Лабораторная работа 10:   
Параллелизм: расширенные возможности - Использование RTOS

Александр Г. Дин   
Факультет Электротехники и Компьютерной Инженерии

Университет штата Северная Каролина  
8/24/2015, Версия 1.0



**

*Содержание*

[Обзор 5](#_Toc459361747)

[Подготовка 5](#_Toc459361748)

[Установка RTOS и создание проекта 5](#_Toc459361749)

[Открытие и конфигурирование проекта 5](#_Toc459361750)

[Загрузка кода 6](#_Toc459361751)

[Изучение начальной программы 6](#_Toc459361752)

[Использование Мьютекс-семафора для совместного доступа к OLED дисплею 6](#_Toc459361753)

[Подготовка мьютекса 6](#_Toc459361754)

[Получение мьютекса 7](#_Toc459361755)

[Выдача мьютекса 7](#_Toc459361756)

[Проверка 7](#_Toc459361757)

[Добавление задачи вывода времени (ClockTask) 7](#_Toc459361758)

[Подготовка задачи 7](#_Toc459361759)

[Проверка 8](#_Toc459361760)

[Исправление периода задачи 8](#_Toc459361761)

[Использование двоичного семафора для запуска задачи 9](#_Toc459361762)

[Подготовка семафора 9](#_Toc459361763)

[Ожидание семафора 9](#_Toc459361764)

[Сигнализация семафором 9](#_Toc459361765)

[Проверка 10](#_Toc459361766)

[Использование очередей для передачи и получения последовательных данных 10](#_Toc459361767)

[Подготовка UART 10](#_Toc459361768)

[Подготовка очереди 10](#_Toc459361769)

[Код передачи уровня задач 11](#_Toc459361770)

[Процедура обработки прерывания передатчика 11](#_Toc459361771)

[Код приема уровня задач 12](#_Toc459361772)

[Код процедуры обработки прерывания приемника 12](#_Toc459361773)

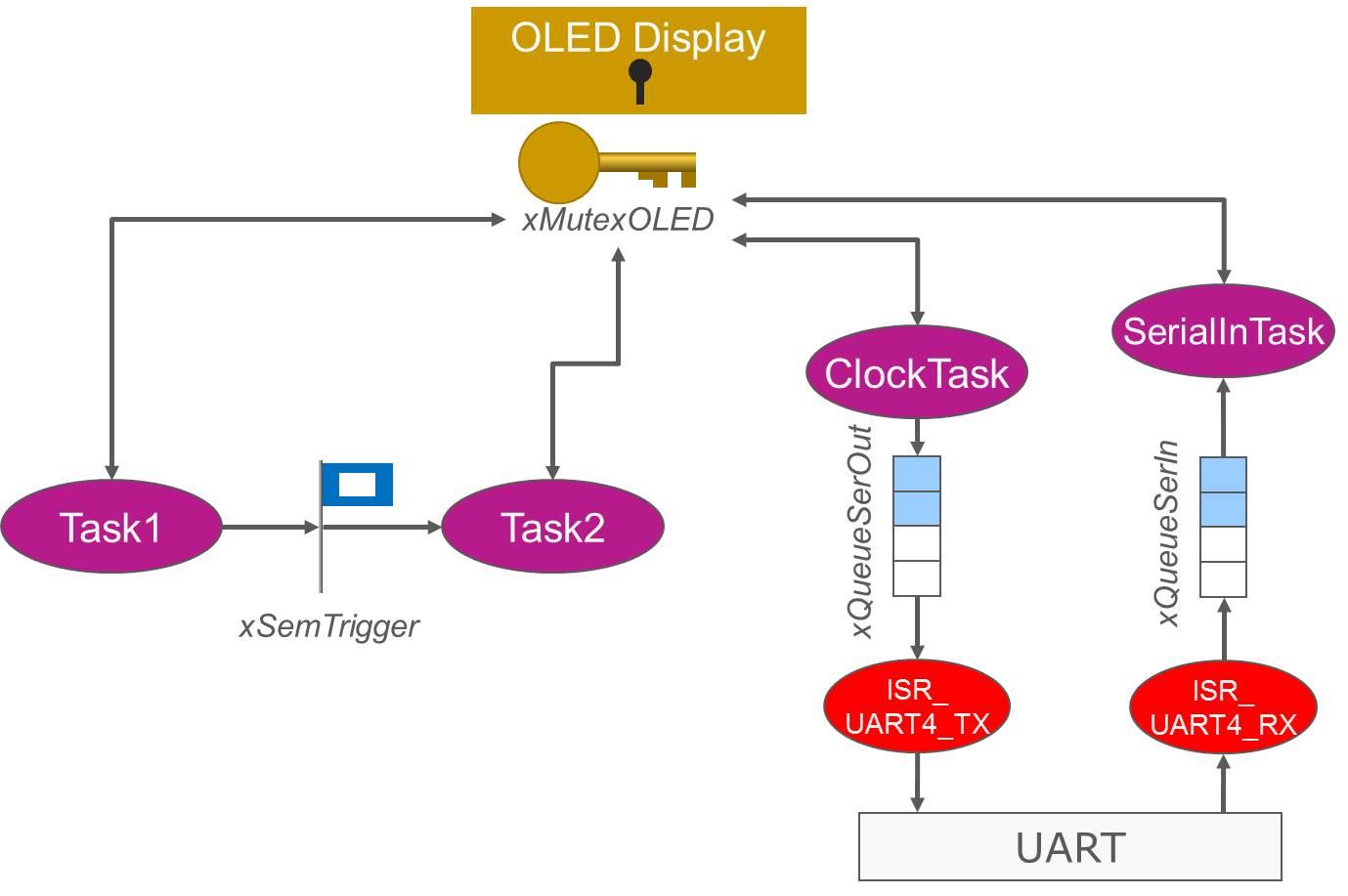
[Проверка 13](#_Toc459361774)

[Завершение 13](#_Toc459361775)

# Обзор

В этой лабораторной работа вам нужно будет завершить разработку программы: добавить код, который использует RTOS для управления задачами и коммуникаций. Для вывода информации на OLED дисплей используется плата Basic I/O Shield.

На приведенной ниже диаграмме показана архитектура системы, которую вы будете разрабатывать в этой работе. Вам, возможно, будет полезно иногда возвращаться к этой диаграмме по мере добавления функций к вашей программе.



# Подготовка

## Установка RTOS и создание проекта

1. Если вы не сделали этого до сих пор, загрузите и установите FreeRTOS на ваш ПК. Она доступна по ссылке [http://www.freertos.org](http://www.freertos.org/).
2. Распакуйте архив Module10\_Adv\_Concurrency\_Code.zip в каталог, где установлена FreeRTOS: FreeRTOS/Demo/PIC32MZ\_MPLAB. При этом должны создаться три каталога. Для этой лабораторной работы следует использовать проект в каталоге Lab10\_Adv\_Concurrency.

## Открытие и конфигурирование проекта

1. Используя колодку или кабель, вставьте отладчик PICkit 3 в разъем JP1 (ICSP) платы chipKIT. Удостоверьтесь в том, что белый треугольник отладчика PICkit 3 указывает на контакт 1 разъема JP1 (квадратная площадка).
2. Установите плату Basic I/O Shield на chipKIT.
3. Присоедините отладчик PICkit 3 и плату chipKIT Wi-Fire к ПК с помощью USB-кабелей.
4. На ПК запустите программу MPLAB X IDE.
5. В меню **File** MPLAB X, нажмите **Open Project… ,** перейдите в каталог FreeRTOS/Demo/PIC32MZ\_MPLAB и выберите проект **Lab10\_Adv\_Concurrency**.
6. Нажмите на иконку с метлой и молотком (в верхней части окна среды разработки MPLAB X), это очистит проект и соберет его заново.   
   
7. На вкладке output, в нижней части правой панели среды разработки MPLAB X, можно наблюдать за ходом процесса и его результатами. Обратите внимание на сообщение BUILD SUCCESSFUL, которое указывает на общее состояние.

## Загрузка кода

1. Теперь вы можете загрузить программу на плату chipKIT. Для этого нажмите на иконку с зеленой стрелкой вниз (в верхней части окна среды разработки MPLAB X).  
    
2. Во время программирования светодиод состояния программатора PICkit3 будет мигать красным и зеленым светом, а после завершения останется зеленым. В окне Output откроется вкладка **PICkit 3 debugger**, на которой будет показано состояние программирования.

# Изучение начальной программы

Начальная программа содержит две задачи (Task1, Task2), которые периодически выполняются. Каждая задача обновляет OLED дисплей: выводит информацию о количестве своих запусков.

1. Понаблюдайте за OLED дисплеем при выполнении программы. Обратите внимание на то, что информация на OLED дисплее мигает, нестабильна, иногда выводится в неправильном месте. Это происходит потому, что задача, которая имеет доступ к OLED дисплею, вытесняется другой задачей, которая тоже обращается к нему.

# Использование Мьютекс-семафора для совместного доступа к OLED дисплею

Для исправления ошибок при доступе к OLED дисплею мы будем использовать мьютекс (mutual exclusion, взаимное исключение), семафор, который будет контролировать доступ к OLED дисплею. Перед доступом к OLED задача должна захватить мьютекс xMutexOLED, это разрешает доступ. После завершения доступа задача должна выдать мьютекс.

## Подготовка мьютекса

1. Добавьте следующие директивы включения заголовочных файлов в файл OLED.h после комментария “TODO: Add RTOS header file includes”.

#include "FreeRTOS.h"

#include "semphr.h"

1. Объявите дескриптор мьютекса в файле OLED.h после комментария “TODO: Declare mutex handle here”.

SemaphoreHandle\_t xMutexOLED;

1. Определите дескриптор мьютекса в файле OLED.c после комментария “TODO: Define mutex handle here”.

SemaphoreHandle\_t xMutexOLED;

1. Добавьте следующий код инициализации мьютекса в функцию CreateTaskSyncStructures (в файле main.c) после комментария “TODO: Create mutex here”.

xMutexOLED = xSemaphoreCreateMutex();

if( xMutexOLED == NULL )

{ // Ошибка - семафор не создан

while (1)

;

}

## Получение мьютекса

1. Добавьте следующий код захвата мьютекса в функции Task1 и Task2 (в файле main.c) после каждого из комментариев “TODO: Take mutex here”.

xSemaphoreTake(xMutexOLED, portMAX\_DELAY);

## Выдача мьютекса

1. Добавьте следующий код выдачи мьютекса в функции Task1 и Task2 (в файле main.c) после каждого из комментариев “TODO: Give mutex here”.

xSemaphoreGive(xMutexOLED);

## Проверка

1. Вновь соберите, загрузите и запустите программу. Информация на OLED дисплее должна быть стабильной и обновляться корректно.

# Добавление задачи вывода времени (ClockTask)

Добавьте задачу ClockTask, которая каждую секунду выводит на дисплей прошедшее время. Для безопасного доступа к OLED этот код использует мьютекс xMutexOLED.

## Подготовка задачи

1. Добавьте следующий код в файл user.h после комментария “TODO: Declare function ClockTask here”.

void ClockTask ( void \* pvParameters);

1. Добавьте следующий код задачи в файл user.h после комментария “TODO: Define function ClockTask here”.

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. Добавьте следующий код создания задачи в функцию main (в файле user.c) после комментария “TODO: Create task ClockTask here”.

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

## Проверка

1. Вновь соберите, загрузите и запустите программу. Последняя строка дисплея должна показывать увеличивающееся время. Она, однако, обновляется не один раз в секунду.

## Исправление периода задачи

1. Нам нужно, чтобы задача, выводящая время, запускалась один раз в секунду. Это контролируется вторым параметром при вызове функции vTaskDelayUntil(). Вспомните, что существует конфигурационный параметр configTICK\_RATE\_HZ, устанавливающий количество тактов диспетчер в секунду. Измените вызов функции vTaskDelayUntil() на следующий:

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

1. Вновь соберите, загрузите и запустите программу. Последняя строка дисплея должна показывать увеличивающееся время с правильной скоростью.

# Использование двоичного семафора для запуска задачи

Давайте изменим задачу ClockTask так, чтобы она запускала задачу Task2 один раз в секунду в интервале от 5 секунды до 15, границы интервала включены.

Мы будем использовать двоичный семафор xSemTrigger в user.c и user.h для управления запуском задачи.

## Подготовка семафора

1. Объявите дескриптор семафора в файле user.h после комментария “TODO: Declare semaphore handle here”.

SemaphoreHandle\_t xSemTrigger;

1. Определите дескриптор семафора в файле user.c после комментария “TODO: Define semaphore handle here”.

SemaphoreHandle\_t xSemTrigger;

1. Добавьте следующий код инициализации семафора в функцию CreateTaskSyncStructures (в файле main.c) после комментария “TODO: Create semaphore here”.

xSemTrigger = xSemaphoreCreateBinary();

if( xSemTrigger == NULL )

{ // Ошибка - семафор не создан

while (1)

;

}

## Ожидание семафора

1. Добавьте следующий код захвата семафора в функцию Task2 (в файле user.c) после комментария “TODO: Take semaphore here”.

xSemaphoreTake(xSemTrigger, portMAX\_DELAY);

1. Удалите вызов vTaskDelay в Task2. Он не нужен, поскольку Task2 будет выполняться один раз, когда xSemTrigger сигналит, а затем останавливаться и ожидать следующего сигнала.

## Сигнализация семафором

1. Добавьте следующий код выдачи семафора в функцию ClockTask (в файле user.c) после комментария “TODO: Give semaphore here”.

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

xSemaphoreGive(xSemTrigger);

}

## Проверка

1. Вновь соберите, загрузите и запустите программу. Удостоверьтесь в том, что OLED дисплей показывает счет выполнений задачи 2 (task2) только в интервале от 5 до 15 секунды.

# Использование очередей для передачи и получения последовательных данных

Давайте изменим программу так, чтобы она могла передавать и получать последовательные данные.

* Передача: ClockTask будет каждую секунду предавать информацию о времени по последовательному порту.
* Прием: Новая задача SerInTask будет показывать в верхней строке OLED дисплея все символы, полученные по последовательному порту.

Для хранения данных мы будем использовать две очереди xQueueSerIn и xQueueSerOut в файлах UART.c и UART.h. Т.к. OLED дисплей является общим ресурсом, то для безопасного доступа к нему используется мьютекс xMutexOLED.

## Подготовка UART

1. Добавьте следующий код в функцию InitApp (в файле user.c) после комментария “TODO: Add UART4 Initialization call here”.

UART4\_init();

1. Добавьте следующий код тестирования UART в функцию main (в файле main.c) после комментария “TODO: Add UART4 test code here”. При старте этот код передает сообщение по последовательному порту без использования прерываний или очередей. Из-за своей простоты этот код удобен для проверки работоспособности основных функций последовательных коммуникаций.

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

// Wait until serial transmission is done

while (!U4STAbits.TRMT)

;

## Подготовка очереди

1. Определите два дескриптора очередей в файле UART.c после комментария “TODO: Define queue handles here”.

QueueHandle\_t xQueueSerIn;

QueueHandle\_t xQueueSerOut;

1. Объявите два дескриптора очередей в файле UART.h после комментария “TODO: Declare queue handles here”.

QueueHandle\_t xQueueSerIn;

QueueHandle\_t xQueueSerOut;

1. Добавьте следующий код в функцию CreateTaskSyncStructures (в файле main.c) после комментария “TODO: Create queues here”. Этот код создает две очереди, каждая может содержать 64 элемента. Размер каждого элемента равен одному байту (определено sizeof(char)).

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)

;

}

## Код передачи уровня задач

1. Определите функцию UART4\_RTOS\_qputs в файле UART.c после комментария “TODO: Define UART4\_RTOS\_qputs here”.

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. Объявите эту функцию в файле UART.h после комментария “TODO: Declare UART4\_RTOS\_qputs here”.

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

1. Измените функцию ClockTask в файле user.c после комментария “TODO: Add serial transmit code here”, новый код добавляет символы возврата каретки и перевода строки к строке времени перед постановкой ее в очередь передачи:

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

UART4\_RTOS\_qputs(buff);

## Процедура обработки прерывания передатчика

1. Определите процедуру обработки прерывания для прерывания по передаче UART 4 после комментария “TODO: Declare ISR\_UART4\_TX here.”

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);

}

## Код приема уровня задач

1. Определите функцию задачи, которая будет выводить на OLED дисплей данные, полученные через последовательный порт. Добавьте следующий код в файл user.c после комментария “TODO: Define SerialInTask here.”

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. Объявите эту функцию задачи в файле UART.h после комментария “TODO: Declare function SerialInTask here.”

void SerialInTask(void \* pvParameters);

1. Добавьте вызов функции создания задачи для SerialInTask в функцию main (в файле main.c) после комментария “TODO: Create SerialInTask here”.

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

## Код процедуры обработки прерывания приемника

1. Определите процедуру обработки прерывания для прерывания приема UART 4 после комментария “TODO: Declare ISR\_UART4\_TX here.”

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);

}

## Проверка

1. Соберите, загрузите и запустите программу.
2. Удостоверьтесь в том, что OLED дисплей правильно считает количество выполнений задач 1 и 2 и прошедшее время.
3. Установите Teraterm или другую подобную программу (эмулятор терминала) на ваш ПК, как вы делали ранее, когда выполняли задачи, связанные с последовательными коммуникациями. Выберите скорость 115,200 бод (baud), отсутствие контроля по четности (no parity), один стоп-бит (one stop bit).
4. Проверьте, что каждую секунду Teraterm показывает новую метку времени, которая советует времени, выводимому на OLED дисплей. Также убедитесь в том, что левый светодиод (LD6, последовательный порт) платы ChipKIT WiFIRE светится при передаче каждой временной метки.
5. Понажимайте клавиши ПК при активной программе Teraterm и проверьте, что соответствующие символы появляются в верхней строке OLED дисплея. Также убедитесь в том, что правый светодиод (LD5, последовательный порт) платы ChipKIT WiFIRE светится при нажатии клавиши в программе Teraterm (и передаче символа).
6. Удостоверьтесь в том, что информация на дисплее и Teraterm правильно обновляются даже при передаче большого количества символов (автоповтор или быстрая печать).

# Завершение

1. Вы можете закрыть среду разработки MPLAB X и отсоединить отладчик PICkit 3 и плату chipKIT.