Лабораторная работа №7.
Тема: "Знакомство с Arduino и Proteus". RTOS.
Цель работы: освоить базовые навыки работы с Arduino IDE и Proteus.

Теоретическая часть

1. Что такое RTOS?

RTOS (Real-Time Operating System) — это операционная система реального времени, которая гарантирует выполнение задач **строго в заданные временные рамки**. В отличие от обычных ОС (Windows, Linux), где задачи выполняются "когда получится", RTOS обеспечивает **детерминированное поведение** (предсказуемое время отклика).

Где применяется RTOS?

- Промышленная автоматизация.
- Медицинские устройства.
- Робототехника.
- Авионика и космические системы.
- Умные устройства с жесткими временными ограничениями.

2. Зачем RTOS в Arduino?

Обычно Arduino работает в **однопоточном режиме** (loop() + delay()), что вызывает проблемы:

- **Блокировка кода** (если delay(1000), то система "зависает" на 1 секунду).
- **Нет многозадачности** (сложно одновременно опрашивать датчики, управлять моторами и общаться по UART).
- **Нет приоритетов** (критичные задачи могут "простаивать").

RTOS решает эти проблемы:

- Позволяет запускать **несколько задач параллельно** (квази-параллельно на одном ядре).
- Дает контроль над **приоритетами задач**.
- Позволяет избежать delay() через **системные таймеры и семафоры**.

3. Популярные RTOS для Arduino

1. FreeRTOS

Самая распространенная бесплатная RTOS для микроконтроллеров.
Поддерживаемые Arduino:

- ESP32, ESP8266 (из коробки).
- AVR (Arduino Uno, Nano) — с ограничениями (мало RAM).

Основные функции:

- Создание задач (xTaskCreate).
- Очереди (xQueue).
- Семафоры (xSemaphore).
- Таймеры (xTimer).

Пример кода для ESP32:

```
#include <Arduino.h>
#include <FreeRTOS.h>

void task1(void *pvParam) {
  while(1) {
    Serial.println("Task 1");
    vTaskDelay(1000 / portTICK_PERIOD_MS); // Не блокирующая задержка
  }
}

void task2(void *pvParam) {
  while(1) {
    Serial.println("Task 2");
    vTaskDelay(2000 / portTICK_PERIOD_MS);
  }
}

void setup() {
  Serial.begin(115200);
  xTaskCreate(task1, "Task 1", 1000, NULL, 1, NULL);
  xTaskCreate(task2, "Task 2", 1000, NULL, 1, NULL);
}

void loop() {} // Не используется в FreeRTOS
```

---

2. ChibiOS / ChibiOS-ARM

Более мощная RTOS, поддерживает **STM32** (Arduino с ядрами ARM).
Особенности:

- Поддержка **потоков, мьютексов, событий**.

- Высокая надежность (используется в промышленности).

3. Zephyr RTOS

Современная RTOS от Linux Foundation, поддерживает **nRF52, ESP32, STM32**.
Плюсы:

- Поддержка **BLE, WiFi, файловых систем**.
- Хорошая документация.

4. Когда использовать RTOS, а когда хватит loop()?

Критерий	Обычный loop()	RTOS
Сложность проекта	Простые скетчи (до 3 задач)	Многозадачные системы
Время отклика	Нет гарантий	Жесткие временные рамки
Ресурсы МК	Мало RAM (Uno: 2 КБ)	Нужно ≥ 8 КБ RAM (ESP32)
Параллелизм	Только кооперативная многозадачность	Реальная псевдопараллельность

5. Практический пример: Многозадачность на Arduino + FreeRTOS

Задача:

- **Задача 1:** Мигаем светодиодом каждые 500 мс.
- **Задача 2:** Отправляем данные датчика в Serial каждые 1 сек.

Код для ESP32:

```
#include <Arduino.h>
#include <FreeRTOS.h>

#define LED_PIN 2

void blinkTask(void *pvParam) {
  pinMode(LED_PIN, OUTPUT);
  while(1) {
    digitalWrite(LED_PIN, !digitalRead(LED_PIN));
    vTaskDelay(500 / portTICK_PERIOD_MS);
  }
}

void sensorTask(void *pvParam) {
  while(1) {
```

```
    int val = analogRead(A0);
    Serial.println("Sensor: " + String(val));
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}

void setup() {
  Serial.begin(115200);
  xTaskCreate(blinkTask, "Blink", 1000, NULL, 1, NULL);
  xTaskCreate(sensorTask, "Sensor", 1000, NULL, 1, NULL);
}

void loop() {} // FreeRTOS берет управление на себя
```

Вывод

- **RTOS нужен**, если требуется:
  - Многозадачность без delay().
  - Жесткий контроль времени (ПИД-регуляторы, роботы).
- **Для простых проектов** хватит loop() + millis().
- **Лучшие RTOS для Arduino:** FreeRTOS (ESP32), ChibiOS (STM32), Zephyr (nRF52).

Подробное объяснение основных функций FreeRTOS в Arduino

1. Создание задач (xTaskCreate)

Назначение

Функция для создания новой задачи (потока выполнения) в RTOS. Каждая задача работает "параллельно" с другими (на самом деле переключается планировщиком).

Синтаксис

```
BaseType_t xTaskCreate(
  TaskFunction_t pvTaskCode,     // Функция задачи
  const char * const pcName,     // Имя задачи (для отладки)
  configSTACK_DEPTH_TYPE usStackDepth, // Размер стека (в словах)
  void *pvParameters,            // Параметры для передачи в задачу
  UBaseType_t uxPriority,        // Приоритет (0 - самый низкий)
  TaskHandle_t *pxCreatedTask    // Указатель на handle задачи
);
```

Пример

```
void myTask(void *pvParam) {
  while(1) {
    Serial.println("Task is running");
    vTaskDelay(1000 / portTICK_PERIOD_MS);
```

```
  }
}

void setup() {
  xTaskCreate(
    myTask,       // Функция задачи
    "MyTask",     // Имя
    1000,         // Размер стека (байт)
    NULL,         // Параметры
    1,            // Приоритет
    NULL          // Handle (не сохраняем)
  );
}
```

Ключевые моменты

- **Размер стека** зависит от сложности задачи (минимум 256 для простых задач).
- **Приоритеты**: 0 (низкий) → configMAX_PRIORITIES-1 (высокий).
- **vTaskDelay** — неблокирующая задержка.

---

2. Очереди (xQueueCreate, xQueueSend, xQueueReceive)

Назначение

Механизм для **безопасного** обмена данными между задачами.

Синтаксис

```
QueueHandle_t xQueueCreate(
  UBaseType_t uxQueueLength, // Длина очереди
  UBaseType_t uxItemSize     // Размер элемента (байт)
);

BaseType_t xQueueSend(
  QueueHandle_t xQueue,      // Handle очереди
  const void * pvItemToQueue,// Данные для отправки
  TickType_t xTicksToWait    // Время ожидания (portMAX_DELAY - бесконечно)
);

BaseType_t xQueueReceive(
  QueueHandle_t xQueue,      // Handle очереди
  void *pvBuffer,            // Буфер для приема
  TickType_t xTicksToWait    // Время ожидания
);
```

Пример

```
QueueHandle_t queue;

void senderTask(void *pvParam) {
  int data = 0;
  while(1) {
    xQueueSend(queue, &data, portMAX_DELAY);
```

```
    data++;
    vTaskDelay(500 / portTICK_PERIOD_MS);
  }
}

void receiverTask(void *pvParam) {
  int received;
  while(1) {
    if(xQueueReceive(queue, &received, portMAX_DELAY) {
      Serial.println(received);
    }
  }
}

void setup() {
  queue = xQueueCreate(5, sizeof(int));
  xTaskCreate(senderTask, "Sender", 1000, NULL, 1, NULL);
  xTaskCreate(receiverTask, "Receiver", 1000, NULL, 1, NULL);
}
```

Ключевые моменты

- **Очередь FIFO** (первый пришел — первый вышел).
- **Блокирующий режим**: если очередь пуста/полна, задача ждет.
- **Можно передавать структуры** (но лучше избегать сложных объектов C++).

---

3. Семафоры (xSemaphoreCreateBinary, xSemaphoreGive, xSemaphoreTake)

Назначение

Синхронизация задач (например, доступ к общему ресурсу).

Типы семафоров

- **Бинарные**: 0 или 1 (как мьютекс).
- **Счетные**: несколько разрешений.

Синтаксис

```
SemaphoreHandle_t xSemaphoreCreateBinary();
SemaphoreHandle_t xSemaphoreCreateCounting(UBaseType_t maxCount, UBaseType_t initCount);

BaseType_t xSemaphoreGive(SemaphoreHandle_t xSemaphore);
BaseType_t xSemaphoreTake(SemaphoreHandle_t xSemaphore, TickType_t xTicksToWait);
```

Пример (доступ к Serial)

```
SemaphoreHandle_t serialSem;

void task1(void *pvParam) {
  while(1) {
    xSemaphoreTake(serialSem, portMAX_DELAY);
    Serial.println("Task 1 using Serial");
```

```
    xSemaphoreGive(serialSem);
    vTaskDelay(1);
  }
}

void task2(void *pvParam) {
  while(1) {
    xSemaphoreTake(serialSem, portMAX_DELAY);
    Serial.println("Task 2 using Serial");
    xSemaphoreGive(serialSem);
    vTaskDelay(1);
  }
}

void setup() {
  serialSem = xSemaphoreCreateBinary();
  xSemaphoreGive(serialSem); // Изначально свободен
  xTaskCreate(task1, "Task1", 1000, NULL, 1, NULL);
  xTaskCreate(task2, "Task2", 1000, NULL, 1, NULL);
}
```

Ключевые моменты

- **Мьютекс** — частный случай бинарного семафора.
- **Deadlock**: если задача не отпустит семафор, другие зависнут.

---

4. Таймеры (xTimerCreate, xTimerStart)

Назначение

Выполнение функций по таймеру (в контексте службы таймера, не задачи).

Синтаксис

```
TimerHandle_t xTimerCreate(
  const char *pcTimerName,        // Имя
  TickType_t xTimerPeriod,        // Период (в тиках)
  UBaseType_t uxAutoReload,       // pdTRUE - автоповтор
  void *pvTimerID,                // ID таймера
  TimerCallbackFunction_t pxCallbackFunction // Функция
);

BaseType_t xTimerStart(TimerHandle_t xTimer, TickType_t xTicksToWait);
```

Пример

```
cpp

Copy

void timerCallback(TimerHandle_t xTimer) {
  Serial.println("Timer fired!");
}

void setup() {
```

```
  TimerHandle_t timer = xTimerCreate(
    "MyTimer",
    1000 / portTICK_PERIOD_MS, // 1 сек
    pdTRUE,                    // Автоповтор
    NULL,
    timerCallback
  );
  xTimerStart(timer, 0);
}
```

Ключевые моменты

- **Таймеры выполняются в контексте службы таймера** (не в задачах!).
- **Не блокируют** другие задачи.

Сравнение механизмов

Механизм	Когда использовать	Преимущества
Очереди	Обмен данными между задачами	Безопасная передача
Семафоры	Синхронизация доступа к ресурсам	Защита от гонок данных
Таймеры	Периодические действия	Точно по времени, без создания задач

Практические советы

1. **Размер стека**: Начинайте с 1024 байт, увеличивайте при stack overflow.
2. **Приоритеты**: Критичные задачи — высокий приоритет.
3. **Очереди**: Всегда проверяйте возвращаемые значения (pdPASS/pdFAIL).
4. **Семафоры**: Используйте xSemaphoreTake с таймаутом, чтобы избежать deadlock.

Пример комплексного использования (датчик + вывод + сеть):

```
QueueHandle_t sensorQueue;
SemaphoreHandle_t i2cSem;

void readSensorTask(void *pvParam) {
  float data;
  while(1) {
    xSemaphoreTake(i2cSem, portMAX_DELAY);
    data = readI2CSensor(); // Чтение датчика
```

```
    xSemaphoreGive(i2cSem);
    xQueueSend(sensorQueue, &data, portMAX_DELAY);
    vTaskDelay(100 / portTICK_PERIOD_MS);
  }
}

void sendDataTask(void *pvParam) {
  float received;
  while(1) {
    xQueueReceive(sensorQueue, &received, portMAX_DELAY);
    sendToWiFi(received); // Отправка
  }
}
```

Задание:

Задание 1. Многозадачность — мигание двух светодиодов
Цель: научиться использовать RTOS для выполнения нескольких задач одновременно.
Компоненты Proteus:

- Arduino Uno.
- два светодиода.
- два резистора по 220 Ом).

Описание:

- Первый светодиод подключите к D8 через резистор 220 Ом.
- Второй светодиод подключите к D9 через резистор 220 Ом.

Напишите программу, используя FreeRTOS:
Создайте две задачи:

- Первая задача мигает первым светодиодом каждые 500 мс.
- Вторая задача мигает вторым светодиодом каждые 1000 мс.
- Используйте функцию vTaskDelay() для управления временными интервалами.

Задание 2. Многозадачный опрос датчиков (I2C + UART)
Цель: организовать параллельный опрос датчика и вывод значения.
Компоненты в Proteus:

- Arduino Uno
- Виртуальный терминал (Serial Monitor)
- Датчик температуры LM35 (аналоговый вход A0)
- I2C LCD 1602 (для вывода)

Код (FreeRTOS):
#include <Arduino_FreeRTOS.h>

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 16, 2);
// Задача 1: Опрос температуры (LM35)
void TaskReadTemp(void *pvParameters) {
  while (1) {
    float temp = analogRead(A0) * 0.488; // LM35: 10 мВ/°C
    Serial.print("Temp: "); Serial.print(temp); Serial.println(" C");
    vTaskDelay(1000 / portTICK_PERIOD_MS);
  }
}


// Задача 2: Вывод времени работы на LCD
void TaskLCD(void *pvParameters) {
  lcd.init(); lcd.backlight();
  while (1) {
    lcd.setCursor(0, 0);
    lcd.print("Uptime: ");
    lcd.print(millis() / 1000);
    lcd.print(" s");
    vTaskDelay(500 / portTICK_PERIOD_MS);
  }
}
void setup() {
  Serial.begin(9600);
  Wire.begin();
  xTaskCreate(TaskReadTemp, "Temp", 128, NULL, 1, NULL);
  xTaskCreate(TaskLCD, "LCD", 128, NULL, 1, NULL);
  vTaskStartScheduler();
}
void loop() {}
```

Проверка:
В Serial Monitor выводятся показания температуры
На LCD обновляется время работы

Задание 3. Управление сервоприводом и светодиодом
Цель: научиться управлять устройствами с разными задачами в RTOS.
Компоненты Proteus:

- Arduino Uno.
- Servo Motor (сервопривод).
- светодиод)
- резистор 220 Ом).

Описание:

- Сигнальный провод сервопривода подключите к D10.
- Питание (+5V) подключите к +5V.
- Землю (GND) подключите к GND.
- Анод светодиод подключите к D8 через резистор 220 Ом.
- Катод подключите к GND.

Напишите программу, используя FreeRTOS:
Создайте две задачи:

- Первая задача управляет сервоприводом: поворачивает его от 0° до 180° за 2 секунды, затем обратно.
- Вторая задача мигает светодиодом каждые 500 мс.
- Используйте vTaskDelay() для управления временными интервалами.

Задание 4. Чтение данных с потенциометра и управление светодиодом
Цель: научиться обрабатывать данные с аналогового входа в одной задаче и управлять устройством в другой.
Компоненты Proteus:
Arduino Uno.

- потенциометр, например, 10 кОм.
- светодиод.
- резистор 220 Ом).

Описание:
Подключите потенциометр:

- Левый вывод подключите к +5V.
- Правый вывод подключите к GND.
- Средний вывод (движок) подключите к A0.

Подключите светодиод:

- Анод подключите к D9 через резистор 220 Ом.
- Катод подключите к GND.

Напишите программу, используя FreeRTOS:
Создайте две задачи:

- Первая задача считывает значение с потенциометра каждые 200 мс.
- Вторая задача устанавливает яркость светодиода пропорционально значению потенциометра.

Используйте глобальную переменную или очередь (xQueue) для передачи данных между задачами.

Задание 5. Реализация простого светофора
Цель: научиться создавать многозадачное приложение с использованием RTOS.
Компоненты Proteus:

- Arduino Uno.
- три светодиода: красный, жёлтый, зелёный.
- три резистора по 220 Ом.

Описание:

Подключите три светодиода:

- Красный светодиод подключите к D8 через резистор 220 Ом.
- Жёлтый светодиод подключите к D9 через резистор 220 Ом.
- Зелёный светодиод подключите к D10 через резистор 220 Ом.

Напишите программу, используя FreeRTOS:
Создайте три задачи:

- Первая задача включает красный светодиод на 5 секунд.
- Вторая задача включает жёлтый светодиод на 2 секунды.
- Третья задача включает зелёный светодиод на 5 секунд.

Используйте vTaskDelay() для управления временными интервалами.

Задание 6. Управление дисплеем и кнопкой
Цель: научиться работать с внешними устройствами в многозадачной среде.
Компоненты Proteus:

- Arduino Uno.
- LCD Display (например, LCD 16x2).
- кнопка.
- Резистор 10 кОм) — для подтяжки кнопки.

Описание:
Подключите LCD дисплей:

- RS -> D7.
- E -> D6.
- D4 -> D5.
- D5 -> D4.
- D6 -> D3.
- D7 -> D2.
- V0 -> средний вывод потенциометра.
- VSS -> GND.
- VDD -> +5V.

Подключите кнопку:

- Один вывод кнопки подключите к D2.
- Второй вывод кнопки подключите к GND.
- Добавьте подтягивающий резистор (10 кОм) между D2 и +5V.

Напишите программу, используя FreeRTOS:
Создайте две задачи:

- Первая задача отображает текущее время (счётчик секунд) на LCD дисплее.
- Вторая задача проверяет нажатие кнопки и сбрасывает счётчик времени при нажатии.

Используйте глобальную переменную или семафор (xSemaphore) для синхронизации задач.

Задание 7. Управление температурным датчиком и светодиодом
Цель: научиться работать с датчиками и устройствами в многозадачной среде.
Компоненты Proteus:

- Arduino Uno.
- LM35 Temperature Sensor (температурный датчик).
- светодиод.
- резистор 220 Ом).

Описание:
Подключите LM35:

- Vout подключите к A0.
- Vcc подключите к +5V.
- GND подключите к GND.

Подключите светодиод:

- Анод подключите к D9 через резистор 220 Ом.
- Катод подключите к GND.

Напишите программу, используя FreeRTOS:
Создайте две задачи:

- Первая задача считывает температуру с LM35 каждые 1000 мс.
- Вторая задача включает светодиод, если температура превышает определённый порог (например, 30°C).

Используйте глобальную переменную или очередь для передачи данных между задачами.

Задание 8. Создание системы мониторинга
Цель: научиться создавать сложные многозадачные системы с использованием RTOS.
Компоненты Proteus:

- Arduino Uno.
- LDR (фоторезистор).
- LM35 Temperature Sensor (температурный датчик).
- LCD Display (например, LCD 16x2).
- светодиод.
- Два резистора, например, 10 кОм и 220 Ом).

Описание:
Подключите LDR через делитель напряжения:

- Один вывод фоторезистора подключите к +5V.
- Второй вывод фоторезистора подключите к A0 и через резистор 10 кОм к GND.

Подключите LM35:

- Vout подключите к A1.
- Vcc подключите к +5V.

- GND подключите к GND.

Подключите LCD дисплей (как в предыдущих заданиях).
Подключите светодиод:

- Анод подключите к D9 через резистор 220 Ом.
- Катод подключите к GND.

Напишите программу, используя FreeRTOS:
Создайте три задачи:

- Первая задача считывает уровень освещённости с LDR и отображает его на LCD.
- Вторая задача считывает температуру с LM35 и отображает её на LCD.
- Третья задача включает светодиод, если уровень освещённости ниже определённого порога.

Используйте очереди или семафоры для передачи данных между задачами.