

# Лабораторная работа №3.



## Управление временем.

### Функции `delay()` и `millis()`

Цель:

Освоить принципы работы с временными интервалами в Arduino. Понять ключевую разницу между блокирующей функцией `delay()` и неблокирующим подходом с использованием `millis()`. Реализовать светомузыкальный эффект (мигание светодиода в такт мелодии) без остановки основной

## Теоретическая часть

### Проблема однозадачности: функция `delay()`

Функция `delay(пауза)` приостанавливает выполнение всей программы на заданное количество миллисекунд.

- + Очень проста в использовании.
- Является блокирующей.

Во время паузы микроконтроллер "засыпает" и не может опрашивать датчики, обрабатывать кнопки или управлять другими устройствами.

### Пример проблемы:

Если в цикле `loop()` сначала идет `delay(1000)`, а затем проверка датчика, то вы будете проверять датчик только раз в секунду, и можете легко пропустить важное событие.

### Решение: функция `millis()` и неблокирующее программирование

Функция `millis()` возвращает количество миллисекунд, прошедших с момента начала выполнения программы (с момента подачи питания или сброса). Тип возвращаемого значения — `unsigned long`.

- + Не блокирует выполнение программы. Код продолжает работать.
- Требует более сложной логики для отслеживания времени.

## Принцип работы:

- 1 Запоминаем время начала какого-либо действия (`previousMillis`).
- 2 В каждом цикле `loop()` мы постоянно считываем текущее время (`currentMillis = millis()`).
- 3 Вычисляем, сколько времени прошло с момента `previousMillis`.
- 4 Если полученная разница больше или равна нужному нам интервалу,
- 5 Выполняем действие (например, мигаем светодиодом) и обновляем время `previousMillis` на текущее.

## Сравнение подходов:

Параметр	<code>delay()</code>	<code>millis()</code>
<b>Тип выполнения</b>	Блокирующий	Неблокирующий
<b>Сложность</b>	Очень просто	Сложнее, требует логики
<b>Подходит для</b>	Простых скетчей, где паузы допустимы	Сложных проектов, многозадачности
<b>Реакция на события</b>	Медленная, может пропускать события	Мгновенная

# Практическая часть

## Задание 1: Блокирующее мигание (с `delay()`)

**Цель:** Понять работу `delay()` на простом примере.

**Задание:** Загляните в коды лабораторных работ по RGB-светодиодам и пьезодинамику. Обратите внимание на `delay()`. Поменяйте значение задержки. Определите, где она нужна для предотвращения наложения двух действий друг на друга, а где – для видимой глазу скорости работы программы.

## Задание 2: Неблокирующее мигание (с `millis()`)

**Оператор `if`:**

Оператор `if` используется, чтобы проверить условие и выполнить некоторые действия только если это условие правда (истина).

Сначала программа смотрит на условие внутри скобок после слова `if`.

Если условие истинно (например, время прошло или значение равно нужному), то выполняется код в фигурных скобках `{ }`.

Если условие ложно, то эти команды пропускаются и программа идёт дальше.

**Пример**

```
if (currentMillis - previousBlinkTime >= blinkInterval) {  
    previousBlinkTime = currentMillis;  
    ledState = 255 - ledState;  
    rgb(ledState, ledState, ledState);  
}
```



- Здесь проверяется, сколько времени прошло с прошлого мигания.
- Если прошло столько времени, сколько указано в `blinkInterval` (например, 500 мс), то программа обновляет время и меняет состояние светодиода (вкл/выкл).
- Если времени ещё не прошло, код внутри {} не выполняется, и светодиод не меняет состояние.

**Цель:** Научиться использовать `millis()` для управления двумя независимыми процессами.

**Код:**

```
// Переменные для светодиода
int previousBlinkTime = 0;
int blinkInterval = 500; // Интервал мигания (мс)
bool ledState = 0; // Флаг: 255 - светодиод вкл, 0 - выкл

// Переменные для динамика
int previousBeepTime = 0;
int beepInterval = 300; // Интервал звука (мс)
bool beepState = false; // Флаг: true - звук вкл, false - выкл

void loop() {
    int currentMillis = millis(); // Считываем текущее время ОДИН раз за цикл

    // Блок 1: Управление светодиодом
    if (currentMillis - previousBlinkTime >= blinkInterval) {
        previousBlinkTime = currentMillis; // Сохраняем время последнего переключения

        // Инвертируем состояние светодиода
        ledState = 255-ledState;
        rgb(ledState, ledState, ledState);
    }
}
```

```

// Блок 2: Управление динамиком
if (currentMillis - previousBeepTime >= beepInterval) {
    previousBeepTime = currentMillis; // Сохраняем время
последнего переключения

if (beepState) {
    tone(TONE_PIN, 660); // Включаем звук (более высокий)
} else {
    noTone(TONE_PIN); // Выключаем звук
}
beepState = !beepState; // Инвертируем флаг
}

// ЗДЕСЬ МОЖЕТ БЫТЬ ЛЮБОЙ ДРУГОЙ КОД: опрос датчиков, кнопок и т.д.
// Он будет выполняться постоянно, без задержек!
}

```

### Задание 1:

- 1** Загрузите код и наблюдайте, как светодиод и динамик работают с разными интервалами, не мешая друг другу.
- 2** Измените переменные `blinkInterval` и `beepInterval` на разные значения (например, 1000 и 250). Объясните результат.

### Задание 2: Светомузыка (моргание с музыкой)

Примените **неблокирующий** подход для синхронизации света и сложной последовательности нот.

## Контрольные вопросы

1. Объясните своими словами, в чем заключается главный недостаток функции `delay()`.
2. Какой тип данных возвращает функция `millis()` и почему именно он?
3. Что произойдет с переменной `previousMillis`, когда значение `millis()` переполнится и обнулится (примерно через 50 дней)? Будет ли корректно работать ваш код? (Подсказка: благодаря использованию вычитания и беззнаковой арифметики — да!)
4. В Задании 3, что будет, если значительно увеличить `blinkInterval` (например, до 500 мс)? Опишите визуальный эффект.