МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»

Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____ 2021 г.

СИСТЕМА КОНТРОЛЯ СОСТОЯНИЯ ЗДАНИЯ С ИСПОЛЬЗОВАНИЕМ УМНЫХ ДАТЧИКОВ И ЧАТ-БОТА TELEGRAM

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____В.А. Парасич
«___»_____2021 г.

Автор работы,
студент группы КЭ-405
_____Г.И. Малов
«___»_____2021 г.

Нормоконтролёр,
ст. преп. каф. ЭВМ
_____ С.В. Сяськов
«___»_____2021 г.

Челябинск-2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра «Электронные вычислительные машины»

УТВЕРЖДАЮ
Заведующий кафедрой ЭВМ
_____ Г.И. Радченко
«___»_____2021 г.

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405
Малову Глебу Ивановичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы: «**Система контроля состояния здания с использованием умных датчиков и чат-бота Telegram» утверждена приказом по университету от 26 апреля 2021 г. №714-13/12 (приложение №73).
2. **Срок сдачи студентом законченной работы:** 1 июня 2021 г.
3. **Исходные данные к работе:**

   Реализовать функционал устройств:

   – реализация API Telegram для работы с чат-ботом
   – считывание данных с датчиков;
   – контроль за устройствами управления.

4. **Перечень подлежащих разработке вопросов:**

   – обзор аналогичных решений разрабатываемого устройства;
   – установка требований к системе;

– проектирование архитектуры разрабатываемого устройства;
– выбор датчиков для мониторинга за участком;
– выбор микроконтроллера для хаба;
– выбор языка программирования для сервера;
– выбор микроконтроллера для передатчиков;
– разработка модели создаваемого устройства;
– тестирование получившейся системы.

5. **Дата выдачи задания:** 1 декабря 2020 г.

Руководитель работы _____/ *В.А. Парасич* /

Студент _____/ *Г.И. Малов* /

КАЛЕНДАРНЫЙ ПЛАН

Этап	Срок сдачи	Подпись руководителя
Введение и обзор литературы	01.03.2021	
Разработка модели, проектирование	01.04.2021	
Реализация системы	01.05.2021	
Тестирование, отладка, эксперименты	15.05.2021	
Компоновка текста работы и сдача на нормоконтроль	24.05.2021	
Подготовка презентации и доклада	30.05.2021	

Руководитель работы _____/ *В.А. Парасич* /

Студент _____/ *Г.И. Малов* /

Аннотация

Г.И. Малов. Система контроля состояния здания с использованием умных датчиков и чат-бота Telegram. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2021, 82 с., 32 ил., библиогр. список – 18 наим.

В рамках выпускной квалификационной работы производится создание программно-аппаратного комплекса слежения и контроля за домом с использованием чат-бота Telegram. Выполняется обзор доступных технологических решений и выбор самых подходящих. Составляется проект системы и прорабатываются отдельные подсистемы. В результате появляется модель аппаратного комплекса, которая разрабатывается и тестируется.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Домашняя автоматизация в современных условиях — чрезвычайно гибкая система, которую пользователь конструирует и настраивает самостоятельно в зависимости от собственных потребностей. Это предполагает, что каждый владелец умного дома самостоятельно определяет, какие устройства куда установить и какие задачи они будут исполнять.

Наиболее распространенные примеры автоматических действий в «умном доме» — автоматическое включение и выключение света, автоматическая коррекция работы отопительной системы или кондиционера и автоматическое уведомление о вторжении, возгорании или протечке воды.

Система умного дома включает три типа устройств:

- контроллер (хаб) — управляющее устройство, соединяющее все элементы системы друг с другом и связывающее её с внешним миром;
- датчики (сенсоры) — устройства, получающие информацию о внешних условиях;
- актуаторы — исполнительные устройства, непосредственно исполняющие команды. Это самая многочисленная группа, в которую входят умные (автоматические) выключатели, умные (автоматические) розетки, умные (автоматические) клапаны для труб, сирены, климат-контроллеры и так далее.

В большинстве современных умных домов контроллер общается с остальными устройствами системы через радиосигналы. Самые распространенные стандарты радиосвязи для домашней автоматизации — Z-Wave (частота зависит от страны, в Европе 868 МГц, в России 869 МГц) и ZigBee (868 МГц или 2,4 ГГц), Wi-Fi (2,4 ГГц), Bluetooth (2,4 ГГц). Почти все они используют шифрование данных (AES-128), в Wi-Fi применяется

шифрование WPA, WPA2 или WEP. Для связи с внешним миром контроллер, как правило, подключается к интернету.

Системы безопасности:

- датчики движения, датчики присутствия, датчики вибрации, датчики разбития стекла, датчики открытия окна или двери;
- видеонаблюдение;
- видеодомофоны и видеоглазки;
- электронные замки (умные замки, смартлоки) и модули управления воротами;
- сирены.

Эти устройства позволяют сконструировать подходящую систему безопасности, от сравнительно простой до достаточно сложной.

Среди основных алгоритмов:

- регистрация нежелательного проникновения;
- уведомление владельцев;
- включение сирены;
- запуск видеосъемки;
- запирание входных или межкомнатных дверей.

Системы безопасности умного дома интегрируются с охранными системами, по тревоге высылающими группы реагирования. В большинстве стран рынок охранных систем существует достаточно давно, в то время как системы умного дома стали широко распространяться лишь в 2010-х годах. Отдельные поставщики охранных услуг позволяют интегрировать свою сигнализацию с умными устройствами, которые устанавливает сам пользователь, либо соглашаются высылать группы реагирования по сигналам тревоги с таких устройств.

Электронные замки, видеодомофоны и видеоглазки позволяют также организовать систему контроля доступа с возможностями дистанционного управления, видеозаписи и так далее.

Управление освещением:

- умные выключатели и диммеры;
- модули управления шторами, жалюзи и рольставнями;
- RGB- и RGBW-контроллеры для управления светодиодными светильниками, прежде всего светодиодными лентами;
- датчики движения и присутствия;
- датчики освещенности.

Такие устройства позволяют автоматизировать управление светом и чаще всего используются, чтобы:

- автоматически включать свет, когда люди входят в помещение, и выключать, когда выходят;
- автоматически поддерживать освещенность на постоянном уровне, регулируя яркость светильников и положение жалюзи или штор;
- автоматически регулировать освещенность в зависимости от сезона и времени суток или по другим заранее заданным правилам.

Управление климатом:

- датчики влажности;
- датчики температуры;
- термостаты для поддержания постоянной температуры или её автоматического регулирования;
- терморегуляторы для управления мощностью батарей отопления;

– климат-контроллеры, передающие команды умного дома на технику предыдущих поколений, которая управляется обычными дистанционными пультами, прежде всего на кондиционеры;
– гигростаты для поддержания постоянной влажности или её регулирования.

Основная задача устройств умного дома в этом случае — автоматически регулировать работу климатических систем так, чтобы одновременно обеспечить комфортный микроклимат и сократить расходы на его поддержание. Наиболее распространенные функции умного дома здесь:

– автоматически поддерживать комфортную температуру в помещениях, где находятся люди;
– автоматически снижать мощность батарей и кондиционеров в отсутствие людей и ночью;
– автоматически поддерживать влажность, комфортную для людей и подходящую для помещения и предметов обстановки;
– автоматически вентилировать помещения и очищать воздух, поддерживая комфортное качество воздуха.

Острота проблемы безопасности данных зависит от применения устройств. Чем серьёзнее потенциальные последствия, тем опаснее взлом. Если для автоматизации в промышленности или медицинских учреждениях риски могут быть чрезвычайно велики, то для домашней автоматизации, отвечающей за управление светом или системой датчиков, они значительно ниже.

Производители создают устройства на собственном программном обеспечении, с собственными мобильными приложениями и контроллерами. Это усложняет взаимодействие устройств и создание единой сети из устройств различных производителей.

Крайний случай представляет собой применение проприетарного софта с закрытым кодом. Работающие на таком программном обеспечении устройства зачастую вообще невозможно связать с устройствами других производителей.

Некоторые протоколы, в первую очередь Z-Wave, создавались с целью преодолеть эту проблему и дать производителям возможность создавать устройства, способные взаимодействовать друг с другом. Конструировать устройства с возможностью свободного взаимодействия друг с другом стали также производители на ZigBee. Готовность производителей создавать устройства на одном и том же стандарте — один из путей решения проблемы, они объединяют свои усилия в рамках единого консорциума (например, Z-Wave Alliance) и совместно развивают стандарт.

Второй путь — разработка устройств, способных взаимодействовать с разными стандартами. Некоторые производители встраивают в главный контроллер домашней сети возможность управлять устройствами на нескольких стандартах, например на Z-Wave, ZigBee, Bluetooth LE и KNX. В этом случае устройства все ещё не могут взаимодействовать напрямую, но получают возможность работать друг с другом через хаб, который переводит сигналы с одного стандарта на другой.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. ОБЗОР АНАЛОГОВ

На рынке автоматизации зданий и создании умных домов работает огромное количество как крупных, так и мелких компании. Большинство систем созданы профессионалами и обладают огромными возможностями [1]. В этом разделе будут рассмотрены некоторые из них.

Xiaomi Smart Home

Xiaomi, или Сяоми (в русском произношении) — китайская компания, которая была образована в 2010 году. В конце 2014 года Xiaomi представила систему домашней автоматизации, состоящую из умной розетки (Mi Smart Power Plug), камеры наблюдения (Yi Camera), умной лампочки (Yeelight LED) и инфракрасного блока управления бытовой техникой (IR Remote Controller).

Эти устройства объединяются в единую систему при помощи приложения MiHome и управляются по протоколу Wi-Fi 2.4 GHz.

Рисунок 1.1.1 – Комплект Xiaomi Smart Home

Преимущества Xiaomi Smart Home:

– возможность выбрать уже готовый к пользованию продукт;
– прост в использовании.

Недостатки Xiaomi Smart Home:

– высокая стоимость оборудования;
– полная документация на китайском языке.

Умный дом от “Ajax”

Компания Ajax systems занимается проектированием, монтажом и техническим обслуживанием систем безопасности. Она также занимается установкой системы умный дом, которая сотрудничает с частными охранными предприятиями и в случае сигнализации могут незамедлительно выехать на место.

Рисунок 1.1.2 - Ретранслятор сигнала системы безопасности Ajax

Преимущества системы Умный дом от “Ajax”:

- частное охранное предприятие;
- управление устройствами с телефона.

Недостатки системы Умный дом от “Ajax”:

- невозможность установки оборудования без специальной подготовки или помощи специалиста;
- высокая стоимость оборудования (от 15000 рублей минимальный комплект установки и ежемесячное обслуживание от охранных предприятий, 500+ рублей в месяц).

EasySmartBox

Готовые к установке индивидуальные комплекты системы Умный Дом на базе немецкого профессионального оборудования.

Рисунок 1.1.3 – Минимальный комплект EasySmartBox

Преимущества EasySmartBox:

– установка стандартного комплекта не требует навыков программирования или опыта работы с аналогичными системами. Может быть освоена любым человеком, имеющим некоторую техническую подготовку;
– программа управления имеет интуитивно понятный и красивый интерфейс. Управлять всеми системами дома просто и удобно.

Недостатки EasySmartBox:

– высокая стоимость оборудования.

ВЫВОД

Таким образом, имеется несколько готовых к использованию комплектов устройств для “Умного” дома. Все они имеют недостатки в виде высокой стоимости комплектации оборудования и производства за рубежом. Следовательно, создание приложения, контролирующего весь спектр домашних устройств, которые находятся в более дешевом ценовом разделе, является перспективной задачей.

1.2. АНАЛИЗ ОСНОВНЫХ ТЕХНОЛОГИЧЕСКИХ РЕШЕНИЙ

1.2.1. Обзор используемых датчиков

Датчик движения D203S

Датчик регистрирует движущиеся объекты, отличные по температуре [6].

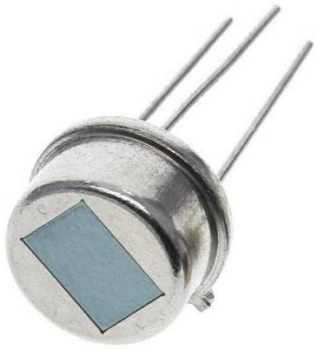

Рисунок 1.2.1.1 - Датчик движения D203S

Датчик влажности почвы (вилы) с компаратором на LM393

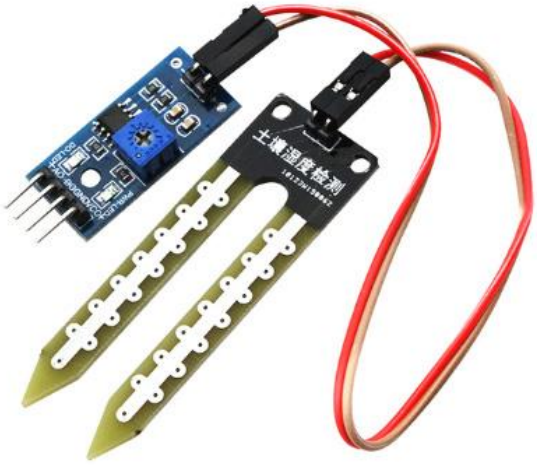

Рисунок 1.2.1.2 – Датчик влажности почвы и компаратор на LM393

Датчик влажности и температуры DHT11

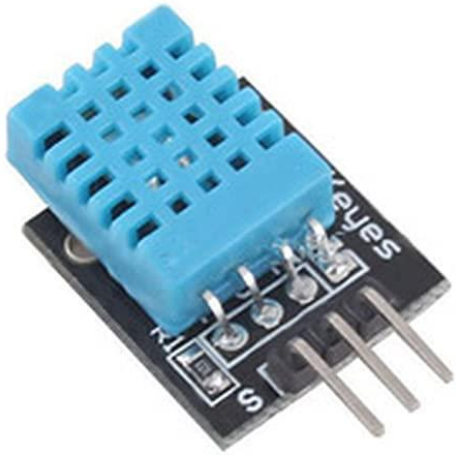

Рисунок 1.2.1.3 – Датчик влажности и температуры DHT11

Датчик уровня жидкости

Простой и надежный датчик уровня воды предназначен для контроля уровня воды в различных емкостях [7].

Рисунок 1.2.1.4 – Датчик уровня жидкости

nRF24L01 микросхема, которая дает возможность организовать беспроводную связь с устройствами через радиосвязь.

Рисунок 1.2.1.5 – Приемопередатчик nRF24L01

1.2.2. Выбор микроконтроллера

Микроконтроллер — это специальная микросхема, предназначенная для управления различными электронными устройствами, сочетает на одном кристалле функции процессора и периферийных устройств.

Вычислительных способностей современных микроконтроллеров будет достаточно для выполнения всех задач, стоящих перед данной системой.

Самые популярными микроконтроллерами, используемыми для таких систем, являются: STM от STMicroelectronics, AVR от Atmel, PIC от Microchip. Микроконтроллеры каждой из компаний обладает своими преимуществами и недостатками.

Решения на базе STM активно используются благодаря производительности микроконтроллера, его удачной архитектуре, малом энергопотреблении, небольшой цене. В настоящее время STM32 состоит уже из нескольких линеек для самых разных предназначений. Низкая цена, удобство использования, большой выбор сред разработки, микроконтроллера дают возможность для активного развития и производства систем на базе микроконтроллеров данного типа.

Микроконтроллеры AVR от Atmel используются по всему миру в различных системах. Благодаря большому функционалу, низкой стоимости и высокой эффективности микроконтроллеров этой компании, их устройства используются в системах различного назначения. У Atmel много бесплатно распространяемых программных продуктов. Сторонние производители выпускают полный спектр компиляторов, разъемов и адаптеров.

Однако, с 2016 года набирает популярность микроконтроллер ESP8266, от компании Espressif. Основной особенностью данного микроконтроллера является наличие Wi-Fi интерфейса. Это дает возможность разрабатывать умные устройства с беспроводным управлением. Помимо встроенного Wi-Fi интерфейса, этот микроконтроллер отличается высокой надежностью и эффективностью. Микроконтроллер может исполнять данные ему программы из внешней флеш-память посредством интерфейса SPI.

В данной работе выбран микроконтроллер ESP8266, имеющий следующие технические параметры.

- поддерживает WiFi протоколы 802.11 b/g/n с WPA, WPA2;
- может работать по Wi-Fi в режимах: станция, программная точка доступа, программная точка доступа + станция;
- имеет UART, HSPI, I2C, I2S, инфракрасный интерфейс удаленного управления;
- рабочее напряжение от 2,2 до 3,6 В;
- имеет встроенную флэш-память 4 Мб;
- потребляемый ток до 300 мА в зависимости от выбранного режима.

Доступ к внутренней периферии можно получить не из документации, а из API набора библиотек. Производителем указывается приблизительное количество ОЗУ – 50 кБ [17].

Особенности плат для микроконтроллера ESP8266:

– удобное подключение к компьютеру – через USB кабель;
– наличие встроенного преобразователя напряжения 3,3В;
– наличие 4 Мб флеш-памяти;
– встроенные кнопки для перезагрузки и перепрошивки.

В данной работе в качестве хаба будет использоваться платформа NodeMCU ESP8266.

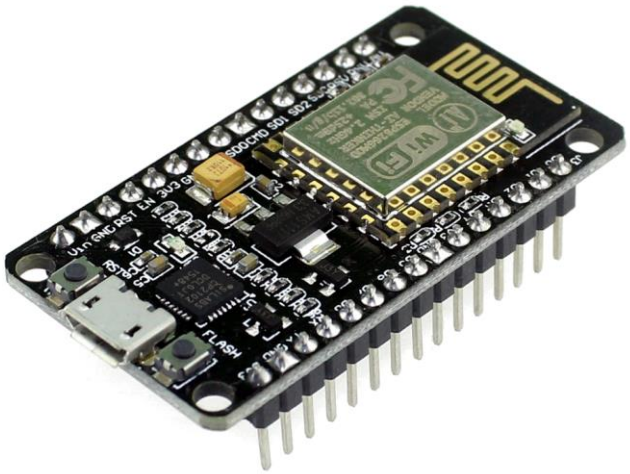

Рисунок 1.2.2.1 – Платформа NodeMCU ESP8266

NodeMcu – платформа на основе ESP8266 для создания различных устройств интернета вещей. Рядом кнопки для отладки и перезагрузки микроконтроллера. Программирование прошивки данной платформы будем производить в среде Arduino IDE.

Данный микроконтроллер, благодаря своей структуре и установке, может использоваться в качестве центрального управляющего устройства в нашей системы.

В качестве передатчиков будут использоваться две платы Arduino Nano, построенные на микроконтроллере ATmega168 (рисунок 1.2.2.2).

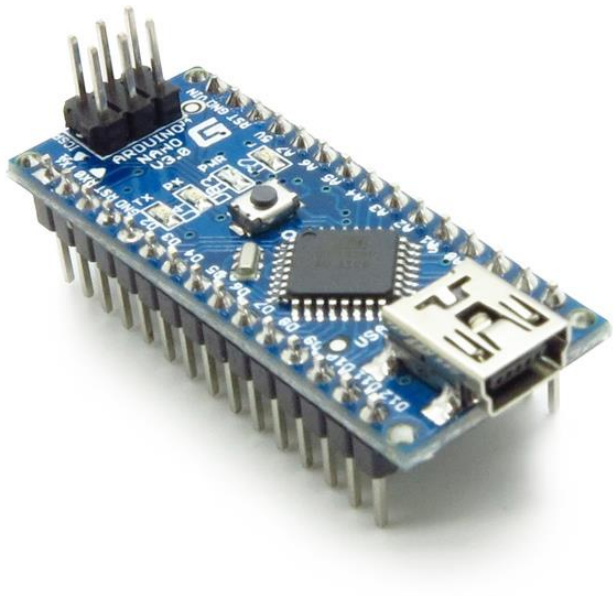

Рисунок 1.2.2.2 – Arduino Nano

Данный микроконтроллер имеет сравнительно небольшие размеры (1.85 см x 4.2 см), 16 Кб flash-памяти для хранения кода программы (при этом 2 Кб используются для загрузчика), 14 Цифровые Входов/Выходов (6 из которых могут использоваться как выходы ШИМ) и 8 аналоговых входов.

Платформа программируется посредством ПО Arduino. Из меню Tools> Board выбирается «Arduino Nano w/ ATmega168»

Микроконтроллеры ATmega168 поставляются с записанным загрузчиком, облегчающим запись новых программ без использования внешних программаторов. Связь осуществляется оригинальным протоколом STK500 [13].

К данным микроконтроллерам будет подключены посредством SPI модуль nRF24L01 и необходимая периферия, включающая в себя датчики или управляемые устройства.

1.2.3. Выбор технологии передачи данных

Проводное или прямое подключение периферии.

Проводное подключение, безусловно, является одним из вариантов подключения устройств к нашей системе. Преимущества в виде скорости работы, наличия центрального питания, легкости в подключении дают возможность поставить нашу систему только в условиях малого объема помещения. В остальных случаях подключение по проводной связи может повлечь лишние затраты и занимать огромное количество места.

Беспроводное подключение.

К беспроводным подключениям можно отнести:

- Wi-Fi;
- Bluetooth;
- LoRaWAN;
- Радиосвязь;

Все эти типы беспроводных подключений являются перспективными и используемыми, у каждого из них есть свои преимущества и недостатки.

Wi-Fi работает на расстоянии, которое предоставляет для раздачи роутер, также подвержен помехам из-за толстых стен. Высокая скорость работы, но большое для датчиков энергопотребление дает возможность использовать этот вид беспроводного подключения только для связи хаба с сетью Интернет.

Bluetooth в настоящее время развивается и также может предложить радиолюбителям свой вариант подключения датчиков, с помощью Bluetooth low energy. Данный тип связи дает возможность датчикам работать более года на одной миниатюрной батарейке типа таблетка без подзарядки. Также имеет довольно ограниченный радиус связи.

LoRaWAN создана для передачи небольшого количества информации на большие расстояния и очень энергоэффективно [5]. LoRa обеспечивает скорость передачи в беспроводном канале от 0.3 до 50 кбит/с [4]. Такая скорость является довольно малой, что компенсируется расстоянием, на котором может работать данный тип связи.

Радиосвязь — это еще один беспроводной тип радиосвязи. Частный пример этому - nRF24L01. Это микросхема с пониженным потреблением энергии, скоростью передачи данных до 2Мбит/с для диапазона 2,4 ГГц. При помощи этого модуля можно связать несколько устройств для передачи данных по радиоканалу. Данная микросхема может связать устройства на расстоянии до 100 метров и её цена намного ниже, чем у микросхем других типов беспроводной связи.

На данном этапе лучше всего по характеристикам и ценовому сегменту подходит NRF беспроводная связь, которая предоставляет нужные для нашей системы параметры беспроводной связи и подключения к ней устройств.

1.2.4. Выбор языка программирования для сервера

Для разработки веб-приложения можно использовать следующие языки: Python, Pascal, Basic, C, Assembler. Далее рассмотрим достоинства и недостатки каждого языка.

Python — высокоуровневый язык программирования общего назначения с динамической типизацией и автоматическим управлением памятью. Язык является полностью объектно-ориентированным [15].

Преимущества языка Python:

- свободно типизированный язык программирования;
- не имеет традиционных статических переменных;
- имеется большое количество IDE;
- имеет большое сообщество.

Недостатки языка Python:

- низкое быстродействие;
- язык “Высокого уровня”. Значит язык абстрагирован от мельчайших деталей базовой системы;

– изначальная ограниченность средств для работы с базами данных.

Pascal — один из наиболее известных языков программирования, используется для обучения программированию в старших классах и на первых курсах вузов, является основой для ряда других языков [2].

Преимущества Pascal:

– простой в обучении язык;
– большое количество ответвлений;
– большое количество документации;
– огромное сообщество.

Недостатки Pascal:

– он используется в крайне узких и сложно электронных устройствах;
– довольно старый язык программирования;
– имеет ряд ограничений в первоначальном виде.

C — компилируемый статически типизированный язык программирования общего назначения, разработанный в 1969—1973 годах сотрудником Bell Labs Деннисом Ритчи как развитие языка Би. Первоначально был разработан для реализации операционной системы UNIX, но впоследствии был перенесён на множество других платформ [16].

Преимущества языка C:

– один из наиболее популярных языков программирования;
– синтаксис языка C является основой для многих других языков программирования;
– сочетает в себе черты языков низкого и высокого уровней;
– большое число готовых библиотек, примеров, статей и книг;
– огромное количество компиляторов;
– работа с большим количеством микроконтроллеров.

Недостатки языка C:

– долгое время разработки, отладки;

– накладываются ограничения, связанные с простой АЛУ (Арифметико-логическое устройство), малым числом ресурсов и низкой разрядностью;
– возможно непонимание кода, написанного другими программистами.

Assembler — тип языка программирования низкого уровня, представляющий собой формат записи машинных команд, удобный для восприятия человеком [3].

Преимущества Assembler:

– программирование на Assembler — это обращение со всей периферией контроллера напрямую;
– простая отладка программ;
– максимальная адаптация под соответствующий процессор;
– максимальная оптимизация программ, как по скорости выполнения, так и по размеру.

Недостатки Assembler:

– трудоемкость чтения;
– трудоемкость написания программы больше, чем языке высокого уровня;
– ассемблер малопригоден для совместных проектов;
– непереносимость на другие платформы, кроме совместимых.

Для создания системы было принято взять язык С. Главными его преимуществами являются:

– простота в использовании;
– поддержка большого количества баз данных;
– огромное количество документации и литературы;
– эффективность использования.

1.2.5. Обзор используемого мессенджера

Telegram — кроссплатформенный мессенджер с функциями VoIP, позволяющий обмениваться текстовыми, голосовыми и видео сообщениями, стикерами и фотографиями, файлами многих форматов. Также можно совершать видео- и аудио звонки, организовывать конференции, многопользовательские группы и каналы. Клиентские приложения Telegram доступны для Android, iOS, Windows Phone, Windows, macOS и GNU/Linux [8].

Информация и файлы клиентов на серверах компании хранятся зашифрованными, причем разные коды применены к разным частям данных. Чтобы их расшифровать, придется поработать. Причем, кластеры информации размещены на серверах, расположенных в разных странах. Выемка информации с одного из дата-центров, например, от правоохранительных органов, ничего не даст. Можно получить только часть, по которой восстановить сообщение не удастся. Ограничено и время сохранности сообщения в секретных чатах. Восстановить его после удаления невозможно [9].

Преимуществами данного мессенджера, по словам разработчиков, являются:

- приватность и безопасность;
- облачная структура;
- скорость;
- всеобъемлемость;
- открытость;
- бесплатность;
- кроссплатформенность;
- мощность;
- поддержка.

Приватность и безопасность заключается в том, что письма и чаты в данном мессенджере строго зашифрованы и также имеют возможность удаления по

требованию пользователя. Облачная структура состоит в том, что для хранения информации используются удаленные серверы, поэтому доступ к сообщениям возможно получить со всех устройств. Скорость высока за счет того, что дата-центры расположены в разных городах-странах-континентах, поэтому сообщения отправляются за секунду. Открытый код Telegram и API MTProto дают возможность каждому желающему поковыряться и создать свою, улучшенную или дополненную версию. Регистрация и использование данной программой является абсолютно бесплатным, что является большим преимуществом. Также, Telegram имеет программы на большом количестве известных операционных системах. Мощность серверов мессенджера позволяет обмениваться различными сообщениями с большим кругом лиц без каких-либо помех и перебоев. Если произойдет какая-либо ошибка, связанная с мессенджером, всегда можно обратиться к технической поддержке, которая принимает отчеты на свой Twitter аккаунт [11].

Данным мессенджером очень комфортно пользоваться, именно поэтому он выбран в качестве основного приложения по взаимодействию с нашей системой.

2. ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

2.1. ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

Можно выделить следующие требования к системе в целом:

- система должна содержать следующие подсистемы: датчики мониторинга параметров, хаб для датчиков и сервер для чат-бота Telegram;
- система должна обязательно включать в себя такие датчики как: датчики задымления, датчики затопления, датчики открытых окон, датчики открытых дверей, датчики открытых ворот, датчик напряжения;

- система должна иметь возможность масштабирования и внедрения до 100 дополнительных датчиков, которые могут работать одновременно;
- каждый датчик должен быть разработан под индивидуальные особенности места, в котором он будет устанавливаться;
- система должна использовать в качестве основного интерфейса для общения с пользователем чат-бота мессенджера Telegram;
- система, в случае отклонения от заданных нормальных значений выбранных датчиков должна предупреждать об этом пользователей через Telegram чат-бота;
- система, в случае отказа датчика (случай, когда данные не приходят в систему) должна предупреждать об этом пользователей через Telegram чат-бота;
- пользователь системы по своей команде должен иметь возможность получения отчета состояния со всех датчиков через чат-бота Telegram;
- у выбранных пользователей системы должна быть возможность изменять допустимые значения для определённых датчиков через интерфейс чат-бота Telegram.

К нефункциональным требованиям можно отнести:

- датчики и хаб должны работать по беспроводной технологии передачи данных;
- «общение» между хабом и датчиками должно быть постоянным, как минимум 1 раз в минуту датчик должен сообщать свои значения хабу;
- датчики должны работать с хабом на расстоянии до 30 метров.

Требования для подсистемы «хаб»:

- хаб должен работать от сети, при этом у него должна быть возможность работы автономно в случае отключения от сети.

– время автономной работы хаба от батареи должно составлять не менее чем 16 часов после отключения от питания;
– устройство хаба должно иметь встроенный блок питания;
– хаб должен также работать и как сервер для чат-бота Telegram.

Требования для подсистемы «датчики»:

– внутри датчика должен быть встроен микроконтроллер, способный автономно обрабатывать считанные параметры в цифровые значения и передавать их в хаб по беспроводной технологии;
– датчики должны работать автономно от встроенной батареи;
– время автономной работы датчика от батареи должно составлять не менее чем 3 года;
– уличные датчики должны работать при температурах окружающей среды от -30 °C до +50 °C;
– уличные датчики должны быть защищены по стандарту IP67 (полная пыленепроницаемость и защита от погружения в воду до 1 метра).

Требования для подсистемы «сервер»:

– сервер должен работать в одном устройстве с хабом;
– сервер должен работать с интернетом по технологиям связи Wi-Fi и GSM (в случае отключения Wi-Fi);
– чат-бот должен работать одновременно как минимум с пятью пользователями и при этом не должен выходить из строя.

2.2. НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

Объем данных: объем данных ограничен трафиком 1 GB в месяц. Это лимит тарифа для мобильного интернета в подсистеме «хаб».

Временные характеристики: задержка между возникновением происшествия в доме и передачей сообщения в чат-боте пользователю должна быть не более чем 2 минуты.

Входные/выходные данные: входные данные – это состояния объектов, которые должны контролироваться системой; выходные данные – это оцифрованные значения состояния с датчиков.

Обучение: интерфейсом для работы системой является чат-бот Telegram, поэтому пользователю будет достаточно иметь на своем смартфоне или ПК установленную программу Telegram и понимание работы с чатами в ней.

3. ПРОЕКТИРОВАНИЕ

3.1. АРХИТЕКТУРА ПРЕДЛАГАЕМОГО РЕШЕНИЯ

Физически и условно вся система состоит из нескольких подсистем. Это хаб, который взаимодействует с Telegram и двумя передатчиками. На рисунке 3.1.1 представлена схема, которая описывает взаимодействия устройств друг с другом.

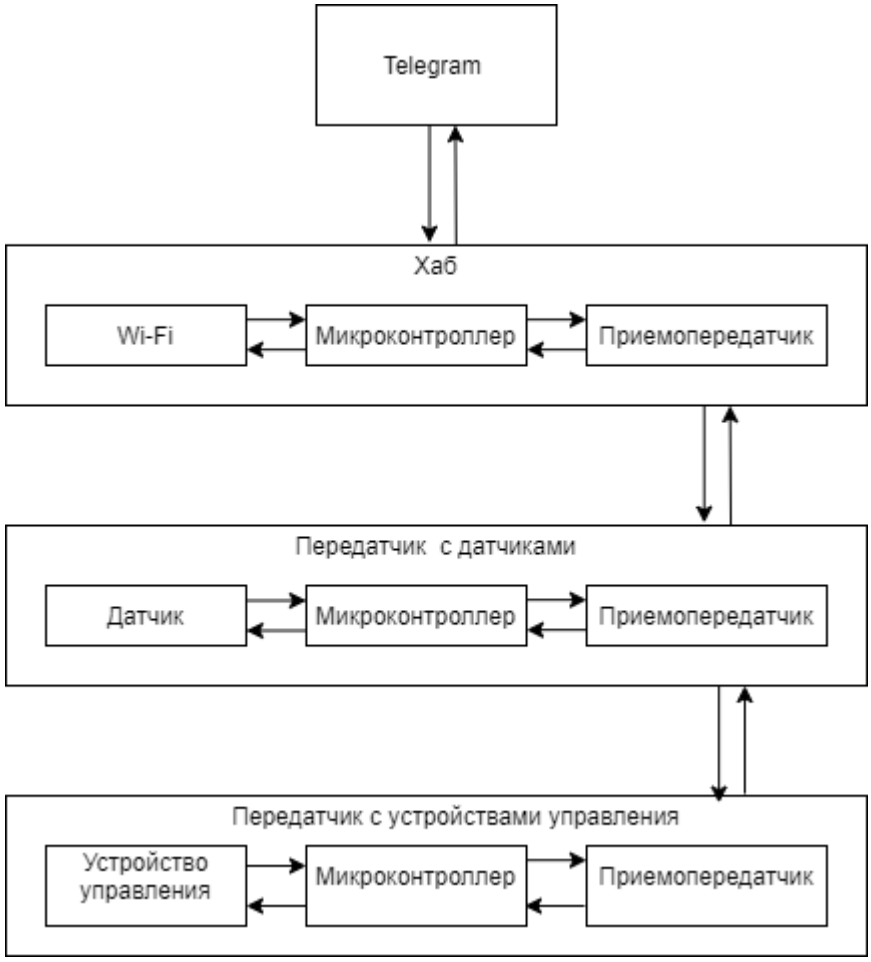

Рисунок 3.1.1 – Архитектура предполагаемого решения

3.1.1. Хаб

Хаб состоит из 3 устройств: микроконтроллер, приемопередатчик, связывающий устройства между собой и Wi-Fi модуль для передачи данных и

управления системой через мессенджер Telegram. Устройство должно снабжаться блоком питания, так как работает от сети переменного тока.

Основная функция микроконтроллера хаба – это получение и обработка данных о статусе со всех подключенных к нему устройств.

В постоянной памяти микроконтроллера хранится информация о значениях данных с датчиков. При получении команды на запрос значение с датчиков данные помещаются в его оперативную память.

Данные об изменении значений с датчиков принимаются через приемопередатчик и записываются по соответствующему адресу в оперативную память. Если запрос с Telegram не поступал, то эти данные не попадают в постоянную память хаба.

Микроконтроллер работает с Wi-Fi модулем передачи данных. Это дает возможность получить доступ к данным с помощью бота Telegram. Все эти изменения происходят в оперативной памяти хаба и сохраняются в постоянной памяти устройства.

3.1.2. Передатчик с датчиками

Передатчик, который работает с датчиками состоит из: приемопередатчика, по которому отправляются и принимаются данные, микроконтроллера для обработки данных и датчиков, с которыми он работает. Это устройство также работает от сети, поэтому требуется блок питания.

3.1.3. Передатчик с устройствами управления

Передатчик, который работает с устройствами управления состоит из: приемопередатчика, по которому отправляются и принимаются данные, микроконтроллера для обработки данных и устройств управления. Это устройство также работает от сети, поэтому требуется блок питания.

3.1.4. Telegram Bot

В мессенджере Telegram существует функция создания так называемых ботов (от англ. bot). Это специальная программа, выполняющая автоматически и/или по заданному расписанию какие-либо действия через интерфейсы, предназначенные для людей. Боты выполняют функции, которые пользователь задает вручную. Управление данной компьютерной программой может осуществляться по заранее подготовленному пользовательскому интерфейсу или же самостоятельно сделанному функционалу. Боты в мессенджере Telegram являются специальными аккаунтами, которые не требуют дополнительного телефонного номера или электронной почты для настройки. Эти учетные записи служат интерфейсом для кода, выполняемого где-то на отдельном сервере. Также у ботов нет “онлайн” статуса, вместо этого показывается отличительная надпись, что с вами общается бот.

Чтобы использовать данную функцию, не нужно ничего знать о том, как работает протокол шифрования MTProto. Промежуточный сервер Telegram будет обрабатывать все шифрование и связь с Telegram API за пользователя. Общение с сервером через простой HTTPS интерфейс, который предлагает упрощенную версию Telegram API [10].

Данные программы могут выполнять большое количество задач начиная от передачи сообщений в другие сервисы, такие как Gmail, GitHub, Wikipedia и другие до полноценной передачи денежных средств.

Работа с ботом Telegram проходит как обмен сообщениями с обычным пользователем за исключением того, что в этом случае ответы будут запрограммированы заранее. Все происходит довольно просто: запрос боту отправляется прямо из поля ввода сообщений или нажатия заранее подготовленной и появившейся на экране кнопки. Далее промежуточный сервер

Telegram обрабатывает полученный запрос и, в зависимости от функционала конкретного бота отправляет ответ.

Чтобы создать Telegram бота нужно:

– найти в поиске Telegram бота @BotFather. Это официальный бот, созданный специально для управления ботами;
– далее в личном сообщении данному боту отправить команду: /newbot;
– на этом шаге @BotFather спросит какое будет у нашего бота имя. Стоит подметить, что это имя обязательно должно заканчиваться на “_bot”;
– @BotFather поздравит с успешным созданием бота и предоставит API ключ для управления данной программой извне.

Для того, чтобы правильно использовать Telegram бота с данной системой, будет использоваться специальная библиотека Universal Telegram Bot, разработанной пользователем под никнеймом /witnessmenow [12].

Эта библиотека предоставляет интерфейс для Telegram Bot API. Её установка происходит в программе Arduino IDE, где происходит сама программная разработка. Также, для работы обязательна установка еще одной важной библиотеки: ArduinoJson, которая была предоставлена пользователем Benoît Blanchon. Установка данных библиотек проводится в разделе управления библиотеками в Arduino IDE.

Эта библиотека дает возможность пользоваться такими функциями как:

Таблица 3.1.4.1 – Функции библиотеки Universal Telegram Bot

Особенность	Описание
Receiving Messages	Бот может читать отправленные ему сообщения. Это полезно для отправки команд на Arduino.
Sending messages	Бот может отправлять сообщения в любой Telegram или группу. Можно заставить Arduino уведомлять о событии,

Продолжение таблицы 3.1.4.1

Особенность	Описание
	например. Нажатие кнопки и т. д. (Боты могут отправлять сообщения только в том случае, если сначала им отправили сообщение)
Reply Keyboards	Бот может отправлять ответные клавиатуры, которые можно использовать как тип меню.
Inline Keyboards	*InlineKeyboard* — это клавиатура, привязанная к сообщению, использующая обратный вызов (*CallbackQuery*), вместо отправки сообщения с обыкновенной клавиатуры. Бот может отправлять встроенные, привязанные к сообщению клавиатуры.
Send Photos	Это возможность отправлять фотографии от бота. Можно отправлять изображения из Интернета или напрямую из Arduino.
Chat Actions	Бот может отправлять действия в чате, такие как ввод текста или отправка фотографии, чтобы сообщить пользователю, что бот выполняет какие-либо действия.
Location	Бот может получать данные о местоположении либо из одной точки данных о местоположении, либо в реальном времени.
Channel Post	Читает сообщения с каналов.
Long Poll	Это установка времени, которое должно пройти перед проверкой получения новых сообщений. Это уменьшит количество запросов и данных, используемых ботом, но будет связывать Arduino, пока он ожидает сообщений.
Update Firmware and SPIFFS	Это возможность обновить прошивку, отправив файлы как обычный файл с определенным заголовком.

Продолжение таблицы 3.1.4.1

Особенность	Описание
Set bot's commands	Возможность программно устанавливать команды бота из кода. Команды будут отображаться в специальном месте в области ввода текста.

3.2. АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧИ

На рисунках 3.2.1 и 3.2.2 представлен простой алгоритм работы хаба.

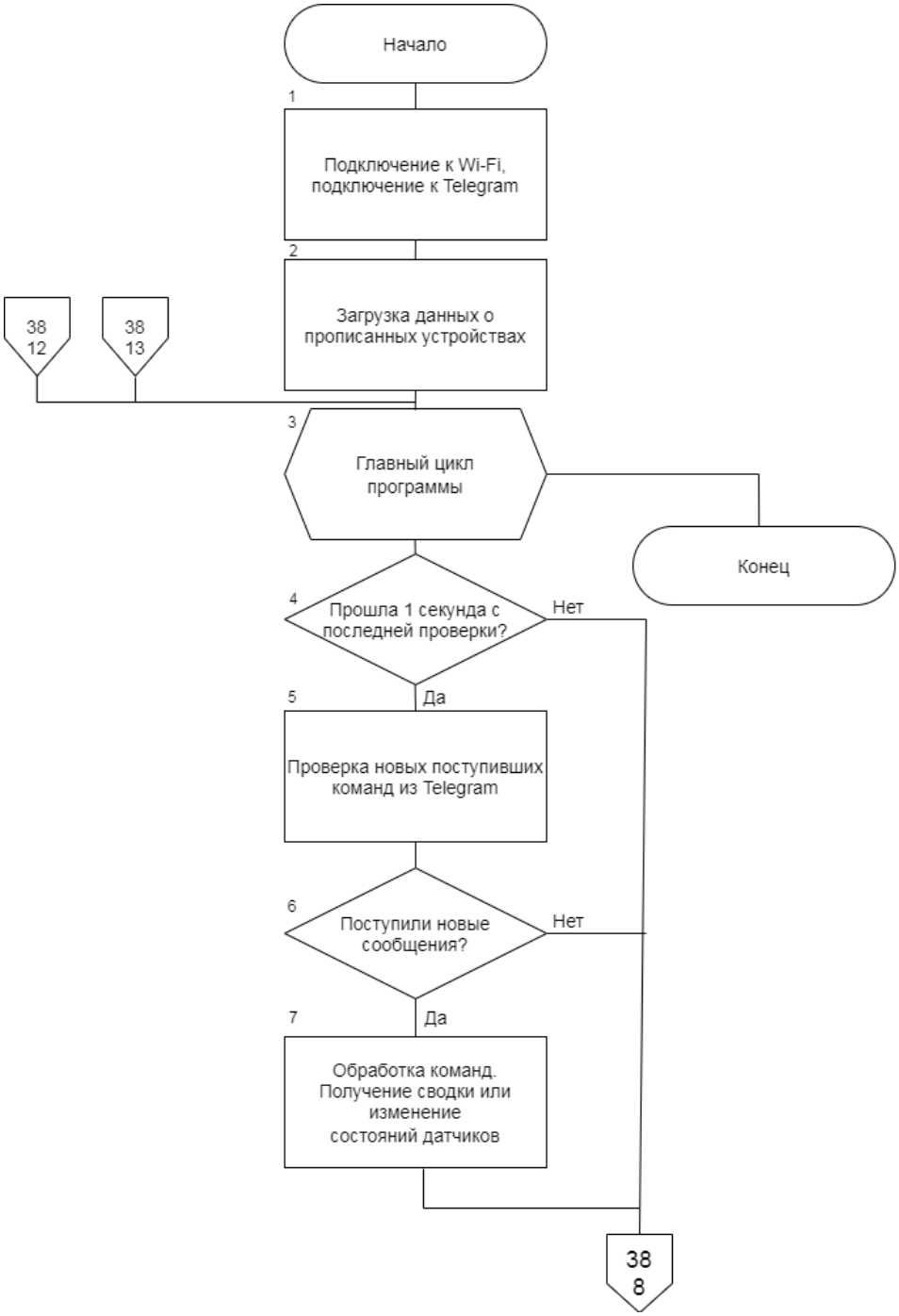

Рисунок 3.2.1 – Алгоритм работы хаба

Рисунок 3.2.2 – Продолжение алгоритма работы хаба

На рисунке 3.2.3 и 3.2.4 представлен алгоритм работы программы микроконтроллера с контроллером.

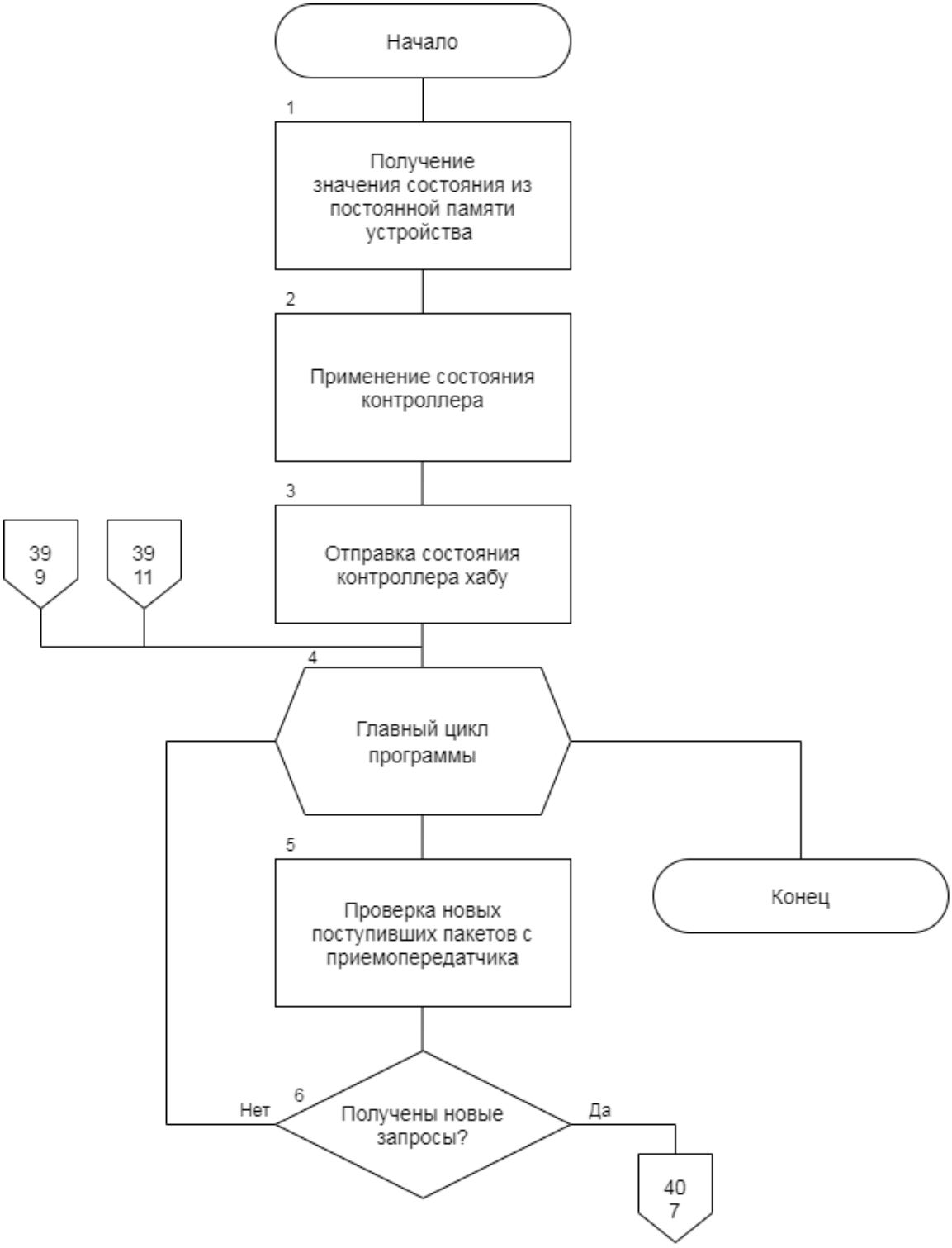

Рисунок 3.2.3 – Алгоритм работы контроллера

Рисунок 3.2.4 – Продолжение алгоритма работы контроллера

3.3. ОПИСАНИЕ ДАННЫХ

Для передачи данных между устройствами используется единый формат пакета (таблица 3.3.1). Первым байтом обязательно включается код команды. По данному байту устройство определяет, что от него требуется и затем исполняет заложенный функционал. Список некоторых команд представлен в таблице 3.3.2.

Адрес устройства указывается всегда. Это уникальный прошитый адрес датчика или контроллера, с которого или на который отправляется команда.

Тип устройства показывает, контроллер это или датчик.

4 байта данных является необязательным полем при пересылке.

Таблица 3.3.1 – Структура пакета данных

Байты	1	4	1	4
Название	Код команды	Адрес устройства	Тип устройства	Данные

В листинге 3.3.1 показан код создания пользовательского типа данных DeviceRadioData. Это структура, которая является шаблоном для передачи данных между устройствами во всей системе, она присутствует в коде каждого микроконтроллера.

Листинг 3.3.1 – Структура данных

```
typedef struct __attribute__((__packed__)) {
        uint8_t command;
        uint32_t device_address;
        uint8_t device_type;
        uint32_t data;
      } DeviceRadioData;
```

Таблица 3.3.2 – Структура пакета данных

Команда	**Описание команды**
0x01	Изменение состояние контроллера
0x02	Запрос состояния контроллера
0x21	Отправка состояния устройства
0x22	Отправка значения с датчиков

4. РЕАЛИЗАЦИЯ

4.1. СБОРКА ПРОТОТИПА

Для сборки и проверки работы данной системы нужно собрать прототип, в котором используются вышенаписанные компоненты. Все устройства и периферия была закуплена в магазине “ПРОКОНТАКТ74” [14].

4.1.1. Микроконтроллер “NodeMCU ESP8266”

Рисунок 4.1.1 – Микроконтроллер “NodeMCU ESP8266”

Таблица 4.1.1 - Технические характеристики модуля NodeMCU ESP8266

Размер	6 x 3 см
Внешнее питание	3.6–20 В
Микроконтроллер	32-битный RISC процессор Tensilica Xtensa® LX106, работающий на 80 МГц
Flash-память	4 Мб
Стоимость	275 рублей
Источник	https://procontact74.ru/14-arduino-elektronnye-moduli/222/nodemcu-lua-wifi-14107

4.1.2. Микроконтроллер «Arduino Nano»

Рисунок 4.1.2 – Микроконтроллер “Arduino Nano”

Таблица 4.1.2 - Технические характеристики модуля Arduino Nano

Размер	1.85 см x 4.2 см
Внешнее питание	7-12 В
Микроконтроллер	Atmel ATmega168
Flash-память	16 Кб
Стоимость	377 рублей
Источник	https://procontact74.ru/14-arduino-elektronnye-moduli/arduino/arduino-nano-v3-atmega168p-ch340-klon-piny-pripayany-14261

4.1.3. Датчик температуры и влажности «DHT11»

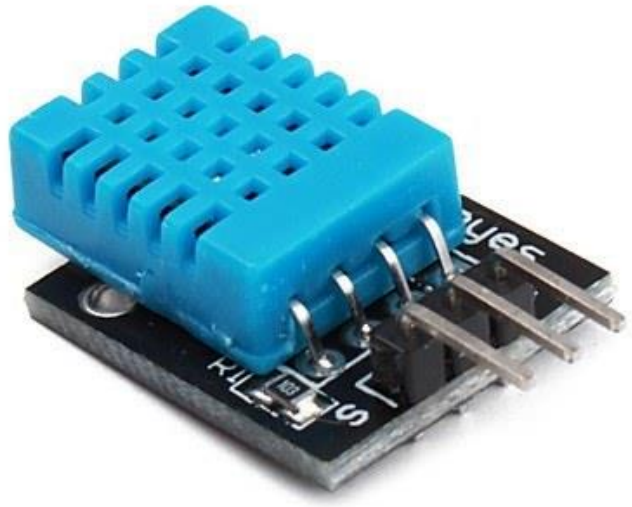

Рисунок 4.1.3 – Датчик DHT11

Таблица 4.1.3 - Технические характеристики датчика DHT11.

Размер	15.5 мм x 12 мм x 5.5 мм
Питание	3,5 – 5,5 В
Измерение влажности	В диапазоне от 20% до 80%
Измерение температуры	0°C до 50°C с точностью ± 2%
Стоимость	100 рублей
Источник	https://procontact74.ru/01-elektronnyie-komponentyi-/34-datchiki/datchik-temperatury-i-vlazhnosti-dht11-32538

4.1.4. Датчик влажности почвы с компаратором на LM393

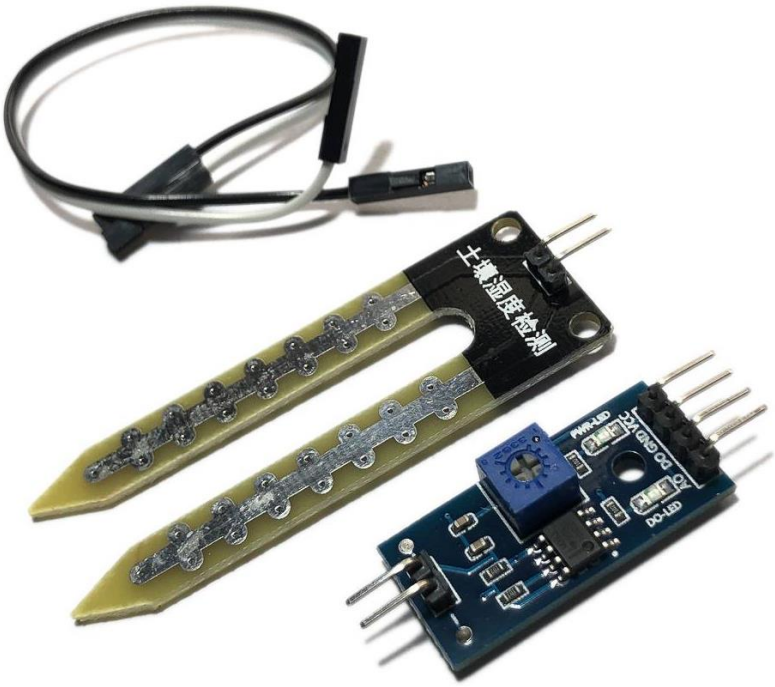

Рисунок 4.1.4 – Датчик влажности почвы

Таблица 4.1.4 - Технические характеристики датчика влажности почвы.

Питание	3,5 – 5,5 В
Глубина	43 мм
Толщина	7 мм
Ширина	14 мм
Стоимость	71 рублей
Источник	https://procontact74.ru/14-arduino-elektronnye-moduli/moduli-datchikov1/datchik-vlazhnosti-pochvy-vily-s-komparatorom-na-lm393-14254

4.1.5. Датчик уровня жидкости

Рисунок 4.1.5 – Датчик уровня жидкости

Таблица 4.1.5 – Технические характеристики датчика уровня жидкости

Размер	62 мм x 20 мм x 8 мм
Питание	3 - 5 В
Измерение влажности	10% - 90% без конденсации
Рабочая температура	10°C до 30°C
Стоимость	36 рублей
Источник	https://procontact74.ru/14-arduino-elektronnye-moduli/moduli-datchikov1/vysokochuvstvitelnyj-datchik-vody-14068

4.1.6. Радио модуль NRF24L01

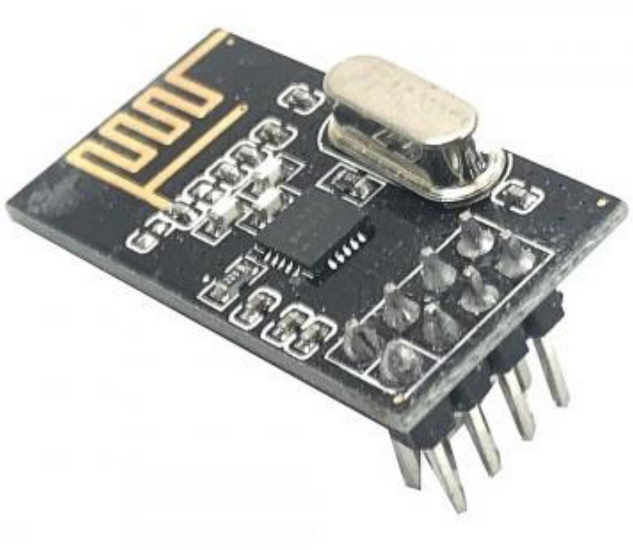

Рисунок 4.1.6 - Радио модуль NRF24L01

Таблица 4.1.6 - Технические характеристики радио модуля NRF24L01

Размер	29 мм x 15 мм
Питание	1,9 - 3.6 В
Скорость передачи данных	до 2 Мбит/с
Частота передачи/приема	2.4 ГГц
Расстояние передачи/приема	до 100 м, в помещении - до 30 м
Стоимость	132 рублей

Источник	https://procontact74.ru/14-arduino-elektronnye-moduli/moduli-du%2C-rfid/nrf24l01-32507

4.1.7. Сборка макета

Для экономии средств всего используется две отладочные платы Arduino. Таким образом, можно разделить всю систему на 3 части. На основе первого микроконтроллера работают все выбранные датчики. Вторая плата объединяет все контроллеры. Третья отладочная плата – NodeMCU. Её основное предназначение получение данных с датчиков и управление контроллерами, но также она является модулем, который работает с приложением Telegram. В каждом случае к микроконтроллеру подключается приемопередатчик NRF24L01+ по интерфейсу SPI.

Электрические схемы моделей датчиков, контроллеров и хаба представлены на рисунках 4.1.7.1 – 4.1.7.3.

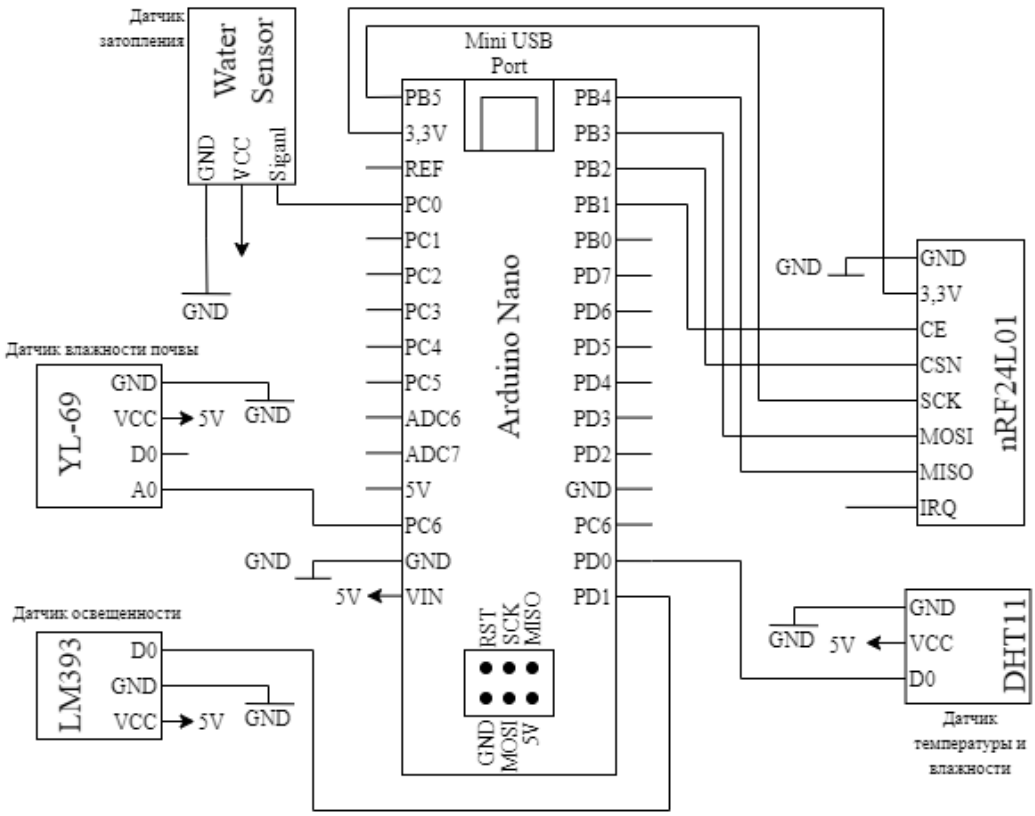

Рисунок 4.1.7.1 – Электрическая схема системы датчиков

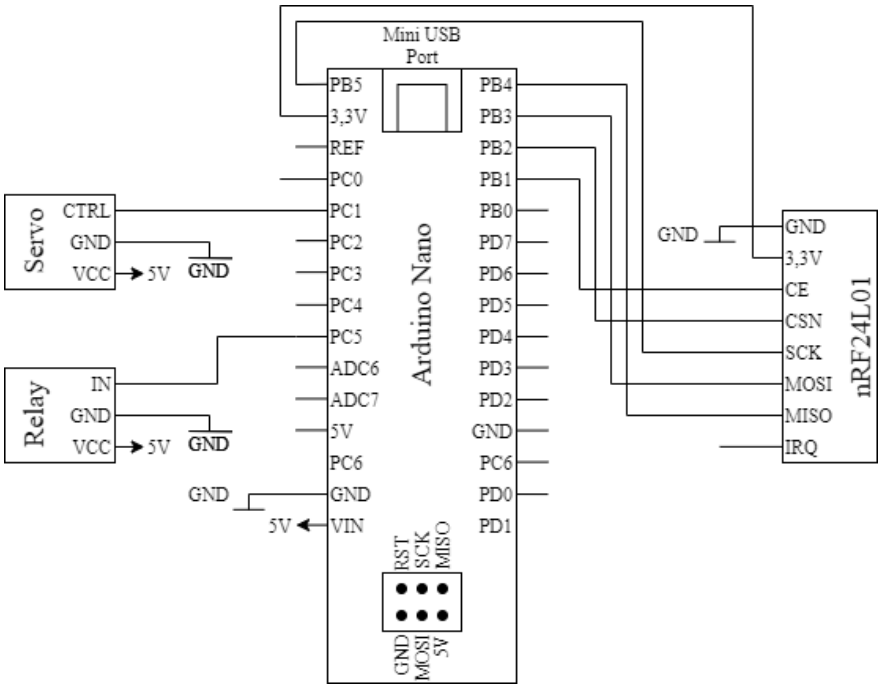

Рисунок 4.1.7.2 – Электрическая схема системы контроллеров

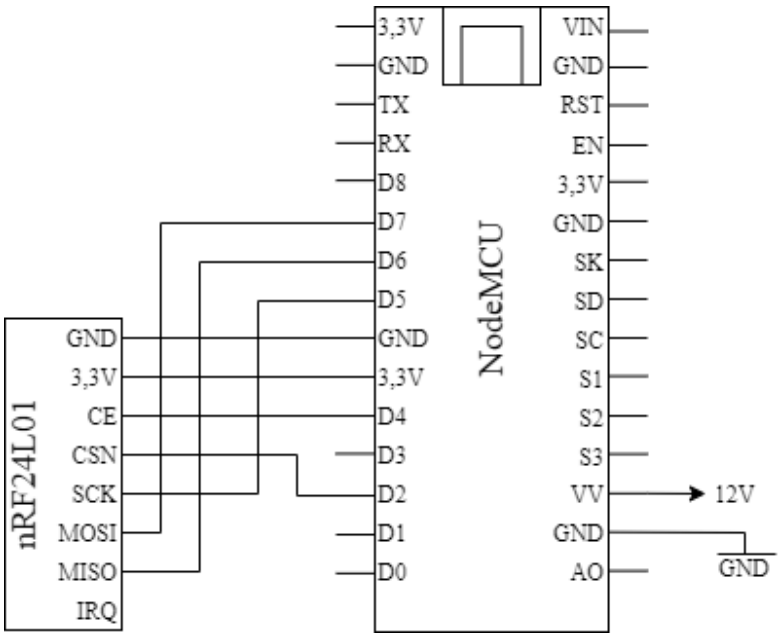

Рисунок 4.1.7.3 – Электрическая схема хаба

Фотографии собранных моделей датчиков, контроллеров и хаба представлены на рисунках 4.1.7.4 – 4.1.7.6.

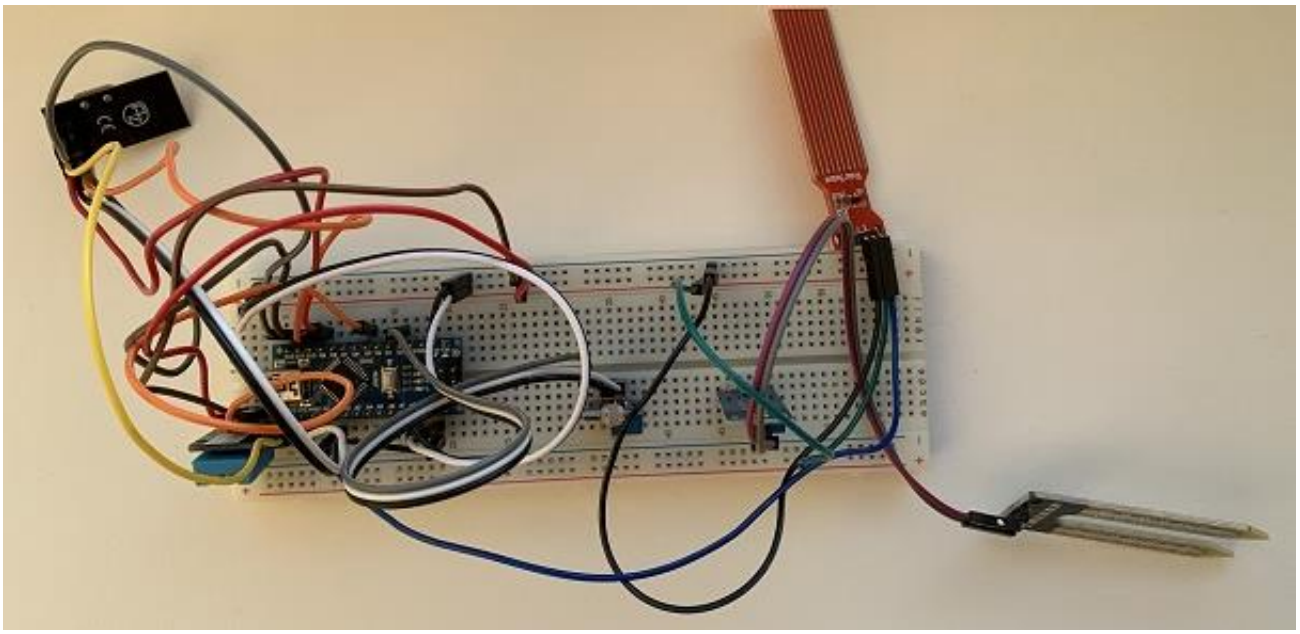

Рисунок 4.1.7.4 – Фотография собранного макета с датчиками

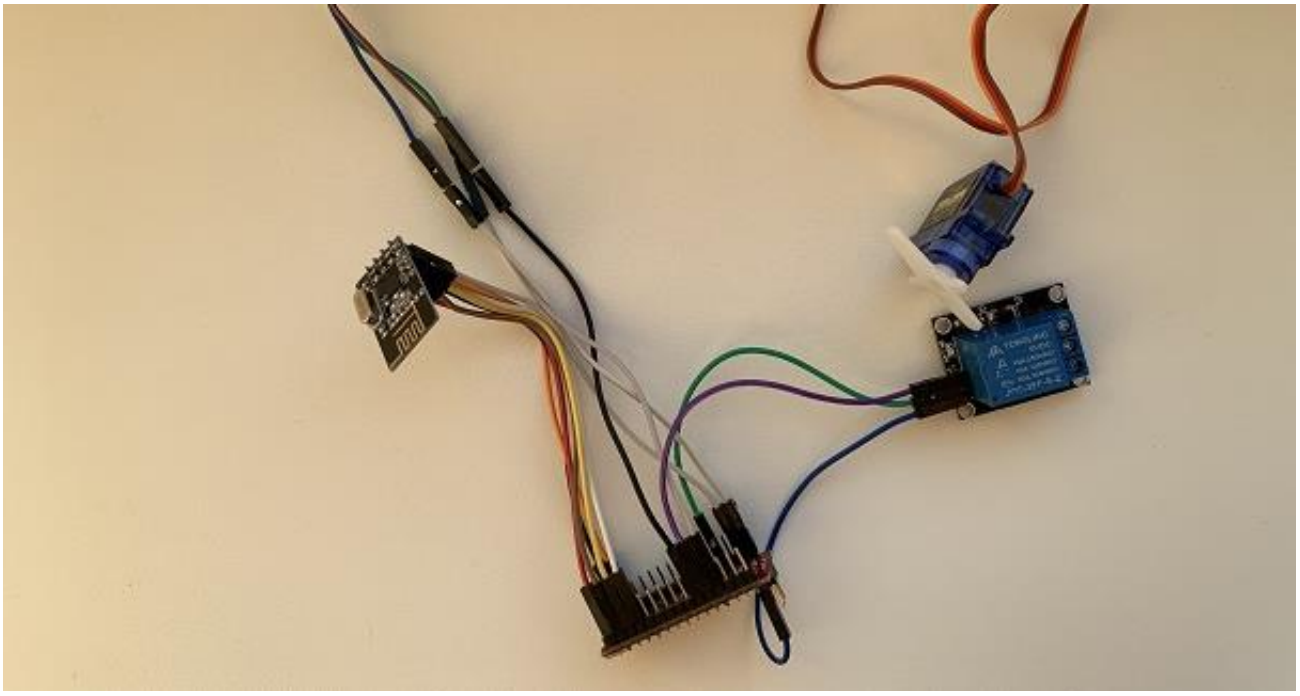

Рисунок 4.1.7.5 – Фотография собранного макета с контроллерами

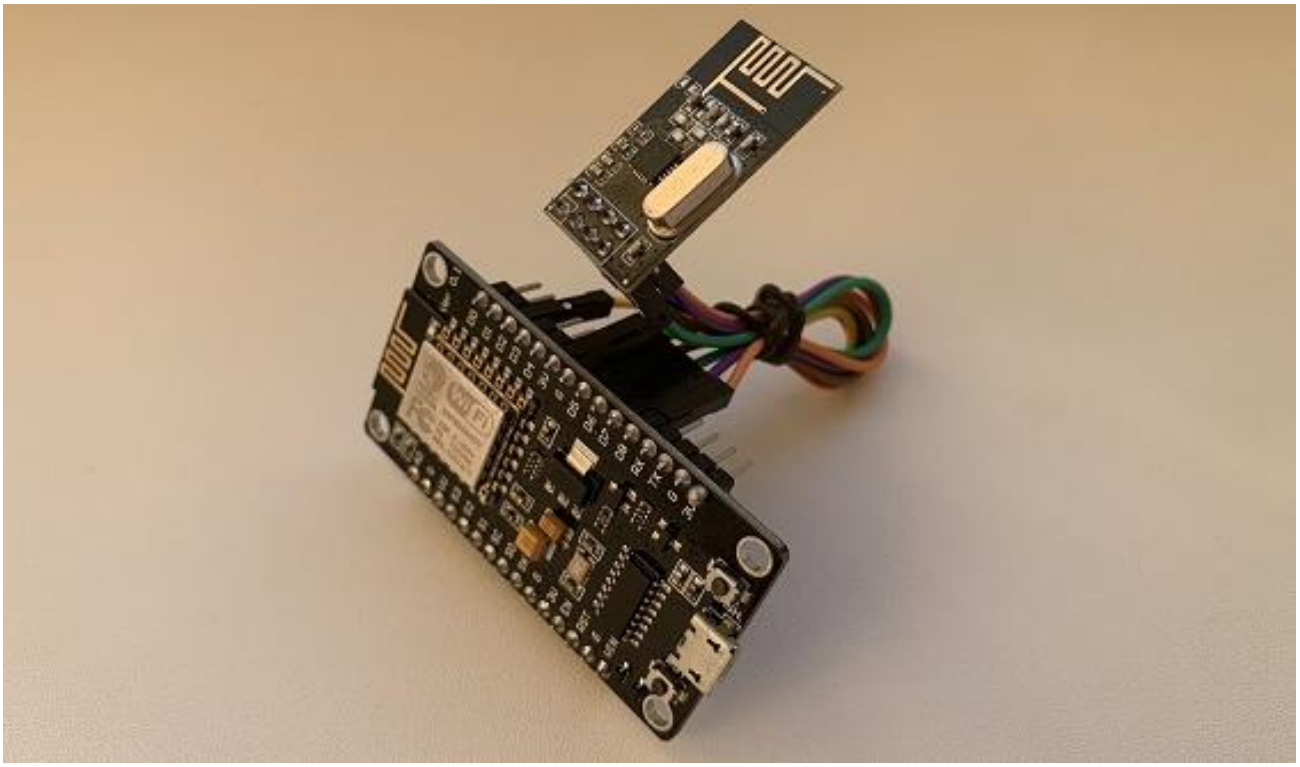

Рисунок 4.1.7.6 – Фотография платы NodeMCU и приёмопередатчика

4.2. ПРОГРАММИРОВАНИЕ УСТРОЙСТВ

При написании программного кода для устройств Arduino используется программа Arduino IDE (рисунок 4.2.6). Arduino IDE — это программная среда разработки, использующая C++ и предназначенная для программирования всех плат ряда Ардуино. IDE (от англ. Integrated Development Enviroment) – это интегрированная среда разработки с помощью которой программисты могут быстро и удобно писать программный код. Arduino IDE позволяет составлять программы удобном текстовом редакторе, компилировать их в машинный код, и загружать на все версии платы Arduino [18].

Для того, чтобы загрузить готовый программный код на имеющееся устройство, нужно:

- запустить готовый файл в формате “**.ino” в программе Arduino IDE (рисунок 4.2.1);
- в разделе “Инструменты” – “Плата” убедиться, что установлена подходящая для нашей платы настройка. В данном случае при программировании NodeMCU ESP8266 должна стоять настройка “Generic ESP8266 Module” (рисунок 4.2.2);
- далее необходимо убедиться, что устройство корректно подключено и в настройках программы отображается порт, к которому оно подключено (рисунок 4.2.3);
- далее в разделе “Скетч” нужно нажать кнопку “Проверить/Компилировать”. Это необходимо для того, чтобы проверить программу на наличие ошибок (рисунок 4.2.4);
- так как для прошивки используемого микроконтроллера нет необходимости использовать программатор, поэтому после успешной проверки программы в разделе “Скетч” необходимо только нажать кнопку “Загрузка” (рисунок 4.2.5);

– после успешной загрузки программного кода на устройство оно автоматически перезагрузится и запустит загруженную на него прошивку.

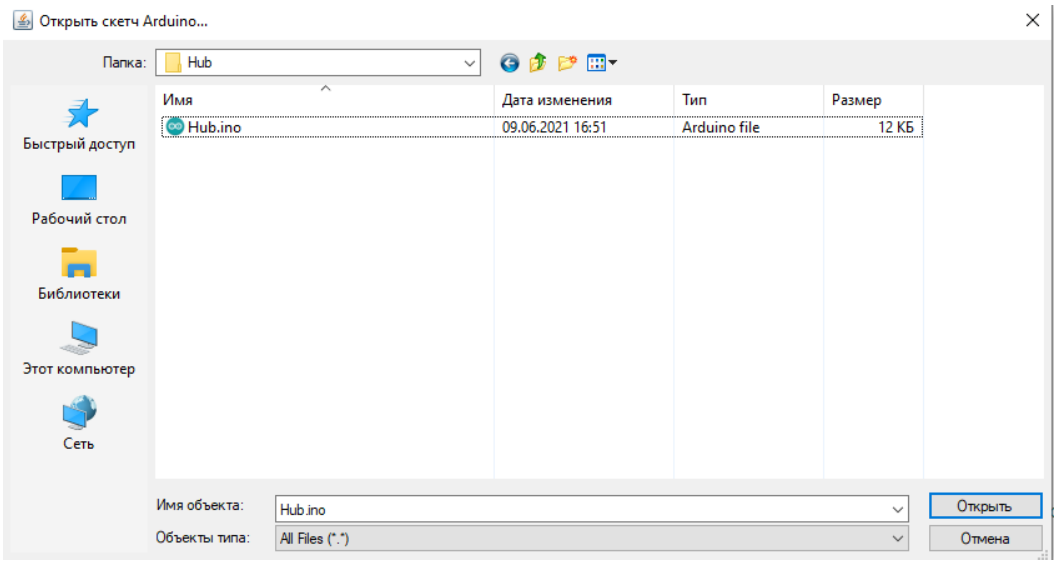

Рисунок 4.2.1 – Выбор готового скетча

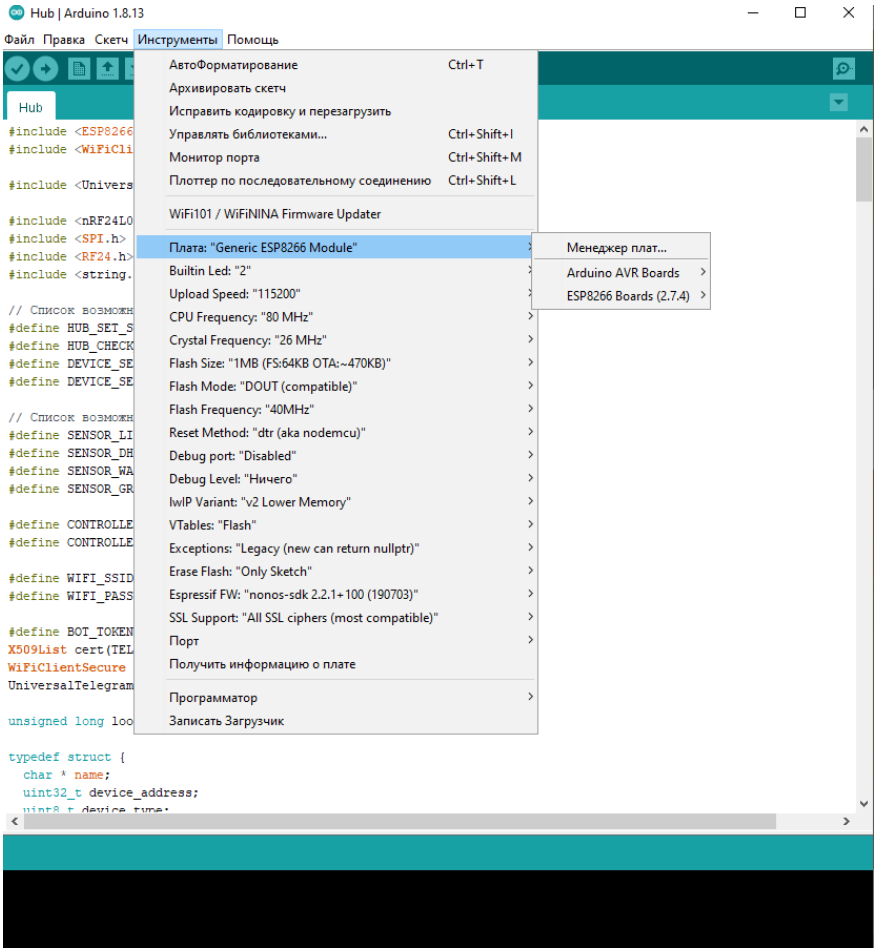

Рисунок 4.2.2 – Выбор нужной платы

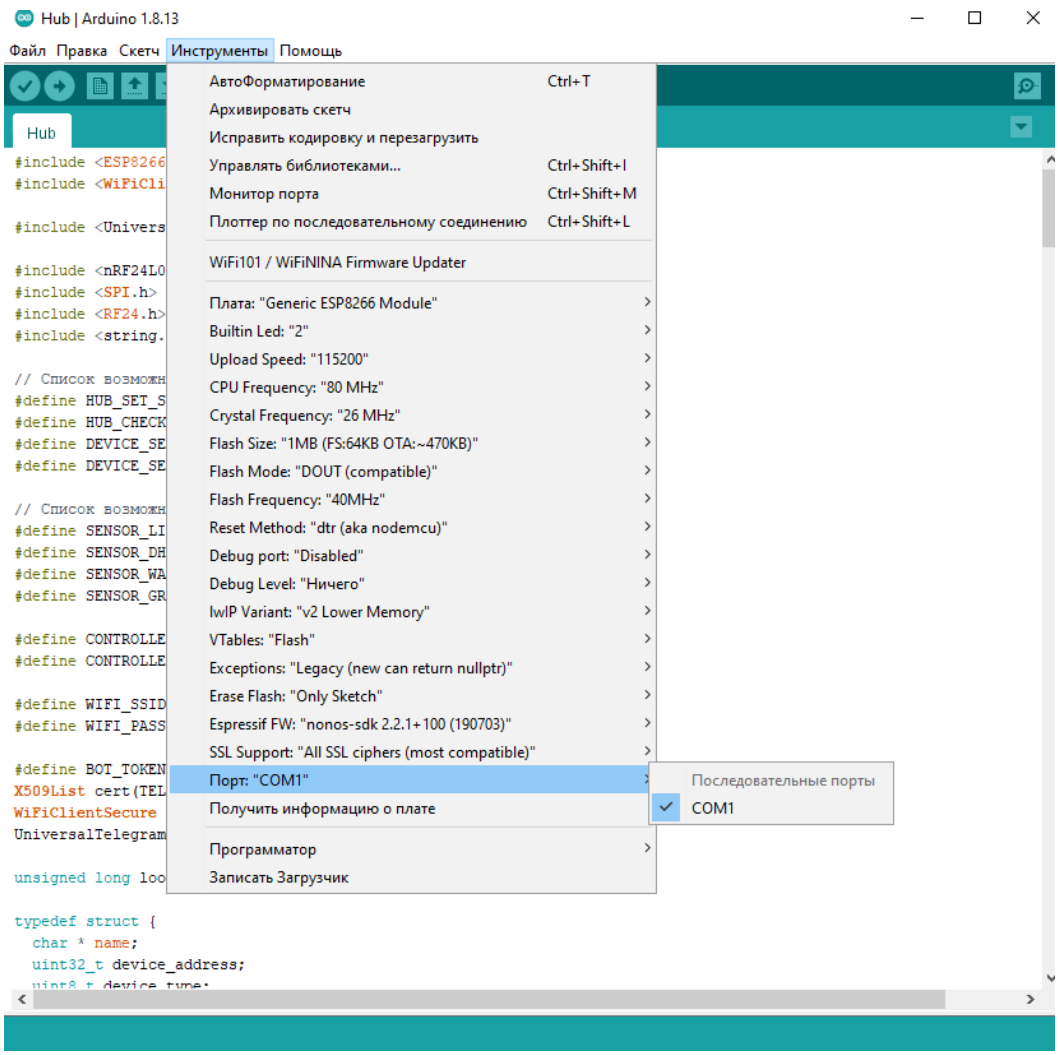

Рисунок 4.2.3 – Выбор последовательного порта

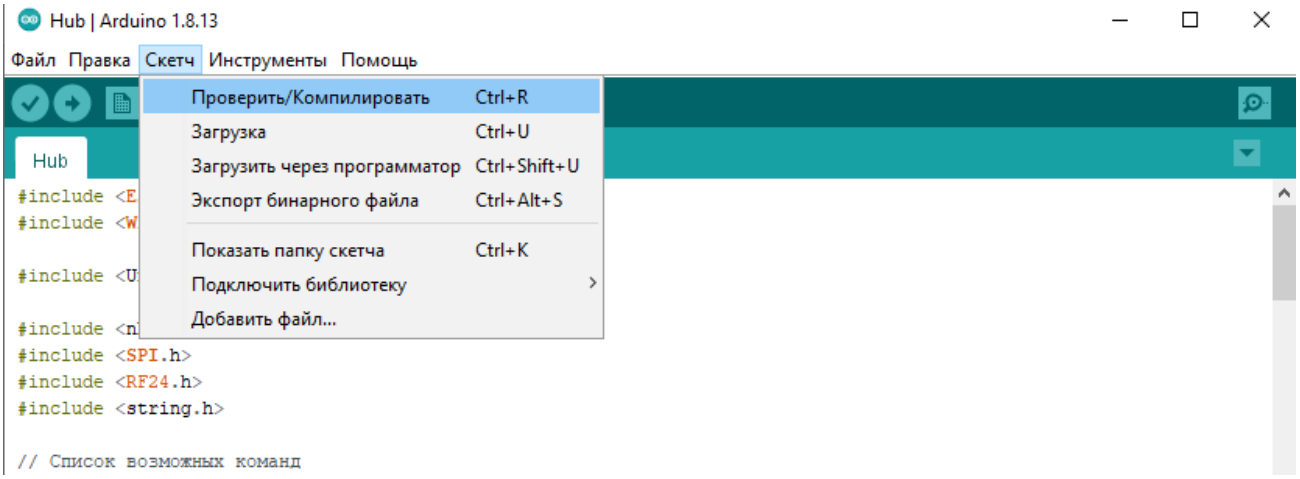

Рисунок 4.2.4 – Проверка программного кода на ошибки

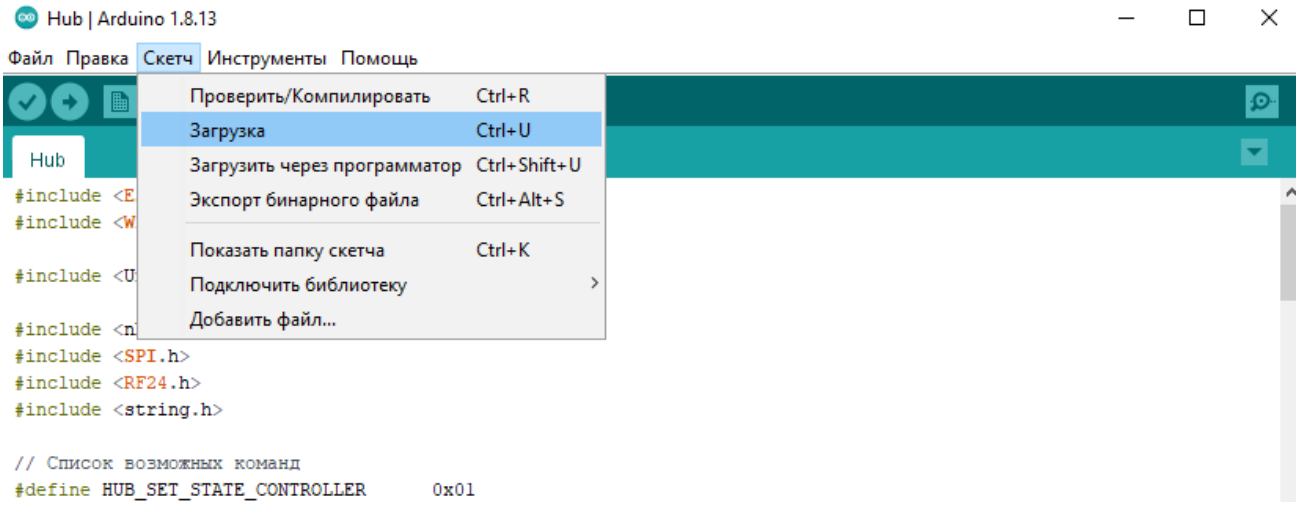

Рисунок 4.2.5 – Загрузка прошивки на устройство

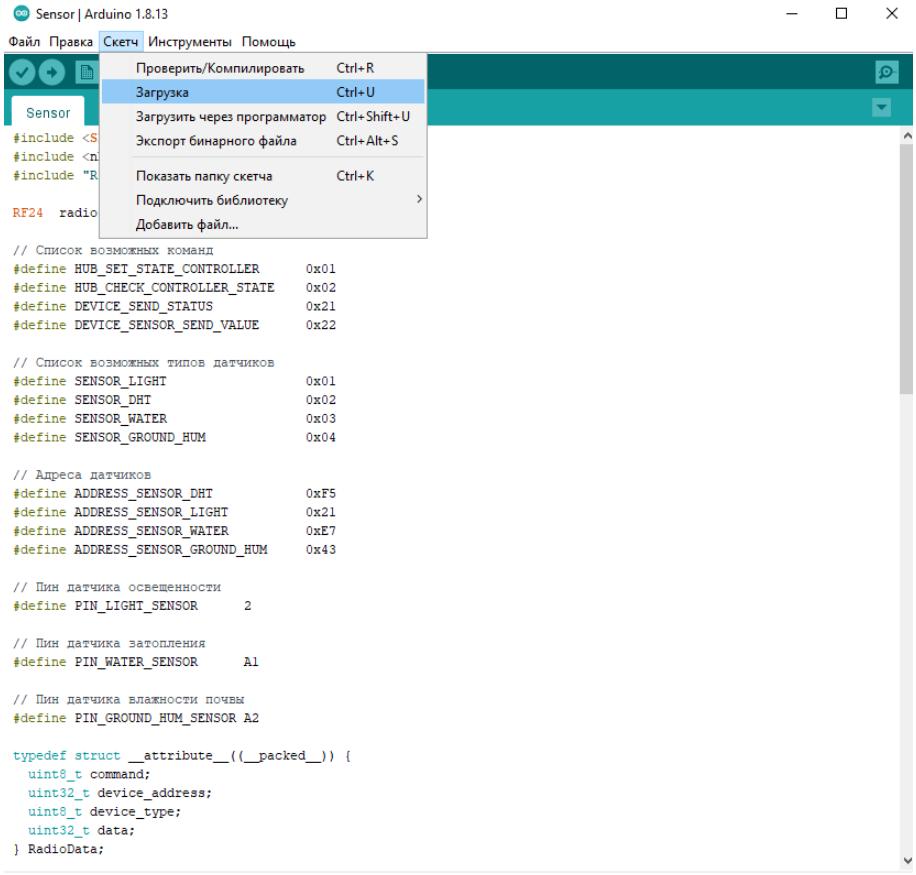

Рисунок 4.2.6 – Программа Arduino IDE

В процессе написания программного кода использовались сторонние библиотеки “nrf24l01” и “spi”, для работы приемопередатчика, “UniversalTelegramBot” для работы микроконтроллера с ботом Telegram.

Исходный код хаба представлен в листинге А.1. приложения А. Исходный код передатчика с датчиками представлен в листинге Б.1. приложения Б. Исходный код передатчика с устройствами управления представлен в листинге В.1. приложения В.

5. ТЕСТИРОВАНИЕ

5.1. МЕТОДОЛОГИЯ ТЕСТИРОВАНИЯ

Так как данная система является программно-аппаратным комплексом, тестирование требует большого количество внимания к обеим частям системы. Было проведено альфа-тестирование разработанного функционала системы.

Альфа-тестирование – это один из видов тестирования, которое обычно проводится на поздней стадии разработки продукта и включает имитацию реального использования продукта. Альфа-тестирование дает возможность проверить работоспособность системы на соответствие требованиям. Данный вид тестирования дает возможность посмотреть в реальных условиях на результат работы и также возможность выявления критических неполадок для дальнейшего исправления.

5.2. ПРОВЕДЕНИЕ ПРОЦЕДУРЫ ТЕСТИРОВАНИЯ

Для начала происходит подключение устройства к сети и проверка их работоспособности. Для этого открывается “Монитор порта” в Arduino IDE и отображается, что отправляет устройство.

Листинг 5.2.1 – Определение данных Wi-Fi сети и бот-токена Telegram

```
#define WIFI_SSID "***"
#define WIFI_PASSWORD "***"

#define BOT_TOKEN "**************************"
```

Листинг 5.2.2 – Открытие канала связи по NRF24L01

```
void setup(void) {
  SPI.setHwCs(true);
  SPI.begin();
  SPI.setDataMode(SPI_MODE0);
  SPI.setBitOrder(MSBFIRST);
// Инициализация модуля NRF24L01
  radio.begin();
// Обмен данными ведется на пятом канале (2,405 ГГц)
  radio.setChannel(5);
// Скорость обмена данными 1 Мбит/сек
  radio.setDataRate(RF24_1MBPS);
// Выбор высокой мощности передатчика (-6dBm)
  radio.setPALevel(RF24_PA_HIGH);
  // Открытие канала получения значений с датчиков
  radio.openReadingPipe(1, 0x00000011LL);
  // Открытие канала получения состояния контроллеров
  radio.openReadingPipe(2, 0x000000BBLL);
  radio.openWritingPipe(0x000000CCLL);
// Начало прослушивания открывамой трубы
  radio.startListening();
  Serial.begin(115200);

configTime(0, 0, "pool.ntp.org");      // Получение времени с сервера
  secured_client.setTrustAnchors(&cert);  //  Добавление  корневого
сертификата для api.telegram.org
  Serial.print("Connecting to Wifi SSID ");
  Serial.print(WIFI_SSID);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD); // Подключение к Wi-Fi сети
по указанным выше данным
  while (WiFi.status() != WL_CONNECTED)
  {
    Serial.print(".");
    delay(500);
  }
  Serial.print("\nWiFi connected. IP address: "); // При подключении
к Wi-fi сети отправляется сообщение с IP адресом
  Serial.println(WiFi.localIP());
```

В листинге 5.2.1 показано какие значения заданы по умолчанию для успешного подключения к Wi-Fi сети и Telegram боту. В листинге 5.2.2 показана инициализация подключения к устройствам по NRF24L01, установка параметров и открытие соединения, и подключению к Wi-Fi сети. В листинге 5.2.3 показано

подключения передатчика с управляемыми устройствами к каналам по беспроводной радиосвязи. В листинге 5.2.4 показано подключение передатчика с датчиками по радиосвязи к хабу для передачи данных с датчиков.

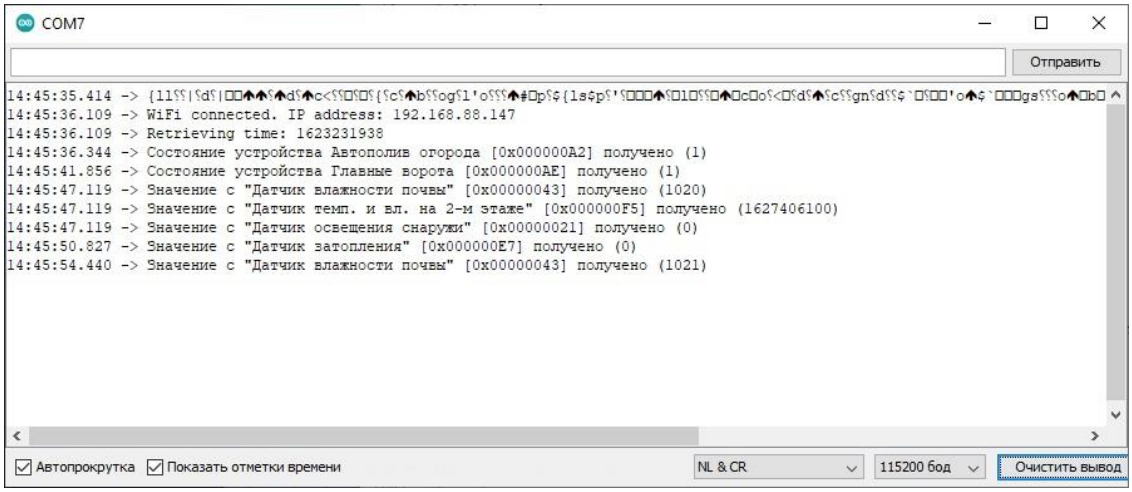

Рисунок 5.2.1 – Результат включения хаба

После подключения хаба к Wi-Fi сети устройство определяет подключенные к нему датчики и управляемые устройства.

Листинг 5.2.3 – подключение передатчика с управляемыми устройствами по NRF24l01

```
radio.begin(); // Инициализация модуля NRF24L01
    radio.setChannel(5); // Обмен данными ведется на пятом канале (2,405 ГГц)
    radio.setDataRate(RF24_1MBPS); // Скорость обмена данными 1 Мбит/сек
    radio.setPALevel(RF24_PA_HIGH); // Выбор высокой мощности передатчика (-6dBm)
    radio.openReadingPipe(1, 0x000000CCLL); // Чтение команд, поступивших с хаба посредством Telegram
    radio.openWritingPipe(0x000000BBLL); // Открытие канала отправки данных с управляемых устройств
    radio.startListening(); // Начало прослушивание открываемой трубы
```

Листинг 5.2.4 – подключение передатчика с датчиками по NRF24l01

```
    radio.begin(); // Инициализация модуля NRF24L01
        radio.setChannel(5); // Обмен данными ведется на пятом канале
(2,405 ГГц)
        radio.setDataRate(RF24_1MBPS); // Скорость обмена данными 1
Мбит/сек

        radio.setPALevel(RF24_PA_HIGH); // Выбор высокой мощности
передатчика (-6dBm)

        radio.openWritingPipe(0x00000011LL);
```

Отличия инициализации каналов связи на разных устройствах состоят в том, что хаб должен считывать данные со всех устройств, передатчик с управляемыми устройствами должен также считывать команды, поступившие с хаба, а передатчик с датчиками лишь отправляет данные, которые были считаны напрямую с датчиков.

Далее попробуем отправить некоторые команды и посмотреть результат.

Листинг 5.2.5 – Попытка отправки команд и получения результатов

```
    // Функции Telegram
    void handleNewMessages(int numNewMessages)
    {
      Serial.println("Получены новые команды из Telegram");

      for (int i = 0; i < numNewMessages; i++)
      {
        String chat_id = bot.messages[i].chat_id;
        String text = bot.messages[i].text;

        Serial.println(bot.messages[i].text);

        char str[128];
        sprintf(str, "[[\"Сводка\"],[\"%s ворота\", \"%s полив\"]]",
                (devicesData[3].actual_data == 0x00 ? "Открыть" :
"Закрыть"),
                (devicesData[2].actual_data == 0x00 ? "Включить" :
"Отключить"));
        String keyboardJson = String(str);

    // Команда отправляет доступные для использования команды
        if (text == "/start")
        {
```

```
            bot.sendMessageWithReplyKeyboard(chat_id, "Доступные команды",
"", keyboardJson, true, true);
          }
          if (text == "/devices")
          {
            bot.sendMessage(chat_id, getListDevices(), "Markdown");
          }
      // Команда отправляет состояние датчиков и контроллеров
          if (text == "/status" || text == "Сводка")
          {
            bot.sendMessageWithReplyKeyboard(chat_id, getListValues(), "",
keyboardJson, true, true);
          }
      // Команда отправляет задачу на открытие ворот
          if (text == "/gatesclose" || text == "Открыть ворота")
          {
            setState(0xAE, 1);
          }
      // Команда отправляет задачу на закрытие ворот
          if (text == "/gatesopen" || text == "Закрыть ворота")
          {
            setState(0xAE, 0);
          }
      // Команда отправляет задачу на включение реле на автополив
          if (text == "/relayon" || text == "Включить полив")
          {
            setState(0xA2, 1);
          }
      // Команда отправляет задачу на отключение реле на автополив
          if (text == "/relayoff" || text == "Отключить полив")
          {
            setState(0xA2, 0);
          }
        }
      }
```

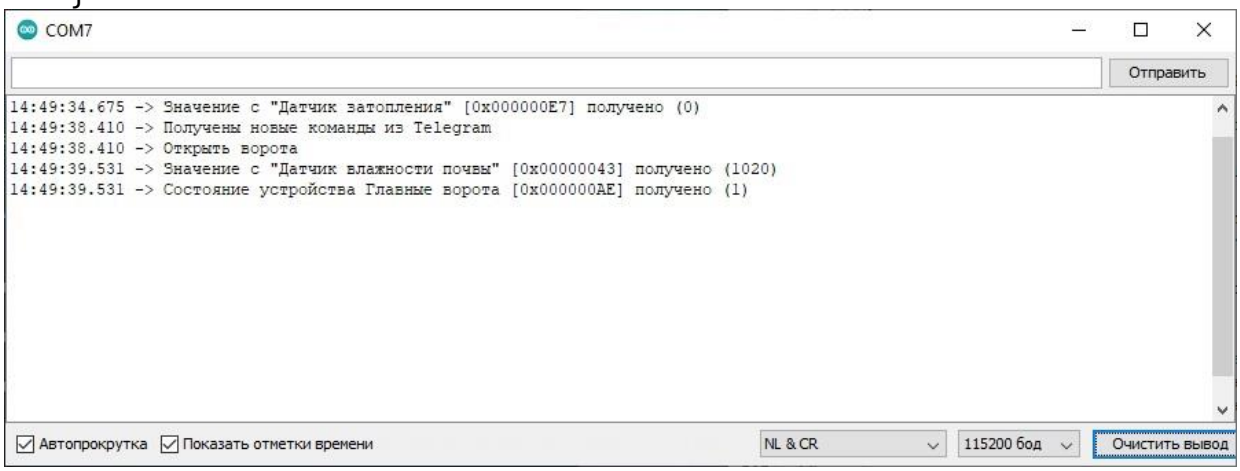

Рисунок 5.2.2 – Результат выполнения команд

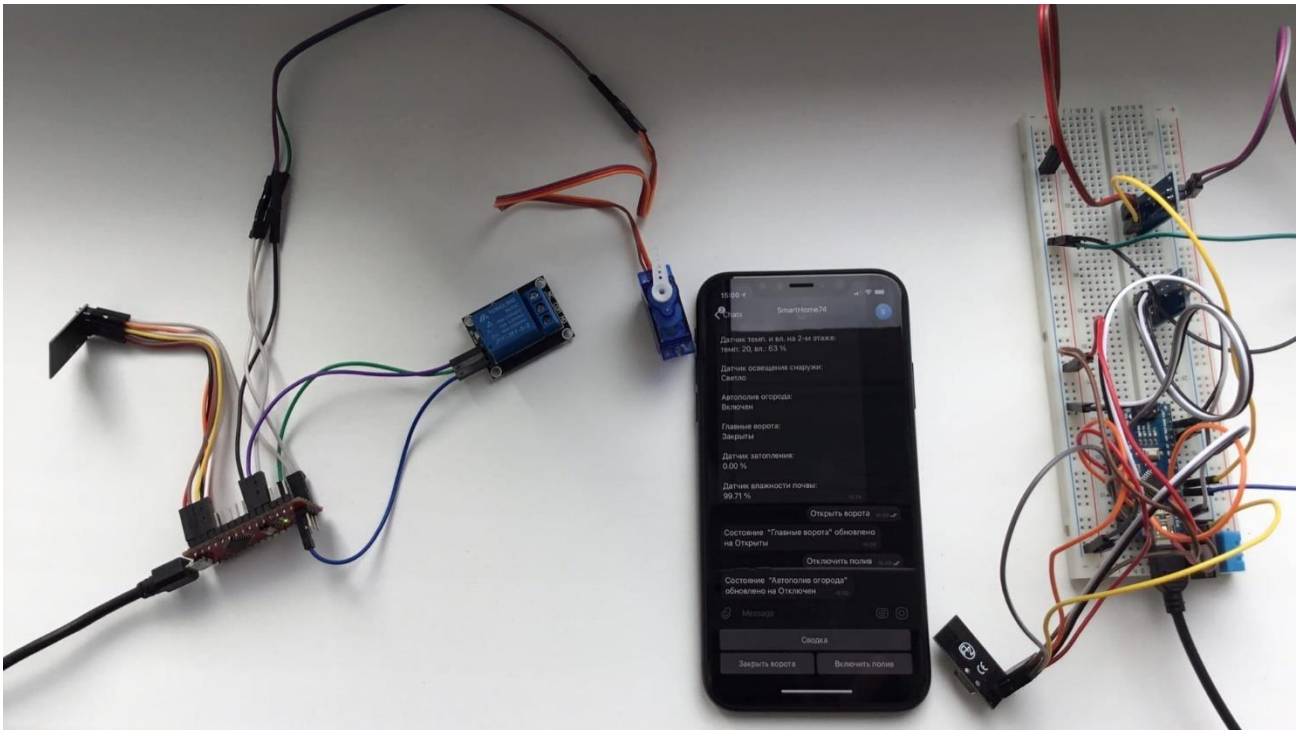

Рисунок 5.2.3 – Внешний вид подключенной системы при отправке команды

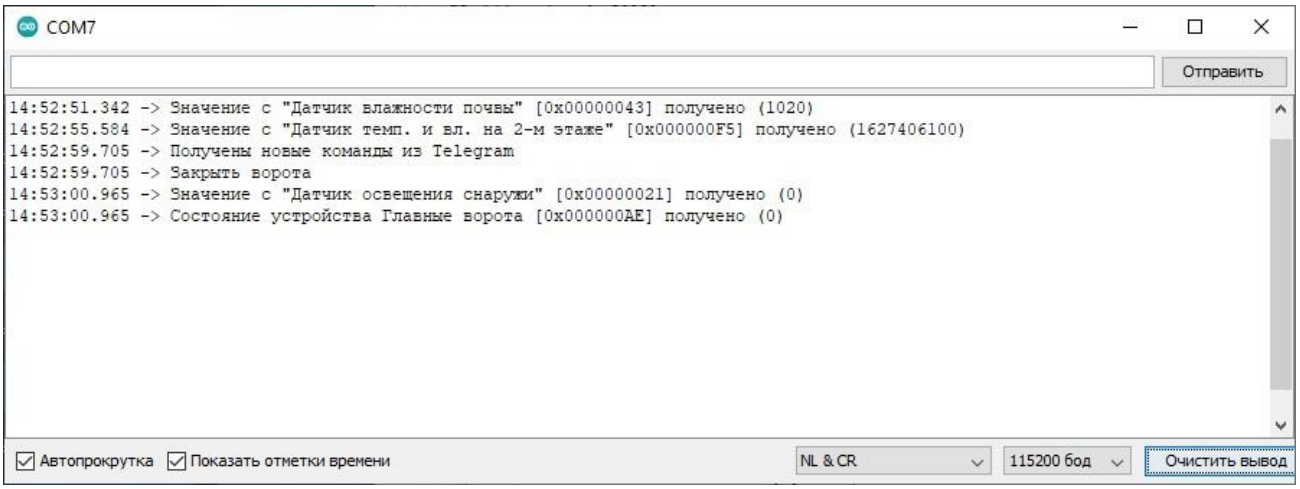

Рисунок 5.2.4 – Результат выполнения команд

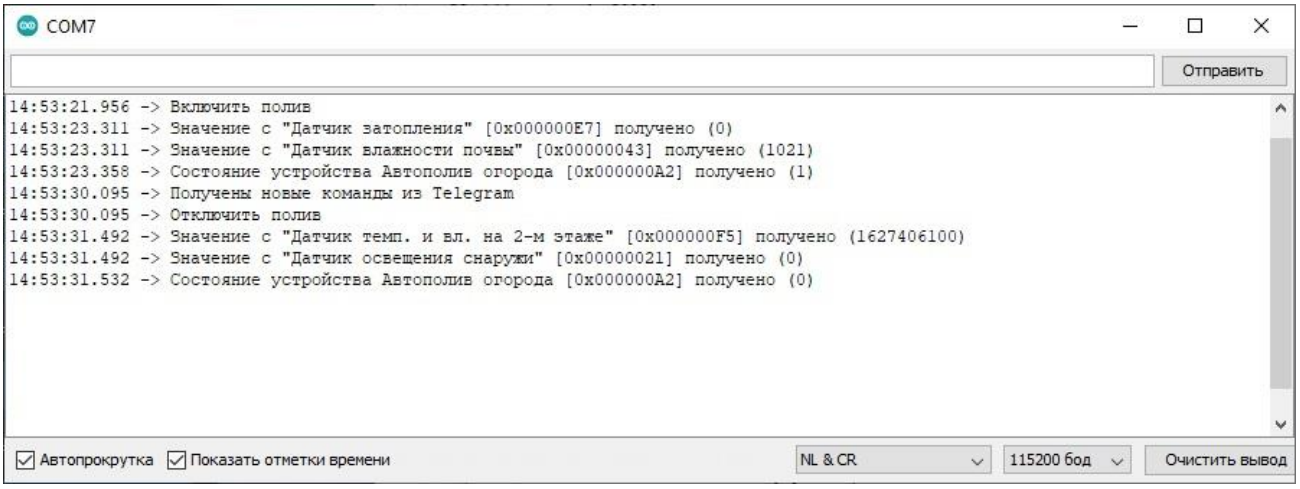

Рисунок 5.2.5 – Результат выполнения команд

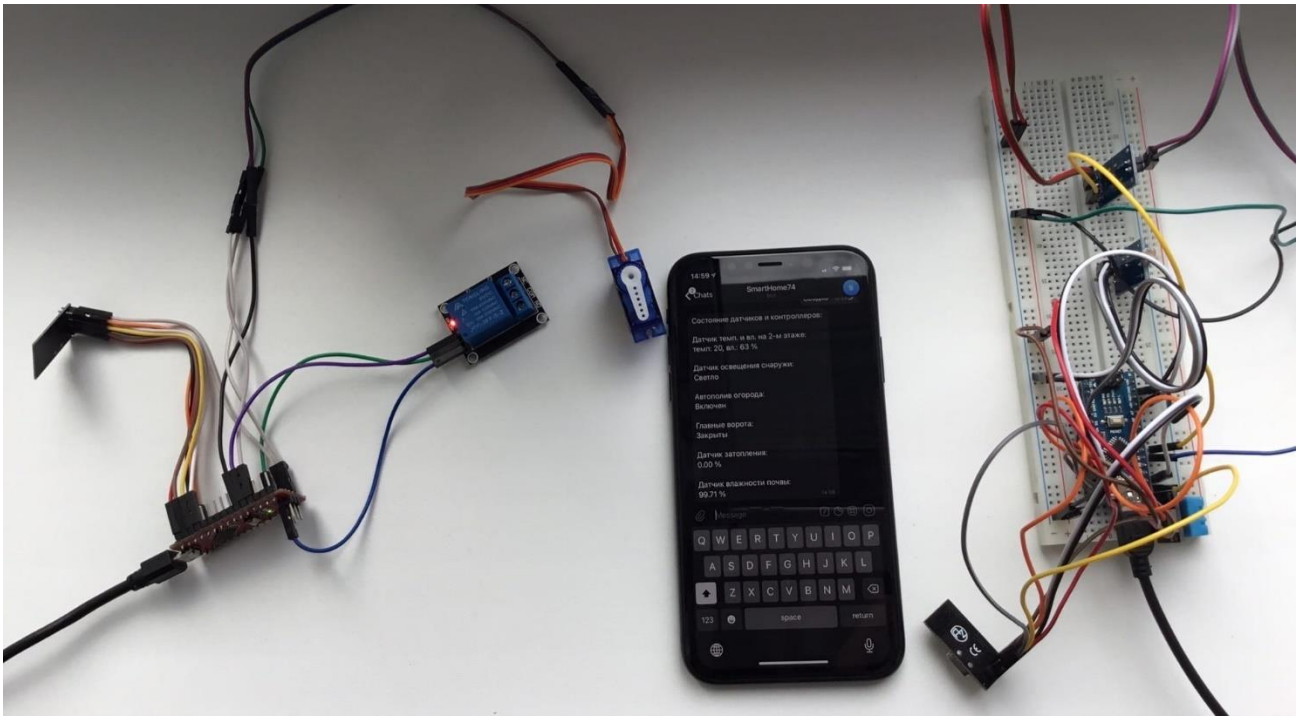

Рисунок 5.2.6 – Внешний вид системы после выполнения команд

Как можно заметить, тестирование основных функций данной системы выполняются успешно, обмен данными между устройствами и прием/отправка команд и данных проходят без ошибок, значит тестирование системы проведено успешно.

5. ЗАКЛЮЧЕНИЕ

В данной работе были проработаны все этапы, которые связаны с проектированием, разработкой, тестированием системы.

В результате обзора аналогичных решений было замечено, что многие из них имеют существенный недостаток в виде высокой стоимости оборудования, а также некоторые из рассмотренных систем не могут быть установлены без специализированной помощи.

Также было проведено сравнение нескольких языков программирования для разработки ПО для данной системы. Были приведены преимущества и недостатки рассмотренных языков программирования.

Было проведено сравнение некоторых типов беспроводной связи по разными критериям и выявление самого подходящего из них для использования в данной системе.

Был разработан и введен в эксплуатацию программно-аппаратный комплекс, который дает возможность получать значения с датчиков и возможность управления устройством открытия ворот и авто полива растений.

Производилось альфа-тестирование системы. Было показано, что данный программно-аппаратный комплекс выполняет все данные ему функции с помощью мессенджера Telegram и специально разработанного бота.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Дементьев, А.Д. «Умный» дом XXI века / А.Д. Дементьев. – Екатеринбург: Издательство Ridero, 2018 г. – 163 с.
2. Паскаль – Wikipedia. – https://ru.wikipedia.org/wiki/Pascal. Дата обращения 18.05.2021.
3. Язык ассемблера. – https://prog-cpp.ru/asm/. Дата обращения: 27.02.2021
4. Что такое LoRa и зачем она нужна? – https://sprut.ai/client/article/1802. Дата обращения 18.04.2021.
5. Спецификация LoRaWAN. – https://habr.com/ru/post/316954. Дата обращения 18.04.2021.
6. Документация на D203S. – https://www.compel.ru/iДокументация_на_D203S. Дата обращения 18.04.2021.
7. 3DiY – Датчик уровня воды. – https://3d-diy.ru/product/datchik-vody-arduino. Дата обращения 18.04.2021.
8. Telegram – Wikipedia. – https://ru.wikipedia.org/wiki/Telegram. Дата обращения 18.05.2021.
9. Telegram: что это за программа. – https://semantica.in/blog/telegram-chto-eto-za-programma.html. Дата обращения 18.05.2021.
10. Telegram APIs. – https://core.telegram.org/api. Дата обращения 18.05.2021.
11. Telegram. Что это такое и почему это круто. – https://keddr.com/2014/11/telegram-chto-eto-takoe-i-pochemu-eto-kruto/. Дата обращения 18.05.2021.
12. Universal Telegram Bot Library. – https://github.com/witnessmenow/Universal-Arduino-Telegram-Bot. Дата обращения 18.05.2021.

13. Arduino Nano | Аппаратная платформа Arduino. – http://arduino.ru/Hardware/ArduinoBoardNano. Дата обращения 18.05.2021.
14. ПРОКОНТАКТ. Все для радиотехника. – https://procontact74.ru/. Дата обращения 18.05.2021.
15. Python – Wikipedia. – https://ru.wikipedia.org/wiki/Python. Дата обращения 18.05.2021.
16. C – Wikipedia. – https://ru.wikipedia.org/wiki/C. Дата обращения 18.05.2021.
17. WiFi ESP8266 в проектах Arduino. – https://arduinomaster.ru/platy-arduino/arduino-esp8266/. Дата обращения 18.05.2021.
18. Arduino IDE. Что это за программа? – https://blog-programmista.ru/post/80-arduino-ide-cto-eto-za-programma.html. Дата обращения 18.05.2021.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ХАБА

Листинг А.1 – Исходный код программы хаба

```
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>
#include <UniversalTelegramBot.h>
#include <nRF24L01.h>
#include <SPI.h>
#include <RF24.h>
#include <string.h>

// Список возможных команд
#define HUB_SET_STATE_CONTROLLER      0x01
#define HUB_CHECK_CONTROLLER_STATE    0x02
#define DEVICE_SEND_STATUS            0x21
#define DEVICE_SENSOR_SEND_VALUE      0x22

// Список возможных типов устройств
#define SENSOR_LIGHT                  0x01
#define SENSOR_DHT                    0x02
#define SENSOR_WATER                  0x03
#define SENSOR_GROUND_HUM             0x04

#define CONTROLLER_RELAY              0x81
#define CONTROLLER_SERVO              0x82

// Данные Wi-Fi сети
#define WIFI_SSID "*****"
#define WIFI_PASSWORD "******"

// Индивидуальный токен бота
#define BOT_TOKEN "********"
X509List cert(TELEGRAM_CERTIFICATE_ROOT);
WiFiClientSecure secured_client;
UniversalTelegramBot bot(BOT_TOKEN, secured_client);

unsigned long loopDelayTimer;

typedef struct {
  char * name;
```

```
  uint32_t device_address;
  uint8_t device_type;
  uint8_t initialized;
  volatile uint32_t actual_data;
  char * captionStatusON;
  char * captionStatusOFF;
} DeviceData;

typedef struct __attribute__((__packed__)) {
  uint8_t command;
  uint32_t device_address;
  uint8_t device_type;
  uint32_t data;
} DeviceRadioData;

typedef struct {
  int8_t t;
  uint8_t h;
} DHTData;

uint8_t devicesCount = 0;
DeviceData devicesData [10];

int saveSensorValue(uint32_t, uint32_t);
//int setState(uint32_t, uint32_t);
void handleNewMessages(int);
String getListDevices(void);

RF24 radio(4, 15);

void setup(void) {
  SPI.setHwCs(true);
  SPI.begin();
  SPI.setDataMode(SPI_MODE0);
  SPI.setBitOrder(MSBFIRST);
  radio.begin(); // Инициализация модуля NRF24L01
  radio.setChannel(5); // Обмен данными ведется на пятом канале
(2,405 ГГц)
  radio.setDataRate(RF24_1MBPS); // Скорость обмена данными 1
Мбит/сек
  radio.setPALevel(RF24_PA_HIGH); // Выбор высокой мощности
передатчика (-6dBm)
  // Канал получения значений с датчиков
  radio.openReadingPipe(1, 0x00000011LL);
```

```
        // Канал получения состояния контроллеров
        radio.openReadingPipe(2, 0x000000BBLL);
        radio.openWritingPipe(0x000000CCLL);
        radio.startListening(); // Начало прослушивания открывамой трубы
        Serial.begin(115200);

        configTime(0, 0, "pool.ntp.org");      // Получение времени с
сервера
        secured_client.setTrustAnchors(&cert); // Добавление корневого
сертификата для api.telegram.org
        Serial.print("Connecting to Wifi SSID ");
        Serial.print(WIFI_SSID);
        WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
        while (WiFi.status() != WL_CONNECTED)
        {
          Serial.print(".");
          delay(500);
        }
        Serial.print("\nWiFi connected. IP address: ");
        Serial.println(WiFi.localIP());


        Serial.print("Retrieving time: ");
        time_t now = time(nullptr);
        while (now < 24 * 3600)
        {
          Serial.print(".");
          delay(100);
          now = time(nullptr);
        }
        Serial.println(now);

        //uint8_t deviceCount = 3;
        //devicesData = (DeviceData*) malloc(deviceCount);

        devicesData[0] = { "Датчик темп. и вл. на 2-м этаже", 0xF5,
SENSOR_DHT, 0x00, 0x00 };
        devicesData[1] = { "Датчик освещения снаружи", 0x21, SENSOR_LIGHT,
0x00, 0x00 };
        devicesData[2] = { "Автополив огорода", 0xA2, CONTROLLER_RELAY,
0x00, 0x00, "Включен", "Отключен" };
        devicesData[3] = { "Главные ворота", 0xAE, CONTROLLER_SERVO, 0x00,
0x00, "Открыты", "Закрыты" };
```

```
        devicesData[4] = { "Датчик затопления", 0xE7, SENSOR_WATER, 0x00,
0x00 };
        devicesData[5] = { "Датчик влажности почвы", 0x43,
SENSOR_GROUND_HUM, 0x00, 0x00 };
        devicesCount = 6;


        //Serial.println(devicesData[0].name);
        //Serial.println(devicesData[0].sensor_type);

        initControllers();
      }
      void loop(void) {

        if (millis() - loopDelayTimer > 1000)
        {
          int numNewMessages = bot.getUpdates(bot.last_message_received +
1);

          while (numNewMessages)
          {
            handleNewMessages(numNewMessages);
            numNewMessages = bot.getUpdates(bot.last_message_received +
1);
          }
          initControllers();
          loopDelayTimer = millis();
        }

        uint8_t pipe;
        if (radio.available(&pipe))
        {
          DeviceRadioData rx_data;
          radio.read(&rx_data, sizeof(rx_data));

          if (pipe == 1) {
            // Получение значения с датчиков
            if (rx_data.command == DEVICE_SENSOR_SEND_VALUE)
            {
              uint8_t response = saveSensorValue(rx_data.device_address,
rx_data.data);

              DeviceData sensorData =
getDataByAddress(rx_data.device_address);
```

```
            char str[128];
            sprintf(str, "Значение с \"%s\" [0x%08X] получено (%d)",
sensorData.name, rx_data.device_address, rx_data.data);
            Serial.println(str);
          }
        }
        else if (pipe == 2)
        {
          // Получение состояния устройства
          if (rx_data.command == DEVICE_SEND_STATUS)
          {
            String res = saveControllerState(rx_data.device_address,
rx_data.data);
            if (res != "") {
              char str[128];
              sprintf(str, "Состояние устройства %s [0x%08X] получено
(%d)", res.c_str(), rx_data.device_address, rx_data.data);
              Serial.println(str);

              sendExecuteCommandReply(rx_data.device_address);
            }
          }
        }
      }

    }

    /* Функции устройств */
    /* Сохранение полученных с датчиков данных в ОЗУ по адресу */
    int saveSensorValue(uint32_t __device_address, uint32_t __data) {
      for (uint8_t i = 0; i < devicesCount; i++)
      {
        if (devicesData[i].device_address == __device_address)
        {
          devicesData[i].actual_data = __data;
          return 1;
        }
      }
      return 0;
    }

    /* Обновление статуса контроллеров */
    String saveControllerState(uint32_t __device_address, uint32_t
__data) {
```

```
  for (uint8_t i = 0; i < devicesCount; i++)
  {
    if (devicesData[i].device_address == __device_address)
    {
      devicesData[i].initialized = 0x01;
      devicesData[i].actual_data = __data;
      return devicesData[i].name;
    }
  }
  return "";
}

DeviceData getDataByAddress(uint32_t __device_address) {
  for (uint8_t i = 0; i < devicesCount; i++)
  {
    if (devicesData[i].device_address == __device_address)
    {
      return devicesData[i];
    }
  }
  return DeviceData {};
}

int setState(uint32_t address, uint8_t state) {
  radio.stopListening();

  DeviceRadioData data;
  data.command = HUB_SET_STATE_CONTROLLER;
  data.device_address = address;
  data.device_type = 0x00;
  data.data = state;

  radio.write(&data, sizeof(data));

  radio.startListening();
}

void initControllers() {
  for (uint8_t i = 0; i < devicesCount; i++)
  {
    if (devicesData[i].device_type > 0x80
        && devicesData[i].initialized == 0x00)
    {
      radio.stopListening();
```

```
            DeviceRadioData data;
            data.command = HUB_CHECK_CONTROLLER_STATE;
            data.device_address = devicesData[i].device_address;
            data.device_type = devicesData[i].device_type;
            data.data = 0x00;

            radio.write(&data, sizeof(data));

            radio.startListening();

            delay(100);
          }
        }
      }

      /* Функции телеграмма */
      void handleNewMessages(int numNewMessages)
      {
        Serial.println("Получены новые команды из Telegram");

        for (int i = 0; i < numNewMessages; i++)
        {
          String chat_id = bot.messages[i].chat_id;
          String text = bot.messages[i].text;

          Serial.println(bot.messages[i].text);

          char str[128];
          sprintf(str, "[[\"Сводка\"],[\"%s ворота\", \"%s полив\"]]",
                  (devicesData[3].actual_data == 0x00 ? "Открыть" :
"Закрыть"),
                  (devicesData[2].actual_data == 0x00 ? "Включить" :
"Отключить"));
          String keyboardJson = String(str);

      // Команда отправляет доступные для использования команды
          if (text == "/start")
          {
            bot.sendMessageWithReplyKeyboard(chat_id, "Доступные команды",
"", keyboardJson, true, true);
          }
          if (text == "/devices")
```

```
          {
            bot.sendMessage(chat_id, getListDevices(), "Markdown");
          }
      // Команда отправляет состояние датчиков и контроллеров
          if (text == "/status" || text == "Сводка")
          {
            bot.sendMessageWithReplyKeyboard(chat_id, getListValues(), "",
keyboardJson, true, true);
          }
      // Команда отправляет задачу на открытие ворот
          if (text == "/gatesclose" || text == "Открыть ворота")
          {
            setState(0xAE, 1);
      // Команда отправляет задачу на закрытие ворот
          }
          if (text == "/gatesopen" || text == "Закрыть ворота")
          {
            setState(0xAE, 0);
          }
      // Команда отправляет задачу на включение реле на автополив
          if (text == "/relayon" || text == "Включить полив")
          {
            setState(0xA2, 1);
          }
      // Команда отправляет задачу на отключение реле на автополив
          if (text == "/relayoff" || text == "Отключить полив")
          {
            setState(0xA2, 0);
          }
        }
      }

      String getListDevices()
      {
        String str = "Список устройств:\n";
        for (uint8_t i = 0; i < devicesCount; i++) {
          char buf[64];
          sprintf (buf, "- %s\n", devicesData[i].name);
          str += String(buf);
        }
        return str;
```

```
      }

      String getListValues()
      {
        String str = "Состояние датчиков и контроллеров:\n";
        for (uint8_t i = 0; i < devicesCount; i++)
        {
          str += "\n";
          str += String(devicesData[i].name);
          str += ":\n";

          if (devicesData[i].device_type == SENSOR_DHT)
          {
            str += getDHTStringValues(devicesData[i].actual_data);
          }
          else if (devicesData[i].device_type == SENSOR_LIGHT) {
            if (devicesData[i].actual_data == 0) {
              str += "Светло";
            }
            else {
              str += "Темно";
            }
          }
       else if (devicesData[i].device_type > 0x80) {
            if (devicesData[i].actual_data == 0) {
              str += devicesData[i].captionStatusOFF;
            }
            else {
              str += devicesData[i].captionStatusON;
            }
          }
          else {
            char str_temp[6];
            dtostrf((devicesData[i].actual_data / 1024.0) * 100.0, 4, 2,
str_temp);
            char buf[32];
            sprintf (buf, "%s %", str_temp);
            str += String(buf);
          }
          str += "\n";
        }
        return str;
      }
```

```
String getDHTStringValues(uint32_t dht_data)
{
  DHTData dhtData;
  dhtData = *(DHTData*)&dht_data;
  char buf[32];
  sprintf (buf, "темп: %d, вл.: %d %", dhtData.t, dhtData.h);
  return String(buf);
}

void sendExecuteCommandReply(uint32_t __device_address)
{

  char str[128];
  sprintf(str, "[[\"Сводка\"],[\"%s ворота\", \"%s полив\"]]",
          (devicesData[3].actual_data == 0x00 ? "Открыть" :
"Закрыть"),
          (devicesData[2].actual_data == 0x00 ? "Включить" :
"Отключить"));
  String keyboardJson = String(str);

  for (uint8_t i = 0; i < devicesCount; i++) {
    if (devicesData[i].device_address == __device_address)
    {
      sprintf(str, "Состояние  \"%s\" обновлено на %s",
              devicesData[i].name,
              (devicesData[i].actual_data == 0x00 ?
devicesData[i].captionStatusOFF : devicesData[i].captionStatusON));
      bot.sendMessageWithReplyKeyboard("********", String(str), "",
keyboardJson, true, true);
    }
  }
}
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД ПЕРЕДАТЧИКА С ДАТЧИКАМИ

Листинг Б.1 – исходный код программы передатчика с датчиками

```
#include <SPI.h>
#include <nRF24L01.h>
#include "RF24.h"

RF24 radio(9, 10);

// Список возможных команд
#define HUB_SET_STATE_CONTROLLER      0x01
#define HUB_CHECK_CONTROLLER_STATE    0x02
#define DEVICE_SEND_STATUS            0x21
#define DEVICE_SENSOR_SEND_VALUE      0x22

// Список возможных типов датчиков
#define SENSOR_LIGHT                  0x01
#define SENSOR_DHT                    0x02
#define SENSOR_WATER                  0x03
#define SENSOR_GROUND_HUM             0x04

// Адреса датчиков
#define ADDRESS_SENSOR_DHT            0xF5
#define ADDRESS_SENSOR_LIGHT          0x21
#define ADDRESS_SENSOR_WATER          0xE7
#define ADDRESS_SENSOR_GROUND_HUM     0x43

// Пин датчика освещенности
#define PIN_LIGHT_SENSOR      2

// Пин датчика затопления
#define PIN_WATER_SENSOR      A1

// Пин датчика влажности почвы
#define PIN_GROUND_HUM_SENSOR A2

typedef struct __attribute__((__packed__)) {
  uint8_t command;
  uint32_t device_address;
  uint8_t device_type;
```

```
  uint32_t data;
} RadioData;

typedef struct {
  int8_t t;
  uint8_t h;
} DHTData;

void setup(){

    radio.begin(); // Инициализация модуля NRF24L01
    radio.setChannel(5); // Обмен данными ведется на пятом канале
(2,405 ГГц)
    radio.setDataRate(RF24_1MBPS); // Скорость обмена данными 1
Мбит/сек

    radio.setPALevel(RF24_PA_HIGH); // Выбор высокой мощности
передатчика (-6dBm)

    radio.openWritingPipe(0x00000011LL);
    pinMode(PIN_LIGHT_SENSOR, INPUT);
    pinMode(PIN_WATER_SENSOR, INPUT);
    pinMode(PIN_GROUND_HUM_SENSOR, INPUT);
}
void loop() {

  // Датчик DHT
  RadioData data;

  DHTData dhtData;
  dhtData.t = 20;
  dhtData.h = 63;

  data.command = DEVICE_SENSOR_SEND_VALUE;
  data.device_address = ADDRESS_SENSOR_DHT;
  data.device_type = SENSOR_DHT;
  data.data = *(uint32_t*)&dhtData;

  radio.write(&data, sizeof(data));

  delay(4000);

  // Датчик освещенности
  uint8_t lightState = digitalRead(PIN_LIGHT_SENSOR);
```

```
 data.command = DEVICE_SENSOR_SEND_VALUE;
  data.device_address = ADDRESS_SENSOR_LIGHT;
  data.device_type = SENSOR_LIGHT;
  data.data = lightState;

  radio.write(&data, sizeof(data));  delay(4000);

  // Датчик уроавня воды
  uint16_t waterSensorValue = analogRead(PIN_WATER_SENSOR);

  data.command = DEVICE_SENSOR_SEND_VALUE;
  data.device_address = ADDRESS_SENSOR_WATER;
  data.device_type = SENSOR_WATER;
  data.data = waterSensorValue;

  radio.write(&data, sizeof(data));

  delay(4000);
  // Датчик влажности почвы
  uint16_t groundSensorValue = analogRead(PIN_GROUND_HUM_SENSOR);

  data.command = DEVICE_SENSOR_SEND_VALUE;
  data.device_address = ADDRESS_SENSOR_GROUND_HUM;
  data.device_type = SENSOR_GROUND_HUM;
  data.data = groundSensorValue;

  radio.write(&data, sizeof(data));

  delay(4000);
}
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД ПЕРЕДАТЧИКА С УСТРОЙСТВАМИ УПРАВЛЕНИЯ

Листинг В.1 – Исходный код программы передатчика с устройствами управления

```
#include <SPI.h>
#include <nRF24L01.h>
#include "RF24.h"
#include <Servo.h>
#include <EEPROM.h>

RF24  radio(9, 10);

// Список команд
#define HUB_SET_STATE_CONTROLLER      0x01
#define HUB_CHECK_CONTROLLER_STATE    0x02
#define DEVICE_SEND_STATUS            0x21
#define DEVICE_SENSOR_SEND_VALUE      0x22

// Адреса контроллеров
#define ADDRESS_CONTROLLER_RELAY      0xA2
#define ADDRESS_CONTROLLER_SERVO      0xAE

// Типы контроллеров
#define CONTROLLER_RELAY              0x81
#define CONTROLLER_SERVO              0x82

// Адреса контроллеров EEPROM
#define EEPROM_CONTROLLER_SERVO       0x01
#define EEPROM_CONTROLLER_RELAY       0x02

// Пин реле
#define PIN_RELAY                     2
// Пин сервопривода
#define PIN_SERVO                     3
Servo servo;

uint8_t servoState;
uint8_t relayState;
```

```
typedef struct __attribute__((__packed__)) {
  uint8_t command;
  uint32_t device_address;
  uint8_t device_type;
  uint32_t data;
} RadioData;

void setup(){
    Serial.begin(115200);
    radio.begin(); // Инициализация модуля NRF24L01
    radio.setChannel(5); // Обмен данными ведется на пятом канале
(2,405 ГГц)
    radio.setDataRate(RF24_1MBPS); // Скорость обмена данными 1
Мбит/сек
    radio.setPALevel(RF24_PA_HIGH); // Выбор высокой мощности
передатчика (-6dBm)
    radio.openReadingPipe(1, 0x000000CCLL);
    radio.openWritingPipe(0x000000BBLL);
    radio.startListening(); // Начало прослушивания открывамой трубы

    pinMode(PIN_RELAY, OUTPUT);
    servo.attach(PIN_SERVO);


    servoState = EEPROM.read(EEPROM_CONTROLLER_SERVO);

    if(servoState == 0) {
      servo.write(0);
    }
    else {
      servo.write(180);
    }

    relayState = EEPROM.read(EEPROM_CONTROLLER_RELAY);
    digitalWrite(PIN_RELAY, relayState);

    sendStateToHub(ADDRESS_CONTROLLER_SERVO, CONTROLLER_SERVO,
servoState);
    sendStateToHub(ADDRESS_CONTROLLER_RELAY, CONTROLLER_RELAY,
relayState);
    Serial.println("Controller Ready");

}
void loop() {
```

```
       if(radio.available())
       {
        RadioData rx_data;
        radio.read(&rx_data, sizeof(rx_data));

        //Serial.println(rx_data.command);

        // Изменение состояния привода по запросу
        if(rx_data.command == HUB_SET_STATE_CONTROLLER
        && rx_data.device_address == ADDRESS_CONTROLLER_SERVO)
        {
          servo.write(180);
          if(rx_data.data == 0x00) {
            servo.write(0);
          }
          else if(rx_data.data == 0x01) {
            servo.write(180);
          }

          servoState = (uint8_t)rx_data.data;
          EEPROM.write(EEPROM_CONTROLLER_SERVO, servoState);
          sendStateToHub(ADDRESS_CONTROLLER_SERVO, CONTROLLER_SERVO,
rx_data.data);
          Serial.println("Обновлено");
        }

        // Получение и отправка состояния привода
        if(rx_data.command == HUB_CHECK_CONTROLLER_STATE
        && rx_data.device_address == ADDRESS_CONTROLLER_SERVO)
        {
          sendStateToHub(ADDRESS_CONTROLLER_SERVO, CONTROLLER_SERVO,
servoState);
        }

        /* ****************** */

        // Изменение состояния реле по запросу
        if(rx_data.command == HUB_SET_STATE_CONTROLLER
        && rx_data.device_address == ADDRESS_CONTROLLER_RELAY)
        {
          relayState = (uint8_t)rx_data.data;
          digitalWrite(PIN_RELAY, relayState);
          EEPROM.write(EEPROM_CONTROLLER_RELAY, relayState);
```

```
          sendStateToHub(ADDRESS_CONTROLLER_RELAY, CONTROLLER_RELAY,
relayState);
          Serial.println("Обновлено");
        }

        // Получение и отправка состояния реле
        if(rx_data.command == HUB_CHECK_CONTROLLER_STATE
        && rx_data.device_address == ADDRESS_CONTROLLER_RELAY)
        {
          sendStateToHub(ADDRESS_CONTROLLER_RELAY, CONTROLLER_RELAY,
relayState);
        }

       }

       delay(10);
      }

      void sendStateToHub(uint32_t address, uint32_t type, uint8_t data) {

          radio.stopListening();


          RadioData tx_data;
          tx_data.command = DEVICE_SEND_STATUS;
          tx_data.device_address = address;
          tx_data.device_type = type;
          tx_data.data = data;

          radio.write(&tx_data, sizeof(tx_data));

          radio.startListening();
      }
```