**3** РАЗРАБОТКА ФУНКЦИОНАЛЬНОЙ СХЕМЫ

В разделе разработки функциональной схемы рассматривается проектирование устройство на функциональном уровне, в соответствии с полученной ранее структурной схемой.

Данный дипломный проект включает в себя 2 части: аппаратную и программную, которые будут рассматриваться в следующих пунктах подробно.

Стоит отметить, что для обмена данными и командами между аппаратной и программной частью были использованы сокеты. Сокеты позволяют подключаться к адресу и порту в локальной сети, на который настроен wi-fi модуль и общаться посредством отправки строк через консоль.

**3.1** Аппаратная часть

Аппаратная часть представляет собой некоторый набор из блоков, которые были определены в ходе анализа структурной схемы, а именно:

– Центральный контроллер. Представляет собой микроконтроллер Arduino Uno.

– Блок беспроводной связи. Представляет собой wi-fi модуль ESP8266. Имеет плату расширения для возможности подключения нескольких устройств для взаимодействия.

– Блок управления моторами – драйвер моторов Motor Shield L298N, подключается к центральному контроллеру.

– Блок питания. Используется для питания устройства и представляет собой бокс с 3-мя аккумуляторами.

– Блок определения местоположения – хорошо известный датчик определения расстояния HC-SR04, который напрямую подключен к центральному контроллеру.

Рассмотрим подключение данных блоков, которые и составляют основу функциональной схемы.

**3.1.1** Подключение и назначение плат и элементов

Рассмотрим основные характеристики платы Arduino Uno.

В таблице 3.1 представлены основные характеристики главного контроллера.

Чтобы знать, как подключать те или иные платы и элементы к центральному контроллеру, нужно знать распиновку для этого контроллера и какой пин за что отвечает, а также изображение данного контроллера для дальнейшего понимания и понимания расположения пинов.

Таблица 3.1 – Основные характеристики Arduino Uno.

|  |  |
| --- | --- |
| Микроконтроллер | ATmega328 |
| Рабочее напряжение | 5 В |
| Входное напряжение (рекомендуемое) | 7-12 В |
| Входное напряжение (предельное) | 6-20 В |
| Цифровые Входы/Выходы | 14 (6 из которых могут использоваться как выходы [ШИМ](http://arduino.ru/Tutorial/PWM)) |
| Аналоговые входы | 6 |
| Постоянный ток через вход/выход | 40 мА |
| Постоянный ток для вывода 3.3 В | 50 мА |
| Флеш-память | 32 Кб (ATmega328) из которых 0.5 Кб используются для загрузчика |
| ОЗУ | 2 Кб (ATmega328) |
| EEPROM | 1 Кб (ATmega328) |
| Тактовая частота | 16 МГц |

В таблице 3.2 представлена распиновка микроконтроллера Arduino Uno. В последующем описании подключения блоков будет использоваться терминология и обозначения из данной таблицы.

Таблица 3.2 – Распиновка микроконтроллера Arduino Uno

|  |  |  |
| --- | --- | --- |
| **Пин ардуино** | **Адресация на плате** | **Специальное назначение** |
| Цифровой пин 0 | 0 | RX |
| Цифровой пин 1 | 1 | TX |
| Цифровой пин 2 | 2 | Ввод для прерываний |
| Цифровой пин 3 | 3 |  |
| Цифровой пин 4 | 4 |  |
| Цифровой пин 5 | 5 |  |
| Цифровой пин 6 | 6 |  |
| Цифровой пин 7 | 7 |  |
| Цифровой пин 8 | 8 |  |
| Цифровой пин 9 | 9 |  |
| Цифровой пин 10 | 10 | SPI(SS) |
| Цифровой пин 11 | 11 | SPI(MOSI) |
| Цифровой пин 12 | 12 | SPI(MISO) |
| Цифровой пин 13 | 13 | SPI(SCK) |
| Аналоговый пин А0 | А0 или 14 |  |
| Аналоговый пин А1 | А1 или 15 |  |
| Аналоговый пин А2 | А2 или 16 |  |
| Аналоговый пин А3 | А3 или 17 |  |
| Аналоговый пин А4 | А4 или 18 | I2C (SCA) |
| Аналоговый пин А5 | А5 или 19 | I2C (SCL) |
| VIN | VIN |  |
| GND | GND | Вывод земли |
| 5V | 5V | Запитывание устройства (5В) |
| 3.3V | 3.3V | Запитывание устройства (3.3В) |

Рассмотрим расположение пинов для платы центральном контроллере в графическом варианте. Сделано это для более подробного представления и ознакомления с разработкой функциональной схемы.

На рисунке 3.1 представлено расположение пинов на Arduino Uno.



Рисунок 3.1 – Расположение пинов на Arduino Uno [7]

После подробного ознакомления с Arduino Uno можно знакомиться глубже с другими блоками и их подключением.

Рассмотрим самый главный блок по значимости после центрального контроллера – блок беспроводной связи.

Блок беспроводной связи – wi-fi модуль с платой расширения. Является ключевым компонентом дипломного проекта и функциональной схемы. Представляет собой соединительное звено между аппаратной частью и программной. Этот модуль имеет возможность подключения к любой открытой wi-fi точке доступа, возможность конфигурирования и переподключения в активном режиме работы.

В своем арсенале возможностей модуль беспроводной связи имеет способность напрямую связываться и передавать данные центральному контроллеру, Arduino Uno. Происходит это посредством использования пинов RX, TX на центральном контроллере и соответственно на wi-fi модуле. Запитывание wi-fi модуля может происходить через USB-кабель, а также через пины 5V и GND.

В связке с беспроводным модулем идет плата расширения. Она позволяет, не задумываясь, подключать несколько устройств напрямую к модулю и обеспечивать корректную связь и передачу данных. Также на плате расширения помимо пинов, через которые идет связь – RX, TX, присутствует большой набор пинов 5V и GND, что в свою очередь позволяет не только обеспечивать связь в передачи данных, но и поддерживать работоспособность устройств, т.е. обеспечивать дополнительное запитывание.

На рисунке 3.2 представлена подробная распиновка wi-fi модуля ESP8266 NodeMCU v.3 Lua.



Рисунок 3.2 – Расположение пинов на ESP8266 NodeMCU v.3 Lua [8]

На рисунке 3.3 показано соединение платы расширения и wi-fi модуля.

В дальнейшем будет рассматриваться именно такая связка.

Нужно обратить внимание, что обеспечение связи между модулем беспроводной связи и центральным контроллером происходит через подключение RX и TX пинов на центральном контроллере к TX и RX пинам на wi-fi модуле. Только так можно наладить обмен данными напрямую.

Следующим для рассмотрения возьмем блок, отвечающий за управление моторами. Максимальное количество моторов, которые можно подключить к плате L298N равняется 4. В данном дипломном проекте как раз и использовалось 4 мотора. Для удобной реализации движения и поворотов платформы было решено объединить по 2 мотора с правой и левой стороны вместе. Это решает проблему поиска просто решения управления моторами. В случае поворота в левую сторону активными остаются моторы с провой стороны платформы, в случае поворота направо – с левой. В случае движения платформы вперед или назад активными остаются все 4 мотора, которые, соответственно, вращаются в нужном направлении.



Рисунок 3.3 – Wi-fi модуль в связке с платой расширения [9]

Рассмотрев логику объединения моторов, можно глубже ознакомиться с распиновкой модуля управления моторами и, конечно, подключением его к центральному контроллеру.

На рисунке 3.4 представлена распиновка платы L298N.

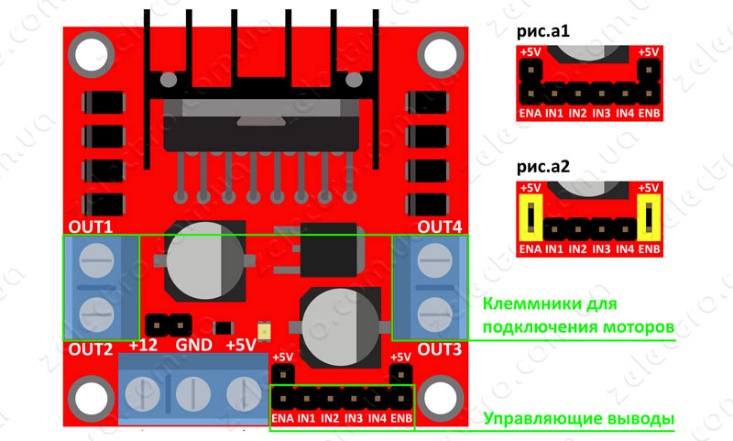


Рисунок 3.4 – Распиновка модуля управления моторами

В случае подключения платы обратимся к рисунку выше. Из него следует, что плата L298N обладает слотами для запитывания и земли.

Для максимальной производительности платы и активного поведения моторов нужно использовать питание в диапазоне 5-12В. Отдельно про блок, отвечающий за питание устройства, будет рассмотрено далее.

Подключение происходит довольно просто – от пина «5V» центрального контроллера протягиваем провод «папа-папа» к слоту с запитыванием платы L298N. С пином «GND» поступает точно таким же образом.

На данный момент плата, отвечающее за управление моторами запитано от центрального контроллера. Теперь нужно обеспечить взаимосвязь и прослушивание команд от центрального контроллера на плате L298N.

Обращаясь к рисунку 3.4 можно заметить, что пины «IN1» – «IN4» необходимо подключить к цифровым входам центрального контроллера. Пины «IN1» – «IN4» отвечают за слоты, куда подключаются моторы, а также обеспечивают обмен и прослушивание команд посылаемых от центрального контроллера.

Итак, плата L298N успешно подключена и запитана.

Перейдем к рассмотрению блока питания.

Данный блок представляет собой обычный бокс с 4-мя аккумуляторами, дающий на выходе 9В и обеспечивающий надежным и долгим питанием все устройство.

На рисунке 3.5 изображен бокс с 4-мя аккумуляторами и разъемом для подключения к Arduino uno.



Рисунок 3.5 – Блок питания

Блок питания имеет разъем для подключения к Arduino Uno.

Следующим блоком является блок определения местоположения. Данный блок представляет собой датчик расстояния HC-SR04 и способен в течение очень малого промежутка времени отсылать инфракрасные лучи для определения препятствий, вспомогательных стен и т.д.

Как происходит взаимодействие датчика с центральным контроллером можно понять если углубиться в его распиновку.

На рисунке 3.6 представлена распиновка модуля, отвечающего за определение местоположения платформы.

Vcc – отвечает за запитывание датчика. Требуемое напряжение равняется 5В.

Ground – обеспечивает землю и подключается к земле на центральном контроллере.

Trigger – вход, импульс 10 мкс уровень TTL.

Eho – выход, сигнал уровень TTL – ШИМ длительность от 150 мкс до бесконечности если нет эха.



Рисунок 3.6 – Распиновка датчика HC-SR04

Подключение датчика происходит напрямую к центральному контроллеру. Пины «VCC», «GND» подключаются напрямую к таким же пинам. Пины «Eho» и «Trigger» датчика определения местоположения обеспечивают обмен информацией, получаемой датчиком и сообщение ее центральному контроллеру. Подключение данных пинов происходит так же напрямую. Пины «Eho» и «Trigger» датчика подключаются к цифровым пинам Arduino Uno.

Датчик HC-SR04 питается от центрального контроллера и вольтаж равняется 5В, что в свою очередь обеспечивает корректную и надежную работу датчика.

Следует отметить, что сам датчик следует располагать осторожно на платформе и направить его следует ровно прямо.

Так же происходят иногда сбои в показаниях датчика. Это обуславливается близким расположением предметов или препятствий.

Центральный контроллер в свою очередь обрабатывает входящую информацию с датчика и посредством алгоритма определения местоположения устанавливает местоположение платформы.

Объединение данных модулей рассмотрим в пункте 3.1.2

А теперь остановимся подробнее на функционировании аппаратной части, рассмотрим методы и свойства.

Исходный код аппаратной части делится на 2 раздела:

– главный контроллер;

– wi-fi модуль.

Рассмотрим методы главного контроллера.

Метод *setup* представляет собой точку вхождения.

В нем инициализируются пины устройства, активируется серийный порт и wi-fi модуль.

Метод *loop* представляет бесконечный цикл программы, в котором выполняются методы. Как и предыдущий метод является основным методом для работы программы.

В данном методе происходит прием входящих команд от клиента, а именно когда клиент подключен напрямую к Arduino Uno.

Метод *handleCmd* является обработчиком входящих команд от контроллера.

В таблице 3.3 представлен перечень команд для настройки wi-fi модуля, которые выполняются на контроллере.

Таблица 3.3. – Перечень команд для главного контроллера.

|  |  |
| --- | --- |
| Команда | Назначение |
| GET\_IP\_ADDRESS | Команда для получения IP-адреса, к которому подключен wi-fi модуль |
| SET\_SSID | Команда для установления SSID. SSID – имя wi-fi точки |
| GET\_SSID | Команда для получения SSID |
| GET\_LOCAL\_SSID | Команда для получения SSID в случае, если wi-fi модуль автоматически подключился к wi-fi точке |
| SET\_PASSWORD | Команда для установления пароля для подключения к wi-fi точке |
| GET\_PASSWORD | Команда для получения пароля wi-fi точки |
| CONNECT\_TO\_WIFI | Команда, которая использует данные SSID и password, находит данную wi-fi точку по SSID и инициирует подключение |
| WIFI\_SESSION | Команда, которая активирует wi-fi сессию. Wi-fi сессия используется для работы с wi-fi модулем, напрямую посылая команды ему с пользовательского компьютера с помощью сокетов |

Метод *movementMenu* предоставляет пользователю, который запрашивает данные о возможностях контроллера, меню движения.

В данном меню можно выбрать и установить движение платформы. Как и в обычном управлении роботом, можно выбрать 4 стороны направление: прямо, назад, влево, вправо. Также возможно команда «стоп» и выход из меню.

За движение робота отвечают методы:

– moveForward;

- moveBack;

- moveRight;

- moveLeft;

- moveStop.

Для осуществления движения в основной библиотеке Arduino предусмотрены методы установления высокого и низкого напряжений.

Для примера возьмем движение вперед. Для этого нужно сделать следующее:

digitalWrite (RIGHT\_DOWN, LOW);

digitalWrite (RIGHT\_UP, HIGH);

digitalWrite (LEFT\_UP, HIGH);

digitalWrite (LEFT\_DOWN, LOW),

где RIGHT\_DOWN – пин на модуле управления моторами, отвечающий за движение правой стороны вниз,

RIGHT\_UP – пин на модуле управления моторами, отвечающий за движение правой стороны вверх,

LEFT\_DOWN – пин на модуле управления моторами, отвечающий за движение левой стороны вниз,

LEFT\_UP – пин на модуле управления моторами, отвечающий за движение левой стороны вверх.

В случае движения в другие стороны и полной остановки используется тот же принцип.

**3.1.2** Архитектура аппаратной части

Архитектура аппаратной части представлена на рисунке 3.7

Клиентом в данной архитектуре выступает пользователь, который подключается к главному контроллеру через USB-кабель. Далее на компьютере он выбирает конкретный порт, к которому присоединено устройство. После того, как пользователь подключился – он может конфигурировать устройство. Конфигурация происходит через терминал или другие программные средства.

Остальные модули и их взаимодействие с главным контроллером были рассмотрены выше.



Рисунок 3.7 – Архитектура аппаратной части

**3.2** Программная часть

Программная часть представляет собой остальной набор модулей, которые были представлены в структурной схеме, а именно:

– Модуль обмена данными. Используется для обмена и обработки данных между программной и аппаратной частью.

– Модуль управления. Представляет собой модуль, который отвечает за ввод команд на стороне клиента.

– Модуль отображения данных. Используется для вывода получаемой информации от центрального контроллера.

Рассмотрим модуль обмена данными.

Использование данного модуля является ключевым звеном в объединении в одно целое аппаратной и программной части. Также явлется связующим модулем в программной части.

Модуль обмена данными выполняет роль распределения входящей информации от модуля управления и модуля беспроводной связи, а также отправления данных на модуль отображения, если это необходимо.

Модуль обмена данными для корректного обмена с модулем беспроводной связи использует технологию взаимодействия,

именуемая как сокеты.

На рисунке 3.8 изображено классическое взаимодействие с помощью сокета клиента и сервера.



Рисунок 3.8 – Взаимодействие клиента и сервера через сокет

Сокеты позволяют подключаться к любой открытой wi-fi точке, тем самым создавая TCP-окно.

Подробнее о влиянии сокетов в разработке проекта рассмотрим в пункте 3.2.2

В данном дипломном проекте можно рассматривать аппаратную часть, как серверную часть, а программное средство – клиентская часть.

Рассмотрим следующий модуль – модуль управления.

Данный модуль представляет собой программный модуль, написанный как и вся клиентская часть, на языке Python. Использование данного блока позволяет отправлять команды модулю обработки данных, который в свою очередь определяет к чему относится данная команда – к модулю отображения или к аппаратной части.

Последний модуль – модуль отображения.

Использование данного модуля необходимо для отображения данных, чтобы пользователь понимал, что происходит и что команда, введенная им, повлияла на процесс работы устройства.

Теперь рассмотрим более детально функционирование программной части, рассмотрим значение методов и свойств исполняемых файлов.

– *Main.py* представляет собой главный исполняемый файл на клиентской части. К нему подключаются основные модули, а также дополнительные библиотеки.

– *Connection.py* представляет собой исполняем файл, который отвечает за подключение клиента к модули беспроводной связи. Также выполняет проверку IP-адреса и наличие сети.

– *Communication.py* представляет собой исполняемый файл, в котором происходит общение между аппаратной и клиентской частью. Выполняет проверку валидности отправляемых данных и дополнительную конвертацию в пригодный для чтения вид.

– *Error.py* представляет собой модуль обработки ошибок, которые могут возникать в ходе выполнения программы.

– *Menu.py* представляет модуль, который отвечает за представление дополнительной графической информации и за оповещение клиента о предоставляемых ему возможностях в ходе выполнения программы.

– *Executor.py* представляет собой модуль, который выполняет команды, отправленные клиентом.

– *Parser.py* представляет собой модуль конвертации приходящих и исходящих команд.

– *Constants.py* представляет собой модуль с нужными константами для работы клиентской программы.

Используя такое разбиение на модули, обеспечивается ясная и последовательная картина исполнения команд, их обработки и отображения.

В случае непредвиденных ошибок мы используем дополнительно модуль обработки ошибок.

Связь между исполняемыми файлами программной части представлена на рисунке 3.9



Рисунок 3.9 – Связь между исполняемыми файлами

**3.2.1** Архитектура программной части

Архитектура программной части изображена на рисунке 3.10

Сервером является аппаратная часть. Выше было указано, что такое упрощение оправдывает себя.

Модуль обмена данными, как и говорилось выше, является связующим звеном в построении аппаратно-программной архитектуры.

Клиентом является пользователь, который запустил главный исполняемый файл, отвечающий за подключение к IP-адресу, на который настроен wi-fi модуль.



Рисунок 3.10 – Архитектура программной части

**3.2.2** Использование сокетов и TCP-соединения

Сокет — [программный интерфейс](https://ru.wikipedia.org/wiki/API) для обеспечения обмена данными между [процессами](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)" \o "Процесс (информатика)). Процессы при таком обмене могут исполняться как на одной компьютере, так и на различных, связанных между собой [сетью](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C" \o "Компьютерная сеть).

Также сокет это абстрактный объект, представляющий конечную точку соединения.

Существуют клиентские и серверные сокеты.

Клиентские сокеты можно сравнить с конечными аппаратами [телефонной сети](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BB%D0%B5%D1%84%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C_%D0%BE%D0%B1%D1%89%D0%B5%D0%B3%D0%BE_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F" \o "Телефонная сеть общего пользования), а серверные — с [коммутаторами](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BB%D0%B5%D1%84%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BC%D1%83%D1%82%D0%B0%D1%82%D0%BE%D1%80" \o "Телефонный коммутатор). Клиентское приложение (например, [браузер](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80" \o "Браузер)) использует только клиентские сокеты, а серверное (например, [веб-сервер](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80" \o "Веб-сервер), которому браузер посылает запросы) — как клиентские, так и серверные сокеты.

Как известно, для взаимодействия между машинами с помощью стека протоколов [TCP/IP](https://ru.wikipedia.org/wiki/TCP/IP" \o "TCP/IP) используются адреса и порты. Первое на текущий момент представляет собой 32-битный адрес (для протокола [IPv4](https://ru.wikipedia.org/wiki/IPv4" \o "IPv4), 128-битный для [IPv6](https://ru.wikipedia.org/wiki/IPv6" \o "IPv6)), наиболее часто его представляют в символьной форме mmm.nnn.ppp.qqq (адрес, разбитый на четыре поля, разделённых точками, по одному байту в поле). Номер порта в диапазоне от 0 до 65535 (для протокола [TCP](https://ru.wikipedia.org/wiki/TCP" \o "TCP)).

Эта пара и есть сокет («гнездо», соответствующее [адресу](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%82%D0%B5%D0%B2%D0%BE%D0%B9_%D0%B0%D0%B4%D1%80%D0%B5%D1%81" \o "Сетевой адрес) и [порту](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82_(%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5_%D1%81%D0%B5%D1%82%D0%B8))).

В процессе обмена, как правило, используется два сокета — сокет отправителя и сокет получателя.

Каждый [процесс](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81_(%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) может создать «слушающий» сокет (серверный сокет) и привязать его к [порту](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82_(TCP/UDP)" \o "Порт (TCP/UDP)) операционной системы.

Слушающий процесс обычно находится в цикле ожидания, то есть просыпается при появлении нового соединения. При этом сохраняется возможность проверить наличие соединений на данный момент, установить тайм-аут для операции и т. д.

Каждый сокет имеет свой адрес. ОС семейства UNIX могут поддерживать много типов адресов, но обязательными являются [INET-адрес](https://ru.wikipedia.org/wiki/INET-%D0%B0%D0%B4%D1%80%D0%B5%D1%81" \o "INET-адрес) и [UNIX-адрес](https://ru.wikipedia.org/w/index.php?title=UNIX-%D0%B0%D0%B4%D1%80%D0%B5%D1%81&action=edit&redlink=1). Если привязать сокет к UNIX-адресу, то будет создан специальный файл (файл сокета) по заданному пути, через который смогут сообщаться любые локальные процессы путём чтения/записи из него. Сокеты типа [INET](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BA%D0%B5%D1%82%D1%8B_%D0%91%D0%B5%D1%80%D0%BA%D0%BB%D0%B8" \o "Сокеты Беркли) доступны из сети и требуют выделения номера порта.

Обычно клиент явно «подсоединяется» к слушателю, после чего любое чтение или запись через его [файловый дескриптор](https://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D1%8B%D0%B9_%D0%B4%D0%B5%D1%81%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%80" \o "Файловый дескриптор) будут передавать данные между ним и сервером.

В исполняемом файле «*Connection.py»* выполняется подключение к IP-адресу, на который настроен wi-fi модуль.

В данном модуле изначально происходит проверка на валидность введенного IP-адреса.

Регулярное выражение для проверки IP-адреса выглядит так:

'^(([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])\.){3}([0-9]|[1-9][0-9]|1[0-9]{2}|2[0-4][0-9]|25[0-5])$'

С помощью Python библиотеки «*re*» мы производим проверку.

Теперь подробнее рассмотрим установление TCP-соединения между клиентом и модулем беспроводной связи, а именно wi-fi модулем.

Для этого нужно обратить внимание на библиотеку «*socket*», которая является основной в программной части.

**3.3** Архитектура проекта

На рисунке 3.11 представлена архитектура данного проекта, состоящего из аппаратной и программной части.

Как можно заметить Клиент является главным звеном.

Клиент может управлять как и аппаратной частью через USB-соединение на любой ЭВМ, а также с помощью программного средства подключаться к wi-fi модулю и выполнять дополнительные команды.



Рисунок 3.11 – Архитектура проекта