安装基本I/O屏蔽。
3. 使用USB电缆将PICkit 3调试器和chipKIT Wi-Fire板连接到PC。
4. 在PC上启动MPLAB X IDE程序。
5. 在MPLAB X IDE的**“File”（文件）**菜单中选择**“Open Project…”（打开项目…）**，导航至FreeRTOS/Demo/PIC32MZ\_MPLAB，然后选择名为**Lab10\_Adv\_Concurrency**的项目。
6. 单击“扫帚和锤子”图标（位于MPLAB X IDE窗口的顶部）以清除并编译主项目。   
   
7. MPLAB X IDE右下角窗格中的输出选项卡将显示这些操作的进度和结果。请注意消息“BUILD SUCCESSFUL”（编译成功），它指示的是总体状态。

## 下载代码

1. 现在，您可以将程序下载到chipKIT中。单击绿色向下箭头图标（位于MPLAB X IDE窗口的顶部）完成此操作。   
    
2. PICkit3上的状态指示灯将在编程期间呈红色和绿色闪烁状态，然后在完成时保持绿色常亮状态。“Output”（输出）窗口将打开名为**“PICkit 3 debugger”（PICkit 3调试器）**的选项卡以显示编程状态。

# 评估起始程序

起始程序有两个定期运行的任务（Task1和Task2）。每个任务都会更新OLED以指示其已运行的次数。

1. 在程序运行时检查OLED。请注意，OLED上显示的信息会闪烁、不稳定并且有时会处于错误的位置。这是因为当前正在访问OLED的任务被另一个随后也会访问此OLED的任务抢占。

# 使用互斥信号量实现OLED共用

我们将通过互斥信号量来控制对OLED的访问，进而更正OLED操作。任务必须先获得互斥量xMutexOLED，然后才允许访问OLED，并且在完成后，必须释放互斥量。

## 互斥量准备工作

1. 在OLED.h中的“TODO: Add RTOS header file includes”注释位置添加以下头文件包含语句。

#include "FreeRTOS.h"

#include "semphr.h"

1. 在OLED.h中的“TODO: Declare mutex handle here”注释位置声明互斥量句柄。

SemaphoreHandle\_t xMutexOLED;

1. 在OLED.c中的“TODO: Define mutex handle here”注释位置定义互斥量句柄。

SemaphoreHandle\_t xMutexOLED;

1. 在CreateTaskSyncStructures函数（在main.c中）中的“TODO: Create mutex here”注释位置添加以下互斥量初始化代码。

xMutexOLED = xSemaphoreCreateMutex();

if( xMutexOLED == NULL )

{ // Error - the semaphore was not created successfully.

while (1)

;

}

## 获取互斥量

1. 在Task1和Task2函数（在user.c中）对应的两个“TODO: Take mutex here”注释位置添加以下互斥量获取代码。

xSemaphoreTake(xMutexOLED, portMAX\_DELAY);

## 提供互斥量

1. 在Task1和Task2函数（在user.c中）对应的两个“TODO: Give mutex here”注释位置添加以下互斥量指定代码。

xSemaphoreGive(xMutexOLED);

## 测试

1. 再次编译、下载和运行程序。OLED上的信息现在应稳定显示并正确更新。

# 添加时钟任务

添加一个名为ClockTask的任务，该任务每秒都会在LCD上显示经过的时间。该代码使用xMutexOLED来安全地访问OLED，因为它属于共享资源。

## 任务准备工作

1. 在user.h中的“TODO: Declare function ClockTask here”注释位置添加以下代码。

void ClockTask ( void \* pvParameters);

1. 在user.c中的“TODO: Define function ClockTask here”注释位置添加以下任务代码。

void ClockTask(void \* pvParameters) {

int h=0, m=0, s=0;

char buff[16];

TickType\_t xLastWakeTime;

// Initialise the xLastWakeTime variable with the current time.

xLastWakeTime = xTaskGetTickCount();

while (1) {

s++;

if (s>59) {

m++;

if (m>59) {

h++;

}

}

sprintf(buff, "%2u:%02u:%02u", h, m, s);

xSemaphoreTake(xMutexOLED, portMAX\_DELAY);

OledSetCursor(0,3);

OledPutString(buff);

xSemaphoreGive(xMutexOLED);

vTaskDelayUntil(&xLastWakeTime, 1513);

}

}

1. 在main函数（在main.c中）中的“TODO: Create task ClockTask here”注释位置添加以下任务创建代码。

xTaskCreate(ClockTask, "Clock Task", configMINIMAL\_STACK\_SIZE, NULL, 1, NULL );

## 测试

1. 再次编译、下载和运行程序。显示屏的最后一行应显示时间递增。不过，它不会以每秒一次的频率运行。

## 更正任务周期

1. 我们希望时钟任务每秒运行一次。这由vTaskDelayUntil()调用中的第二个参数控制。请记住，每秒有configTICK\_RATE\_HZ个调度程序时标。将对vTaskDelayUntil()的调用更改为对以下内容的调用：

vTaskDelayUntil(&xLastWakeTime, configTICK\_RATE\_HZ);

1. 再次编译、下载和运行程序。显示屏的最后一行应显示时间以合适的速度递增。

# 使用二进制信号量实现任务触发

添加更改时钟任务，使其在5到15秒（包括5秒和15秒）的范围内每秒触发一次任务2。

我们将在user.c和user.h中使用二进制信号量xSemTrigger来控制任务触发。

## 信号量准备工作

1. 在user.h中的“TODO: Declare semaphore handle here”注释位置声明信号量句柄。

SemaphoreHandle\_t xSemTrigger;

1. 在user.c中的“TODO: Define semaphore handle here”注释位置定义信号量句柄。

SemaphoreHandle\_t xSemTrigger;

1. 在CreateTaskSyncStructures函数（在main.c中）中的“TODO: Create semaphore here”注释位置添加以下信号量初始化代码。

xSemTrigger = xSemaphoreCreateBinary();

if( xSemTrigger == NULL )

{ // Error - the semaphore was not created successfully.

while (1)

;

}

## 等待信号量

1. 在Task2函数（在user.c中）中的“TODO: Take semaphore here”注释位置添加以下信号量获取代码。

xSemaphoreTake(xSemTrigger, portMAX\_DELAY);

1. 删除Task2中的vTaskDelay调用。不再需要此调用，因为每次发出xSemTrigger信号时，Task2都会运行一次，然后会停止，等待下一次发出信号。

## 发出信号量信号

1. 在ClockTask函数（在user.c中）中的“TODO: Give semaphore here”注释位置添加以下信号量指定代码。

if ((s >= 5) && (s <= 15)) {

xSemaphoreGive(xSemTrigger);

}

## 测试

1. 再次编译、下载和运行程序。验证OLED显示的内容是否仅为任务2在5到15秒运行的次数。

# 使用队列发送和接收串行数据

更改程序，使其可发送和接收串行数据。

* 发送：ClockTask每秒会向串行端口发送一次时间信息。
* 接收：新任务SerInTask会在OLED的顶行显示从串行端口接收到的所有字符。

我们将在UART.c和UART.h中使用xQueueSerIn和xQueueSerOut这两个队列保存串行数据。该代码还使用xMutexOLED来安全地访问OLED，因为它属于共享资源。

## UART准备工作

1. 在InitApp函数（在user.c中）中的“// TODO: Add UART4 Initialization call here”注释位置添加以下代码。

UART4\_init();

1. 在main函数（在main.c中）中的“// TODO: Add UART4 test code here”注释位置添加以下UART测试代码。启动时，该测试代码会将一条消息打印输出到串行端口，无需使用中断或队列。由于它很简单，因此成为验证基本串行通信是否起作用的有效途径。

UART4\_puts("RTOS Lab!\n\r");

// Wait until serial transmission is done

while (!U4STAbits.TRMT)

;

## 队列准备工作

1. 在UART.c中的“TODO: Define queue handles here”注释位置定义两个队列句柄。

QueueHandle\_t xQueueSerIn;

QueueHandle\_t xQueueSerOut;

1. 在UART.h中的“TODO: Declare queue handles here”注释位置声明这两个队列句柄。

QueueHandle\_t xQueueSerIn;

QueueHandle\_t xQueueSerOut;

1. 在CreateTaskSyncStructures函数（在main.c中）中的“TODO: Create queues here”注释位置添加以下代码。该代码将创建两个队列，每个队列均可保存64个元素。每个元素的大小均为一个字节，由sizeof(char)确定。

xQueueSerIn = xQueueCreate(64, sizeof(char));

if (xQueueSerIn == NULL)

{ // Error - the queue was not created successfully.

while (1)

;

}

xQueueSerOut = xQueueCreate(64, sizeof(char));

if (xQueueSerOut == NULL)

{ // Error - the queue was not created successfully.

while (1)

;

}

## 任务级发送器代码

1. 在UART.c中的“TODO: Define UART4\_RTOS\_qputs here”注释位置定义名为UART4\_RTOS\_qputs的函数。

void UART4\_RTOS\_qputs (char \*s) {

while (\*s) {

// Load queue with data to send, block if full

xQueueSend(xQueueSerOut, s++, portMAX\_DELAY);

// Ensure transmitter is enabled

U4STAbits.UTXEN = 1;

}

}

1. 在UART.h中的“TODO: Declare UART4\_RTOS\_qputs here”注释位置声明该函数。

void UART4\_RTOS\_qputs (char \*s);

1. 在“TODO: Add serial transmit code here”注释位置修改ClockTask函数（在user.c中），方法是添加以下代码，该代码的作用是在将要发送的时间字符串排队之前，为其附加硬回车和换行：

strcat(buff, "\r\n"); // append CR/LF

UART4\_RTOS\_qputs(buff);

## 发送器ISR代码

1. 在“TODO: Declare ISR\_UART4\_TX here”注释位置为UART 4发送中断事件定义ISR。

void \_\_ISR(\_UART4\_TX\_VECTOR, IPL2SOFT) ISR\_UART4\_TX (void) {

char txc;

// More data to send? xQueue... returns true if so.

if (xQueueReceiveFromISR(xQueueSerOut, &txc, NULL)) {

U4TXREG = txc;

} else { // no more data to send, disable transmitter

U4STAbits.UTXEN = 0;

}

// Reset interrupt flag

IFS5bits.U4TXIF = 0;

portEND\_SWITCHING\_ISR(1);

}

## 任务级接收器代码

1. 定义一个任务函数，以在OLED上显示接收到的串行数据。在user.c中的“TODO: Define SerialInTask here”注释位置放置该代码。

void SerialInTask(void \* pvParameters) {

// Display incoming characters on row 0 of OLED

char c = '\0';

int x = 0, y = 0; // starting character location

IEC5bits.U4RXIE = 1; // Enable receive interrupt

while (1) {

xQueueReceive(xQueueSerIn, &c, portMAX\_DELAY);

xSemaphoreTake(xMutexOLED, portMAX\_DELAY);

OledSetCursor(x, y);

OledPutChar(c);

if (x == 0) { // At beginning of line again?

OledPutString(" "); // erase rest of line

}

xSemaphoreGive(xMutexOLED);

x++;

if (x > 15)

x = 0;

}

}

1. 在user.h中的“TODO: Declare function SerialInTask here”注释位置声明该任务函数。

void SerialInTask(void \* pvParameters);

1. 在main函数（在main.c中）中的“TODO: Create SerialInTask here”注释位置添加一个调用以便为SerialInTask创建一个任务。

xTaskCreate(SerialInTask, "Serial In Task", configMINIMAL\_STACK\_SIZE, NULL, 4, NULL );

## 接收器ISR代码

1. 在“TODO: Declare ISR\_UART4\_TX here”注释位置为UART 4发送中断事件定义ISR。

void \_\_ISR(\_UART4\_RX\_VECTOR, IPL1SOFT) ISR\_UART4\_RX (void) {

char rxc; // received character

rxc = U4RXREG; // Read character from UART

if (xQueueSendFromISR(xQueueSerIn, &rxc, NULL)) {

BIOS\_LD1\_PORT\_BIT = 0; // Turn off LED if space in queue

} else {

BIOS\_LD1\_PORT\_BIT = 1; // Turn on LED to signal queue full error

}

// Reset interrupt flag

IFS5bits.U4RXIF = 0;

// Exit ISR, allowing context switch

portEND\_SWITCHING\_ISR(1);

}

## 测试

1. 编译、下载和运行程序。
2. 验证OLED是否仍正确显示任务1和任务2的计数以及经过的时间。
3. 按照串行通信练习中之前的操作，启动PC上的Teraterm或类似串行通信程序。选择115,200波特，无奇偶校验，1个停止位。
4. 验证Teraterm是否每秒都会显示一个与OLED上所显示的时间匹配的新时间戳（例如，00:01:31）。还应验证每次发送新时间戳时ChipKIT WiFIRE板上的左侧串行LED（LD6）是否都会点亮。
5. 验证在Teraterm处于激活状态时按下PC键盘上的按键，相应字符是否会显示在OLED的顶行。还应验证每次在Teraterm中按下按键（和发送字符）时，右侧串行LED（LD5）是否都会点亮。
6. 验证显示屏和Teraterm是否正确更新，甚至包括持续发送突发字符（例如，使用自动重复或快速输入功能）的情况。

# 完成

1. 现在可以关闭MPLAB X IDE并断开PICkit 3调试器和chipKIT板。