UNIVERSITATEA DE STAT DIN MOLDOVA

FACULTATEA DE FIZICĂ ȘI INGINERIE

DEPARTAMENTUL FIZICA TEORETICĂ “IU.PERLIN”

ABDULLAEV RAMIL

ELABORAREA HARDWARE ȘI SOFTWARE A UNEI LAMPE DECORATIVE

061 TEHNOLOGII ALE INFORMAȚIEI ȘI COMUNICAȚIILOR

TEHNOLOGII INFORMAȚIONALE ÎN MODELARE (MP)

Teză de master

Șef departament **\_\_\_\_\_\_\_\_\_\_\_\_\_\_ NICA Denis, dr.habil., prof.univ.**

Conducător ştiinţific **\_\_\_\_\_\_\_\_\_\_\_\_\_\_ NICA Denis, dr.hab., prof.univ.**

Autorul \_\_\_\_\_\_\_\_\_\_\_\_\_\_ **ABDULLAEV Ramil**

CHIȘINĂU-2023

МОЛДАВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ФИЗИКИ И ИНЖИНЕРИИ

ДЕПАРТАМЕНТ ТЕОРЕТИЧЕСКОЙ ФИЗИКИ ИМ. Ю. ПЕРЛИНА

АБДУЛЛАЕВ РАМИЛ

РАЗРАБОТКА АППАРАТНОЙ ЧАСТИ И ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЕКОРАТИВНОГО СВЕТИЛЬНИКА

061 ТЕХНОЛОГИИ ИНФОРМАЦИИ И КОММУНИКАЦИЙ

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В МОДЕЛИРОВАНИИ (MP)

Магистерская диссертация

Зав. департаментом: **\_\_\_\_\_\_\_\_\_\_\_\_\_\_Денис НИКА, др. хаб., доцент**

Научный руководитель: **\_\_\_\_\_\_\_\_\_\_\_\_\_\_Денис НИКА, др. хаб., доцент**

Автор \_\_\_\_\_\_\_\_\_\_\_\_\_\_**Рамил АБДУЛЛАЕВ**

**КИШИНЕВ-2023**

СОДЕРЖАНИЕ

[СПИСОК АББРЕВИАТУР 5](#_Toc131338639)

[АННОТАЦИЯ 6](#_Toc131338640)

[ADNOTARE 7](#_Toc131338641)

[ВВЕДЕНИЕ 8](#_Toc131338642)

[Глава I. ОБЗОР ТЕХНИЧЕСКИХ ВОЗМОЖНОСТЕЙ ПОСТРОЕНИЯ СОВРЕМЕННЫХ УСТРОЙСТВ ДЕКОРАТИВНОГО ОСВЕЩЕНИЯ 10](#_Toc131338643)

[1.1 Краткий обзор различных архитектур микропроцессоров 11](#_Toc131338644)

[1.2 Способы передачи данных 12](#_Toc131338645)

[1.3 Платформа Arduino 14](#_Toc131338646)

[1.4 Программирование Arduino 17](#_Toc131338647)

[1.5 Светодиодные ленты 20](#_Toc131338648)

[1.6 Разновидности светодиодных лент 21](#_Toc131338651)

[1.7 Заключение 25](#_Toc131338653)

[1.8 Новая глава 26](#_Toc131338654)

[Глава II. АНАЛИЗ РАЗЛИЧНЫХ МОДЕЛЕЙ ПРЕДСТАВЛЕНИЯ ЦВЕТА – ЦВЕТОВЫХ ПРОСТРАНСТВ 26](#_Toc131338655)

[2.1 Постановка задачи 26](#_Toc131338656)

[2.2 Общие понятия цвета. Восприятие цвета человеком 26](#_Toc131338657)

[2.3 Цветовая модель RGB 28](#_Toc131338658)

[2.4 Цветовая модель CMYK 31](#_Toc131338659)

[2.5 Цветовая модель HSV 33](#_Toc131338660)

[2.6 Заключение 35](#_Toc131338669)

[Глава III. РАЗРАБОТКА ДЕКОРАТИВНОГО СВЕТИЛЬНИКА 37](#_Toc131338670)

[3.1 Принципиальная схема 37](#_Toc131338671)

[3.2 Корпус светильника 38](#_Toc131338672)

[3.3 Разработка программы управления 42](#_Toc131338673)

[3.3.1 Глобальные переменные 43](#_Toc131338674)

[3.3.2 Начальная инициализация 45](#_Toc131338675)

[3.3.3 Работа основной части программы 47](#_Toc131338676)

[3.3.4 Менеджер эффектов 48](#_Toc131338677)

[3.3.5 Отработка нажатий на кнопку управления 51](#_Toc131338678)

[3.4 Заключение 58](#_Toc131338679)

[ВЫВОДЫ И РЕКОМЕНДАЦИИ 59](#_Toc131338680)

[БИБЛИОГРАФИЯ 60](#_Toc131338681)

[[14] Цветовая температура, http://www.ylati.ru/czvetovaya\_temperatura.html 60](#_Toc131338682)

[ДЕКЛАРАЦИЯ ОБ ОТВЕТСТВЕННОСТИ 63](#_Toc131338683)

СПИСОК АББРЕВИАТУР

ARM – Advanced RISC (Reduced Instruction Set Computer) Machine

AVR – Alf and Vegard's RISC processor

EEPROM – Electrically Erasable Programmable Read-Only Memory

LED **-** *Light-emitting diode*

RGB – Red, Green, Blue (аддитивная цветовая модель)

HSV - *Hue, Saturation, Value* (цветовая модель)

ПО – Программное обеспечение

АННОТАЦИЯ

Касимова Марина, Разработка схемы аппаратной части и программного обеспечения устройства декоративного освещения, Кишинев, 2020. Введение, три главы, выводы и рекомендации, библиография из 15 наименований, 63 страниц, 2 приложения, 17 рисунков, 1 таблица.

**Ключевые слова**

Arduino, микроконтроллер, RGB, светодиод, цветовое пространство.

**Цели и задачи работы**

Целью работы является разработка устройства декоративного освещения, основанного на адресной светодиодной ленте RGB. Для осуществления поставленной цели в работе необходимо решить следующие задачи: разработка электрической принципиальной схемы, сборка конструктива и реализация программного обеспечения.

**Теоретическая и прикладная значимость**

В процессе выполнения работы были изучены основы работы светодиодов - полупроводниковых источников оптического излучения. Кроме этого были получены базовые навыки программирования микроконтроллеров в среде Arduino.

ADNOTARE

Casimova Marina, Elaborarea hardware și software a unui dispozitiv dispozitiv de iluminat decorativ, Chișinău, 2019. Introducere, trei capitole, concluzii și recomandări, bibliografie din 15 titluri, 63 pagini, 2 anexe, 17 figuri, 1 tabelа.

**Cuvintele-cheie**

arduino, microcontroler, RGB, lumina diodice, proiectare a circuitelor.

**Scopul si obiectivele cercetarii**

Scopul acestei lucrări este elaborarea dispozitivului iluminat decorativ bazat pe bandă led de adresă. Pentru a îndeplini scopul stabilit în lucrare este necesar de rezolvat următoarele obiective: dezvoltarea unui concept electric, asamblarea unei construcții și implementarea de software.

**Valoarea teoretică şi aplicativă**

În procesul de realizare a lucrării, au fost studiate elementele de bază ale funcționării LED-urilor - surse de radiații optice semiconductoare. În plus, au fost obținute abilitățile de bază ale programării microcontrolerelor în mediul Arduino.

ВВЕДЕНИЕ

*Актуальность и важность темы*

В последнее время светодиоды получили широкое распространение как источники света, так и в сфере устройств декоративного освещения. Появление адресных светодиодных лент породило массу новых возможностей. Такие ленты стали основой при разработке устройств декоративного освещения, управляемых светильников и даже полноценных LED-видео экранов, используемых для наружной рекламы и систем массового оповещения (в аэропортах и на вокзалах).

Устройства отображения информации, построенные на современных светодиодах, имеют хорошие показатели яркости и контрастности, благодаря чему хорошо справляются с поставленной задачей как в темных, так и в светлых помещениях.

*Цели и задачи работы*

Целью работы является изучение технических возможностей современных RGB-светодиодов и конструкций построенных на их основе. Для достижения поставленной цели в работе предлагается осуществить следующие задачи:

* разработка принципиальной схемы двухмерной матрицы RGB-светодиодов;
* сборка макета-конструктива настольного светильника;
* реализация программного обеспечения для управления различными режимами свечения светодиодов.

*Методологическая и теоретико-научная основы работы (методы исследования)*

Цели данной работы были достигнуты при использовании следующих технологий: микроконтроллеры семейства Arduino, RGB-светодиоды, технология каскадной передачи данных последовательно подключенным устройствам.

*Теоретическая и практическая важность*

Результаты данной работы могут послужить обогащению знаний в проектировании устройств декоративного освещения, программирования микроконтроллеров, а также построения простых систем управления.

*Новизна исследованной темы*

Новизна полученных в работе результатов, в основном связана с проведенным анализом существующих на рынке технологий и компонент для создания декоративных светодиодных светильников в домашних условиях.

*Ключевые слова:* arduino, микроконтроллер, RGB-светодиоды, светодиод, схемотехника.

Глава I. ОБЗОР ТЕХНИЧЕСКИХ ВОЗМОЖНОСТЕЙ ПОСТРОЕНИЯ СОВРЕМЕННЫХ УСТРОЙСТВ ДЕКОРАТИВНОГО ОСВЕЩЕНИЯ

Понятие устройств декоративного освещения

Устройства декоративного освещения, построенные на базе светодиодов, в настоящее время получили широкое распространение в различных технических устройствах. Технологии производства полупроводниковых источников света достигли высокой надежности светодиодов, их компактности, а также долговечности.

Одной из отраслей применения светодиодов является создание устройств отображения информации (дисплеев). Ярким примером использования полупроводниковых светодиодов является технология OLED. Данная технология предполагает использование тонкоплёночных многослойных структур, состоящих из слоев нескольких полимеров. Эмиссионный слой такого светодиода излучает свет (фотоны в области видимого света) за счет рекомбинации электронов, теряющих энергию, что сопровождается излучением (эмиссией).

Другим хорошим примером применения светодиодов является создание Светодиодных графических экранов. Определение такого экрана взято из свободного источника (Wikipedia):

***Светодиодный экран*** *(LED screen, LED display) — устройство отображения и передачи визуальной информации (*[*дисплей*](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BF%D0%BB%D0%B5%D0%B9)*,* [*монитор*](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%BD%D0%B8%D1%82%D0%BE%D1%80_(%D1%83%D1%81%D1%82%D1%80%D0%BE%D0%B9%D1%81%D1%82%D0%B2%D0%BE))*,* [*телевизор*](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BB%D0%B5%D0%B2%D0%B8%D0%B7%D0%BE%D1%80)*), в котором каждой точкой - пикселем - является один или несколько полупроводниковых светодиодов (LED).*

Такие экраны могут использоваться в местах общего пользования (вокзалы, аэропорты, остановки общественного транспорта) для отображения необходимой информации, либо для отображения рекламы. Хорошие характеристики яркости и контрастности светодиодов позволяют без проблем использовать их как в закрытых, так и в открытых местах. Управление такими экранами возможно как с помощью различных аппаратных реализаций, так и с помощью микропроцессора. Программная реализация управления экраном более гибкая и позволяет менять отображаемую информацию.

* 1. Краткий обзор различных архитектур микропроцессоров

Архитектура, то есть логическая организациямикропроцессора, однозначно определяет свойства, особенности и возможности построения вычислительной системы на базе данного микропроцессора.[1]

Большинство современных микропроцессоров при всем разнообразии их типов, моделей и производителей имеют одну из трех типов архитектуры: CISC, RISC и MISC (это относится к микропроцессорам универсального, а не специального применения).

**Архитектура CISC** (Complex Instruction Set Computer) - командо-комплексная система управления компьютером. Отличается повышенной гибкостью и расширенными возможностями персонального компьютера (РС), выполненного на микропроцессоре, и характеризуется:

1. большим числом различных по длине и формату команд;
2. использованием различных систем адресации;
3. сложной кодировкой команд.

**Архитектура RISC** (Reduced Instrucktion Set Computer) - командо-однородная система управления компьютером, имеет свои особенности:

1. использует систему команд упрощенного типа: все команды имеют одинаковый формат с простой кодировкой, обращение к памяти осуществляется командами загрузки (данных из RAM в регистр микропроцессора) и записи (данных из регистра микропроцессора в память), остальные используемые команды - формата регистр-регистр;
2. при высоком быстродействии допускается более низкая тактовая частота;
3. команда меньше нагружает RAM;
4. отладка программ на RISC более сложна, чем на CISC;
5. с архитектурой CISC программно несовместима.

**Архитектура MISC** (Multipurpose Instruction Set Computer) - многоцелевая командная система управления компьютером, сочетает в себе преимущества CISC и RISC. Элементная база состоит из отдельных частей (могут быть объединены в одном корпусе): основная часть (HOST - ведущая), архитектуры RISC CPU, а расширяемая часть - с подключением ПЗУ (ROM) микропрограммного управления. При этом вычислительная система приобретает свойства CISC: - основные команды работают на HOST, а команды расширения образуют адрес микропрограммы для своего выполнения. HOST выполняет команды за один такт, а расширение эквивалентно CPU со сложным набором команд (CISC). Наличие ПЗУ устраняет недостаток RISC, связанный с тем, что при компиляции с языка высокого уровня код операции (микропрограмма) уже дешифрирована и открыта для программиста.

В данной работе применена плата семейства Arduino Leonardo, построенная на базе микроконтроллера Atmega32U4. Данный микроконтроллер построен на RISC архитектуре. Более подробно особенности выбранной аппаратной платформы будут рассмотрены ниже.

* 1. Особенности работы микроконтроллеров

Современные микроконтроллеры (МК) относятся к полупроводниковым компонентам, основное назначение которых – управление электронными устройствами. Они встраиваются непосредственно в исполнительный модуль, задавая режимы работы обслуживаемых изделий. На их основе реализована оригинальная идея, позволяющая совместить в одном кристалле сразу несколько функций.

Подобно классическому компьютеру они содержат в своем составе не только центральный процессор. Помимо него в МК входят рабочая память, ПЗУ, а также элементы периферии, объединенные в похожем на микросхему корпусе.

**Что такое микроконтроллеры**

Современный микроконтроллер, можно сравнить с основным элементом любого компьютерного устройства – микропроцессором. Последний представляет собой центральную часть ПК, выполненную на основе интегральных схем. Название «микропроцессор» (МП) указывает на то, что именно в этом модуле происходят все вычислительные операции, определяющие производительность системы в целом.

Чтобы получить на его основе полноценную ЭВМ – необходимо дополнить МП целым рядом внешних устройств. Добиться этого удается добавлением к этому модулю оперативной памяти и периферийных портов ввода-вывода информации. В результате получается законченное автономное устройство или компьютер.

Подобно ПК микроконтроллеры содержат в своем составе следующие компоненты:

* Микропроцессор.
* Оперативную память.
* Комплект периферии, состоящей из портов обмена информацией с обслуживаемым устройством.

В отличие от ПК он не является полностью самостоятельным (автономным) изделием. Микроконтроллеры, как правило, встраиваются в гаджеты, игрушки и в другие исполнительные устройства, функционированием которых они управляют.

Отличия микроконтроллера от микропроцессора

  Разница между микропроцессором и микроконтроллером заключается в наличии оперативной памяти, ПЗУ и других периферийных устройств в микроконтроллере. Микропроцессор содержит только процессор и не имеет других компонентов.

Микропроцессор и микроконтроллер, оба являются основными процессорами, предназначенными для работы компьютеров. Функции обоих процессоров одинаковы. Основное различие между ними состоит в том, что микропроцессоры выполняют различные функции, тогда как микроконтроллеры - это небольшие компьютеры, предназначенные для конкретных задач.

* 1. Микроконтроллеры семейства AtXmega

Микроконтроллеры **AVR XMEGA** выполнены на основе: Технологии picoPower второго поколения Инновационной системы обработки событий "Event System", которая обеспечивает независимую от ЦПУ быстродействующую передачу данных между внутренними периферийными устройствами 4-канального контроллера ПДП, улучшающего характеристики микроконтроллера Быстродействующих 12-битных АЦП и ЦАП Ускорителя криптографических алгоритмов AES и DES  
      Микроконтроллеры AVR XMEGA производят такое же ошеломляющее впечатление, которое производили их предшественники микроконтроллеры AVR после своего появления. Новые микроконтроллеры способны работать при напряжении питания всего лишь 1.6 В и достигать производительности 32 MIPS на тактовой частоте 32 МГц. Микроконтроллеры содержат флэш-память размером 16…384 кбайт и поставляются в 44…100-выводных корпусах. Микроконтроллеры XMEGA являются микроконтроллерами общего назначения и могут использоваться в широком числе применений, в т.ч. аудиосистемы, ZigBee® системы, медицинская техника, контроллеры автоматизации, коммуникационное оборудование, измерительные приборы, оптические трансиверы, системы управления электроприводами, бытовое электрооборудование, оптические трансиверы и любая другая продукция с батарейным питанием.

**Рис. 1.2.** Печатная плата Arduino Leonardo

В таблице 1.1. приведены пояснения к приведенной выше схеме: перечень основных компонентов микросхемы Arduino UNO.

**Таблица 1.1.** Компоненты микросхемы Arduino UNO

|  |  |
| --- | --- |
| **Номер компонента** | **Назначение** |
| 1 | Кнопка аппаратного сброса контроллера. |
| 2 | Вывод analog reference; используется для задания верхнего предела для аналоговых входов на плате Arduino. |
| 3 | Цифровые выводы; мультиплексные выводы для приема/подачи дискретного сигнала. Некоторые из этих выводов (помеченные символом «~») могут использоваться для формирования ШИМ-сигнала (PWM). |
| 4 | Разъём для подключения внешнего программатора. |
| 5 | Микроконтроллер ATMega32U4. |
| 6 | Выводы, которые могут использоваться для работы с аналоговыми сигналами. |
| 7 | Вывод Vin; используется для подачи внешнего питания на схему Arduino. |
| 8 | Выводы, подключенные к общей шине платы Arduino. |
| 9 | Выводы питания; применяются для питания внешних модулей под управлением Arduino (3,3 и 5 вольт соответственно). |
| 10 | Вывод сброса программы; работает по тому же принципу что и кнопка сброса. |
| 11 | Вход питания; коаксиальный вход для питания схемы Arduino. |
| 12 | Регулятор напряжения; стабилизирует напряжение, подаваемое на схему. |
| 13 | Источник опорной частоты микроконтроллера. |
| 14 | USB вход; применяется для программирования и питания платы в процессе разработки. |

* 1. Среда разработки IAR 5.3

Язык программирования устройств Arduino основан на C/C++.

Файл проекта Arduino называется Скетч. Для полноценной работы программы в среде Arduino должны быть реализованы две функции: *setup();* и *loop();*

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **void** **setup**() {  }  **void** **loop**() {  } |

* 1. Способы передачи данных между микросхемами.

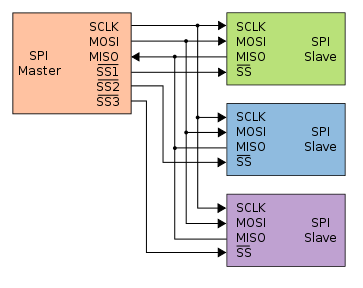
**SPI** (англ. Serial Peripheral Interface, SPI bus — последовательный периферийный интерфейс, шина SPI) — последовательный синхронный стандарт передачи данных в режиме полного дуплекса, предназначенный для обеспечения простого и недорогого высокоскоростного сопряжения микроконтроллеров и периферии. SPI также иногда называют четырёхпроводным (англ. four-wire) интерфейсом.



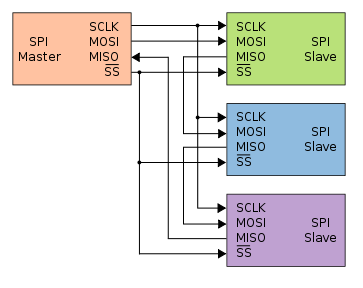
1. **MOSI** — выход ведущего, вход ведомого (англ. Master Out Slave In). Служит для передачи данных от ведущего устройства ведомому.
2. **MISO** — вход ведущего, выход ведомого (англ. Master In Slave Out). Служит для передачи данных от ведомого устройства ведущему.
3. **SCLK** — последовательный тактовый сигнал (англ. Serial Clock). Служит для передачи тактового сигнала для ведомых устройств.
4. **CS** или **SS** — выбор микросхемы, выбор ведомого (англ. Chip Select, Slave Select).

**Принцип работы**  
В простейшем случае к ведущему устройству подключено единственное ведомое устройство и необходим двусторонний обмен данными. В таком случае используется трехпроводная схема подключения. Интерфейс SPI позволяет подключать к одному ведущему устройству несколько ведомых устройств, причем подключение может быть осуществлено несколькими способами.  
Радиальная структура связи с несколькими ведомыми устройствами через SPI.

**Первый способ** позволяет реализовать радиальную структуру связи (топология типа «звезда»), его принято считать основным способом подключения нескольких ведомых устройств. В данном случае для обмена более чем с одним ведомым устройством ведущее устройство должно формировать соответствующее количество сигналов выбора ведомого устройства (SS). При обмене данными с ведомым устройством, соответствующий ему сигнал SS переводится в активное (низкое) состояние, при этом все остальные сигналы SS находятся в неактивном (высоком) состоянии. Выводы данных MISO ведомых устройств соединены параллельно, при этом они находятся в неактивном состоянии, а перед началом обмена один из выходов (выбранного ведомого устройства) переходит в активный режим.  
Кольцевая структура связи с несколькими ведомыми устройствами через SPI



**Второй способ** позволяет реализовать структуру связи типа «кольцо». В данном случае для активации одновременно нескольких ведомых устройств используется один сигнал SS, а выводы данных всех устройств соединены последовательно и образуют замкнутую цепь. При передаче пакета от ведущего устройства этот пакет получает первое ведомое устройство, которое, в свою очередь, транслирует свой пакет следующему ведомому устройству и так далее. Для того, чтобы пакет от ведущего устройства достиг определенного ведомого устройства, ведущее устройство должно отправить еще несколько пакетов.



**Особенности:**  
  
- Многие SPI чипы и микроконтроллеры поддерживают работу на частоте более 10 Мгц, обечпечивая намного более высокую скорость передачи данных, чем интерфейс I2C

**Использование:**

Часто этот интерфейс используется для работы с SPI-Flash в которой хранится программное обеспечение.

**Протокол I2C.**

**I2C (Inter-Integrated Circuit)** — последовательная шина данных для связи интегральных схем, использующая две двунаправленные линии связи (SDA и SCL). Используется для соединения низкоскоростных периферийных компонентов с материнской платой, встраиваемыми системами и мобильными телефонами.  
Иногда эту шину называют "квадратичной" или "квадратной" или "Ай-Ту-Си"

**Принцип работы:**



Микроконтроллер на рисунке это ведущий элемент **(Master1)** им может быть процессор. На рисунке представлено 3 ведомых перефириных элемента **Slave** В качестве **Slave** могут быть память, ЦАП, АЦП и пр. К шине может быть подключено до 127 устройств.  
  
Процессор с памятью соединен в данном случае по двум шинам:  
**SDA** (Serial DATA)- шина последовательной передачи данных. Данные по этой шине могут передаваться в двух направлениях.  
**SCL** (Serial Clock) - шина по которой идет тактирование шины данных. Шина синхронизации данных. Она также определяет в какой момент куда пойдут данные. В схеме Master-Master первым битом определяется, кто займет главную роль.  
**Скорость передачи данных.** Так как  передаются по 1 биту за 1 такт, то скорость передачи данных составляет 1/8 от тактовой частоты.



**Состояние СТАРТ и СТОП**

В протоколе I2C существуют два важных состояния - СТАРТ (START) и СТОП (STOP). Они играют ключевую роль в инициации и завершении передачи данных по шине I2C. Вот краткое описание каждого из них:

**1.СТАРТ (START):**

* Состояние СТАРТ инициирует начало передачи данных по шине I2C.
* Мастер-устройство (генератор тактовых импульсов) генерирует СТАРТ-сигнал.
* СТАРТ-сигнал состоит из перехода линии SDA с высокого уровня (1) на низкий уровень (0) при нахождении линии SCL на высоком уровне (1).
* После СТАРТ-сигнала мастер выбирает адрес устройства, с которым он хочет общаться, и передает его по шине SDA.

**2.СТОП (STOP):**

* Состояние СТОП используется для завершения передачи данных по шине I2C.
* Мастер-устройство генерирует СТОП-сигнал.
* СТОП-сигнал состоит из перехода линии SDA с низкого уровня (0) на высокий уровень (1) при нахождении линии SCL на высоком уровне (1).

СТОП-сигнал указывает, что передача данных завершена, и освобождает шину I2C для других устройств.

СТАРТ- и СТОП-сигналы определяют начало и конец каждой транзакции данных по шине I2C. Они служат важным механизмом для синхронизации и контроля передачи данных между мастером и слейвами на шине I2C.

* 1. Разновидности светодиодных лент

Светодиодные ленты производятся с использованием SMD- и DIP-технологий. Цифры в обозначении означают размер чипа кристалла в десятых долях миллиметра (SMD 3528 — размер 3,5 мм на 2,8 мм). В зависимости от типа светодиодов ленты разделяются по величине светового потока (количеству светодиодов в 1 метре ленты) и цвету свечения. Бывают ленты с монохромным свечением (красного, зелёного, синего, жёлтого, белого цвета) и многоцветные (с возможностью создания практически любого оттенка, RGB). Так же, как и светодиоды с белым цветом, светодиодные ленты бывают различной цветовой температуры — от 2700 К до 10000 К.

Цветовая температура - это эффективная величина, равная температуре абсолютно черного тела, при которой отношение энергетических яркостей для двух длин волн его спектра равно отношению этих же величин для спектра исследуемого источника света. Цветовая температура (CCT - Correlated Colour Temperature) измеряется в градусах по шкале Кельвина (K). Чем выше значение K, тем холоднее свет[14].

Ниже приведены различные значения цветовой температуры и соответствующие им оттенки цвета:

* Очень тёплый свет: 2500-2800 K;
* Тёплый свет: 2800-3500 K;
* Нейтральный свет: 3500-5000 K;
* Дневной свет: 6000 К;
* Холодный свет: 5000 K и выше.

Человеческий глаз имеет особенность приспосабливаться к изменению цветовой температуры в диапазоне 3000 К - 10000 К. Однако при выборе источников искусственного освещения не стоит забывать что комфортная цветовая температура лежит в диапазоне от 3000 К до 5000 К.

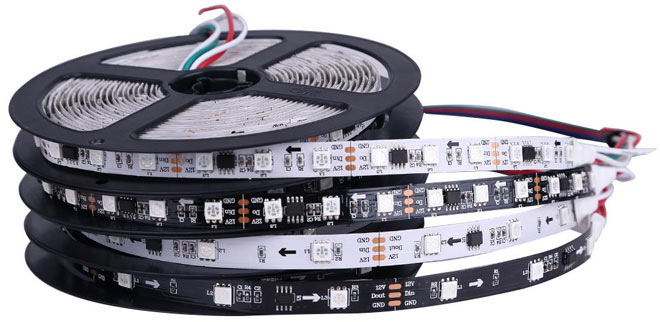
Многоцветные светодиодные ленты в свою очередь могут быть построены двумя способами. Первый способ заключается в размещении чередующихся светодиодов трёх цветов (красный, зелёный и синий). Недостаток такой ленты – плохое смешивание цветов без дополнительных рассеивателей. Второй способ изготовления многоцветных лент – использование RGB-светодиодов. Такие светодиоды совмещают в одном корпусе три светодиода.

**Рис. 1.4.** Светодиодная лента RGB с управлением от питания.

Каждый светодиод такой ленты имеет 4 вывода - один общий +12V (анод) и три катода на каждый цвет, т.е. внутри одного корпуса находится три светодиода разных цветов. Соответственно, такие же выводы имеет и лента: 12V, G, R, B. Подавая питание на общий +12V и отрицательное напряжение на любой из цветов - мы включаем этот цвет. Подадим на все три – получим белый цвет. Подадим на зелёный и красный – получится жёлтый цвет, и так далее. Для таких лент существуют контроллеры с пультами. Типичный контроллер представляет собой три полевых транзистора на каждый цвет и микроконтроллер, который управляет транзисторами. Таким образом, давая возможность включить любой цвет.

*Адресные светодиодные ленты*

В отличие от обычной светодиодной RGB-ленты, в которой все светодиоды одинаково реагируют на сигнал с RGB-контроллера, в адресной LED-ленте каждый светодиод получает индивидуальную команду управления[15]. В результате пользователь может максимально точно подбирать оттенок для каждого светодиода, создавать световые эффекты и собирать матрицы с 16-млн цветов. Адресная светодиодная лента состоит из RGB-светодиодов в SMD корпусе 5050 и микросхем ШИМ-драйверов. В настоящее время наиболее популярными являются адресные LED-ленты с использованием чипов WS2811 и WS2812B. Модификация WS2811 представляет собой интегральную микросхему (ИМС) в корпусе DIP-8 (9,2х6,4 мм) или SOP-8 (5,1х4,0 мм).



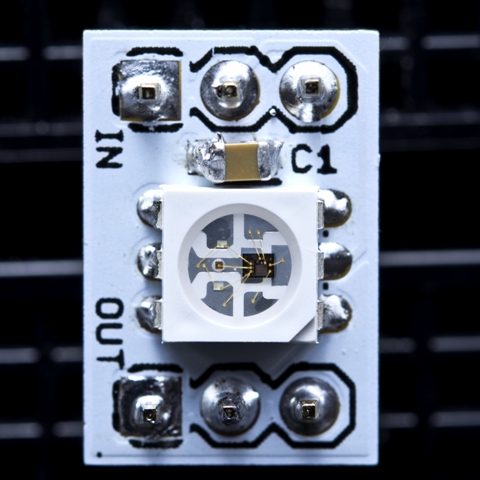
**Рис.1.5. Адресная светодиодная лента WS2811.**

В проекте используется лента со светодиодами типа WS2812B. Рассмотрим этот тип подробнее. Данный тип ШИМ-драйвера намного компактнее и размещается непосредственно в корпусе светодиода SMD 5050.



**Рис.1.6.** Адресная светодиодная лента WS2812B.

На более детальном изображении видны все детали чипа WS2812B.



**Рис.1.7.** Детальное изображение WS2812B.

* 1. Заключение

В данной главе были рассмотрены аппаратные и программные средства для реализации устройства декоративного освещения.

Были рассмотрены существующие архитектуры процессоров и в том числе архитектура AVR, используемая во встроенных системах. Были описаны способы обмена процессором информацией с периферийными устройствами посредством шин: последовательная и параллельная передача данных. Был рассмотрен микропроцессор Arduino, его аппаратные компоненты и процесс написания программ под него.

Из периферийных устройств были представлены различные варианты светодиодных лент. В том числе были рассмотрены Адресные светодиодные ленты – как пример того с чем пришлось иметь дело в процессе данной работы.

Понимание принципа работы аппаратной и программной части является необходимым требованием и будет использоваться в дальнейшем при построении и анализе оптимальности построенного устройства.

Глава II. АНАЛИЗ РАЗЛИЧНЫХ МОДЕЛЕЙ ПРЕДСТАВЛЕНИЯ ЦВЕТА – ЦВЕТОВЫХ ПРОСТРАНСТВ

1. Постановка задачи

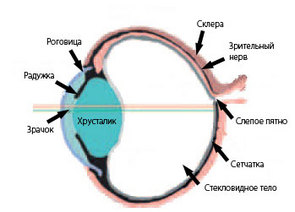
Целью данной главы является изучение различных способов представления цвета в современной технике. Хотелось бы немного затронуть тему понимания природы восприятия цвета. Найти отличительные особенности различных способов представления цвета, а также определить их различия. Определить какие цветовые пространства и для чего удобнее использовать.

1. Общие понятия цвета. Восприятие цвета человеком

Глаз человека способен воспринимать свет – электро-магнитное излучение определенного диапазона длин волн. Видимой области соответствует диапазон длин волн от 400 нм (фиолетовая граница) до 760 нм (красная граница), что составляет ничтожную часть полного электромагнитного спектра. Однако не следует принижать этой фразой сложность и важность человеческого зрения. Дело в том, что именно для этого диапазона воздух атмосферы Земли является наиболее прозрачным. То есть эволюция в паре с естественым отбором создали прибор наиболее подходящий для конкретных условий нашей Планеты. В жёстких условиях выживания зрение сыграло очень важную роль как при защите от хищников, так и при охоте. Цветовое восприятие во много раз преувеличило способности зрения.

Цвет — зрительное, субъективное восприятие человеком видимого света, различий в его спектральном составе, ощущаемых глазом. У людей цветовое зрение развито намного лучше, чем у других млекопитающих. Восприятие цвета определяется индивидуальностью человека, а также спектральным составом, цветовым и яркостным контрастом с окружающими источниками света, а также несветящимися объектами.[8]

Свет действует на фоточувствительные рецепторы сетчатки глаза, и те, в свою очередь, вырабатывают сигнал, который передаётся в мозг. Ощущение цвета, как и всё многоступенчатое зрительное восприятие, сложным образом формируется в цепочке: глаз (экстерорецепторы и нейронные сети сетчатки) — зрительные области мозга.



**Рис. 2.1.** Человеческий глаз

Распознавание цвета человеком зависит от освещения объекта, отражающего света, от глаз и мозга наблюдателя.

Цвет - интерпретация окружающей действительности в мозге человека, которая возникает в результате обработки сигналов, возникших в процессах возбуждения и торможения светочувствительных клеток — колбочек, рецепторов глазной сетчатки человека.

При этом колбочки отвечают за восприятие цвета, палочки за сумеречное зрение.

Глаз реагирует на три первичных цвета: красный, зеленый и синий. Человеческий мозг, в свою очередь, воспринимает цвет как сочетание этих трех сигналов. Если в сетчатке глаза ослаблено или исчезает восприятие одного из трёх основных цветов, то человек не воспринимает какой-то цвет. Встречаются люди, которые, например, не могут отличить красный цвет от зелёного. Так, около семи процентов мужчин и около половины процента женщин страдают такими проблемами. Полная "цветовая слепота", при которой рецепторные клетки не работают вообще, встречается крайне редко. У некоторых людей проявляются трудности ночного видения, что объясняется слабой чувствительностью палочек - наиболее высокочувствительных рецепторов сумеречного зрения. Это может быть наследственным фактором или вследствие недостатка витамина А. Однако человек приспосабливается к "цветовым расстройствам", и их практически невозможно обнаружить без специального обследования.[8]

С целью унификации понятий цвета в технике было введено понятие цветовых моделей. Основная задача цветовых моделей – сделать возможным задание цветов унифицированным образом. По сути, цветовые модели задают определённые системы координат, которые позволяют однозначно определить цвет. Очень часто понятие Цветовая Модель заменяют понятием Цветовое Пространство, что в принципе допустимо, так как оба этих понятия можно описать одинаково.

Цветовое пространство — модель представления цвета, основанная на использовании цветовых координат.

Цветовое пространство строится таким образом, чтобы любой цвет был представлен точкой, имеющей определённые координаты. Чаще всего одному набору координат будет соответствовать один цвет.

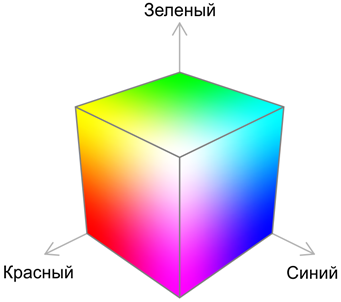
Цветовые пространства описываются набором цветовых координат и правилами построения цветов. К примеру, RGB является трёхмерным цветовым пространством, где каждый цвет описан набором из трёх координат. Каждая из них отвечает компоненте цвета в разложении на красный, зелёный и синий цвета. Количество координат задаёт размерность пространства. Существует много цветовых пространств различной размерности. Например, начиная с одномерных пространств, которые могут описать исключительно монохромное изображение и заканчивая шестимерными пространствами, такими как CMYKLcLm (Cyan, Magenta, Yellow, Key color, lightCyan, lightMagenta). Пространства высокой размерности чаще всего используются в целях печати на плоттерах или аппаратах для цветопроб.

1. Цветовая модель RGB

RGB (аббревиатура английских слов red, green, blue — красный, зелёный, синий) или КЗС (красный, зелёный, синий) — аддитивная цветовая модель, описывающая способ кодирования цвета для цветовоспроизведения с помощью трёх цветов, которые принято называть основными. Выбор основных цветов обусловлен особенностями физиологии восприятия цвета сетчаткой человеческого глаза.[9]

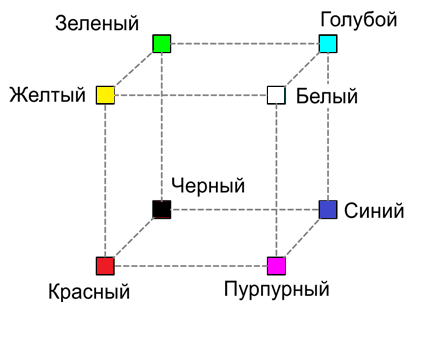
RGB-модель является аддитивной, где цвета получаются путём добавления (англ. addition) к чёрному цвету. При отсутствии краски - нет никакого цвета. Чёрный это минимальное смещения, а максимальное даёт белый цвет. Если цвет экрана, освещённого цветным прожектором, обозначается в RGB как (r1, g1, b1), а цвет того же экрана, освещённого другим прожектором, — (r2, g2, b2), то при освещении двумя прожекторами цвет экрана будет обозначаться как (r1+r2, g1+g2, b1+b2).

В модели RGB (от англ. red – красный, green – зелёный, blue – голубой) все цвета получаются путём смешения трёх базовых (красного, зелёного и синего) цветов в различных пропорциях. Доля каждого базового цвета в итоговом цвете может восприниматься как координата в соответствующем трёхмерном пространстве, поэтому данную модель часто называют цветовым кубом. На рисунке ниже представлена модель цветового куба в пространстве RGB.



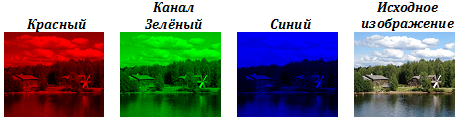
**Рис. 2.2.** Система координат цветового пространства RGB.

Чаще всего модель строится так, чтобы куб был единичным. Точки, соответствующие базовым цветам, расположены в вершинах куба, лежащих на осях: красный – (1;0;0), зелёный – (0;1;0), синий – (0;0;1). При этом вторичные цвета (полученные смешением двух базовых) расположены в других вершинах куба: голубой — (0;1;1), пурпурный — (1;0;1) и жёлтый – (1;1;0). Чёрный и белый цвета расположены в начале координат (0;0;0) и наиболее удалённой от начала координат точке (1;1;1). На рисунке показаны только вершины куба[10].



**Рис. 2.3.** Система координат цветового пространства RGB.

Цветные изображения в модели RGB строятся из трёх отдельных изображений-каналов. На следующем рисунке показано разложение исходного изображения на цветовые каналы.



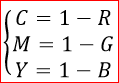
**Рис. 2.4.** Разложение исходного изображения на цветовые каналы.

В модели RGB для каждой составляющей цвета отводится определённое количество бит, например, если для кодирования каждой составляющей отводить 1 байт (8 бит), то с помощью этой модели можно закодировать 2^(3\*8)≈16 млн. цветов. Такая разрядность цвета используется в светодиодной ленте, которую я применила в проекте.

1. Цветовая модель CMYK

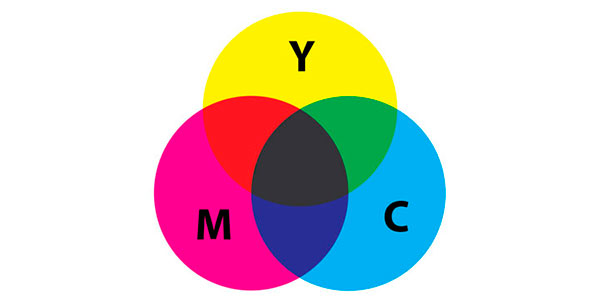
Субтрактивная модель CMY (от англ. cyan — голубой, magenta — пурпурный, yellow — жёлтый) используется для получения твёрдых копий (печати) изображений, и в некотором роде является антиподом цветового RGB-куба. Если в RGB модели базовые цвета – это цвета источников света, то модель CMY – это модель поглощения цветов.[10]

Например, бумага, покрытая жёлтым красителем, не отражает синий свет, т.е. можно сказать, что жёлтый краситель вычитает из отражённого белого света синий. Аналогично голубой краситель вычитает из отражённого света красный, а пурпурный краситель вычитает зелёный свет. Именно поэтому данную модель принято называть субтрактивной. Алгоритм перевода из модели RGB в модель CMY очень прост:



**Рис. 2.5.** Алгоритм перевода из модели RGB в модель CMY.

При этом предполагается, что цвета RGB находятся в интервале [0;1]. Легко заметить, что для получения чёрного цвета в модели CMY необходимо смешать голубой, пурпурный и жёлтый в равных пропорциях. Этот метод имеет два серьёзных недостатка: во-первых, полученный в результате смешения чёрный цвет будет выглядеть светлее «настоящего» чёрного, во-вторых, это приводит к существенным затратам красителя. Поэтому на практике модель СMY расширяют до модели CMYK, добавляя к трём цветам чёрный (по аналогии с английским словом black, в котором в конце буква K).



**Рис. 2.6.** Смешение цветов CMYK.

Схема CMYK обладает сравнительно с RGB меньшим цветовым охватом. Печать четырьмя красками, соответствующими CMYK, также называют печатью триадными красками[11].

Цвет в CMYK зависит не только от спектральных характеристик красителей и от способа их нанесения, но и их количества, характеристик бумаги и других факторов. Фактически, цифры CMYK являются лишь набором аппаратных данных для фотонаборного автомата или CTP и не определяют цвет однозначно.

Так, исторически в разных странах сложилось несколько стандартизованных процессов офсетной печати. Сегодня это американский, европейский и японский стандарты для мелованной и немелованной бумаг. Именно для этих процессов разработаны стандартизованные бумаги и краски (например, стандарты ECI). Для них же созданы соответствующие цветовые модели CMYK, которые используются в процессах цветоделения. Однако многие типографии, в которых работают специалисты с достаточной квалификацией (или способные на время пригласить такого специалиста), нередко создают профиль, описывающий печатный процесс конкретной печатной машины с конкретной бумагой.

1. Цветовая модель HSV

Рассмотренные ранее цветовые модели RGB и CMY(K) весьма просты в плане аппаратной реализации, но у них есть один существенный недостаток. Человеку очень тяжело оперировать цветами, заданными в этих моделях, так как человек, описывая цвета, пользуется не содержанием в описываемом цвете базовых составляющих, а несколько иными категориями.

Чаще всего люди оперируют следующими понятиями: цветовой тон, насыщенность и светлота. При этом, говоря о цветовом тоне, обычно имеют в виду именно цвет. Насыщенность показывает насколько описываемый цвет разбавлен белым (розовый, например, это смесь красного и белого). Понятие светлоты наиболее сложно для описания, и с некоторыми допущениями под светлотой можно понимать интенсивность света[10].

Дадим определение цветового пространства HSV или HSB (англ. Hue, Saturation, Brightness — тон, насыщенность, яркость). Это цветовая модель, в которой координатами цвета являются:

* Hue — цветовой тон, (например, красный, зелёный или сине-голубой). Варьируется в пределах 0—360°, однако иногда приводится к диапазону 0—100 или 0—1.
* Saturation — насыщенность. Варьируется в пределах 0—100 или 0—1. Чем больше этот параметр, тем «чище» цвет, поэтому этот параметр иногда называют чистотой цвета. А чем ближе этот параметр к нулю, тем ближе к нейтральному серому цвету.
* Value (значение цвета) или Brightness — яркость. Также задаётся в пределах 0—100 или 0—1.

Если рассмотреть проекцию RGB-куба в направлении диагонали белый-чёрный, то получится шестиугольник:

1. Заключение

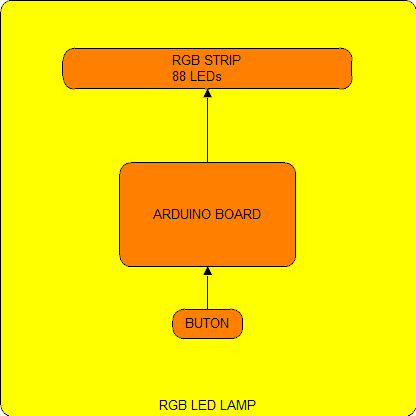
Вторая глава была посвящена изучению теории восприятия цвета человеческим глазом, а также представление цвета в технике. Было рассмотрено понятие цветового пространства, а также самые распространенные цветовые пространства. Рассмотренные цветовые пространства можно разделить на два типа – аддитивные и субтрактивные. Яркий пример аддитивного цветового пространства – RGB. Эта система применяется для объяснения восприятия цвета глазом человека и является самой важной для понимания. Субтрактивное цветовое пространство и CMYK получило широкое применение в печати. Важное отличие – на белой бумаге уже есть белый цвет и для получения необходимого цвета необходимо лишь отсечь все лишние цвета. Наконец цветовое пространство HSV является самым удобным при работе с цветами в различных графических редакторах. В удобстве этой системы мы убедимся при написании программы в данной работе. Однако не стоит забывать, что система пространство HSV является промежуточной и для отображения цветов необходимо переходить в пространство RGB.

Глава III. РАЗРАБОТКА ДЕКОРАТИВНОГО СВЕТИЛЬНИКА

В результате практической части данной работы был разработан декоративный светильник, удовлетворяющий поставленным во второй главе условиям. Для создания законченного устройства была разработана схема электрическая принципиальная, корпус изделия со световым рассеивателем и программное обеспечение.

1. Принципиальная схема

Принципиальная схема разрабатываемого устройства состоит из трех основных компонентов - платы Arduino, светодиодной ленты RGB и механической кнопки.



**Рис 3.1.** Принципиальная схема светильника.

Принципиальная схема проста в понимании. В разработке использовались только выводы микроконтроллера, включенные в цифровом режиме. Кнопка соединена с ножкой микроконтроллера, включенной в режим цифрового входа с включенной внутренней подтяжкой к положительному напряжению питания. Программа будет отслеживать потенциал на выбранной ножке и переключать режимы в зависимости от нажатий. Второй использованный вывод микроконтроллера включен в режим цифрового выхода. Через этот пин будет происходить загрузка данных в светодиодную ленту. Процедура передачи данных в светодиодную ленту реализована с помощью сторонней библиотеки.

1. Корпус светильника

Нижняя часть светильника представляет собой конструктив, за основу которого взята канализационная муфта диаметром 110 мм. Выбранная муфта лучше всего подошла для конструирования. На рисунке ниже фотография такой муфты.



**Рис 3.2.** Канализационная муфта диаметром 110 мм

Муфта не дорогая, а полипропилен, из которого она изготовлена, позволяет без проблем сверлить в ней необходимые отверстия.

Верхняя и нижняя сторона муфты закрыты пластиковой заглушкой. Заглушка также выполнена из полипропилена.



**Рис 3.3.** Полипропиленовая заглушка.

Первоначально предполагалось разместить внутри корпуса блок внешний питания, но эксперименты со светодиодной лентой показали, что потребляемый ток способен обеспечить блок зарядного устройства от мобильного телефона. Было принято решение использовать зарядное устройство от мобильного телефона в качестве преобразователя переменного напряжения сети 220 Вольт в постоянное напряжения 5 Вольт для питания платы Arduino, а также светодиодной ленты.

Кроме блока питания в нижней части светильника, выполненной из пластиковой муфты, расположена плата Arduino. Плата закреплена винтами диаметром 3 мм.



**Рис 3.4.** Фотография расположения платы Arduino в корпусе.

Верхняя половина светильника состоит из картонной трубки, металлической конструкции и бумажного рассеивателя. На картонной трубке расположены светодиоды. Металлическая конструкция удерживает рассеиватель.

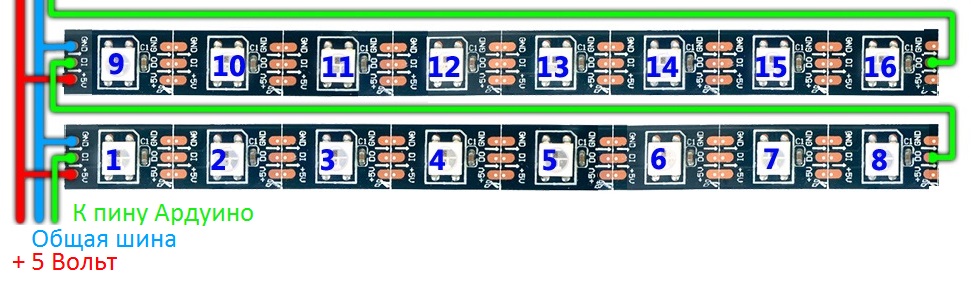
Изначально в наличии было 3 светодиодные ленты длиной по одному метру и по 30 светодиодов на каждой ленте. Расстояние между светодиодами 33 мм. Длина окружности картонной трубки, имевшаяся в наличии, вместила ровно восемь светодиодов. Для светильника я нарезала ленты на 11 отрезков по 8 светодиодов на каждом. Три отрезка пришлось спаять и двух частей. Получившиеся отрезки наклеила на картонную трубку при помощи термо клея.

Из-за большого расстояния между светодиодами на ленте(33 мм) отрезки располагаем со смещением по горизонтали на половину расстояния между светодиодами. Получается, что светодиоды расположены в шахматном порядке. Это вынужденная мера – без этого светодиодная матрица выглядела бы совсем несимметричной. Однако шахматное расположение, как показала практика, не совсем удобно в некоторых режимах. Подробнее об этом недостатке будет рассказано в части описания программы.



**Рис 3.5.** Фотография расположения светодиодов на картонной трубке.

После того как кусочки ленты были закреплены на картонной трубке их пришлось заново соединить последовательно в одну ленту. При соединении необходимо учитывать направление ленты. На ленте с одной стороны управляющий пин промаркирован DI – digita input, а с другой стороны DO – digital output. При последовательном соединении надо подключить DI первого отрезка к пину Arduino, а DO к DI следующего отрезка. Схема последовательного подключения представлена на рисунке.



**Рис. 3.6.** Схема последовательного подключения светодиодной ленты.

В результате всех действий получается светодиодная матрица шириной 8 и высотой 11 светодиодов. Управляющая программа учитывает размеры получившейся матрицы - на этом строятся некоторые эффекты.

Последним этапом работы была установка рассеивателя. Рассеиватель был изготовлен из плотной бумаги формата А3. Вычислила длину окружности рассеивателя и склеила из бумаги трубку, предварительно отмерив необходимую высоту рассеивателя. Верхний и нижний края подгибаем и прижимаем металлической конструкцией держателя. Бумага хорошо рассеивает свет даже при близком расположении к ней светодиодов.

1. Разработка программы управления

Программное обеспечение для данной работы было написано в программной среде Arduino версии 1.8.12. Язык программирования внешне не отличается от C/C++. На официальном сайте Arduino [12] этот язык называется – Язык программирования Arduino (основан на программной среде с открытым исходным кодом ***Wiring***). Компилируются и собираются программы при помощи компилятора avr-gcc. Главный плюс среды Arduino это отлаженная работа с библиотеками. Интернет сообщества разработчиков делятся своим опытом и с целью взаимопомощи и ускорения процессов программирования. На сайте Arduino даже есть подробные рекомендации к созданию библиотек[13]. В рамках текущего проекта мною были использованы несколько библиотек:

1. Библиотека работы со встроенной энергонезависимой памятью EEPROM.
2. Библиотека передачи данных по одной линии от микроконтроллера к светодиодной ленте FastLED-3.2.9.
3. Библиотека работы с кнопкой GyverButton.

Как было сказано выше, программа для платы Arduino делится на две подпрограммы:

1. Функция *setup()* - инициализация всех глобальных переменных и ножек микроконтроллера;
2. Функция loop() - основной цикл работы программы;

За пределами этих функций определяются глобальные переменные и описываются другие функции необходимые для работы программы.

* 1. Глобальные переменные

В этой части необходимо определить необходимое для полной работы программы количество глобальных переменных. Конечным результатом работы программы будет отображение какой-либо информации светодиодами. Используемая в проекте библиотека FastLED предлагает свой класс для хранения и обработки значений цветов для светодиодов – тип CRGB. В программном коде создается массив элементов этого класса. Количество элементов массива равно количеству светодиодов в светильнике. Для начала найдем в файлах библиотеки описание этого класса:

struct CRGB {

union {

struct {

union {

uint8\_t r;

uint8\_t red;

};

union {

uint8\_t g;

uint8\_t green;

};

union {

uint8\_t b;

uint8\_t blue;

};

};

uint8\_t raw[3];

};

};

**Листинг 3.1.** Описание класса CRGB.

Как видно из описания, каждый объект такого класса содержит три байта информации. Каждый байт описывает одну из цветовых компонент светодиода – красную, зелёную и синюю. Кроме того, к элементам можно обращаться напрямую через массив элементов raw[3]. В классе также описаны некоторые процедуры работы с элементами класса.

В следующем листинге представлен пример использования директивы #define для определения размерности матрицы светодиодов.

#define WIDTH 8

#define HEIGHT 11

#define NUM\_LEDS WIDTH \* HEIGHT

**Листинг 3.2.** Определение размерности матрицы светодиодов.

При желании можно применить программный код для другого количества светодиодов. Эти определения будут использоваться во всей программе при работе со светодиодами. Первое применение одного из определений – создание массива объектов из 88 светодиодов.

CRGB leds[NUM\_LEDS];

**Листинг 3.3.** Создание массива обьектов.

Для работы с кнопкой создаем объект класса GButton. Атрибуты метода создания объекта это:

1. Номер пина к которому подключена кнопка (BTN\_PIN);
2. Активация внутренней подтяжки к питанию (HIGH\_PULL);
3. Исходное состояние кнопки – разомкнутое состояние (NORM\_OPEN)

GButton touch(BTN\_PIN, HIGH\_PULL, NORM\_OPEN);

**Листинг 3.4.** Создание обьекта для работы с кнопкой.

Для хранения настроек режимов работы в программе объявляется массив структур режимов. Каждый режим характеризуется тремя характеристиками: общая яркость, скорость выполнения эффекта, масштаб эффекта. Не все режимы используют каждый и этих параметров, но для единообразия это оказалось удобно. Для определения количества режимов используется директива #define. При изменении номера режима в сторону увеличения, программа остановится на определенном ранее количестве. Таким образом, всегда можно описать свой режим и добавить его в обработчик режимов или уменьшить количество режимов.

**struct** {

byte brightness = 50;

byte speed = 100;

byte scale = 20;

} modes[MODE\_AMOUNT];

**Листинг 3.5.** Объявление массива структур режимов.

* 1. Начальная инициализация

Правильная начальная инициализация переменных и объектов это залог хорошей работы всей программы. Программная среда Arduino предлагает использовать специальную функцию *setup()* для начальной инициализации объектов. Эта функция вызывается один раз при старте программы.

Первая часть функции настраивает объект класса FastLED при помощи процедуры *FastLED*.*addLeds*. Необходимо указать тип используемых светодиодов - WS2812B в нашем случае. Следующий параметр – номер пина, к которому подключена лента - LED\_PIN. Этот номер определён с помощью директивы #define выше в программе. Такое решение позволяет оперативно менять номер при настройке параметров платы. Также необходимо указать порядок цветов при передаче данных в ленту. Данные в ленту передаются последовательно – бит за битом. Для каждого светодиода необходимо передать три байта или двадцать четыре бита. Для разных лент эта последовательность может отличаться. В нашем случае это GRB – то есть сначала передается байт зеленого, за ним красный, а последним синий. Ниже представлен листинг части функции *setup(),* отвечающей за настройку параметров библиотеки светодиодной ленты.

//Настройка ленты

FastLED.addLeds<WS2812B, LED\_PIN, COLOR\_ORDER>(leds, NUM\_LEDS);

FastLED.setBrightness(BRIGHTNESS);

//Установка максимального тока блока питания

**if** (CURRENT\_LIMIT > 0)

FastLED.setMaxPowerInVoltsAndMilliamps(5, CURRENT\_LIMIT);

FastLED.clear();//Установка всех значений в ноль

FastLED.show();//Запись значений в светодиодную ленту для отображения

//

**Листинг 3.6.** Настройка библиотеки FastLED.

В проекте также используется библиотека работы с кнопкой. Для правильной работы кнопки устанавливаем временные интервалы между кратковременными нажатиями, а также тайм-аут окончания нажатия.

//настройка параметров кнопки

touch.setStepTimeout(100);//интервал между кликами

touch.setClickTimeout(500; )//тайм-аут

**Листинг 3.7.** Настройка параметров кнопки.

После первого включения и до того как настройки будут сохранены все режимы конфигурируются одинаково – настройками по умолчанию. Конфигурация по умолчанию происходит при определении массива структур. Значения настроек по умолчанию были подобраны по личному усмотрению. При каждом сохранении настроек в первый байт энергонезависимой памяти (с адресом ноль) записывается число сто два. Это своеобразный признак того, что настройки в памяти это не случайный набор чисел.

**if** (EEPROM.read(0) == 102) {// если было сохранение настроек, то восстанавливаем их из EEPROM

currentMode = EEPROM.read(1);

**for** (byte x = 0; x < MODE\_AMOUNT; x++) {

modes[x].brightness = EEPROM.read(x \* 3 + 11); // (2-10 байт - резерв)

modes[x].speed = EEPROM.read(x \* 3 + 12);

modes[x].scale = EEPROM.read(x \* 3 + 13);

}

}

**Листинг 3.8.** Чтение настроек из энергонезависимой памяти.

Кроме всего вышеперечисленного в функции начальной инициализации происходит настройка последовательно порта, при помощи которого происходила отладка программы. Так как в программе есть эффекты, которые требуют генерации случайных чисел, тут же вызывается настройка генератора псевдослучайных чисел. В листинге ниже представлен полный код функции *setup().*

**void** setup() {

//Настройка ленты

FastLED.addLeds<WS2812B, LED\_PIN, COLOR\_ORDER>(leds, NUM\_LEDS);

FastLED.setBrightness(BRIGHTNESS);

//Установка максимального тока блока питания

**if** (CURRENT\_LIMIT > 0)

FastLED.setMaxPowerInVoltsAndMilliamps(5, CURRENT\_LIMIT);

FastLED.clear();//Установка всех значений в ноль

FastLED.show();//Запись значений в светодиодную ленту для отображения

//

//настройка параметров кнопки

touch.setStepTimeout(100);//интервал между кликами

touch.setClickTimeout(500; )//тайм-аут

//

//Функция RandomSeed() инициализирует генератор псевдослучайных чисел.

randomSeed(analogRead(0));

//

//�нициализация последовательного порта

Serial.begin(9600);

  #ifdef DEBUG\_PRINT

delay(1000);

Serial.println();

Serial.println("LED lamp start");

  #endif

//

//Чтение Настроек из EEPROM

// если было сохранение настроек, то восстанавливаем их из EEPROM

**if** (EEPROM.read(0) == 102) {

currentMode = EEPROM.read(1);

**for** (byte x = 0; x < MODE\_AMOUNT; x++) {

modes[x].brightness = EEPROM.read(x \* 3 + 11); // (2-10 байт - резерв)

modes[x].speed = EEPROM.read(x \* 3 + 12);

modes[x].scale = EEPROM.read(x \* 3 + 13);

}

}

}

**Листинг 3.9.** Полный код функции *setup().*

* 1. Работа основной части программы

В программной среде Arduino основной цикл программы определяется функцией *loop().* Для лучшего понимания можно себе представить, что содержимое этой функции вставляется в один из вариантов бесконечного цикла. Другой вариант – вызов функции *loop()* внутри того же бесконечного цикла. В моей программе содержимое этой функции это последовательный вызов двух других функций. Одна из них это **Менеджер эффектов**, а вторая **Отработка нажатий на кнопку**. Обе эти функции будут описаны ниже.

**void** loop() {

effectsTick();//Менеджер эффектов

buttonTick();//Отработка нажатий на кнопку

}

**Листинг 3.10.** Полный код функции *loop().*

* 1. Менеджер эффектов

Основная часть работы светильника это индикация текущего режима. Программный Менеджер эффектов индицирует текущий режим в зависимости от состояния переменной currentMode. Эта переменная является глобальной для проекта. Её начальное состояние определяется при создании. Если настройки были сохранены, то номер текущего режима загрузится из энергонезависимой памяти в функции setup(). Номер текущего режима можно изменить быстрым двойным нажатием на кнопку управления. Выбор режима для отображения в программе производится при помощи оператора принятия решений switch. Номер режима, хранящийся в переменной currentMode должен быть в интервале от 0 до значения, определенного директивой define.

#define MODE\_AMOUNT 9//количество режимов

**Листинг 3.11.** Определение количества режимов*.*

В случае если оператор *switch()* по каким-то причинам не нашел совпадения, то программа перейдет к оператору default и будет индицироваться режим по умолчанию – белая лампа (*whiteLamp();*).

Важно отметить как именно скорость текущего режима (в программном коде это *modes[currentMode].speed*) влияет на работу. В этом мне очень помогла функция, которая является стандартной для Программной среды Arduino - *millis().* Эта функция возвращает количество миллисекунд с момента начала выполнения текущей программы на плате Arduino. Это количество сбрасывается на ноль, в следствие переполнения 32-х битного счетчика, приблизительно через 50 дней. Этого времени с запасом хватает для работы программы. Моя программа засекает текущий момент при каждом вызове функции *effectsTick()* и дожидается когда пройдет интервал времени, определённый в параметре скорости текущего режима. Если необходимое время еще не прошло, то происходит выход из подпрограммы. В случае, когда необходимый интервал времени прошел, происходит вызов функции отображения текущего режима и сохранение текущего значения времени для ожидания следующего момента времени.

**void** effectsTick() {

uint32\_t CurrentTime = millis() - effTimer;

**if** (ONflag && (CurrentTime >= modes[currentMode].speed)) {

effTimer = millis();//сохранение текущего времени

//Вызов одной из функций индикации

**switch** (currentMode) {

**case** SPARKLES\_MODE:

sparklesRoutine();

**break**;

**case** FILL\_MODE:

fillRoutine();

**break**;

**case** RNBOW\_HORION\_MODE:

rainbowHorizontal();

**break**;

**case** RNBOW\_VERTIC\_MODE:

rainbowVertical();

**break**;

**case** COLOR\_S\_MODE:

colorsRoutine();

**break**;

**case** COLOR\_MODE:

colorRoutine();

**break**;

**case** SNOWFALL\_MODE:

snowRoutine();

**break**;

**case** MATRIX\_MODE:

matrixRoutine();

**break**;

**case** WHITE\_LAMP\_MODE:

whiteLamp();

**break**;

**default**:

whiteLamp();

**break**;

}

FastLED.show();//Запись массива индикации в Ленту

}

}

**Листинг 3.12.** Подпрограмма индикации выбранного режима*.*

Остановимся более подробно на первом режиме индикации – режим разноцветных вспышек. В проекте подпрограмма этого режима называется *sparklesRoutine.* Этот режим полностью построен на системной для программной среды Arduino функции генератора псевдослучайных чисел. Активация генератора псевдослучайных чисел была описана выше. Вызов функции активации производится внутри функции *setup*. Ниже представлен листинг подпрограммы.

**void** sparklesRoutine() {

**for** (byte i = 0; i < modes[SPARKLES\_MODE].scale; i++) {

byte x = random(0, WIDTH);

byte y = random(0, HEIGHT);

**if** (getPixColorXY(x, y) == 0)

leds[getPixelNumber(x, y)] = CHSV(random(0, 255), 255, 255);

}

fader(70);

}

**Листинг 3.13.** Подпрограмма индикации режима разноцветных вспышек*.*

Для того чтоб не загромождать листинг я удалила все комментарии из исходного кода. Описание работы подпрограммы представлено ниже.

Первая часть подпрограммы это цикл for по переменной i. Количество проходов цикла зависит от значения параметра scale описываемого режима. В программе мы обращаемся к параметру scale элемента массива режимов с индексом *SPARKLES\_MODE*. Параметр scale определяет, сколько светодиодов за один вызов функции будет включено. Внешне это выглядит как прореживание активных светодиодов. Тройное нажатие на кнопку с последующим удержанием меняет этот параметр.

В каждом проходе цикла определяются два случайных числа. Эти числа определяют координаты светодиода в двумерном массиве. В программе задается диапазон чисел для генератора случайных чисел. Для первого числа диапазон от нуля до параметра WIDTH – координата x. Для второго числа задается диапазон от нуля до параметра HEIGHT – координата y. Полученные координаты определяют светодиод ленты (матрицы) с которым мы работаем. По задумке этот режим должен выглядеть как вспыхивание светодиодов производным цветом и плавное затухание. Поэтому если светодиод, определенный в данном проходе координатами ещё не погас, то программа не меняет его значение. Если же значения всех трёх цветов выбранного светодиода равны нулю, то в него записываются новый цвет с максимальной яркостью и насыщенностью. Цвет представлен в цветовом пространстве HSV. Компонента Hue – случайное число, а компоненты Saturation и Value записывается число 255. Это и есть максимальная яркость и насыщенность. После этого вызывается функция fader. Эта функция осуществляет плавное гашение всех светодиодов ленты.

Описанный режим использует все три переменных параметра. Параметр brightness влияет на общую яркость светодиодов ленты. Параметр speed влияет на то, с какой частотой происходит вызов функции отображения режима. Частота вызова влияет на скорость затухания светодиодов и как следствие на общую скорость режима. Параметр scale определяет, сколько светодиодов зажигается за один проход. Внешне это прореживает активные светодиоды.

* 1. Отработка нажатий на кнопку управления

Отработка нажатий на кнопку это вторая половина основного цикла программного кода. Работа с кнопкой в проекте основана на библиотеке GyverButton. Данная библиотека написана для многофункциональной отработки нажатия кнопки. Возможности данной библиотеки представлены в следующем списке:

1. Работа с нормально замкнутыми и нормально разомкнутыми кнопками.
2. Работа с подключением PULL\_UP и PULL\_DOWN.
3. Опрос кнопки с программным анти-дребезгом контактов.
4. Настраиваемое время антидребезга.
5. Отработка нажатия, удерживания, отпускания, клика по кнопке.
6. Настраиваемый таймаут удержания.
7. Отработка одиночного, двойного и тройного нажатия (вынесено отдельно).
8. Отработка любого количества нажатий кнопки (функция возвращает количество нажатий).
9. Настраиваемый таймаут повторного нажатия/удержания.
10. Функция изменения значения переменной с заданным шагом и заданным интервалом по времени.
11. Возможность опрашивать не кнопку, а напрямую давать величину (все возможности библиотеки для матричных и резистивных клавиатур).

Этих возможностей с избытком хватило для реализации поставленных задач. Кнопка находится на верхней крышке изделия и является единственным интерфейсом светильника. В процессе разработки приходили новые идеи и функцинал кнопки сильно увеличился. Ниже представлен список изменяемых кнопкой параметров и то, каким способом это делается.

1. Включение и выключение светильника – кратковременное одиночное нажатие на кнопку без удержания.
2. Изменение порядкового номера режима в сторону увеличения - кратковременное двойное нажатие на кнопку без удержания.
3. Изменение порядкового номера режима в сторону уменьшения - кратковременное тройное нажатие на кнопку без удержания.
4. Сохранение текущих настроек в энергонезависимую память – четырех кратное нажатие без удержания.
5. Сброс настроек на настройки по умолчанию – пяти кратное нажатие без удержания.
6. Изменение яркости текущего параметра – удержание кнопки после одиночного нажатия. Направление изменения в сторону уменьшения или увеличения меняется после каждого вызова режима.
7. Изменение скорости текущего параметра – удержание кнопки после двойного нажатия. Направление изменения в сторону уменьшения или увеличения меняется после каждого вызова режима.
8. Изменение масштаба текущего параметра – удержание кнопки после тройного нажатия. Направление изменения в сторону уменьшения или увеличения меняется после каждого вызова режима.

Одиночное нажатие это самое простое, что можно сделать с кнопкой. Поэтому на одиночное нажатие я возложила функцию включения/выключения светильника. При каждом одиночном нажатии состояние переменной логического типа ONflag меняется на противоположное. В этой переменной хранится текущее состояние светильника. В выключенном состоянии не работает подпрограмма Менеджер эффектов, а также программа не реагирует ни на какие другие нажатия на кнопку – только повторное одиночное нажатие для включения.

//одиночное нажатие - ВКЛ/Выкл

**if** (touch.isSingle()) {

**if** (ONflag) {//если сейчас включено, то выключаем

ONflag = **false**;

changePower();

} **else** {//если же выключено, то включаем

ONflag = **true**;

changePower();

}

}

**Листинг 3.14.** Отработка одиночного нажатия*.*

Для того чтобы включение и выключение светильника выглядело красиво, яркость изменяется не резко, а плавно. Плавное гашение и включение светодиодов реализовано в функции changePower. При изменении общего состояния светильника один раз вызывается функция отображения текущего режима. Вызов функции отображения режима нужен для заполнения массива индикации новыми значениями. После этого при включении плавно увеличивается общая яркость всей ленты от нуля до яркости текущего режима. При выключении яркость уменьшается от текущего значения до нуля. Первая половина функции changePower, которая выполняется в случае, если переменная ONflag принимает истинное значение, описывает плавное включение. Вторая половина выполняется в противном случае и описывает плавное выключение.

void changePower() { // плавное включение/выключение

if (ONflag) // плавное включение

effectsTick();

**for** (**int** i = 0; i < modes[currentMode].brightness; i += 8) {

FastLED.setBrightness(i);

delay(1);

FastLED.show();

}

FastLED.setBrightness(modes[currentMode].brightness);

delay(2);

FastLED.show();

}

**else** //Плавное выключение

{

effectsTick();

**for** (**int** i = modes[currentMode].brightness; i > 8; i -= 8) {

FastLED.setBrightness(i);

delay(1);

FastLED.show();

}

FastLED.clear();

delay(2);

FastLED.show();

}

}

**Листинг 3.15.** Плавное включение и выключение светильника*.*

Изменение номера текущего режима в сторону увеличения, проиводится двойным нажатием. Используемая библиотека работы с кнопкой имеет в своём арсенале методы, которые возвращают истинное значение, если производилось двойное нажатие и тройное нажатие без удержания. Название этих методов isDouble и isTriple. Если метод двойного нажатия возвращает истинное значение, то производится инкремент номера текущего режима с проверкой максимального номера режима. Если номер больше определенного, то устанавливается номер первого режима равный нулю. Листинг отработки двойного нажатия представлен ниже.

//двойное нажатие - изменение номера режима вверх

**if** (touch.isDouble()) {

**if** (++currentMode >= MODE\_AMOUNT)

currentMode = 0;

FastLED.setBrightness(modes[currentMode].brightness);

FastLED.clear();

delay(1);

}

**Листинг 3.16.** Отработка двойного нажатия*.*

Если метод тройного нажатия возвращает истинное значение, то производится декремент номера текущего режима с проверкой минимального номера режима. Если номер меньше нуля, то устанавливается номер последнего режима. Листинг отработки тройного нажатия представлен ниже.

//тройное нажатие - изменение номера режима вниз

**if** (touch.isTriple()) {

**if** (--currentMode < 0)

currentMode = MODE\_AMOUNT - 1;

FastLED.setBrightness(modes[currentMode].brightness);

FastLED.clear();

delay(1);

}

**Листинг 3.17.** Отработка тройного нажатия*.*

В процессе использования светильника можно менять настройки каждого режима. Для каждого из имеющихся режимов можно отдельно настроить яркость, скорость и масштаб отображаемого эффекта. Также можно изменить номер отображаемого режима. Если не сохранить все эти изменения, то после выключения питания светильник включится с настройками по умолчанию, либо с такими настройками, которые были сохранены в предыдущий раз. Для сохранения текущих настроек в энергонезависимую память используется четырех кратное нажатие на кнопку управления. Для сохранения ресурса энергонезависимой памяти по количеству записей, перезаписываются только ячейки, содержимое которых отличается от нового. Признаком того, что настройки в памяти были сохранены хотя бы один раз, является запись числа сто два в нулевой ячейке.

// если было 4-х нажатие на кнопку, то производим сохранение параметров

**if** ((hasClick) && (touch.getClicks() == 4)) {

**if** (EEPROM.read(0) != 102) EEPROM.write(0, 102);//это признак того, что настройки сохранялись хотя бы раз

**if** (EEPROM.read(1) != currentMode) EEPROM.write(1, currentMode); // запоминаем текущий эфект

**for** (byte x = 0; x < MODE\_AMOUNT; x++) {// сохраняем настройки всех режимов

**if**(EEPROM.read(x \* 3 + 11) != modes[x].brightness)//новое значение

EEPROM.write(x \* 3 + 11, modes[x].brightness);

**if**(EEPROM.read(x \* 3 + 12) != modes[x].speed)//новое значение

EEPROM.write(x \* 3 + 12, modes[x].speed);

**if**(EEPROM.read(x \* 3 + 13) != modes[x].scale)//новое значение

EEPROM.write(x \* 3 + 13, modes[x].scale);

}

// индикация сохранения

ONflag = **false**;

changePower();

delay(200);

ONflag = **true**;

ChangePower ();

}

**Листинг 3.18.** Отработка четырехкратного нажатия с сохранение настроек*.*

По завершении процедуры сохранения светильник выключится и включится снова с паузой в двести миллисекунд. Эта процедура является обратной связью с пользователем – своего рода подтверждением того, что настройки сохранены.

В процессе разработки светильника и многократным изменениям настроек становилось непонятно, какие настройки установлены в данный момент. Некоторые параметры получалось загнать в предельные значения. В этих случаях использование светильника не приносило никакого удовольствия. Тогда и пришла мысль о необходимости ввести режим возврата к настройкам по умолчанию. Настройки по умолчанию это такие настройки, при которых все параметры установлены примерно в середину всего рабочего диапазона. Единственным возможным способом реализации вызова этого режима оставалось пятикратное нажатие. Это не очень удобно, но с другой стороны у пользователя не получится сделать это случайно. Окончание процедуры установки настроек по умолчанию, как и сохранение настроек, сопровождается дополнительной индикацией. В этом случае это трёхкратное моргание всеми светодиодами светильника.

// если было пятикратное нажатие на кнопку, то производим переход в настройки по умолчанию

// и сохраняем настройки

**if** ((hasClick) && (touch.getClicks() == 5)) {

//set all modes to default

**for** (byte x = 0; x < MODE\_AMOUNT; x++) {

modes[x].brightness = 200;

modes[x].speed = 150;

modes[x].scale = 20;

}

currentMode = 0;

**if** (EEPROM.read(0) != 102) EEPROM.write(0, 102);

**if** (EEPROM.read(1) != currentMode) EEPROM.write(1, currentMode); // запоминаем текущий эфект

**for** (byte x = 0; x < MODE\_AMOUNT; x++) { // сохраняем настройки всех режимов

**if** (EEPROM.read(x \* 3 + 11) != modes[x].brightness) EEPROM.write(x \* 3 + 11, modes[x].brightness);

**if** (EEPROM.read(x \* 3 + 12) != modes[x].speed) EEPROM.write(x \* 3 + 12, modes[x].speed);

**if** (EEPROM.read(x \* 3 + 13) != modes[x].scale) EEPROM.write(x \* 3 + 13, modes[x].scale);

}

// индикация сохранения - triple switch on/off

ONflag = **false**;//if ONflag is false - smoothly blow out all LED's

changePower();

delay(500);

ONflag = **true**;//if ONflag is true - smoothly burn up all LED's

changePower();

delay(500);

// индикация сохранения - triple switch on/off

ONflag = **false**;//if ONflag is false - smoothly blow out all LED's

changePower();

delay(500);

ONflag = **true**;//if ONflag is true - smoothly burn up all LED's

changePower();

delay(500);

}

**Листинг 3.19.** Отработка пятикратного нажатия с установкой настроек по умолчанию*.*

Следующим этапом разработки программного обеспечения стала отработка изменения параметров режимов. Эти изменения были реализованы с помощью различных комбинаций нажатий на кнопку с удержанием. Для полной работы светильника необходимо менять три параметра – яркость, скорость и масштаб эффектов. Соответствующий параметр меняется для выбранного в данный момент режима. Для изменения яркости я использовала комбинацию единократного нажатия на кнопку с удержанием при включенном светильнике. При первом после подачи питания единократном нажатии с удержанием яркость будет увеличиваться. При следующем входе в режим настройки яркости направление изменения изменится на противоположное. Таким способом можно уменьшить яркость.

Аналогичным образом изменяются два других параметра каждого режима. Для изменения скорости отображаемого режима используется двойное нажатие с удержанием. А для изменения масштаба выбранного параметра нужно трижды нажать на кнопку управления и удерживать её нажатой.

Интервал изменения настроек светильника, при удержании кнопки в нажатом положении, определён в параметрах используемой библиотеки. В текущей реализации он составляет четыреста миллисекунд. Другими словами при удержании кнопки настройки режима будет изменяться каждые четыреста миллисекунд. Шаг изменения настроек был выбран в одну двадцать пятую часть от всего диапазона. Для проверки диапазона настроек я использовала библиотечную функцию *constrain(x,a,b)*. Эта функция проверяет, что значение параметра x входит в диапазон чисел от a до b. Если x меньше числа а, то возвращается число а. Если же число x больше числа b, то возвращается число b.

//Обработка режимов настройки с удержанием кнопки

**if** (touch.isHolded()) { // изменение яркости при удержании кнопки

brightDirection = !brightDirection;//каждый раз меняем направление коррекции

numHold = 1;

}

**if** (touch.isHolded2()) { // изменение скорости "speed" при двойном нажатии и удержании кнопки

speedDirection = !speedDirection;//каждый раз меняем напрвление коррекции

numHold = 2;

}

**if** (touch.isHolded3()) { // изменение масштаба "scale" при тройном нажатии и удержании кнопки

scaleDirection = !scaleDirection;//каждый раз меняем напрвление коррекции

numHold = 3;

}

//следующее изменение параметров с удержанием

//только после истечения необходимого интервала времени

//это нудно для того чтоб параметры не изменялись слишком быстро

**if** (touch.isStep()) {

**if** (numHold != 0)

numHold\_Timer = millis();

**switch** (numHold) {

**case** 1://изменение яркости

modes[currentMode].brightness = constrain(modes[currentMode].brightness + (modes[currentMode].brightness / 25 + 1) \* (brightDirection \* 2 - 1), 1 , 255);

**break**;

**case** 2://изменение скорости

modes[currentMode].speed = constrain(modes[currentMode].speed + (modes[currentMode].speed / 25 + 1) \* (speedDirection \* 2 - 1), 1 , 255);

**break**;

**case** 3://изменение масштаба

modes[currentMode].scale = constrain(modes[currentMode].scale + (modes[currentMode].scale / 25 + 1) \* (scaleDirection \* 2 - 1), 1 , 255);

**break**;

}

}

**Листинг 3.20.** Отработка нажатий с удержанием кнопки*.*

1. Заключение

В третьей главе был описан весь процесс разработки декоративного светильника. Работа светильника основана на ленте светодиодов RGB. В процессе разработки были изучены возможности использования сторонних библиотек в разработке программного обеспечения. Использование библиотеки работы со светодиодной намного упростило и ускорило процесс разработки светильника. Использованная библиотека позволяет работать с различными видами светодиодных лент. Для использования выбранной библиотеки осталось только правильно настроить все параметры. Этого не получилось бы без хорошей документации на выбранную библиотеку. Можно сделать вывод, что при разработке собственной библиотеки необходимо уделить особое внимание качественному описанию интерфейса библиотеки. Без этого использование библиотеки другими программистами станет просто невозможным.

В конструировании корпуса помогло использование различных доступных технических материалов. Корпус изделия был собран из пластиковой муфты, а блок питания представляет собой простое зарядное устройство от мобильного телефона. Рассеивателем света послужил лист бумаги формата А3.

Большое внимание было уделено программному обеспечению светильника. Текст программы написан с необходимыми комментариями, а названия всех функций и переменных соответствуют их функциональному назначению.

При помощи одной кнопки управления было реализовано переключение и настройка режимов, а также сохранение настроек.

ВЫВОДЫ И РЕКОМЕНДАЦИИ

В данной работе были рассмотрены компоненты и подходы к созданию устройств декоративного освещения. В работе представлены наиболее распространенные на рынке компоненты для построения устройств декоративного освещения, такие как платы Arduino, построенные на базе микроконтроллеров AVR и светодиодные ленты, состоящие из RGB светодиодов.

В ходе работы были проанализированы различные архитектуры микропроцессоров и способы организации аппаратного обеспечения для достижения лучших результатов с точки зрения экономичности и простоты разработки.

В результате работы был сконструирован декоративный светильник, управляемый микроконтроллером(AtMega32U4) на плате Arduino(Leonardo) и использующий ленты RGB светодиодов в качестве источника света. Готовый светильник оправдал ожидания – яркости достаточно для демонстрации и в дневное время, а реализованные световые эффекты позволяют оценить потенциальные возможности светодиодных лент.

На базе полученных результатов можно рекомендовать использование светодиодных лент, управляемых платами Arduino, для изготовления декоративных светильников с необычными световыми эффектами.

.

БИБЛИОГРАФИЯ

БИБЛИОГРАФИЯ

[1] Таненбаум Э., Остин Т., Архитектура компьютера. 6-е изд. - СПб.:Питер, 2017, стр. 69

[2] Arduino Commnication, https://www.tutorialspoint.com/arduino/arduino\_communication.htm, (посещено 05.06.2019)

[3] Arduino Leonardo, <http://arduino.ru/Hardware/ArduinoBoardLeonardo>

[4] Arduino Program Structure, https://www.tutorialspoint.com/arduino/arduino\_program\_structure.htm

[5] Arduino I/O Functions, https://www.tutorialspoint.com/arduino/arduino\_io\_functions.htm

[6] Arduino Pulse Width Modulation, https://www.tutorialspoint.com/arduino/arduino\_pulse\_width\_modulation.htm

[7] Arduino Interrupts, https://www.tutorialspoint.com/arduino/arduino\_interrupts.htm

[8] Восприятие цвета мозгом www.tikkurila.ru/dlya\_professionalov/tsveta/vvedenie\_v\_teoriyu\_tsveta/vospriyatie\_tsveta\_mozgom

[9] Цветовая модель RGB https://ru.wikipedia.org/wiki/RGB

[10] О цветовых пространствах, https://habr.com/ru/post/181580/

[11] Цветовая модель CMYK,https://ru.wikipedia.org/wiki/CMYK

[12] Language Reference https://www.arduino.cc/reference/en

[13] Writing a Library for Arduino, https://www.arduino.cc/en/Hacking/LibraryTutorial

[14] Цветовая температура, <http://www.ylati.ru/czvetovaya_temperatura.html>

[15] Адресная светодиодная лента и её подключение к Arduin, https://ledjournal.info/spravochnik/adresnaja-svetodiodnaja-lenta.html

Приложение 1. Основной код программного обеспечения

|  |  |
| --- | --- |
|  |  |

**void** setup() {

//Настройка ленты

FastLED.addLeds<WS2812B, LED\_PIN, COLOR\_ORDER>(leds, NUM\_LEDS);

FastLED.setBrightness(BRIGHTNESS);

//Установка максимального тока блока питания

**if** (CURRENT\_LIMIT > 0)

FastLED.setMaxPowerInVoltsAndMilliamps(5, CURRENT\_LIMIT);

FastLED.clear();//Установка всех значений в ноль

FastLED.show();//запись значений в светодиодную ленту для отображения

//

//настройка параметров кнопки

touch.setStepTimeout(100);

touch.setClickTimeout(500);//интервал между кликами

//

//Функция RandomSeed() инициализирует генератор псевдослучайных чисел.

randomSeed(analogRead(0));

//

//Инициализация последовательного порта

Serial.begin(9600);

  #ifdef DEBUG\_PRINT

delay(1000);

Serial.println();

Serial.println("LED lamp start");

  #endif

//

//Чтение Настроек из EEPROM

// если было сохранение настроек, то восстанавливаем их из EEPROM

**if** (EEPROM.read(0) == 102) {

currentMode = EEPROM.read(1);

**for** (byte x = 0; x < MODE\_AMOUNT; x++) {

modes[x].brightness = EEPROM.read(x \* 3 + 11); // (2-10 байт - резерв)

modes[x].speed = EEPROM.read(x \* 3 + 12);

modes[x].scale = EEPROM.read(x \* 3 + 13);

}

}

}

**void** loop() {

effectsTick();//Менеджер эффектов

buttonTick();//Отработка нажатий на кнопку

}

Приложение 2. Менеджер эффектов

|  |  |
| --- | --- |
|  |  |

**void** effectsTick() {

uint32\_t CurrentTime = millis() - effTimer;

**if** (ONflag && (CurrentTime >= modes[currentMode].speed)) {

effTimer = millis();

**switch** (currentMode) {

**case** SPARKLES\_MODE:

sparklesRoutine();

**break**;

**case** FILL\_MODE:

fillRoutine();

**break**;

**case** RNBOW\_HORION\_MODE:

rainbowHorizontal();

**break**;

**case** RNBOW\_VERTIC\_MODE:

rainbowVertical();

**break**;

**case** COLOR\_S\_MODE:

colorsRoutine();

**break**;

**case** COLOR\_MODE:

colorRoutine();

**break**;

**case** SNOWFALL\_MODE:

snowRoutine();

**break**;

**case** MATRIX\_MODE:

matrixRoutine();

**break**;

**case** WHITE\_LAMP\_MODE:

whiteLamp();

**break**;

**default**:

whiteLamp();

**break**;

}

FastLED.show();

}

}

ДЕКЛАРАЦИЯ ОБ ОТВЕТСТВЕННОСТИ

Нижеподписавшаяся, заявляю под личную ответственность, что материалы, представленные в магистерской диссертации, являются результатом личных научных исследований и разработок. Осознаю, что в противном случае, буду нести ответственность в соответствии с действующим законодательством.

Касимова Марина

25.05.2020