Практическое занятие №
Тема: «Работа с памятью МК»

Цель работы: приобрести практические навыки по программированию записи и чтения данных с EEPROM на платформе Arduino.

Последовательность выполнения работы:

– Собрать схемы на макетной плате, иначе при отсутствии набора Arduino в web-приложениях (https://wokwi.com/projects/new/arduino-uno или https://www.tinkercad.com/) для приведенных примеров.

– Запрограммировать микроконтроллер согласно заданию в примере.

– Выполнить задание для самостоятельной работы.

Содержание отчета:

– Название практического занятия, его цель.

– Фото или скриншоты собранной схемы.

– Написанный программный код вставить текстом, Courier New, 12 кегль, одинарный отступ без абзацев.

– Вывод о проделанной работе.

– Файл Fritzing с принципиальной и монтажной схемой.

ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

EEPROM (Electrically Erasable Programmable Read-Only Memory) — это энергонезависимая память, которая сохраняет данные даже после отключения питания. В отличие от оперативной памяти (RAM), которая очищается при отключении питания, EEPROM хранит информацию годами.

Характеристики EEPROM в Arduino Uno:

Параметр	Значение	Описание
Объем	1 КБ (1024 байта)	Доступно для пользователя
Адреса	0-1023	Каждый байт имеет уникальный адрес
Срок хранения	> 100 лет	При нормальных условиях
Количество циклов записи	~100,000	После чего ячейка может выйти из строя
Скорость записи	~3.3 мс на байт	Медленнее RAM в тысячи раз
Напряжение стирания	Не требуется	Стирается автоматически при записи

Организация памяти в Arduino

Адресное пространство			
Flash (32 КБ)	SRAM (2 КБ)	EEPROM (1 КБ)	Регистры I/O
Программа код	Переменные Heap/ Stack	Сохраненные данные	Периферия

Библиотека EEPROM.h

Arduino предоставляет встроенную библиотеку для работы с EEPROM:
Основные функции:

```
// Чтение одного байта
byte value = EEPROM.read(address);

// Запись одного байта
EEPROM.write(address, value);

// Обновление (только если значение изменилось)
EEPROM.update(address, value);
```

Расширенные функции (шаблонные):

```
// Запись любого типа данных
EEPROM.put(address, data);

// Чтение любого типа данных
EEPROM.get(address, data);
```

Практические примеры

1. Простая запись/чтение

cpp

```
#include <EEPROM.h>

void setup() {
  Serial.begin(9600);

  // Запись
  EEPROM.write(0, 42);          // Запись числа
  EEPROM.write(1, 'A');         // Запись символа
  EEPROM.write(2, 0xFF);        // Запись в HEX
```

```
  // Чтение
  byte num = EEPROM.read(0);    // Прочитать число
  char ch = EEPROM.read(1);     // Прочитать символ
}
```

2. Работа со структурами

cpp

```
struct Settings {
  int mode;
  char name[20];
  float brightness;
};

Settings config = {3, "Default", 0.75f};

// Сохранение всей структуры
EEPROM.put(0, config);

// Чтение структуры
Settings loadedConfig;
EEPROM.get(0, loadedConfig);
```

3. Массивы в EEPROM

cpp

```
// Запись массива
int data[] = {100, 200, 300};
for(int i = 0; i < 3; i++) {
  EEPROM.put(i * sizeof(int), data[i]);
}

// Чтение массива
int readData[3];
for(int i = 0; i < 3; i++) {
  EEPROM.get(i * sizeof(int), readData[i]);
}
```

Важные особенности и ограничения

1. Ограничение циклов записи

```
// НЕПРАВИЛЬНО - быстро исчерпает ресурс EEPROM:
void loop() {
  EEPROM.write(0, analogRead(A0)); // Запись каждую итерацию
цикла!
  delay(100);
}
```

```
// ПРАВИЛЬНО - запись только при изменении:
int lastValue = -1;
void loop() {
  int current = analogRead(A0);
  if(current != lastValue) {
    EEPROM.update(0, current); // Использует update
    lastValue = current;
  }
  delay(100);
}
```

2. Выравнивание данных

```
struct Data {
  byte a;     // Адрес 0
  int b;      // Адрес 2 (пропуск 1 из-за выравнивания!)
  float c;    // Адрес 6
};            // Общий размер: 10 байт

// Используйте pragma pack для точного контроля:
#pragma pack(push, 1)
struct PackedData {
  byte a;     // Адрес 0
  int b;      // Адрес 1
  float c;    // Адрес 5
};            // Общий размер: 9 байт
#pragma pack(pop)
```

3. Проверка целостности данных

```
bool isDataValid(int address) {
  // Метод контрольной суммы
  byte checksum = 0;
  for(int i = address; i < address + 10; i++) {
    checksum ^= EEPROM.read(i); // XOR всех байтов
  }
  return checksum == 0;
}

// Метод сигнатуры
bool hasSignature(int address) {
  return EEPROM.read(address) == 0xAA &&
         EEPROM.read(address + 1) == 0x55;
}
```

ТИПИЧНЫЕ ПРИМЕНЕНИЯ В ПРОЕКТАХ

1. Сохранение последнего состояния

```
// Сохранить состояние при выключении
void saveBeforePowerOff() {
  EEPROM.put(0, currentMode);
  EEPROM.put(10, brightness);
  EEPROM.put(20, colorScheme);
}
```

2. Калибровочные данные датчиков

```
struct Calibration {
  float offset;
  float scale;
  int zeroPoint;
};

Calibration calibData;
EEPROM.get(50, calibData);

float readSensor() {
  int raw = analogRead(A0);
  return (raw - calibData.zeroPoint) * calibData.scale +
calibData.offset;
}
```

3. Счетчики событий

```
unsigned long readEventCount() {
  unsigned long count;
  EEPROM.get(100, count);
  return count;
}

void incrementEventCount() {
  unsigned long count = readEventCount();
  count++;
  EEPROM.put(100, count);
}
```

ОШИБКИ И КАК ИХ ИЗБЕЖАТЬ

Частая запись в одну ячейку

```
// Плохо:
void loop() { EEPROM.write(0, millis()/1000); }

// Хорошо:
void loop() {
  static unsigned long lastSave = 0;
  if(millis() - lastSave > 60000) { // Раз в минуту
    EEPROM.update(0, millis()/60000);
    lastSave = millis();
  }
}
```

Выход за границы памяти

```
// Всегда проверяйте границы:
bool safeWrite(int address, byte data) {
  if(address >= 0 && address < EEPROM.length()) {
    EEPROM.update(address, data);
    return true;
  }
  return false;
}
```

Игнорирование выравнивания данных

```
// Используйте sizeof() для расчета адресов:
int addr1 = 0;
EEPROM.put(addr1, intValue);

int addr2 = addr1 + sizeof(intValue); // Правильный расчет
EEPROM.put(addr2, floatValue);
```

Эмуляция EEPROM во Flash (для больших объемов)

```
// Для плат без EEPROM (ESP8266, ESP32)
#include <FlashAsEEPROM.h>

void setup() {
  EEPROM.begin(512); // Инициализация эмулированной EEPROM
  EEPROM.write(0, 42);
  EEPROM.commit();   // Обязательно для Flash!
}
```

2. Сжатие данных

```
// Хранение нескольких значений в одном байте
byte packedData = 0;

void setLEDState(bool red, bool green, bool yellow) {
  packedData = (red << 2) | (green << 1) | yellow;
  EEPROM.update(0, packedData);
}

void getLEDState() {
  packedData = EEPROM.read(0);
  bool red    = (packedData >> 2) & 1;
  bool green  = (packedData >> 1) & 1;
  bool yellow = packedData & 1;
}
```

3. Шифрование данных

```
byte encrypt(byte data, byte key) {
  return data ^ key; // Простой XOR шифр
}

void saveSecure(int address, byte data, byte key) {
  EEPROM.write(address, encrypt(data, key));
}

byte loadSecure(int address, byte key) {
  return encrypt(EEPROM.read(address), key);
}
```

EEPROM в Arduino — это мощный инструмент для:

- Сохранения настроек между сеансами
- Хранения калибровочных данных
- Ведения логов и счетчиков
- Сохранения состояния системы

При правильном использовании **EEPROM** позволяет создавать устройства с «памятью», которые сохраняют свои настройки и состояние годами, делая проекты на Arduino более профессиональными и удобными в использовании.

ЗАДАНИЯ

Написать программный код:

ИЗУЧИТЬ 3 ПРИМЕРА НИЖЕ ПО РАБОТЕ С ПАМЯТЬЮ EEPROM.

1. Запись и чтение EEPROM

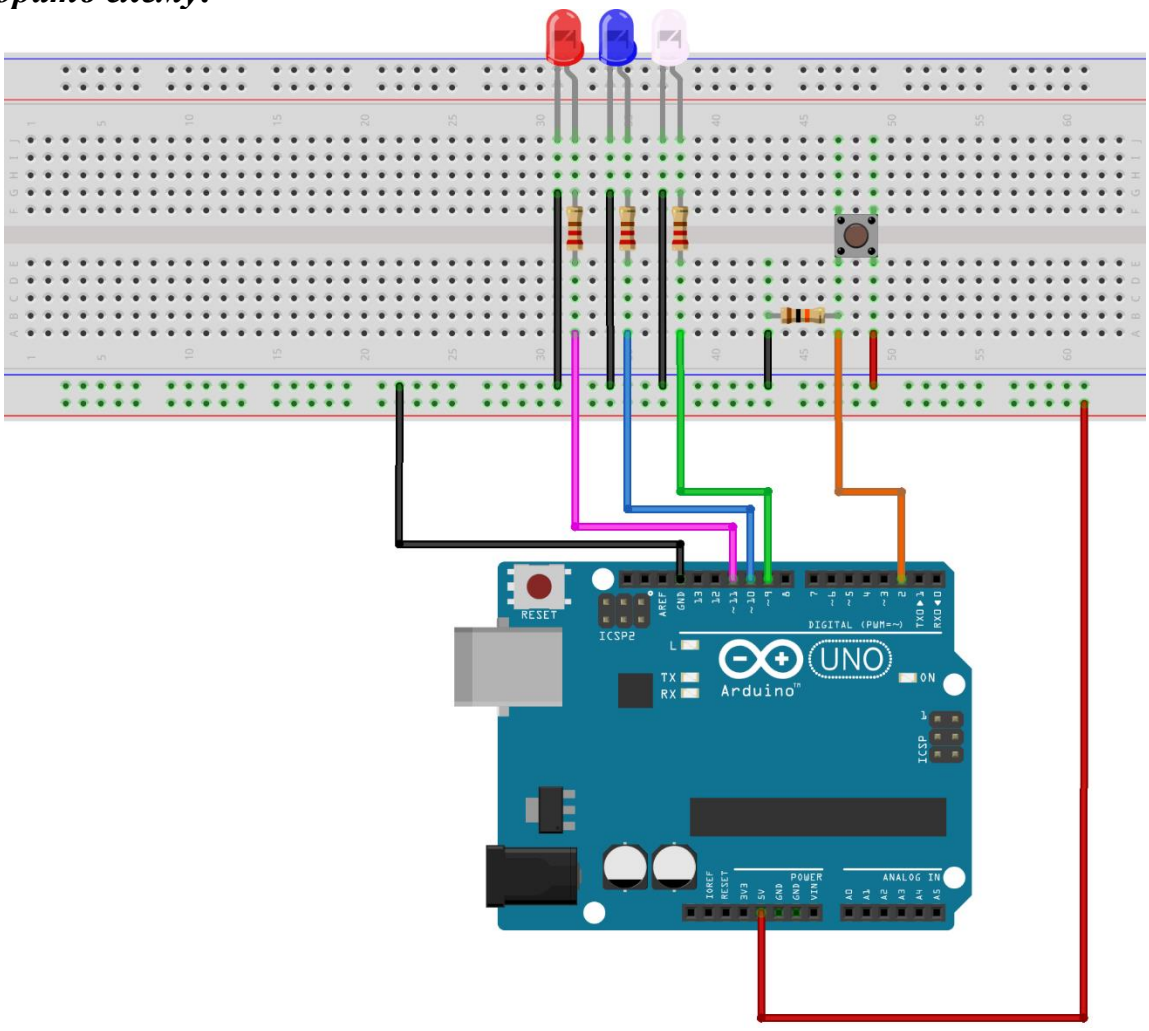
```
#include <EEPROM.h>

void setup() {
  Serial.begin(9600);

  // Запись тестовых данных
  Serial.println("Запись тестовых данных в EEPROM...");
  for (int i = 0; i < 10; i++) {
    EEPROM.write(i, 65 + i); // Буквы A-J
  }

  // Чтение и вывод
  Serial.println("Чтение EEPROM:");
  for (int i = 0; i < 10; i++) {
    Serial.print("Адрес: ");
    Serial.print(i);
    Serial.print(": ");
    Serial.println((char)EEPROM.read(i));
  }
}

void loop() {}
```

2. Очистка EEPROM

```
#include <EEPROM.h>

void setup() {
  Serial.begin(9600);

  Serial.println("Reading EEPROM:");
  for (int i = 0; i < 10; i++) {
    Serial.print("Address ");
    Serial.print(i);
    Serial.print(": ");
    Serial.println((char)EEPROM.read(i));
```

```
  }

  Serial.println("Чистка EEPROM...");
  for (int i = 0; i < 1024; i++) {
    EEPROM.write(i, 0);
  }

  Serial.println("EEPROM очищена!!");

  // Проверка
  Serial.println("Проверка:");
  for (int i = 0; i < 20; i++) {
    Serial.print(EEPROM.read(i));
    Serial.print(" ");
  }
}

void loop() {}
```

Собрать схему:



Написать и ИЗУЧИТЬ программный код:

```
/*
  Система управления 3 светодиодами с 10 эффектами,
сохраняющая состояние
  в энергонезависимую память (EEPROM). При отключении
питания Arduino
  "запоминает" последний выбранный эффект и восстанавливает
его при включении.

  - 10 различных световых эффектов
  - Сохранение номера эффекта в EEPROM
  - Сохранение структуры с номером и названием эффекта
  - Вывод отладочной информации в Serial Monitor
  - Обработка кнопки с антидребезгом
  ============================================================
==================
*/

//
============================================================
===============
// БИБЛИОТЕКИ
//
============================================================
===============

/*
  Подключаем библиотеку для работы с EEPROM
(энергонезависимая память).
  EEPROM - это память, которая сохраняет данные даже при
отключении питания.
  Объем на Arduino Uno: 1024 байта (1 КБ).
  Важно: каждая ячейка памяти выдерживает примерно 100,000
циклов записи.
*/
#include <EEPROM.h>

//
============================================================
===============
// КОНСТАНТЫ И МАКРОСЫ
//
============================================================
===============
```

```
/*
  Определение пинов для подключения компонентов.
  Используем define вместо переменных для экономии
оперативной памяти.
*/
#define LED_RED 9      // Пин для красного светодиода (ШИМ-
пин)
#define LED_GREEN 10   // Пин для зеленого светодиода (ШИМ-
пин)
#define LED_YELLOW 11  // Пин для желтого светодиода (ШИМ-
пин)
#define BUTTON_PIN 2   // Пин для кнопки (пин 2)

/*
  Константы проекта
  MODES_COUNT - количество доступных эффектов (0-9)
  EEPROM_MODE_ADDR - начальный адрес в EEPROM для сохранения
номера режима
  EEPROM_STRUCT_ADDR - адрес для сохранения структуры с
данными
  DEBOUNCE_DELAY - время антидребезга кнопки в миллисекундах
*/
#define MODES_COUNT 10          // 10 режимов (0-9)
#define EEPROM_MODE_ADDR 0      // Адрес для хранения номера
режима
#define EEPROM_STRUCT_ADDR 10   // Адрес для хранения
структуры
#define DEBOUNCE_DELAY 50       // 50 мс для подавления
дребезга контактов

/*
  Структура для хранения расширенных данных о режиме.
  Структуры позволяют группировать связанные данные.
  В EEPROM эта структура займет 17 байт:
  - modeNumber: 1 байт (0-255)
  - modeName: 16 байт (15 символов + нуль-терминатор)
*/
struct ModeData {
  byte modeNumber;    // Номер текущего режима (0-9)
  char modeName[16];  // Название режима (максимум 15
символов + завершающий нуль)
};
```

```
//
================================================================
================
// ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ
//
================================================================
================

/*
  Массив названий режимов.
  Хранится в памяти программ (Flash), а не в оперативной
памяти.
  Ключевое слово 'const' означает, что данные неизменяемы.
*/
const char* modeNames[MODES_COUNT] = {
  "Base Pulse",      // Режим 0: Плавное пульсирование всех
светодиодов
  "Binary Counter",  // Режим 1: Двоичный счетчик на 3
светодиодах (0-7)
  "Running Light",   // Режим 2: Бегущий огонь
  "Police Signal",   // Режим 3: Сигнал полиции (2
светодиода мигают в противофазе)
  "Breathing",       // Режим 4: Эффект "дыхания" (плавное
изменение яркости)
  "Color Mix",       // Режим 5: Плавное смешение цветов
  "Random Twinkle",  // Режим 6: Случайное мерцание
  "Sequence Flash",  // Режим 7: Последовательное включение
  "Slow Fade",       // Режим 8: Медленное затухание
  "Fast Blink"       // Режим 9: Быстрое мигание всеми
светодиодами
};

/*
  Переменные состояния системы.
  Эти переменные хранятся в оперативной памяти (RAM) и
теряются при отключении питания.
*/
int currentMode = 0;                // Текущий выбранный
режим (0-9)
unsigned long lastButtonPress = 0;  // Время последнего
нажатия кнопки (для антидребезга)
bool buttonPressed = false;         // Флаг нажатия кнопки
(true = кнопка нажата)
```

```
//
================================================================
================
// ФУНКЦИЯ SETUP - ВЫПОЛНЯЕТСЯ ОДИН РАЗ ПРИ ЗАПУСКЕ
//
================================================================
================

void setup() {
  /*
    Инициализация пинов микроконтроллера.
    pinMode() настраивает пин как вход (INPUT) или выход
(OUTPUT).
  */
  pinMode(LED_RED, OUTPUT);     // Настраиваем пин красного
светодиода как выход
  pinMode(LED_GREEN, OUTPUT);   // Настраиваем пин зеленого
светодиода как выход
  pinMode(LED_YELLOW, OUTPUT);  // Настраиваем пин желтого
светодиода как выход
  pinMode(BUTTON_PIN, INPUT);   // Настраиваем пин кнопки
как вход

  /*
    Инициализация последовательного порта для отладки.
    9600 - скорость передачи (бодрейт) в битах в секунду.
    Serial Monitor в Arduino IDE должен быть настроен на ту
же скорость.
  */
  Serial.begin(9600);

  /*
    Небольшая задержка для стабилизации системы после
включения.
    Особенно важно для EEPROM и последовательного порта.
  */
  delay(100);

  /*
    Загружаем последний сохраненный режим из EEPROM.
    Эта функция читает номер режима, который был сохранен
перед выключением.
  */
  loadLastMode();
```

```
  /*
    Выводим информационное сообщение в Serial Monitor.
    \n - символ новой строки
    println - выводит строку и переводит курсор на новую
строку
  */
  Serial.println("Система инициализирована.");
  Serial.print("Загружен режим из памяти: ");
  Serial.println(currentMode);

  /*
    Показываем информацию о текущем режиме.
    Эта функция выводит номер и название режима.
  */
  printModeInfo();

  /*
    Дополнительно: загружаем и показываем расширенные данные
из EEPROM.
    Это демонстрирует второй способ хранения данных
(структурой).
  */
  loadModeWithName();
}

//
================================================================
================
// ФУНКЦИЯ LOOP - ВЫПОЛНЯЕТСЯ ПОСТОЯННО В ЦИКЛЕ
//
================================================================
================

void loop() {
  /*
    Главный цикл программы выполняется примерно 16,000 раз в
секунду
    (зависит от сложности операций внутри).

    Порядок выполнения в каждом цикле:
    1. Проверка нажатия кнопки
    2. Воспроизведение текущего светового эффекта
  */
```

```
  // Шаг 1: Обработка нажатия кнопки
  handleButton();

  // Шаг 2: Воспроизведение текущего эффекта
  playEffect(currentMode);

  /*
    Важно: здесь НЕТ delay() в основном цикле!
    Использование delay() остановило бы обработку кнопки.
    Все эффекты используют неблокирующие задержки на основе
millis().
  */
}

//
================================================================
================
// РАЗДЕЛ: ФУНКЦИИ РАБОТЫ С ПАМЯТЬЮ (EEPROM)
//
================================================================
================

/*
  ФУНКЦИЯ: loadLastMode()
  НАЗНАЧЕНИЕ: Загружает последний сохраненный номер режима
из EEPROM
  ПРИНЦИП РАБОТЫ:
    1. Читает байт из ячейки EEPROM по адресу
EEPROM_MODE_ADDR (0)
    2. Проверяет, что значение в допустимом диапазоне (0-9)
    3. Если значение корректно - устанавливает currentMode
    4. Если нет - устанавливает режим 0 и записывает его в
EEPROM
  ПАМЯТЬ: Использует EEPROM.read() - быстрое чтение
(микросекунды)
*/
void loadLastMode() {
  /*
    EEPROM.read(address) - читает один байт из указанного
адреса.
    Адреса от 0 до 1023 на Arduino Uno.
    Данные сохраняются даже после отключения питания.
  */
```

```
  byte savedMode = EEPROM.read(EEPROM_MODE_ADDR);

  /*
    Проверяем, что прочитанное значение является допустимым
номером режима.
    savedMode < MODES_COUNT означает: если savedMode меньше
10 (0-9 допустимы)
  */
  if (savedMode < MODES_COUNT) {
    // Значение корректно, используем его
    currentMode = savedMode;
    Serial.print("[EEPROM] Загружен режим: ");
    Serial.println(currentMode);
  } else {
    /*
      Значение некорректно (например, 255 при первом
запуске).
      Устанавливаем режим по умолчанию (0).
    */
    currentMode = 0;
    /*
      Записываем режим по умолчанию в EEPROM.
      EEPROM.write() записывает один байт по указанному
адресу.
      Время записи: около 3.3 мс.
    */
    EEPROM.write(EEPROM_MODE_ADDR, 0);
    Serial.println("[EEPROM] Ошибка чтения! Установлен режим
по умолчанию (0).");
  }
}

/*
  ФУНКЦИЯ: saveCurrentMode()
  НАЗНАЧЕНИЕ: Сохраняет текущий номер режима в EEPROM
(простой способ)
  ОПТИМИЗАЦИЯ: Использует EEPROM.update() вместо
EEPROM.write()
  ПРЕИМУЩЕСТВО: EEPROM.update() проверяет, изменилось ли
значение,
                и записывает только если нужно. Это экономит
циклы записи EEPROM.
  ЦИКЛЫ ЗАПИСИ: Каждая ячейка EEPROM выдерживает ~100,000
циклов записи.
```

```
*/
void saveCurrentMode() {
  /*
    EEPROM.update() - умная запись:
    1. Сначала читает текущее значение из EEPROM
    2. Сравнивает с новым значением
    3. Записывает только если значения различаются
    Это продлевает срок службы EEPROM.
  */
  EEPROM.update(EEPROM_MODE_ADDR, currentMode);

  // Выводим отладочную информацию
  Serial.print("[EEPROM] Сохранен режим ");
  Serial.print(currentMode);
  Serial.print(" (");
  Serial.print(modeNames[currentMode]);
  Serial.println(")");
}

/*
  ФУНКЦИЯ: saveModeWithName()
  НАЗНАЧЕНИЕ: Сохраняет структуру с номером режима и его названием
  ПРИМЕНЕНИЕ: Демонстрирует работу со структурами в EEPROM
  РАЗМЕР: Структура ModeData занимает 17 байт в EEPROM
  МЕТОД: Использует EEPROM.put() для записи любых типов данных
*/
void saveModeWithName() {
  /*
    Создаем локальную переменную структуры.
    Она будет уничтожена после выхода из функции (хранится в стеке).
  */
  ModeData data;

  // Заполняем поля структуры
  data.modeNumber = currentMode;  // Текущий номер режима

  /*
    Копируем название режима в поле структуры.
    strncpy() безопасно копирует строку с ограничением длины.
    Параметры:
```

```
    1. Куда копируем (data.modeName)
    2. Откуда копируем (modeNames[currentMode])
    3. Сколько символов копируем (максимум 15)
  */
  strncpy(data.modeName, modeNames[currentMode], 15);
  data.modeName[15] = '\0';  // Гарантируем завершение
строки нуль-терминатором

  /*
    EEPROM.put() записывает любые данные в EEPROM.
    Автоматически вычисляет размер данных и записывает
нужное количество байт.
    Адрес EEPROM_STRUCT_ADDR (10) выбран чтобы не
пересекаться с простым сохранением.
  */
  EEPROM.put(EEPROM_STRUCT_ADDR, data);

  Serial.print("[EEPROM] Сохранена структура: ");
  Serial.print(data.modeNumber);
  Serial.print(" - ");
  Serial.println(data.modeName);
}

/*
  ФУНКЦИЯ: loadModeWithName()
  НАЗНАЧЕНИЕ: Загружает и выводит расширенные данные из
EEPROM
  ДЕМОНСТРАЦИЯ: Показывает как читать структуры из EEPROM
  ИСПОЛЬЗОВАНИЕ: В основном для отладки и демонстрации
*/
void loadModeWithName() {
  // Создаем локальную переменную для хранения загруженных
данных
  ModeData data;

  /*
    EEPROM.get() читает данные любого типа из EEPROM.
    Автоматически определяет размер данных по типу
переменной.
  */
  EEPROM.get(EEPROM_STRUCT_ADDR, data);

  /*
    Проверяем корректность загруженных данных.
```

```
    data.modeNumber < MODES_COUNT - проверяем что номер
режима допустим
    data.modeName[0] != '\0' - проверяем что строка не
пустая
    data.modeName[0] != 255 - проверяем что не прочитали
"мусор" (255 = пустая EEPROM)
  */
  if (data.modeNumber < MODES_COUNT && data.modeName[0] !=
'\0' && data.modeName[0] != 255) {

    Serial.print("[EEPROM] Последнее сохранение: ");
    Serial.print("Режим ");
    Serial.print(data.modeNumber);
    Serial.print(" - '");
    Serial.print(data.modeName);
    Serial.println("'");
  } else {
    Serial.println("[EEPROM] Нет сохраненных
структурированных данных.");
  }
}

//
================================================================
================
// РАЗДЕЛ: ОБРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ВВОДА (КНОПКА)
//
================================================================
================

/*
  ФУНКЦИЯ: handleButton()
  НАЗНАЧЕНИЕ: Обрабатывает нажатие кнопки с защитой от
дребезга
  АЛГОРИТМ:
    1. Читает состояние кнопки
    2. Обнаруживает фронт нажатия (переход с LOW на HIGH)
    3. Проверяет время с последнего нажатия (антидребезг)
    4. При валидном нажатии: переключает режим и сохраняет в
EEPROM
  ТЕХНИКА: Неблокирующая обработка (без delay())
*/
void handleButton() {
  /*
```

```
    digitalRead() возвращает состояние пина:
    HIGH (5V) - кнопка нажата (у нас подключение к 5V через кнопку)
    LOW (0V) - кнопка отпущена (подтягивающий резистор 10кОм к GND)
  */
  int buttonState = digitalRead(BUTTON_PIN);

  /*
    Условие обнаружения нажатия:
    1. buttonState == HIGH - кнопка нажата сейчас
    2. !buttonPressed - кнопка не была нажата в предыдущем цикле (обнаруживаем фронт)
    3. (millis() - lastButtonPress) > DEBOUNCE_DELAY - прошло достаточно времени с последнего нажатия

    millis() - возвращает количество миллисекунд с момента старта Arduino
    (переполняется через ~50 дней)
  */
  if (buttonState == HIGH && !buttonPressed && (millis() - lastButtonPress) > DEBOUNCE_DELAY) {

    /*
      Фиксируем факт нажатия:
      - Устанавливаем флаг buttonPressed
      - Запоминаем время нажатия
    */
    buttonPressed = true;
    lastButtonPress = millis();

    /*
      Переключаем режим:
      % MODES_COUNT обеспечивает циклическое переключение (0,1,2,...,9,0,1,...)
      Оператор % (модуль) возвращает остаток от деления
    */
    currentMode = (currentMode + 1) % MODES_COUNT;

    Serial.println("[КНОПКА] Нажатие обнаружено!");

    /*
      Сохраняем новый режим в EEPROM двумя способами:
      1. Простой способ (только номер)
```

```
      2. Расширенный способ (структура с номером и
названием)
    */
    saveCurrentMode();   // Сохраняем простым способом
    saveModeWithName();  // Сохраняем расширенным способом

    // Выводим информацию о новом режиме
    printModeInfo();
  }

  /*
    Обнаружение отпускания кнопки:
    Когда кнопка отпущена (LOW), но флаг buttonPressed
установлен,
    сбрасываем флаг.
    Это позволяет обнаружить следующее нажатие.
  */
  if (buttonState == LOW && buttonPressed) {
    buttonPressed = false;
    // Можно добавить обработку отпускания, если нужно
  }
}

/*
  ФУНКЦИЯ: printModeInfo()
  НАЗНАЧЕНИЕ: Выводит информацию о текущем режиме в Serial
Monitor
  ФОРМАТ ВЫВОДА:
    ----------------------
    Mode: X / 9
    Name: Название режима
    ----------------------
  ИСПОЛЬЗОВАНИЕ: Для отладки и информирования пользователя
*/
void printModeInfo() {
  Serial.println("----------------------");
  Serial.print("Режим: ");
  Serial.print(currentMode);
  Serial.print(" из ");
  Serial.println(MODES_COUNT - 1);
  Serial.print("Название: ");
  Serial.println(modeNames[currentMode]);
  Serial.print("Описание: ");
```

```
  // Добавляем краткое описание каждого режима
  switch (currentMode) {
    case 0: Serial.println("Плавное пульсирование"); break;
    case 1: Serial.println("Двоичный счет 0-7"); break;
    case 2: Serial.println("Бегущий огонь"); break;
    case 3: Serial.println("Сигнал полиции"); break;
    case 4: Serial.println("Эффект дыхания"); break;
    case 5: Serial.println("Смешение цветов"); break;
    case 6: Serial.println("Случайное мерцание"); break;
    case 7: Serial.println("Последовательность"); break;
    case 8: Serial.println("Медленное затухание"); break;
    case 9: Serial.println("Быстрое мигание"); break;
  }
  Serial.println("----------------------");
}

//
================================================================
================
// РАЗДЕЛ: СВЕТОВЫЕ ЭФФЕКТЫ
//
================================================================
================

/*
  ФУНКЦИЯ: playEffect(int mode)
  НАЗНАЧЕНИЕ: Вызывает соответствующую функцию эффекта по
номеру режима
  ПАРАМЕТР: mode - номер режима (0-9)
  ПРИНЦИП: Использует оператор switch для выбора нужного
эффекта
  ВАЖНО: Все функции эффектов неблокирующие (не используют
delay())
*/
void playEffect(int mode) {
  /*
    Оператор switch - эффективная альтернатива множественным
if-else
    при работе с целыми числами и перечислениями.
  */
  switch (mode) {
    case 0:           // Режим 0: Плавное пульсирование
      pulseAll(500);  // Параметр: длительность полного
цикла в мс
```

```
      break;

    case 1:                 // Режим 1: Двоичный счетчик
      binaryCounter(300);  // Параметр: время между счетами
в мс
      break;

    case 2:                // Режим 2: Бегущий огонь
      runningLight(150);  // Параметр: скорость переключения
в мс
      break;

    case 3:                // Режим 3: Сигнал полиции
      policeSignal(200);  // Параметр: частота мигания в мс
      break;

    case 4:            // Режим 4: Эффект дыхания
      breathing(20);  // Параметр: задержка между шагами в
мс
      break;

    case 5:            // Режим 5: Смешение цветов
      colorMix(400);  // Параметр: длительность полного
цикла в мс
      break;

    case 6:                 // Режим 6: Случайное мерцание
      randomTwinkle(100);  // Параметр: время между
изменениями в мс
      break;

    case 7:                 // Режим 7: Последовательное
включение
      sequenceFlash(250);  // Параметр: время между шагами в
мс
      break;

    case 8:           // Режим 8: Медленное затухание
      slowFade(30);  // Параметр: задержка между шагами в мс
      break;

    case 9:             // Режим 9: Быстрое мигание
      fastBlink(100);  // Параметр: период мигания в мс
      break;
```

```
    default:  // Запасной вариант (не должен выполняться)
      // Все светодиоды выключены
      digitalWrite(LED_RED, LOW);
      digitalWrite(LED_GREEN, LOW);
      digitalWrite(LED_YELLOW, LOW);
      break;
  }
}

//
================================================================
================
// РЕАЛИЗАЦИЯ КОНКРЕТНЫХ ЭФФЕКТОВ
//
================================================================
================

/*
  ЭФФЕКТ 0: pulseAll(int duration)
  ОПИСАНИЕ: Все светодиоды плавно пульсируют с одинаковой
яркостью
  ПАРАМЕТР: duration - время полного цикла (от минимума до
максимума и обратно)
  ПРИНЦИП: Синусоидальное изменение яркости всех светодиодов
  ШИМ: Использует analogWrite() для плавного управления
яркостью (0-255)
*/
void pulseAll(int duration) {
  /*
    Статические переменные сохраняют свои значения между
вызовами функции.
    Это необходимо для анимации без использования глобальных
переменных.
  */
  static unsigned long lastTime = 0;  // Время последнего
обновления
  static int brightness = 0;          // Текущая яркость (0-
255)
  static bool rising = true;          // Направление
изменения (true = увеличиваем)

  /*
    Неблокирующая задержка:
```

```
    Вычисляем, прошло ли достаточно времени с последнего
обновления.
    duration / 256 - время между изменениями яркости на 1
единицу.
    Это обеспечивает плавность анимации.
  */
  if (millis() - lastTime > duration / 256) {
    lastTime = millis();  // Обновляем время последнего
изменения

    /*
      Устанавливаем одинаковую яркость для всех светодиодов.
      analogWrite(pin, value) генерирует ШИМ-сигнал:
      - value = 0: всегда выключено
      - value = 255: всегда включено
      - 1-254: различная скважность (яркость)
    */
    analogWrite(LED_RED, brightness);
    analogWrite(LED_GREEN, brightness);
    analogWrite(LED_YELLOW, brightness);

    /*
      Изменяем яркость в зависимости от направления.
      rising = true: увеличиваем яркость до 255
      rising = false: уменьшаем яркость до 0
    */
    if (rising) {
      brightness++;              // Увеличиваем яркость
      if (brightness >= 255) {   // Достигли максимума
        rising = false;          // Меняем направление
      }
    } else {
      brightness--;            // Уменьшаем яркость
      if (brightness <= 0) {   // Достигли минимума
        rising = true;         // Меняем направление
      }
    }
  }
}

/*
  ЭФФЕКТ 1: binaryCounter(int delayTime)
  ОПИСАНИЕ: Светодиоды показывают двоичный счет от 0 до 7
```

```
  ПРИНЦИП: Каждый светодиод - один бит (красный = бит 0,
зеленый = бит 1, желтый = бит 2)
  ВИЗУАЛИЗАЦИЯ: 000, 001, 010, 011, 100, 101, 110, 111, 000,
...
  ФУНКЦИИ: bitRead() - читает указанный бит числа
*/
void binaryCounter(int delayTime) {
  static int count = 0;               // Текущее значение
счетчика (0-7)
  static unsigned long lastTime = 0;  // Время последнего
обновления

  if (millis() - lastTime > delayTime) {
    lastTime = millis();

    /*
      Устанавливаем состояние светодиодов в соответствии с
битами числа.
      bitRead(number, bit) возвращает значение указанного
бита (0 или 1).
      Бит 0 (младший) -> красный светодиод
      Бит 1 -> зеленый светодиод
      Бит 2 (старший) -> желтый светодиод
    */
    digitalWrite(LED_RED, bitRead(count, 0));     // Бит 0
    digitalWrite(LED_GREEN, bitRead(count, 1));   // Бит 1
    digitalWrite(LED_YELLOW, bitRead(count, 2));  // Бит 2

    // Увеличиваем счетчик по модулю 8 (0-7)
    count = (count + 1) % 8;
  }
}

/*
  ЭФФЕКТ 2: runningLight(int speed)
  ОПИСАНИЕ: Бегущий огонь - светодиоды поочередно включаются
  ПОСЛЕДОВАТЕЛЬНОСТЬ: Красный -> Зеленый -> Желтый ->
Красный -> ...
  ПАРАМЕТР: speed - время горения каждого светодиода
*/
void runningLight(int speed) {
  static int pos = 0;  // Текущая позиция (0=красный,
1=зеленый, 2=желтый)
  static unsigned long lastTime = 0;
```

```
  if (millis() - lastTime > speed) {
    lastTime = millis();

    /*
      Включаем только светодиод на текущей позиции.
      (pos == 0) возвращает true если pos равен 0, false в
других случаях.
      digitalWrite принимает 0 (LOW) или 1 (HIGH), true
преобразуется в 1.
    */
    digitalWrite(LED_RED, (pos == 0));
    digitalWrite(LED_GREEN, (pos == 1));
    digitalWrite(LED_YELLOW, (pos == 2));

    // Переходим к следующей позиции (0->1->2->0->...)
    pos = (pos + 1) % 3;
  }
}

/*
  ЭФФЕКТ 3: policeSignal(int speed)
  ОПИСАНИЕ: Имитация сигнала полиции (мигание красным и
желтым в противофазе)
  ЗЕЛЕНЫЙ: Не горит (можно было бы использовать как синий)
  ПАРАМЕТР: speed - частота мигания
*/
void policeSignal(int speed) {
  static bool state = false;  // Текущее состояние
(true/false)
  static unsigned long lastTime = 0;

  if (millis() - lastTime > speed) {
    lastTime = millis();
    state = !state;  // Инвертируем состояние

    /*
      Красный и желтый мигают в противофазе:
      - Когда state = true: красный горит, желтый не горит
      - Когда state = false: красный не горит, желтый горит
      Зеленый всегда выключен.
    */
    digitalWrite(LED_RED, state);      // Красный: состояние
```

```
    digitalWrite(LED_GREEN, false);    // Зеленый: всегда
выключен
    digitalWrite(LED_YELLOW, !state);  // Желтый: обратное
состояние
  }
}

/*
  ЭФФЕКТ 4: breathing(int stepDelay)
  ОПИСАНИЕ: Эффект "дыхания" - плавное изменение яркости по
синусоиде
  ОСОБЕННОСТЬ: Каждый светодиод имеет свою фазу
  МАТЕМАТИКА: Использует функцию sin() для плавных переходов
*/
void breathing(int stepDelay) {
  static int breath = 0;  // Угол для синуса (0-359
градусов)
  static unsigned long lastTime = 0;

  if (millis() - lastTime > stepDelay) {
    lastTime = millis();

    /*
      Вычисляем яркость по синусоиде:
      sin() возвращает значения от -1 до 1
      Умножаем на 127.5 и прибавляем 127.5 чтобы получить 0-
255
      Градусы переводим в радианы: градусы * PI / 180
    */
    float angle = breath * 3.14159 / 180.0;  // Градусы ->
радианы
    int val = sin(angle) * 127.5 + 127.5;    // -1..1 ->
0..255

    /*
      Устанавливаем разные фазы для светодиодов:
      - Красный: обычная синусоида
      - Зеленый: инвертированная синусоида (255 - val)
      - Желтый: "треугольная" форма (abs(128 - val) * 2)
    */
    analogWrite(LED_RED, val);                    //
Красный: синус
    analogWrite(LED_GREEN, 255 - val);            //
Зеленый: инверсный синус
```

```
    analogWrite(LED_YELLOW, abs(128 - val) * 2);  // Желтый: треугольник

    // Увеличиваем угол (0-359)
    breath = (breath + 1) % 360;
  }
}

/*
  ЭФФЕКТ 5: colorMix(int cycleTime)
  ОПИСАНИЕ: Плавное смешение цветов (как RGB-светодиод)
  ОСОБЕННОСТЬ: Каждый светодиод имеет свой цветовой канал
  МАТЕМАТИКА: Три синусоиды со сдвигом фазы 120 градусов
*/
void colorMix(int cycleTime) {
  /*
    Вычисляем положение в цикле (0.0 - 1.0).
    millis() % cycleTime - остаток от деления текущего времени на длительность цикла
    / (float)cycleTime - преобразуем к диапазону 0.0-1.0
    float - явное указание типа для точного деления
  */
  float t = (millis() % cycleTime) / (float)cycleTime;

  /*
    Вычисляем значения для каждого канала с фазовым сдвигом:
    - Красный: фаза 0°
    - Зеленый: фаза 120° (0.333 от цикла)
    - Желтый: фаза 240° (0.666 от цикла)

    Формула: sin(2π * (t + phase)) * 127.5 + 127.5
    2π * t - угол в радианах для полного цикла
  */
  int r = sin(t * 2 * 3.14159) * 127.5 + 127.5;            // Красный
  int g = sin((t + 0.333) * 2 * 3.14159) * 127.5 + 127.5;  // Зеленый (+120°)
  int b = sin((t + 0.666) * 2 * 3.14159) * 127.5 + 127.5;  // Желтый (+240°)

  // Устанавливаем вычисленные яркости
  analogWrite(LED_RED, r);
  analogWrite(LED_GREEN, g);
  analogWrite(LED_YELLOW, b);
```

```
}

/*
  ЭФФЕКТ 6: randomTwinkle(int delayTime)
  ОПИСАНИЕ: Случайное мерцание светодиодов
  ГЕНЕРАЦИЯ: random(2) возвращает 0 или 1 случайно
  ОСОБЕННОСТЬ: Каждый светодиод независим
*/
void randomTwinkle(int delayTime) {
  static unsigned long lastTime = 0;

  if (millis() - lastTime > delayTime) {
    lastTime = millis();

    /*
      random(2) генерирует случайное число 0 или 1.
      Каждый светодиод получает независимое случайное
состояние.
      digitalWrite интерпретирует 0 как LOW, 1 как HIGH.
    */
    digitalWrite(LED_RED, random(2));     // Случайно 0 или
1
    digitalWrite(LED_GREEN, random(2));   // Случайно 0 или
1
    digitalWrite(LED_YELLOW, random(2));  // Случайно 0 или
1
  }
}

/*
  ЭФФЕКТ 7: sequenceFlash(int delayTime)
  ОПИСАНИЕ: Последовательное включение светодиодов с паузой
  ПОСЛЕДОВАТЕЛЬНОСТЬ: Красный -> Зеленый -> Желтый -> Пауза
-> ...
  ОТЛИЧИЕ ОТ runningLight: есть пауза когда все выключены
*/
void sequenceFlash(int delayTime) {
  static int step = 0;  // Текущий шаг (0-3)
  static unsigned long lastTime = 0;

  if (millis() - lastTime > delayTime) {
    lastTime = millis();

    /*
```

```
      switch для последовательности:
      0: горит только красный
      1: горит только зеленый
      2: горит только желтый
      3: все выключены (пауза)
    */
    switch (step % 4) {
      case 0:
        digitalWrite(LED_RED, HIGH);
        digitalWrite(LED_GREEN, LOW);
        digitalWrite(LED_YELLOW, LOW);
        break;
      case 1:
        digitalWrite(LED_RED, LOW);
        digitalWrite(LED_GREEN, HIGH);
        digitalWrite(LED_YELLOW, LOW);
        break;
      case 2:
        digitalWrite(LED_RED, LOW);
        digitalWrite(LED_GREEN, LOW);
        digitalWrite(LED_YELLOW, HIGH);
        break;
      case 3:
        digitalWrite(LED_RED, LOW);
        digitalWrite(LED_GREEN, LOW);
        digitalWrite(LED_YELLOW, LOW);
        break;
    }

    step++;  // Переход к следующему шагу
  }
}

/*
  ЭФФЕКТ 8: slowFade(int stepDelay)
  ОПИСАНИЕ: Медленное плавное затухание светодиодов со
сдвигом
  ПРИНЦИП: Каждый светодиод имеет свою фазу затухания
  ДИАПАЗОН: Яркость меняется от 0 до 255 циклически
*/
void slowFade(int stepDelay) {
  static int fadeVal = 0;  // Базовое значение яркости (0-
255)
  static unsigned long lastTime = 0;
```

```
  if (millis() - lastTime > stepDelay) {
    lastTime = millis();

    /*
      Устанавливаем яркости со сдвигом:
      - Красный: fadeVal (0-255)
      - Зеленый: fadeVal + 85 (сдвиг на 1/3 цикла)
      - Желтый: fadeVal + 170 (сдвиг на 2/3 цикла)
      % 256 обеспечивает циклическое изменение (255+1=0)
    */
    analogWrite(LED_RED, fadeVal);
    analogWrite(LED_GREEN, (fadeVal + 85) % 256);    // +85
≈ +1/3 от 256
    analogWrite(LED_YELLOW, (fadeVal + 170) % 256);  // +170
≈ +2/3 от 256

    // Увеличиваем базовое значение (0-255)
    fadeVal = (fadeVal + 1) % 256;
  }
}

/*
  ЭФФЕКТ 9: fastBlink(int delayTime)
  ОПИСАНИЕ: Быстрое синхронное мигание всеми светодиодами
  ПРОСТОТА: Все светодиоды включаются и выключаются
одновременно
  ПАРАМЕТР: delayTime - период мигания (включено+выключено)
*/
void fastBlink(int delayTime) {
  static bool state = false;  // Текущее состояние
(вкл/выкл)
  static unsigned long lastTime = 0;

  if (millis() - lastTime > delayTime) {
    lastTime = millis();
    state = !state;  // Инвертируем состояние

    // Все светодиоды в одном состоянии
    digitalWrite(LED_RED, state);
    digitalWrite(LED_GREEN, state);
    digitalWrite(LED_YELLOW, state);
  }
}
```

```
//
================================================================
================
// КОНЕЦ ПРОГРАММЫ
//
================================================================
================
```