МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Институт машиностроения
(наименование института полностью)

Кафедра «Промышленная электроника»
(наименование)

13.03.02 Электроэнергетика и электротехника
(код и наименование направления подготовки / специальности)

Интеллектуальные энергетические системы
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Бактерицидный смарт-рециркулятор

Обучающийся С.О. Третьяков _____
(Инициалы Фамилия) (личная подпись)

Руководитель канд. техн. наук, доцент М.В. Позднов
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант канд. пед. наук, доцент С.А. Гудкова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2023

Аннотация

Название дипломной работы «Разработка устройства для обеззараживания воздуха». Данная работа состоит из пояснительной записки на 83 страницы, включая 39 рисунков, 4 таблиц, списка из 20 источников используемой литературы, в том числе 5 источников на иностранном языке и двух приложений, и чертежей на 7 листах формата А1.

Цель работы – разработка устройства для обеззараживания воздуха в помещениях с удобным управлением на расстоянии.

В данной дипломной работе проводится анализ существующих аналогов бактерицидных рециркуляторов, для того чтобы понять каких функций недостаёт большинству устройств. Далее была рассмотрена реализация блока системы управления. Были указаны все блоки, которые необходимы в данной схеме. Так же была описана схема блока питания устройства. Следующий шаг, это программирование блока системы управления и клиентской части, для этого нужно выбрать библиотеки и фреймворки, необходимые для разработки.

Для того чтобы выйти на массовое производство была рассмотрена сертификация устройства.

В итоге можно сделать следующий вывод, что после возникновения пандемии вырос спрос на бактерицидные рециркуляторы. Они применяются в школах, больницах и офисах. Именно для этого была проведена работа, чтобы предложить потенциальным покупателям наиболее дешевый вариант реализации устройства с наибольшим количеством функций и наиболее удобным для них.

Abstract

The title of the graduation work is « Development of an air disinfection device ». The senior paper consists of 83 pages, including 39 figures, 4 tables, a list of 20 sources of the literature used, including 5 sources in a foreign language and one appendix, and drawings on 7 sheets of A1 format.

The aim of the work is development of a device for disinfection of indoor air with convenient remote control.

In this graduation work the analysis of existing analogues of bactericidal recirculators is carried out in order to understand what functions most devices lack. The research methods were used – factor analysis, synthesis, statistical processing of results. Next, the implementation of the control system block was considered. All the blocks that are needed in this scheme were specified. The circuit of the power supply unit of the device was also described. The next step is programming the control system unit and the client part, for this you need to select the libraries and frameworks necessary for development.

In order to enter mass production, the certification of the device was considered.

Finally, the following conclusion can be drawn, that after the pandemic, the demand for bactericidal recirculators increased. They are used in schools, hospitals and offices. That's why work was carried out to offer potential buyers the cheapest version of the implementation of the device with the largest number of functions and the most convenient for them.

Содержание

Введение

Обстановка с вирусными заболеваниями, передающимися воздушно-капельным путем в условиях сезонных эпидемий ОРВИ, вынуждает искать методы и средства уменьшения вреда, наносимого человеку. Так, законодательно принят ряд документов, указывающих на необходимость использования мероприятий по дезинфекции воздуха (СанПиН 3.3686-21, СанПиН 2.1.3.2630-10, п.11.12).

Одним из средств борьбы с вирусом выступили ультрафиолетовые лампы. Ультрафиолетовое излучение убивает бактерии и вирусы. Они существуют достаточно давно, многие знают про процедуру кварцевания в больницах. Это когда комнату оставляют под излучением кварцевых ламп.

Для того чтобы как-то ограничить доступ человека к лампе и инкапсулировать ему определенный интерфейс взаимодействия существуют устройства под названием “обеззараживатели воздуха” или “рециркуляторы”.

Рециркуляторы довольно часто используется как способ обработки воздуха и воды от бактерий и прочих микроорганизмов. Поэтому в медицинских учреждениях часто используются бактерицидные рециркуляторы, которые очищают воздух от патогенов.

Этот метод очень популярен благодаря сочетанию простоты использования и эффективности обеззараживания воздуха. Кроме того, такие устройства не нуждаются в постоянном обслуживании, благодаря чему они могут работать на протяжении долгого времени. Многие рециркуляторы оснащены дополнительными функциями, благодаря которым пользоваться данным устройством становится гораздо проще. Например, функция подсчета срока службы лампы. С помощью этой функции, можно будет заранее узнать о том, что настало время заменить лампу на новую.

В отличие от традиционных кварцевых УФ-ламп такие устройства можно использовать в помещении с людьми из-за отсутствия эффекта озонообразования.

1 Постановка задачи

1.1 Поиск и анализ технических параметров аналогичных устройств

Мы проанализировали доступные в продаже обеззараживатели.

Анализ показал, что они в массе своей не обладают набором качеств, являющихся полезными и необходимыми: 1) дешевизной; 2) беспроводным управлением; 3) удобством управления; 4) расширенным функционалом.

Отметим ряд функций и особенностей, которые необходимы и полезны при работе рециркулятора.

1. при работе устройства необходимо следить, например, за ресурсом ламп. Ввиду того, что со временем мощность полезного УФ-излучения уменьшается, лампы должны быть заменены после периода гарантированной работы, устанавливаемого производителем в часах.

2. в помещениях с режимной работой (начало - конец рабочего дня) удобно использовать функцию, при которой будет осуществляться автоматическое включение и выключение устройства. Это исключит ситуации, при которых устройство не будет выполнять заданные функции по обеззараживанию.

3. предварительная обработка помещения требует функции таймера, когда устанавливается время непрерывной работы устройства с последующим выключением.

4. управление функциями рециркулятора можно осуществлять беспроводно. В противном случае, в конструкции необходимо предусматривать ряд кнопок и экран, что снижает технологичность изготовления устройства и влияет на его цену.

Поиск аналогичных рециркуляторов – одного класса и уровня производительности – показал наличие дешевого сегмента таких устройств

(3000-5000 руб.). Однако в них чаще всего присутствует только функция включения устройства и отсутствуют все вышеперечисленные опции.

Увеличение количества функций резко удорожает рециркулятор.

Для наглядности приведем по одному примеру рециркуляторов от производителей Armed, Мегидез и Поток:

Начнем с компании Поток.

Компания занимается производство медицинского оборудования более 13 лет, их продукция соответствует не только Российским нормам, но и европейского союза.

Рассмотрим их продукцию:

Рециркулятор бактерицидный Поток 1001(рисунок 1).

Рисунок 1 – Бактерицидный рециркулятор Поток 1001

Основываясь на поставленной задаче, исследуем технические параметры этого рециркулятора занесенные в таблице 1.

Таблица 1 – Технические характеристики рециркулятора Армед СН 211-115 М

Название технического параметра	Сведения о данном параметре
Наличие воздушный фильтр	Есть
Количество ламп	1 шт
Мощность лампы, Вт	15 В
Производительность м^3/ч	35м^3/ч
Наличие системы управления	Таймер (сенсорное управление)
Наличие счетчика ресурса ламп	Нет
Уровень звуковой мощности, дБ	Не более 35 дБ
Корпус	Металл
Стоимость, руб	8290 руб

Далее рассмотрим следующий вариант:

Armed – это компания, которая занимается производством медицинского оборудования с 1994 года.

Рассмотрим пример их рециркулятора. Его внешний вид изображен на рисунке 2.

Рисунок 2 – Бактерицидный рециркулятор Армед СН 211-115 М

Ряд технических параметров, данного устройства, отличается по сравнению с предыдущим рециркулятором. К примеру, у данной модели имеется больше ламп, и стоимость в несколько раз больше. Как и с прошлым устройством проанализируем его и запишем технические характеристики. В таблице 2 можно увидеть технические параметры устройства.

Таблица 2 – Технические характеристики рециркулятора Армед СН 211-115 М

Название технического параметра	Сведения о данном параметре
Наличие воздушный фильтр	Есть
Количество ламп	2 шт
Мощность лампы, Вт	15 Вт
Производительность $\frac{м^3}{ч}$	$50\frac{м^3}{ч}$
Наличие системы управления	Таймер (кнопки)
Наличие счетчика ресурса ламп	Нет
Уровень звуковой мощности, дБ	Не более 50 дБ
Корпус	Металл, пластик
Стоимость, руб	6990 руб

Сравнив два устройства одного производителя мы видим, что из-за отличий в параметрах их разница в цене колеблется в пределах 10000 рублей. Так же помимо увеличения стоимости устройства, увеличилась и производительность рециркулятора за счёт еще одной ультрафиолетовой лампы, плюс ко всему добавился подсчет ресурса, что тоже повлияло на увеличение цены.

Следующий на очереди еще один популярный производитель рециркуляторов МЕГИДЕЗ.

Его внешний вид изображен на рисунке 3.

Как и с ранее рассмотренными вариантами мы проанализируем модель от данного производителя.

Рисунок 3 – Бактерицидный рециркулятор Мегидез 3909

Проанализировав технические характеристики рециркулятора, можно сделать вывод, что данный вариант имеет преимущества над другими, а именно в высокой производительности ($60\frac{м^3}{ч}$) и низком уровне звуковой мощности (не более 40 дБ).

Подведем итоги анализа трех устройств.

Проведя анализ характеристик и функционала всех трех рециркуляторов, можно выделить характеристики, которые стоит проработать в нашем прототипе, тем самым сделать его более привлекательным для потребителей.

Выделим недостатки анализируемых рециркуляторов:

1. не все рециркуляторы имеют таймер работы, что затрудняет использование устройства;
2. в данных устройствах отсутствует какая-либо информация об отработке ресурса ламп;
3. наиболее важным недостатком является высокая стоимость устройств, которая варьируется от 6990 до 10000 рублей, в зависимости от встроенных характеристик.

Далее проанализируем разрабатываемый прототип, который будет иметь некоторые преимущества в сравнении с остальными бактерицидными рециркуляторами.

На данный момент можно выделить следующие преимущества прототипа:

- наличие мобильного приложения, с помощью которого можно дистанционно управлять бактерицидным рециркулятором;
- наличие управления по протоколу Bluetooth;
- низкая стоимость при сохранении всех качественных характеристик;
- подсчет ресурса ультрафиолетовых ламп;
- наличие таймера.

Для удобства все характеристики анализируемых рециркуляторов и нашего прототипа внесем в таблицу 3.

Таблица 3 – Сравнение технических характеристик рециркуляторов

Название	Производительность, $м^3/ч$	Радио управление	Подсчет ресурса ламп	Включение по расписанию	Включение по таймеру	Цена, руб.
Поток 1001	35	Wi-fi	Нет	Да	Нет	8290
Армед СН211-115	50	Нет	Да	Да	Да	6990
МЕГИДЕЗ 3909	60	Нет	Да	Нет	Нет	10000
Прототип	60	Bluetooth	Да	Да	Да	5 000

В таблице 3 можно наглядно увидеть преимущества нашего прототипа.

1.2 Обеззараживание воздуха ультрафиолетовым излучением

Один из основных способов обеззараживания воздуха от вредоносных микроорганизмов и вирусов, это обучение ультрафиолетовым излучением (УФ). Длина волны варьируется в диапазоне 100 – 400 нм, но наиболее эффективными свойствами обладает УФ излучение длиною в 205 – 315нм. Ниже на рисунке 4 можно увидеть процесс обеззараживания воздуха с помощью бактерицидного рециркулятора.

Проблема в том что человеку нельзя находиться в одной комнате с лампой, так как выделяющийся озон в это время, способен нанести вред для здоровья.

Поэтому был придуман другой вид ламп – безозоновые (без озона). Они позволяют человеку находиться рядом с ними сколько угодно.

На рисунке 4 можно увидеть графики относительной спектральной мощности от длины волны для кварцевых ламп.

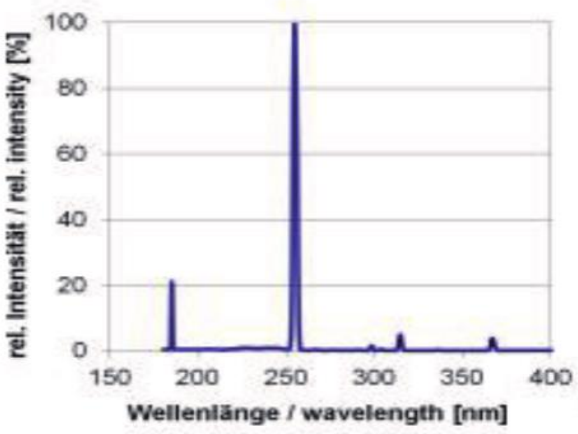

Рисунок 4 – График зависимости спектральной мощности от длины волны

На рисунке 5 можно увидеть график зависимости спектральной мощности от длины волны для безозоновых ламп.

Рисунок 5 – График зависимости спектральной мощности от длинны волны для безозоновых ламп

Таким образом мы убедились, что безозоновые лампы намного лучше кварцевых, а так же определились в выборе лампы для своего портотипа.

1.3 Принцип работы рециркулятора

Рециркулятор прокачивает вентилятором воздух внутрь корпуса, в котором установлены УФ-лампы и проводят его облучение. Таким образом в засасываемом прибором зараженном воздухе по мере его перемещения в корпусе падает концентрация различных вирусов, бактерий и спор грибов. На выходе устройства выдает обеззараженный воздух. На рисунке 6 изображена схема работы рециркулятора.

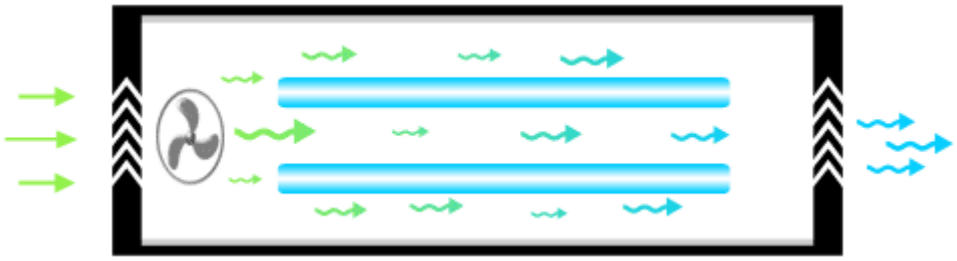

Рисунок 6 – Схема рециркулятора

Мы проанализировали предлагаемые на рынке прототипы устройства, выявили их основные параметры. Анализ показал, что нет устройств, которые обладают низкой ценой и управлением по протоколу Bluetooth, а так же имеющих дополнительные сервисные функции а именно подсчет ресурсов лампы, включение по таймерам

Так же было рассмотрено устройство и работа рециркулятора и технология применения безозоновых ламп, которые показывают, что можно обрабатывать воздух, при этом не производя ядовитого озона. То есть возможно использовать аналогичные устройства в присутствии людей.

2 Конструкторское проектирование устройства

2.1 Структура устройства

Для того чтобы разработать аппаратную часть устройства для начала составить функциональную схему. Она изображена на рисунке 7.

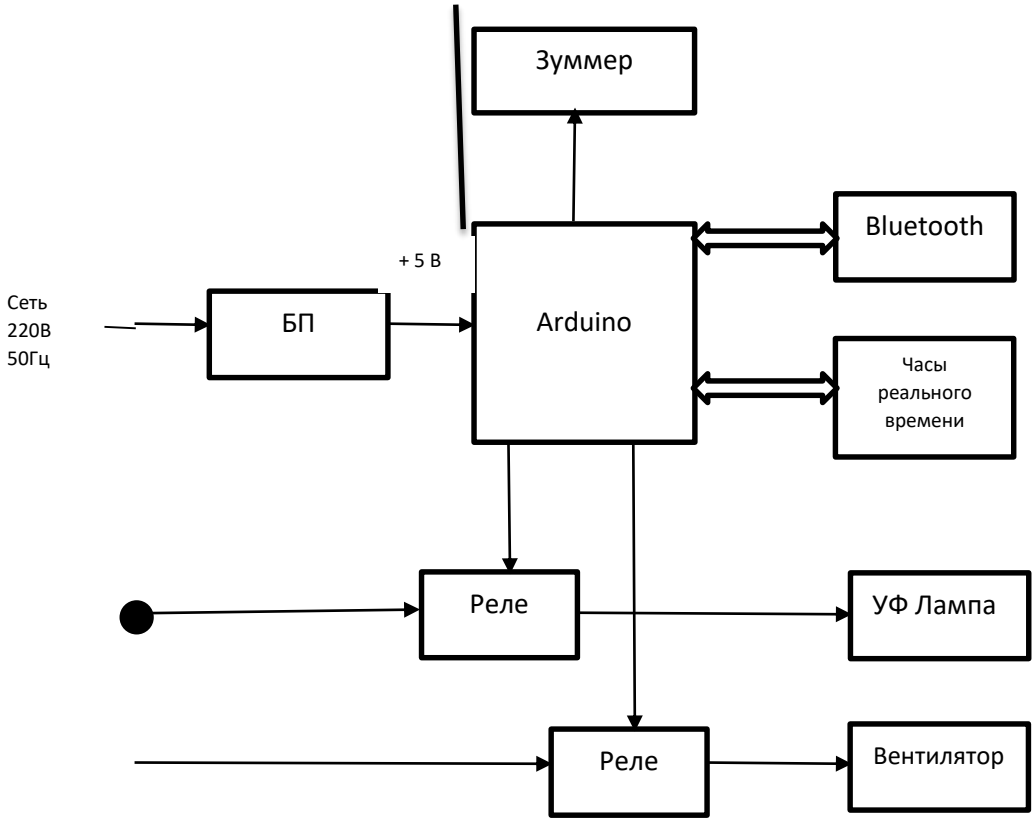

Рисунок 7 – Функциональная схема устройства

Функциональная схема состоит из блока питания который преобразует переменное напряжение 220 вольт и 50 герц в постоянное напряжение 5в необходимое для питания Arduino и остальных блоков схемы. Arduino – аппаратно-программная платформа для управления элементами рециркулятора. Она осуществляет работу по установленной программе, принимает данные с Bluetooth модуля а так же может отправлять их обратно для связи с Android приложением. Для системных предупреждений

предусмотрен зуммер – элемент который выдает звуковые сигналы постоянно частоты. Bluetooth модуль это модуль для передачи информации на Android приложение.Это позволяет управлять и принятия команд в обратную сторону и передачи его в модуль управления Arduino. Часы реального времени – это аппаратный блок необходимый для синхронизации процессов работы рециркулятора. Часы реального времени передают данные в ардуино и тогда она может синхронизировать часы по переданной информации из Android приложения. Коррекция времени необходима так как в телефоне может быть другое время и его всегда надо поддерживать в аппаратной части для исключения сбоев модуля управления. И ориентироваться нужно на время на стороне клиента, тк пользователь будет ориентироваться на него. Ардуино управляют двумя реле которые обеспечивают включение/выключение ультрафиолетовой лампы, а второе реле – ступенчатое изменение скорости вентилятора.

2.2 Разработка электронной части устройства

2.2.1 Полная схема устройства

Фаза и нейтраль приходят на разъем DB, с выхода RC цепочки снимается напряжение, далее это напряжение выпрямляется через диодный мост, далее все это ограничивается через стабилитрон до заданного значения,далее остаточные пульсации сглаживаются на конденсаторе с выпрямителя на 6,8 В и уже питает сам ардуино [9]. Далее идет зуммер SG1. С пинов D12,D10 он подключается для получения двухполярного напряжения Так же установлен модуль реального времени A3 и Bluetooth модуль A2,которые имеют стандартное подключение, по протоколу i2c, шина sda scl. Схему управления можно разделить на 2 части. В силовой части управление питанием осуществляется через тиристорные оптопары подключением фазы через тиристор. Первая оптопара DA1 управляет напряжением фазы через тиристор к лампе и к вентилятору. Паралельно

тиристору стоит цепочка C7 R15. Это демпфирующая цепочка [4]. Для включения вентилятора и регулирования скорости вентилятора есть вторая оптопара DA2. Она работает точно также как и первая оптопара, только здесь C4,R1 имеют другой номинал. C4 здесь существенно больше – 0,68 микрофарад это нужно для того чтобы питать вентилятор.

Когда тиристор выключен ток идет через вентилятор с ограничением через цепочку R1C3, то есть ток ограничен и вентилятор слабо крутится. Когда мы подаем импульс управления на светодиод от ардуино, тиристор включается и закорачивает R1C4 и на вентилятор подается напрямую напряжение 220в [12]. Соответственно обороты возрастают. Управление оптопарой DA1 так же происходит от ардуино. Для этого нужно подключать светодиод и у той и у этой оптопары через ограничительный резистор. Это R10 – 360 ом. Он ограничивает ток на уровне 10 – 15 милиампер. На рисунке 8 изображена принципиальная схема устройства.

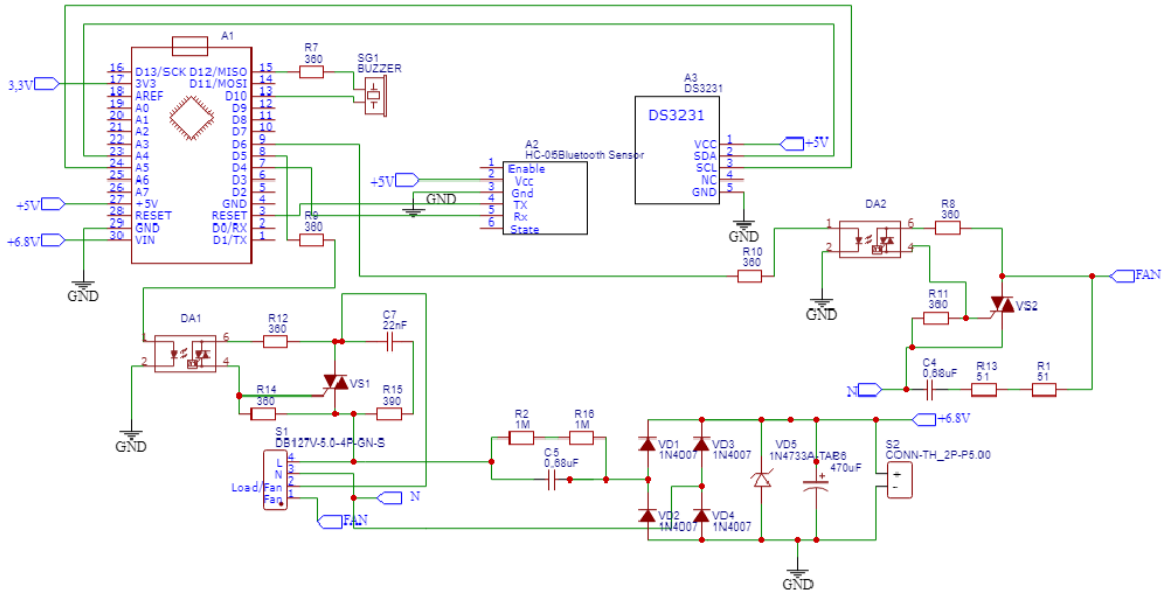

Рисунок 8 – Принципиальная схема устройства

Высокий уровень напряжения на выводе ардуино соответствующий 5В приводит к включению светодиода и соответственно оптопары.

2.2.2 Разработка схемы питания

Чтобы устройство функционировало его нужно питать электрическим током из сети, но оно часто не подходит для электронных устройств, тк выводит их из строя. Поэтому нами был разработан блок питания преобразующий ток из сети в подходящий для нормального функционирования устройства.

«Схема состоит из: VG1 – источник с параметрами 220 В, 50 Гц. Далее идёт балластный конденсатор C1=680 нФ и шунтирующий его резистор R1=1МОм. Для выпрямления переменного тока далее по схеме расположен диодный мост, который собран на диодах 1N4007, стабилизатор состоящий из стабилитрона 1N3305, параллельно к которому подключен сглаживающий конденсатор C3=470мкФ. Параллельно конденсатору C3 согласно заданию подключен эквивалент нагрузки - 30 резистор R2 =U / i = 6 В / 0,04 А =150 Ом. R3 и R4 токоизмерительные сопротивления. Балластный конденсатор C1 осуществляет основное ограничение входного тока для стабилизатора, собранного на стабилитроне Z1 со сглаживающим конденсатором C3. Ток нагрузки составляет 40 мА, следовательно ток короткого замыкания через конденсатор C1 должен быть больше для того, чтобы часть тока протекала через Z1 и осуществлялась стабилизация выходного напряжения. При C1=680нФ этот действующий ток составляет 47 мА. Следует отметить, что для получения диаграммы тока надо напряжение i_VD умножить на 1000. Стабилитрон выбран с напряжением стабилизации 6,2В, несколько выше заявленного, но вход питания из-за наличия в Arduino стабилизаторов напряжения 3,3 и 5 В позволяет подавать напряжения от 5 до 12 В [16].

Ток стабилизатора 31 изменяется волнообразно от 0,9 до 3 мА, при среднем токе примерно Iст=1,2 мА, что вызывает выделение средней мощности в стабилизаторе P = Uст*Iст = 0,007 Вт. При отключении дисплея и уменьшении потребления до нуля (режим холостого хода), балластный ток

через стабилитрон увеличится. Напряжение Uст = 6,2…6,3 В не изменится, а ток стабилитрона станет в пределах 11…65 мА, средний ток 41мА, что вызовет выделение мощности P = Uст*Iст = 6,25*0,041=0,26 Вт. Указанный диапазон мощностей соответствует допустимой мощности маломощных стабилитронов – обычно до 0,5Вт. Изменение напряжения на входе схемы также может влиять на параметры питания. Обычно напряжение в пределах нормы отклонения 10% может изменяться от 198…242 В. Моделирование при этом изменении показало, что выходное напряжение в режиме холостого хода - 3,5 В при 198 В и 6,3 В при 240 В. Как видно нижний порог напряжения выводит за пределы минимального допустимого напряжения в 5 В при питании Arduino. Следовательно, требуется увеличить балластный конденсатор до 1 мкФ. Это приводит к повышению нижнего уровня напряжения питания до 5,09 В при 198 В. При этом средний ток стабилитрона на 198В отсутствует, а на 240 В ток вырастает до 28 мА, а мощность стабилитрона становится P = Uст*Iст = 6,25*0,028=0,18 Вт. Таким образом выбор конденсатора в 1мкФ позволяет стабилизировать напряжение на выходе от 5,09…6,3 В при допустимом изменении сетевого напряжения. Напряжение на конденсаторе в силу балластного его характера почти полностью равно сетевому, выбирать в качестве конденсатора стоит конденсатор пленочного или бумажного типа на полное действующее сетевое напряжение 250В. Из особенностей схемы следует указать реактивный характер ее как нагрузки для сети. Интерес так же представляет пусковой режим такой схемы с точки зрения изменения входного тока при полностью разряженных конденсаторах (нулевых условиях). Так же нужно учитывать, что при включении фаза напряжения может меняться и необходимо определить режим, при котором пусковой ток станет максимальным. Этот режим наблюдается при фазе 90 град и 270 град и составляет 2,5А при входном напряжении 220 В. Следует учесть этот фактор при выборе конденсаторов C1 и C2.» [6]. На рисунке 9 изображена принципиальная схема блока питания.

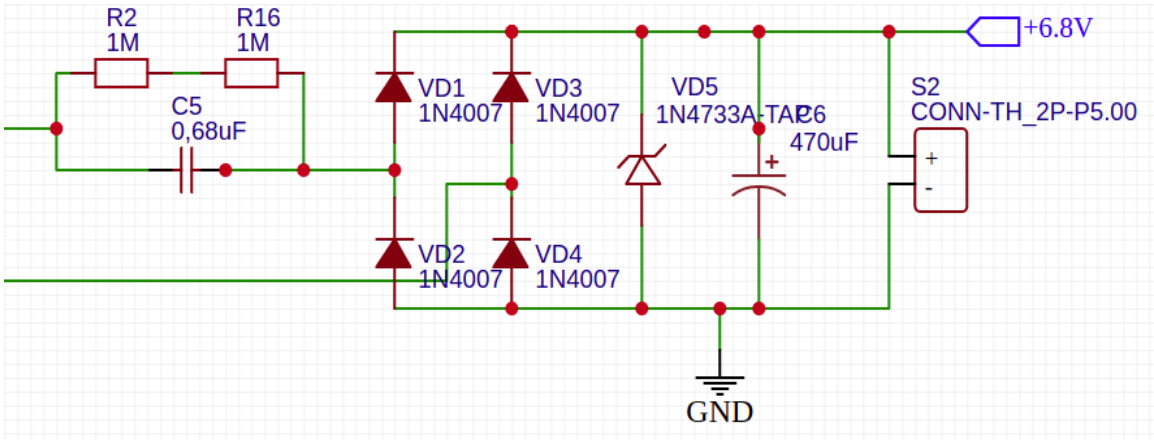

Рисунок 9 – Схема блока питания устройства

2.2.3 Разработка управляющей схемы устройства

Рассмотрим управляющие модули. В схеме управления мы используем модули с готовый схемотехникой, что значительно упростить выполнение задачи. Это Bluetooth модуль, модуль реального времени.

Элементы используемые в системе управления.

Данные функциональные возможности реализованы при помощи такого набора инструментов, которые обозначены в таблице 4.

Таблица 4 – Перечень используемого оборудования

№	Название	Количество
1	Микроконтроллер Arduino Nano	1
2	Модуль часов реального времени DS3231 (рис. 25)	1
3	Зуммер-динамик (рис. 27)	1

Все элементы, приведенные в таблице, подбирались так чтобы их габариты были как можно меньше. Один из таких элементов, модуль часов реального времени DS3231, мы взяли вариант исполнения с наименьшими габаритами, который имеет довольно малые размеры 13,6 x 13.4 x 10.5. На рисунке 10 изображено фото этого модуля.

Рисунок 10 – Модуль часов реального времени DS3231

Данный элемент необходим для отслеживания текущего времени. Также данный модуль необходим для работы рециркулятора точно по таймеру, отличительной особенностью является то, что данные часы реального времени способны работать при отсутствии питания контроллера.

Ещё один элемент, без которого не может работать модуль управления это Bluetooth модуль, который служит для связи устройства с телефоном клиента. Он изображен на рисунке 11.

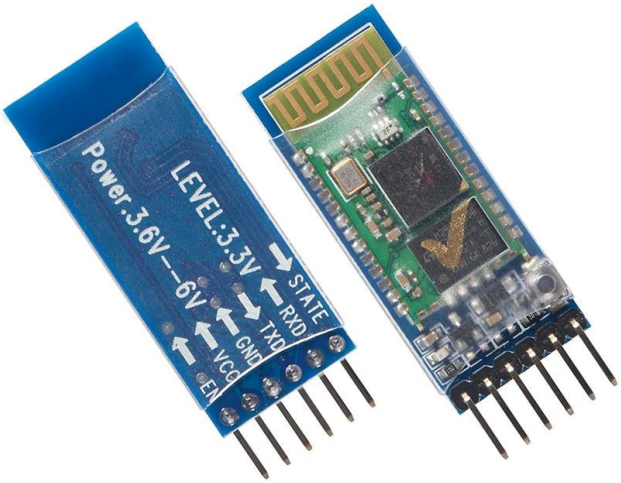

Рисунок 11 – Bluetooth модуль

Третьим, необходимым нам модулем, является активный зуммер, который входит в состав, проектируемой нами, системы управления. (рис. 27). «При подаче на динамик напряжение он начинает издавать звуковой сигнал.» [1]. Данный модуль идеально подходит для разрабатываемой системы управления, так как он имеет рабочее напряжение 5В. На рисунке 12 изображено фото этого модуля[13].

Рисунок 12 – Активный зуммер

Схема включения симистора.

В процессе отладки в предыдущей схемы в которой управление вентилятором и включением выключением лампы происходило через оптимисторную оптопару MOC3061 выяснилось, что при включении вентилятора и лампы проходит скачок тока в несколько ампер. Это ток приводит к выходу из строя оптимистора указанный от оптопары. Для устранения этого недостатка была найдена схема усилением тока с помощью силового симистора. На рисунке 13 изображена принципиальная схема управления с симистором.

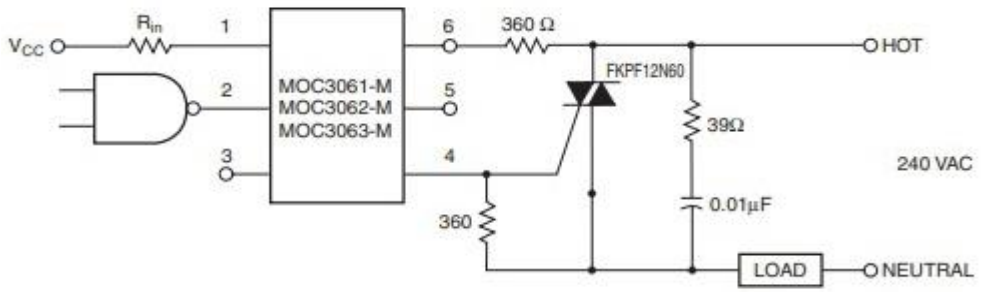

Рисунок 13 – Схема управления симистором

Из руководства использования оптопары была взята схема усиления сигнала для внешнего симистора которая приведена на рисунке 7. Эта схема содержит силовой симистор ток управления в котором формируется с помощью оптимистора ограничение тока производится резистором 360 ом, подключённого между анодом силового симистора и управляющим электродом этого симистра дополнительно для случайного включения увеличения силового симистора параллельно управляющему электроду и катоду симистра подключен резистор 360 ом. Для облегчения коммутационных процессов в симистре параллельно аноду и катоду симистра подключено демпфирующая RC цепь. Параметры этой цепи резистора 39 ом конденсатор 0,01 микрофарада.

Таким образом, была создана функциональная схема устройства, принципиальная схема его блок питания, подобраны основные модули необходимые для работы рециркулятора а так же проанализировали недостатки предыдущей версии схемы. Был проведен оценочный расчет, который удовлетворяет заданным параметрам питания периферических устройств рециркулятора, а также сформированы рекомендации по выбору параметров элементов. На этапе анализа пришли к выводу, что нужно изменить конфигурацию схемы заменив оптимисторные оптопары на силовые симисторы. Это необходимо для правильного функционирования устройства и избегания поломок [8].

3 Разработка алгоритма и управляющей программы

3.1 Модуль управления устройства

Разработка управления является одной из самых важнейших частей в данной работе. По сравнению с аналогичными устройствами это является большим преимуществом, так как подобная функция встречается в моделях от 15 000р.

В состав «модуля управления входит плата Arduino Nano (рисунок 14), которая работает на программируемом микроконтроллере ATMega 328, тактовой частотой 16 МГц [14].

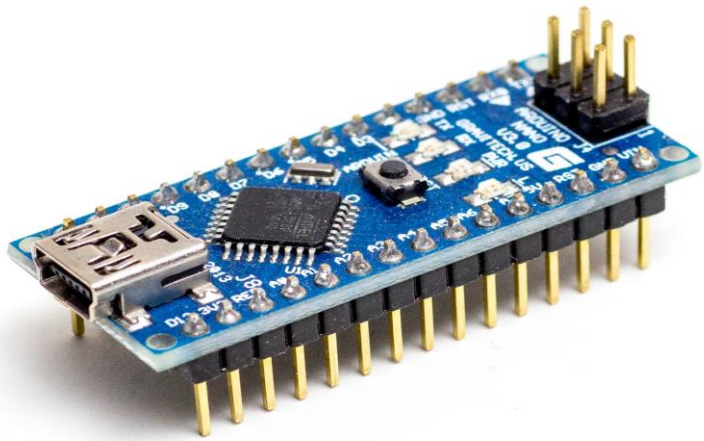

Рисунок 14 – Плата Ардуино Нано

Краткая история. «Arduino Первая плата Arduino была создана в 2005 году в итальянском Институте проектирования взаимодействий (Interaction Design Institute) в городе Иврea, близ Турина. Целью было создание недорогого и простого в использовании инструмента для обучения студен тов искусству проектирования интерактивных систем. Программное

обеспечение для Arduino, которое обеспечило этой плате значительную долю успеха, является доработкой открытого фреймворка с названием Wiring, созданного студентом этого же института. Доработанная версия для Arduino получилась очень близкой к оригиналу Wiring, а среда разработки Arduino IDE написана с использованием фреймворка Processing, старшего брата Wiring, способного работать на PC, Mac и других персональных компьютерах.» [2].

«Одним из преимуществ этой платы является ее небольшие габариты 42x19, при весе всего лишь в 19 грамм. В данной плате присутствуют как 27 цифровые, так и аналоговые порты. Из которых 8 аналоговых и 14 цифровых. На рисунке 15 изображена распиновка данной платы. Потребление тока в рабочем режиме – 24 мА, что является одним из важных критериев.» [3].

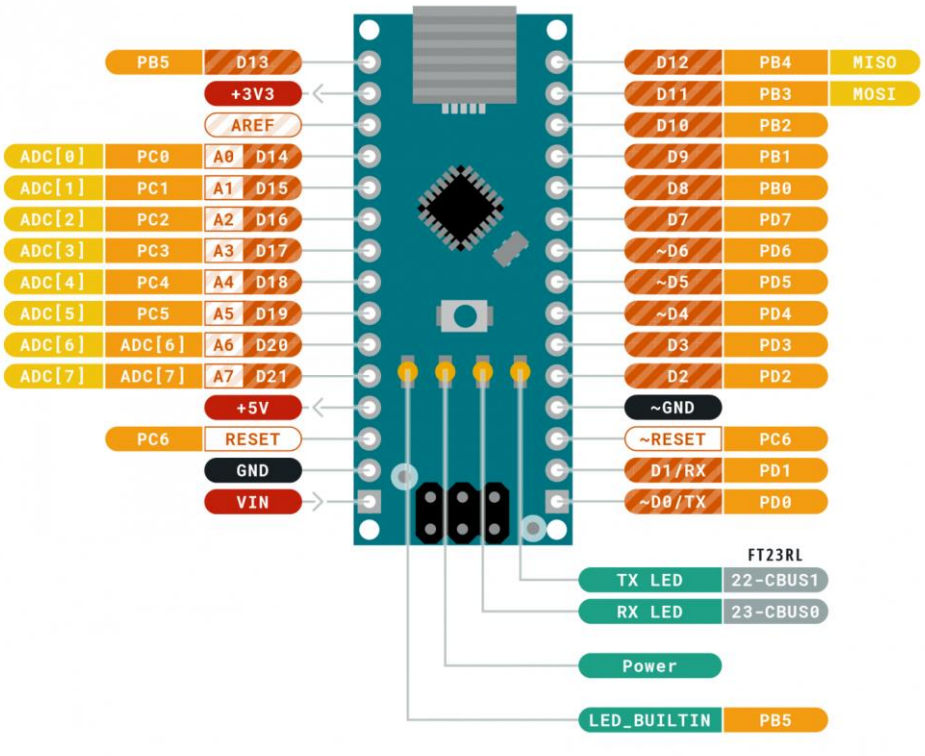

Рисунок 15 – Распиновка платы Arduino Nano

Программа для микроконтролера написанная на языке Arduino C и представляет собой язык C++ с фреймворком Wiring [5].

Фреймворк создает небольшой слой абстракции между языком c++ и разработчиком, позволяя не вдаваться в тонкости этого языка [18].

«Первое, что бросится в глаза - это совсем другая структура кода, вместо привычного в C++:» Разницу между встроенным фреймворком и без него представлен на листинге 1.

Листинг 1

```
#include <iostream>
using namespace std;

int main() {
return 0;
}
```

Нас встретят две функции:

void setup() { } - Функция, которая выполняется один раз, при включении микроконтроллера (скр.: мк). В ней производятся предварительные настройки или другим словом - **инициализация**

void loop() { } - В этой функции, точнее ее можно описать как бесконечный цикл (см. ниже), выполняется основной код вашей программы.

На самом деле функция main() никуда не исчезает. Фреймворк скрывает от нас лишнее и предлагает работать в более удобном варианте (по мнению разработчиков) [19].

На самом деле без фреймворка код программы будет выглядеть так:

```
#include "WProgram.h"// Определения всего функционала Arduino.

void setup();// Объявление функции setup и loop
void loop();
```

```
void setup() {
// Инициализация вашей программы
}

void loop() {
// Основной цикл вашей программы
}

int main(void) { // Основной цикл программы
init(); // Скорее всего инициализация фреймворка
setup(); // Инициализация вашей программы
for ( ; ; ) // Бесконечный цикл в котором "крутится" основной код вашей
программы

loop();
return 0; // Никогда не выполнится
}
```

3.2 Описание программы

3.2.1 Общая структура программы

Написанная программа может выполнять функции включения выключения устройства, включение выключение по таймеру, включение по таймеру, подсчет ресурсов лампы.

Сначала мы инициализируем и объявляем наши переменные нужные нам для работы программы, номера подключения пинов для наших модулей.

В функции setup мы так же инициализируем некоторые классы библиотек нужные для работы с модулями, устанавливаем соединение с компьютером для загрузки нашего кода и дополнительное соединение для приема передачи сообщений с Bluetooth модулем. Для этих целей

используются библиотеки Serial и SoftwareSerial. С помощью конструкции if проверяем работоспособность модуля реального времени.

В функции loop сначала проверяется не пришел ли какой-то сигнал на пины последовательного порта с помощью функции SoftwareSerial.available(). Этот метод получает информацию о количестве байтов (символов), доступных для считывания через программно-последовательный порт. Это данные, которые уже прибыли и хранятся в последовательном буфере для входящих данных. Далее мы считываем информацию по одному байту. Если информация представленная в строчном виде то это значит, что мы считываем по одному символу.

Этот символ присваивается переменной res_str которая хранит данные на одну итерацию цикла.

После считывания мы прибавляем к переменной res_sum переменную res_str и после каждой итерации в res_sum накапливается строка которая формирует сообщение.

Чтобы понять, что сообщение закончилось мы в каждой итерации с помощью управляющей конструкции if проверяем является ли содержимое res_str специальным символом из таблицы unicode. Преобразуем символ в числовой тип и ожидаем таким образом проверки, сравнивая его с числами 10 или 13, которые являются числовым представлением служебного символа.

Далее с помощью управляющих конструкций if повторяющихся каждая за собой по очереди, мы определяем какое сообщение пришло с последовательного порта. Если в первом if условие не выполнилось и команда оказалась не подходящей, значит мы спускаемся в следующий if, где мы сравниваем res_sum со следующей строкой.

В самом конце функции loop программа работает с энергонезависимой памятью EEPROM. Она проверяет переменную lampWorkFlag типа boolean и хранит в себе два значение true или false. Когда она равна true то запускается алгоритм добавляющий число 1 в ячейку памяти EEPROM каждые 2 минуты. При переполнении ячейки, а именно про достижении числа

255(максимальное число для хранения в одной ячейке, так как размер ее размер 1 байт) алгоритм переходит на следующую ячейку. Для учета всех занятых ячеек под эту операцию, мы зарезервируем первые 7 ячеек и будем действовать по аналогичному алгоритму.

Всего у программы имеется 6 состояний:

– включение;
– выключение;
– таймер включения;
– таймер включения/выключения;
– подсчет ресурса лампы;
– режим высоких оборотов вентилятора.

Нам будут приходить сообщения типа String из последовательного порта которые обозначает каждое из этих 6 состояний.

Имеется 6 видов сообщений:

– “вкл” Команда включения лампы с вентилятором;
– “выкл” Команда выключения лампы с вентилятором;
– “высскор” Команда переключения вентилятора на высокие обороты;
– “вклтайм” Команда включения таймера включения;
– “вклтайм2” команда включения таймера включения/выключения;
– “ресурсы” команда вызова подсчитанных ресурсов лампы.

В зависимости от того, какое сообщение будет получено от клиента такая функция и будет вызвана. Это реализовано с помощью управляющей конструкции if [21]. Если в каком-то из условий при проверке полученной сообщение совпадет с тем которое проверяется, то состояние поменяется на то которое подразумевает это сообщение и будет вызвана соответствующая подпрограмма [11]. Фрагмент кода можно увидеть в листинге 2.

Листинг 2

```
void setup() {
```

```
  Serial.begin(9600);
  mySerial.begin(9600);
  pinMode(buz1, OUTPUT);pinMode(buz2, OUTPUT);
  pinMode(power, OUTPUT);pinMode(fan, OUTPUT);
  // проверка наличия модуля на линии i2c
  if (!rtc.begin()) {
    Serial.println("DS3231 not found");
    for(;;);
  }else {
    Serial.println("DS3231 found");
  }
  time_timer=millis();
  SW=0;
}
void loop() {

  while (mySerial.available()) // цикл сбора информации с порта блютус в string переменную res_sum оканчивается знаком перевода строки с кодом 10
  { res_str=mySerial.read();
  if (res_str==13 or res_str==10) break;
   res_sum = res_sum + res_str;
  }
    // определение запуска функций на принятую комманду по блютусу
   if  (res_sum=="вкл") {digitalWrite(power,1);ledSignalOn();Serial.println("Signal_On");lampWorkFlag = true;} // комманда включения лампы с вентилятором
   if  (res_sum=="выкл") {digitalWrite(power,0);ledSignalOff();Serial.println("Signal_Off"); lampWorkFlag = false;}// комманда выключения лампы с вентилятором
```

3.2.2 Описание внутренних функций

Опишем каждую подпрограмму:

Структурировать описание подпрограмм

ledSignalOn отправляет указания микроконтролеру подать цифровой сигнал на указанный пин (на пин D8), далее с помощью резисторного ограничителя на светодиод оптопары e2 подается ток в 10 мА, после чего силовым семестром на лампу коммутируется напряжение из сети. Функция реализована через функцию digitalWrite с параметрами содержащими константу HIGH.

ledSignalOff отправляет указания микроконтролеру перестать подавать ток на оптопарую.Он так же пользуется тем же слоем абстракции, конкретно методом digitalWrite с константой LOW которая кладется к методу в параметры.

highSpeed функция командующая ардуино подать цифровой сигнал на пин D6. С помощью этого включается оптосимистор шунтирующий токоорганичительную RC цепь и на на вентилятор подается большее напряжение и он увеличивает свои обороты.

onTimer выполняет написанный алгоритм таймера который будет пользоваться. Этот метод отслеживает каждую секунду время переданное в формате HH:MM:SS с модуля DS3231 и сравнивает его с переданной строкой со временем с телефона по протоколу Bluetooth. Так же там постоянно отслеживаются и другие команды с телефона, чтобы устройство при заведения таймера не прекращало выполнять и другие более примитивные функции если потребуется, такие как включение выключение и повышение оборотов . То есть реализована псевдомногопоточность.

checkCounter работает с EEPROM и получает из нее данные о накопленном времени работы лампы. В локальную переменную sum мы добавляем содержимое ячеек в которых хранятся данные об времени работы лампы. Это реализуется с помощью цикла for размером от 7 до индекса последней использованной ячейки. Код функции показан в листинге 3.

Листинг 3

```
int sum = 0;
    for(int i = 7; i <= EEPROM[0]; i++) {
      sum = sum + EEPROM[i];
    }
   mySerial.println(String(sum*2)+"e");
```

3.2.3 Описание алгоритма таймера

Опишем алгоритм таймера.

В аргументы метода передается переменная типа String под названием timer_on_value которая хранит в себе данные времени когда таймер должен включиться и локальное время на клиенте. Это нужно для того, чтобы микроконтролер сработал в правильное для пользователя время, такое по которому он ориентируется со своего телефона. Эти данные передаются в оперативную память микроконтроллера с приложения клиента. Сначала происходит сравнение локального времени на устройстве, которое мы получаем с модуля реального времени(DS3231) и локального времени отправленного с клиента. Для этого используется управляющая конструкция if. Ниже приведен пример этого сравнения.

```
if(rtc.getTimeString()!=timer_on_value.substring(timer_on_value.lastIndexOf("#")
+1)) {
int hours = timer_on_value.substring(timer_on_value.lastIndexOf("#")+1,
timer_on_value.lastIndexOf("#")+3).toInt();
int minutes = timer_on_value.substring(timer_on_value.lastIndexOf("#")+4,
timer_on_value.lastIndexOf("#")+7).toInt();
int seconds = timer_on_value.substring(timer_on_value.lastIndexOf("#")+7,
timer_on_value.lastIndexOf("#")+9).toInt();
Serial.println(timer_on_value.substring(timer_on_value.lastIndexOf("#")+1,
timer_on_value.lastIndexOf("#")+3));
```

```
rtc.setTime( seconds, minutes, hours, 21, 8, 2022);
}
```

Если они не совпадают то время на модуле переписывается под время с клиента.

Далее запускается управляющая конструкция в виде цикла while(true), который является бесконечным циклом. Из него можно выйти только после метода, который будет вызван, когда таймер придет к заданному времени срабатывания или при вызове нового таймера. Выход из цикла осуществляется оператором break. В нём сравнивается полученное время срабатывания таймера из переменной timer_on_value с обновляющимся в каждой итерации локальном временем. Если эти данные будут равны то сработает функция ledSignalOn.

```
if(rtc.getTimeString().substring(0,2).toInt() ==
timer_on_value.substring(timer_on_value.indexOf("#")+1,timer_on_value.indexO
f("#")+3).toInt() && rtc.getTimeString().substring(3,6).toInt() ==
timer_on_value.substring(timer_on_value.indexOf("#")+4,timer_on_value.indexO
f("#")+7).toInt()) {
    Serial.println("таймер сработал");
    ledSignalOn();
    break;
  }
```

Так же в этом цикле алгоритм работает аналогично циклу loop и принимает сообщения от клиента, и точно так же обрабатывает их и вызывает методы ledSignalOn, ledSignalOff, highSpeed, onTimer и onOffTimer о которой речь пойдет ниже. Это и есть объяснение термину “псевдомногопточность” в контексте этой программы.

onOffTimer функция аналогичной onTimer, но она не выходит из цикла при срабатывании таймера и вызова метода ledSignalOn, она продолжает ожидать время выключения устройства, тк в аргументы к этой функции

передается еще одна дополнительная строка с временем отключения устройства.

Пример кода отключения устройства.

```
if(rtc.getTimeString().substring(0,2).toInt() ==
value.substring(value.indexOf("/")+1,value.indexOf("/")+3).toInt() &&
rtc.getTimeString().substring(3,6).toInt() ==
value.substring(value.indexOf("/")+4,value.indexOf("/")+7).toInt()) {
    Serial.println("таймер сработал");
    ledSignalOff();
    break;
  }
```

Таким образом мы рассмотрели модуль управления устройства и разобрали программное обеспечение для этого модуля. Полный код программы представлен в Приложении А.

3.3 Клиентская часть

3.3.1 Описание функций работы клиентской части

Клиентская часть разрабатываемого комплекса отвечает следующим требованиям функционала:

1. Установление соединения с аппаратной частью, реализуемое посредством протокола Bluetooth. Устройством – обнаружителем является мобильное устройство с операционной системой Android, отвечающее всем правилам современных Bluetooth модулей. Аппаратная часть принимает запрос на соединение от устройства – обнаружителя, чем является клиентская часть.

2. Отправка данных аппаратной части, реализуемое с помощью протокола Bluetooth в режиме передачи данных. С помощью встроенного в устройство – передатчик Bluetooth модуль осуществляется передача данных.

3. Формирование команд для отправки в аппаратную часть, где данные обрабатываются и вызывается определенная функция устройства в зависимости от команды.

4. Разрыв соединения с устройством, реализуемое на стороне клиентского приложения.

Описание программы и алгоритма клиентской части.

Программа для клиентской части, а именно устройства с операционной системой Android разрабатывается в IDE Android Studio. Android Studio - это интегрированная среда разработки (IDE), предназначенная для разработки мобильных приложений на операционной системе Android. Android Studio разработана компанией Google на основе IntelliJ IDEA. Она предоставляет разработчикам широкий набор инструментов для создания, отладки и тестирования приложений для Android, включая редактор кода, визуальный макет-редактор, инструменты для управления проектом, средства отладки, эмуляторы устройств и многое другое.

Android Studio имеет обширную документацию и сообщество, которое поддерживает разработчиков, помогая им решать проблемы и получать новые знания и навыки. Также Android Studio интегрирована с сервисами Google, такими как Google Play Store, Google Analytics, Firebase и другими, что облегчает разработку, тестирование и публикацию приложений в магазине приложений Google Play.

Android Studio также позволяет разрабатывать приложения на других языках программирования, таких как Kotlin и Java, а также поддерживает разработку приложений для других платформ, таких как Android Wear, Android TV и Android Auto. Для разработки проекта выбран язык Kotlin - «надстройка» над java которая универсальным языком для разработки программного обеспечения для множества платформ.

Проект Android Studio состоит из нескольких компонентов:

Файлы исходного кода - это файлы, содержащие исходный код на языке Java. Они разбиты на пакеты и классы, каждый из которых выполняет определенную функцию в приложении.

«Файл манифеста – это файл, содержащий метаданные для группы сопутствующих файлов, которые являются частью набора или согласованного блока. Например, файлы компьютерной программы могут содержать манифест с описанием имени, номера версии 4.2, лицензии и составляющих файлов программы.» [15].

XML-файлы макетов - это файлы, в которых описан пользовательский интерфейс (UI) приложения. Они содержат описание элементов интерфейса, таких как кнопки, текстовые поля и изображения, а также информацию о том, как они должны быть расположены и выглядеть на экране.

Ресурсы – это папка с файлами, содержащие данные, необходимые для приложения, такие как изображения, звуки, строки, цвета и так же сама верстка экранов приложения. Они хранятся в различных папках внутри проекта.

Фреймворки - это программные продукты, которые упрощают создание и поддержку технически сложных либо нагруженных проектов. Он содержит только базовые программные модули, а все специфичные компоненты реализуются программистом на их основе, таких как работа с сетью или базами данных или сама разработка андроид приложения (Android framework).

Файлы конфигурации - это файлы, которые содержат информацию о настройках приложения, таких как версия SDK, API-ключи и настройки сборки. Они используются для настройки проекта и сборки приложения.

Файлы сборки – это файлы которые использует система сборки для управления зависимостями и сборки приложения. Под зависимостями имеется ввиду информация о библиотеках, используемых в проекте, которая нужна системе сборки для того чтобы скачать библиотеку из официального репозитория библиотек для Java Development Kid. Также это является

инструкцией для сборки приложения в определенный формат, в нашем случае это APK-файл.

Все эти папки и файлы совместно образуют проект Android Studio для Java Development Kid и позволяют разработчикам создавать приложения для платформы Android.

В Android Framework существуют так называемые компоненты приложения, с помощью которых мы можем разделить функциональность приложения на отдельные куски(классы) и установить с ними связь. Для наших потребностей нам пригодится компонент Activity. Activity представляет собой класс, в котором описана логика создания экрана приложения и взаимодействия его с пользователем. Каждая Activity должна быть прописана в файле манифеста приложения, так как он докладывает операционной системе о наличии этого компонента и тогда она сможет запустить его.

Она может вызывать другие активности и фрагменты в зависимости от потребностей приложения. Если приложение содержит несколько экранов, то MainActivity может вызывать другие активности или фрагменты для отображения этих экранов. MainActivity имеет свой жизненный цикл, который состоит из ряда состояний, таких как создание, запуск, приостановка, возобновление и уничтожение. Каждое состояние может быть использовано для управления процессом создания и управления пользовательским интерфейсом и создания бизнес логики приложения [20].

Современное Android приложение строиться по архитектуре SingleActivity, поэтому мы будем использовать в проекте для всех экранов класс Fragment, а Activity будет использоваться как хранилище контекста и бизнес логики приложения. Наши фрагменты будут обращаться к ней за этим.

Fragment — “модульная часть activity, у которой свой жизненный цикл и свои обработчики различных событий” [10].

Context – “это объект, который предоставляет доступ к базовым функциям приложения: доступ к ресурсам, к файловой системе, вызов активности и т. д”.

Adapter — объект указывает системе как нужно отобразить на экране список

Broadcast Receiver — «объект, являющийся компонентом приложения, который прослушивает события системы и позволяет приложению реагировать на них.» [17].

SharedPreferences – «постоянное хранилище на платформе Android, используемое приложениями для хранения своих настроек, например. Это хранилище является относительно постоянным, пользователь может зайти в настройки приложения и очистить данные приложения, тем самым очистив все данные в хранилище.» [7].

BluetoothAdapter позволяет выполнять основные задачи Bluetooth, такие как инициация обнаружения устройства, запрос списка связанных (сопряженных) устройств, создание экземпляра с использованием BluetoothDevice известного MAC-адреса, создание экземпляра BluetoothServerSocket для прослушивания запросов на подключение от других устройств и запуск сканирования Bluetooth.

RecyclerView – это компонент пользовательского интерфейса, с помощью которого можно создать список [20].

В Android Studio можно создавать UI с помощью визуального редактора интерфейса или путем написания кода на языке программирования. Каждая Activity и Fragment имеет свой жизненный цикл, который состоит из ряда состояний, таких как создание, запуск, приостановка, возобновление и уничтожение. Эти состояния могут быть использованы для управления процессом создания и управления пользовательским интерфейсом.

3.3.2 Описание программы

Приложение имеет одно Activity и одинадцать Fragment, один Broadcast Receiver, который следит за установкой соединения по Bluetooth и выдает соответствующее сообщение на экран используется объект Toast. Так же имеется класс Adapter.

1. MainActivity — это основная Activity в Android. Этот класс наследуется от AppCompatActivity класса и хранит в себе весь основной функционал, которым пользуются остальные компоненты приложения.

Переменная preferences типа SharedPreferences. Модификатор lateinit говорит о том, что данная переменная будет инициализирована позже. Мы используем ее для хранения данных об устройствах к которым ранее уже подключалось приложение. Это дает возможность быстрого подключения к устройству во второй раз вместо процесса выбора его в списке всех возможных подключений. ConnectionsFragment игнорируется и сразу откроется ControlsFragment. Речь об этих экранах последует ниже.

Переменная bluetoothAdapter хранит в себе объект типа BluetoothAdapter. Он пригодится нам при подключении, отключении и поиску устройств находящихся рядом.

Далее идут переменные deviceSet, connectThread и connectedThread. Первая хранит список устройств доступных к подключению по Bluetooth, а connectedThread и connectedThread являются написанными нами классами для установки соединения и приема сообщений по этому соединению.

В методе onCreate мы проверим есть ли в памяти приложения какие-либо устройства и если есть попытаемся подключиться к ним, а если нет то выведем экран выбора подключений. Им является класс BluetoothDevicesListFragment.

Далее мы проверяем наличие Bluetooth на самом устройстве и работает ли он. Если работает то вызывается BluetoothDevicesListFragment а если нет то BluetoothOffFragment. Так же мы проверяем наличие подключения к устройству при помощи переменных connectThread и connectedThread. Если они равны null то подключение отсутствует.

Метод onDisconected отрабатывает при разрыве соединения с устройством.

В нем вызывается метод launchFragment, который занимается созданием фрагментов [18].

launchFragment является написанным нами методом, который упрощает работу с фрагментами. В нем вызывается класс supportFragmentManager и выполняются его методы, необходимые для создания фрагмента и складывания его в стек фрагментов для правильной навигации по приложению.

Класс ConnectThread наследуется от класса Thread и устанавливает соединение в другом потоке инициализируя класс ConnectedThread и присваивает переменной isConnected значение true.

Класс ConnectedThread создает у себя цикл while, который будет работать пока переменная isConnected будет равна true. В цикле происходит прием и обработка сообщений.

В методе onDestroy мы обрываем все соединения с устройством путем присвоения переменным connectThread и connectedThread значения null

У нас есть перегруженный метод enableButton который формирует сообщение для отправки на подключенное устройство. Перегружен он потому что для разных таймеров нужно разное кол во аргументов.

MainActivity реализуют интерфейсы для остальных экранов. Обычная практика реализовывать методы для всех экранов в MainActivity. Это single activity подход к архитектуре приложения который является самым современным на сегодняшний день. Все фрагменты используют методы из Activity для своих нужд. Все эти методы представляют собой вызовы launchFragment или enableButton с разными аргументами [19].

2. Класс ControlsFragment - это экран меню, где находится навигация по приложению, а именно кнопки, при нажатии на которые откроется экран с выбранным режимом для рециркулятора.

В функции setUpView реализованы слушатели, реагирующие на нажатие кнопок и вызывающие методы из Activity и Android Framework. Activity.onBackPressedDispatcher.addCallBack(){} работает как обработчик события нажатия системной кнопки назад в Android.

Переменная binding типа FrgamnetControlsBinding хранит в себе все объекты из xml файлов верстки. На все эти объекты объявляются слушатели событий нажатия, потому что все объекты являются кнопками. В зависимости от того на какую кнопку нажал пользователь будет вызван метод из MainActivity, который либо создаст новый фрагмент либо сразу отправит команду на устройство (это относится к кнопкам включить, выключить).

Всего на экране расположено 6 кнопок

- включить устройство;
- выключить устройство;
- завести таймер на включение;
- завести таймер на включение и выключение;
- усилить обороты вентилятора;
- узнать ресурс лампы.

Первые две при нажатии вызывают метод enableButton, который формирует и отправляет команду на включение или выключение лампы и вентилятора.

Третья кнопка при нажатии на которую вызовется метод launchFragment, который создаст и откроет фрагмент меню таймера, в котором можно выбрать часы и минуты когда включится лампа и вентилятор.

Четвертая кнопка открывает фрагмент меню таймера, в котором нужно ввести время включения и время выключения лампы с вентилятором.

Пятая кнопка при нажатии на которую будет вызван метод enableButton, который сформирует и отправит команду на усиление оборотов вентилятора.

При нажатии на шестую кнопку откроется фрагмент показывающий на сколько процентов износилась лампа. На рисунке 16 показан графический интерфейс класса.

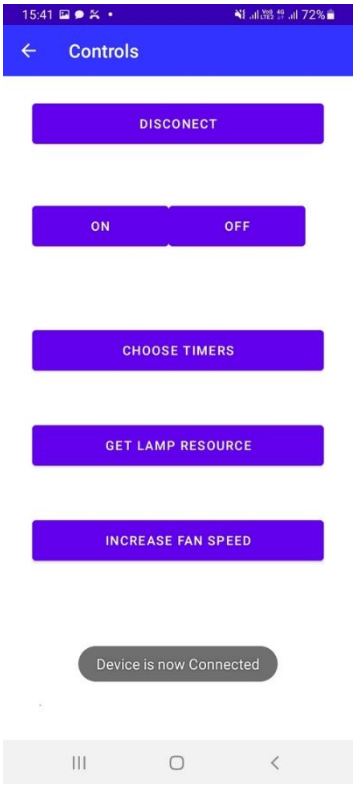

Рисунок 16 – Графический интерфейс ControlsFragment

3. Класс BluetoothDevicesListFragment - это экран со списком устройств доступных к подключению по Bluetooth. Он создается если соединение по Bluetooth отсутствует. В функции setUp инициализирован RecyclerView. Он берет информацию всех доступных устройств для подключения и создает список. Пример кода показан в листинге 4.

Листинг 4

```
recyclerView.apply {
        layoutManager = LinearLayoutManager(context)
        adapter = BtlistAdapter (
           {delegate.getBoundedBtDevices()},
```

```
            requireContext()
            )
        }
```

В функции так же инициализирована переменная switchEnableBt, которая является переключателем состояния Bluetooth между включением и выключением. Пример кода показан в листинге 5.

Листинг 5

```
switchEnableBt.setOnCheckedChangeListener { buttonView, isChecked ->
            if (buttonView == switchEnableBt) {
                delegate.enableBt(isChecked)

                if (!isChecked) {
                    delegate.moveToBtNoFragment()
                }
            }
        }
```

На рисунке 17 показан графический интерфейс класса.

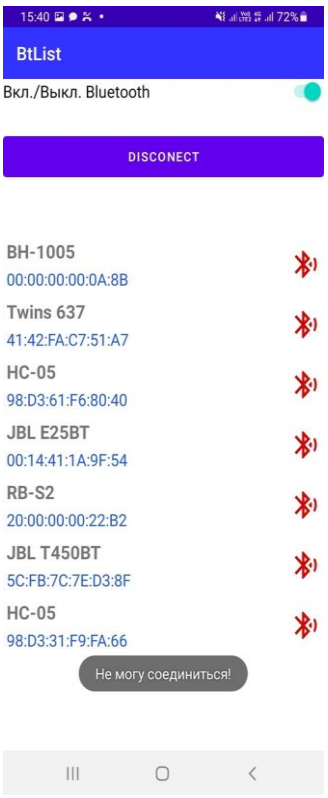

Рисунок 17 - Графический интерфейс BluetoothDevicesListFragment

4. Класс BluetoothOffFragment — это экран, который отображается если Bluetooth отключен. В его функции setUp инициализирована переменная switchEnableBt, которая является переключателем состояния Bluetooth.

Графический интерфейс изображен на рисунке 18.

Рисунок 18 - Графический интерфейс BluetoothOffFragment

5. Класс BtListAdapter реализует методы для отображения списка всех возможных устройств, к которым можно подсоедениться, находящихся рядом. Этот класс использует BtViewHolder для своей работы.

6. TimePickerFragment наследуется от класса DialogFragment и реализует его метод onCreateDialog, в котором реализуется диалоговое окно с выбором времени для таймера. Метод показан в листинге 6.

Листинг 6

```
@NonNull
   @Override
   public Dialog onCreateDialog(Bundle savedInstanceState) {
      Calendar c = Calendar.getInstance();
      int hour = c.get(Calendar.HOUR_OF_DAY);
      int minute = c.get(Calendar.MINUTE);
```

```
        return new TimePickerDialog(getContext(),
(TimePickerDialog.OnTimeSetListener) getContext(), hour, minute,
DateFormat.is24HourFormat(getContext()));
    }
```

Графический интерфейс изображен на рисунке 19.

Рисунок 19 – Графический интерфейс TimePickerFragment

6. Класс TimersFragment является экраном, в котором отображены все таймеры, а именно кнопки при нажатии на которые происходит переход в фрагмент с отдельным таймером.

Фрагмент кода из этого класса показан в листинге 7.

Листинг 7

```
 timerOnOffButton.setOnClickListener {
            delegate.moveToTimerOnOffFragment()
        }
```

```
        timer.setOnClickListener {
            delegate.moveToTimerFragment()
        }
        timerOn.setOnClickListener {
            val timePicker: androidx.fragment.app.DialogFragment =
TimePickerFragment()
            timePicker.show(parentFragmentManager, "g")
        }
```

Графический интерфейс изображен на рисунке 20.

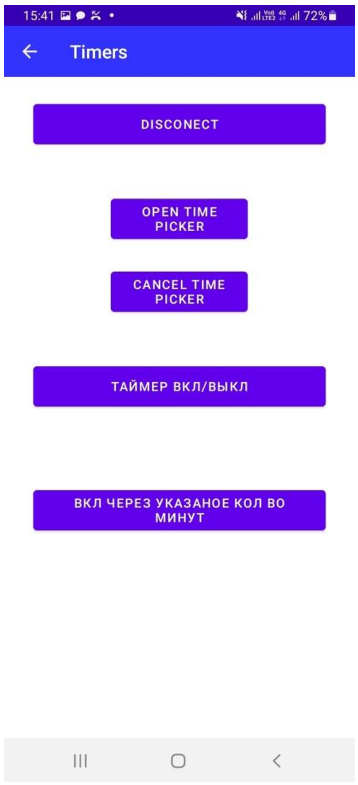

Рисунок 20 – Графический интерфейс TimersFragment

7. Класс TimerFragment — это экран, который предоставляет таймер на включение устройства через указанное количество часов и минут. Переменная timerHoursEditText и timerMinutesEditText хранят в себе введенное пользователем значение. Если пользователь хочет установить

таймер меньше чем на час, то программа позволит это сделать, положив в аргументы метода sendTimerCommand только значение timerMinutesEditText.

Фрагмент кода из этого класса показан в листинге 8.

Листинг 8

```
setTimerButton.setOnClickListener {
            if (timerHoursEditText.text.toString() == "") {
                delegate.sendTimerCommand(0,
timerMinutesEditText.text.toString().toInt())
            } else {

delegate.sendTimerCommand(timerHoursEditText.text.toString().toInt(),
timerMinutesEditText.text.toString().toInt())
            }
         }
```

Графический интерфейс изображен на рисунке 21.

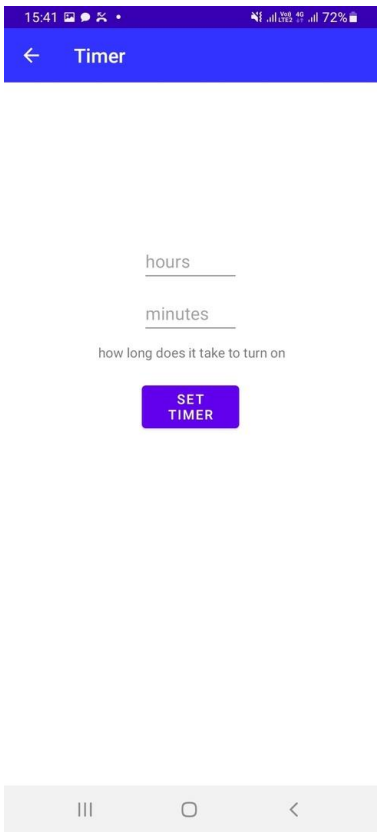

Рисунок 21 – Графический интерфейс TimerFragment

8. Класс TimerOnOffFragment — это экран это экран, который предоставляет таймер на включение и выключение устройства через указанное количество часов и минут. Переменные editTextHourOn, editTextMinutesOn, editTextHourOff и editTextMinutesOff хранят в себе данные о времени включения и выключения устройства. При нажатии на кнопку эти данные кладутся в аргументы методу sendTimerOnOffCommand.

Фрагмент кода изображен в листинге 9.

Листинг 9

```
setTimerButton.setOnClickListener {

delegate.sendTimerOnOffCommand(editTextHourOn.text.toString().toInt(),
editTextMinuteOn.text.toString().toInt(),
        editTextHourOff.text.toString().toInt(),
editTextMinuteOff.text.toString().toInt())
```

}

Графический интерфейс изображен на рисунке 22.

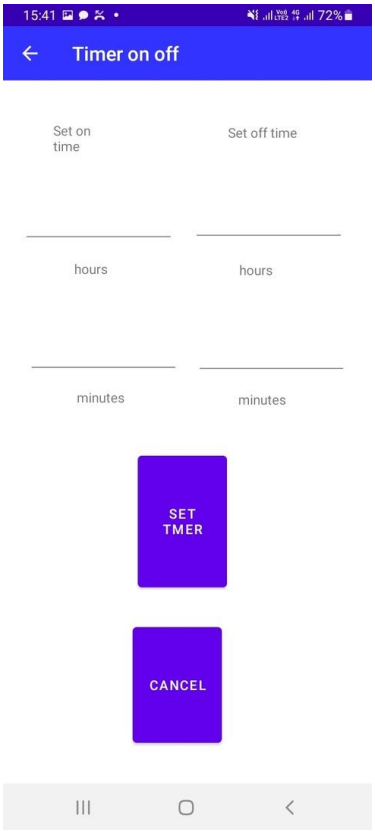

Рисунок 22 – Графический интерфейс TimerOnOffFragment

9. Класс GetLampResourceFragment предоставляет информацию об уровне износа лампы. Переменная batteryInfo из MainActivity, которая хранит в себе данные по лампе, полученные с устройства. Переменной resourceProcent типа Text присваивается значение batteryInfo. На экране отобразится данные об ресурсах лампы. На рисунке 23 изображен графический интерфейс экрана.

Рисунок 23 – Графический интерфейс GetLampResourceFragment

В этом разделе были даны необходимые сведения про инструменты разработки Android приложения и разобрана Android программа для клиентской части. Были показаны фрагменты кода приложения, изображения экранов приложения. Подробный код программы можно рассмотреть в Приложении Б.

4 Разработка печатной платы

4.1 Конструирование схемы в сервисе EasyEDA

Установка всех электронных компонентов схемы должна производиться на печатную плату. Разработка печатной платы производится в среде автоматизированного проектирования «EasyEDA».

Онлайн сервис EasyEDA предоставляет функционал для дальнейшего преобразования электрической принципиальной схемы в формат PCB, содержащий информацию о конструкции печатной платы, такую как расположение компонентов, проводников, отверстий и других элементов.

Этот формат используются для передачи информации о внешнем виде печатной платы из программы проектирования PCB в программу производства, которая занимается созданием физической копии печатной платы. На участке 10x10 удалось разместить две копии схемы, позволяя оптимизировать габаритные размеры. Разработанный участок PCB показан на рисунке 24.

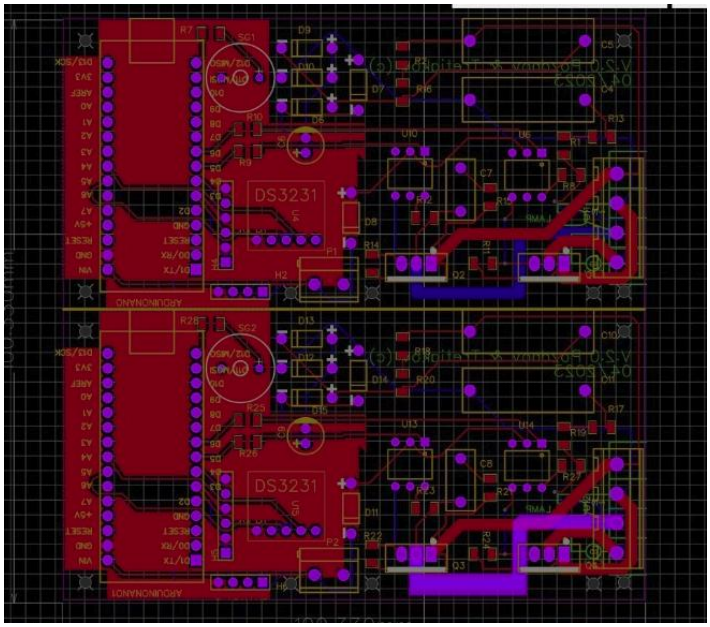

Рисунок 24 – PCB файл разрабатываемой платы

Так же на рисунке 25 приведен вид схемы в 3d формате. Это реализовано в среде разработки EasyEDA.

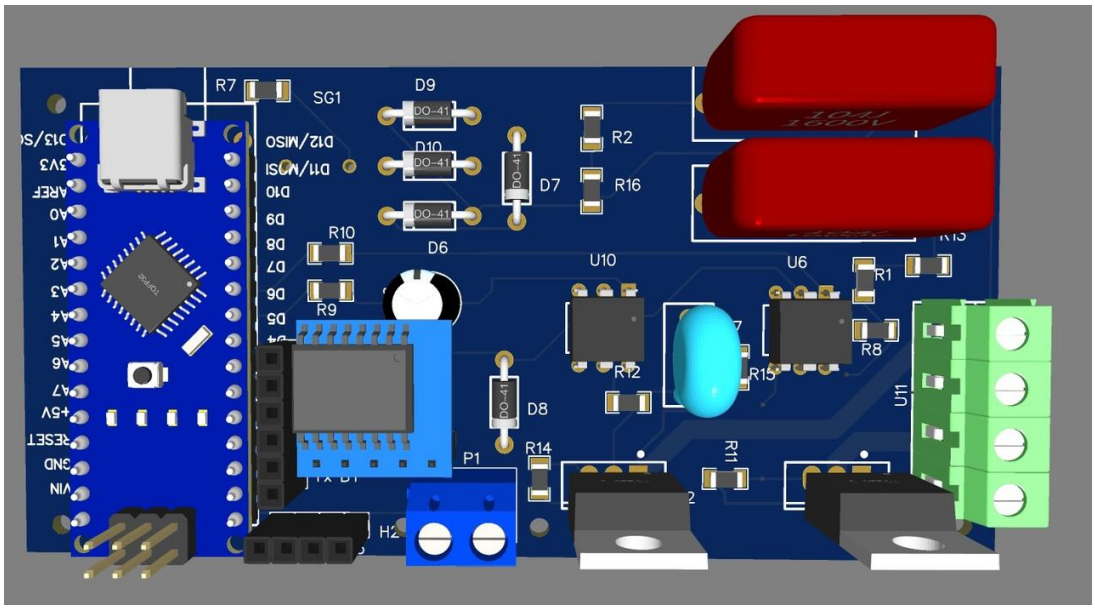

Рисунок 25 – 3d вид платы в программе EasyEDA

Плата доставлялась в течении месяца. Она полностью готова для монтажа на нее микроэлементов и имеет нанесенную шелкографию с картинками всех элементов, чтобы исключить ошибки монтажа на плату.

На рисунке 26 показано фото печатной платы.

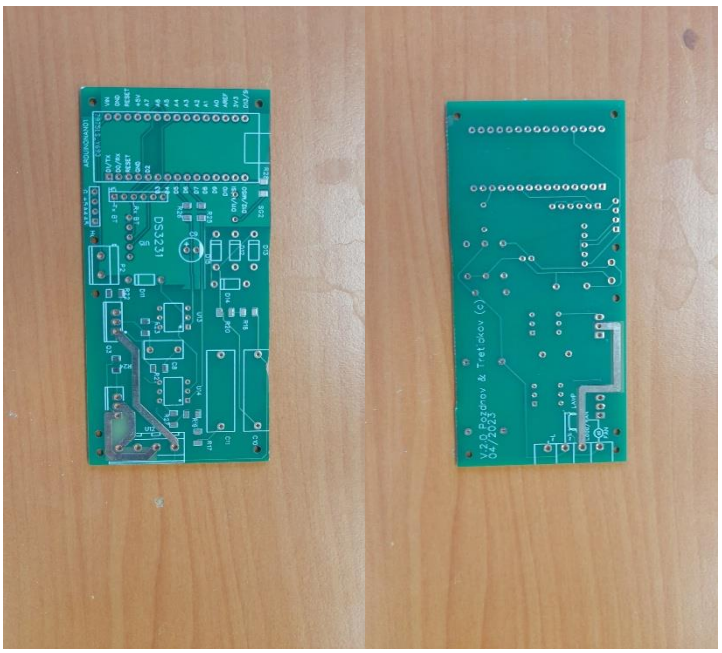

Рисунок 26 – Фото печатной платы

В данном разделе было кратко продемонстрировано описание среды разработки EasyEDA. Описан pcb файл и показана готовая печатная плата и печатный узел в виде 3d модели. Был пройден путь производства от принципиальной схемы устройства до готовой платы.

5. Экспериментальные исследования

Для демонстрации работоспособности и отладки печатного узла были проведены опыты с частями схемы по отдельности. Для измерения тока в схеме последовательно с нагрузкой устанавливался шунтовой резистор 0,39 Ом.

На осциллограмме (рисунок 27) виден импульсный ток при включении лампы. Этот ток протекает через симистор, который питает лампу. По осцилограмме можно определить значение пикового тока. Он составляет 0,85а. Как и ранее предполагалось это значение велико для опто симистра, однако для силового симистора, который был установлен в схему это значение не превышает допустимого.

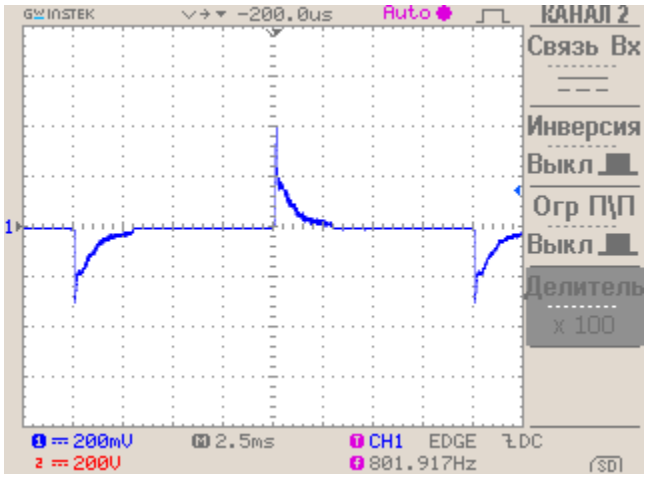

Рисунок 27 – Импульсный ток при включении лампы

На осциллограмме (рисунок 28) запечатлён момент включения лампы. Об этом говорит переход синусоидального напряжения снятого на симисторе (второй канал напряжения) в нулевое напряжение и появление импульсного тока на первом канале осциллографа.

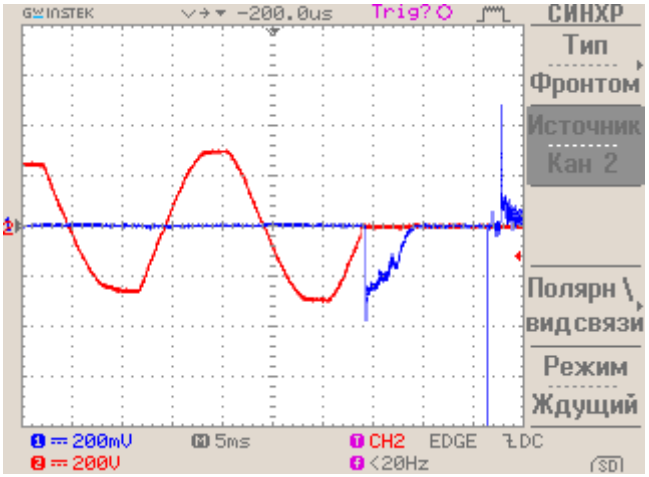

Рисунок 28 – Напряжение на симисторе и ток на токоизмерительном шунте в момент отключения лампы

Отдельно был записан момент пуска лампы от симистора. Как видно по осцилограмме (рисунок 29) напряжения на токоизмерительным шунте. Максимальное значение напряжения на нём составляет 7,5 в, что в переводе через сопротивление шунта составляет 15 А. Это ток допустим для работы симистора, однако точно выводит из строя оптимистр оптопары при варианте схемы без использования силового симистра.

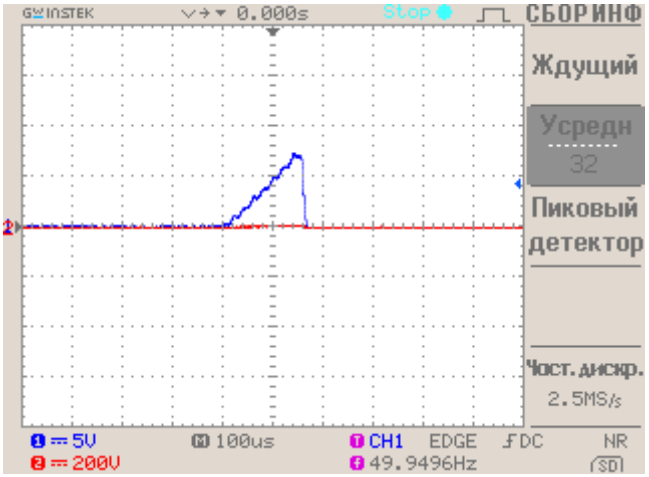

Рисунок 29 – Напряжение на токоизмерительном шунте в момент пуска лампы

На осцилограмме (рисунок 30) изображены напряжение на симисторе и ток потребляемой лампы. До момента отключения симистора видно, что ток

импульсного характера прекращается через симистор а напряжение плавно косинусоидальному закону увеличивается.

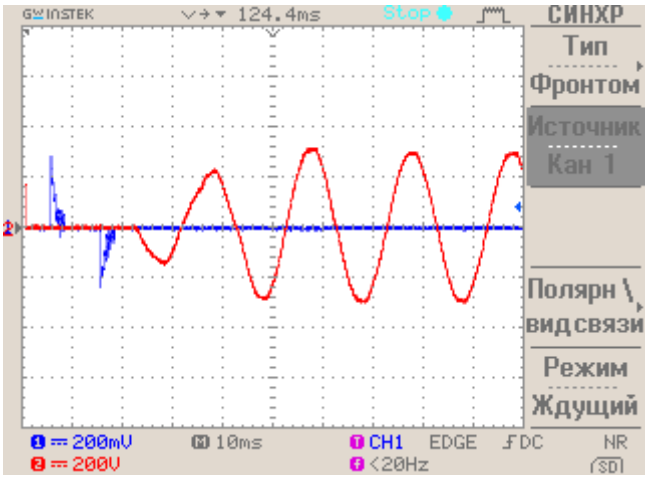

Рисунок 30 – Напряжение на симисторе и ток на токоизмерительном шунте в момент отключения лампы

На данной осцилограмме (рисунок 31) показан ток при процессе переключения лампы с низких на высокие обороты. По диаграмме заметно что в момент переключения происходят переходные процессы с увеличением тока лампы выше установившегося. Пиковое значение тока составляет 370 миллиампер.

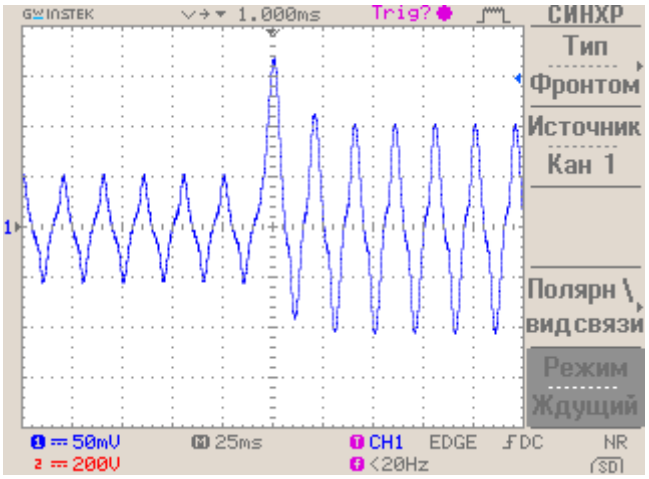

Рисунок 31 – Напряжение на токоизмерительном шунте в цепи вентилятора при переключении вентилятора с низких оборотов на высокие

На рисунках 32, 33 приведены осцилограммы токов на одной и на другой ступени управляющие скоростью вентилятора на двух ступенях управления.

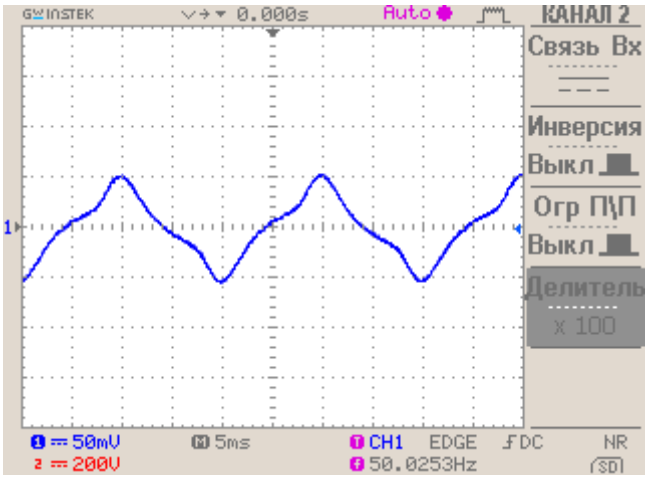

Рисунок 32 – Напряжение на токоизмерительном шунте в цепи вентилятора на низких оборотах

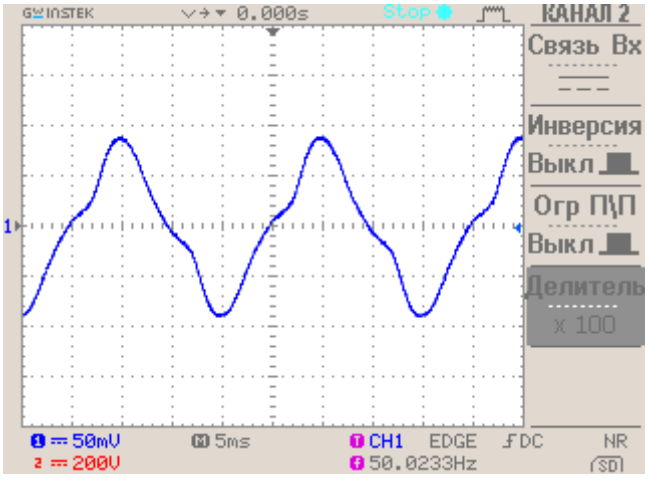

Рисунок 33 – Напряжение на токоизмерительном шунте в цепи вентилятора на высоких оборотах

На рисунке 34 на осциллограмме указано напряжение на тиристоре в выключенном состоянии. Пиковое значение напряжения составляет порядка 400 в.

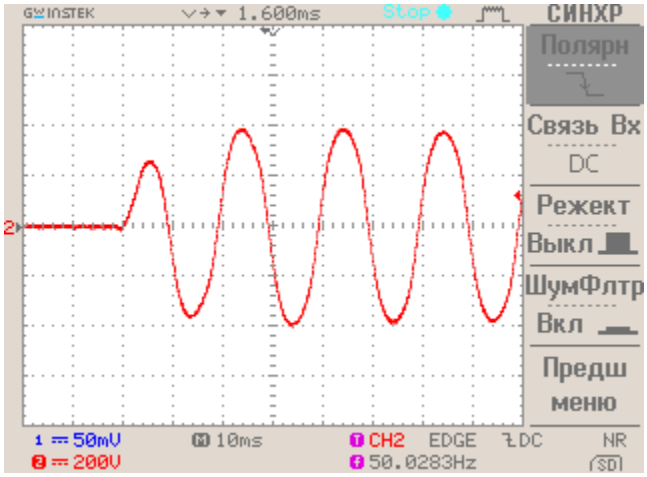

Рисунок 34 – Напряжение на тиристоре в выключенном состоянии

На осцилограммах на рисунках 35 и 36 указано напряжение и ток при включении вентилятора на низких оборотах и диаграммы напряжения и тока при включении на высоких оборотах.

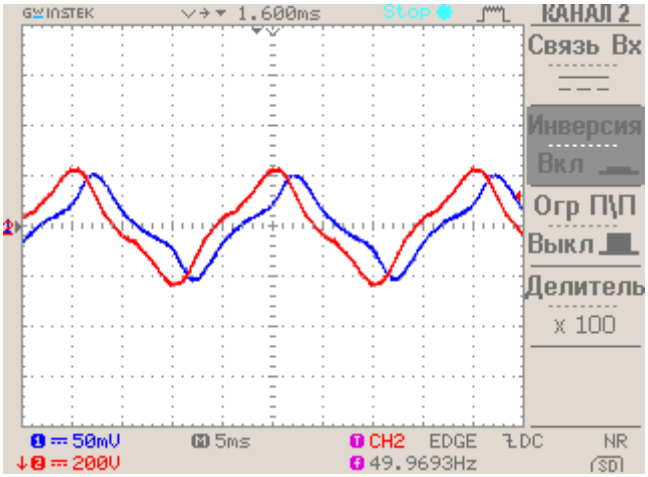

Рисунок 35 – Напряжение на вентиляторе и напряжение на токоизмерительном шунте в его цепи на низких оборотах

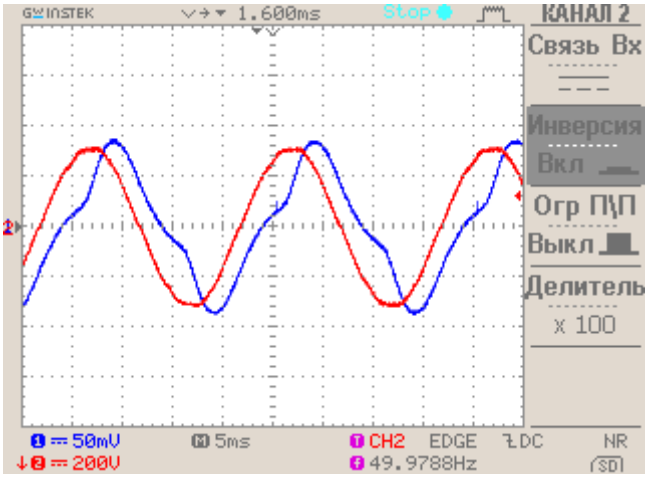

Рисунок 36 – Напряжение на вентиляторе и напряжение на токоизмерительном шунте в его цепи на высоких оборотах

Напряжение на тиристоре при выключение тиристора изображено на осцилограмме на рисунке 37.

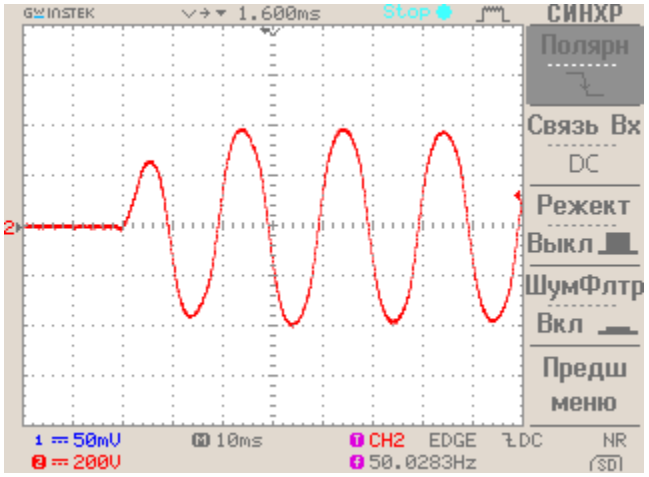

Рисунок 37 – Напряжение на тиристоре при выключении тиристора

При отсутствии демпфирующей цепи и при её наличии изображено на рисунке 38, 39.

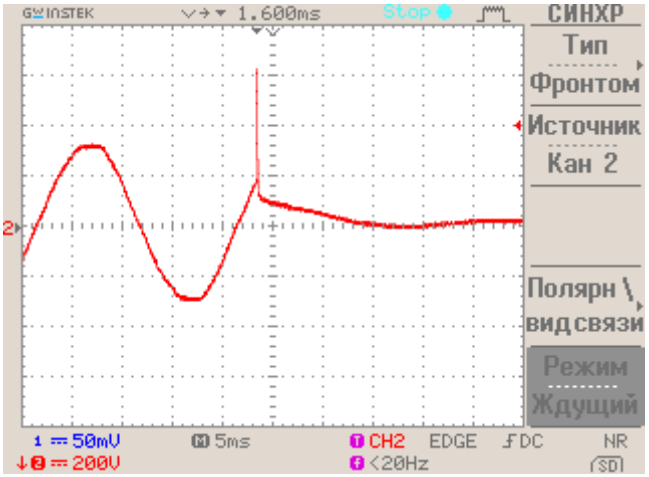

Рисунок 38 – Напряжение на нагрузке на этапе включения при отсутствии демпфирующей цепи

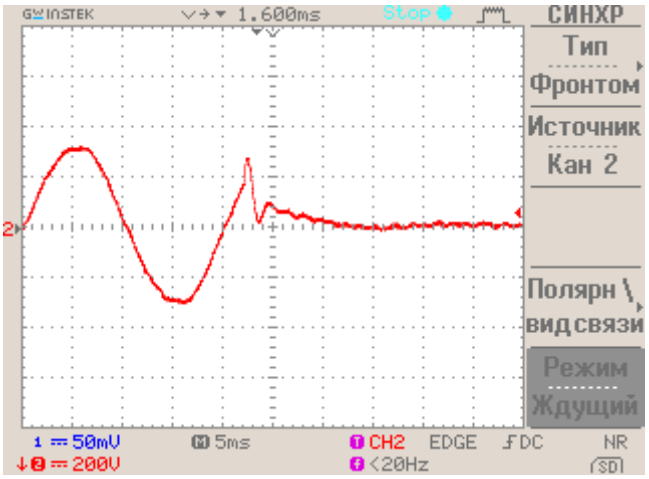

Рисунок 39 – Напряжение на нагрузке на этапе включения с демпфирующей цепью

Как видно наличие демпфирующей цепи уменьшает максимальное пиковая напряжение 600 до 300 в.

В этом разделе были описаны опыты над управляющей схемой. Приведены осциллограммы и дано описание под каждую из них. Эксперименты показали работоспособность схемы, в которой силовой симистор управляется симисторной оптопарой. Так же были получены режимы работы вентилятора на разных ступенях.

Заключение

По итогам выпускной квалификационной работы было разработано решение по обеззараживанию воздуха от вредоносных бактерий и вирусов путем разработки бактерицидного смарт рециркулятора.

Была разработана и произведена печатная плата и написано программное обеспечение под нее. Данное устройство имеет источник питания с сетевым напряжением 220 В, мощностью 20вт, ступенчатое регулирование потока обрабатываемого воздуха, таймер времени работы устройства, Оповещение об износе ультрафиолетовой лампы ,форму корпуса круглую, управление устройством по радиоканалу и управление устройством через Android-приложение.

Оно исправно выполняет свои задачи, не уступая аналогам на рынке, при этом имея ряд дополнительных функций. В планах на будущее планируется доработка платы управления и доработка и поддержка клиентской части. На текущий момент планируемые цели и задачи выпускной квалификационной работы успешно достигнуты.

Список используемых источников

1. ГОСТ 2.105.-95 Единая система конструкторской документации. Общие требования к текстовым документам.
2. ГОСТ 2.106-96 Единая система конструкторской документации. Текстовые документы.
3. Ван Вик Кристофер Дж., Седжвик Роберт. Алгоритмы на C++. Анализ структуры данных. Сортировка. Поиск. Алгоритмы на графах. 2019. – 1056с. ISBN 978-5-907144-21-7.
4. Васильев В. И., Гусев Ю. М., Миронов В. Н. Электронные промышленные устройства. Высш. шк., 1988. – 303 с.: ил. ISBN 5–06–001287–5.
5. Мазиди М. Р. Архитектура микроконтроллеров PIC. - Санкт-Петербург: Питер, 2017. - 448 с.
6. Майоров Р.Н. Разработка устройства для обеззараживания воздуха. Тольяттинский государственный университет, Институт машиностроения, Кафедра Промышленная электроника. URL: https://dspace.tltsu.ru/handle/123456789/24831/ (Дата обращения 15.06.2023)
7. Маккрейди К., Маккрейди Д. Программирование микроконтроллеров AVR на языке C. - М.: ДМК Пресс, 2018. - 416 с.
8. Мелешин В. И. Транзисторная преобразовательная техника. Москва: Техносфера, 2006. – 632с. ISBN 5-94836-051-2.
9. Ненахов, С.А. Инженерные расчеты импульсных регуляторов напряжения / С.А.Ненахов, А.Н.Кукаев // Электрическое питание .- 2005 .- №4 .- С.25 – 28.
10. O Fragment// Сайт «Habr» URL: https://habr.com/ru/articles/207036/ (Дата обращения 15.06.2023).
11. Прата Стивен. Язык программирования C++. Лекции и упражнения 2018. – 1244 с. ISBN 978-5-907114-00-5.

12. Рама Редди С. Основы силовой электроники. Москва: Техносфера. 2006. – 288с. ISBN 5-94836-055-5.

13. Техническое описание активного зуммера // Официальный сайт «IARDUINO» URL: https://wiki.iarduino.ru/page/zummer-trema-modul/ (Дата обращения 15.06.23).

14. Технические описание платы Arduino NANO. // Официальный сайт «arduino-nano.ru» URL: http://arduino-nano.ru/#desc (Дата обращения 15.06.23).

15. Файл манифеста // Сайт “Wikipedia” URL: https://en.wikipedia.org/wiki/Manifest_file (дата обращения: 20.06.2023).

16. Чиженко И. М., Руденко В. С., Сенько В. И. Основы преобразовательной техники. Учебн. Пособие для специальности Промышленная электроника. Высш. школа, 1974.

17. Broadcast Receiver // Сайт “Alexander Klimov” URL: https://developer.alexanderklimov.ru/android/theory/ (дата обращения: 12.06.2023).

18. Эккель Брюс. Филасофия java. Питер. -1168 с. ISBN 978-5-4461-1107-7.

19. Колисниченко Д. Н. Программирование для Android. БХВ. -288с. ISBN 978-5-457-44185-9.

20. Мартин Роберт. Чистый код. Питер. -464с. ISBN 978-5-4461-0960-9.

21. Mazidi, Muhammad Ali, McKinlay, Rolin D., Causey, Danny. PIC Microcontroller and Embedded Systems: Using Assembly and C for PIC18. - Prentice Hall, 2013. 832 с.

22. Monk, Simon. Programming Arduino: Getting Started with Sketches. - McGraw-Hill Education, 2016. 192 с. ISBN 978-1-484-20941-7.

23. Shared Preferences// Сайт “Fan Android.info” URL: https://www.fandroid.info/sharedpreferences-sohranenie-dannyh-v-postoyannoe-hranilishhe-android/ (дата обращения: 12.06.20123).

24. Purdum Jack: Beginning C for Arduino, Second Edition: Learn C Programming for the Arduino 2nd ed. Edition. 2015. – 414c. ISBN13:978- 1-484-20941-7.

Приложение А

Код программы для Arduino

```
// демо возможностей библиотеки
#include <microDS3231.h>
#include <EEPROM.h>
MicroDS3231 rtc;
#include <SoftwareSerial.h>
SoftwareSerial mySerial(7, 3); // RX, TX
//5,6 пин это на отправку команд на платы
//5 - это включение выключение
//6 - поменять скорость лампы дневной ночной режим


int i1;
int buz1=12; int buz2=10;  //D12, D10 подключение динамика
int a=10;
int res=0;
char res_str;
byte power=5;
byte fan = 6;
String timer_res_sum = "";
String timer_res_str = "";
byte Hours_timer;  // сколько часов работать таймеру
long time_timer;
long Time_lump=8000000; //ресурс лампы для примера в сек
String  res_sum;
byte i=0; //счетчик символов принятой строки в мониторе
byte SW; // Режим работы рециркулятора по времени 0 нет режима, 1 таймер, 2 график
long timer;
byte Hours_rtc ; // текцщие час и минуты вытащеные из RTC
byte Minutes_rtc ;
byte Hours_on, Minutes_on,Hours_off, Minutes_off; // час минуты вкл и выкл по графику
byte Seconds; byte Hours ; byte Minutes ; byte Day;byte Month; int year_ ; // час минута день год текущей даты
int address = 10;
float value = 0.0;
int minutes = 0;
int hours = 0;
int oldMinutes = 0;
bool lampWorkFlag = false;
```

```
byte Direct_hours;



void sound() { //подпрограмма пищания динамиком
long  timerSound=millis();
 while((millis()-timerSound)<2000)
        {digitalWrite(buz1,1);digitalWrite(buz2,0);
        delayMicroseconds(220);
        digitalWrite(buz1,0);digitalWrite(buz2,1);
        delayMicroseconds(220);
        }
        digitalWrite(buz1,0);digitalWrite(buz2,0);



  }

void(* resetFunc) (void) = 0;//объявляем функцию reset с адресом 0

void ledSignalOn() {
  digitalWrite(13, HIGH);   // turn the LED on (HIGH is the voltage level)
}

void ledSignalOff() {
  digitalWrite(13, LOW);
}

void onTimer(String timer_on_value) {
  if(rtc.getTimeString() != timer_on_value.substring(timer_on_value.lastIndexOf("#")+1)) {
    int hours =
timer_on_value.substring(timer_on_value.lastIndexOf("#")+1,timer_on_value.lastIndexOf("#")+3).toInt();
    int minutes =
timer_on_value.substring(timer_on_value.lastIndexOf("#")+4,timer_on_value.lastIndexOf("#")+7).toInt();
    int seconds =
timer_on_value.substring(timer_on_value.lastIndexOf("#")+7,timer_on_value.lastIndexOf("#")+9).toInt();

Serial.println(timer_on_value.substring(timer_on_value.lastIndexOf("#")+1,timer_on_value.lastIndexOf("#")+3));
    rtc.setTime( seconds, minutes, hours, 21, 8, 2022);
  }
  while(true) {
```

```
            //Serial.println(timer_on_value + " " + rtc.getTimeString().substring(3,6) + "rtc get time");
            //Serial.println(rtc.getTimeString());
            //Serial.println(value.substring(value.indexOf("#")+1,value.indexOf("#")+9));
            Serial.println(rtc.getTimeString().substring(0,2).toInt());

Serial.println(timer_on_value.substring(timer_on_value.indexOf("#")+1,timer_on_value.indexOf("#")+3).toInt());
            Serial.println(rtc.getTimeString().substring(3,6).toInt());



            // определение запуска функций на принятую комманду по блютусу
           if  (res_sum=="вкл") {
            digitalWrite(power,1);
            ledSignalOn();
            lampWorkFlag = true;} // комманда включения лампы с вентилятором
           if  (res_sum=="выкл") {
            digitalWrite(power,0);

            ledSignalOff();res_sum="";
            lampWorkFlag = false;}
            if (res_sum=="отмена") {
              break;
            }
            res_sum="";
            }
            if(rtc.getTimeString().substring(0,2).toInt() ==
timer_on_value.substring(timer_on_value.indexOf("#")+1,timer_on_value.indexOf("#")+3).toInt() &&


rtc.getTimeString().substring(3,6).toInt() ==
timer_on_value.substring(timer_on_value.indexOf("#")+4,timer_on_value.indexOf("#")+7).toInt()) {
              Serial.println("таймер сработал");
              ledSignalOn();
              break;
            }
          }
        }

        void onOffTimer(String value) {
```

```
            if(rtc.getTimeString() != value.substring(value.lastIndexOf("#")+1)) {
              int hours = value.substring(value.lastIndexOf("#")+1,value.lastIndexOf("#")+3).toInt();
              int minutes = value.substring(value.lastIndexOf("#")+4,value.lastIndexOf("#")+7).toInt();
              int seconds = value.substring(value.lastIndexOf("#")+7,value.lastIndexOf("#")+9).toInt();
              Serial.println(value.substring(value.lastIndexOf("#")+1,value.lastIndexOf("#")+3));
              rtc.setTime( seconds, minutes, hours, 21, 8, 2022);
            }
            while(true) {
              Serial.println(rtc.getTimeString().substring(0,2) + "rtc get time");
              //Serial.println(rtc.getTimeString());
              //Serial.println(value.substring(value.indexOf("#")+1,value.indexOf("#")+9));
              if (mySerial.available() > 0) {
                 while (mySerial.available()) // цикл сбора информации с порта блютус в string переменную
res_sum оканчивается знаком перевода строки с кодом 10
                 { res_str=mySerial.read();
                 if (res_str==13 or res_str==10) break;
                 res_sum = res_sum + res_str;
                 //Serial.println(res_sum);

               }
                 Serial.println(res_sum);
                 Serial.println(rtc.getTimeString());

              // определение запуска функций на принятую комманду по блютусу
             if  (res_sum=="вкл") {
              digitalWrite(power,0);
              ledSignalOn();
              lampWorkFlag = true;} // комманда включения лампы с вентилятором
             if  (res_sum=="выкл") {
              digitalWrite(power,0);


              ledSignalOff();res_sum="";
              lampWorkFlag = false;}
              if (res_sum=="отмена") {
                break;
              }
              res_sum="";
```

```
            }

            if(rtc.getTimeString().substring(0,2).toInt() ==
value.substring(value.indexOf("#")+1,value.indexOf("#")+3).toInt() && rtc.getTimeString().substring(3,6).toInt()
== value.substring(value.indexOf("#")+4,value.indexOf("#")+7).toInt()) {
              Serial.println("таймер сработал");
              ledSignalOn();
              lampWorkFlag = true;
            }
            if(rtc.getTimeString().substring(0,2).toInt() ==
value.substring(value.indexOf("/")+1,value.indexOf("/")+3).toInt() && rtc.getTimeString().substring(3,6).toInt() ==
value.substring(value.indexOf("/")+4,value.indexOf("/")+7).toInt()) {
              Serial.println("таймер сработал");
              ledSignalOff();
              lampWorkFlag = false;
              break;
            }
          }
        }

        void resourceCounter() {
         int sum = 0;
           for(int i = 7; i <= EEPROM[0]; i++) {
             sum = sum + EEPROM[i];
           }
           //mySerial.println("succsess1" + sum);
           //mySerial.println(sum*2+1);
          // Serial.println(sum*2+"1");
          mySerial.println(String(sum*2)+"e");
          }

           if(EEPROM[0] == 255) {
             EEPROM[0] = 7;

             }

           //Serial.println(EEPROM[0]);
```

```
}


void setup() {
  Serial.begin(9600);
  mySerial.begin(9600);
  pinMode(buz1, OUTPUT);pinMode(buz2, OUTPUT);
  pinMode(power, OUTPUT);pinMode(fan, OUTPUT);
  // проверка наличия модуля на линии i2c
  if (!rtc.begin()) {
    Serial.println("DS3231 not found");
    for(;;);
  }else {
    Serial.println("DS3231 found");
  }
  time_timer=millis();
  SW=0;
}
void loop() {

  while (mySerial.available()) // цикл сбора информации с порта блютус в string переменную res_sum оканчивается знаком перевода строки с кодом 10
  { res_str=mySerial.read();
  if (res_str==13 or res_str==10) break;
   res_sum = res_sum + res_str;
  }
    // определение запуска функций на принятую комманду по блютусу
   if  (res_sum=="вкл") {digitalWrite(power,1);ledSignalOn();Serial.println("Signal_On");lampWorkFlag = true;} // комманда включения лампы с вентилятором
   if  (res_sum=="выкл") {digitalWrite(power,0);ledSignalOff();Serial.println("Signal_Off"); lampWorkFlag = false;}// комманда выключения лампы с вентилятором
   if  (res_sum=="вклвент") {digitalWrite(fan,1);sound();}// комманда включения  вентилятора на высокие обороты
   if  (res_sum=="выклвент") {digitalWrite(fan,0);sound();} // комманда включения  вентилятора на низкие обороты
   if  (res_sum.indexOf("времтайм")!= -1) // задание таймера в часах, например  таймер05 - 5 часов
       { byte index = res_sum.indexOf("времтайм");
         Hours_timer = (res_sum.substring(index+16, index+18)).toInt();
         Serial.print("Hours_timer="); Serial.println(Hours_timer);
         sound();
```

```
                }
if(res_sum.indexOf("вклтайм") >= 0) {

  onTimer(res_sum);
  }
if(res_sum.indexOf("вклтайм2") >= 0) {

  onOffTimer(res_sum);
  }
if(res_sum == "высскор") {
 digitalWrite(6, HIGH);
 }

 if(res_sum == "ресурсы") {
 int sum = 0;
 for(int i = 7; i <= EEPROM[0]; i++) {
  sum = sum + EEPROM[i];
 }
 //mySerial.println("succsess1" + sum);
 //mySerial.println(sum*2+1);
// Serial.println(sum*2+"1");
mySerial.println(String(sum*2)+"e");
}
 if(EEPROM[0] == 255) {
  EEPROM[0] = 7;
  }
 //Serial.println(EEPROM[0]);

 //Serial.println(EEPROM.get(address, value));
 if(lampWorkFlag == true) {
  // Serial.println(minutes);
  if(hours == 0) {
     hours = rtc.getTimeString().substring(0,3).toInt();
    }
  if(minutes == 0){
     minutes = rtc.getTimeString().substring(3,5).toInt();
    }
  if(rtc.getTimeString().substring(0,3).toInt() != hours ) {
    minutes = rtc.getTimeString().substring(3,5).toInt();
    hours = rtc.getTimeString().substring(0,3).toInt();
```

```
      EEPROM[0] = EEPROM[0] + 1;
      EEPROM[EEPROM[0]] = 1;
      minutes = rtc.getTimeString().substring(3,5).toInt();
     } else {
      Serial.println("Succsess eeprom");
      //EEPROM.update(EEPROM[0], EEPROM[EEPROM[0]] + 1);
      EEPROM[EEPROM[0]] = EEPROM[EEPROM[0]] + 1;
      minutes = rtc.getTimeString().substring(3,5).toInt();
     }
     }
    }
    Serial.println(EEPROM[EEPROM[0]]);
     res_sum = "";
}
```

Приложение Б

Код для клиента

```
package com.example.a151022btlockapp

import android.app.AlertDialog
import android.app.TimePickerDialog
import android.bluetooth.BluetoothAdapter
import android.bluetooth.BluetoothDevice
import android.bluetooth.BluetoothSocket
import android.content.Context
import android.content.Intent
import android.content.IntentFilter
import android.content.SharedPreferences
import android.os.Bundle
import android.os.Handler
import android.os.Looper
import android.text.method.ScrollingMovementMethod
import android.util.Log
import android.widget.Switch
import android.widget.TimePicker
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import androidx.fragment.app.Fragment
import com.example.a151022btlockapp.bluetoothList.BluetoothDevicesListFragment
import com.example.a151022btlockapp.bluetoothList.BluetoothOffFragment
import com.example.a151022btlockapp.timers.LampResourceFragment
import com.example.a151022btlockapp.timers.TimerFragment
import com.example.a151022btlockapp.timers.TimerOnOffFragment
import com.google.android.things.bluetooth.BluetoothConnectionCallback
import java.io.BufferedInputStream
import java.io.IOException
import java.io.InputStream
import java.io.OutputStream
import java.text.DateFormat
import java.text.SimpleDateFormat
import java.util.*
const val APP_PREFERENCES = "APP_PREFERENCES"
class MainActivity : AppCompatActivity(), ControlsFragmentDelegate,
```

```
TimersFragmentDelegate, TimePickerDialog.OnTimeSetListener, BluetoothConnectionCallback{
    var batteryInfo: String? = null;
    lateinit var preferences: SharedPreferences
    val handler = Handler(Looper.getMainLooper())
    val bluetoothAdapter: BluetoothAdapter = BluetoothAdapter.getDefaultAdapter()
    val deviceSet: Set<BluetoothDevice> = bluetoothAdapter.getBondedDevices()
    private val TAG = "serega"
    private val REQ_ENABLE_BT = 10
    val BTN_SET_TIME = 31
    val ON_BUTTON = 30
    val RESOURCE_BUTTON = 40
    val OFF_BUTTON = 32
    val BT_BOUNDED = 21
    val CANCEL_BUTTON = 33
    val BTN_SET_TIME_ON_OFF = 34
    private var switchEnableBt: Switch? = null
    val BTN_SET_TIMER_AFTER_HOURS_MINUTS = 35
    val BTN_HIGH_SPEED = 34
    //private val connectedThread: ConnectedThread? = null
    //val connectedThread = ConnectThread()
    var connectThread: ConnectThread? = null
    var connectedThread: ConnectedThread? = null

    val dialog by lazy { AlertDialog.Builder(this)
        .setCancelable(false)
        .setTitle("Загрузка")
        .setMessage("Подождите, идет загрузка")
        .create() }

    val receiver by lazy { com.example.a151022btlockapp.broadcastreceiver.BroadcastReceiver() }



    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)


            var btDevice:BluetoothDevice? = null
```

```
    preferences = getSharedPreferences(APP_PREFERENCES,Context.MODE_PRIVATE)
    deviceSet.forEach {
        if (it.toString() == preferences.getString("PREF_SOME_VALUE",
                Context.MODE_PRIVATE.toString()
            ))
            btDevice = it
    }
    if (btDevice == null) {
        moveToDevicesListFragment()
    } else {
        connectThread = ConnectThread(btDevice, this)
        connectThread!!.start()
    }

    val filter = IntentFilter();
    filter.addAction(BluetoothDevice.ACTION_ACL_CONNECTED);
    filter.addAction(BluetoothDevice.ACTION_ACL_DISCONNECT_REQUESTED);
    filter.addAction(BluetoothDevice.ACTION_ACL_DISCONNECTED);
   // this.registerReceiver(broadcastReceiver, filter);


    val filter2 = IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED)
    /*val filter = IntentFilter()
filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_STARTED)
filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED)
filter.addAction(BluetoothDevice.ACTION_FOUND)
//registerReceiver(receiver, filter)*/
    //registerReceiver(bState, filter2)


    if (bluetoothAdapter == null) {
        Toast.makeText(this, R.string.bluetooth_not_supported, Toast.LENGTH_SHORT).show()
        Log.d(TAG, "onCreate: " + getString(R.string.bluetooth_not_supported))
        finish()
    }

    if (bluetoothAdapter.isEnabled()) {
        supportFragmentManager
            .beginTransaction()
            .add(R.id.container, BluetoothDevicesListFragment())
```

```
                .commit()
            //switchEnableBt.setChecked(true)
            //setListAdapter(MainActivity.BT_BOUNDED)
        } else {
            supportFragmentManager
                .beginTransaction()
                .add(R.id.container, BluetoothOffFragment())
                .commit()
        }

        if (connectThread != null && connectedThread != null) {
            supportFragmentManager
                .beginTransaction()
                .replace(R.id.container, ControlsFragment())
                .commit()
            //switchEnableBt.setChecked(true)
            //setListAdapter(MainActivity.BT_BOUNDED)
        }

    }

 override fun onDisconnected(bluetoothDevice: BluetoothDevice?, profile: Int) {
    super.onDisconnected(bluetoothDevice, profile)
    launchFragment(BluetoothDevicesListFragment())
 }

/* override fun onConnected(bluetoothDevice: BluetoothDevice?, profile: Int) {
    super.onConnected(bluetoothDevice, profile)
    launchFragment(ControlsFragment())
 }

 override fun onDisconnected(bluetoothDevice: BluetoothDevice?, profile: Int) {
    super.onDisconnected(bluetoothDevice, profile)
    launchFragment(BluetoothDevicesListFragment())

 }*/
```

```
override fun moveToTimersFragment() {
    launchFragment(TimersFragment())
}

override fun moveToLampResourceDataFragment() {
    launchFragment(LampResourceFragment())
}

override fun enableBt(checked: Boolean) {
    enableBt_(checked)
}
override fun moveToBtNoFragment() {
    supportFragmentManager
        .beginTransaction()
        .replace(R.id.container, BluetoothDevicesListFragment())
        .commit()
}

override fun moveToDevicesListFragment() {
    supportFragmentManager
        .beginTransaction()
        .replace(R.id.container, BluetoothDevicesListFragment())
        .commit()
}

override fun getBoundedBtDevices(): ArrayList<BluetoothDevice> {
    val deviceSet: Set<BluetoothDevice> = bluetoothAdapter.getBondedDevices()
    val tmpArrayList = ArrayList<BluetoothDevice>()
    if (deviceSet.size > 0) {
        for (device in deviceSet) {
            tmpArrayList.add(device)
        }
    }
    return tmpArrayList
}

override fun moveControlsFragment() {
    launchFragment(ControlsFragment())
}
```

```
override fun sendCommandOn() {
    enableButton(ON_BUTTON, 0, 0)
}

override fun sendCommandOff() {
    enableButton(OFF_BUTTON,0,0)
}

override fun sendTimerOnOffCommand(
    hoursOn: Int,
    minutesOn: Int,
    hoursOff: Int,
    minutesOff: Int
) {
    enableButton(BTN_SET_TIME_ON_OFF, hoursOn, minutesOn, hoursOff,  minutesOff)
}

override fun sendTimerCommand(hours: Int, minutes: Int) {
    enableButton(BTN_SET_TIMER_AFTER_HOURS_MINUTS, hours, minutes)
}

override fun sendCommandHighSpeed() {
    enableButton(BTN_HIGH_SPEED, 0, 0)
}

override fun sendCommandToGetResource() {
    enableButton(RESOURCE_BUTTON, 0,0)
    Thread.sleep(2000);
    launchFragment(GetLampResourceFragment())
}


override fun moveToTimerOnOffFragment() {
    launchFragment(TimerOnOffFragment())
}

override fun moveToTimerFragment() {
    launchFragment(TimerFragment())
}
```

```
private fun launchFragment(fragment:Fragment) {
    supportFragmentManager
        .beginTransaction()
        .addToBackStack(null)
        .setReorderingAllowed(true)
        .replace(R.id.container, fragment)
        .commit()
}

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == REQ_ENABLE_BT) {
        if (resultCode == RESULT_OK && bluetoothAdapter.isEnabled) {
            supportFragmentManager
                .beginTransaction()
                .replace(R.id.container, BluetoothDevicesListFragment())
                .commit()
        } else if (resultCode == RESULT_CANCELED) {
            enableBt_(true)
        }
    }
}

override fun onTimeSet(view: TimePicker?, hourOfDay: Int, minute: Int) {
    enableButton(BTN_SET_TIME, hourOfDay, minute)
}
private fun enableBt_(flag: Boolean) {
    if (flag) {
        val intent = Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE)

        startActivityForResult(intent, REQ_ENABLE_BT)
    } else {
        bluetoothAdapter.disable()
    }
}

class ConnectThread(device: BluetoothDevice?, val context: Context) : Thread() {
    val delegate = context as MainActivity
    private var bluetoothSocket: BluetoothSocket? = null
    private var success = false
```

```
init {
    try {
        val method = device!!.javaClass.getMethod(
            "createRfcommSocket", *arrayOf<Class<*>?>(
                Int::class.javaPrimitiveType
            )
        )
        bluetoothSocket = method.invoke(device, 1) as BluetoothSocket
        delegate.dialog.show()
    } catch (e: Exception) {
        e.printStackTrace()
    }
}

override fun run() {
    try {
        bluetoothSocket!!.connect()
        success = true
        //progressDialog.dismiss()
        delegate.dialog.dismiss()
    } catch (e: IOException) {
        e.printStackTrace()
        delegate.handler.post {
            delegate.dialog.dismiss()
            Toast.makeText(context, "Не могу соединиться!", Toast.LENGTH_SHORT)
                .show()
        }
        cancel()
    }
    if (success) {
        delegate.connectedThread = ConnectedThread(bluetoothSocket, context)
        delegate.connectedThread!!.start()

        delegate.handler.post {
            delegate.moveControlsFragment()
            delegate.dialog.dismiss()
        }
    }
}
```

```
if (minuteOn / 10 == 0) {
   minute = "0$minuteOn"
}
when (btn) {
   CANCEL_BUTTON -> {
      command = "отмена$simbol1"
      Toast.makeText(
         this@MainActivity,
         "отправил сигнал в ардуинку cancel",
         Toast.LENGTH_SHORT
      ).show()
   }
   OFF_BUTTON -> command = "выкл$simbol1"
   ON_BUTTON -> {
      command = "вкл$simbol1"
      Toast.makeText(
         this@MainActivity,
         "отправил сигнал в ардуинку on",
         Toast.LENGTH_SHORT
      ).show()
   }
   BTN_SET_TIME -> {
      command = "вклтайм#$hour:$minute#$timeText$simbol1"
      Toast.makeText(
         this@MainActivity,
         "отправил сигнал в ардуинку set time",
         Toast.LENGTH_SHORT
      ).show()
   }
   BTN_SET_TIMER_AFTER_HOURS_MINUTS -> {
      var timerMinutesAfterNow = timetextIntMinutes + minuteOn
      var timerHoursAfterNow = timetextIntHours + hourOn
      if (timerMinutesAfterNow > 60) {
         timerHoursAfterNow = timerHoursAfterNow + timerMinutesAfterNow / 60
         timerMinutesAfterNow = timerMinutesAfterNow - (timerMinutesAfterNow - 60)
      }
      command = "вклтайм#$timerHoursAfterNow:$timerMinutesAfterNow#$timeText$simbol1"
      Toast.makeText(
         this@MainActivity,
```

```
                    Toast.LENGTH_SHORT
                ).show()
            }
            BTN_HIGH_SPEED -> {
                command = "высскор"
                Toast.makeText(
                    this@MainActivity,
                    "отправил сигнал в ардуинку set time hours minutes",
                    Toast.LENGTH_SHORT
                ).show()
            }
            RESOURCE_BUTTON -> {
                command = "ресурсы"
            }
        }
        connectedThread!!.write(command)
    }
}

private fun enableButton(btn: Int, hourOn: Int, minuteOn: Int, hourOff: Int, minuteOff: Int) {
    if (connectedThread != null && connectThread!!.isConnect) {
        var command = ""
        var hour = hourOn.toString()
        var minute = minuteOn.toString()
        val simbol1 = 10.toChar()
        val currentDate = Date()
        val timeFormat: DateFormat = SimpleDateFormat("HH:mm:ss", Locale.getDefault())
        val timeText = timeFormat.format(currentDate)
        if (hourOn / 10 == 0) {
            hour = "0$hourOn"
        }
        if (minuteOn / 10 == 0) {
            minute = "0$minuteOn"
        }
        when (btn) {
            BTN_SET_TIME_ON_OFF -> {
                command =
                    "вклтайм2" + "#" + hour + ":" + minute + "/" + hourOff + ":" + minuteOff + "#" + timeText
+ simbol1
```