Федеральное государственное бюджетное образовательное учреждение высшего

образования

«Чувашский государственный университет им. И.Н. Ульянова»

Кафедра вычислительной техники

Лабораторная работа №3

**Управление светодиодами с использованием датчика и конечного автомата на Arduino Nano.**

Выполнил: Иванов В.С.

студент группы ИВТ-41-22

Проверил: Степанов С. В.

Чебоксары, 2025

Цель работы: Изучить работу с аналоговыми датчиками, реализовать конечный автомат для управления светодиодами, а также освоить использование математических функций для обработки данных.

Задание: 1.

Сборка схемы: - Подключите три светодиода к цифровым пинам с поддержкой ШИМ (например, D3, D5, D6) через резисторы 220 Ом.

- Подключите катоды светодиодов (короткие ножки) к земле (GND) на Arduino Nano. - Подключите кнопку к цифровому пину (например, D2). Один контакт кнопки подключите к пину D2, а другой — к земле (GND). Используйте внутренний подтягивающий резистор.

- Подключите потенциометр (или фоторезистор) к аналоговому пину (например, A0). Средний вывод потенциометра подключите к A0, а крайние выводы — к +5V и GND.

Написание программы:

- Напишите программу, которая будет выполнять следующие действия: 1. Используйте потенциометр (или фоторезистор) для управления яркостью одного из светодиодов (например, D3). Яркость должна изменяться плавно в зависимости от положения потенциометра.

2. Реализуйте конечный автомат с тремя состояниями:

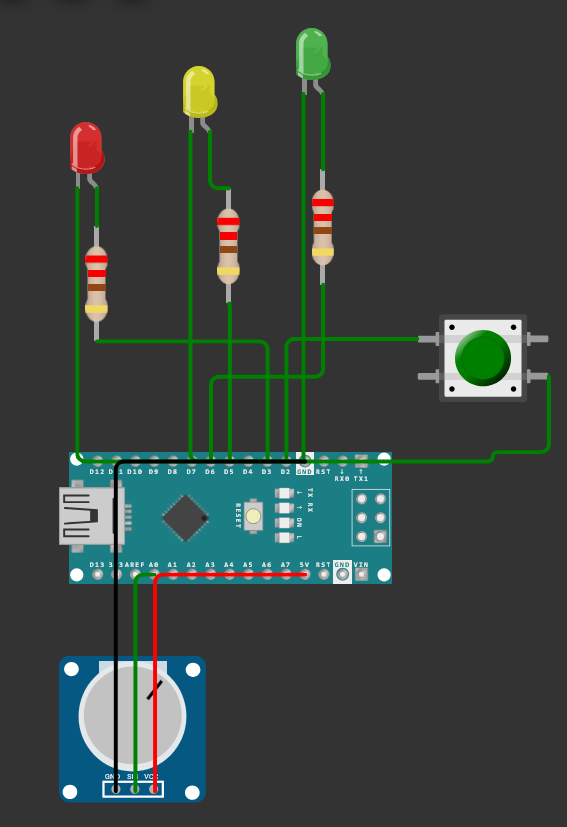
- Состояние 1:Все светодиоды выключены.

- Состояние 2:Светодиоды мигают поочередно с частотой, заданной потенциометром.

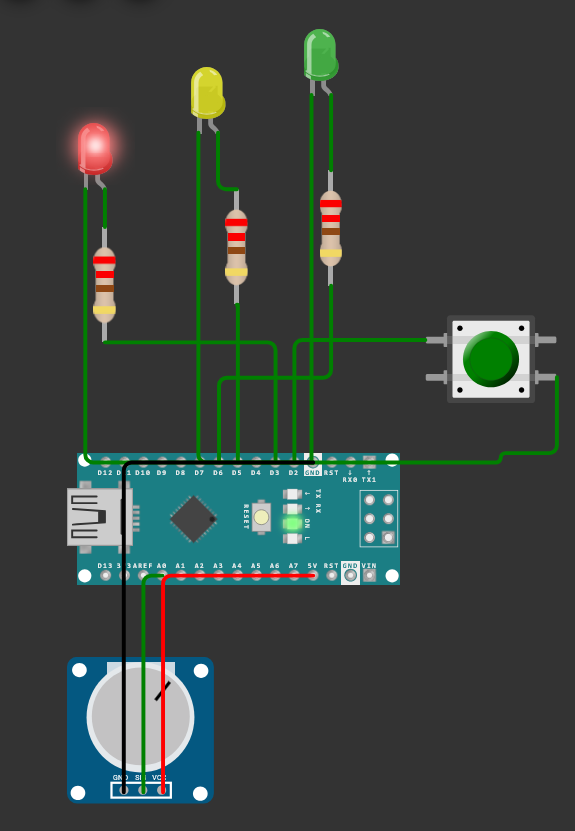
- Состояние 3:Светодиоды плавно изменяют яркость в противофазе (один увеличивает яркость, другой уменьшает).

3. Переключение между состояниями должно происходить по нажатию кнопки. 4. Выводите текущее состояние и значение с потенциометра в последовательный порт.

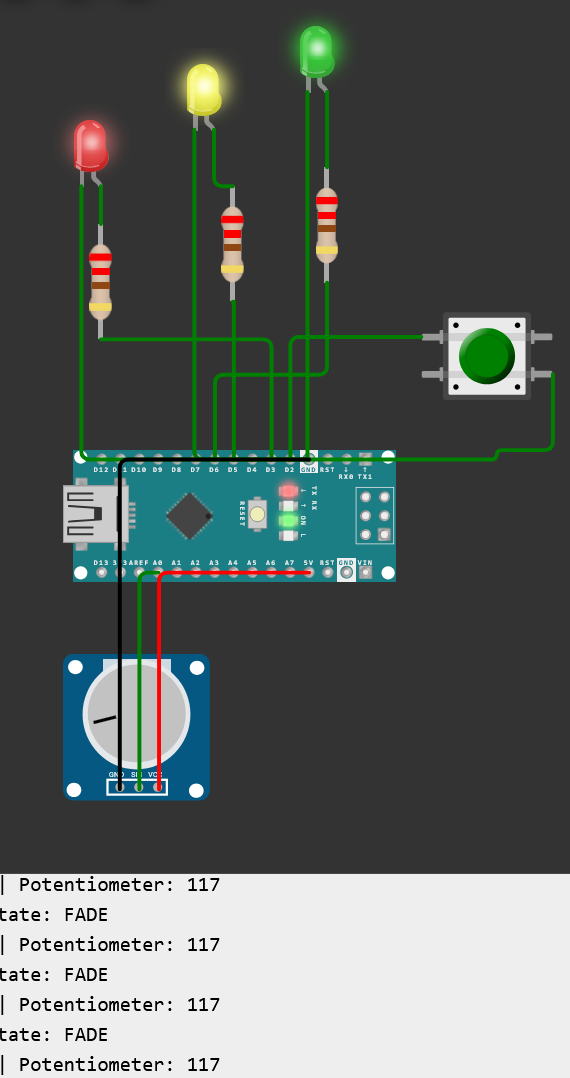
Начальное состояние все выключено 1)



Состояние 2) Моргнаие по очереди светодиодов

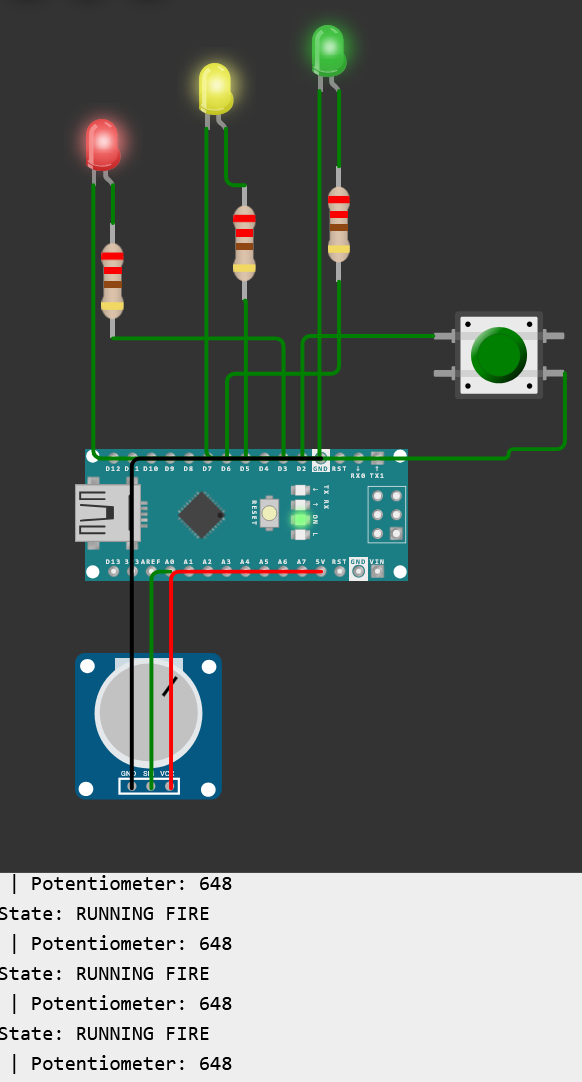


Состояние 3) Светодиоды плавно меняют свою яркость в противофазе:



Состояние 4 (Доп задание))

Бегущий огонек :



Код программы:

#include <EEPROM.h>

// Определение пинов

#define LED1\_PIN 5

#define LED2\_PIN 3

#define LED3\_PIN 6

#define BUTTON\_PIN 2

#define POTENTIOMETER\_PIN A0

// Состояния конечного автомата

enum State {

STATE\_OFF,

STATE\_BLINK,

STATE\_FADE,

STATE\_RUNNING\_FIRE,

NUM\_STATES

};

State currentState = STATE\_OFF;

unsigned long lastStateChangeTime = 0;

int ledIndex = 0;

int fadeValue = 0;

bool fadeDirection = true;

// Переменные для антидребезга

unsigned long lastDebounceTime = 0;

unsigned long debounceDelay = 50; // Задержка антидребезга (мс)

bool lastButtonState = HIGH; // Состояние кнопки при прошлой проверке

bool buttonPressed = false;

void setup() {

pinMode(LED1\_PIN, OUTPUT);

pinMode(LED2\_PIN, OUTPUT);

pinMode(LED3\_PIN, OUTPUT);

pinMode(BUTTON\_PIN, INPUT\_PULLUP);

Serial.begin(9600);

// Восстановление состояния из EEPROM

int savedState = EEPROM.read(0);

if (savedState >= 0 && savedState < NUM\_STATES) {

currentState = (State)savedState;

}

Serial.print("Initial state: ");

printState(currentState);

}

void loop() {

int potValue = analogRead(POTENTIOMETER\_PIN);

int speed = map(potValue, 0, 1023, 50, 1000);

// Обработка кнопки с антидребезгом

bool currentButtonState = digitalRead(BUTTON\_PIN);

// Если состояние кнопки изменилось (нажатие/отпускание)

if (currentButtonState != lastButtonState) {

lastDebounceTime = millis(); // Сброс таймера антидребезга

}

// Если прошло достаточно времени после последнего изменения

if ((millis() - lastDebounceTime) > debounceDelay) {

// Если кнопка нажата (LOW, так как INPUT\_PULLUP) и ранее не была нажата

if (currentButtonState == LOW && !buttonPressed) {

buttonPressed = true;

changeState();

}

// Если кнопка отпущена

else if (currentButtonState == HIGH) {

buttonPressed = false;

}

}

lastButtonState = currentButtonState; // Сохраняем текущее состояние кнопки

// Вывод информации в Serial

static unsigned long lastPrintTime = 0;

if (millis() - lastPrintTime > 500) {

lastPrintTime = millis();

Serial.print("State: ");

printState(currentState);

Serial.print(" | Potentiometer: ");

Serial.println(potValue);

}

// Выполнение текущего состояния

switch (currentState) {

case STATE\_OFF:

allLedsOff();

break;

case STATE\_BLINK:

blinkLeds(speed);

break;

case STATE\_FADE:

fadeLeds(speed);

break;

case STATE\_RUNNING\_FIRE:

runningFire(speed);

break;

}

}

void changeState() {

currentState = (State)((currentState + 1) % NUM\_STATES);

lastStateChangeTime = millis();

EEPROM.write(0, currentState);

Serial.print("State changed to: ");

printState(currentState);

// Добавляем небольшую задержку перед возможным следующим нажатием

delay(200); // 200 мс (можно изменить под свои нужды)

}

void allLedsOff() {

analogWrite(LED1\_PIN, 0);

analogWrite(LED2\_PIN, 0);

analogWrite(LED3\_PIN, 0);

}

void blinkLeds(int speed) {

unsigned long currentTime = millis();

if (currentTime - lastStateChangeTime > speed) {

lastStateChangeTime = currentTime;

allLedsOff();

switch (ledIndex) {

case 0:

analogWrite(LED1\_PIN, 255);

break;

case 1:

analogWrite(LED2\_PIN, 255);

break;

case 2:

analogWrite(LED3\_PIN, 255);

break;

}

ledIndex = (ledIndex + 1) % 3;

}

}

void fadeLeds(int speed) {

unsigned long currentTime = millis();

if (currentTime - lastStateChangeTime > speed / 100) {

lastStateChangeTime = currentTime;

if (fadeDirection) {

fadeValue++;

if (fadeValue >= 255) {

fadeDirection = false;

}

} else {

fadeValue--;

if (fadeValue <= 0) {

fadeDirection = true;

}

}

analogWrite(LED1\_PIN, fadeValue);

analogWrite(LED2\_PIN, 255 - fadeValue);

analogWrite(LED3\_PIN, fadeValue);

}

}

void runningFire(int speed) {

unsigned long currentTime = millis();

if (currentTime - lastStateChangeTime > speed / 10) {

lastStateChangeTime = currentTime;

static int pos = 0;

static int trail[3] = {0, 0, 0};

trail[pos] = 255;

pos = (pos + 1) % 3;

for (int i = 0; i < 3; i++) {

trail[i] = max(0, trail[i] - 85);

}

analogWrite(LED1\_PIN, trail[0]);

analogWrite(LED2\_PIN, trail[1]);

analogWrite(LED3\_PIN, trail[2]);

}

}

void printState(State state) {

switch (state) {

case STATE\_OFF:

Serial.println("OFF");

break;

case STATE\_BLINK:

Serial.println("BLINK");

break;

case STATE\_FADE:

Serial.println("FADE");

break;

case STATE\_RUNNING\_FIRE:

Serial.println("RUNNING FIRE");

break;

}

}

Ответы на вопросы:

**. Как работает конечный автомат, и в каких задачах он может быть полезен?**

**Конечный автомат (Finite State Machine, FSM)** — это модель поведения системы, которая может находиться в **конечном числе состояний** и переключаться между ними в зависимости от **входных сигналов или событий**.

**Как работает FSM?**

* **Состояния (States)** — фиксированные режимы работы (например, STATE\_OFF, STATE\_BLINK).
* **Переходы (Transitions)** — условия, при которых система меняет состояние (например, нажатие кнопки).
* **Действия (Actions)** — операции, выполняемые в каждом состоянии (например, мигание светодиодов).

**Где полезен конечный автомат?**

* **Управление устройствами** (светофоры, роботы, умный дом).
* **Игровая логика** (анимации персонажей, AI врагов).
* **Сетевые протоколы** (автомат TCP-соединения).
* **Пользовательские интерфейсы** (меню, кнопки, диалоги).

Пример из моего кода:

enum State {

STATE\_OFF, // Все светодиоды выключены

STATE\_BLINK, // Поочерёдное мигание

STATE\_FADE, // Плавное изменение яркости

STATE\_RUNNING\_FIRE // Эффект "бегущего огня"

};};

### ****2. Почему важно использовать функцию****map()****для преобразования значений с аналогового датчика?****

Функция map() преобразует значение из одного диапазона в другой.

Пример из моего кода:

int potValue = analogRead(POTENTIOMETER\_PIN); // 0–1023

int speed = map(potValue, 0, 1023, 50, 1000); // 50–1000 мс

#### ****Зачем это нужно?****

1. **Аналоговые датчики** (потенциометр, фоторезистор) возвращают значения **0–1023**, но нам часто нужно другое значение (например, скорость от 50 до 1000 мс).
2. **Нормализация данных** — чтобы удобно управлять параметрами (яркостью, задержкой).
3. **Гибкость** — можно легко изменить диапазон без пересчёта вручную.

**Без** map()**пришлось бы писать:**

int speed = potValue / 1023.0 \* (1000 - 50) + 50; // Сложнее и менее читаемо

### ****3. Как можно улучшить обработку нажатия кнопки (дебаунс)?****

Дребезг (bounce) — это **ложные срабатывания** при нажатии кнопки из-за нестабильного контакта.

#### ****Стандартный дебаунс (как в коде выше):****

unsigned long lastDebounceTime = 0;

unsigned long debounceDelay = 50; // Задержка 50 мс

if (currentButtonState != lastButtonState) {

lastDebounceTime = millis(); // Сброс таймера при изменении состояния

}

if ((millis() - lastDebounceTime) > debounceDelay) {

// Обработка стабильного состояния кнопки

}

#### ****Улучшения:****

1. **Оптимальная задержка (**debounceDelay**)**
   * Обычно **10–50 мс**, но можно подобрать экспериментально.
   * Некоторые кнопки требуют **100 мс**.
2. **Аппаратный дебаунс**
   * Добавить **RC-фильтр** (резистор + конденсатор) для сглаживания сигнала.
3. **Библиотеки**
   * Bounce2 — готовая библиотека для дебаунса.

#include <Bounce2.h>

Bounce debouncer = Bounce();

debouncer.attach(BUTTON\_PIN, INPUT\_PULLUP);

debouncer.interval(25); // 25 мс

if (debouncer.fell()) { // Нажатие без дребезга

changeState();

}

1. **Учет времени между нажатиями**
   * Добавить задержку delay(200)**после смены состояния**, чтобы избежать множественных срабатываний.
2. **Обработка фронтов (**rising**/**falling**)**
   * Ловить не просто состояние кнопки, а **момент нажатия (**falling**)** или **отпускания (**rising**)**.

**Итог:**  
Лучший вариант — **библиотека**Bounce2 или **аппаратный RC-фильтр + программный дебаунс**.