Федеральное государственное автономное образовательное учреждение   
высшего образования

«Уральский федеральный университет

имени первого Президента России Б.Н. Ельцина»

Институт новых материалов и технологий

Кафедра «Теплофизика и информатика в металлургии»

ДОПУСТИТЬ К ЗАЩИТЕ:

Макуху Дмитрия Алексеевича

Зав. кафедрой Н.А. Спирин

« » 2017 г.

Магистерская диссертация

**Разработка программного обеспечения для беспроводной системы связи сети автономных датчиков**

**МД 230400.000000.502 ПЗ**

Руководитель

Старший преподаватель В.Ю. Носков

(должность, уч. ст., зван.) (подпись) (расшифровка подписи)

Нормоконтролер

доцент, к.т.н. Е.В. Киселев

(должность, уч. ст., зван.) (подпись) (расшифровка подписи)

Студент

НМТМ-253901 Д.А. Макуха

(группа.) (подпись) (расшифровка подписи)

Екатеринбург,

2017 г.

РЕФЕРАТ

Пояснительная записка содержит 76 страниц, 22 рисунков, 6 библиографических наименований.

БЕСПРОВОДНАЯ СЕТЬ, ДАТЧИК, СЕНСОР, АЛГОРИТМ, ТОПОЛОГИЯ, ATMEL, ATMEGA, МИКРОКОНТРОЛЛЕР.

Настоящая выпускная квалификационная работа посвящена разработке программного обеспечения для реализации работы беспроводной системы связи сети автономных датчиков.

В ходе работы были исследованы существующие технологии и стандарты по созданию алгоритмов работы для беспроводных систем связи автономных датчиков, изучены их достоинства и недостатки. Подобраны программные и аппаратные компоненты необходимые для реализации системы. Разработана методика тестирования реализованных устройств, а также проведены испытания готовой продукции.

Результатом работы стал программно-аппаратный продукт обеспечивающий работу беспроводной системы связи сети автономных датчиков.

Данный продукт предназначен для отслеживания любых важных технологических параметров исследуемого объекта без участия человека в труднодоступных местах. Например, в металлургических цехах, где в связи со спецификой месторасположения трудно прокладывать кабельную инфраструктуру до каждого конкретного датчика.

ПЕРЕЧЕНЬ ПРИНЯТЫХ СОКРАЩЕНИЙ

|  |  |
| --- | --- |
| IoT | Internet of Things |
| БСС | Беспроводная сенсорная сеть |
| EEPROM | Electrically Erasable Programmable Read-Only Memory |
| RAM | Random Access Memory |
| RISC | Reduced Instruction Set Computer |
| JTAG | Joint Test Action Group Interface |
| SPI | Serial Peripheral Interface |
| ЦПУ | центральный процессор управления |
| OCD | On-Chip Debugger |
| АЛУ | арифметико-логическое устройство |
| LCD | Liquid-Crystal Display |
| USART | Universal Asynchronous Receiver-Transmitter |
| TWI | Two-Wire serial Interface |
| ШИМ | Широтно-импульсная модуляция |
| АЦП | Аналого-цифровой преобразователь |
| IDE | Integrated Development Environment |
| БД | База данных |
| СУБД | Система управления базами данных |

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc484763845)

[1 ОПИСАНИЕ РАЗРАБАТЫВАЕМОГО РЕШЕНИЯ 7](#_Toc484763846)

[1.1 Алгоритм работы 7](#_Toc484763847)

[1.2 Аппаратные компоненты 20](#_Toc484763848)

[1.3 Принципиальные схемы подключения 31](#_Toc484763849)

[1.4 Программные компоненты 33](#_Toc484763850)

[2 РЕЗУЛЬТАТЫ ПРОВЕДЕННЫХ ИСПЫТАНИЙ 43](#_Toc484763851)

[2.1 Методика проведения испытаний 43](#_Toc484763852)

[2.2 Проведение испытаний разработанного программного обеспечения 45](#_Toc484763853)

[ЗАКЛЮЧЕНИЕ 51](#_Toc484763854)

[БИБЛИОГРАФИЧЕСКИЙ СПИСОК 52](#_Toc484763855)

[ПРИЛОЖЕНИЕ А](#_Toc484763856) [Фрагменты кода координатора сети 53](#_Toc484763857)

[ПРИЛОЖЕНИЕ Б](#_Toc484763858) [Фрагменты кода оконечного устройства 61](#_Toc484763859)

[ПРИЛОЖЕНИЕ В](#_Toc484763860) [Фрагменты кода модуля обработки принимаемой информации 76](#_Toc484763861)

# ВВЕДЕНИЕ

В настоящее время информационные технологии все больше входят в нашу жизнь. Сейчас трудно найти человека, у которого нет смартфона и портативного компьютера. Практически любую операцию человек может сделать через интернет, например, заказать еду, либо записаться к врачу.

Важную нишу в данных технологиях занимают те, которые основаны на базе построения беспроводных сетей. Сейчас существует такое понятие как Internet of Things, что в переводе на русский язык означает – интернет вещей. Интернет вещей – концепция [вычислительной сети](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) физических предметов («вещей»), оснащённых встроенными технологиями для взаимодействия друг с другом или с внешней средой [[1]](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82_%D0%B2%D0%B5%D1%89%D0%B5%D0%B9#cite_note-1), рассматривающая организацию таких сетей как явление, способное перестроить экономические и общественные процессы, исключающее из части действий и операций необходимость участия человека [[2]](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82_%D0%B2%D0%B5%D1%89%D0%B5%D0%B9#cite_note-.D0.AD.D1.88.D1.82.D0.BE.D0.BD.E2.80.942009.E2.80.94.E2.80.94-2).

Системы, основанные на данной технологии, сильно упрощают жизнь людей. Т.к. данные системы сетей датчиков автономны, т.е. не требуют человеческого участия, то можно осуществлять мониторинг или управление на производстве с минимальными затратами человеческих ресурсов. Человеческие ресурсы будут требоваться только на замену вышедших из строя датчиков, либо элементов питания, что является малой частью от времени затраченного на мониторинг и управление.

В представленной работе будет рассмотрен разработанный алгоритм построения и функционирования БСС. Будет подобрано аппаратное и программное обеспечение для реализации системы. Затем будет разработано программное обеспечение для реализации сети автономных беспроводных устройств. И в заключении будет проведено тестирование реализованного программного продукта в реальных условиях.

Цели данной работы: разработать алгоритм работы беспроводной системы связи сети автономных датчиков. Подобрать аппаратные компоненты, на базе которых будет разработано программное обеспечение. Подобрать программные компоненты

Выпускная квалификационная работа состоит из введения, двух глав, заключения, библиографического списка и приложения.

Первая глава содержит описание разработанного алгоритма работы БСС. Подбор программных и аппаратных компонентов необходимых для реализации системы, а также принципиальные схемы соединения аппаратных компонентов.

Во второй главе приведено описание разработанной методики проведения испытаний реализованного программного обеспечения, а также само тестирование.

В приложении А приводится исходный текст разработанного программного продукта.

# 1 ОПИСАНИЕ РАЗРАБАТЫВАЕМОГО РЕШЕНИЯ

Разработанное решение можно представить в виде нескольких частей:

* алгоритм работы узлов сети;
* аппаратные компоненты, на основе которых выполнено решение;
* принципиальные схемы подключения;
* программные компоненты, необходимые для построения разработанного решения.

## 1.1 Алгоритм работы

Беспроводная сеть передачи данных должна быть автономной, обладать самостоятельной сходимостью, скважным включением новых оконечных узлов в сеть, поддерживать режим сна для минимизации потребления электроэнергии оконечных устройств, имеющих автономное питание и ретрансляцию данных через транзитные узлы. Также должно быть предусмотрено реагирование на выход из строя некоторых устройств, которые участвуют в передаче транзитного трафика и автоматическая перезагрузка оконечных устройств при зависании.

Для реализации данной сети необходимо разработать 2 типа устройств:

* координатор сети;
* оконечные устройства.

Общая топология беспроводной сети представляет собой, на физическом уровне – звездообразную форму, на логическом – древовидную.

В соответствии с древовидной топологией можно выделить несколько ветвей дерева состоящих из узлов сети, причем каждая ветвь должна являться отдельным направлением, для минимизации широковещательного трафика в сети.

Пример физической организации сети представлен на рисунке 1.1.

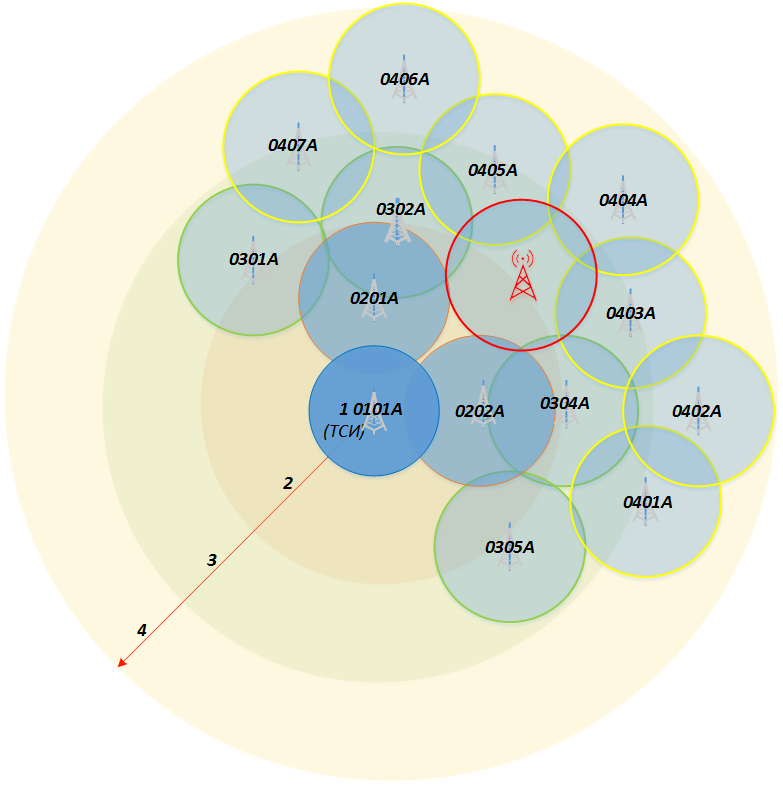


Рисунок 1.1 – Пример физической организации сети

На рисунке 1.2 представлен пример возможной организации сети на логическом уровне с учетом направлений.

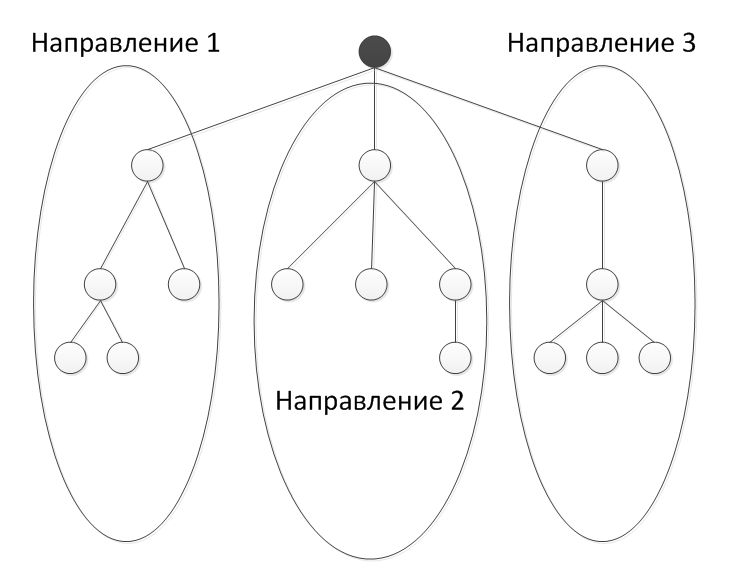


Рисунок 1.2 – Пример возможной организации сети

Координатор является центральным устройством всей сети. Он должен принимать запросы на регистрацию, обрабатывать полученные данные и управлять режимами сна оконечных устройств.

Оконечные устройства могут являться как конечными узлами так и транзитными узлами, т.е. передавать не только свои данные, но и быть ретрансляторами чужих данных. Также данные устройства должны обладать функцией регистрации в сети при первом включении, и перерегистрации в случае недоступности координатора. Должны иметь функцию перезапуска в случае зависания. А также должен поддерживать режим сна для минимизации электропотребления.

Чтобы устройства корректно обрабатывали полученные данные, на всех устройствах сети должна быть согласованная структура передаваемых данных.

Также вся полученная координатором сети информация должна быть сохранена для дальнейшего использования, например, отображения в виде графиков, либо для построения отчетов за определенный период.

Поэтому для реализации технического решения были разработаны:

* алгоритм работы координатора;
* алгоритм работы оконечного устройства в режиме регистрации, в режиме передачи данных, а также с предусмотренным режимом сна;
* структура данных передаваемых по сети;
* модуль обработки полученной информации.

Алгоритм работы каждого из устройств состоит из нескольких согласованных между собой функциональных блоков.

1.1.1 Алгоритм работы координатора. Координатор является центральным устройством всей сети. Он должен принимать запросы на регистрацию, обрабатывать полученные данные и управлять режимами сна оконечных устройств. В соответствии с этим алгоритм данного узла выглядит следующим образом.

При включении координатора в сеть можно выделить следующие шаги:

Шаг 1. Чтение из памяти конфигурационных параметров, таких как:

* количество направлений;
* количество устройств;
* идентификаторы зарегистрированных устройств.

После чего переходим к шагу 2.

Шаг 2. Устройство запускает таймер на 30 минут, после чего переходим к шагу 3.

Шаг 3. Если выставленное время прошло, то координатор отсылает служебное сообщение, обязывающее уснуть половину направлений сети, чередуя каждые 30 минут четность направлений. Данный параметр можно настроить в зависимости от потребностей. Затем переходим к шагу 2. Если заданное время не прошло, то переходим к шагу 4.

Шаг 4. Прослушивание радиоканала для получения данных от оконечного оборудования. Если пришел пакет с данными, то переходим к шагу 5.

Шаг 5. Проверяем, какого типа пришел пакет. Пакеты бывают следующих типов:

* запрос на регистрацию;
* данные;
* пакет с конфигурационной информацией сна;
* ответ на регистрацию.

Последние два пакета посылаются только координатором сети, поэтому до координатора они дойти не могут. В любом случае при получении координатором последних двух пакетов, они отбрасываются, т.к. данные пакеты предназначены для оконечных устройств.

При получении данных с запросом на регистрацию, либо пакета с данными переходим к шагу 6 или 10 соответственно.

Шаг 6. Проверка направления устройства из полученного пакета. Если направление равно «999» - это служебное направление, то происходит переназначение направления устройства на новое равное R, которое рассчитывается по формуле (1.1), количество текущих направлений считывается из памяти устройства. В данном случае это означает, что устройство пришло из непосредственной близости к координатору, а не через транзитный узел. Иначе поле направления остается без изменения. Переходим к шагу 7.

R =r+1, (1.1)

где r – текущее количество направлений.

Шаг 7. Количество направлений инкрементируется и записывается в память координатора и после успешного выполнения операций переходим к шагу 8.

Шаг 8. Производится поиск идентификатора устройства в памяти координатора. Если данное устройство не найдено, то устройству назначается новый идентификатор, не принадлежащий ни одному устройству в сети, а также записывается в память координатора. Если устройство было найдено, то данное поле не изменяется. После чего переходим к шагу 9.

Шаг 9. Изменение типа команды на «ответ на регистрацию» и отправка пакета в нужном направлении до оконечного устройства. Затем переходим к шагу 3.

Шаг 10. При получении пакета с данными, из пакета выгружаются некоторые параметры и передаются к модулю обработки информации. Список параметров:

* идентификатор устройства;
* тип данных;
* данные.

После чего переходим к шагу 3.

Таким образом, координатор циклично обрабатывает все полученные пакеты в соответствии с их назначением, а также каждые 30 минут посылает в сон половину существующих направлений с устройствами. Графическая интерпретация алгоритма представлена на рисунке 1.3.

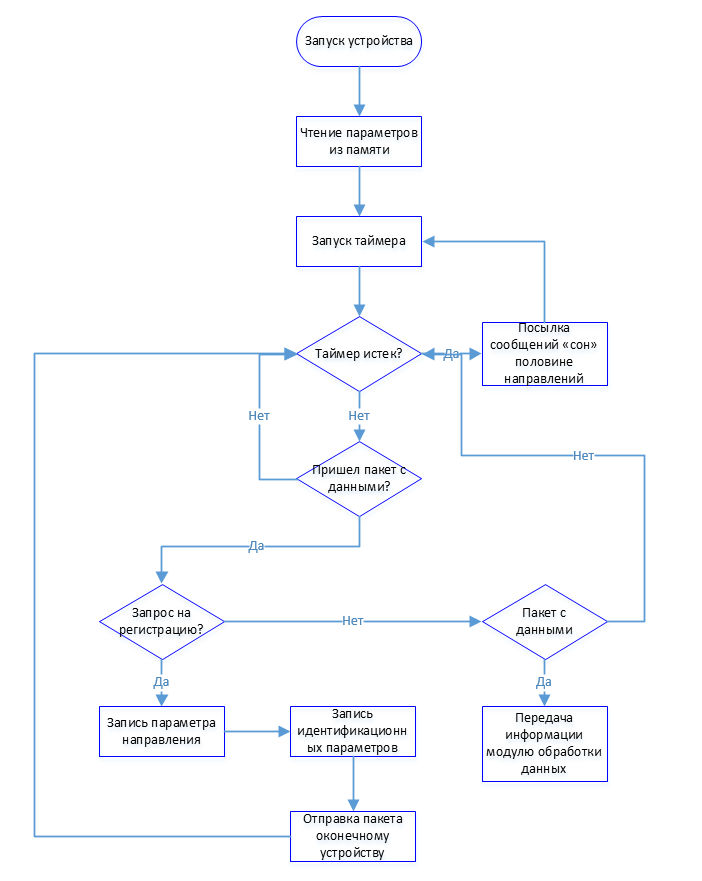


Рисунок 1.3 – Алгоритм работы координатора

1.1.2 Алгоритм работы оконечного устройства. Оконечное устройство служит для передачи, какой-либо информации с прикрепленного датчика координатору. Для полноценной работы сети необходимо, чтобы у оконечных устройств были предусмотрены режимы ретрансляции данных, режима сна по запросу от координатора сети. Возможность перезагрузки в случае зависания устройства, а также режим регистрации при подключении к сети и перерегистрации при возникновении проблем со связью до координатора. В соответствии с этим обобщенный алгоритм данного узла выглядит следующим образом.

При включении оконечного узла в сеть можно выделить следующие шаги:

Шаг 1. Запускаем таймер на 1 час 10 минут, для отслеживания режима сна и переходим к шагу 2. Данный временной интервал зависит от времени отправки координатором служебных сообщений, погружающих устройства в сон. Поэтому при выставлении данного таймера нужно опираться на двойное время выставленное в координаторе плюс небольшое отклонение в большую сторону.

Шаг 2. Если прошло заданное время, то выставляем значение внутреннего параметра «регистрация» в 1, тем самым устройство становится незарегистрированным, из-за потери связи с координатором сети и переходим к шагу 1. Иначе переходим к шагу 3

Шаг 3. Происходит проверка параметров в памяти устройства.

Параметры для проверки:

* «сон»;
* «регистрация».

Параметры проверяются последовательно, если параметр сон выставлен в 1, то переходим к шагу 4. Если параметр регистрация выставлен в 1, то переходим к шагу 6, иначе переходим к шагу 8.

Шаг 4. Устройство засыпает на определенное координатором время, после чего переходим к шагу 5.

Шаг 5. После режима сна переходим к шагу 1.

Шаг 6. Устройство переходит к функциональному блоку регистрации. После чего если регистрация прошла успешно тогда переходим к шагу 7, иначе переходим к шагу 2.

Шаг 7. Выставляем таймер передачи данных на 5 минут и переходим к шагу 8.

Шаг 8. Если время, выставленное на таймере передачи данных, вышло, то переходим к шагу 9, иначе переходим к шагу 10.

Шаг 9. Считываем информацию с датчика, меняем тип команды на «передача данных», отправляем пакет в нужном направлении. После чего переходим к шагу 7.

Шаг 10. Прослушиваем эфир. Если пришел пакет данных, то переходим к шагу 11. Иначе переходим к шагу 2.

Шаг 11. Проверяем, с какого направления пришел пакет. Если направление пакета совпадает с направлением устройства, либо направление равно «999», то переходим к шагу 12. Иначе переходим к шагу 2.

Шаг 12. Проверяем, какого типа пришел пакет. Если пакет пришел с данными, то переходим к шагу 13. Если пакет пришел с запросом на регистрацию, переходим к шагу 14. В случае если пакет поступил с ответом на регистрацию, то переходим к шагу 15. Если пакет приходит с запросом на сон, то переходим к шагу 16.

Шаг 13. Перенаправляем пакет в сторону координатора и переходим к шагу 2.

Шаг 14. Меняем параметр направление на параметр у текущего устройства, после чего отправляем пакет в сторону координатора, и переходим к шагу 2.

Шаг 15. Отправляем пакет в сторону противоположную координатору, и переходим к шагу 2.

Шаг 16. Выставляем параметр «сон» в 1, после чего пересылаем пакет в сторону противоположную координатору и переходим к шагу 2.

Графическое представление общего алгоритма работы оконечного устройства изображено на рисунке 1.4.

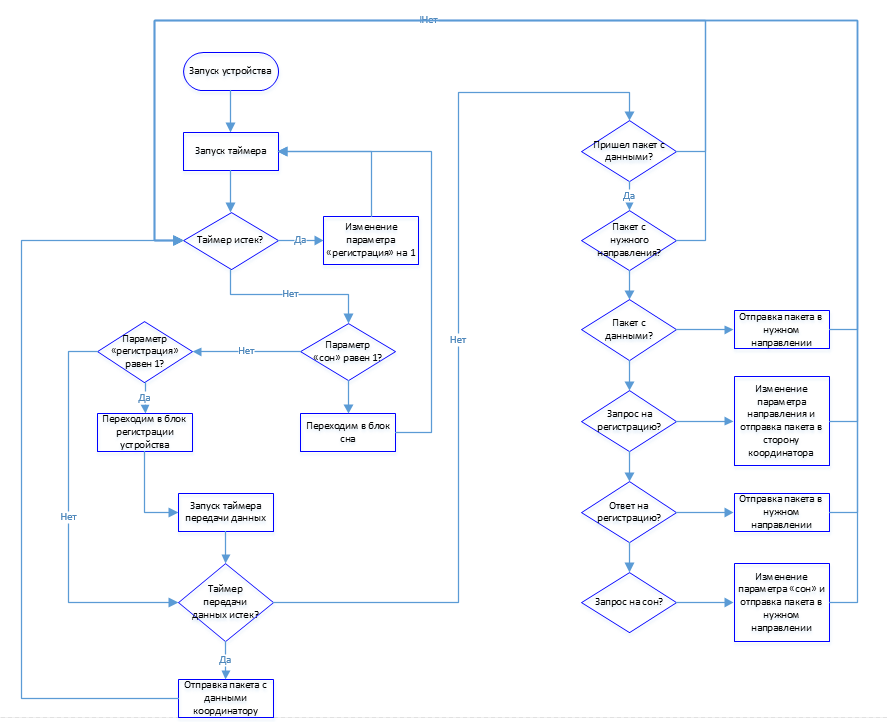


Рисунок 1.4 – Общий алгоритм оконечного устройства

Из общего алгоритма оконечного устройства можно выделить несколько функциональных блоков, каждый из которых выполняет определенную функцию, такие как:

* режим сна;
* регистрационный блок.

Перейдем к рассмотрению каждого из них.

Пошаговый алгоритм работы режима сна представлен ниже.

Шаг 1. Включаем таймер в соответствии с пришедшими временными параметрами, и переходим к шагу 2.

Шаг 2. Если заданное время не вышло, то переходим к шагу 3, иначе переходим к шагу 5.

Шаг 3. Выключаем всю периферию, интерфейсы, порты и внешние прерывания и переходим к шагу 4.

Шаг 4. Погружаем устройство в сон, и переходим к шагу 2.

Шаг 5. Пробуждаем устройство, включаем всю периферию, интерфейсы, порты и внешние прерывания, после чего переходим к шагу 6.

Шаг 6. Выставляем параметр «сон» в 0.

Графическая интерпретация режима сна представлена на рисунке 1.5.

Теперь перейдем к описанию алгоритма регистрации устройства.

Шаг 1. Выставляем таймер регистрации равный 10 секунд и переходим к шагу 2.

Шаг 2. Выставляем тип команды в передаваемом пакете на «запрос на регистрацию» и отправляем пакет в сеть. После чего переходим к шагу 3.

Шаг 3. Пока не пришел пакет с ответом на регистрацию, ожидаем сообщение. Если во время ожидания истек таймер, выставленный на первом шаге, то переходим к шагу 1. Если пришло сообщение с ответом на регистрацию, то переходим к шагу 4.

Шаг 4. Записываем в память идентификатор устройства, направление, и метрику, после чего изменяем параметр «регистрация» на 0. Таким образом устройство становится зарегистрированным и готово к осуществлению передачи информации.

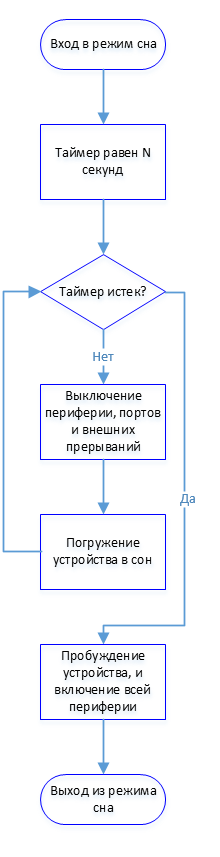


Рисунок 1.5 – Алгоритм режима сна

Алгоритм работы регистрации устройства в графическом виде представлен на рисунке 1.6.

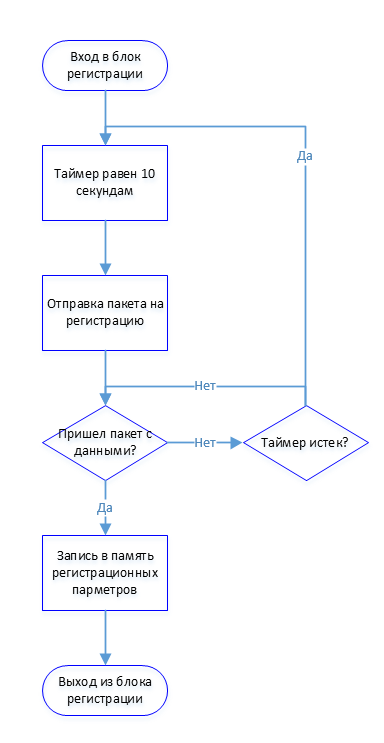


Рисунок 1.6 – Алгоритм регистрации устройства в сети

1.1.3 Структура пересылаемых кадров. Для корректного функционирования сети, чтобы все устройства понимали друг друга, по сети должны передаваться одинаковые пакеты с данными. Также нам необходимо, чтобы передача данных происходила не отдельными значениями, а набором величин, которые будут описывать от какого узла пришли данные, какому узлу предназначаются, какие операции с ними необходимо производить, и собственно, сами данные. Таким образом, мы достигнем универсальности и простоты подключения новых компонентов в нашу систему, а также согласованности и однозначного восприятия данных всеми узлами сети.

Также нам необходимо, чтобы пакеты, пересылаемые по сети, были достаточно малы, тем самым мы сокращаем риск повреждения пакета при передаче, уменьшаем время передачи данных и, следовательно, сокращаем нагрузку на радиоканал.

В итоге с учетом алгоритмов работы узлов сети и вышеизложенной информации, была спроектирована следующая структура данных, приведенная в таблице 1.1.

Таблица 1.1 – Структура данных передаваемых по сети

|  |  |
| --- | --- |
| Название | Описание |
| device\_id | Идентификатор передатчика |
| last\_device\_id | Идентификатор передатчика необходимый для регистрации устройства |
| destination\_id | Идентификатор получателя |
| route | Направление оборудования |
| metric | Метрика устройства |
| command | Идентификатор команды |
| type\_data | Тип данных |
| data | Данные |

1.1.4 Модуль обработки полученной информации. Модуль обработки информации в качестве входных данных получает информацию от устройства в формате, приведенном в предыдущем пункте.

После получения данных, из структуры извлекаются поля идентификатора устройства, тип передаваемых данных, а также сами данные и размещаются в соответствующих кортежах базы данных.

После разработки алгоритмов работы устройств, а также структуры передаваемых данных необходимо подобрать аппаратное обеспечение, на базе которого будет реализована система.

## 1.2 Аппаратные компоненты

Для реализации взаимосвязанной системы беспроводной сети было необходимо подобрать следующие компоненты:

* микроконтроллеры для координаторов сети, а также для оконечных узлов;
* приемники и передатчики на базе радиоканала на частоте 433 МГц;
* подключаемые датчики;
* вспомогательные радиокомпоненты.

Перейдем к подробному описанию каждого компонента.

1.2.1 Микроконтроллеры. В качестве микроконтроллеров, на базе которых будет простроена вся вычислительная инфраструктура беспроводной сети, были выбраны микроконтроллеры семейства AVR.

AVR — семейство восьмибитных микроконтроллеров фирмы Atmel. Микроконтроллер изнутри - это компьютер со своим вычислительным устройством, постоянной и динамической памятью, портами ввода-вывода и разной периферией. На рисунке 1.7 представлена архитектура AVR микроконтроллеров.

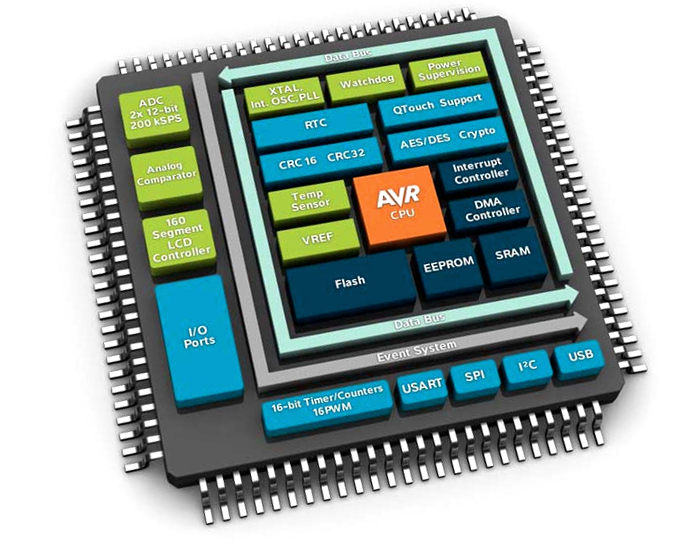


Рисунок 1.7 – Архитектура AVR микроконтроллеров

Внутри микроконтроллер содержит:

* Быстродействующий процессор с RISC-архитектурой.
* FLASH-память.
* EEPROM-память.
* Оперативную память RAM.
* Порты ввода/вывода.
* Периферийные и интерфейсные модули.

RISC - архитектура с тщательно подобранным набором команд, которые как правило выполняются за один такт работы процессора. Современные AVR микроконтроллеры содержат около 130 команд, которые очень быстро выполняются и не требуют больших затрат как по внутри-процессорным ресурсам, так и по потребляемой мощности.

На рисунке 1.8 представлена структурная схема микроконтроллеров AVR. На данном рисунке видно из каких блоков состоит микроконтроллер и как они связаны между собой.

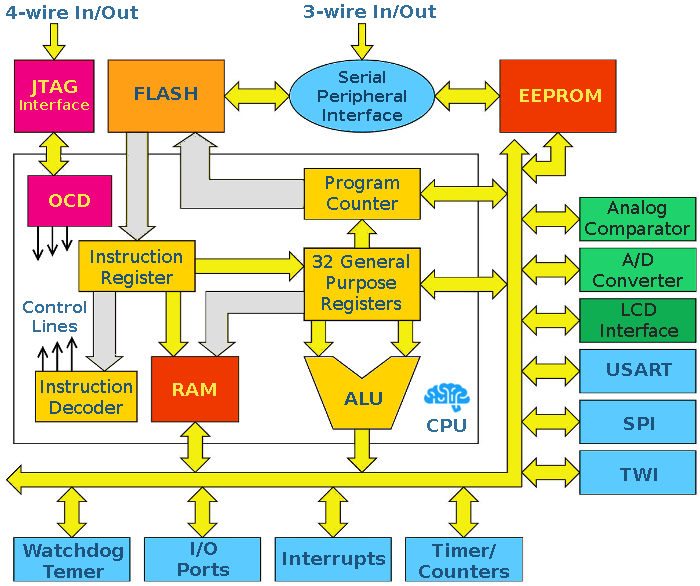


Рисунок 1.8 – Структурная схема микроконтроллеров AVR

Рассмотрим, что изображено на блоках в схеме:

* JTAG Interface - интерфейс внутрисхемной отладки (4 провода).
* FLASH - перепрограммируемая память для сохранения программы.
* SPI - последовательный периферийный интерфейс (3 провода).
* EEPROM - перепрограммируемое ПЗУ, энергонезависимая память.
* ЦПУ - центральный процессор управления, сердце микроконтроллера, 8-битное микропроцессорное ядро.
* АЛУ - арифметико-логическое устройство, основа блока CPU.
* RAM - оперативная память процессора.
* Program Counter -  счетчик команд.
* 32 General Purpose Registers - 32 регистра общего назначения.
* Instruction Register - регистр команд, инструкций.
* Instruction Decoder - декодер команд.
* OCD - блок внутренней отладки.
* Analog Comparator - аналоговый компаратор, блок сравнения аналоговых сигналов.
* A/D Converter - аналогово-цифровой преобразователь.
* LCD Interface - интерфейс для подключения жидко-кристаллического дисплея, индикатора.
* USART, UART - универсальный асинхронный приемопередатчик.
* TWI - последовательный интерфейс с двухпроводным подключением.
* Watchdog Timer - сторожевой или контрольный таймер.
* I/O Ports - порты вода/вывода.
* Interrupts - блок управления и реакции на прерывания.
* Timers/Counters - модули таймеров и счетчиков.

Также микроконтроллеры на данной архитектуре имеют достаточно невысокую стоимость и низкий порог вхождения.

Для реализации центрального координирующего устройства в сети был выбран микроконтроллер Arduino Mega 2560, т.к. он имеет достаточно большой вычислительный запас мощностей и высокую производительность. Микроконтроллер изображен на рисунке 1.9.

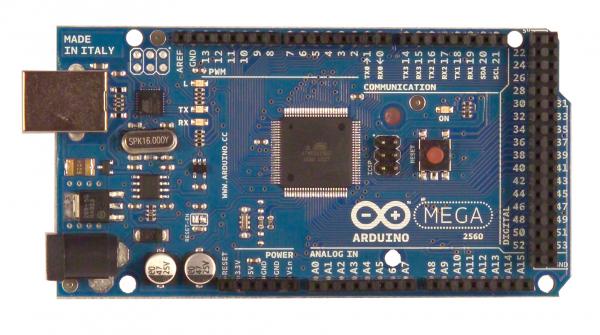


Рисунок 1.9 – Arduino Mega 2560

Arduino Mega построена на микроконтроллере ATmega2560. Плата имеет 54 цифровых входа/выходов (14 из которых могут использоваться как выходы ШИМ), 16 аналоговых входов,4 последовательных порта UART, кварцевый генератор 16 МГц, USB коннектор, разъем питания, разъем ICSP и кнопка перезагрузки [3].

Технические характеристики:

* Микроконтроллер: ATmega2560.
* Тактовая частота: 16 МГц.
* Напряжение логических уровней: 5 В.
* Входное напряжение питания: 7–12 В.
* Портов ввода-вывода общего назначения: 54.
* Максимальный ток с пина ввода-вывода: 40 мА.
* Максимальный выходной ток пина 3.3V: 50 мА.
* Максимальный выходной ток пина 5V: 800 мА.
* Портов с поддержкой ШИМ: 15.
* Портов, подключённых к АЦП: 16.
* Разрядность АЦП: 10 бит.
* Flash-память: 256 Кбайт.
* EEPROM-память: 4 Кбайт.
* Оперативная память: 8 Кбайт.
* Габариты: 101×53 мм.

В качестве оконечных устройств был выбран микроконтроллер Atmel atmega328p. Данное устройство имеет компактные размеры и низкое энергопотребление, что является главным критерием при выборе данного устройства. Микроконтроллер представлен на рисунке 1.10.

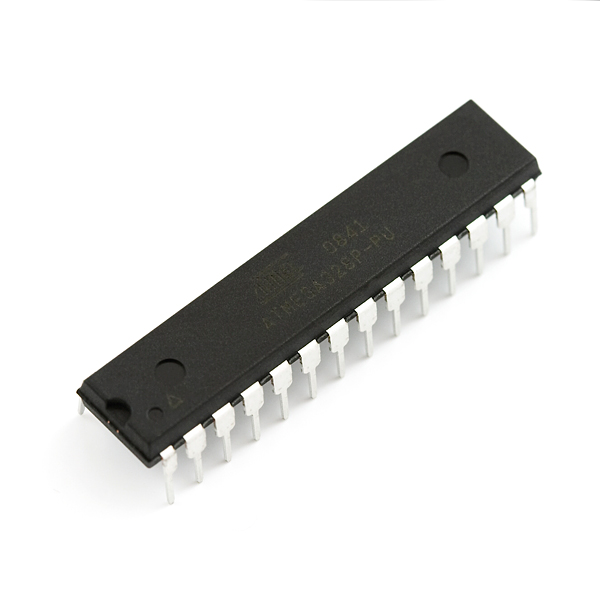


Рисунок 1.10 – Atmel Atmega328p

ATmega328P - микроконтроллер семейства AVR [3], как и все остальные имеет 8-битный процессор и позволяет выполнять большинство команд за один такт.

Технические характеристики:

* Тактовая частота: 0 – 20 МГц.
* Объём Flash-памяти: 32 Кбайт.
* Объём SRAM-памяти: 2 Кбайт.
* Объём EEPROM-памяти: 1 Кбайт.
* Напряжение питания: 1,8 – 5,5 В.
* Потребляемый ток в режиме работы: 0,2 мА (1 МГц, 1,8 В).
* Потребляемый ток в режиме сна: 0,75 мкА (1 МГц, 1,8 В).
* Количество таймеров/счётчиков: 2 восьмибитных, 1 шестнадцатибитный.
* Общее количество портов: 23.
* Количество ШИМ (PWM) выходов: 6.
* Количество каналов АЦП (аналоговые входы): 6.
* Количество аппаратных USART (Serial): 1.
* Количество аппаратных SPI: 1 Master/Slave.
* Количество аппаратных I²C/SPI: 1.
* Разрешение АЦП: 10 бит.

1.2.2 Приемники и передатчики. Для реализации приема и передачи информации в беспроводной сети были выбраны приемники и передатчики на базе радиоканала на частоте 433 МГц, т.к. данные модули обладают довольно низким энергопотреблением, а также имеют компактные размеры, и высокое быстродействие. Модули приема и передачи изображены на рисунке 1.11.



Рисунок 1.11 – Rf модули приема и передачи

Модули имеют всего 4 контакта: питание, земля, цифровой вход и антенна. Никакого протокола передачи не предусмотрено: модуль просто передают или принимают восходящие и нисходящие фронты отправленные модулем-передатчиком, поступающие на вывод «Data in». Это позволяет подключать к модулю напрямую даже такие простые источники сигнала, как кнопка или геркон.

Модулю не требуется время, чтобы запуститься или установить соединение. После подачи питания он сразу готов к работе.

Технические характеристики передатчика:

* Напряжение питания: 3–12 В.
* Несущая частота: 433 МГц.
* Максимальная пропускная способность: 8 Кбит/с.
* Потребляемый ток: 8 мА.
* Выходная мощность: 32 мВт.
* Диапазон рабочих температур: −20 − +80 °C.

Технические характеристики приемника:

* Напряжение питания: 5 В.
* Несущая частота: 433 МГц.
* Максимальная пропускная способность: 5 Кбит/с.
* Потребляемый ток: 4,5 мА.
* Чувствительность: −106 − −110 дБм.
* Диапазон рабочих температур: −20 - +80 °C.

1.2.3 Датчики. В качестве датчиков были выбраны датчики измерения температуры и влажности DHT11. Они также имеют компактные размеры и низкую стоимость, что в данном случае являлось приоритетным. Изображение датчика представлено на рисунке 1.12.

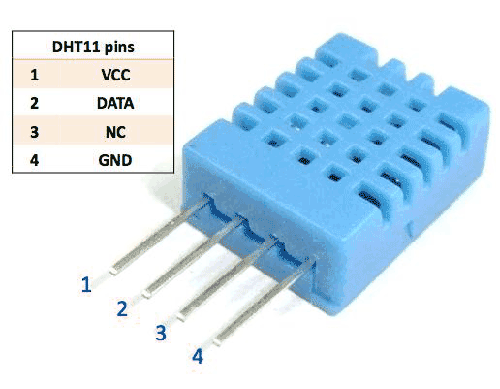


Рисунок 1.12 – Датчик влажности и температуры DHT11

Технические характеристики:

* Интерфейс: 1-проводной.
* Внешние компоненты: подтягивающий резистор на линии данных (в зависимости от длины кабеля).
* Питание: от 3.5 до 5.5 В.
* Измерение температуры:
* Допустимый диапазон: 0°C - 50° C.
* Максимальная погрешность: ±2°C.
* Разрешение шкалы: 1°C.
* Измерение относительной влажности:
* Допустимый диапазон: 20 - 95%.
* Минимальная погрешность: ±4%.
* Максимальная погрешность: ±5%.
* Разрешение шкалы: 1%.
* Минимальное время между считываниями показаний: 1 с.
* Расстояние между выводами: 2.54 мм.

1.2.3 Вспомогательные радиокомпоненты. Для улучшения сигнала беспроводной передачи данных были подобраны дополнительные антенны, представленные на рисунке 1.13.



1.13 – Антенны для усиления сигнала

Для того чтобы узлы сети питались от автономного источника питания, были подобраны модули питания со стабилизатором напряжения до 5 В YwRobot MB102, а также соединительные кабеля для подачи питания к данному модулю от элементов питания типоразмера «Крона». Полученная связка позволяет получить максимальное КПД по расходу тока, а также стабильное напряжение на выходе от 4,98 до 5 В.

Технические характеристики модуля питания:

* Входное постоянное напряжение: 6.5-12 В (DC).
* Выходные напряжения: 3.3 и 5 В.
* Дополнительный разъём для питания внешних устройств: 3.3 / 5 В.
* Максимальный суммарный ток нагрузки двух стабилизаторов: 0.7 А.
* Индикатор наличия питания.
* Выключатель напряжения питания.
* Размеры: 53 x 32 x 23 мм.

Данные устройства представлены на рисунках 1.14 и 1.15 соответственно.



Рисунок 1.14 – Модуль питания со стабилизатором напряжения



Рисунок 1.15 – Соединительный кабель «Крона» - модуль питания

## 1.3 Принципиальные схемы подключения

Перед разработкой программного обеспечения необходимо собрать из подобранных аппаратных компонентов конечные устройства. Для этого необходимо спроектировать принципиальные схемы подключения, согласно которым и будут собраны аппаратные компоненты в единое устройство.

Для проектирования схем был использован web сервис easyeda.com предназначенный для реализации схем.

Схемы подключения оконечных узлов и координатора сети изображены на рисунках 1.16 и 1.17 соответственно.

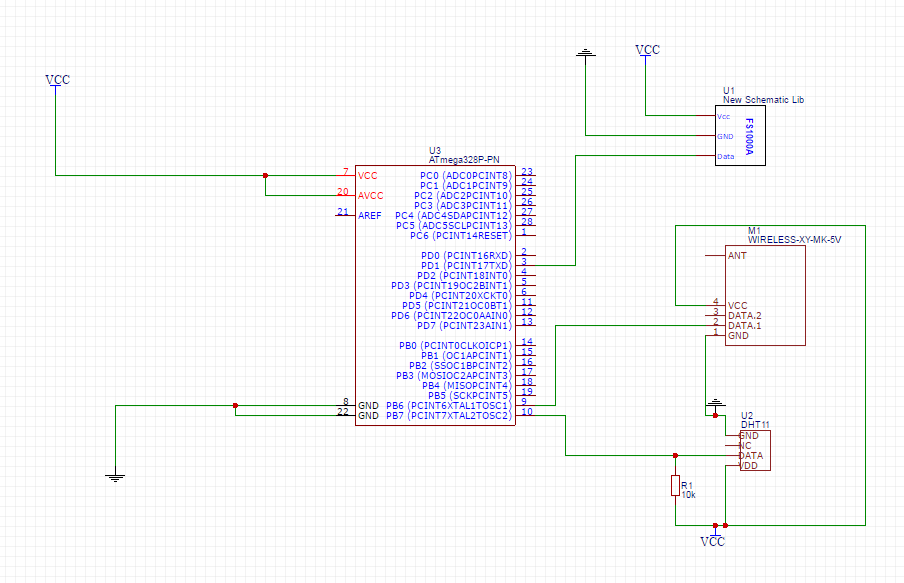


Рисунок 1.16 – Схема подключения оконечного узла

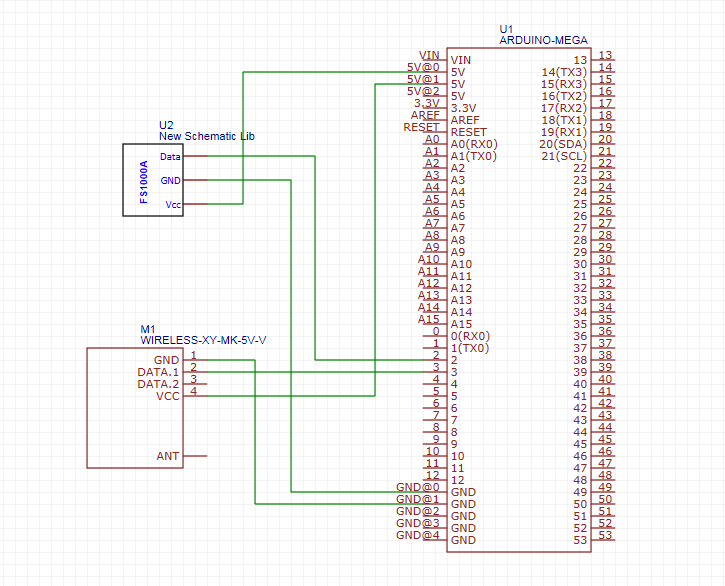


Рисунок 1.17 – Схема подключения оконечного координатора

## 1.4 Программные компоненты

После того как схемы были собраны необходимо было подобрать программные компоненты, с помощью которых можно реализовать систему.

1.4.1 Подбор IDE. Для программирования микроконтроллеров была выбрана среда программирования, которая подходит под следующие критерии выбора:

* интуитивно понятный интерфейс;
* стоимость приобретения IDE;
* простота программирования микроконтроллеров Atmel;
* наличие многих встроенных вспомогательных библиотек;
* кроссплатформенность.

Под все вышеизложенные требования идеально подходит среда программирования Arduino IDE.

Интерфейс среды разработки Arduino IDE [4] содержит следующие основные элементы: текстовый редактор для написания кода, область для вывода сообщений, текстовая консоль, панель инструментов с традиционными кнопками и главное меню. Данный софт позволяет компьютеру взаимодействовать с Arduino как для передачи данных, так и для прошивки кода в контроллер. Также с помощью данной среды разработки, существует возможность прошивать микроконтроллеры Atmel при установке соответствующего загрузчика в микроконтроллер. С новым загрузчиком микроконтроллеры Atmel становятся ничем не отличимыми от Arduino.

Интерфейс среды разработки представлен на рисунке 1.18.

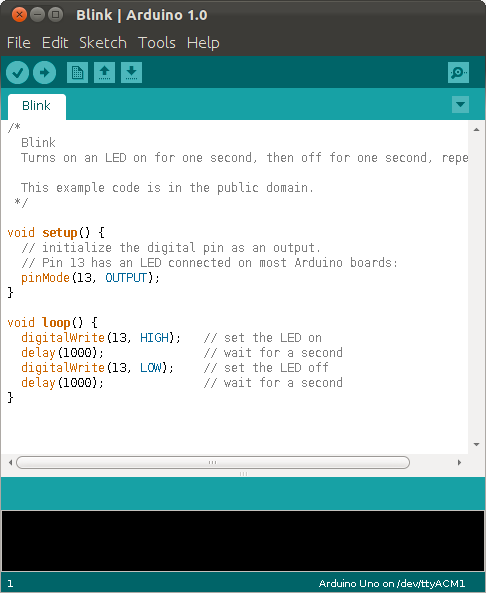


Рисунок 1.18 – Среда разработки Arduino IDE

Программы, создаваемые в среде разработки Ардуино, иногда еще называют скетчами. Скетчи пишутся в текстовом редакторе и сохраняются в файлах с расширением .ino. Встроенный текстовый редактор имеет стандартные инструменты копирования, вставки, поиска и замены текста. Область сообщений в окне программы является, своего рода, обратной связью для пользователя и информирует его о событиях (в том числе и об ошибках), возникающих в процессе записи или экспорта написанного кода. Консоль отображает в виде текста поток выходных данных среды Ардуино, включая все сообщения об ошибках и пр. генерируемую ею информацию. В нижнем правом углу окна программы показывается модель текущей платы и последовательный порт, к которому она подключена. Кнопки на панели инструментов предназначены для создания, открытия, сохранения и прошивки программ в устройство. Отдельная кнопка запускает программу SerialMonitor.

Для того чтобы появилась возможность программирования подобранных микроконтроллеров необходимо прошить в микроконтроллер Atmel Atmega 328p специальный загрузчик, делающий из данного микроконтроллера в конечном приближении Arduino UNO.

Загрузчик или бутлоадер это всего лишь небольшая программка, которая сидит в специальной области памяти микроконтроллера и слушает какой-либо интерфейс. Обычно это UART, но бывает и SPI, USB и даже SoftUSB загрузчики.

При загрузке контроллера управление первым делом передается бутлоадеру и он проверяет есть ли условие для запуска. Условие может быть любым, но обычно это либо наличие спец байта по интерфейсу, либо наличие нужного логического уровня на выбранной ножке контроллера, сигнализирующее о том, что мы хотим обратиться к буту прошивку. Если условие есть — то загрузчик может, например, принять прошивку по UART‘у и сам прошить ее во флэш память. Или, наоборот, считать прошивку из флэш памяти и выдать через UART, считать или записать EEPROM. Но обычно все же с помощью загрузчика осуществляют прошивку микроконтроллера без применения спец программатора.

Если разрешающего условия при старте нет, то бутлоадер завершает свою работу и передает управление основной программе.

1.4.2 Прошивка загрузчика в микроконтроллер Atmel Atmega 328p. В качестве программатора была использована плата Arduino Uno. Для того чтобы сделать из микроконтроллера программатор в микроконтроллер Arduino Uno был загружен специальный скетч из набора уже готовых скетчей под названием Arduino ISP.

После чего было произведено подключение микроконтроллера atmega 328p к микроконтроллеру Arduino Uno. Посредством макетной платы были соединены между собой контакты MISO (12), MOSI (11), SCK (13), SS (10) согласно схемам подключения и распиновки контроллеров представленным на рисунках 1.19, 1.20 и 1.21.

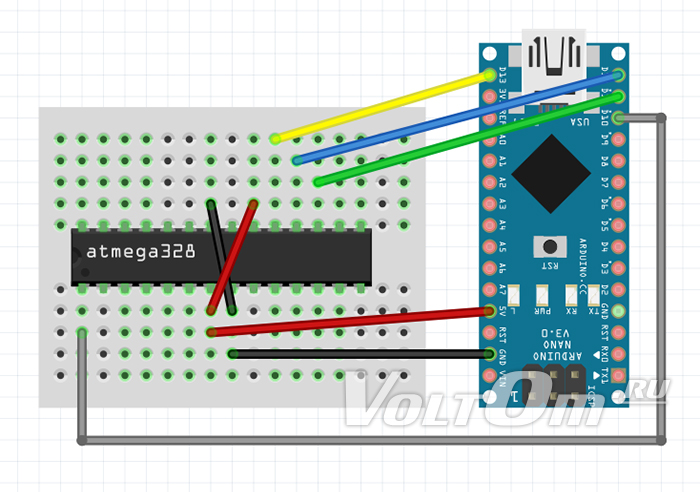


Рисунок 1.19 – Схема подключения микроконтроллеров между собой

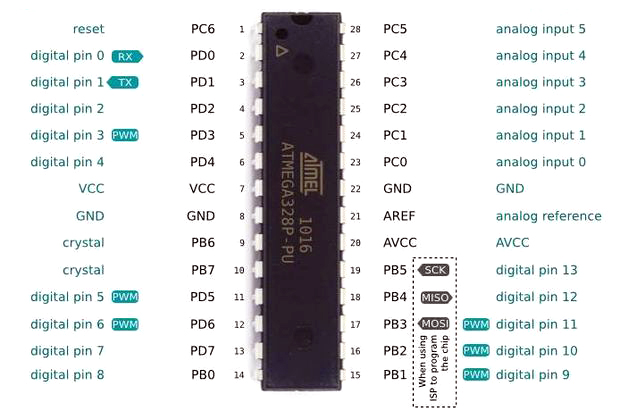


Рисунок 1.20 – Схема распиновки микроконтроллера atmega 328p

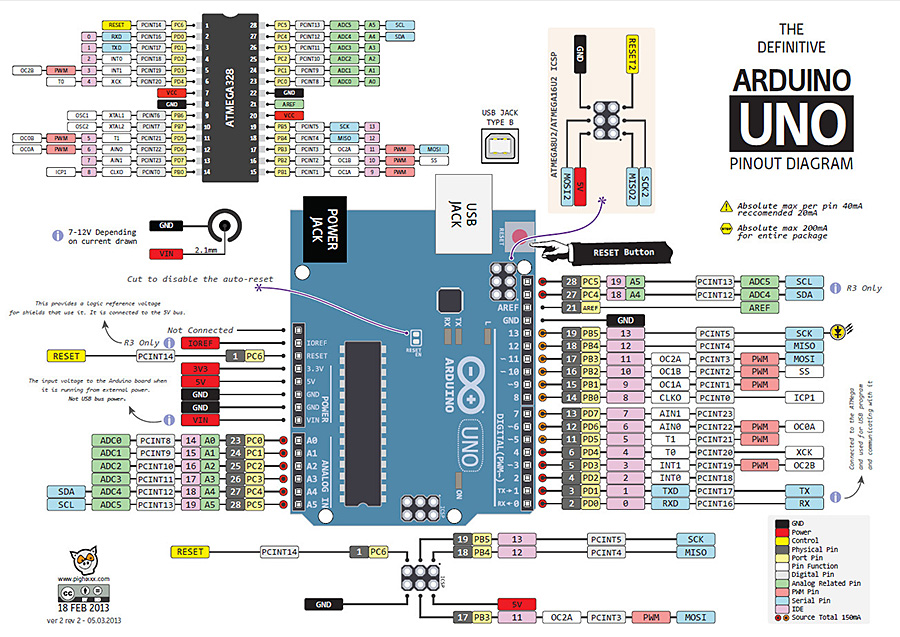


Рисунок 1.21 – Схема распиновки микроконтроллера Arduino UNO

Затем с помощью веб сервиса был собран подходящий для нашего микроконтроллера загрузчик, добавлен в папку с платами в интегрированной среде разработки. Следом были выставлены параметры, которые необходимые для записи, и была произведена непосредственная запись. Все измененные параметры представлены на рисунке 1.22.

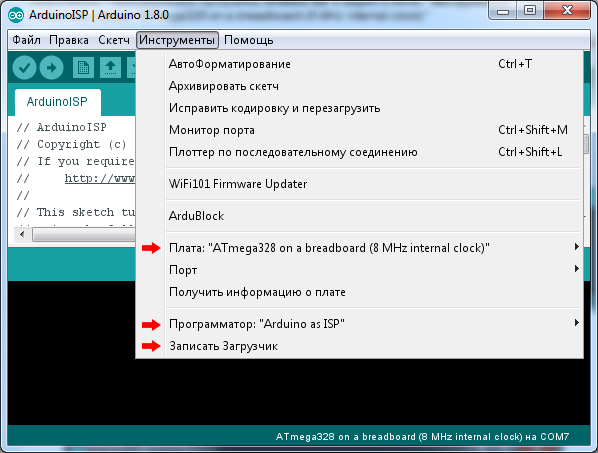


Рисунок 1.22 – Параметры записи загрузчика

Прошивка микроконтроллера осуществлялась по SPI интерфейсу. После успешной записи загрузчика микроконтроллер становится идентичен по программированию микроконтроллеру Arduino и не представляет сложности для программирования.

1.4.3 Разработка программных продуктов. Разработка программного обеспечения — это род деятельности и процесс, направленный на создание и поддержание работоспособности программного обеспечения, используя технологии и практики из информатики, управления проектами, математики, инженерии и других областей знания. Существует множество различных языков программирования, но самых лучших из них всего несколько. Язык программирования должен иметь богатые возможности, оставаясь при этом достаточно гибким. Его синтаксис должен быть лаконичным, но понятным. Он должен облегчить написание кода программ, не создавая дополнительных сложностей. Хороший язык программирования поддерживает современные перспективные решения. Именно таким языком программирования является C++ [5].

C++ — это компилируемый язык со статической типизацией на котором можно создавать программы любого уровня сложности.

С++ является языком программирования общего назначения. Именно этот язык хорошо известен своей эффективностью, экономичностью, и переносимостью.

Указанные преимущества С++ обеспечивают хорошее качество разработки почти любого вида программного продукта.

Использование С++ в качестве инструментального языка позволяет получать быстрые и компактные программы. Во многих случаях программы, написанные на языке С++, сравнимы по скорости с программами, написанными на языке ассемблера.

В соответствии с разработанным алгоритмом и на базе языка C++ было разработано 3 программных продукта:

* координатор сети;
* оконечное устройство сети;
* скрипт обработки полученных данных.

Координатор сети отвечает за координацию работы сети.

Оконечное устройство занимается сбором и предоставлением информации с датчиков.

Скрипт обработки данных помещает все полученные данные в базу для их дальнейшего использования. Скрипт был разработан на языке Python.

Для реализации программных продуктов были использованы дополнительные библиотеки.

* VirtualWire – библиотека, отвечающая за отправку и прием коротких сообщений по широковещательной беспроводной связи. Плюс данной библиотеки в том, что она отвечает за достоверность полученной информации, следит за контрольной суммой сообщения, а также отвечает за согласованность передачи между приемником и передатчиком.
* EasyTransferVirtualWire – выбрана для инкапсуляции в кадр VirtualWire своей собственной структуры данных передаваемой по сети. Также данная библиотека позволяет избежать несанкционированную передачу данных устройствам при наличии другой структуры передаваемых данных.
* EEPROM – необходима для работы с внутренней энергонезависимой памятью, в которой хранятся все параметры устройств в сети.
* AVR/sleep – использовалась для управления режимами сна оконечных устройств.
* AVR/power – управление периферией оконечных устройств.
* AVR/wdt – управление внутренним таймером для отслеживания зависания устройств, и для подсчета времени сна.
* AVR/io – управление входными/выходными портами.
* AVR/interrupt – управление флагами прерываний.

После того как программные продукты для устройств разработаны их можно с помощью программатора загружать в микроконтроллеры, собирать конечные схемы и использовать по конечному назначению.

Параметры:

* тип данных;
* время, через которое необходимо передавать данные;
* время, через которое координатор посылает запросы на сон;
* непосредственное время сна,

перед прошивкой в устройства необходимо корректировать в соответствии с предназначением, целевым размещением и предполагаемым количеством устройств в сети. Координатора сети можно перепрошивать во время работы сети, это не окажет влияния на сеть, оконечное устройство после перепрошивки будет перерегистрировано, и если оно является транзитным устройством в непосредственной близости к координатору, то вся ветвь будет перерегистрирована в течении некоторого времени.

Для работы программного продукта обрабатывающего полученную информацию и помещающего их в базу данных необходимо было подобрать СУБД.

1.4.4 Подбор СУБД. Система управления базами данных - это совокупность языковых и программных средств, которая осуществляет доступ к данным, позволяет их создавать, менять и удалять, обеспечивает безопасность данных и т.д. В общем СУБД - это система, позволяющая создавать базы данных и манипулировать сведениями из них. А осуществляет этот доступ к данным СУБД посредством специального языка - SQL.

При выборе у нас было несколько основных критериев:

* простота развертывания и обслуживания;
* кроссплатформенность;
* open source продукт;
* популярность;
* наличие api для современных языков программирования.

В качестве конечной системы управления базами данных была выбрана MySQL.

MySQL одна из самых популярных на данный момент СУБД [6], является открытой и распространяется по лицензии GPL. MySQL широко применяется в веб технологиях,  малых и средних приложениях. Является бесплатной альтернативой MS SQL и Oracle в решениях, когда от СУБД не требуется высокая производительность и отказоустойчивость, а на первое место выносится простота развертывания и обслуживания. MySQL является кроссплатформенной СУБД, следовательно она может работать как на \*nix подобных системах, так и на платформе Windows. MySQL можно установить как на серверной платформе Windows Server, так и на клиентских ОС, например Windows 8. MySQL имеет [API](https://ru.wikipedia.org/wiki/API) для языков [Delphi](https://ru.wikipedia.org/wiki/Delphi_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [C](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [C++](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_%D0%BF%D0%BB%D1%8E%D1%81_%D0%BF%D0%BB%D1%8E%D1%81), [Эйфель](https://ru.wikipedia.org/wiki/%D0%AD%D0%B9%D1%84%D0%B5%D0%BB%D1%8C_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [Java](https://ru.wikipedia.org/wiki/Java), [Лисп](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%81%D0%BF), [Perl](https://ru.wikipedia.org/wiki/Perl), [PHP](https://ru.wikipedia.org/wiki/PHP), [Python](https://ru.wikipedia.org/wiki/Python), [Ruby](https://ru.wikipedia.org/wiki/Ruby), [Smalltalk](https://ru.wikipedia.org/wiki/Smalltalk), [Компонентный Паскаль](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%BE%D0%BD%D0%B5%D0%BD%D1%82%D0%BD%D1%8B%D0%B9_%D0%9F%D0%B0%D1%81%D0%BA%D0%B0%D0%BB%D1%8C) и [Tcl](https://ru.wikipedia.org/wiki/Tcl" \o "Tcl), библиотеки для языков платформы [.NET](https://ru.wikipedia.org/wiki/.NET_Framework), а также обеспечивает поддержку для [ODBC](https://ru.wikipedia.org/wiki/ODBC) посредством ODBC-драйвера [MyODBC](https://ru.wikipedia.org/wiki/MyODBC" \o "MyODBC).

Установка экземпляра сервера БД не вызывает сложности. Его можно установить в составе сборки состоящей из СУБД, веб сервера, а также языка программирования. Данный программный продукт входит в состав таких сборок, как:

* WAMP.
* [AppServ](https://ru.wikipedia.org/w/index.php?title=AppServ&action=edit&redlink=1).
* LAMP.
* Денвер.
* XAMPP.

Также СУБД можно установить отдельно от сборки, скачав подходящий дистрибутив с официального сайта разработчиков данного программного продукта.

# 2 РЕЗУЛЬТАТЫ ПРОВЕДЕННЫХ ИСПЫТАНИЙ

Для того чтобы проверить работоспособность разработанной системы необходимо разработать методику проведения испытаний. Методика должна затрагивать каждую отдельную функцию как оконечных узлов, так и координатора, поэтому можно выделить следующие категории испытаний:

* регистрация новых узлов в сети;
* передача данных координатору;
* переадресация трафика через транзитные узлы;
* режим сна.

После выбора основных компонентов тестирования, перейдем к рассмотрению методики проведения испытаний

## 2.1 Методика проведения испытаний

2.1.1 Регистрация новых устройств. Для проверки регистрации новых узлов в сети необходимо прошить оконечное устройство готовым программным продуктом, чтобы сбросить все установочные флаги. После чего, при подключенном к компьютеру координаторе, необходимо включить только что прошитое устройство в сеть. На компьютере необходимо открыть монитор COM порта, к которому подключен координатор, и проследить за служебными сообщениями. В данном случае должно появиться сообщение с запросом на регистрацию, и затем через некоторый промежуток времени в случае успешной регистрации будут приходить данные считанные с датчика устройства. Иначе через определенный интервал времени устройство вновь оправит запрос на регистрацию, что мы увидим на мониторе COM порта координатора.

2.1.2 Передача данных координатору. После подключения зарегистрированного устройства в сеть к координатору, подключенному к компьютеру, должны приходить данные от устройства. В случае успешного приема на мониторе COM порта должны появиться служебные сообщения следующего вида.

«Message sent.

Received a message.

It's a data

Device ID - ID

Route - ROUTE

Last device ID - LAST\_ID

Type of data - TYPE

Data - DATA», где:

* ID – идентификатор устройства, которое отправляло данные.
* ROUTE – направление устройства.
* LAST\_ID – идентификатор последнего устройства, через которое прошли данные.
* TYPE – тип переданных данных.
* DATA – данные пришедшие от устройства.

Поля LAST\_ID и ID могут совпадать или нет.

2.1.3 Переадресация трафика через транзитные узлы. Необходимо подключить 2 устройства в сеть. При этом необходимо, чтобы одно из устройств было вне непосредственной досягаемости до координатора. При появлении на мониторе COM порта координатора данных необходимо проверить, чтобы данные в формате, указанном в предыдущем разделе приходили от обоих устройств, а также, чтобы у одного из устройств поля LAST\_ID и ID не совпадали, а принадлежали обоим включенным в сеть устройствам.

2.1.4 Режим сна. Для проверки режима сна необходимо перенастроить прошивку координатора с уменьшением времени до погружения в сон направлений. Время выставим равным 1 минуте. А также время, на которое устройства будут засыпать, выставим 1 минуту. После чего при включении устройств в сеть на мониторе порта мы с вами увидим служебное сообщение о погружении четного или не четного направления в сон. После чего от устройств с тем или иным направлением по четности перестанут приходить данные сроком на 1 минуту.

## 2.2 Проведение испытаний разработанного программного обеспечения

2.2.1 Регистрация новых устройств. Для проверки регистрации нового устройства, в него была прошита свежая прошивка. После чего был подключен к персональному компьютеру устройство, являющееся координатором сети, и в радиусе досягаемости было подключено новое устройство. В результате по прошествии некоторого времени мы увидели, что координатор получил запрос на регистрацию. Затем координатор выставил все необходимые параметры и отправил ответ с регистрационными данными. Через небольшой промежуток времени от устройства стали приходить данные. Результат представлен на рисунке 2.1.

В результате чего была протестирована регистрация устройств в сети, проблем с регистрацией у устройств не возникает. Регистрация при стабильном радиоканале происходит с первого полученного сообщения, что является хорошим показателем. Это означает, что как отправка сообщений оконечным устройством и обработка полученной информации координатором, так и отправка сообщений координатором с последующим приемом на стороне оконечного оборудования происходит успешно.

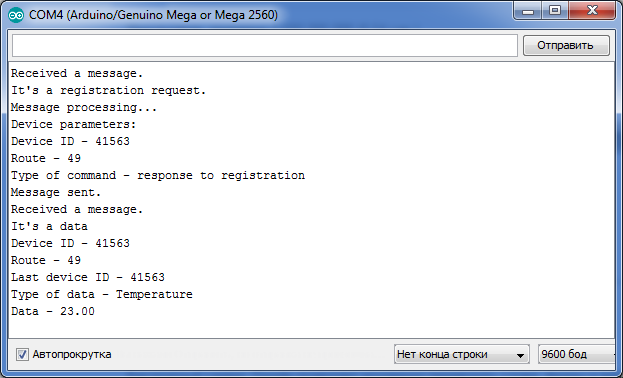


Рисунок 2.1 – Тестирование регистрации устройства

2.2.2 Передача данных координатору. Следующим этапом тестирования устройств была проверка корректности и стабильности передачи данных с устройств. Для этого мы взяли только что зарегистрированное устройство и подключили к сети, также мы подключили к компьютеру координатора, и зашли на монитор COM порта для отслеживания приходящих сообщений. В данном случае в прошивке оконечного устройства время передачи данных было выставлено на 15 секунд. По истечении времени чуть более 1 минуты было зафиксировано получение четырех сообщений от устройства. Результат данного этапа тестирования представлен на рисунке 2.2.

Все пакеты пришли от устройства зарегистрированного в данной сети. Потери пакетов не произошло, это обусловлено стабильностью радиоканала, и отсутствием помех.

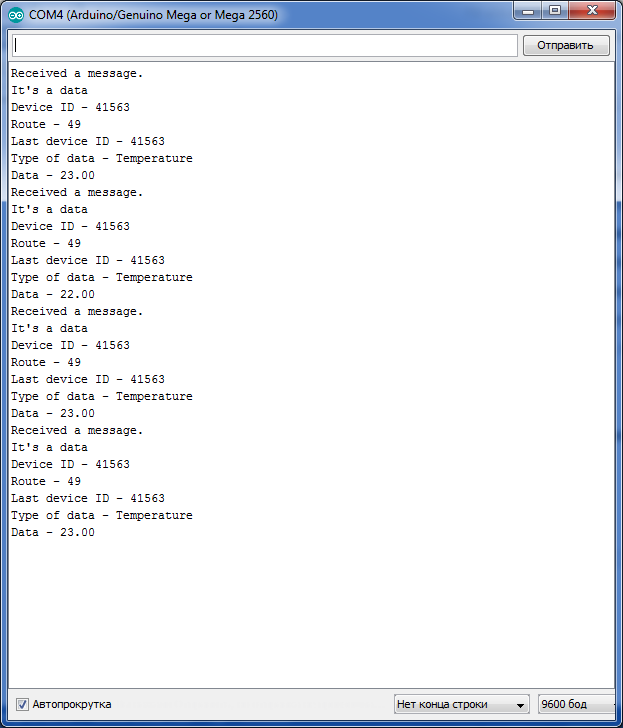


Рисунок 2.2 – Передача данных координатору

2.2.3 Переадресация трафика через транзитные узлы. Для проверки переадресации трафика через транзитный узел было взято второе оконечное устройство, в которое была прошита новая прошивка. После чего были подключены к питанию координатор сети, а также первое ранее прошитое устройство и поставлено в зону досягаемости координатора. Следом в сеть было включено устройство вне зоны непосредственной близости к координатору сети, но в близости к первому оконечному устройству. По истечении некоторого времени от последнего устройства через транзитный узел пришел запрос на регистрацию, а после чего стали приходить данные, при этом данные от первого устройства, также не переставали приходить. Результат данного этапа тестирования представлен на рисунке 2.3.

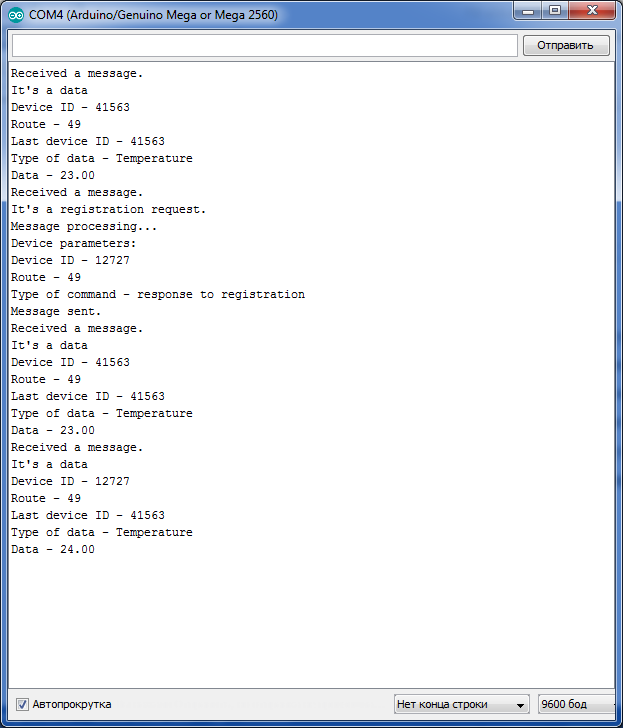


Рисунок 2.3 – Тестирование транзитного трафика

В результате тестирования данного шага было обнаружено, что при нестабильном электропитании приемников или передатчиков, ретрансляция данных может сильно ухудшиться. Поэтому нельзя использовать DC-DC конвертеры напряжения, т.к. выходное напряжение получается скачущим, а использовать стабилизаторы напряжения.

2.2.4 Режим сна. Для тестирования данного шага, был перепрошит координатор сети с выставлением временных интервалов равных 1 минуте и подключены к сети координатор и одно из устройств. После чего на мониторе COM порта были приняты данные от устройства. По прошествии одной минуты координатор сети отправил служебное сообщение указывающее всем устройствам нечетных направлений уснуть, а следовательно и нашему устройству. После чего данные от устройства стали приходить только после сна равного одной минуте. Результат данного этапа представлен на рисунке 2.4.

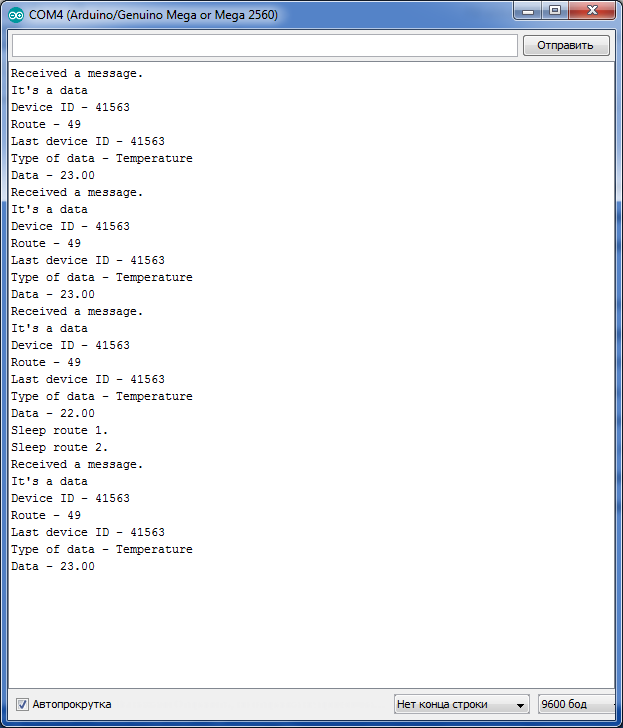


Рисунок 2.4 – Тестирование режима сна

По результатам тестирования было зафиксировано и отмечено, что в зависимости от количества направлений время погружения всех направлений в сон может достигать десятки секунд. Т.к. между каждой отправкой служебного сообщения необходимо выжидать более 100 мс, иначе сообщения не будут обработаны оконечным оборудованием. Таким образом при количестве направлений более 200, время отправки служебных сообщений будет более 10 секунд. Данный факт необходимо учитывать при проектировании временных интервалов на этапе прошивки устройства.

# ЗАКЛЮЧЕНИЕ

В результате проделанной работы были получены следующие результаты:

* разработан алгоритм работы беспроводной системы связи сети автономных датчиков;
* подобрано аппаратное обеспечение для реализации системы;
* подобрано программное обеспечение для реализации системы;
* разработан программный продукт, реализующий необходимую функциональность;
* проведено тестирование разработанного программного продукта с целью выявления и устранения недостатков.

Задачи, поставленные в начале проектирования, были достигнуты. Разработанное программное обеспечение функционирует согласно заявленным в начале проектирования требованиям.

# БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Интернет вещей. [Электронный ресурс] / Режим доступа: http://blogs.gartner.com/it-glossary/internet-of-things/. – (Дата обращения: 20.05.2017).
2. Эштон К. Этот «Интернет вещей». В реальном мире дело большее, чем идея [Текст] / Эштон К. / RFID: 2009. – 1-2 с.
3. Белов А.В. Микроконтроллеры AVR. От азов программирования до создания практических устройств / Белов А.В. – М: Наука и техника, 2016. – 544 с.
4. Блум Д. Изучаем Arduino. Инструменты и методы технического волшебства / Блум Д. – М: BHV, 2015. – 336 с.
5. Страуструп Б. Язык программирования C++ / Страуструп Б. – М: Невский диалект, 2008. —  1056 с.
6. Артеменко Ю.Н. - MySQL. Руководство администратора / Артеменко Ю.Н. – М: [Издательский дом «Вильямс»](https://ru.wikipedia.org/w/index.php?title=%D0%92%D0%B8%D0%BB%D1%8C%D1%8F%D0%BC%D1%81_(%D0%B8%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE)&action=edit&redlink=1), 2005. —  624 с.

# ПРИЛОЖЕНИЕ А

# Фрагменты кода координатора сети

#include <VirtualWire.h>

#include <EasyTransferVirtualWire.h>

#include <EEPROM.h>

// EEPROM 0-1 - my ID, 2 - routes, 3 - count devices, 4-... device Id(2b) + route\_device

// список команд:

// а - регистрация устройства

// х - возврат регистрационных данных к устройству

// d - данные

// s - спать

const int led\_pin = 13;

const int receive\_pin = 2;

const int transmit\_pin = 3;

unsigned int unique\_device\_id = 0;

unsigned int metric = 0;

unsigned int count = 0;

int con = 1;

int sch = EEPROM.read(2);

unsigned long time = millis();

unsigned int timer\_count = 0; // примитивный таймер

int id\_devices[50];

int device\_route[50];

int routes = 0;

boolean flag\_route\_sleep = true;

//create object

EasyTransferVirtualWire ET, ETOUT;

char buf[120];

struct SEND\_DATA\_STRUCTURE

{

//наша структура данных. она должна быть определена одинаково на приёмнике и передатчике

//кроме того, размер структуры не должен превышать 26 байт (ограничение VirtualWire)

unsigned int device\_id; // Адрес отправителя

unsigned int last\_device\_id; // Адрес получателя (если он изменился у координатора)

unsigned int destination\_id; // адрес получателя

unsigned int metric; // уровень

unsigned int route; // направление

char command; // системные флаги

float data; // данные

char data\_type; // тип данных 1 - температура, 2 - влажность, ...

int sleep\_counter; // сколько спать по 8 секунд

};

//переменная с данными нашей структуры

SEND\_DATA\_STRUCTURE mydata\_read, mydata\_out;

//ниже пару функций для записи данных типа unsigned int в EEPROM

void EEPROMWriteInt(int p\_address, unsigned int p\_value)

{

byte lowByte = ((p\_value >> 0) & 0xFF);

byte highByte = ((p\_value >> 8) & 0xFF);

EEPROM.write(p\_address, lowByte);

EEPROM.write(p\_address + 1, highByte);

}

unsigned int EEPROMReadInt(int p\_address)

{

byte lowByte = EEPROM.read(p\_address);

byte highByte = EEPROM.read(p\_address + 1);

return ((lowByte << 0) & 0xFF) + ((highByte << 8) & 0xFF00);

}

bool FindInTheArray(int array[], int id) // ищем в таблице устройств наш id

{

for(int i=0;i<count;i++)

if(array[i] == id)

{

return true;

}

return false;

}

int AddNewIdInArray(int id) // генерация пустого iD

{

int id\_dev = 0;

if(id>10000 || unique\_device\_id<60000) {

return id;

}

while(true)

{

id\_dev = random(10000, 60000);

if( FindInTheArray(id\_devices,id\_dev))

return id\_dev;

}

}

void setup()

{

pinMode(led\_pin, OUTPUT);

Serial.begin(9600); // Debugging only

ET.begin(details(mydata\_read));

ETOUT.begin(details(mydata\_out));

// Initialise the IO and ISR

vw\_set\_rx\_pin(receive\_pin);

vw\_set\_tx\_pin(transmit\_pin); //установка пина, к которому подключен data-вход передатчика

vw\_setup(2000); // скорость приема

vw\_rx\_start(); // запуск режима приема

Device ID

Serial.print("Getting Device ID... ");

EEPROMWriteInt(0, 0);

unique\_device\_id=EEPROMReadInt(0);

routes = EEPROM.read(2);

if(EEPROM.read(3) != 255)

{

count = EEPROM.read(3);

}

if(count !=0)

{

for(int i = 0; i<count; i++)

{

id\_devices[i] = EEPROMReadInt((i+1)\*3+1);

device\_route[i] = EEPROM.read((i+2)\*3);

}

}

if (unique\_device\_id<10000 || unique\_device\_id>60000) {

Serial.print("N/A, updating... ");

unique\_device\_id=random(10000, 60000);\*/

Serial.println(unique\_device\_id);

}

}

void loop()

{

unsigned long time = millis();

if((time/1800000) > timer\_count)

{

if(flag\_route\_sleep == true)

{

con = 1;

sch = EEPROM.read(2);

flag\_route\_sleep = false;

}

else

{

con = 2;

sch = EEPROM.read(2)/2;

flag\_route\_sleep = true;

}

Serial.print("sleep ");

Serial.println(con);

for(int i=0; i< sch; i++)

{

mydata\_out.route = i\*con;

mydata\_out.command = 's';

mydata\_out.sleep\_counter = 224;

ETOUT.sendData();

delay(150);

Serial.print(" SLEEP route ");

Serial.println(i);

}

timer\_count = time/1800000;

}

if(ET.receiveData()) // получили пакет данных, обрабатываем

{

Serial.println("I have data command ");

Serial.println(mydata\_read.command);

Serial.println(" device\_id is : ");

Serial.println(mydata\_read.device\_id);

Serial.println(" route is : ");

Serial.println(mydata\_read.route);

digitalWrite(led\_pin, HIGH);

mydata\_out = mydata\_read;

if(mydata\_read.command == 'a') // регистрация устройства

{

if(mydata\_read.route == 999)

{

Serial.println("Enter in route change");

mydata\_out.route = routes;

Serial.println(mydata\_out.route);

routes++;

EEPROM.write(2,routes);

}

if(FindInTheArray(id\_devices,mydata\_read.device\_id))

{

Serial.println("Enter in Find");

}

else

{

Serial.println("Enter in else Find");

mydata\_out.device\_id = AddNewIdInArray(mydata\_read.device\_id);

id\_devices[count] = mydata\_out.device\_id;

Serial.println(id\_devices[count]);

device\_route[count] = mydata\_out.route;

EEPROMWriteInt((EEPROM.read(3)+1)\*3+1, mydata\_out.device\_id);

EEPROM.write((EEPROM.read(3)+2)\*3,mydata\_out.route);

count++;

EEPROM.write(3, count);

}

mydata\_out.command = 'x';

mydata\_out.metric = 1;

mydata\_out.destination\_id = mydata\_out.device\_id;

mydata\_out.last\_device\_id = 0;

mydata\_out.device\_id = 0;

mydata\_out.data = 0;

delay(200);

ETOUT.sendData();

delay(200);

ETOUT.sendData();

delay(200);

ETOUT.sendData();

}

if(mydata\_read.command == 'd') // пакет с данными

{

Serial.print("id;");

Serial.print(mydata\_read.device\_id);

Serial.print(";d;");

Serial.print(mydata\_read.data);

Serial.print(";r;");

Serial.print(mydata\_read.route);

Serial.print(";lid;");

Serial.println(mydata\_read.last\_device\_id);

}

digitalWrite(led\_pin, LOW);

}

}

# ПРИЛОЖЕНИЕ Б

# Фрагменты кода оконечного устройства

// список команд:

// а - регистрация устройства

// х - возврат регистрационных данных к устройству

// d - данные

// s - спать

// EEPROM 0-1 - ID, 2-setue,3- sleep, 4 - metric ,5 - route, 6 - sleep-count 7 - counter

// Узел сети

#include <VirtualWire.h>

#include <EasyTransferVirtualWire.h>

#include <EEPROM.h> // эта библиотека нужна для работы с энерго-независимой памятью

#include <avr/sleep.h> //AVR MCU power management.

#include <avr/power.h> //AVR MCU peripheries (Analog comparator, ADC, USI, Timers/Counters etc) management.

#include <avr/wdt.h> //AVR MCU watchdog timer management.

#include <avr/io.h> //AVR MCU IO ports management.

#include <avr/interrupt.h> //AVR MCU interrupt flags management.

#include "DHT.h"

#define WDTCR |= \_BV(WDIE)

//dht DHT;

//#define DHT11\_PIN 7

#define DHTPIN 7 // пин подключения контакта DATA

#define DHTTYPE DHT11 // DHT 11

DHT dht(DHTPIN, DHTTYPE);

volatile byte watchdog\_counter;

const int led\_pin = 13;

const int transmit\_pin = 9;

const int receive\_pin = 3;

unsigned long time = millis();

unsigned long time2 = millis();

unsigned int unique\_device\_id = 0; // идентификатор устройства

unsigned int metric = 999; // метрика(уровень в сети)

unsigned int count = 0;

unsigned int route = 999; // направление движения

boolean setue = false; // флаг настроено(определено в сети - метрика порядковый номер и т.п.) устройство или нет

boolean transmit = false;

boolean sleep = false;

unsigned int timer\_count = 0; // примитивный таймер

unsigned int sleep\_timer = 0; // примитивный таймер

EasyTransferVirtualWire ET, ETOUT;

char buf[120];

void all\_pins\_output()

{

for (byte ATtiny\_pin; ATtiny\_pin < 5; ATtiny\_pin++)

{

pinMode(ATtiny\_pin, OUTPUT);

}

}

void all\_pins\_input()

{

for (byte ATtiny\_pin; ATtiny\_pin < 5; ATtiny\_pin++)

{

pinMode(ATtiny\_pin, INPUT);

}

}

struct SEND\_DATA\_STRUCTURE

{

//наша структура данных. она должна быть определена одинаково на приёмнике и передатчике

//кроме того, размер структуры не должен превышать 26 байт (ограничение VirtualWire)

unsigned int device\_id; // Адрес отправителя

unsigned int last\_device\_id; // Адрес получателя (если он изменился у координатора)

unsigned int destination\_id; // адрес получателя

unsigned int metric; // уровень

unsigned int route; // направление

char command; // системные флаги

float data; // данные

char data\_type; // тип данных 1 - температура, 2 - влажность, ...

int sleep\_counter; // сколько спать по 8 секунд

};

//переменная с данными нашей структуры

SEND\_DATA\_STRUCTURE mydata\_out,mydata\_bufer,mydata\_read;

//ниже пару функций для записи данных типа unsigned int в EEPROM

void EEPROMWriteInt(int p\_address, unsigned int p\_value)

{

byte lowByte = ((p\_value >> 0) & 0xFF);

byte highByte = ((p\_value >> 8) & 0xFF);

EEPROM.write(p\_address, lowByte);

EEPROM.write(p\_address + 1, highByte);

}

unsigned int EEPROMReadInt(int p\_address)

{

byte lowByte = EEPROM.read(p\_address);

byte highByte = EEPROM.read(p\_address + 1);

return ((lowByte << 0) & 0xFF) + ((highByte << 8) & 0xFF00);

}

void Sleeping(int counter)

{

count = EEPROM.read(7);

if (EEPROM.read(7) <= counter) //wait for watchdog counter reched the limit (WDTO\_8S \* 4 = 32sec.)

{

count++;

EEPROM.write(7,count);

ATtiny85\_sleep();

}

else{

EEPROM.write(3,0);

sleep\_timer = time/8000;

sleep = false;

count = 0;

EEPROM.write(6,count);

watchdog\_counter = 0; //reset watchdog\_counter

power\_all\_enable(); //enable all peripheries (timer0, timer1, Universal Serial Interface, ADC)

// wdt\_disable();

}

}

// функция регистрации

void Registering()

{

wdt\_reset();

unsigned long time = millis();

// Serial.println("enter in setue");

if(not transmit) // если еще ничего не передавал либо предыдущая попытка неудачна

{

Serial.println("Enter in register");

mydata\_out.device\_id = unique\_device\_id;

Serial.println("device\_id is : ");

Serial.println(mydata\_out.device\_id);

mydata\_out.destination\_id = 0; // всегда 0 т.к. мы отправляем координатору

mydata\_out.route = route;

mydata\_out.metric = metric;

mydata\_out.command = 'a';

mydata\_out.data = 0;

digitalWrite(led\_pin, HIGH); // включаем светодиод для отображения процесса передачи

Serial.print("Transmitting packet ");

Serial.print(" device id ");

Serial.print(mydata\_out.device\_id);

Serial.print(" data: ");

Serial.print(mydata\_out.data);

Serial.print(" ... ");

ETOUT.sendData(); // отправка данных

digitalWrite(led\_pin, LOW);

Serial.println("DONE");

delay(200);

transmit = true;

}

while(transmit == true)

{

wdt\_reset();

if(ET.receiveData())

{

if (mydata\_read.command == 'x')

{

metric = mydata\_read.metric;

route = mydata\_read.route;

unique\_device\_id = mydata\_read.destination\_id;

setue = true;

timer\_count = 0;

EEPROMWriteInt(0,unique\_device\_id);

EEPROM.write(2, 1);

EEPROM.write(4, metric);

EEPROM.write(5, route);

}

}

time = millis();

if((time/5000) > timer\_count)

{

transmit = false;

timer\_count = time/5000;

}

}

}

void setup()

{

// блок инициализации

pinMode(led\_pin, OUTPUT);

setup\_watchdog(WDTO\_8S); //approximately 8 sec. of sleep

ETOUT.begin(details(mydata\_out)); // инициализируем те данные которые будем передавать (привязываем mydata\_out к объекту ЕТOUT)

ET.begin(details(mydata\_read));

vw\_set\_rx\_pin(receive\_pin);

vw\_set\_tx\_pin(transmit\_pin); //установка пина, к которому подключен data-вход передатчика

vw\_setup(2000); //скорость передачи

Serial.begin(9600);

dht.begin();

randomSeed(analogRead(0)); // инициализируем генератор псевдослучайных чисел

vw\_rx\_start();

// Читаем/записываем Device ID

Serial.print("Getting Device ID... ");

if(EEPROM.read(2) == 1)

{

setue = true;

metric = EEPROM.read(4);

route = EEPROM.read(5);

}

unique\_device\_id=EEPROMReadInt(0);

count = EEPROM.read(6);

if (EEPROM.read(3) == 1)

{

sleep = true;

}

else {

sleep = false;

}

if (unique\_device\_id<10000 || unique\_device\_id>60000)

{

Serial.print("N/A, updating... ");

unique\_device\_id=random(10000, 60000);

EEPROMWriteInt(0, unique\_device\_id);

}

Serial.println(unique\_device\_id);

}

void loop()

{

time2 = millis();

if(time2 < 3960000)

{

if (EEPROM.read(3) == 1)

{

sleep = true;

}

else {

sleep = false;

}

if (sleep)

{

Sleeping(EEPROM.read(6));

}

else{

if(not setue)

{

Registering();

}

else

{

wdt\_reset();

if(ET.receiveData())

{

mydata\_out = mydata\_read;

if (mydata\_out.command == 'a')

{

if(mydata\_out.route == 999)

{

mydata\_out.route = route;

}

if(mydata\_out.route == route)

{

if(mydata\_read.metric > metric)

{

mydata\_out.device\_id = mydata\_read.device\_id;

mydata\_out.last\_device\_id = unique\_device\_id;

mydata\_out.destination\_id = mydata\_read.destination\_id;

mydata\_out.metric = metric;

mydata\_out.command = mydata\_read.command;

mydata\_out.data = mydata\_read.data;

ETOUT.sendData(); // отправка данных дальше

}

}

}

if (mydata\_read.command == 'x')

{

if(mydata\_read.route == route)

{

if(mydata\_read.metric <= metric)

{

mydata\_out.device\_id = mydata\_read.device\_id;

mydata\_out.last\_device\_id = unique\_device\_id;

mydata\_out.destination\_id = mydata\_read.destination\_id;

mydata\_out.metric = metric + 1;

mydata\_out.route = mydata\_read.route;

mydata\_out.command = mydata\_read.command;

mydata\_out.data = mydata\_read.data;

ETOUT.sendData(); // отправка данных дальше

}

}

}

if (mydata\_read.command == 'd')

{

if(mydata\_read.route == route)

{

if(mydata\_read.metric > metric)

{

mydata\_out.device\_id = mydata\_read.device\_id;

mydata\_out.last\_device\_id = unique\_device\_id;

mydata\_out.destination\_id = mydata\_read.destination\_id;

mydata\_out.metric = metric - 1;

mydata\_out.route = route;

mydata\_out.command = mydata\_read.command;

mydata\_out.data = mydata\_read.data;

ETOUT.sendData(); // отправка данных дальше

}

}

}

if (mydata\_read.command == 's')

{

if(mydata\_read.route == route)

{

mydata\_out = mydata\_read;

sleep = true;

EEPROM.write(6, mydata\_out.sleep\_counter);

EEPROM.write(3,1);

EEPROM.write(7,0);

ETOUT.sendData();

delay(200);

ETOUT.sendData();

delay(200);

/\* mydata\_out.command = 'd';

mydata\_out.data = mydata\_read.sleep\_counter;

ETOUT.sendData(); \*/ // отладка режима сна

}

}

}

else

{

time = millis();

if((time/15000) > timer\_count)

{

float h = dht.readHumidity();

float t = dht.readTemperature();

mydata\_out.device\_id = unique\_device\_id;

mydata\_out.last\_device\_id = unique\_device\_id;

mydata\_out.destination\_id = 0;

mydata\_out.metric = metric;

mydata\_out.route = route;

mydata\_out.command = 'd'; // т.е. передача данных

mydata\_out.data\_type = '1'; // передача температуры

mydata\_out.data = t;//sensor.tem;

timer\_count = time/15000;

ETOUT.sendData();

}

}

}

}

}

else

{

setue = false;

route = 999;

metric = 999;

EEPROM.write(2, 0);

EEPROM.write(4, 999);

EEPROM.write(5, 999);

}

}

void ATtiny85\_sleep()

{

power\_all\_disable(); //disable all peripheries (timer0, timer1, Universal Serial Interface, ADC)

power\_adc\_disable(); //disable ADC

power\_timer0\_disable(); //disable Timer0

power\_timer1\_disable(); //disable Timer2

power\_usi\_disable(); //disable the Universal Serial Interface module.

set\_sleep\_mode(SLEEP\_MODE\_PWR\_DOWN); //set the sleep type

sleep\_mode(); /\*system stops & sleeps here (automatically sets the SE (Sleep Enable) bit

(so the sleep is possible), goes to sleep, wakes-up from sleep after an

interrupt (if interrupts are enabled) or WDT timed out (if enabled) and

clears the SE (Sleep Enable) bit afterwards).

the sketch will continue from this point after interrupt or WDT timed out

\*/

}

void setup\_watchdog(byte sleep\_time)

{

wdt\_enable(sleep\_time);

}

ISR(WDT\_vect)

{

// WDTCR |= \_BV(WDIE);

watchdog\_counter++;

}

Serial.print("mydata\_read.last\_device\_id is : ");

Serial.println(mydata\_read.last\_device\_id);

Serial.print("unique\_device\_id : ");

Serial.println(unique\_device\_id);

Serial.print("command : ");

Serial.println(mydata\_read.command);

Serial.print("device\_id : ");

Serial.println(mydata\_read.device\_id);

Serial.print("destination\_id : ");

Serial.println(mydata\_read.destination\_id);

Serial.print("metric : ");

Serial.println(mydata\_read.metric);

# ПРИЛОЖЕНИЕ В

# Фрагменты кода модуля обработки принимаемой информации

import serial

import mysql.connector

ser = serial.Serial(“/dev/ttyUSB0”,9600)

print (“Connecting to Database”)

conn = mysql.connector.connect(host=localhost,database=’testdb’,user=’test’,password = ‘testpassword’)

cursor = conn.cursor()

while 1:

data = ser.readline()

my\_array = data.split(‘;’)

cursor.callproc(‘sp\_SaveParameters’, (my\_array[1],my\_array[3]))

conn.commit()