СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc167546118)

[ОСНОВНАЯ ЧАСТЬ 4](#_Toc167546119)

[Теоретическая часть 4](#_Toc167546120)

[Что такое Arduino 4](#_Toc167546121)

[Работа с пинами 5](#_Toc167546122)

[Регистры и порты 6](#_Toc167546123)

[Оптимизация памяти 7](#_Toc167546124)

[Методы Arduino 8](#_Toc167546125)

[Работы с экраном 9](#_Toc167546126)

[Практическая часть 10](#_Toc167546127)

[ЗАКЛЮЧЕНИЕ 14](#_Toc167546128)

[СПИСОК ЛИТЕРАТУРЫ 16](#_Toc167546129)

# ВВЕДЕНИЕ

Целью данной курсовой работы является разработка библиотеки для управления светодиодным дисплеем 8x8 на базе микроконтроллера Arduino Nano. В современных условиях, когда микроконтроллеры используются во множестве устройств, от простых гаджетов до сложных систем, умение эффективно работать с ними является важным навыком. Arduino, благодаря своей простоте и широким возможностям, предоставляет отличную платформу для разработки и обучения.

Основные задачи работы включают изучение принципов работы светодиодных дисплеев и микроконтроллеров, создание оптимального кода для управления светодиодами, а также реализацию функционала для отрисовки различных графических элементов. Особое внимание будет уделено использованию регистров и портов микроконтроллера для достижения высокой производительности и экономии памяти, что особенно важно в условиях ограниченных ресурсов.

В рамках теоретической части будут рассмотрены основные понятия и принципы работы с Arduino, включая структуру программ, методы работы с пинами и регистрами, а также способы оптимизации кода. Практическая часть будет включать разработку и тестирование библиотеки, написание методов для отрисовки точек, линий, изображений и примитивных фигур на светодиодном дисплее, а также оптимизацию этих методов для улучшения производительности.

Результаты демонстрируют, что благодаря использованию регистров и портов, а также правильной организации кода, удалось существенно повысить эффективность работы микроконтроллера. Оптимизация кода, включая использование PROGMEM для хранения данных во флеш-памяти и битовых операций для управления пинами, позволила снизить нагрузку на микроконтроллер и улучшить производительность системы.

# ОСНОВНАЯ ЧАСТЬ

## Теоретическая часть

### Что такое Arduino

Arduino – это платформа для создания электронных устройств, которая состоит из аппаратных и программных компонент. Она сильно упрощает, разработку интерактивных проектов, таких как роботы, автоматизированные системы и другие устройства, взаимодействующие с окружающей средой.

Аппаратная часть включает платы Arduino, которые представляют собой печатные платы с установленными на них микроконтроллерами. Эти микроконтроллеры будут программироваться для выполнения различных задач. В младших моделях плат Arduino, таких как UNO, Nano, Mega и Leonardo, используются микроконтроллеры AVR. В более современных и мощных моделях применяются микроконтроллеры ARM Cortex, предназначенные для более серьезных проектов. Микроконтроллер представляет собой интегральную схему, которая является сердцем любой платы Arduino, выполняя команды программы и управляя периферийными устройствами. Его основными компонентами являются процессор, центральный блок, выполняющий команды программы, а также память, включающая Flash-память для хранения программы (32 КБ на Arduino Uno и Nano), SRAM для временного хранения переменных (2 КБ на Arduino Uno и Nano) и EEPROM для постоянного хранения данных после выключения питания (1 КБ на Arduino Uno и Nano).

Программное обеспечение включает Arduino IDE, среду разработки, используемую для написания, компиляции и загрузки кода на платы Arduino, а также библиотеки, которые представляют собой наборы готовых функций для работы с различными компонентами и модулями, упрощающие разработку проектов.

### Работа с пинами

Пины (pins) — это выводы на плате Arduino, используемые для подключения и управления различными компонентами, такими как датчики, светодиоды, моторы и другие периферийные устройства. Каждый пин имеет определенную функциональность и может быть настроен для выполнения различных задач.

Цифровые пины могут быть настроены как входы (input) или выходы (output) и используются для работы с цифровыми сигналами, которые имеют два состояния: HIGH (высокий уровень, обычно 5В или 3.3В) и LOW (низкий уровень, 0В). На большинстве плат Arduino, таких как Arduino Uno, имеется 14 цифровых пинов (D0 - D13). Некоторые цифровые пины могут генерировать ШИМ-сигналы для создания аналогоподобных сигналов, и на Arduino Uno это D3, D5, D6, D9, D10 и D11.

Аналоговые пины используются для чтения аналоговых сигналов, которые могут принимать любое значение в диапазоне от 0 до 5В (или 0 до 3.3В на некоторых платах). Эти сигналы преобразуются в цифровые значения от 0 до 1023 с помощью встроенного аналого-цифрового преобразователя (АЦП). На большинстве плат Arduino, таких как Arduino Uno, имеется 6 аналоговых входов (A0 - A5), которые используются для измерения напряжения и взаимодействия с аналоговыми датчиками, такими как потенциометры, термометры и фотоэлементы.

Настройка пинов в Arduino осуществляется с помощью функции pinMode(pin, mode), которая используется для задания режима работы пина как входа или выхода. Параметр pin указывает номер пина, который нужно настроить, а параметр mode определяет режим его работы. Возможные значения mode включают: INPUT для настройки пина как входа, OUTPUT для настройки пина как выхода и INPUT\_PULLUP для настройки пина как входа с внутренним подтягивающим резистором.

Для чтения значений с пинов используются функции digitalRead(pin) и analogRead(pin). Функция digitalRead(pin) считывает цифровое значение с пина, настроенного как вход, возвращая HIGH или LOW. Функция analogRead(pin) используется для считывания аналогового значения с аналогового пина, возвращая значение от 0 до 1023.

Запись значений на пины осуществляется с помощью функций digitalWrite(pin, value) и analogWrite(pin, value). Функция digitalWrite(pin, value) устанавливает цифровое значение на пине, настроенном как выход, где параметр pin указывает номер пина, а параметр value — значение, которое нужно установить (HIGH или LOW). Функция analogWrite(pin, value) устанавливает ШИМ-сигнал на пине, поддерживающем ШИМ. Параметр value задает значение ШИМ-сигнала от 0 до 255.

Когда пин настроен как вход, его состояние может быть неопределенным, если к нему ничего не подключено. В таких случаях рекомендуется использовать внутренний подтягивающий резистор для установки стабильного состояния. Настройка режима INPUT\_PULLUP включает внутренний подтягивающий резистор, чтобы избежать неопределенного состояния входа.

### Регистры и порты

Регистры DDR, PORT и PIN играют важную роль в управлении пинами на микроконтроллерах, таких как те, что используются в платах Arduino. Регистры направления данных (DDR) определяют, будет ли каждый пин порта настроен как вход или выход. Если DDR = 1, пин настроен как выход, если DDR = 0, пин настроен как вход. Регистры данных (PORT) используются для управления состоянием пинов, настроенных как выходы, или для включения/выключения внутренних подтягивающих резисторов на пинах, настроенных как входы. Когда PORT = 1, устанавливается высокий уровень (HIGH) на выходном пине или включается подтягивающий резистор на входном пине; если PORT = 0, устанавливается низкий уровень (LOW) на выходном пине или отключается подтягивающий резистор на входном пине. Регистры входных пинов (PIN) используются для чтения текущего состояния пинов, настроенных как входы: если PIN = 1, пин находится в высоком состоянии (HIGH), если PIN = 0, пин находится в низком состоянии (LOW).

Микроконтроллеры содержат группы пинов, называемые портами, каждый из которых обозначается буквой (B, C, D и т.д.) и содержит 8 пинов. Порт B включает в себя цифровые пины 8-13 на плате Arduino Uno, и соответствующие регистры управляют этими пинами. Порт C включает в себя аналоговые пины A0-A5 на плате Arduino Uno, и соответствующие регистры управляют этими пинами. Порт D включает в себя цифровые пины 0-7 на плате Arduino Uno, и соответствующие регистры управляют этими пинами.

Регистры DDR, PORT и PIN обеспечивают возможность задавать направление, устанавливать состояние и считывать состояние пинов, соответственно. Работа с регистрами позволяет быстрее и более эффективно управлять состоянием пинов по сравнению с высокоуровневыми функциями Arduino, такими как digitalWrite и digitalRead. Кроме того, использование портов позволяет управлять группами пинов одновременно, что полезно в задачах, требующих синхронного управления несколькими пинами. Прямое управление регистрами также предоставляет больше возможностей для выполнения сложных и нестандартных операций с пинами, что делает эту методику гибкой и мощной для более сложных проектов.

### Оптимизация памяти

При правильном использовании регистров и портов микроконтроллера достигается оптимизация памяти, что играет ключевую роль в эффективной работе устройств на Arduino. Одним из важных методов оптимизации является использование PROGMEM для хранения больших массивов данных во флеш-памяти, что существенно снижает нагрузку на оперативную память (RAM) и повышает производительность устройства.

PROGMEM — это макрос в Arduino, который позволяет сохранять данные во флэш-памяти микроконтроллера, а не в оперативной памяти (SRAM). Это важно, потому что оперативная память в микроконтроллерах обычно очень ограничена, и использование флэш-памяти помогает избежать переполнения SRAM.

Флэш-память, в отличие от SRAM, доступна в гораздо большем объеме, и ее использование для хранения неизменяемых данных, таких как массивы констант, строки или таблицы, позволяет освободить оперативную память для других нужд. Это особенно полезно для хранения больших объемов данных, которые не изменяются в процессе работы программы.

Чтобы использовать PROGMEM, данные объявляются с этим макросом, что указывает компилятору разместить их во флэш-памяти. Для доступа к этим данным используются специальные функции, такие как pgm\_read\_byte, pgm\_read\_word и другие, которые позволяют читать данные непосредственно из флэш-памяти.

Таким образом, PROGMEM помогает эффективно использовать ресурсы памяти микроконтроллера, позволяя хранить большие объемы данных во флэш-памяти и освобождая оперативную память для других задач.

### Методы Arduino

В Arduino, методы setup() и loop() представляют собой основные элементы программной структуры, используемой для написания скетчей (программ) на этой платформе. Метод setup() используется для инициализации переменных, настройки пинов ввода/вывода, соединений и других необходимых начальных настроек. Он вызывается один раз при запуске программы Arduino, после чего управление передается методу loop(). Метод loop() является основным циклом выполнения программы, в котором все инструкции выполняются в бесконечном цикле после завершения метода setup(). По умолчанию Arduino выполняет loop() бесконечно, пока устройство не будет отключено или перезагружено.

Функция delay() приостанавливает выполнение программы на указанное количество миллисекунд, что полезно для создания временных задержек между выполнением различных действий, например, управления частотой мигания светодиода или задержки между чтением датчика и выполнением других операций. Функция millis() возвращает количество миллисекунд, прошедших с момента запуска программы Arduino, и полезна для создания нескольких задержек в программе без блокировки выполнения других операций, в отличие от функции delay().

### Работы с экраном

Светодиодный экран состоит из некоторого количества светодиодов, организованных в матрицу или сетку. Каждый светодиод может светиться независимо от других и может быть включен или выключен по желанию. Для управления каждым светодиодом на экране используется двумерная матрица, где каждый светодиод расположен в пересечении строки и столбца.

Экран подключается к микроконтроллеру, например, к Arduino, через специальные контроллеры, которые обеспечивают необходимые уровни напряжения и тока для каждого светодиода. Обычно на экране есть две группы проводов или контактов: одна группа для управления строками (включение/выключение ряда светодиодов), другая - для управления столбцами (задание состояния светодиодов в столбце).

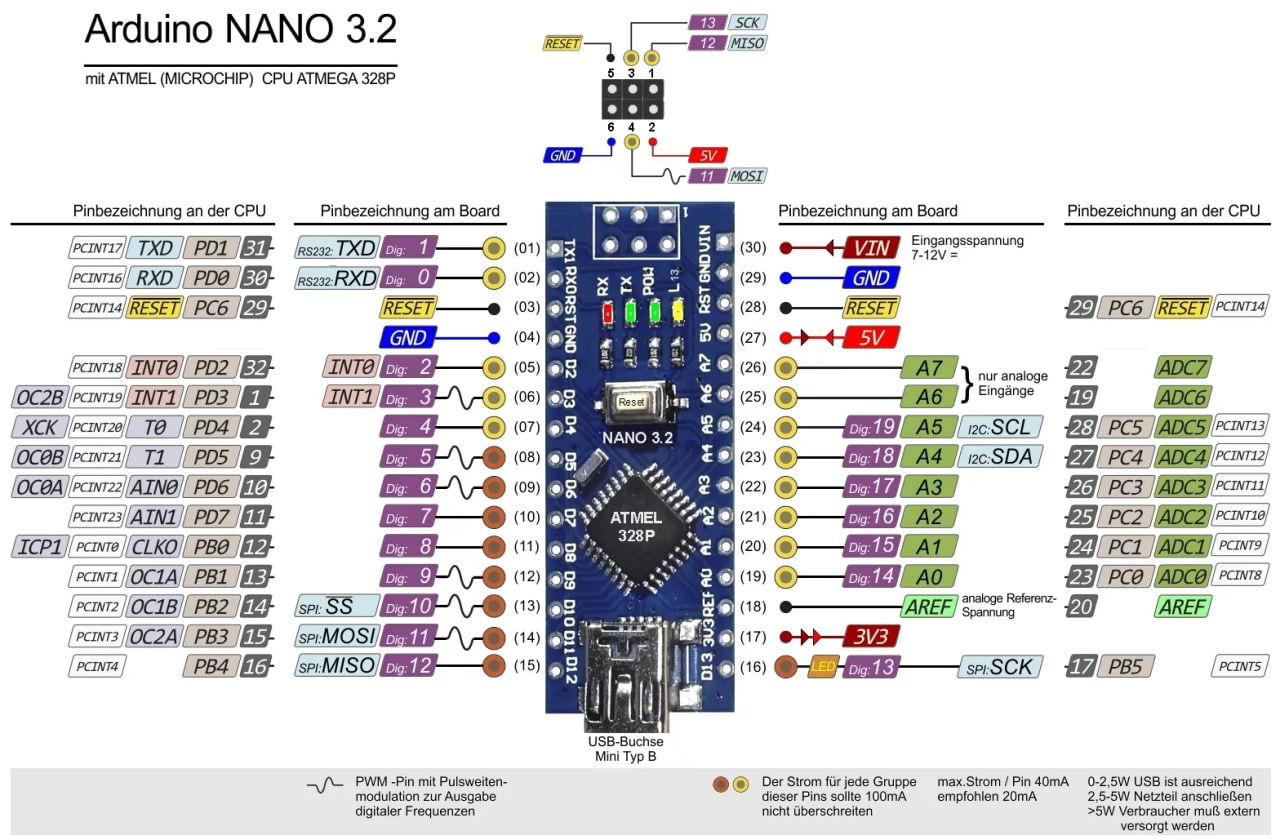
На микроконтроллере каждый пин может быть настроен как вход или выход. Для управления светодиодным экраном используются пины, настроенные на выход. Эти пины подключаются к контроллерам экрана и управляют подачей электричества на строки и столбцы светодиодной матрицы.

Для создания изображения на светодиодном экране микроконтроллер поочередно включает или выключает определенные строки и столбцы. Это приводит к свечению нужных светодиодов. Быстрое переключение состояний строк и столбцов создает впечатление непрерывного изображения.

Работа с светодиодным экраном основана на управлении состоянием строк и столбцов матрицы светодиодов. Каждый светодиод на экране управляется конкретным пином на микроконтроллере. Изображение формируется путем последовательного включения и выключения светодиодов в матрице.

## Практическая часть

На начальном этапе я пытался разобраться, какой пин отвечает за каждую строку и столбец дисплея. Благодаря доступу к самой плате, я сразу смог увидеть, какие пины используются. В модели, с которой я работал, использовались пины D2-D13 и A0-A3.



Управление пинами осуществлялось через регистры DDR, PORT, и PIN. Пины были задействованы из всех портов B, C и D. Для удобства работы со всеми портами сразу, я упаковывал значения всех портов в одну переменную bcd типа short, которая вмещала 16 битов, отвечающих за все строки и столбцы. Я создал два массива, которые хранили номера битов в переменной bcd: один для строк, другой для столбцов. Биты, отвечающие за дисплей, были организованы в переменной bcd следующим образом: старшие разряды занимали биты из порта B, за ними шли биты из порта C, а затем — биты из порта D.

const PROGMEM byte rowBit[] = { 12, 13, 7, 2, 8, 4, 5, 15 };

const PROGMEM byte colBit[] = { 9, 14, 0, 6, 11, 1, 10, 3 };

Я написал метод для установки битов из переменной bcd в регистры. Зная, какой пин за что отвечает, функция отображает значения пинов, переменной bcd типа short в соответствующие регистры. Это решение значительно упростило управление дисплеем.

void Display::setBits(short bcd) {

  PORTD = PORTD & 0b11 | (bcd & 0b111111) << 2;

  PORTC = PORTC & 0b11110000 | (bcd >> 6) & 0b1111;

  PORTB = PORTB & 0b11000000 | (bcd >> 10) & 0b111111;

}

В процессе работы выяснилось, что дисплей функционирует несколько необычно: при подаче тока на пин строки светодиод зажигается, тогда как при подаче тока на пин столбца светодиод затухает. Чтобы зажечь одну точку, необходимо было подать ток на одну строку для зажигания светодиодов и на семь столбцов для их тушения.

Затем я реализовал метод для отрисовки одной точки в заданных координатах.

void Display::  drawDot(byte x, byte y) {

  short bcd = 1 << rowBit[y];

  for (byte i = 0; i < 8; i++)

    if (x != i)

      bcd |= 1 << (colBit[i]);

  setBits(bcd);

  delay(1);

}

Следующим шагом был метод, выводящий изображение на экран. Этот метод принимал массив байтов и там, где стоит единица зажигал светодиод. Светодиоды загорались на 1 миллисекунду, затем затухали и загорались следующие. Используя метод drawDot, каждый из 64 светодиодов зажигался и затухал отдельно на 1 миллисекунду. Этого не хватало, чтобы глаз человека не замечал переключение светодиодов, что обусловлено феноменом фи. Поэтому я создал метод, который выводит байт целиком, зажигая целую строчку на 1 миллисекунду.

void Display::drawRow(byte num, byte row) {

  short bcd = 1 << rowBit[num];

  for (byte i = 0; i < 8; i++)

    if (!((row >> (7 - i)) & 1))

      bcd |= 1 << colBit[i];

  setBits(bcd);

  delay(1);}

void Display::drawImg(byte \_img[8], short x = 0, short y = 0) {

  for (byte i = (y < 0 ? 0 : y); i < 8 + (y < 0 ? y : 0); i++) {

    if (x < 0)

      drawRow(i, pgm\_read\_byte(\_img + i - y) << -x);

    else

      drawRow(i, pgm\_read\_byte(\_img + i - y) >> x);

  }

}

Далее я добавил возможность отрисовывать изображение в заданных координатах и реализовал отрисовку примитивных фигур, таких как квадрат и прямоугольный треугольник. Также был создан метод для отображения транспонированного изображения.

void Display::drawSquare(byte size, short x = 0, short y = 0) {

  drawRow(0 + y, ((1 << size) - 1) << (8 - size - x));

  for (int i = 1; i < size - 1; i++)

    drawRow(i + y, ((1 << size - 1) | 1) << (8 - size - x));

  drawRow(size - 1 + y, ((1 << size) - 1) << (8 - size - x));

}

void Display::drawTriangle(byte size, short x = 0, short y = 0) {

    for (int i = 0; i < size; i++)

      if (x < 0)

        drawRow(i + y, ((1 << i + 1) - 1) << (7 - i) << -x);

      else

        drawRow(i + y, ((1 << i + 1) - 1) << (7 - i) >> x);

void Display::drawTransposition(byte img[8]) {

  for (int i = 7; i >= 0; i--) {

    byte x = ((pgm\_read\_byte(img + 7 - i) & 0x55) << 1) |

((pgm\_read\_byte(img + 7 - i) & 0xAA) >> 1);

    x = ((x & 0xCC) >> 2) | ((x & 0x33) << 2);

    x = (x >> 4) | (x << 4);

    drawRow(i, x);

  }

}

Так же используется оптимизация с помощью PROGMEM для хранения больших массивов данных во флеш-памяти микроконтроллера. Это позволяет сэкономить оперативную память. Данные, объявленные с ключевым словом PROGMEM, сохраняются во флеш-памяти и доступны для чтения, но не могут быть изменены в процессе выполнения программы.

Реализовав библиотеку, я написал программу для поочередной отрисовки изображения с периодичностью в 500 миллисекунд. Ниже приведен код и две картинки, которые поочередно выводятся на экране.

const PROGMEM byte rowBit[] = { 12, 13, 7, 2, 8, 4, 5, 15 };

const PROGMEM byte colBit[] = { 9, 14, 0, 6, 11, 1, 10, 3 };

const PROGMEM byte img1[8] = {

  0b00011000,

  0b00100100,

  0b00100110,

  0b00101001,

  0b01101001,

  0b10010110,

  0b10010000,

  0b01100000

};

const PROGMEM byte img2[8] = {

  0b00011000,

  0b00100100,

  0b00100110,

  0b00101111,

  0b01101111,

  0b11110110,

  0b11110000,

  0b01100000

};

Display display(rowBit, colBit);

byte\* imgs[2] = {img1,img2};

uint32\_t time;

byte index = 0;

void setup() {

}

void loop() {

  display.drawImg(imgs[index]);

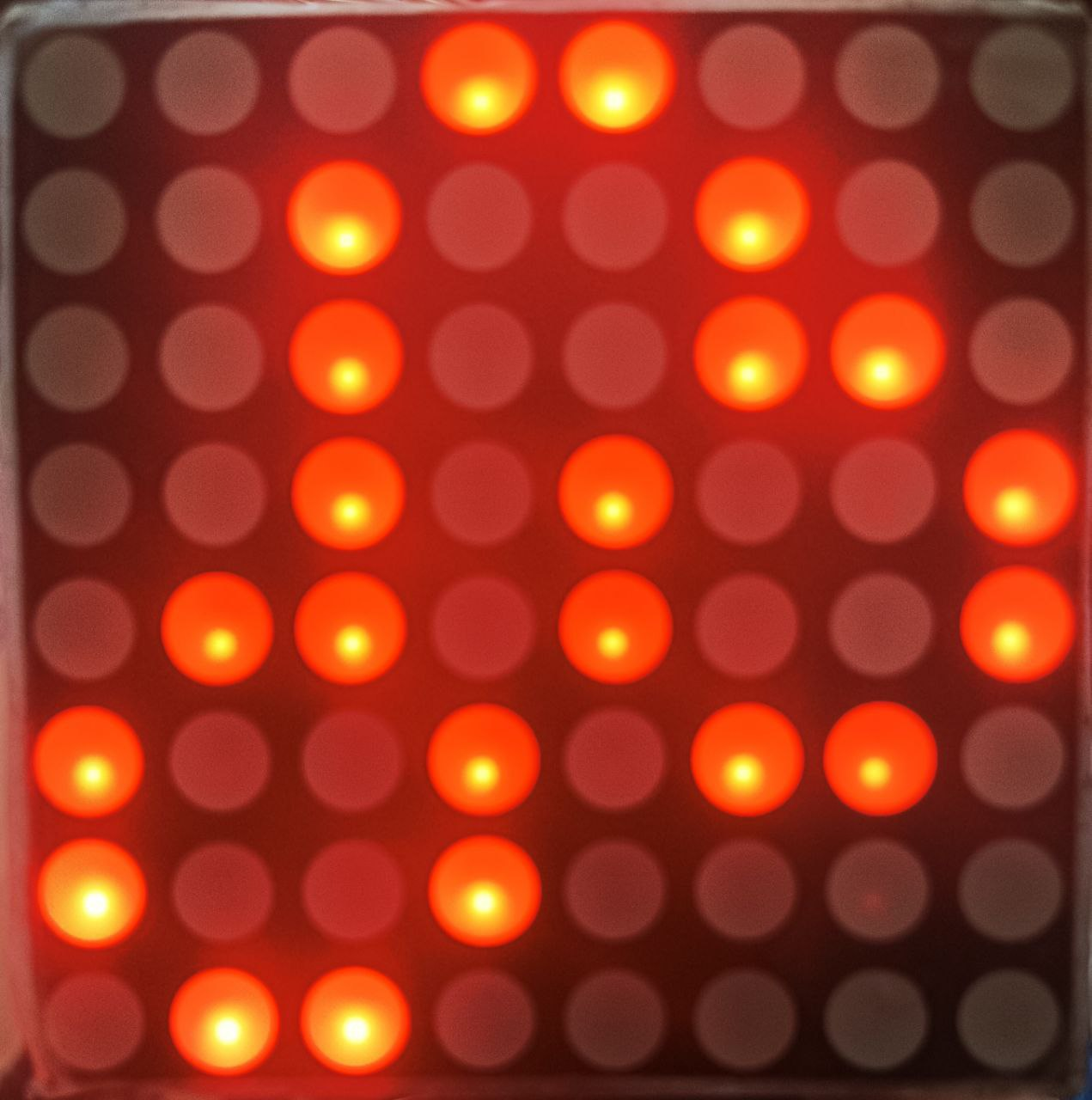
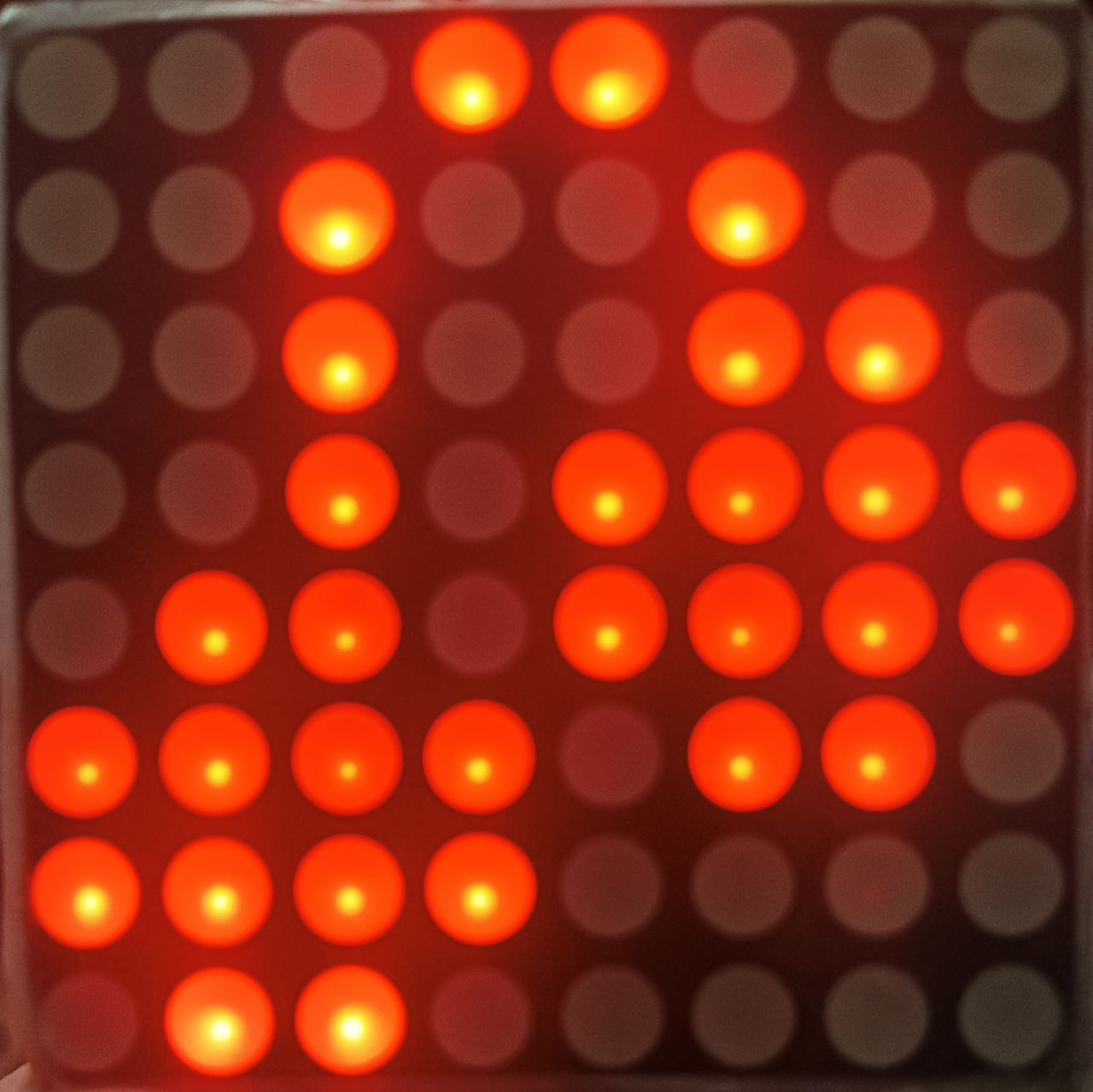
  if (millis() - time > 500){

    index = (index + 1)%2;

    time = millis();

  }

}

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной курсовой работы была разработана и протестирована библиотека для управления светодиодным дисплеем 8x8 на базе Arduino Nano. Основная цель работы — создание функциональной и оптимизированной библиотеки, которая предоставляет удобные методы для отрисовки различных графических элементов на светодиодном дисплее — была успешно достигнута.

Работа над проектом способствовала углублению знаний в области программирования микроконтроллеров, а также практическому освоению методов оптимизации кода и эффективного использования аппаратных ресурсов. Данный проект может служить основой для дальнейших разработок и улучшений в области управления светодиодными дисплеями и другими периферийными устройствами на базе микроконтроллеров.

СПИСОК ЛИТЕРАТУРЫ

1. Саймон Монк. Программирование Arduino: введение в скетчи
2. Джереми Блум. Исследование Arduino: инструменты и техники для инженерных творений
3. Пол Шерз, Саймон Монк. Практическая электроника для изобретателей
4. Arduino documentation [Электронный ресурс]. – Режим доступа: <https://arduino.ru/Reference>
5. AlexGyver [Электронный ресурс]. – Режим доступа: <https://alexgyver.ru/lessons/about-arduino/>