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

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

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

РАЗРАБОТКА СИСТЕМЫ МОНИТОРИНГА ЗАНЯТОСТИ ПАРКОВОЧНЫХ МЕСТ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

Руководитель работы,
к.т.н., доцент каф. ЭВМ
_____П.О. Шабуров
«___»_____2021 г.

Автор работы,
студент группы КЭ-405
_____В.О. Тюменцев
«___»_____2021 г.

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

Челябинск-2021

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

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

ЗАДАНИЕ

на выпускную квалификационную работу бакалавра

студенту группы КЭ-405
Тюменцеву Вячеславу Олеговичу
обучающемуся по направлению
09.03.01 «Информатика и вычислительная техника»

1. **Тема работы:** «Разработка системы мониторинга занятости парковочных мест» утверждена приказом по университету от 26 апреля 2021 г. №714-13/12 (приложение №73).

2. **Срок сдачи студентом законченной работы:** 1 июня 2021 г.

3. **Исходные данные к работе:**

Обеспечить основной функционал устройства:

– определение занятости парковочного места;
– вывод данных о занятости парковочных мест на информационное табло;
– реализация API для внешних информационных систем.

4. **Перечень подлежащих разработке вопросов:**

– анализ аналогов разрабатываемого устройства;

– детализация требований к системе в целом и к её отдельным подсистемам;
– проектирование архитектуры;
– выбор датчиков для устройства определения занятости парковочного места;
– выбор микроконтроллера для устройства определения занятости парковочного места;
– выбор микроконтроллера для передающего устройства;
– выбор дисплея для информационного табло;
– разработка модели системы;
– испытание разработанной системы.

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

Руководитель работы _____/*П.О. Шабуров*/

Студент _____/*В.О. Тюменцев*/

КАЛЕНДАРНЫЙ ПЛАН

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

Руководитель работы _____/*П.О. Шабуров*/

Студент _____/*В.О. Тюменцев*/

Аннотация

В.О. Тюменцев. Разработка конечного устройства для мониторинга занятости парковочных мест. – Челябинск: ФГАОУ ВО «ЮУрГУ (НИУ)», ВШЭКН; 2021, 99 с., 52 ил., библиогр. список – 22 наим.

В рамках выпускной квалификационной работы производится исследование современных решений по мониторингу занятости парковочных мест. Анализируются доступные технологические решения и выбираются подходящие по определенным критериям. Организуется проектирование системы мониторинга и прорабатываются отдельные подсистемы. В итоге создается модель всего комплекса на макетных платах, программируется и тестируется.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Большое количество личных автомобилей на дорогах и ограниченное число стояночных мест в городской среде влечет за собой проблему поиска свободных парковочных зон для владельцев транспортных средств. Часто возникают ситуации, когда водителю требуется найти место для парковки своего ТС и он не может визуально точно и быстро определить, куда ему следует направиться. В таком случае он сталкивается с задачей по поиску свободных мест. Это влечет за собой несколько проблем. Во-первых, водителю требуется тратить неопределённое личное время на поиск свободной позиции, иногда на это может уйти лишь несколько секунд, но также это может растянуться на несколько минут. Во-вторых, пока водитель ищет свободное место, он находится в движении и его внимание в такой момент становиться более рассеянным. Для других участников движения транспортное средство такого водителя может стать помехой, потому что ему нужно замедлять своё движение, резко останавливаться или неожиданно поворачивать, что создает опасность для него и для всех окружающих. В-третьих, пока водитель занимается поиском в транспорте с двигателем внутреннего сгорания, он расходует топливо, а в атмосферу выбрасываются вредные вещества. Это влечет за собой повышение расхода на топливо и экологические последствия для окружающей среды. Из-за этих причин возникает потребность в автоматизации процесса.

Для помощи водителям в поиске свободных стояночных зон предлагается разработать автоматическую систему мониторинга занятости парковочных мест. Разработка должна информировать пользователей о доступности свободных мест и их количестве с помощью информационного табло, установленного при въезде на стоянку и при помощи API для приложений.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

На текущей момент уже существуют различные готовые решения по мониторингу занятости парковочных мест, и их можно разделить на 2 основных типа:

- мониторинг с помощью камер видеонаблюдения;
- мониторинг с помощью датчиков.

В каждом подходе для реализации системы есть свои достоинства и недостатки.

Основным принципом при мониторинге занятости парковок с помощью камер видеонаблюдения является применение алгоритмов по распознаванию окружения и объектов с изображения стояночной зоны. При таком подходе для охвата большого количества мест на парковке может потребоваться лишь одна видеокамера, но при этом нужны и более сложные алгоритмы и серьезные вычислительные мощности для мониторинга в режиме реального времени.

В случае использования датчиков для мониторинга стояночной зоны, требуется внедрение устройств по определению занятости парковки на каждое место под транспортное средство. Такой способ организации отслеживания позволяет вести безошибочный и быстрый мониторинг занятости при любых масштабах, но требует большие материальные вложения для внедрения системы в городскую среду.

Ознакомимся с существующими аналогами и определим их достоинства и недостатки в их сравнении по их возможностям и характеристикам.

1.1. ОБЗОР АНАЛОГОВ

1.1.1. Умные парковки от компании Интерсвязь

Компания Интерсвязь в Челябинске использует и внедряет новые камеры видеонаблюдения по всему городу для использования в работе системы мониторинга «умные парковки». Отслеживать состояние мест возможно только в официальных приложениях для смартфонов на ОС iOS и Android [1]. Других способов для получения актуальной информации, кроме использования приложения, больше нет, а значит нет возможности для применения этих данных в индикации на парковочных зонах для уведомления водителей.

В разделе Умные парковки отображается интерактивная карта, на которой размещены указатели на местах стояночных зон города. О загруженности парковки информирует цвет, в который окрашен данный указатель, и показанное на нём число свободных позиций для размещения на стоянке автомобилей.

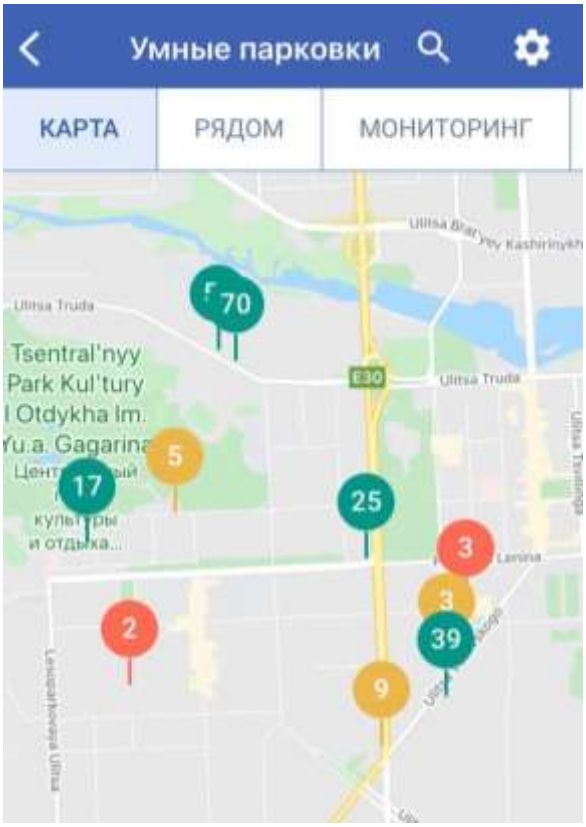


Рисунок 1.1.1 – Интерфейс раздела Умные парковки (карта)

Если выбрать один из указателей, то откроется видео с камеры и некоторые встроенные функции «управления парковкой» для пользователей, позволяющие получать уведомления о освобожденных местах и определять подозрительную активность около выбранного автомобиля.

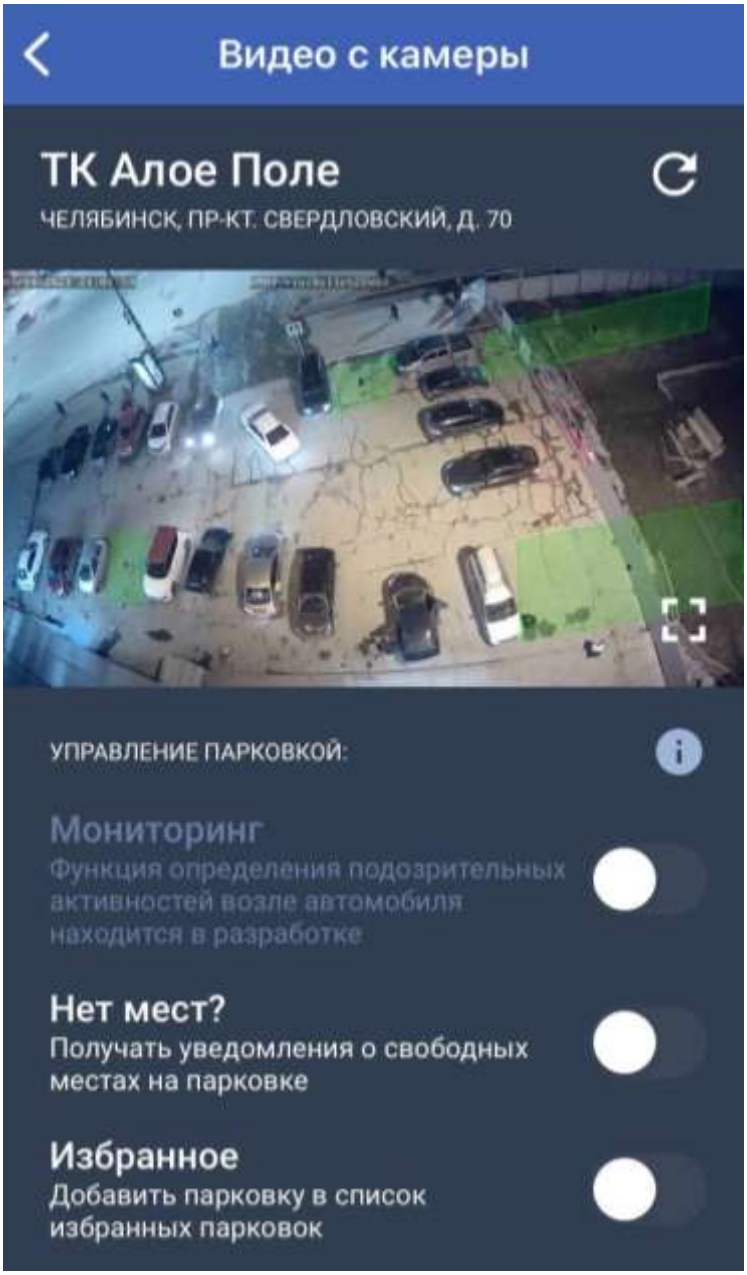


Рисунок 1.1.2 – Видео с камеры в приложении Интерсвязь

На изображение с видеокамеры накладываются прямоугольники – предполагаемые места, в которые по расчетам алгоритмов могут быть припаркованы автомобили. Также в пределе этой области выводится количество доступных возможных мест.

Алгоритмы системы работают не всегда корректно. Иногда они подсвечивают непригодные для парковки места или даже обозначают свободными те области, которые на самом деле заняты транспортом. Не всегда отображаются все доступные места, а в пределах одной области может быть подсчитано неверное количество доступных позиций для стоянки автомобилей.

Рисунок 1.1.3 – Камера на улице Тернопольская (г. Челябинск)

Из рисунка 1.1.3 видно, что правильно было определено только одно свободное место. В двух областях подсчитано неверное количество свободных точек для парковки. А в правой части изображения не были определены свободными целых 2 места.

В данном случае проблема заключается в расположении камеры очень близко к стоянке и под углом, что затрудняет обзор на дальние места.

Есть и другая особенность в мониторинге мест у такого алгоритма: все доступные области для парковки задается вручную, а значит при изменении направления камеры расчет свободных мест может стать полностью неверным.

Рисунок 1.1.4 – Изображение стоянки в ночное время суток (ул. Петра Столыпина, 11, г. Челябинск)

Рисунок 1.1.5 – Изображение стоянки в дневное время суток (ул. Петра Столыпина, 11, г. Челябинск)

Из рисунков 1.1.4 и 1.1.5 видно, что в любое время суток алгоритмы высчитывают и отображают некорректные позиции, которые доступны для парковки.

Также следует обратить внимание на то, что камеры наблюдения работают не в режиме реального времени. Задержка между съемкой

изображения и получением его на экране составляет около одной минуты. А частота обновления составляет 2 кадра в минуту. Это можно увидеть по временной отметке расположенной в верхнем левом углу на картинке видеотрансляции.

1.1.2. Sensit by Nedap [2]

Sensit – это платформа для беспроводного мониторинга парковок транспортных средств. В свою основу она включает устройство для определения занятости транспортных средств, которые представлены в 4-х разных видах и исполнениях, и программное обеспечение для управления этими устройствами и получения состояния занятости со стояночных мест.

Также дополнительно есть возможность подключать информационное табло и использовать приложение официальное приложение для смартфонов от производителя.

Основные достоинства:

- удобная платформа для объединения всех устройств в одной сети интернета вещей;
- работа по стандарту NB-IoT (Narrow Band Internet of Things);
- низкое энергопотребление (от 5 до 10 лет работы от одного заряда).

Недостатки:

- недоступность на российском рынке;
- неразборный корпус, батарею сможет заменить только производитель;
- высокая стоимость компонентов (от 20 тыс. руб. за 1 парковочный датчик) и обслуживания.

1.1.3. Parking Lot Sensor от компании Bosch [3]

Parking Lot Sensor – это беспроводной датчик для определения занятости парковочного места. Устройство задумано производителем для работы в его собственной сети цифрового автопарка Bosch.IO.

Основные достоинства:

– Низкое энергопотребление (до 5 лет работы от одного заряда);

Недостатки:

– Неподходящий рабочий температурный диапазон (-20…+65 °C);
– Недоступность на российском рынке;
– Излишне сложная платформа в рамках реализации мониторинга занятости парковок;
– Отсутствие готовой реализации информационного табло.

1.2. АНАЛИЗ ОСНОВНЫХ ТЕХНОЛОГИЧЕСКИХ РЕШЕНИЙ

1.2.1. Обзор бесконтактных датчиков

Основой системы для мониторинга занятости парковочного места являются датчики, определяющие присутствие и отсутствие транспортного средства в оборудованной зоне. Для обнаружения объектов можно использовать датчики присутствия объектов и датчики приближения. Датчик присутствия лишь определяет, находится ли какой-либо объект внутри заданных границ, а датчики приближения имеют возможность определения расстояния до объекта.

1.2.1.1. Датчики присутствия

Рассмотрим 2 основных типа датчиков присутствия – это оптические и магнитные [4, с. 14].

Оптические датчики подразделяются на работающие на просвет и отражательные.

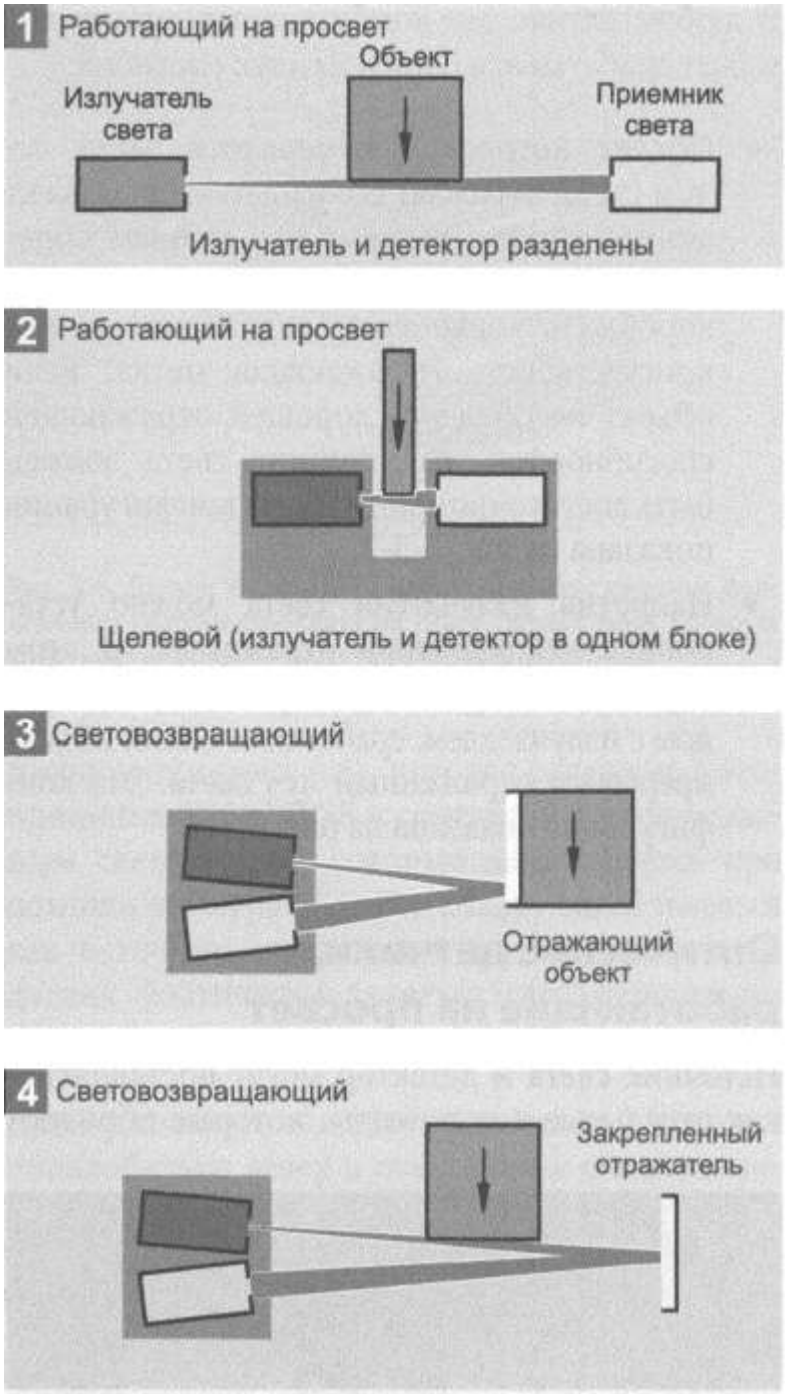


Рисунок 1.2.1 – Варианты оптических датчиков присутствия [4, с. 15]

Можно сразу исключить оптический датчик, работающий на просвет, поскольку его компоненты, излучатель и приемник должны быть разделены на

две части (рисунок 1.2.1) и их нужно расположить таким образом, чтобы между ними помещалось транспортное средство. Такой подход усложнит систему мониторинга из-за разделения одного устройства определения занятости на 2 части.

Световозвращающий датчик подходит для реализации устройства для мониторинга в одном корпусе, т. к. излучатель и приемник должны быть размещены в одном корпусе как единый компонент. Однако, в общем для всех оптических датчиков, существует серьезные недостатки:

– должна быть обеспечена хорошая видимость объекта, а на это могут повлиять погодные условия;
– датчик может случайно срабатывать от внешней засветки.

Магнитные датчики подразделяются на герконовые переключатели и датчики холла.

«Герконовый переключатель (геркон) – это механический переключатель, приводимый в действие магнитным полем. Он состоит из двух металлических контактов, помещенных внутри небольшого корпуса, который обычно представляет собой стеклянную капсулу. Контакты являются магнитными и способны перемещаться под действием магнитного поля. Для активации переключателя служит постоянный магнит.» [4, с. 19]

Хорошим достоинством геркона является простота в его применении, однако он совсем не подходит для использования в мониторинге за транспортным средством, поскольку напряженность магнитного поля автомобиля слишком незначительна для срабатывания датчика. К тому же, хрупкость датчика не позволяет использовать его в условиях сильных вибраций и ударных нагрузок.

В отличие от геркона, датчик Холла имеет непрерывную характеристику сигнала в зависимости от величины напряженности внешнего магнитного поля.

Использовать непрерывность можно для точной оценки обстановки и выявления присутствия ТС в обозначенной зоне.

Недостатками датчика Холла являются:

– влияние на точность окружающих магнитных полей;
– сложность в подстройке датчика под разные модели автомобилей, с разными электромагнитными характеристиками.

1.2.1.2. Датчики приближения

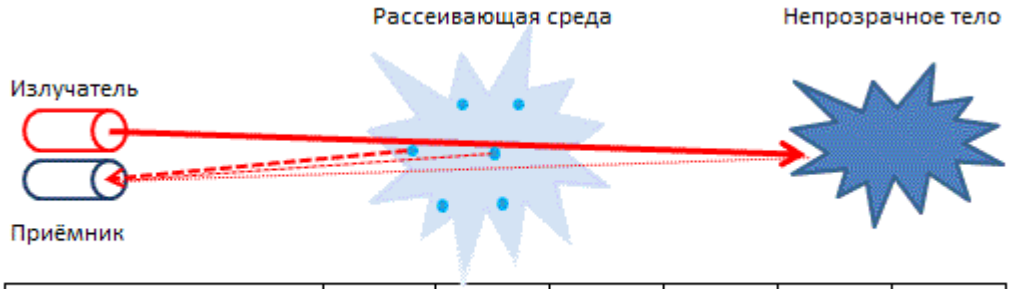
Существует несколько различных типов датчиков приближения. Основные среди них – это индуктивные, ёмкостные, лазерные и ультразвуковые.

Индуктивный датчик. «Принцип действия индуктивного датчика основан на изменении индуктивности обмотки на магнитопроводе в зависимости от положения отдельных элементов магнитопровода (якоря, сердечника и др.). В таких датчиках линейное или угловое перемещение преобразуется в изменение индуктивности датчика. […] Индуктивный датчик распознает и соответственно реагирует на все токопроводящие предметы.» [5]

Такой способ определенно не подходит для мониторинга присутствия ТС, поскольку внешние части кузова современного автомобиля изготавливаются из пластика и при приближении к индукционному датчику никак не могут быть обнаружены.

Емкостной датчик. «Емкостной датчик приближения измеряет расстояние до объекта, который должен обладать электрической проводимостью. […]. Типичный максимальный радиус действия составляет 10 мм. Для больших расстояний лучше применять оптические или ультразвуковые датчики.» [4, с. 43]

Лазерный дальномер – это датчик, определяющий расстояние с помощью измерения времени, которое проходит с момента излучения лазером излучения до получения этого сигнала приемником. Этому способствует способность электромагнитного излучения распространяться с постоянной скоростью.



Расстояние до цели	1 м	10 м	100 м	1 км	10 км	100 км
Время отклика	6.7 нс	67 нс	0.67 мкс	6.7 мкс	67 мкс	0.67 мс

Рисунок 1.2.2 – Принцип действия лазерного дальномера [6]

В качестве примера лазерного датчика расстояния можно рассмотреть доступное решение – модуль VL53L0X (GY-530). Диапазон измерения расстояния составляет до 2-х метров. Диапазон рабочих температур: от -20 до +70 °C. Стоимость составляет 397 рублей [7]. Достоинством данного модуля является его небольшая стоимость и небольшой размер (4,4x2,4x1,0 mm), что позволяет спроектировать устройство определения занятости максимально компактным. К недостаткам можно отнести небольшой диапазон измерения расстояния и неподходящий под зимние уличные условия диапазон рабочих температур.

Ультразвуковой датчик. «Принцип работы ультразвуковых датчиков основан на измерении времени между посылкой ультразвукового импульса и регистрацией отражённого импульса. Диапазон измерений – от нескольких миллиметров до нескольких метров. Точность измерения – 1 мм.» [8]

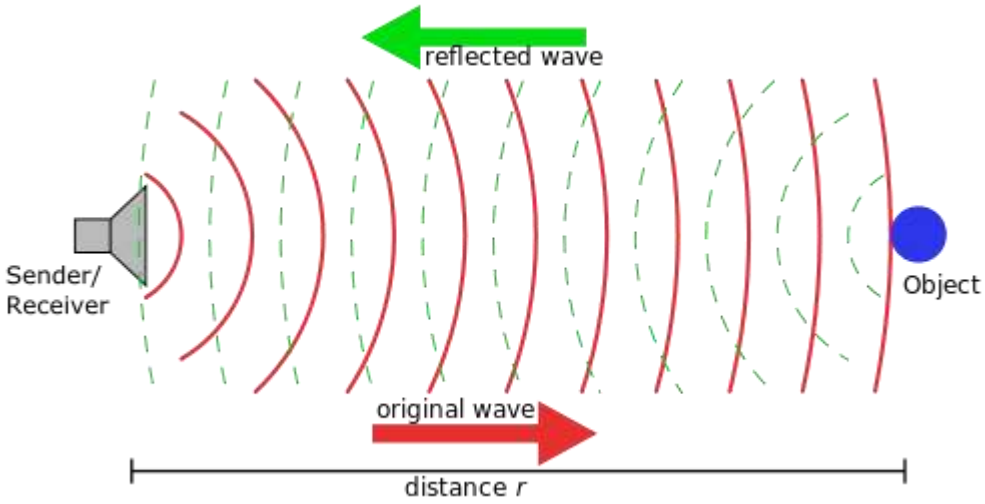


Рисунок 1.2.3 – Принцип действия ультразвукового дальномера [9]

На основе ультразвуковых датчиков изготавливаются парковочные радары (парктроники), которые используются во многих современных автомобилях. Такая система облегчает маневрирование при парковке, предупреждая водителей о приближении к препятствию в слепой зоне. Поэтому можно сделать вывод, что ультразвуковые датчики хорошо зарекомендовали себя в транспортной инфраструктуре и хорошо подходят для реализации определения занятости парковочного места. Принцип работы датчиков остается тот же. Разница при реализации лишь в том, что сенсоры устанавливаются в зоне парковки, в а качестве объекта для определения выступая транспортное средство.

Самым распространённым решением ультразвукового датчика является модуль HC-SR04 [4, с. 38]. Его достоинствами являются низкая стоимость в 148 рублей [10]. Диапазон измерения расстояния от 2 см до 4 м, что хорошо подходит для организации парковочного места по стандарту размером 5,3х2,5 м [11].

К недостаткам можно отнести нижний температурный предел в -20 °С и слишком большой размер модуля (43х20х15мм).

1.2.2. Обзор микроконтроллеров

Микроконтроллер в устройстве определения занятости осуществляет работу по управлению датчиками, оцифровке и обработке полученных с них данных. Алгоритмы, запрограммированные внутри микроконтроллера, выполняют операции по анализу входных параметров и на выходе однозначно определяют, занято место или нет. После получение состояния микроконтроллер направляет данные приемопередатчику, выставляет нужные параметры частоты, канала, адреса приемника и другие метаданные для пересылки. Такой микроконтроллер должен обладать хорошей энергоэффективностью для обеспечения долгой работы от аккумуляторной батареи.

В составе коммутатора (хаба) микроконтроллер играет роль устройства, которое собирает данные со всех назначенных ему устройств определения занятости, хранит в себе всю информацию в оперативной памяти и передает её на информационное табло и удаленный сервер. Устройство хаба должно работать от электросети, значит энергоэффективность не так важна. Однако, в связи с работой с большим объемом данных, микроконтроллер должен иметь высокую производительность и большой объем оперативной памяти.

Можно выделить 2 семейства микроконтроллеров: AVR ATmega и STM32.

Среди всех микроконтроллеров семейства AVR ATmega максимальная частота процессора составляет 20MHz при встроенной оперативной памяти в 128KB, что в несколько раз хуже существующих решений на STM32. Преимуществом микроконтроллера ATmega является простота программирования, например, с помощью Arduino IDE, что позволяет очень быстро писать программы для данной платформы. Недостатком является низкая производительность и более ограниченный функционал по сравнению с STM32.

Чипы STM32F1 способны работать на частоте до 72M и имеют объем оперативной памяти до 96KB. При этом стоимость самих микроконтроллеров ATmega и STM32 примерно одинаковая.

1.2.3. Обзор технологий передачи данных

Чтобы передавать данные от устройств определения занятости к коммутатору, а от коммутатора к информационному табло, нужно определить способ и протоколы передачи данных между всеми устройствами всей системы. Выбираемая технология передачи данных должна работать на больших дистанциях. Модули приёмопередатчи должны быть энергоэффективными и недорогими.

1.2.3.1. Проводной способ подключения датчиков

Достоинства:

- возможность организации работы датчиков от электросети, что исключает обслуживание устройств по замене АКБ;
- уменьшение стоимости и размера устройств определения занятости за счет исключения модуля радио-приёмопередатчика.
- более стабильная передача, по сравнению с беспроводной.

Недостатки:

- сложность внедрения в инфраструктуру: требуются траншеи для прокладки проводов, что значительно увеличивает стоимость монтажа системы, а также портит внешний вид.

1.2.3.2. Zigbee [12]

Достоинства:

- возможность создания сложных разветвленных сетей;

– низкое энергопотребление.

Недостатки:

– дальность работы, ограниченная 100 метрами;
– высокая стоимость модуля (от 600 рублей).

1.2.3.3. Lo-RaWan [12]

Достоинства:

– возможность создания сложных разветвленных сетей;
– дальность работы до 10 км;
– высокая безопасность передачи данных (Basic 64-128 bit encrypted);
– низкое энергопотребление.

Недостатки:

– сложная работа с модулем;
– высокая стоимость модуля (от 600 рублей).

1.2.3.4. NB-IoT (Narrow Band Internet of Things)

Достоинства:

– устройство с таким передатчиком может работать автономно в любой точке, где доступен GSM сигнал;
– передача данных напрямую к серверу без нужды внедрения коммутатора;
– низкое энергопотребление.

Недостатки:

– высокая стоимость модуля (от 623 рублей).

1.2.3.5. Радио модуль NRF24L01 [13]

Достоинства:

– простота работы с модулем по SPI;

– низкая стоимость модулей (от 100 рублей).

Недостатки:

– расстояние приема/передачи данных (до 100 м);
– отсутствие возможности в организации сложных сетей, на одном радиоканале может быть 7 модулей (1 приемник и 6 передатчиков).

1.2.4. Обзор языков программирования

Выбор языка программирования в сфере микроконтроллеров является крайне важной задачей, поскольку от его выбора зависит вся архитектура будущей разработки. При выборе языка следует учитывать следующие критерии:

– совместимость и переносимость кода;
– время на разработку сложных алгоритмов;
– наличие библиотек для современных беспроводных приемопередатчиков;
– быстродействие;
– простота дальнейшего поддержания разработки.

Можно выделить несколько популярных языков программирования для микроконтроллеров:

– Assembler;
– C/C++;
– Arduino C++;
– MicroPython;
– JavaScript.

«Ассемблер является языком самого низкого уровня. При этом он позволяет наиболее полно раскрыть все возможности микроконтроллеров и получить максимальное быстродействие и компактный код. […] Ассемблер

отлично подходит для программирования микроконтроллеров, имеющих ограниченные ресурсы, например 8-ми битных моделей с малым объемом памяти. Для больших программ и тем более 32-разрядных контроллеров, лучше использовать другие языки, отличающиеся более высоким уровнем. Это позволит создавать более сложные и при этом понятные программы.» [15]

Вполне возможно рассматривать Ассемблер для программирования тех узлов системы, которые должны работать крайне быстро и имеют небольшой функционал или несложный алгоритм обработки. Но Ассемблер не подходит, если учитывать критерий времени написания программы и простоты дальнейшего поддержания в рамках разработки данной системы мониторинга. Потребуется разработка функционала работы микроконтроллера с модулями примеопередачи, на что может уйти большая и неоправданная часть времени.

«Язык программирования C/C++, относится к языкам более высокого уровня, по сравнению с Ассемблером. Программа на этом языке лучше понятна человеку. Достоинством C/C++ является огромное число программных средств и библиотек, позволяющих просто создавать необходимый код.» [15]

По сравнению с Ассемблером язык C/C++ лучше всего подходит по большинству выбранных критериев для программирования микроконтроллеров. К незначительным недостаткам лишь можно отнести сложную для изучения структуру.

Arduino C++ – это некая облачка на основе фреймворка Wiring для языка C++. Она упрощает программирование микроконтроллеров, позволяя очень быстро разрабатывать программное обеспечение со сложными алгоритмами, которое нетрудно поддерживать [16]. Но при этом у такого языка есть существенные недостатки по сравнению с обычным C++ в виде низкой производительности и в виде дополнительного расхода свободной Flash памяти при загрузке программ, скомпилированных из языка Arduino C++.

MicroPython и JavaScript являются удобными языками для написания ПО под микроконтроллеры, но для этого требуется использование соответствующих прошивок-загрузчиков, работающих в фоновом режиме и исполняющих загруженный код программы. Это повышает уровень абстракции и позволяет создавать программы очень быстро. Но из этого следует и основной недостаток – низкая производительность по сравнению с более низкоуровневыми языками.

1.3. ВЫВОД

Анализ технических решений показал, что наилучшим выбором среди датчиков определения занятости парковочных мест по критериям диапазона измерения и точности является датчик ультразвукового типа. Для создания прототипа хорошо подходит модуль HC-SR04.

Среди микроконтроллеров лучший выбор по критериям производительности и энергоэффективности – это микроконтроллеры серии STM32. Однако, в рамках разработки модели будущей системы наилучшим образом подходят микроконтроллеры серии ATmega с использованием среды разработки Arduino IDE.

Для передачи и приема данных между устройствами лучше всего подходит стандарт LoRa за счет большой дистанции работы между устройствами и хорошей энергоэффективности. Для реализации макета, в целях уменьшения затрат, хорошо подходит радио модуль NRF24L01.

Для программирования микроконтроллеров был выбран язык C/C++, поскольку он сочетает в себе все самые наилучшие показатели по критериям быстродействия, наличия большого количества библиотек и удобства разработки и поддержки программного обеспечения.

2. ОПРЕДЕЛЕНИЕ ТРЕБОВАНИЙ

Для более подробного и четкого определения требований выделим 3 основные подсистемы, которые обязательно должны быть включены в систему мониторинга для полного функционирования:

- устройства определения занятости;
- информационное табло для информирования водителей о количестве свободных мест;
- коммутирующие устройство (хаб) для объединения устройств определения занятости и информационных табло.

2.1. ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

С помощью диаграммы вариантов опишем взаимодействие актеров (рисунок 2.1.1). Она отражает события, возникающие на стояночной зоне с точки зрения водителя, который собирается припарковать свое транспортное средство и ищет свободную позицию.

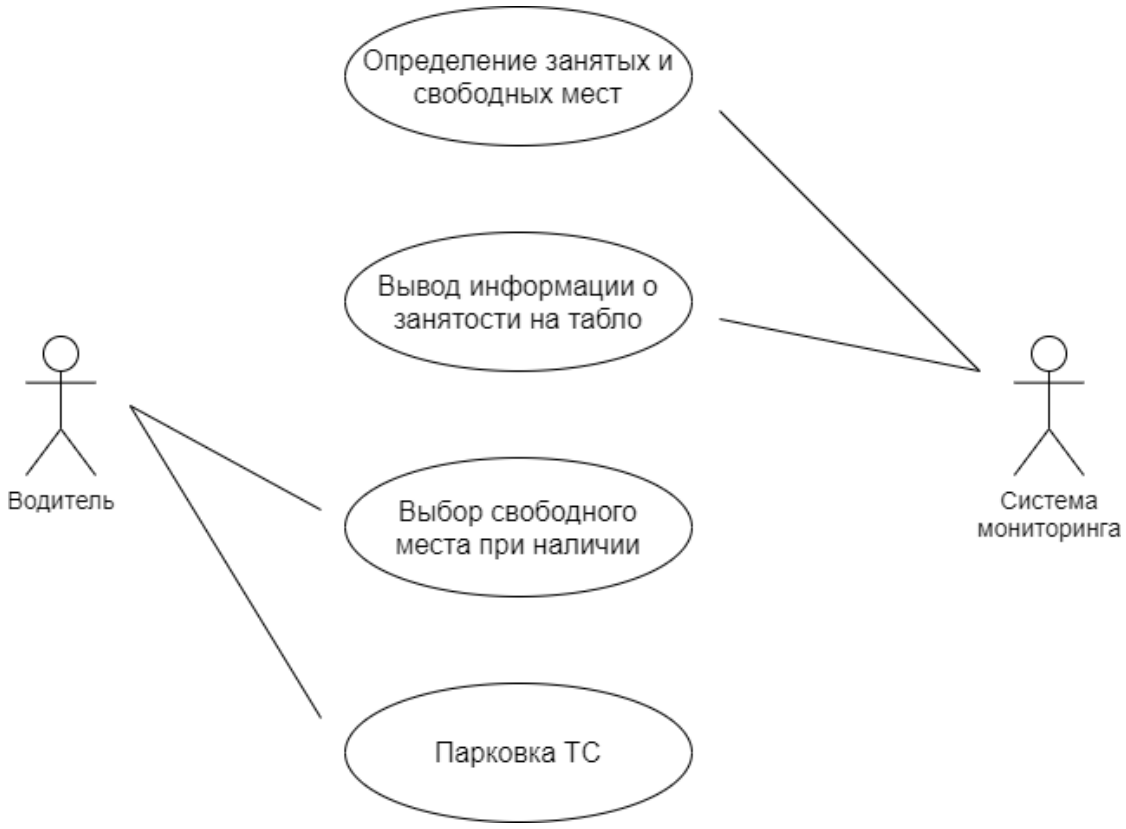


Рисунок 2.1.1 – Диаграмма вариантов использования системы мониторинга занятости парковочных мест

На диаграмме в качестве актёров представлены следующие объекты:

– водитель – человек, который управляет транспортным средством и занимается поиском свободного места на парковочной зоне;
– система мониторинга – вспомогательная система, осуществляющая мониторинг занятости на специально оборудованных зонах и предоставляющая информацию о доступности мест на информационное табло.

Система мониторинга, установленная на стоянке, производит регулярное считывание статуса каждого парковочного места. При изменении состояния

хотя бы в одной позиции на информационном табло должна быть обновлена информация о количестве свободных мест.

2.1.1. Реализуемые задачи

Перечень задач, которые должны быть реализованы:

- проектирование архитектуры для аппаратного комплекса в целом;
- проектирование архитектуры физического устройства и разработка алгоритмов мониторинга для датчиков определения занятости;
- разработка программного обеспечения для работы коммутатора с устройствами определения занятости;
- разработка информационного табло для информирования окружающих водителей;
- разработка функционала удаленного доступа и управления к аппаратному комплексу через Интернет (API).

Функционал удаленного доступа требуются для внешних информационных систем, например, для веб-приложений, с помощью которых должна быть возможность конфигурирования системы.

В перечень конфигурационных настроек системы входят:

- внесение новых устройств определения занятости;
- удаление существующих устройств определения занятости;
- отключение и включение отдельных устройств определения занятости.

Также с помощью функционала удаленного доступа внешние информационные сервисы должны иметь возможность получать информацию о статусе стоянки в целом и о каждом парковочном месте по отдельности. Такими сервисами могут быть, например, картографические сервисы, которые способны размещать в своем интерфейсе расположение свободных и занятых мест на реальной карте города.

2.1.2. Функциональные требования к системе в целом

Перечень функциональных требований к системе в целом:

- в каждой парковочной зоне, в самом начале въезда, устанавливается специальное табло, информирующее водителей о количестве свободных мест и, при возможности, их схематичном расположении в области парковки;
- система мониторинга занятости парковочных мест должна иметь возможность предоставления API для внешних сервисов по сети Интернет;
- должно быть предусмотрено управление системой по сети Интернет, включая функции добавления и удаления устройств определения занятости мест и настройки изменения статуса занятости отдельных парковочных мест;
- система должна работать с неограниченным числом парковочных мест, а устройства определения занятости на этих местах должны строго привязываться к своим выделенным коммутаторам, при этом количество коммутаторов также неограниченно.

2.1.3. Функциональные требования для подсистемы устройств определения занятости паковочного места

Перечень функциональных требований для подсистемы устройств определения занятости паковочного места:

- устройства определения занятости парковочного места должны четко и корректно определять занятость стояночных мест в оборудованной зоне и передавать их на коммутатор;
- устройство должно быть монолитным, без гнезд подключений, без проводов.

2.1.4. Функциональные требования для подсистемы коммутатора

Перечень требований для подсистемы коммутатора:

– коммутатор должен иметь возможность подключать к себе одновременно как минимум 200 парковочных мест;
– коммутатор должен подключаться по мобильной сети к Интернету и предоставлять доступ к внешним информационным системам по API.

2.1.5. Функциональные требования для подсистемы информационного табло

Перечень требований для подсистемы информационного табло:

– дисплей информационного табло должен иметь высокую яркость и контрастность ночью и днем, чтобы видимость была четкая как минимум на расстоянии 50 метров от него.

2.2. НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ

2.2.1. Нефункциональные требования к системе в целом

Перечень нефункциональных требований для системы целом:

– передача данных между устройствами определения занятости и коммутатором должна быть беспроводной;
– коммутатор должен иметь возможность передачи данных с устройствами определения занятости парковочных мест на расстоянии как минимум до 100 метров.

2.2.2. Нефункциональные требования для подсистемы устройств определения занятости паковочного места

Перечень нефункциональных требований для подсистемы устройств определения занятости:

– устройство должно работать автономно от аккумуляторной батареи без подзарядки и замены в течении 3-х и более лет;
– конструкция устройства должна предусматривать возможность замены аккумулятора;
– устройство должно стабильно работать при температурах окружающей среды в диапазоне от -40 °C до +50 °C;
– защита по стандарту IP67 (полная пыленепроницаемость и защита от погружения в воду до 1 метра).

2.2.3. Нефункциональные требования для подсистемы коммутатора

Перечень нефункциональных требований для подсистемы коммутатора:

– информация о количестве доступных мест должна поступать на дисплей с задержкой не более чем 5 секунд;
– информационное табло должно работать вместе с коммутатором по беспроводной сети и минимальное расстояние между этими подсистемами, на котором должна быть обеспечена бесперебойная передача информации, должно быть не менее чем 500 метров;
– устройство должно стабильно работать при температурах окружающей среды в диапазоне от -40 °C до +50 °C;
– защита по стандарту IP67 (полная пыленепроницаемость и защита от погружения в воду до 1 метра).

2.1.4. Функциональные требования для подсистемы коммутатора

Перечень нефункциональных требований для подсистемы коммутатора:

– вычислительной мощности коммутатора должно хватать на обработку как минимум 200 парковочных мест подключенных одновременно, чтобы задержка между обработкой всех запросов составляла не более 5 секунд.

2.2.5. Объем данных передачи по сети Интернет

Объем данных очень мал и измеряется порядками нескольких Кбайт за день работы. Это связно с передачей только данных изменения статуса отдельного парковочного места, что не требует частой пересылки.

2.2.6. Временные характеристики

Коммутатор обновляет данные в своей памяти с задержкой не более 5 секунд и отправляет их на информационное табло.

3. ПРОЕКТИРОВАНИЕ

3.1. АРХИТЕКТУРА ПРЕДЛАГАЕМОГО РЕШЕНИЯ

Рассмотрим принцип работы предполагаемого решения. Физически и условно вся система разделяется на 3 подсистемы. Это устройства определения занятости, хаб (коммутатор) и информационное табло. На рисунке 3.1.1 представлена схема, описывающая взаимодействие одного устройства каждой подсистемы во всём цикле мониторинга и информирования о занятости.

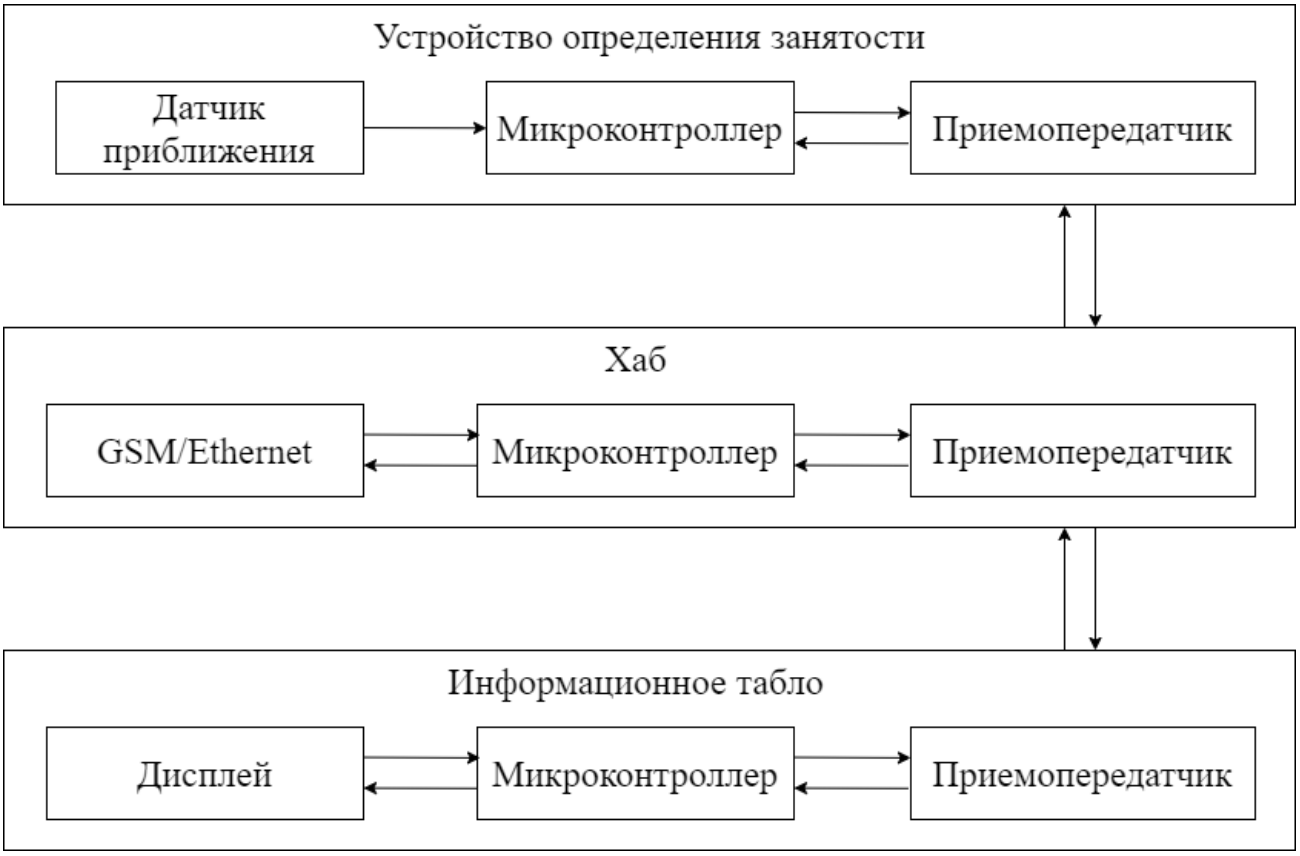


Рисунок 3.1.1 – Архитектура предполагаемого решения

3.1.1. Устройства определения занятости

Устройство определения занятости в одном цельном корпусе объединяет три функциональных компонента: датчик приближения, микроконтроллер и приемопередатчик. Помимо этого, для автономной работы внутри устройства устанавливается аккумулятор.

Для повышения точности определения занятости парковочного места с помощью ультразвуковых датчиков приближения была выбрана перекрестная схема установки сенсоров (см. рисунок 3.1.2). Такое расположение предусматривает монтаж устройств в столбиках для ограждения парковочных зон.

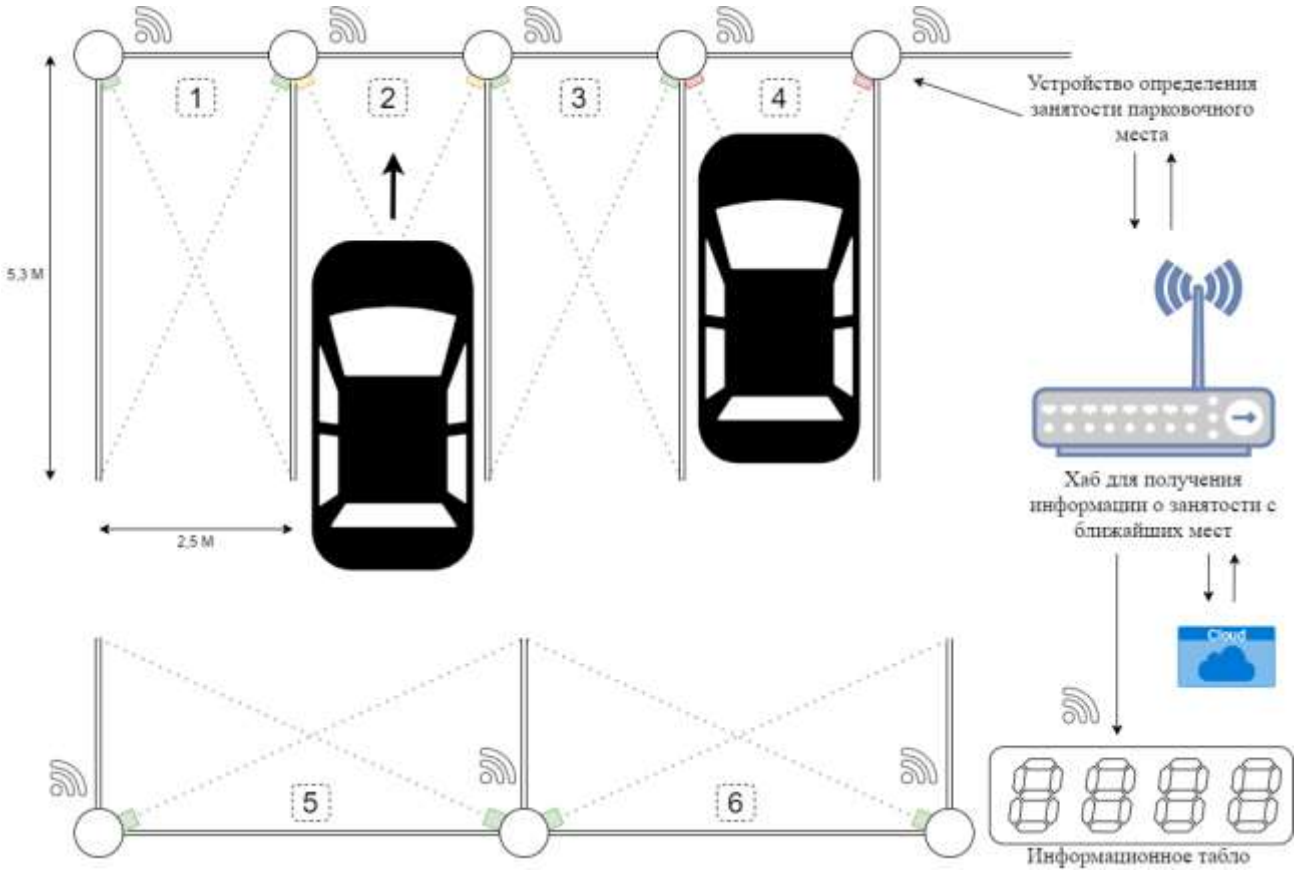


Рисунок 3.1.2 – План-схема расположения датчиков определения занятости, хаба и информационного табло

На рисунке 3.1.2 показано, что место №4 занято, место №2 занимается в данный момент, а остальные – свободны. Состояние занятости определяется по расстоянию до объекта, определенного датчиками. Состояние занятия или освобождение места определяется устройствами соответственно в случаях приближения к датчикам ТС или при отдалении от них.

Микроконтроллер устройства определения занятости работает с каждым своим датчиком приближения по-отдельности и во всей системе мониторинга разграничиваются на левый и правый датчик относительно фронтального расположения (рисунок 3.1.3). Если хоть один из них фиксирует изменения в расстоянии или скорости приближения объекта, то информация об этом

отправляются на хаб одним пакетом данных без учета состояния соседнего датчика.

Отправка и получение информации для микроконтроллера обеспечивается с помощью встроенного в устройство приёмопередатчика.

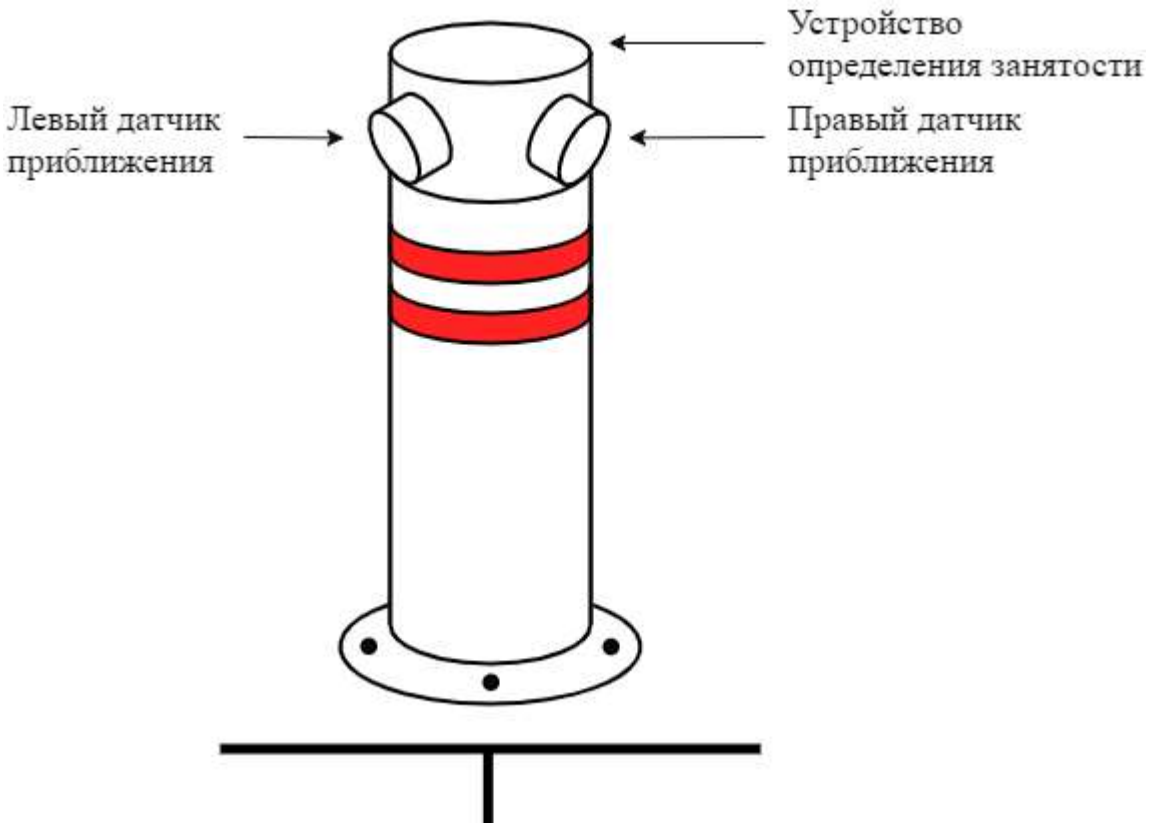


Рисунок 3.1.3 – Расположение устройства определения занятости на ограничительном парковочном столбике

3.1.2. Хаб

Хаб включает в себя 3 ключевых компонента: микроконтроллер, локальный приемопередатчик, связывающий устройства других подсистем на одной парковке между собой и GSM или Ethernet модуль для передачи данных и

управления системой по сети Интернет. Устройство работает от сети переменного тока и должно снабжаться блоком питания.

Самая главная задача микроконтроллера хаба – принятие данных о статусе со всех подключенных к нему устройств определения занятости. Для этого предусматривается высокая производительность и большое количество оперативной встроенной памяти.

В постоянной памяти микроконтроллера хранится информация о парковочных местах. Одна запись о парковочном месте содержит в себе тип конфигурации места, адрес левого и адрес правого устройства определения занятости. При запуске устройства данные помещаются в оперативную память.

Данные об изменении статуса устройств мониторинга принимаются через приемопередатчик и записываются в оперативную память по соответствующему адресу. Если в течении небольшого промежутка времени получены данные с двух датчиков с одного парковочного места, производится их сравнение. И если они приблизительно равны, то обновляется статус всего места.

После изменения статуса какого-либо места на информационное табло передаются число свободных мест.

Микроконтроллер работает с модулем передачи данных по сети Интернет и предоставляет API. Это позволяет в любой момент времени с помощью запроса к хабу получить информацию о занятости парковки в целом и по отдельно выбранным местам. Посредствам API осуществляется настройка новых и уже существующих парковочных мест и прикреплённых к ним устройств мониторинга. Через веб-интерфейс можно определить все доступные устройства определения занятости поблизости, если они не проинициализированы и добавить их к новой парковке. Можно отключить видимость парковочного места или даже удалить его. Все эти изменения происходят в оперативной памяти хаба и сохраняются в постоянной в ПЗУ.

3.1.3. Информационное табло

Информационное табло состоит из 3-х основных компонентов: приёмопередатчик для получения данных от хаба, микроконтроллер для обработки полученных данных и дисплей, отражающий количество свободных мест. Устройство должно работать от сети переменного тока в 220 вольт, поэтому также должен включаться блок питания для преобразования напряжения.

3.2. АЛГОРИТМЫ РЕШЕНИЯ ЗАДАЧИ

Для нормальной работы всей системы обязательно должны функционировать 2 устройства определения занятости и хаб. Два устройства мониторинга на одном парковочном месте фиксируют изменение расстояния до объекта и передают эти значения хабу. Если данные к хабу приходят только от одного датчика с одного места, то это считается ошибкой и статус никак не меняется. Если данные пришли с двух датчиков, но между ними есть существенные различия, то это также считается ошибкой. Допущения в различии значений расстояния до объекта с датчиков зависят от типа парковочных мест.

Можно выделить следующие типы парковочных мест и способы размещения транспортных средств:

- под углом 90° (поперечная расстановка);
- под углами в диапазоне от 90° до 0°, обычно это 60°, 45° или 30° (расстановка «елочкой»);
- под углом 0° (параллельная расстановка вдоль, проезжей части).

На рисунке 3.2.1 представлена условная схема описанных видов стояночных мест.

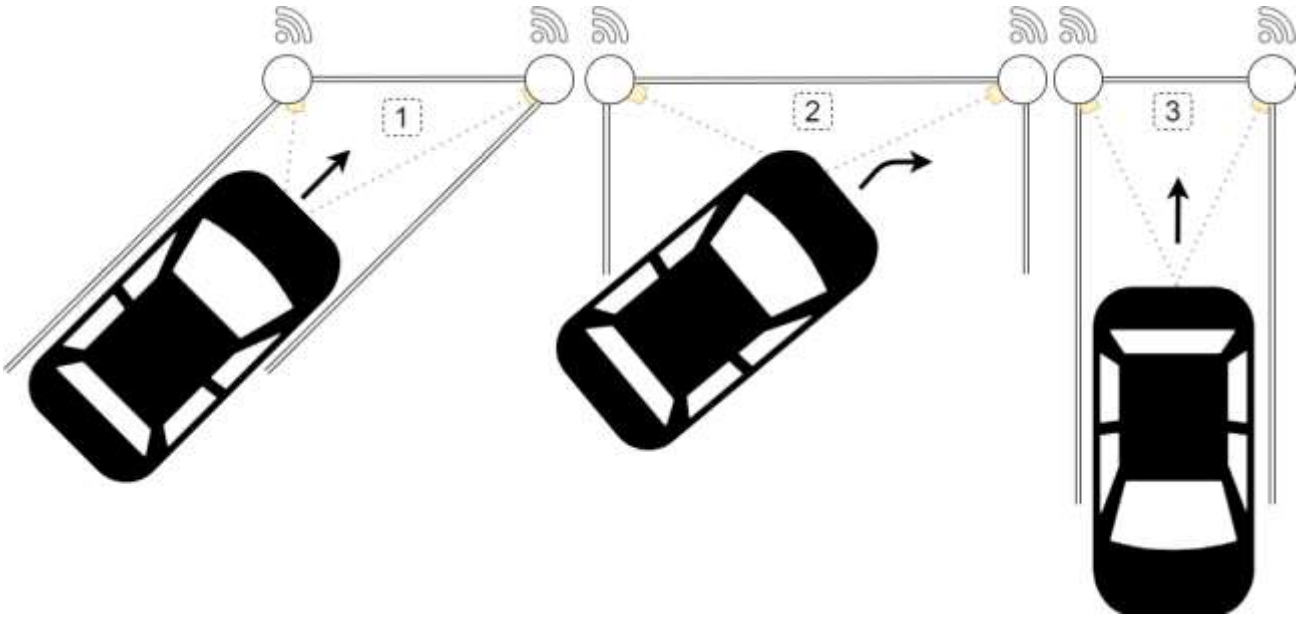


Рисунок 3.2.1 – Виды парковочных мест и способы парковки

С парковочным местом, которое находится под углом 90° всё достаточно просто. Предполагается, что транспортное средство заезжает и выезжает со стоянки прямо и равномерно и расстояния, которые определяют левый и правый датчик до ТС примерно равны с погрешностью до 50 см. При такой конфигурации также возможно определять скорость, и соответственно направление, перемещения водителя, что позволяет сделать алгоритм определения занятости более точным.

Если рассматривать ситуацию с парковкой «ёлочкой», когда карман стоянки расположен под углом в диапазоне от 90° до 0°, то нужно учитывать, что расстояния, которые определяют левый и правый датчик до ТС, будут отличаться. Это расстояние зависит от угла парковки α (рисунок 3.2.2).

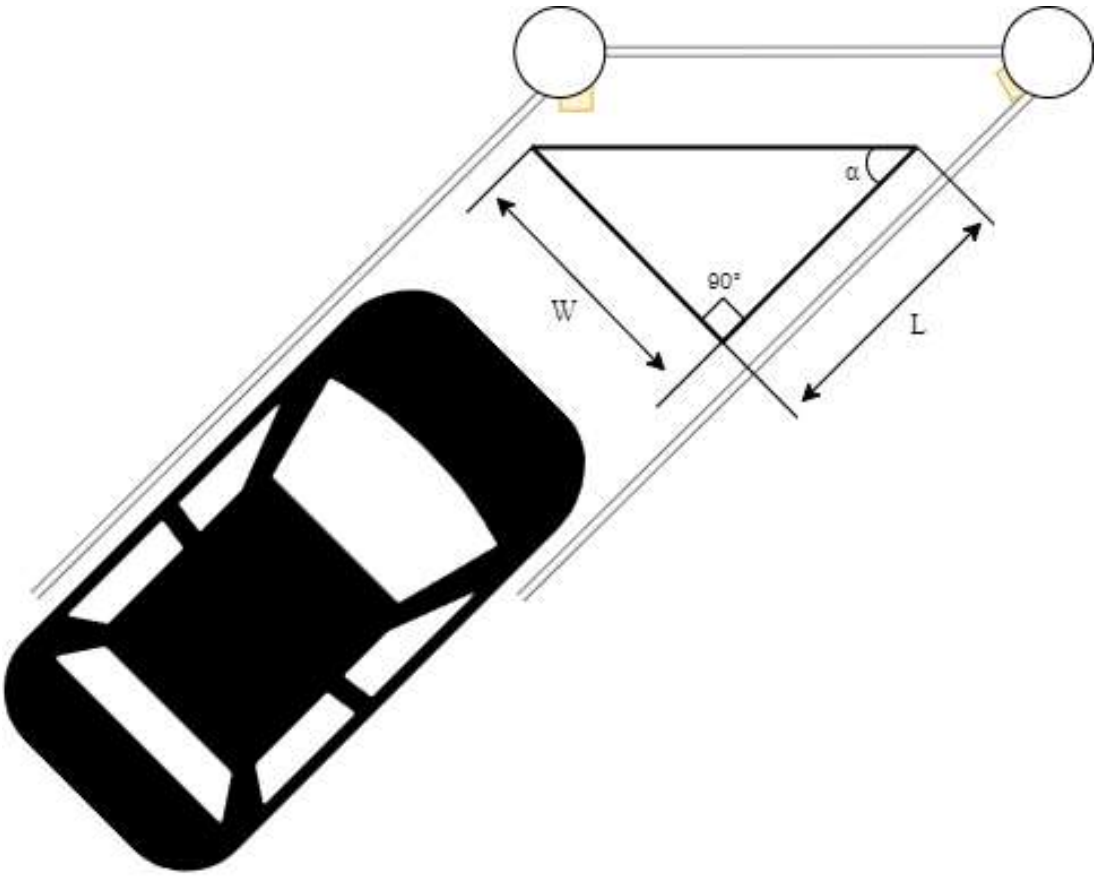


Рисунок 3.2.2 – Определение расстояния до дальнего датчика через вычисление катета прямоугольного треугольника

Можно отметить, что при правильной парковке ТС в кармане парковочного места по типу «елочка», между двумя датчиками и транспортом вписывается прямоугольный треугольник. (рисунок 3.2.2). Известный катет (W) – это ширина парковочного места. Также известен противолежащий данному катету угол (α) – это угол наклона направляющих линий разметки, относительно дальней ограничительной линии, на которой расположены устройства определения занятости. По известным параметрам можно определить неизвестный катет, длина которого является значением расстояния между ТС и дальним датчиком.

Таким образом, расстояние до дальнего датчика можно вычислить по формуле (3.2.1)

$$L = \frac{W}{\tan(\alpha)}, \quad (3.2.1)$$

где W – ширина парковочного места;

α – угол наклона линий разметки.

Таким образом определяется расстояние, которое нужно прибавить к значению дистанции, полученного от дальнего датчика. Можно составить таблицу, в которой будет видно корректировочное значение в зависимости от угла наклона линий парковки при ширине парковки в 2,5 м.

Таблица 3.2.1 – Расстояния до дальнего датчика в зависимости от угла

Угол	**Расстояния до дальнего датчика**
30°	4,33 м.
45°	2,5 м.
60°	1,33 м.
90°	0 м.

Из табл. 3.2.1 можно приметить, что при значении угла в 90° расстояние до дальнего датчика равно 0 м. Это соответствует поперечному типу стояночного места, значит формула универсальна по отношению двум типам парковки, но не подходят для конфигурации параллельной расстановки стоянки.

При заезде и выезде на параллельной парковке (рисунок 3.2.1) расстояния между ТС и каждым датчиком всегда неодинаково. Плюс к этому, все водители могу парковаться по-разному, выбирая по-своему сторону и стиль заезда и выезда. Поэтому определять скорость и направление перемещения ТС в пределах парковочной позиции в таком случае не представляется возможным.

Но всё равно доступно определение занятости по расстоянию, которые считывают датчики и сценарий при обработке этих данных получается схож с тем алгоритмом, который применятся для определения занятости на поперечной парковке (угол в 90°). Только при этом устройства определения занятости располагаются по углам направляющей линии стоянки.

На рисунке 3.2.3 представлен упрощенный алгоритм работы всей системы в целом вне зависимости от конфигурации стояночных мест.

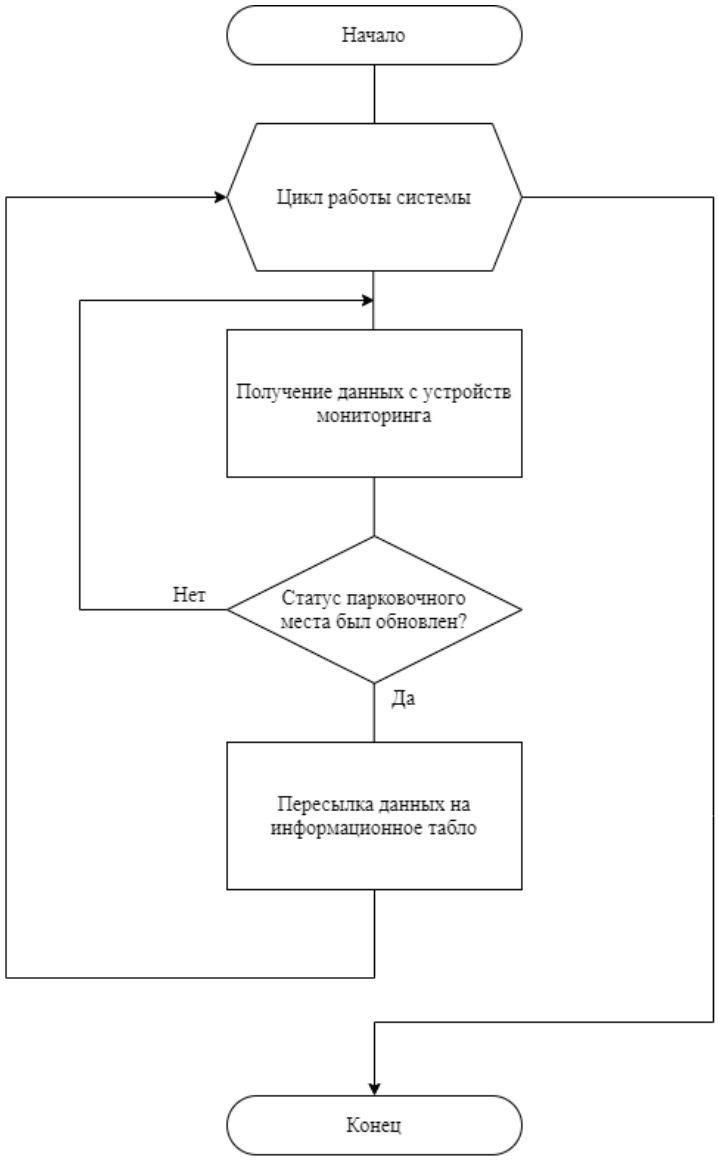


Рисунок 3.2.3 – Алгоритм работы системы мониторинга в целом

На рисунке 3.2.4 представлен упрощенный алгоритм работы устройства определения занятости.

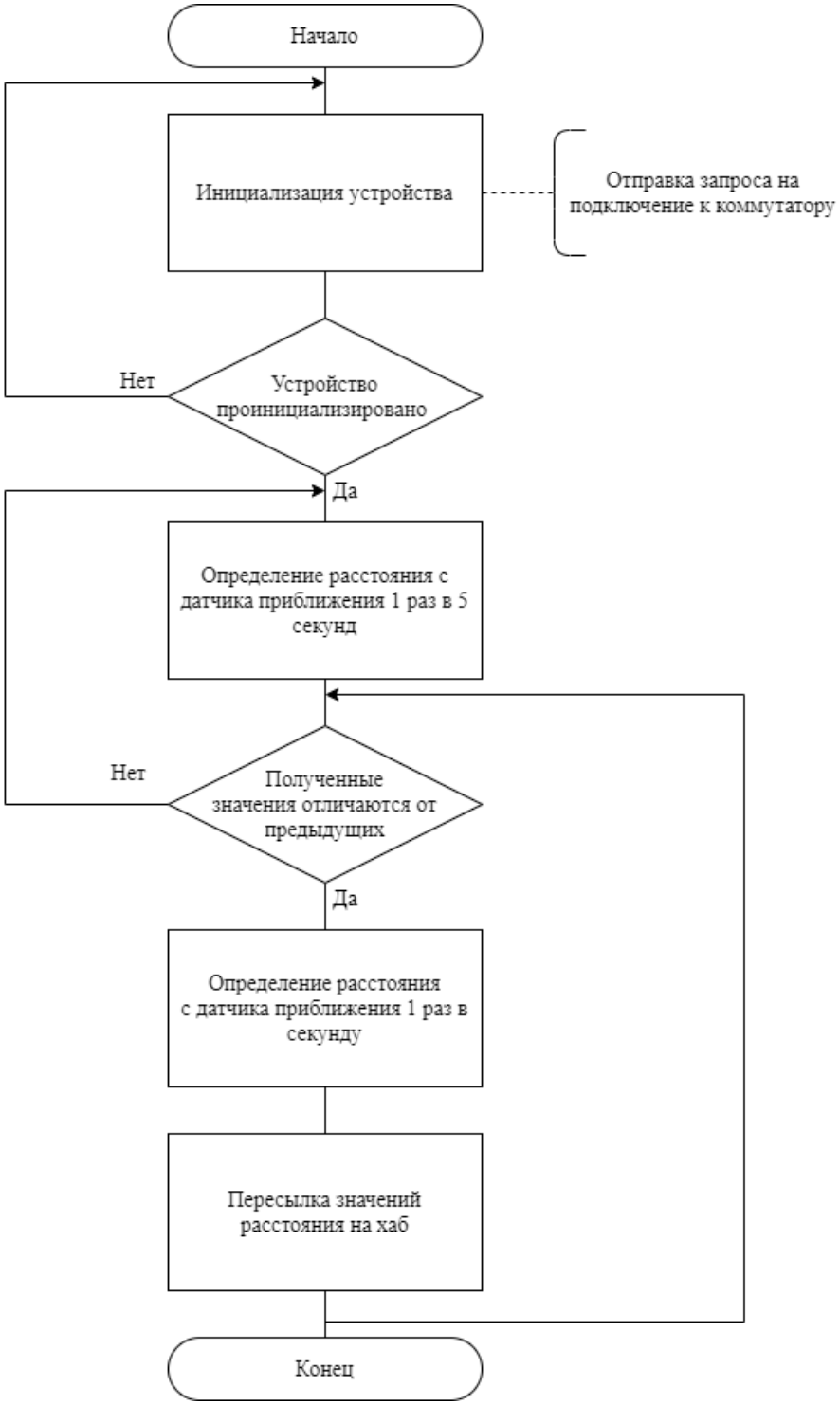


Рисунок 3.2.4 – Алгоритм работы устройства определения занятости

На рисунке 3.2.5 представлен упрощенный алгоритм работы основного цикла программы хаба.

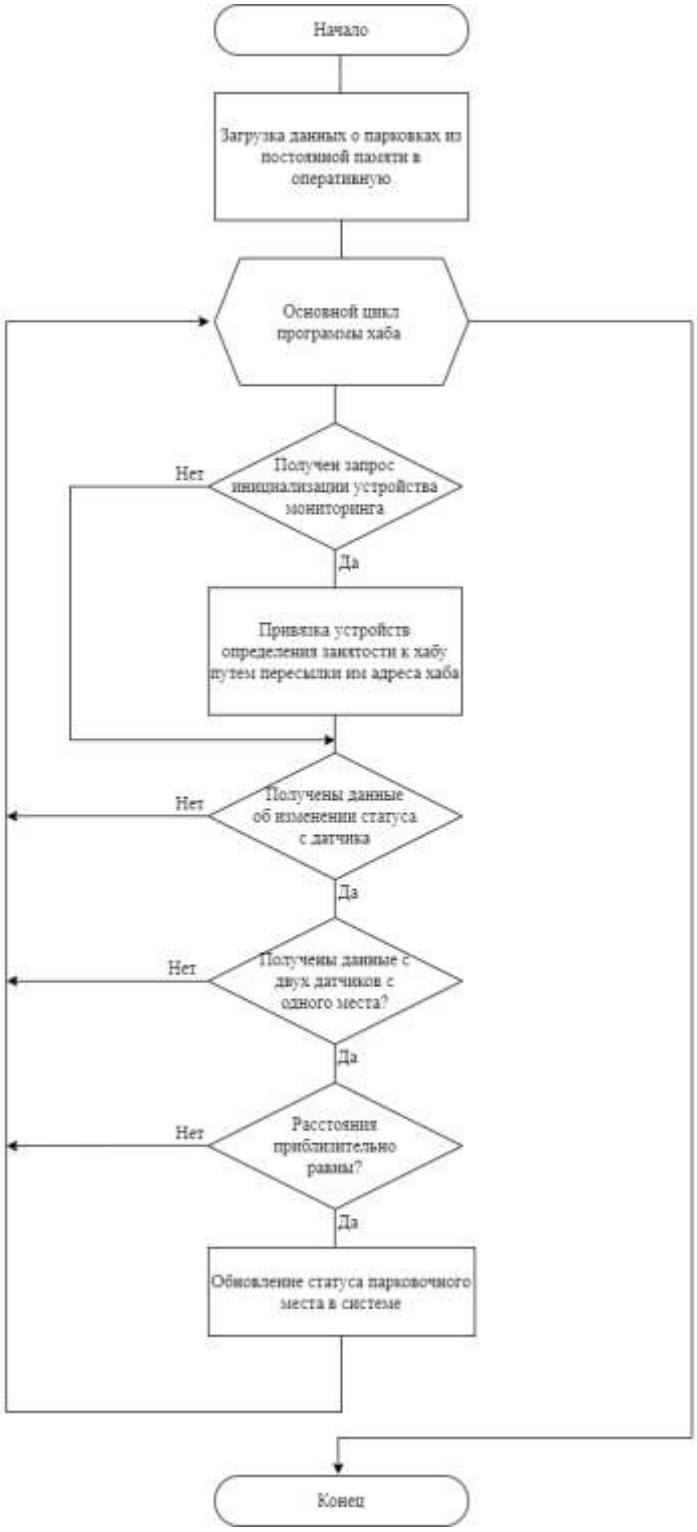


Рисунок 3.2.5 – Алгоритм работы программы хаба

3.3. ОПИСАНИЕ ДАННЫХ

Опишем структуру пакета данных, передаваемых от одного устройства к другому на прикладном уровне (табл. 3.3.1). Весь пакет занимает 14 байт. Передача может осуществляться только между хабом и другими подключенными к нему устройствами.

Таблица 3.3.1 – Структура пакета данных

Байты	1	4	4	5
Название	Код функции	Адрес устройства	Адрес хаба	Данные

Код функции в системе мониторинга – это команда, которая должна быть выполнена адресатом. Список доступных функций представлен в таблице 3.3.2.

Таблица 3.3.2 – Коды функций

Код HEX	**Название функции**
0x11	Запрос инициализации устройства определения занятости
0x12	Передача информации о состоянии устройства мониторинга
0x13	Передача информации об изменении статуса с датчика приближения
0x20	Команда сброса всех подключенных к хабу устройств
0x21	Отправка адреса хаба для инициализации устройства мониторинга
0x22	Отправка запроса устройству мониторинга на отключение от системы (сброс адреса хаба)
0x23	Запрос на получение состояния устройства (например, состояние аккумулятора)
0x24	Отправка данных для информационного табло о количестве свободных мест
0x25	Отправка адреса хаба для инициализации табло
0x31	Запрос инициализации информационного табло

Запрос инициализации устройства отправляется при условии, что оно не привязано ни к одному хабу, то есть, если в устройстве записан адрес коммутатора 0x00. Частота отправки – 1 раз в минуту. Используется адрес коммутатора 0x00. Состав данных не имеет значения. Ожидается, что устройство получит ответ в виде пакета данных, включающего в себя адрес коммутатора, к которому оно привязывается.

Адрес устройства – уникальный номер устройства определения занятости либо информационного табло, записываемый при программировании устройства.

Адрес хаба – 4-х байтовое число. Оно записывается в устройство автоматически при инициализации. Это значение определяет, какому коммутатору назначаются передаваемые данные.

При использовании 2-й функции, «передача информации о состоянии устройства», данные об изменении статуса с датчиков приближения включает в себя значения расстояния и до объекта и скорость его перемещения. По знаку значения скорости перемещения определяется направление перемещения объекта: приближение или отдаление относительно датчиков.

Таблица 3.3.3 – Структура данных в полном пакете данных от устройства определения занятости

Байт	1	2	2
Название	Код датчика приближения	Расстояние до объекта	Скорость приближения объекта

Код датчика приближения может быть 0x01 или 0x02. 0x01 – это левый датчик, 0x02 – правый датчик. Сторона определяется относительно от фронтального расположения устройства (рисунок 3.1.3).

Пример одного сообщения, содержащего в себе данные значений датчиков приближения, от устройства мониторинга показан в таблице 3.3.4.

Таблица 3.3.4 – Пример передачи пакета об изменении статуса с датчика

Название	Функция	Адрес устр.	Адрес хаба	Данные		
Байты	1	4	4	1	2	3
Значение	0x12	0xAF5B1D01	0x00000005	0x01	0x0064	0x0000

Хаб может передать данные к устройствам определения занятости и к информационному табло. Функциональные коды представлены в таблице 3.3.6.

Данные о парковочных местах в микроконтроллере хаба представляются в виде массива структур. Структура представляет из себя запись о стояночном месте. Эти данные можно представить в виде JSON массива (рисунок 3.3.1). Этот формат также удобно можно использовать для передачи хабу настроек по сети Интернет через внедренное API от внешних информационных систем.

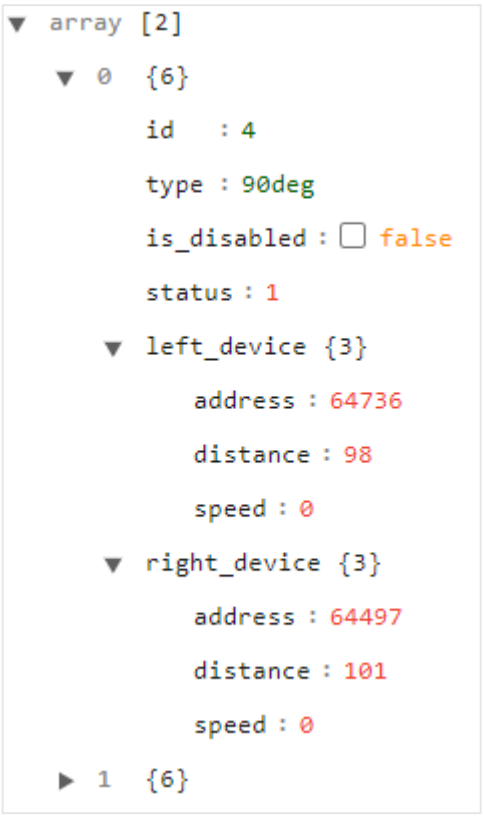


Рисунок 3.3.1 – Структура данных одной записи об одном парковочном месте в формате JSON

Расшифровка полей представлена в таблице 3.3.1.

Таблица 3.3.1 – Расшифровка названий свойств структуры записи JSON

Название	**Расшифровка**
id	Уникальный номер стояночного места в пределах системы
type	Тип парковочного места. Указывается угол положения <90deg…0deg>
is_disabled	Настройка, отключающая парковку из системы
status	Статус занятости парковочного места 1 – место занято, 0 – место свободно
left_device	Запись свойств левого устройства определения занятости
right_device	Запись свойств правого устройства определения занятости

В каждой записи о парковке должна указываться информация о привязанных к ней устройствах мониторинга в полях left_device и right_device в формате JSON записи. Значения ключей и их расшифровка представлены в таблице 3.3.2.

Таблица 3.3.2 – Расшифровка названий свойств структуры для прикрепленных устройств мониторинга (left_device и right_device)

Название	**Расшифровка**
address	Уникальный адрес устройства мониторинга
distance	Значение дистанции, полученное с устройства
speed	Значение скорости, определенное за время изменения дистанции

4. РЕАЛИЗАЦИЯ

Выполним сборку макета с простыми моделями устройств, чтобы проверить работоспособность системы, затем запрограммируем их.

4.1. РЕАЛИЗАЦИЯ МАКЕТОВ УСТРОЙСТВ

Для реализации макета были выбраны оптимальные компоненты по важным критериям функционирования обозначенных в разделе 1.1 и по стоимости приобретения в розничных магазинах. Покупка всех описанных далее запчастей и компонентов производилась в магазине «ПРОКОНТАКТ» в городе Челябинске.

4.1.1. Отладочные платы Ардуино

Для реализации макета за основу были взяты микроконтроллеры серии ATmega. Воспользуемся самыми доступными по стоимости отладочными платами Arduino Nano и Arduino Uno на одном и том же чипе ATmega328P. Они различаются только размером. Преимуществом отладочных плат являются то, что они совмещает множество вспомогательных деталей, которые помогают упростить создание макета. Например, важным элементом является кварцевый резонатор, который позволяет работать микроконтроллеру на частоте в 16 MHz. Без него тактирование МК осуществляется RC-цепочкой, встроенной в чип. Такой способ гораздо менее стабилен, а максимальная частота работы процессора ограничена показателем в 1 MHz.

Также важной особенностью плат является наличие блока пинов SPI интерфейса для программатора USBasp, что оптимизирует процесс загрузки программ напрямую в память устройства, без бутлоадера.

Для устройств мониторинга и для хаба используется Arduino Nano (рисунок 4.1.1.1).

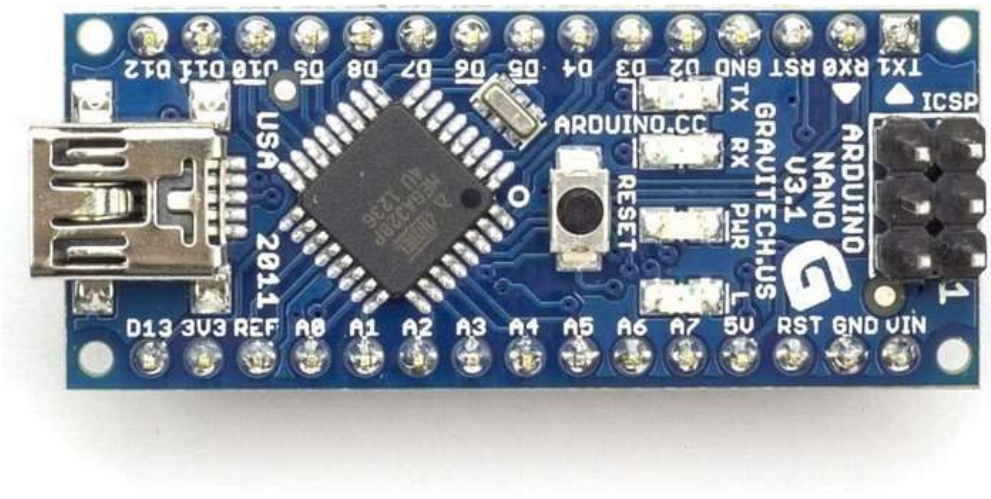


Рисунок 4.1.1.1 – Плата Arduino Nano с микроконтроллером ATmega328P

Для устройства информационное табло используется Arduino Uno (рисунок 4.1.1.2).

Рисунок 4.1.1.2 – Плата Arduino Uno с микроконтроллером ATmega328P

В таблице 4.1.1.1 представлены характеристики для микроконтроллера ATmega328P.

Таблица 4.1.1.1 – Характеристики ATmega328P [17]

Параметр	**Показатель**
Напряжение питания	От 2,7 до 5,5 В
Температурные пределы	От -40°C до +125°C
Макс. частота	16 MHz (при напряжении от 4,5 В)
Объем Flash	32 Kb
Объем SRAM	2 Kb
Потребление	Активный режим: 1,5 mA при V = 3 В и f = 4 MHz Реж. никзк. потребл.: 1 mA при V = 3 В и f = 4 MHz
Кол-во I/O выводов	23 шт.

4.1.2. Ультразвуковой дальномер HC-SR04

Для модели устройства определения занятости в качестве датчика приближения был выбран дальномер HC-SR04. Изображение датчика представлено на рисунке 4.1.2.1.

Рисунок 4.1.2.1 – Ультразвуковой дальномер HC-SR04

Таблица 4.1.2.1 – Характеристики HC-SR04 [18]

Параметр	**Показатель**
Напряжение питания	От 5 В
Температурные пределы	От -20°C до +70°C
Диапазон расстояний	2-400 см
Эффективный угол наблюдения	15°
Рабочий угол наблюдения	30°
Потребление	При работе: 15 мА Потребление в режиме тишины: 2 мА

4.1.3. Приемопередатчик nRF24L01+

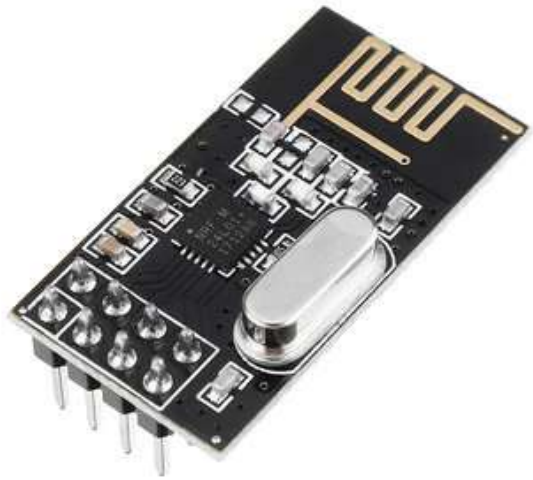


Рисунок 4.1.3.1 – Радиомодуль NRF24L01+

Таблица 4.1.3.1 – Характеристики nRF24L01+ [19]

Параметр	**Показатель**
Напряжение питания	От 1,9 до 3,6 В
Макс. ток потребления	13,5 мА
Температурные пределы	От -40°C до +85°C
Расстояние передачи	До 100 м
Частота передачи	2,4 ГГц
Кол-во каналов	126
Интерфейс	SPI
Протокол передачи	Enhanced ShockBrust™

4.1.4. Стабилизатор напряжения LM1117

Модуль приемапередачи NRF24L01+ способен работать только при напряжении от 1,9 до 3,6 В и может потреблять до 13,5 мА. Чтобы не нагружать

встроенный стабилизатор платы Ардуино, воспользуемся внешним стабилизатором напряжения LM1117T с 5 В на 3,3 В (рисунок 4.1.4.1).

Рисунок 4.1.4.1 – Стабилизатор напряжения LM1117T

4.1.5. Отладочная плата NodeMCU

В качестве модуля, обеспечивающего соединение хаба с Интернетом, была выбрана отладочная плата NodeMCU на основе микроконтроллера ESP8266. Он позволяет подключаться к сети Интернет по технологии Wi-Fi. Изображение платы представлено на рисунке 4.1.5.1, характеристики микроконтроллера ESP8266 отражены в таблице 4.1.5.1.

Данный микроконтроллер способен отправлять HTTP запросы, что требуется для передачи данных внешним информационным системам. Также ESP8266 способен быть использован в качестве веб-сервера. Эта функция нужна для возможности подключения администратора к хабу, что позволяет изменять настройки и проверять состояние системы.

Соединение платы с микроконтроллером Atmega328P выполнено по интерфейсу UART.

Рисунок 4.1.5.1 – Отладочная плата NodeMCU

Таблица 4.1.5.1 – Характеристики NodeMCU [20]

Параметр	**Показатель**
Напряжение питания	От 1,9 до 3,6 В
Макс. ток потребления	170 мА
Температурные пределы	От -40°C до +125°C
Стандарт Wi-Fi	802.11 b/g/n
Процессор	Tensilica L106, 32 бита
Скорость процессора	80...160 МГц
ОЗУ	32 Кб + 80 Кб

4.1.6. Семисегментный индикатор HSN-5643AS-H

Для макета информационного табло был выбран семисегментный индикатор HSN-5643AS-H на 4 разряда (рисунок 4.1.6.1). Характеристики представлены в таблице 4.1.6.1.

Рисунок 4.1.6.1 – Семисегментный индикатор HSN-5643AS-H

Таблица 4.1.6.1 – Характеристики семисегметного индикатора [21]

Параметр	**Показатель**
Напряжение питания	5 В
Максимальный ток	90 мА
Температурные пределы	От -25°C до +85°C
Схема подключения	Общий катод

4.1.7. Сборка макета

Для реализации макета оказалось доступно всего 2 ультразвуковых датчика определения расстояния, поэтому может быть реализована точная работа либо одного парковочного места, либо частичная, но с двумя. Был выбран второй вариант. Два датчика должны работать с одним микроконтроллером, но в системе они будут подразделятся на 2 отдельных парковочных места.

Все модели устройств были собраны на беспаечных макетных платах.

Электрическая схема соединения компонентов устройства мониторинга показана на рисунке 4.1.7.1

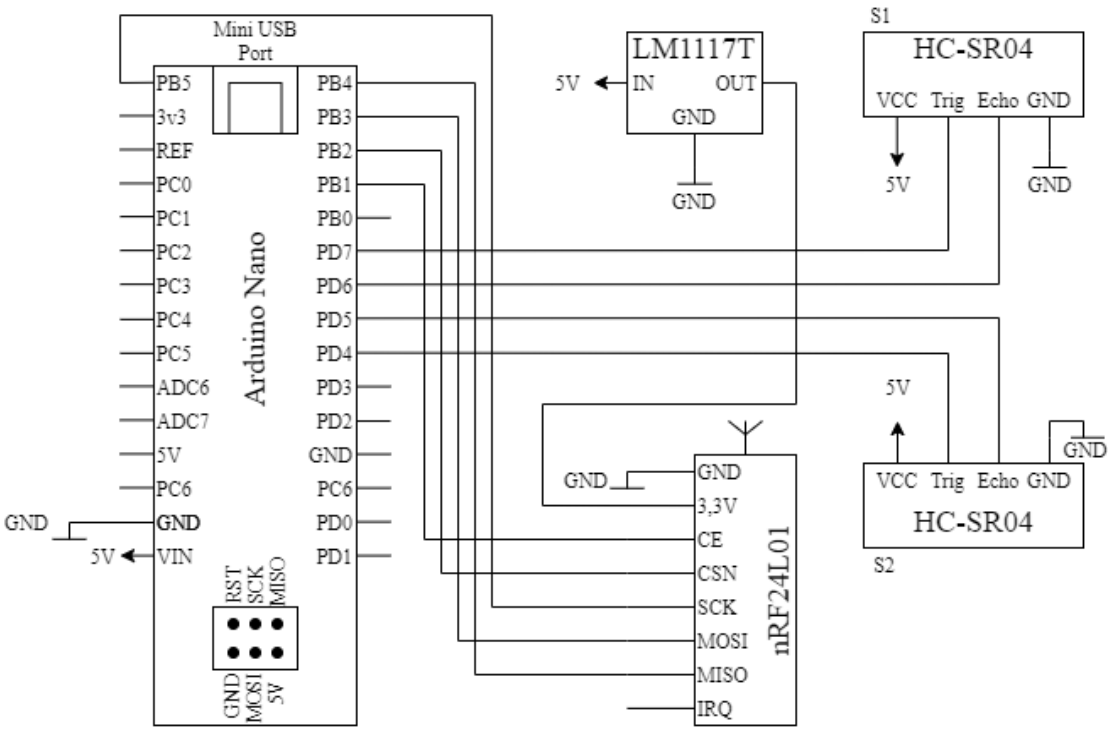


Рисунок 4.1.7.1 – Схема соединения модели устройства мониторинга

Микроконтроллер подключается к радиомодую nRF24L01 по SPI интерфейсу по соответствующим пинам на каждой плате. Для Atmega328P – это пины PB3-PB5. Пины радиомдуля CE и CSN подключаются к выводам платы PB1 и PB2 соответственно. Они нужны для выбора режима передачи данных и для выбора ведомого SPI.

Дальномеры HC-SR04 могут подключаться к любым выводам микроконтроллера. В данном случае выбрано подключение Trig-PD7 и Echo-PD6 для первого датчика и Trig-PD5 и Echo-PD4 для второго. Trig – это вывод сигнала входа, а Echo – это сигнал выхода.

Собранная модель устройства представлена на рисунках 4.1.7.2 и 4.1.7.3.

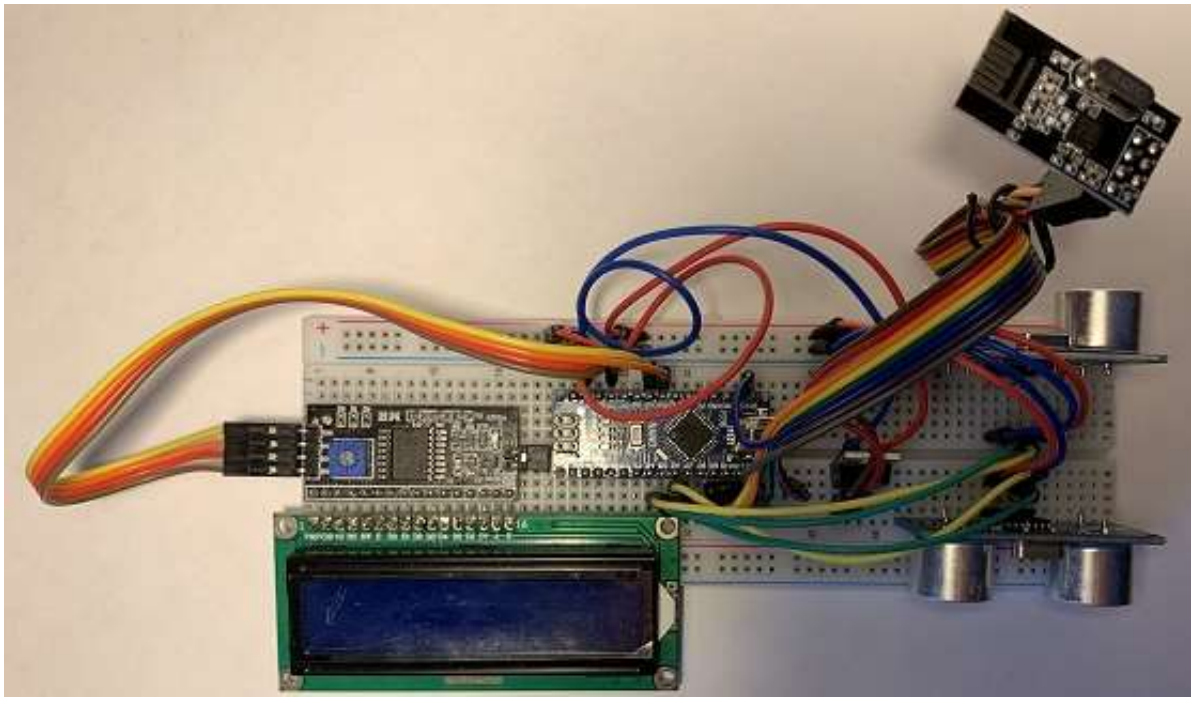


Рисунок 4.1.7.2 – Фотография макета устройства мониторинга сверху

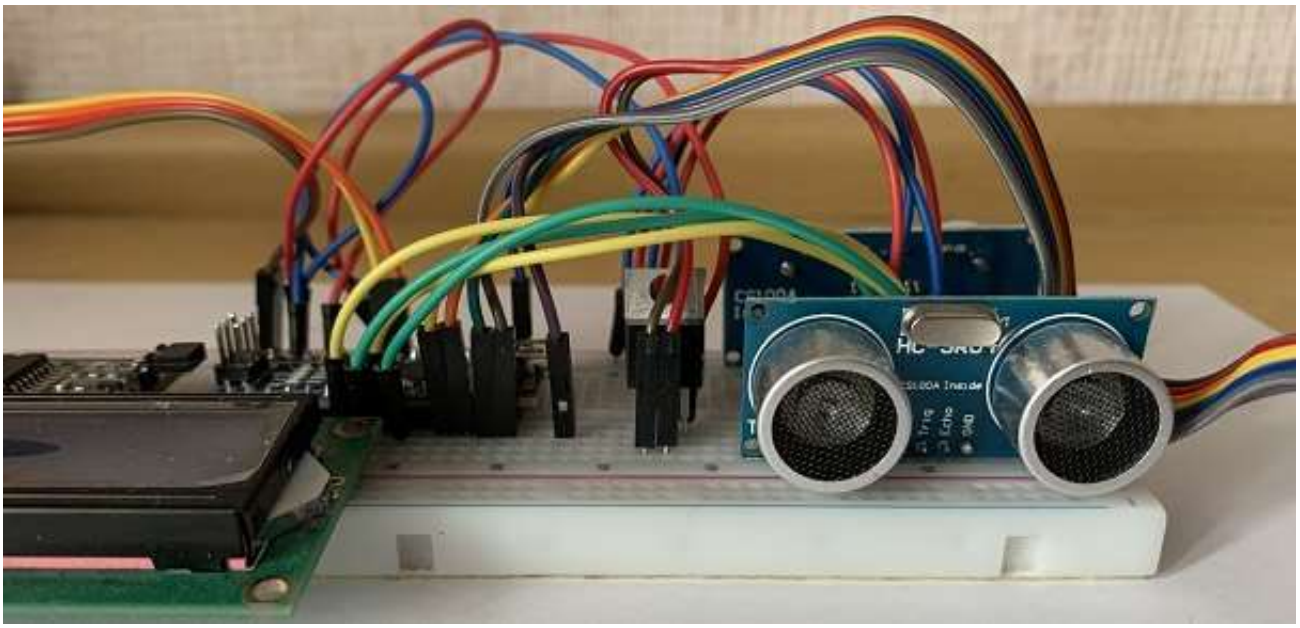


Рисунок 4.1.7.3 – Фотография макета устройства мониторинга сбоку

Хаб подключается к микроконтроллеру Atmega по последовательному интерфейсу UART (рис 4.1.7.4).

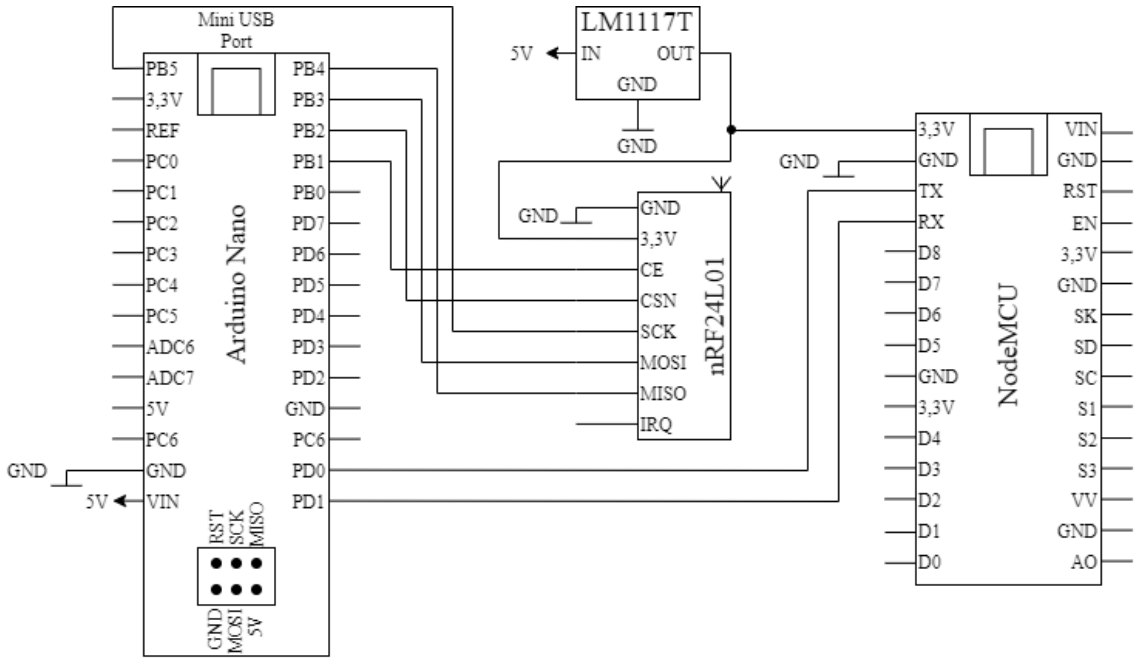


Рисунок 4.1.7.4 – Схема соединения модели устройства хаба

Собранная модель устройства представлена на рисунке 4.1.7.5.

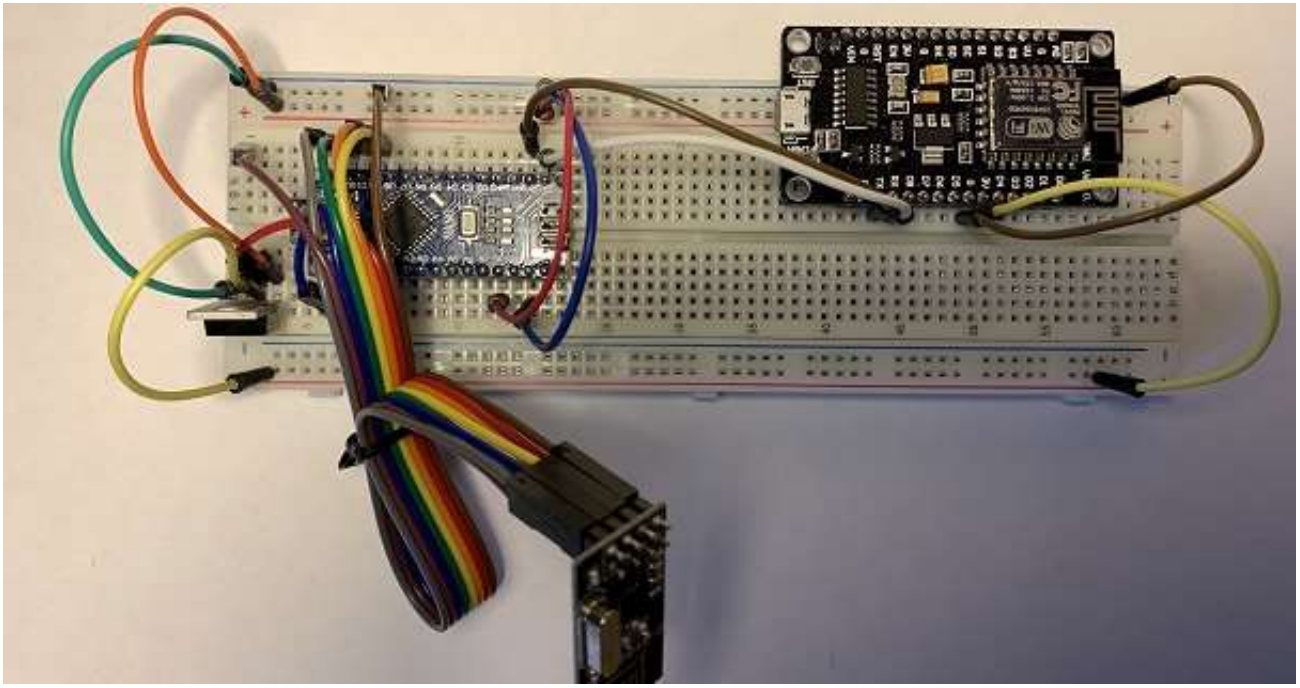


Рисунок 4.1.7.5 – Фотография макета хаба

Семисегментный индикатор в информационном табло подключается следующим образом: аноды сегментов (A, B, C, D, E, F, G) подключаются к пинам МК PD6-PD0. Катоды разрядов (D1, D2, D3, D4) присоединится к пинам МК PC0-PC3. Таким образом порт D управляет подсветкой сегментов, а через порт C осуществляется переключение подсвечиваемого разряда. Это переключение происходит с неощутимой для человеческого глаза частотой около 100 Гц, что позволяет не замечать мерцаний. Электрическая схема устройства показана на рисунке 4.1.7.6, а фотография макета в сборе на рисунке 4.1.7.7.

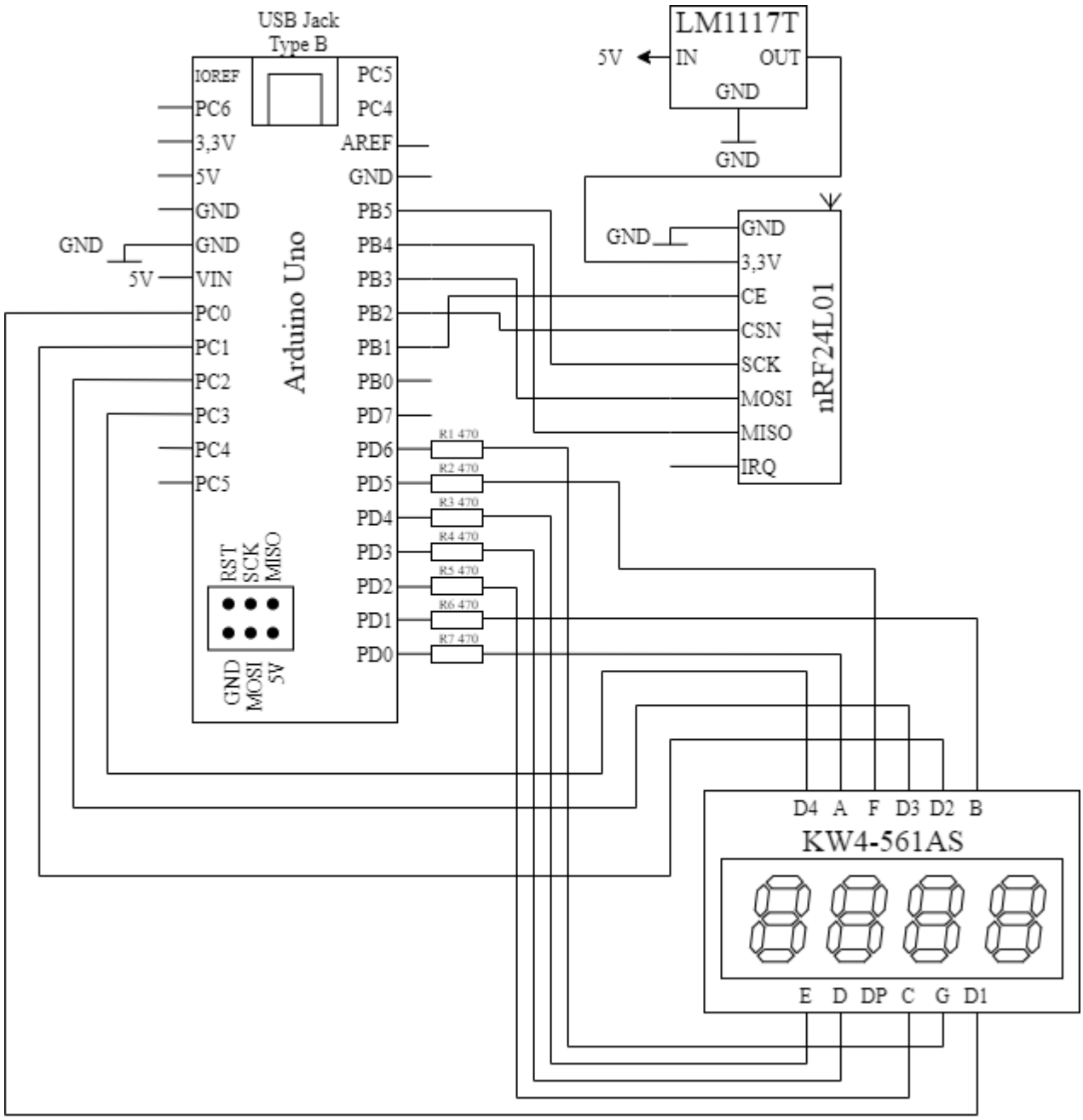


Рисунок 4.1.7.6 – Схема соединения для модели устройства информационного табло

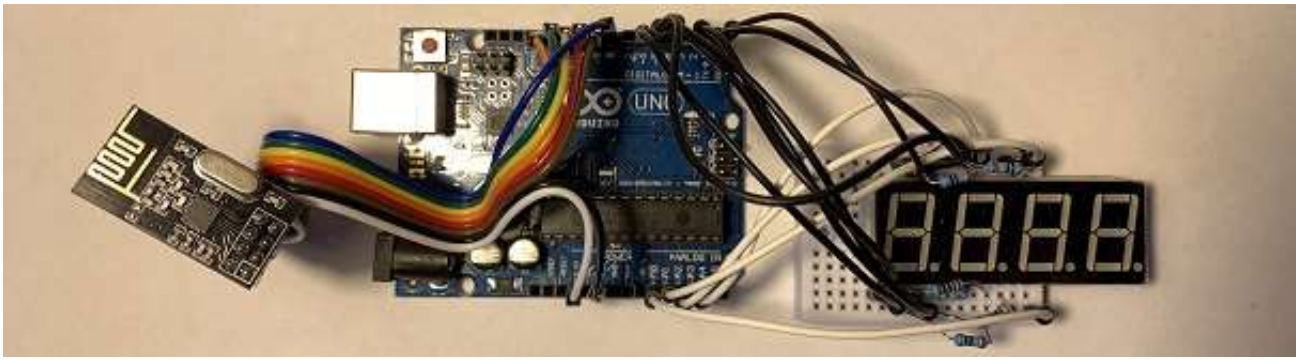


Рисунок 4.1.7.7 – Фотография макета информационного табло

4.2. ПРОГРАММИОВАНИЕ УСТРОЙСТВ

Для программирования микроконтроллеров серии ATmega на чистом языке C без загрузчика, который используется в Ардуино, требуется использование программатора. В рамках создания макета использовался USBasp (рисунок 4.2.1). Он работает по интерфейсу SPI и удобен тем, что его можно подключать через переходник к плате Ардуино по вынесенному отдельно разъему ICSP, который дублирует пины SPI и питания.

Рисунок 4.2.1 – Программатор USBasp

Для загрузки прошивки была использована программа от разработчика Боднара Сергея AVRDUDE_PROG [22]. Эта утилита является оболочкой для avrdudue. Интерфейс показан на рисунке 4.2.2.

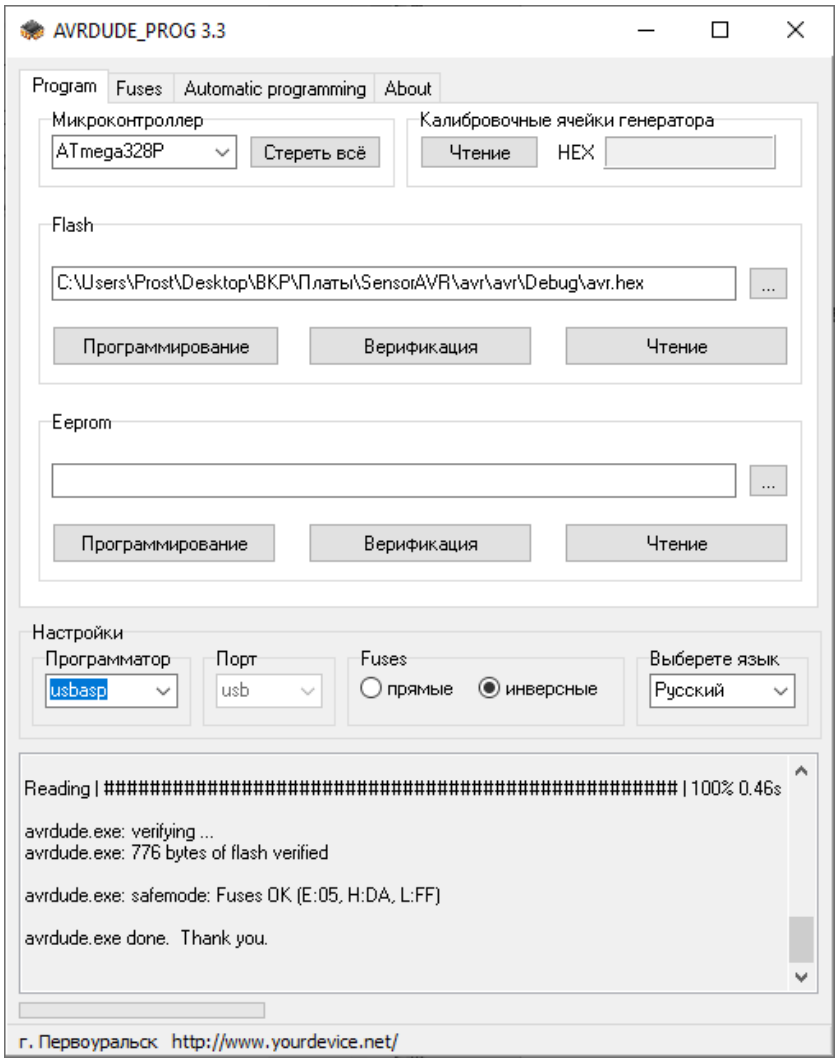


Рисунок 4.2.2 – Программа Avrdudeprog

Написание программ для микроконтроллеров ATmega осуществлялось в бесплатной среде разработки Microchip Studio for AVR® (рисунок 4.2.3).

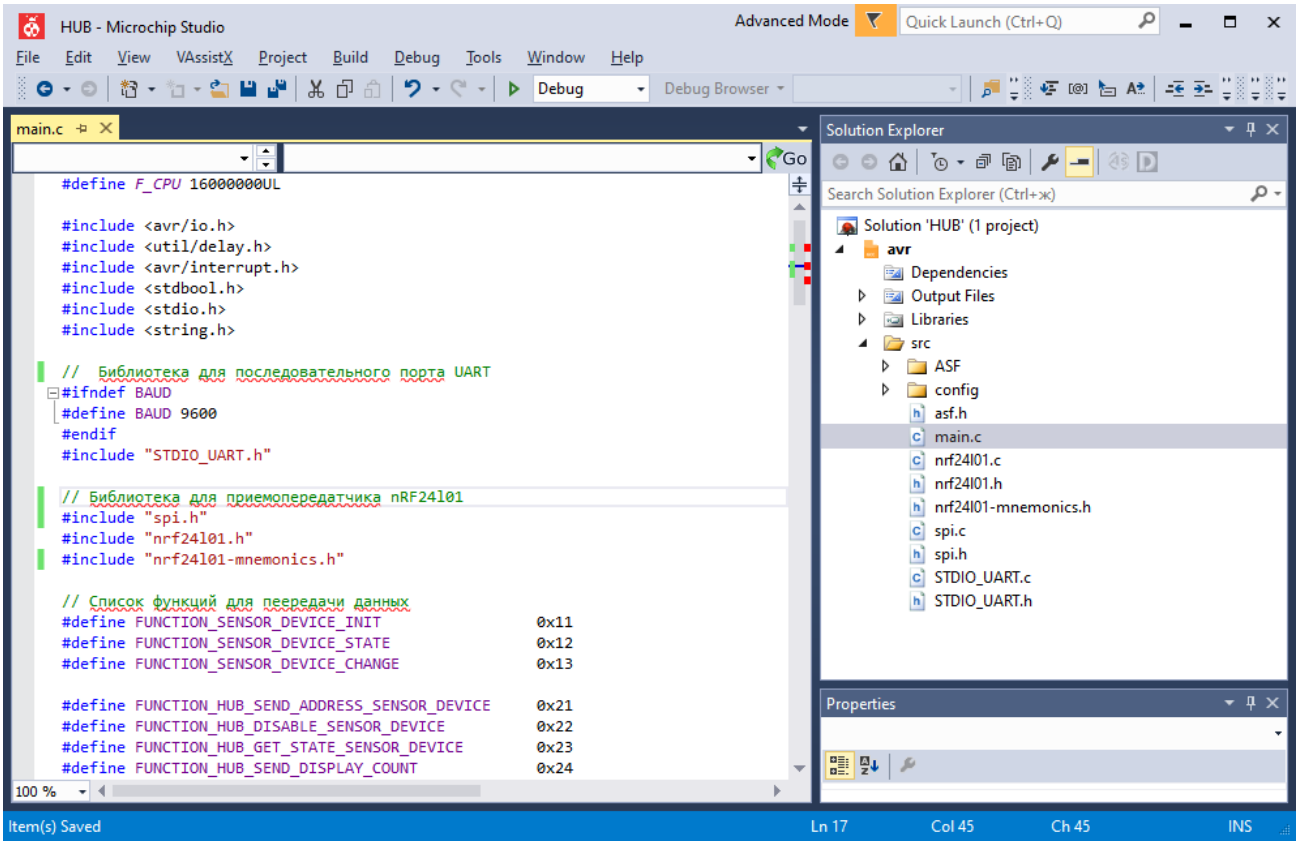


Рисунок 4.2.3 – Интерфейс программы Microchip Studio

При написании кода для микроконтроллеров серии ATmega использовались следующие сторонние библиотеки: «nrf24l01» и «spi» для работы приемопередатчика, «STDIO_UART» для передачи данных МК ATmega к ESP8266. Код устройства мониторинга, хаба и информационного табло соответственно представлен в листинге А.1. приложения А, в листинге Б.1. приложения Б и в листинге В.1. приложения В.

Написание кода для МК ESP8266 осуществлялось в программе Arduino IDE (рисунок 4.2.4).

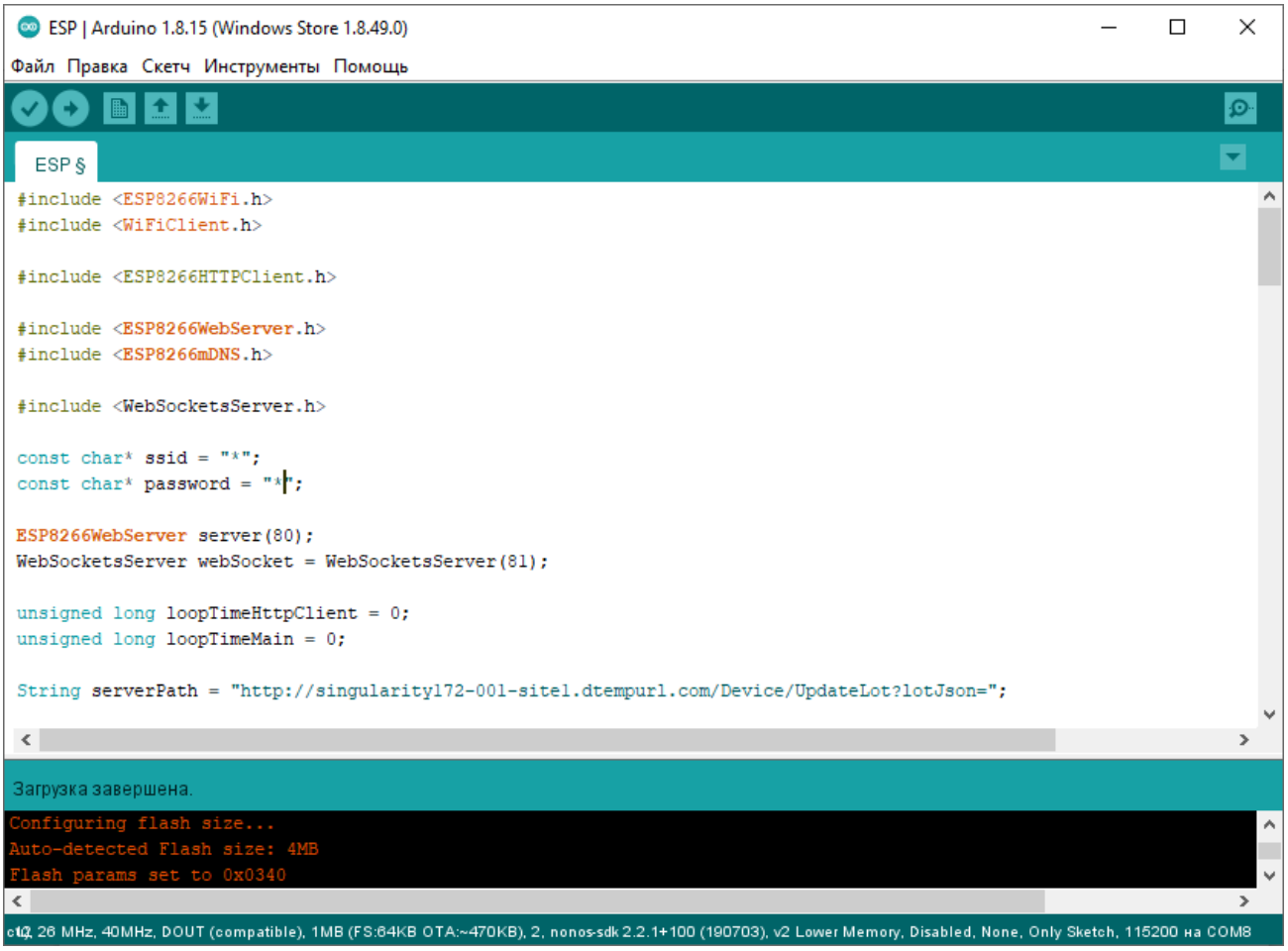


Рисунок 4.2.4 – Интерфейс программы Arduino IDE

Для реализации сервера и клиента на микроконтроллере ESP8266 в среде разработки Arduino IDE использовались следующие библиотеки: «ESP8266WiFi» и «WiFiClient» для работы МК с Wi-Fi; «ESP8266HTTPClient» для создания клиентских запросов к серверу; «ESP8266WebServer», «ESP8266mDNS» и «WebSocketsServer» для создания веб-сервера.

Код модуля представлен в листинге Г.1. приложения Г.

5. ТЕСТИРОВАНИЕ

5.1. МЕТОДОЛОГИЯ ТЕСТИРОВАНИЯ

В рамках данной работы были реализованы только простые модели устройств на макетных платах с использованием лишь двух датчиков приближения на два отдельных парковочных места, что не позволяет проводить полноценные эксперименты в реальных условия. По этой причине возможно проведение только лишь альфа-тестирования с приближенными к реальности условиями.

При тестировании учитывались такие параметры, как правильность и точность определения состояния места и временные задержки между изменением состояния места и отражением количества свободных позиций на экране информационного табло.

5.2. ПРОВЕДЕНИЕ ПРОЦЕДУРЫ ТЕСТИРОВАНИЯ

Модель устройства определения занятости была снабжена LCD дисплеем. На него выводится значения расстояний в сантиметрах, которые определяются с помощью датчиков приближения. Первая строка дисплея — это значения верхнего датчика, а вторая строка – это, соответственно, значения нижнего датчика. Помимо это, на дисплей для каждого датчика на соответствующую строку выводится статус занятости, который был определен микроконтроллером. Место считается занятым, если считанное расстояние меньше, чем 10 см. Это состояние на дисплеи обозначается символом «x».

На рисунке 5.2.1 можно увидеть точность расчета дальномера.

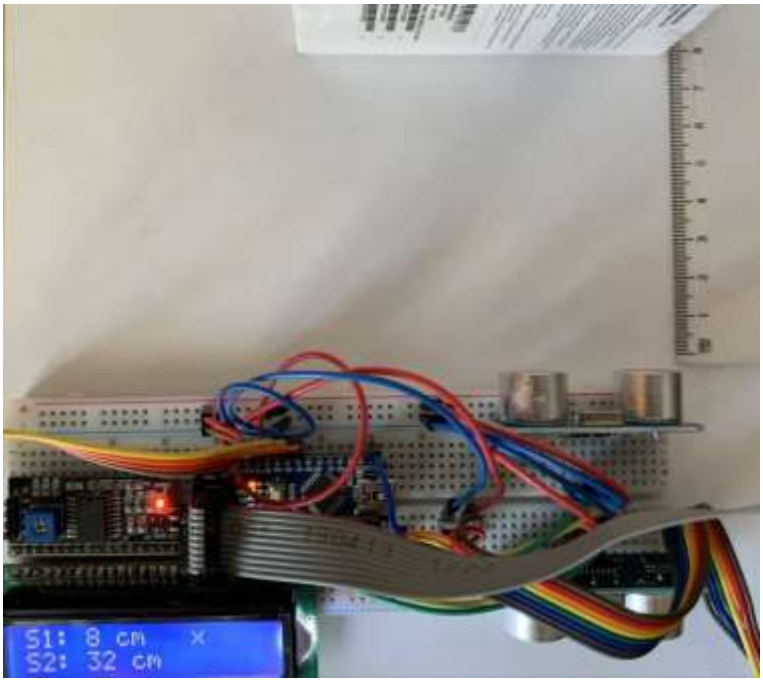


Рисунок 5.2.1 – Фотография устройства мониторинга

Теперь протестируем работу системы для двух парковочных мест. Всю информацию о протекающих в ней процессах можно видеть на веб-странице, которая генерируется микроконтроллером ESP8266. Данный модуль и браузер соединяются по протоколу WebSocket, что позволяет мгновенно передавать данные между устройствами и не требует для этого от пользователя перезагрузки страницы.

На рисунке 5.2.2 показаны сообщения, которые он отправляет. В первой колонке указано время, прошедшее с начала включения веб-сервера. Во второй колонке в квадратных скобках указано, какая подсистема отправляет сообщение. Может быть либо хаб (Serial Hub), либо веб-сервер (Web Server). Новые сообщения появляются сверху.

В данном случае сначала хаб сообщил, что он был включен в работу системы. Затем был получен запрос (Request) на инициализацию от информационного табло (Display) с адресом 0xa1. Он указывается в скобках

после названия устройства. На этот запрос был отправлен ответ, который разрешает подключение и, после этого, информационное табло отсылает подтверждение об успешной инициализации.

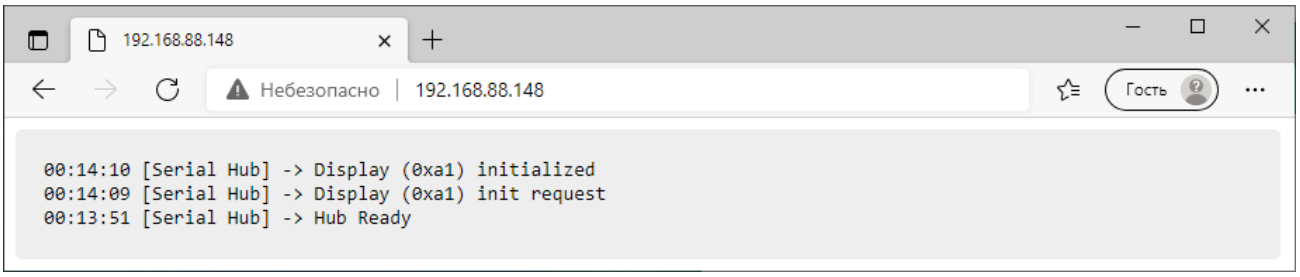


Рисунок 5.2.2 – Скриншот лога об инициализации хаба и дисплея

Подключение устройств мониторинга работает схожим образом (5.2.3), но, кроме этого, хаб получает информацию о состоянии занятости тех парковочных мест, к которым привязаны данные датчики и пересылает их сервису, который показывает состояния стоянок на карте. Данные (Data) пересылаются через веб-сервер модуля ESP8266 в формате JSON (листинг 5.2.1).

Листинг 5.2.1 – JSON запись об изменении состояния занятости места

```
{
  "parking_id": 3,
  "ParkingPlaces": [
    {
      "place_id": 93,
      "place_is_free": true,
      "park_id": 3
    }
  ]
}
```

Ответ (Response) от внешнего сервера также попадает в лог. В данном случае состояние занятости сервиса не было обновлено.

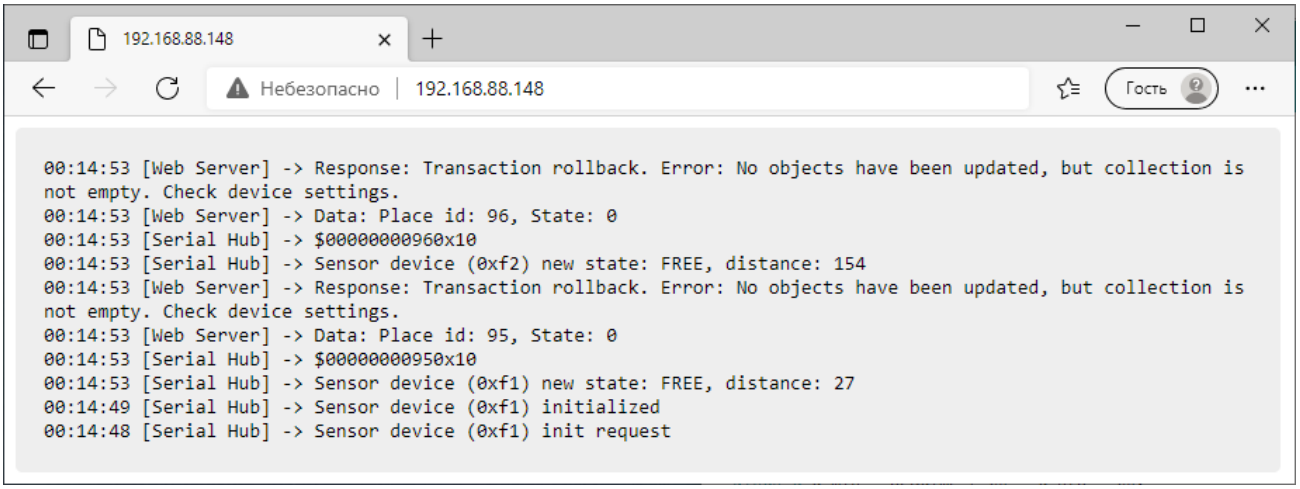


Рисунок 5.2.3 – Скриншот лога об инициализации хаба и дисплея

На рисунке 5.2.4 можно увидеть работу устройства определения занятости и работу семисегментного индикатора в момент, когда оба датчика определяют свободное состояние двух мест.

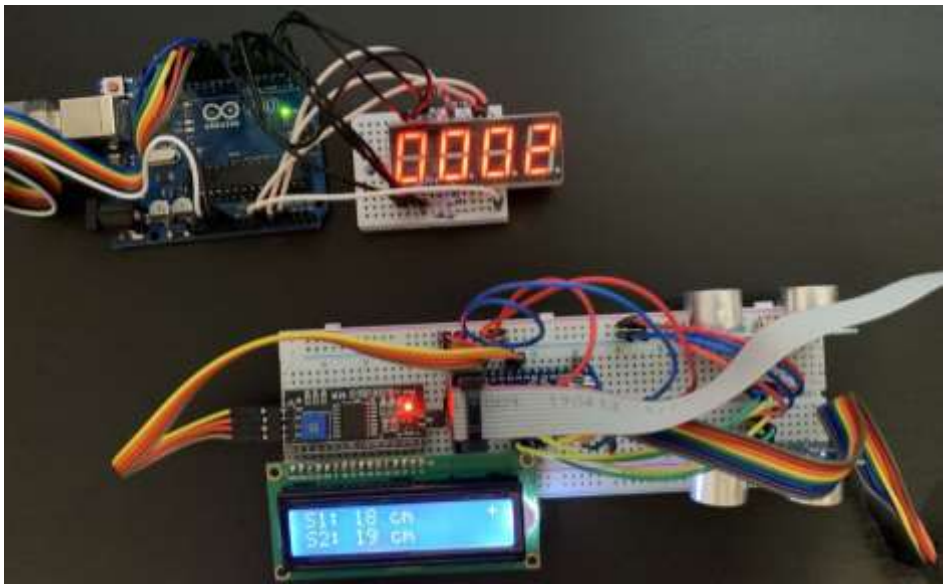


Рисунок 5.2.4 – Свободны 2 парковочных места

На рисунке 5.2.5 показан интерфейс сервиса, с картой, на которой отражаются свободные и заняты места. В данном случае к сервису подключены места 95 и 96. На карте это 2-я и 3-я зеленая точка сверху.

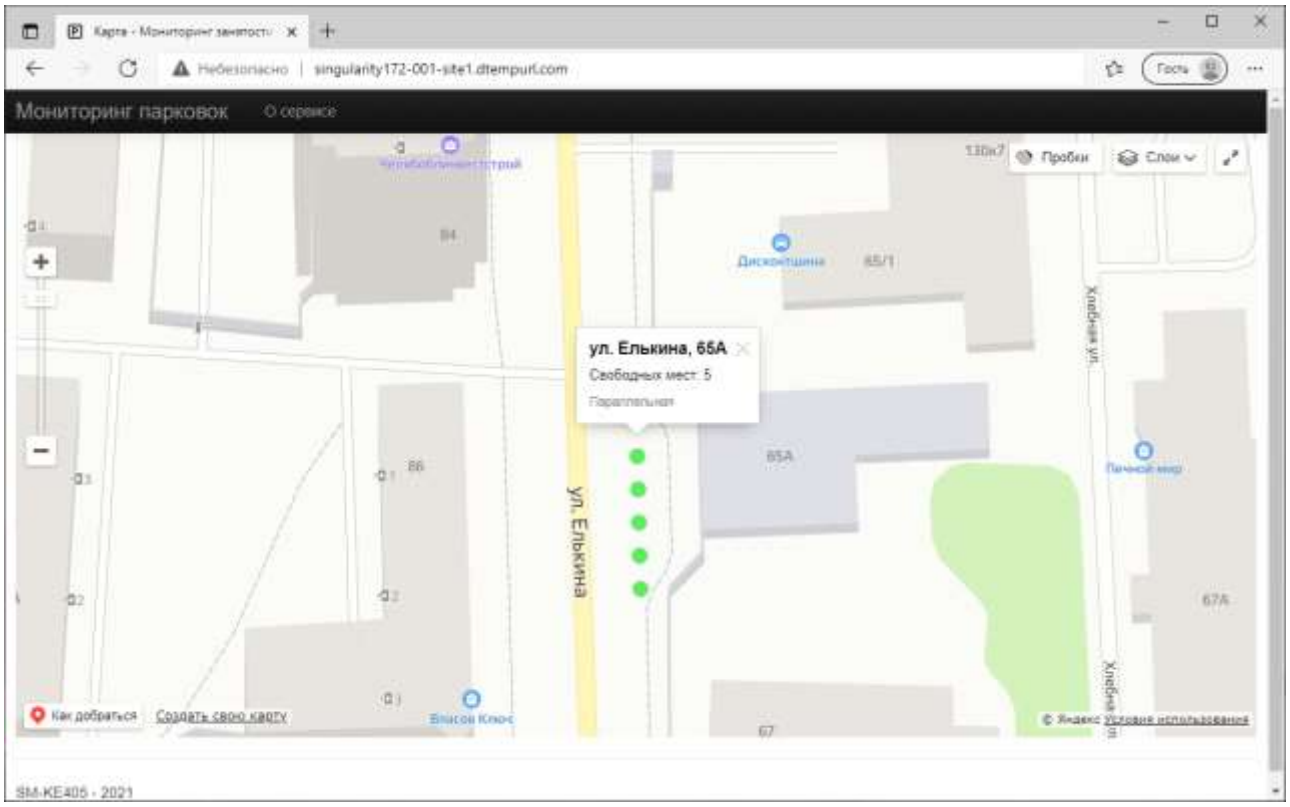


Рисунок 5.2.5 – Скриншот сервиса с картой

Теперь изменим состояние занятости одного места, эмулируя заезд на парковочную позицию сверху. Для этого в роли транспортного средства выступает небольшая коробка, которая выставляется вблизи датчика приближения (рисуснок 5.2.6).

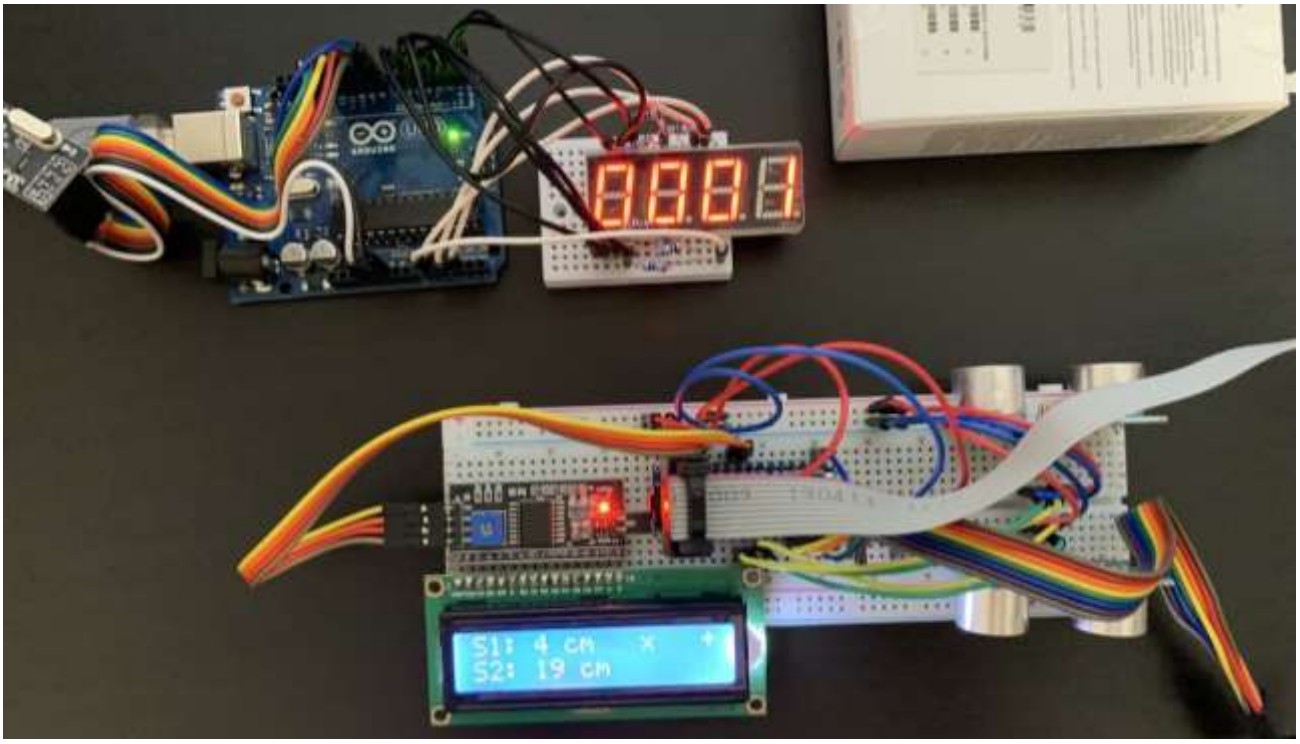


Рисунок 5.2.6 – Фотография эмуляции занятого места

На информационном табло обновилось количество свободных мест. А в логах можно наблюдать соответствующие сообщения об изменении статуса и пересылки данных сервису (рисунок 5.2.7).

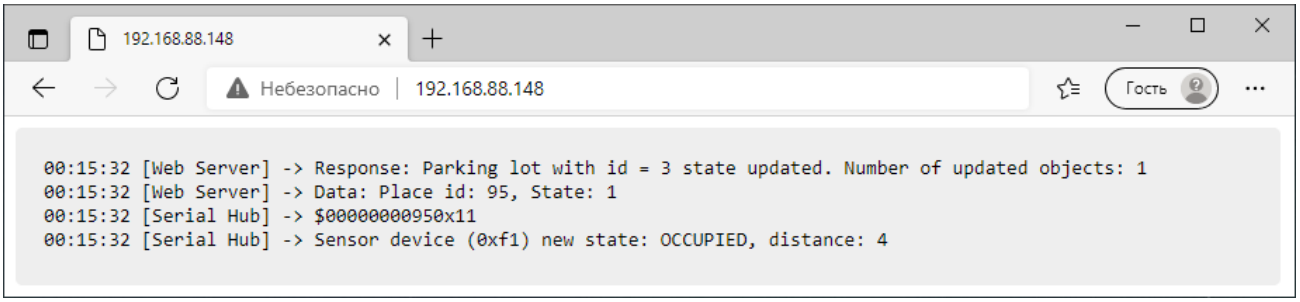


Рисунок 5.2.7 – Скриншот окна с логами об изменении статуса места

На карте также появилось обновление (рисунок 5.2.8).

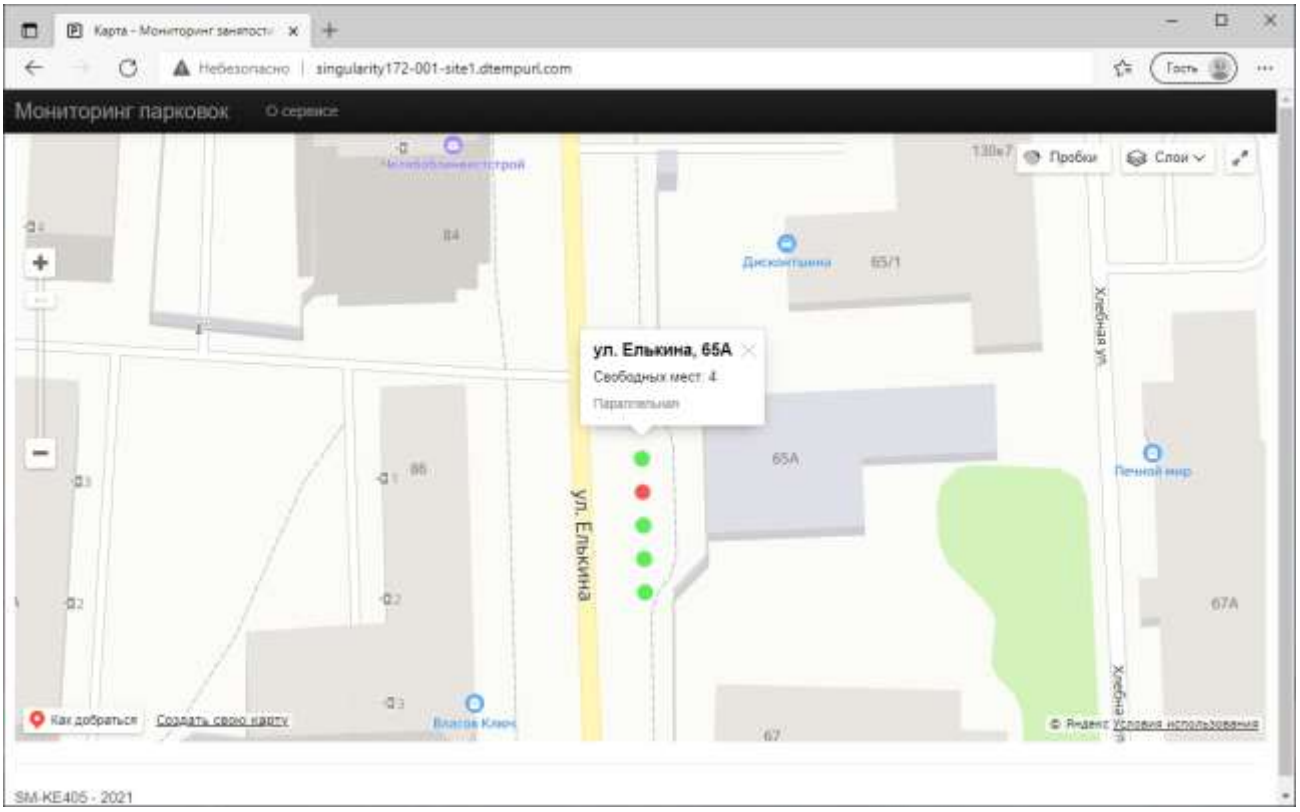


Рисунок 5.2.8 – Скриншот сервиса с картой

Воспользуемся еще одним парковочным местом и также сымитируем парковку ТС на стоянке, установив коробку рядом с датчиком снизу (рисунок 5.2.9).

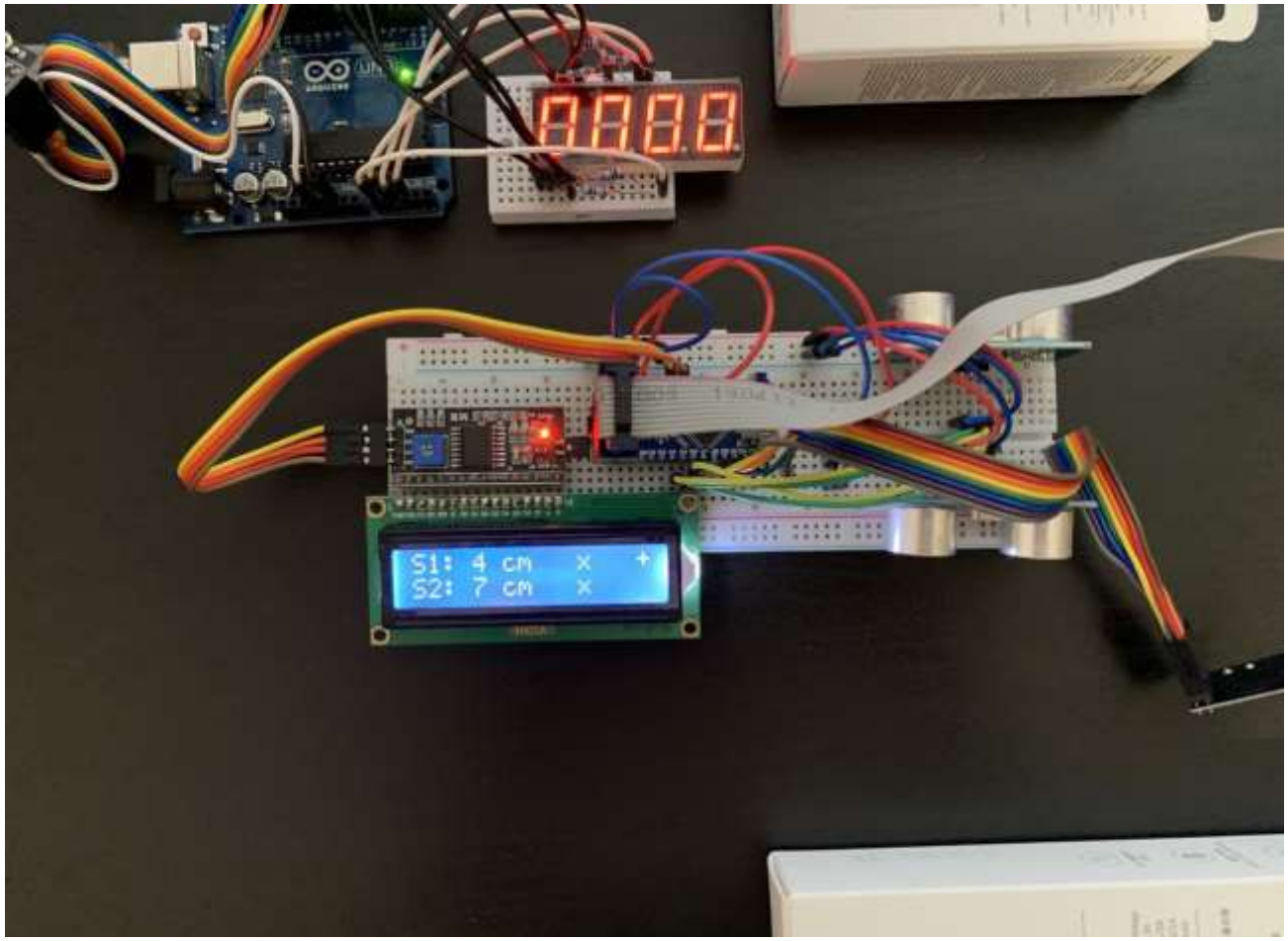


Рисунок 5.2.9 – Фотография эмуляции двух занятых мест

В логах, на дисплеи и на карте появились изменения (рисунок 5.2.10, 5.2.11).

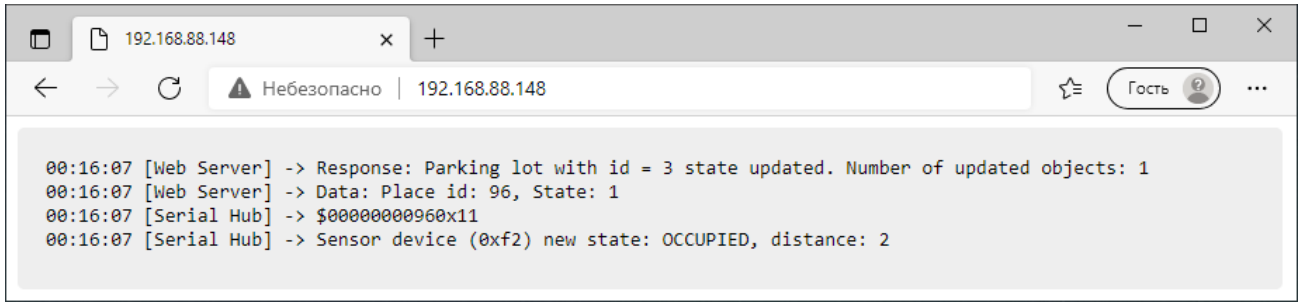


Рисунок 5.2.10 – Скриншот окна с логами об изменении статуса места

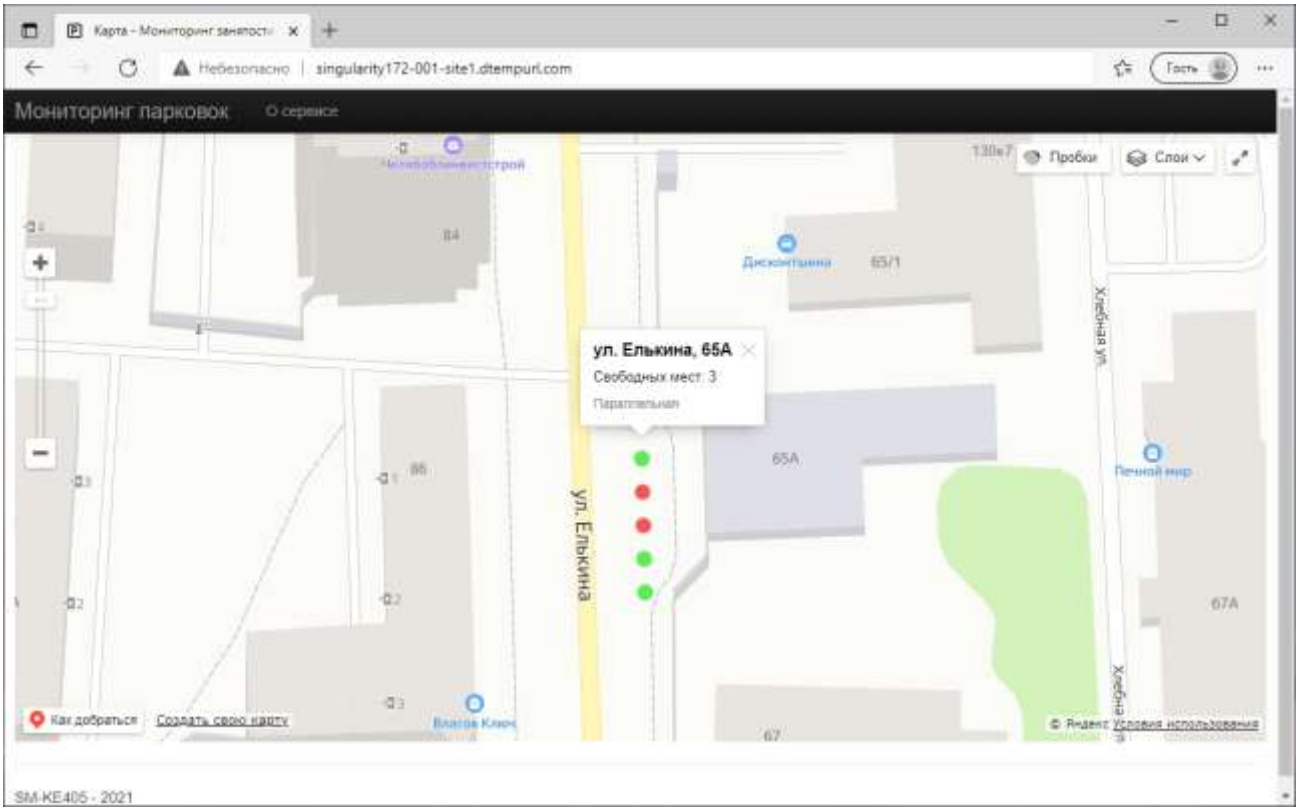


Рисунок 5.2.11 – Скриншот сервиса с картой

Освободим верхнее место, которые было занято в самом начале эксперимента (рисунок 5.2.12).

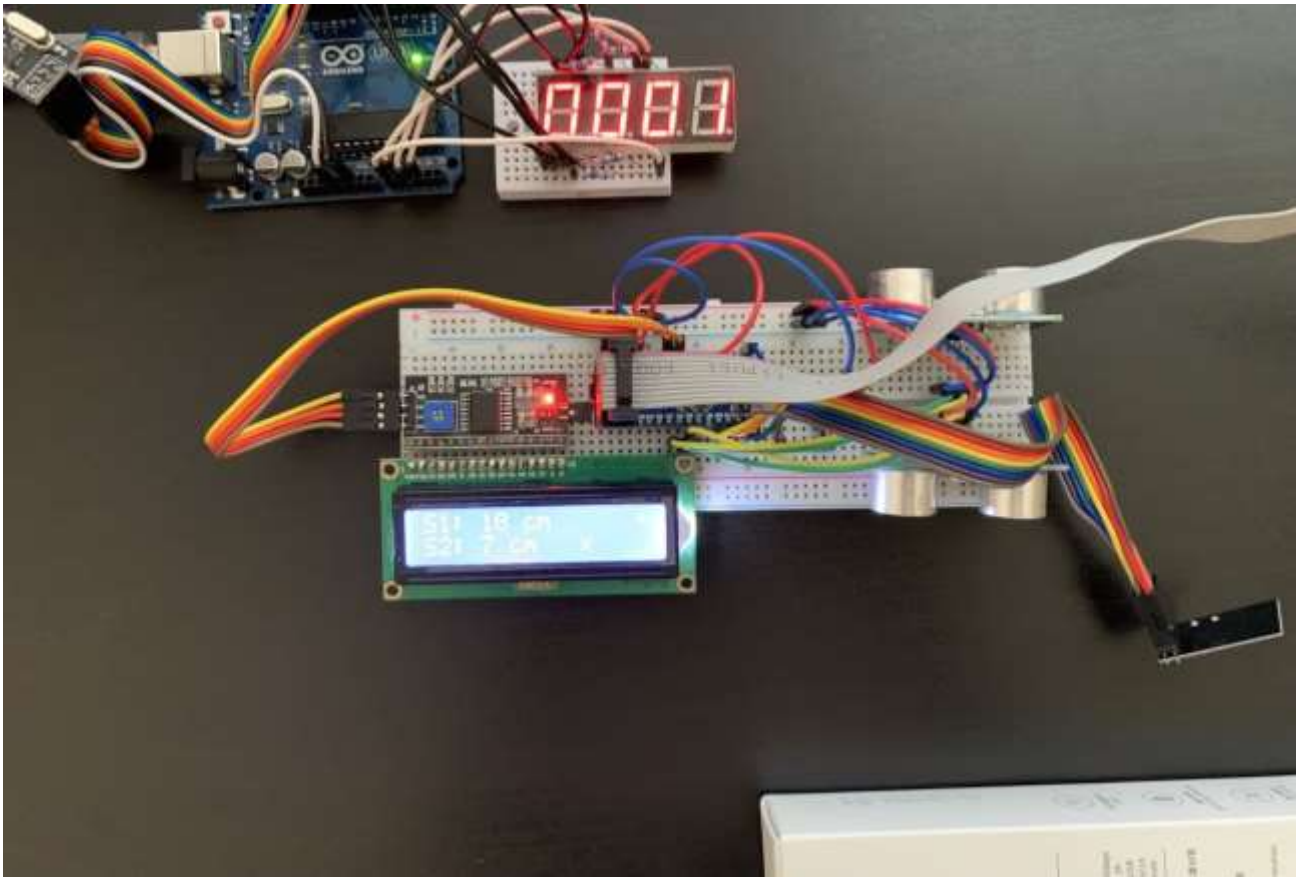


Рисунок 5.2.12 – Фотография эмуляции освобождения места

В логах, на дисплеи и на карте появились изменения (рисунок 5.2.13, 5.2.14).

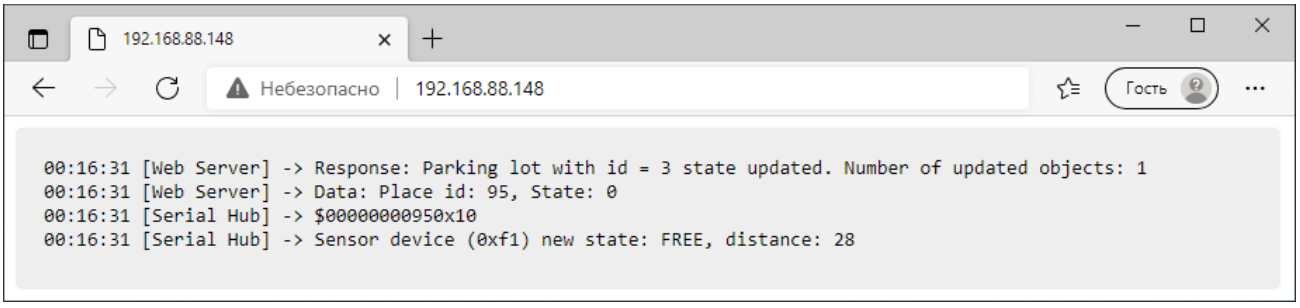


Рисунок 5.2.13 – Скриншот окна с логами об изменении статуса места

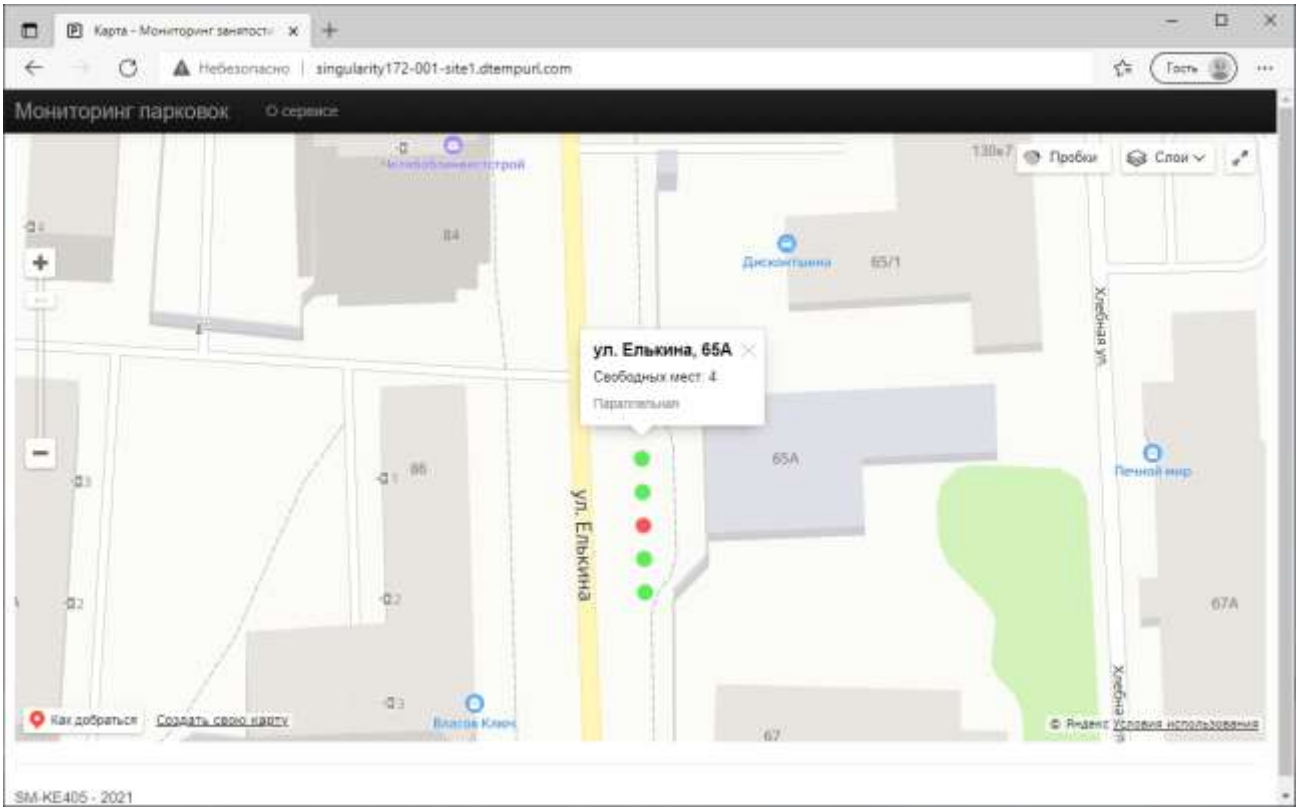


Рисунок 5.2.14 – Скриншот сервиса с картой

Данные тесты показали, что определение занятости парковочных мест работает без ошибок. А измеренные в процессе эксперимента временные задержки между изменением состояния места и отражением количества свободных позиций на экране информационного табло составляют не более 2-х секунд.

6. ЗАКЛЮЧЕНИЕ

В данной выполненной выпускной квалификационной работе были рассмотрены и проанализированы существующее решения по мониторингу парковочных мест. Анализ отразил почти полное отсутствие предложения подобных систем на отечественном ранке, а зарубежные системы являются недоступными или крайне дорогими в России.

В итоге было проведено проектирование собственного решения мониторинга занятости, которое отличается своей простотой и дешевизной. Однако при реализации не удалось достичь всех целей, которые были поставлены на этапе проектирования из-за недостаточности финансовой составляющей, которые накладывают ограничение на качество и на количество доступных компонентов. Можно выделить самые главные недостатки, которые следуют из этого:

– не удалось собрать макеты двух и более парковочных мест;
– были использованы радиомодули nRF24L01, которые имеют ограничение в 6 устройств, подключённых одновременно, и могу работать на дистанции лишь до 100 м.

Полученная модель системы в итоге прошла альфа-тестирование. Важно, что были проработаны многие аспекты архитектуры системы и алгоритмы её алгоритмы, которые могут работать с другими, более качественными компонентами.

В дальнейшем следует перевести систему на микроконтроллеры STM, поменять модули и протоколы приемопередачи, нарастить количество устройств определения занятости, а также провести полноценное бета-тестирование в реальных условиях.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Официальный сайт компании Интерсвязь. – https://www.is74.ru. Дата обращения: 09.04.2021.
2. SENSIT parking sensors. – https://www.nedapidentification.com/products/sensit/. Дата обращения: 09.04.2021.
3. Smart Parking Lot Sensor | Bosch Connected Devices and Solutions. – https://www.bosch-connectivity.com/products/connected-mobility/parking-lot-sensor. Дата обращения: 09.04.2021.
4. Ч. Платт, Ф. Янссон. Энциклопедия электронных компонентов. Том 3. Датчики местоположения, присутствия, ориентации – BHV, 2017 – 288 с.;
5. Классификация датчиков, основные требования к ним. – http://electrolibrary.info/subscribe/sub_16_datchiki.htm. Дата обращения: 13.04.2021.
6. Лазерный дальномер. – https://ru.wikipedia.org/wiki/%D0%9B%D0%B0%D0%B7%D0%B5%D1%80%D0%BD%D1%8B%D0%B9_%D0%B4%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%BC%D0%B5%D1%80. Дата обращения: 21.04.2021.
7. Датчик расстояния лазерный VL53L0X (GY-530). – https://mcustore.ru/store/datchiki-i-sensory/datchik-rasstoyaniya-lazernyj-vl53l0x-gy-530. Дата обращения: 21.04.2021.
8. Бесконтактные датчики приближения. – http://www.maxplant.ru/article/proximity_sensor.php. Дата обращения: 21.04.2021.

9. How Does an Ultrasonic Sensor Work? – https://www.teachengineering.org/lessons/view/umo_sensorswork_lesson06. Дата обращения: 21.04.2021.
10. Датчик расстояния ультразвуковой HC-SR04. – https://mcustore.ru/store/datchiki-i-sensory/datchik-rasstoyaniya-ultrazvukovoj-hc-sr04. Дата обращения: 21.04.2021.
11. Размер парковки для легкового автомобиля. – https://2ann.ru/razmer-parkovki-dlya-legkovogo-avtomobilya. Дата обращения: 21.04.2021.
12. Top wireless standards for IoT devices. – https://iot.eetimes.com/top-wireless-standards-for-iot-devices. Дата обращения: 21.04.2021.
13. Радио модуль NRF24L01. – https://3d-diy.ru/wiki/arduino-moduli/radio-modul-nrf24l01/. Дата обращения: 22.04.2021.
14. NB IoT. – https://ru.wikipedia.org/wiki/NB_IoT. Дата обращения: 22.04.2021.
15. Языки программирования микроконтроллеров. – https://mcucpu.ru/index.php/soft/42-lmcu/67-programmlang. Дата обращения: 23.05.2021.
16. Как программируют Arduino. – https://thecode.media/arduino-code/. Дата обращения: 26.05.2021.
17. ATmega328P DATASHEET. – https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf. Дата обращения: 27.05.2021.
18. Ultrasonic Ranging Module HC-SR04. – https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf. Дата обращения: 27.05.2021.

19. nRF24L01+ DATASHEET. – https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf. Дата обращения: 27.05.2021.
20. ESP8266EX Datasheet. – https://cdn-shop.adafruit.com/product-files/2471/0A-ESP8266__Datasheet__EN_v4.3.pdf. Дата обращения: 27.05.2021.
21. Индикатор цифровой 0.56" 4-digit KEM-5461AR (HSN-5643AS-H красный общ. Катод/-). – https://procontact74.ru/26-osveshenie-i-indikaciya/268-cifrovye-indikatory/indikator-cifrovoj-kem-5461ar-red-katod-26041. Дата обращения: 27.05.2021.
22. ПРОГРАММА ДЛЯ ПРОГРАММИРОВАНИЯ AVR - AVRDUDE_PROG. – http://yourdevice.net/proekty/avrdude-prog. Дата обращения: 27.05.2021.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ УСТРОЙСТВА ОПРЕДЕЛЕНИЯ ЗАНЯТОСТИ

Листинг А.1 – Исходный код программы устройства мониторинга

```
#define F_CPU 16000000UL

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>

// Библиотека nRF24L01+
#include "nrf24l01.h"
#include "nrf24l01-mnemonics.h"
#include "spi.h"

// LCD дисплей (для отладки)
#include "hd44780pcf8574.h"
void hd44780_init(void);

// Константы, функции и переменные ультразвукового датчика

#define US_DDR                          DDRD
#define US_PORT                         PORTD
#define US_PIN                          PIND
#define US_A_TRIGER             PIND4
#define US_A_ECHO               PIND5
#define US_B_TRIGER             PIND6
#define US_B_ECHO               PIND7

#define US_DISTANCE_PARKING10
#define US_MAX_DISTANCE         200
#define US_SENSOR_A             0
#define US_SENSOR_B             1

void usound_init(void);
uint32_t readDistance(int);
uint16_t us_value[2] = { US_MAX_DISTANCE, US_MAX_DISTANCE };
uint16_t us_value_prev[2] = { US_MAX_DISTANCE, US_MAX_DISTANCE };
uint8_t us_state[2] = { 0, 0 };
uint8_t us_state_wait_sent[2] = { 1, 1 };

// Список функций для передачи данных
#define FUNCTION_SENSOR_DEVICE_INIT                             0x11
#define FUNCTION_SENSOR_DEVICE_STATE                    0x12
#define FUNCTION_SENSOR_DEVICE_CHANGE                   0x13
```

```
#define FUNCTION_HUB_REQUEST_RESET                                  0x20
#define FUNCTION_HUB_SEND_ADDRESS_SENSOR_DEVICE       0x21
#define FUNCTION_HUB_DISABLE_SENSOR_DEVICE                  0x22
#define FUNCTION_HUB_GET_STATE_SENSOR_DEVICE          0x23

// Уникальный адрес устройства
const uint8_t device_addressA = 0xF1;
const uint8_t device_addressB = 0xF2;
// Адрес хаба, к которому назначено устройство
uint8_t linked_hub_address = 0x00;

// Структура пакета данных
typedef struct {
	uint8_t function_code;
	uint8_t device_address;
	uint8_t hub_address;
	uint8_t data;
} RegularMsg;

// Функция подтверждения инициализации устройства
void apply_init(void);

int main (void) {

	/* ****************************************** */
	//     Инициализация приемопередатчика

	nrf24_init();
	nrf24_start_listening();

	/* ****************************************** */
	// Ининциализация первого таймера для обработки датчиков

	TCCR1B |= (1<<WGM12);	// сброс по совпадению
	TCCR1B |= (1<<CS02); // предделитель

	OCR1A = 31250;
	TIMSK1 |= (1<<OCIE1A);

	TCNT1 = 0;


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

	// Инициализция дисплея
	hd44780_init();

	// Инициализация датчиков
	usound_init();

	sei();


	while(1) {

		// Если устройство не подключено к хабу, отправляется запрос на инициализацию
		if(linked_hub_address == 0x00) {
```

```
                        RegularMsg hubMsg;
                        hubMsg.function_code = FUNCTION_SENSOR_DEVICE_INIT;
                        hubMsg.device_address = device_addressA;
                        hubMsg.hub_address = 0x00;
                        hubMsg.data = 0x00;

                        char tx_message[sizeof(RegularMsg)];
                        memcpy(tx_message, &hubMsg, sizeof hubMsg);


                        nrf24_send_message(tx_message);

                        _delay_ms(1000);
                }


                // Пересылка данных с датчиков приближения
                if(linked_hub_address != 0x00) {
                        for(uint8_t i = 0; i < 2; i++) {
                                if(us_state_wait_sent[i] != 0) {
                                        RegularMsg hubMsg;
                                        hubMsg.function_code = FUNCTION_SENSOR_DEVICE_CHANGE;

                                        if(i == US_SENSOR_A) hubMsg.device_address =
device_addressA;
                                        if(i == US_SENSOR_B) hubMsg.device_address =
device_addressB;

                                        hubMsg.hub_address = linked_hub_address;
                                        hubMsg.data = us_value[i];

                                        char tx_message[sizeof(RegularMsg)];
                                        memcpy(tx_message, &hubMsg, sizeof hubMsg);

                                        nrf24_send_message(tx_message);

                                        us_state_wait_sent[i] = 0;
                                }
                                _delay_ms(100);
                        }
                }

                // Обработка поступающих запросов
                if(nrf24_available()) {

                        const char *rx = nrf24_read_message();

                        RegularMsg *rx_struct_msg = (RegularMsg *) rx;

                        // Получен адрес хаба
                        if(rx_struct_msg->function_code ==
FUNCTION_HUB_SEND_ADDRESS_SENSOR_DEVICE) {
                                if(rx_struct_msg->device_address == device_addressA) {
                                        linked_hub_address = rx_struct_msg->hub_address;
                                        HD44780_PCF8574_PositionXY(PCF8574_ADDRESS, 15, 0);

                                        HD44780_PCF8574_DrawString(PCF8574_ADDRESS, "+");
                                        apply_init();
```

```
                                us_state_wait_sent[0] = 1;
                                us_state_wait_sent[1] = 1;
                                _delay_ms(3000);
                        }
                }
                // Получен запрос сброса от хаба
                if(rx_struct_msg->function_code == FUNCTION_HUB_REQUEST_RESET
                && rx_struct_msg->device_address == 0x01
                && rx_struct_msg->hub_address == linked_hub_address) {
                        linked_hub_address = 0x00;
                        HD44780_PCF8574_PositionXY(PCF8574_ADDRESS, 15, 0);
                        HD44780_PCF8574_DrawString(PCF8574_ADDRESS, " ");
                }

        }

        _delay_ms(10);
    }

    return 0;
}


// Прерывание-таймер получения значений с датчиков
ISR (TIMER1_COMPA_vect)
{
    uint16_t distance;

    for(uint8_t i = 0; i < 2; i++)
    {

        distance = readDistance(i);
        if(distance < 300)
        {

                if(us_value_prev[i] <= US_DISTANCE_PARKING
                && distance <= US_DISTANCE_PARKING
                && us_state[i] != 1)
                {
                        us_state[i] = 1;
                        us_state_wait_sent[i] = 1;
                        HD44780_PCF8574_PositionXY(PCF8574_ADDRESS, 11, i);
                        HD44780_PCF8574_DrawString(PCF8574_ADDRESS, "x");
                }
                else if(us_value_prev[i] > US_DISTANCE_PARKING
                && distance > US_DISTANCE_PARKING
                && us_state[i] != 0)
                {
                        us_state[i] = 0;
                        us_state_wait_sent[i] = 1;
                        HD44780_PCF8574_PositionXY(PCF8574_ADDRESS, 11, i);
                        HD44780_PCF8574_DrawString(PCF8574_ADDRESS, " ");
                }


                char str[10];
                sprintf(str, "%d cm", distance);
                HD44780_PCF8574_PositionXY(PCF8574_ADDRESS, 4, i);
```

```
                        HD44780_PCF8574_DrawString(PCF8574_ADDRESS, "       ");
                        HD44780_PCF8574_PositionXY(PCF8574_ADDRESS, 4, i);
                        HD44780_PCF8574_DrawString(PCF8574_ADDRESS, str);

                        us_value_prev[i] = us_value[i];
                        us_value[i] = distance;
                }
        }


        TCNT1 = 0; // Сброс счетчика / рестарт таймера
}

void usound_init(void)
{
        US_DDR |= (1 << US_A_TRIGER) | (0 << US_A_ECHO);
        US_PIN |= (1 << US_A_ECHO);

        US_DDR |= (1 << US_B_TRIGER) | (0 << US_B_ECHO);
        US_PIN |= (1 << US_B_ECHO);
}

uint32_t readDistance(int sensor)
{
        uint8_t readStatus = 0;
        uint32_t disTime = 0;

        uint8_t pinTrig;
        uint8_t pinEcho;

        if(sensor == US_SENSOR_A) {
                pinTrig = US_A_TRIGER;
                pinEcho = US_A_ECHO;
        }
        else {
                pinTrig = US_B_TRIGER;
                pinEcho = US_B_ECHO;
        }

        _delay_us(2);
        US_PORT |= (1 << pinTrig);
        _delay_us(10);
        US_PORT &= ~(1 << pinTrig);


        while(readStatus == 0)
        {
                while( US_PIN  &  (1 << pinEcho) )
                {
                        disTime++;
                        readStatus = 1;
                }
        }

        return disTime * 0.01000;
}

/* Функции передатчика */
```

```
void apply_init() {
	RegularMsg hubMsg;
	hubMsg.function_code = FUNCTION_SENSOR_DEVICE_STATE;
	hubMsg.device_address = device_addressA;
	hubMsg.hub_address = linked_hub_address;
	hubMsg.data = 0x00;

	char tx_message[sizeof(RegularMsg)];
	memcpy(tx_message, &hubMsg, sizeof hubMsg);


	nrf24_send_message(tx_message);
}


/* Функции дисплея */
void hd44780_init()
{
	HD44780_PCF8574_Init(PCF8574_ADDRESS);
	HD44780_PCF8574_DisplayClear(PCF8574_ADDRESS);
	HD44780_PCF8574_DisplayOn(PCF8574_ADDRESS);
	HD44780_PCF8574_DrawString(PCF8574_ADDRESS, "S1:");
	HD44780_PCF8574_PositionXY(PCF8574_ADDRESS, 0, 1);
	HD44780_PCF8574_DrawString(PCF8574_ADDRESS, "S2:");
   }
```

ПРИЛОЖЕНИЕ Б
ИСХОДНЫЙ КОД ПРОГРАММЫ ХАБА

Листинг Б.1 – Исходный код программы хаба

```
// Частота процессора 16 мГц
#define F_CPU 16000000UL

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>

//     Библиотека для последовательного порта UART
#ifndef BAUD
#define BAUD 74880
#endif
#include "STDIO_UART.h"

// Библиотека для приемопередатчика nRF24l01
#include "spi.h"
#include "nrf24l01.h"
#include "nrf24l01-mnemonics.h"

// Список функций для пеередачи данных
#define FUNCTION_SENSOR_DEVICE_INIT                                 0x11
#define FUNCTION_SENSOR_DEVICE_STATE                        0x12
#define FUNCTION_SENSOR_DEVICE_CHANGE                       0x13

#define FUNCTION_HUB_REQUEST_RESET                                  0x20
#define FUNCTION_HUB_SEND_ADDRESS_SENSOR_DEVICE       0x21
#define FUNCTION_HUB_DISABLE_SENSOR_DEVICE                  0x22
#define FUNCTION_HUB_GET_STATE_SENSOR_DEVICE          0x23
#define FUNCTION_HUB_SEND_DISPLAY_COUNT                     0x24
#define FUNCTION_HUB_SEND_ADDRESS_DISPLAY                   0x25

#define FUNCTION_DISPLAY_GET_HUB_ADDRESS              0x31
#define FUNCTION_DISPLAY_SEND_HUB_STATE                     0x32


// Уникальный адрес данного хаба
const uint8_t hub_address = 0xB1;

// Уникальный адрес привязанного дисплея
uint8_t display_address = 0x00;

// Структура пакета данных
typedef struct {
       uint8_t function_code;
```

```
	uint8_t device_address;
	uint8_t hub_address;
	uint8_t data;
} RegularMsg;


uint8_t parking_place_1 = 95;
uint8_t parking_place_2 = 96;
uint8_t parking_place_1_state = 0;
uint8_t parking_place_2_state = 0;

#define US_DISTANCE_PARKING10

void init_hub(void);
void update_display(void);


/* NRF */

int main (void) {

	/* ************************************** */
	//     Инициализация приемопередатчика

	nrf24_init();

	/* ************************************** */
	//     Инициализация UART

	uart_init();

	/* ************************************** */
	// Ининциализация первого таймера

	//TCCR1B |= (1<<WGM12);    // сброс по совпадению
	//TCCR1B |= (1<<CS11);     // предделитель

	//OCR1A = 1000;
	//TIMSK1 |= (1<<OCIE1A);

	//TCNT1 = 0;

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

	sei();

	printf("Hub Ready\n");

	nrf24_start_listening();

	init_hub();

	while(1) {

		// Обработка поступающих запросов
		if(nrf24_available()) {

			const char *rx = nrf24_read_message();
```

```
                    RegularMsg *rx_struct_msg = (RegularMsg *) rx;

                    // Получен запрос на инициализацию устройства занятости
                    if(rx_struct_msg->function_code == FUNCTION_SENSOR_DEVICE_INIT) {
                           printf("Sensor device (%#04x) init request\n", rx_struct_msg-
>device_address);

                           RegularMsg displayMsg;
                           displayMsg.function_code =
FUNCTION_HUB_SEND_ADDRESS_SENSOR_DEVICE;
                           displayMsg.device_address = rx_struct_msg->device_address;
                           displayMsg.hub_address = hub_address;
                           displayMsg.data = 0x00;

                           //display_address = rx_struct_msg->device_address;

                           char tx_message[sizeof(RegularMsg)];
                           memcpy(tx_message, &displayMsg, sizeof displayMsg);

                           nrf24_send_message(tx_message);
                    }
                    // Получен ответ о подтверждении инициализации устройства занятости
                    if(rx_struct_msg->function_code == FUNCTION_SENSOR_DEVICE_STATE) {
                           printf("Sensor device (%#04x) initialized\n", rx_struct_msg-
>device_address);
                    }


                    // Получено новое состояние с устройства занятости
                    if(rx_struct_msg->function_code == FUNCTION_SENSOR_DEVICE_CHANGE) {

                           uint8_t stateValue = (rx_struct_msg->data <= US_DISTANCE_PARKING
? 1 : 0);
                           uint8_t stateValueCode = (rx_struct_msg->data <=
US_DISTANCE_PARKING ? 0x11 : 0x10);
                           char *stateName = (rx_struct_msg->data <= US_DISTANCE_PARKING ?
"OCCUPIED" : "FREE");

                           printf("Sensor device (%#04x) new state: %s, distance: %d\n",
rx_struct_msg->device_address, stateName, rx_struct_msg->data);

                           uint8_t pp = 0x00;
                           if(rx_struct_msg->device_address == 0xF1) {
                                  pp = parking_place_1;
                                  parking_place_1_state = stateValue;
                           }
                           if(rx_struct_msg->device_address == 0xF2) {
                                  pp = parking_place_2;
                                  parking_place_2_state = stateValue;
                           }

                           update_display();

                           printf("$%010d%#04x\n", pp, stateValueCode);

                    }
```

```
                        /*
********************************************************************** */

                        // Получен запрос на инициализацию дисплея
                        if(rx_struct_msg->function_code == FUNCTION_DISPLAY_GET_HUB_ADDRESS) {
                               printf("Display (%#04x) init request\n", rx_struct_msg-
>device_address);

                               RegularMsg displayMsg;
                               displayMsg.function_code = FUNCTION_HUB_SEND_ADDRESS_DISPLAY;
                               displayMsg.device_address = rx_struct_msg->device_address;
                               displayMsg.hub_address = hub_address;
                               displayMsg.data = 0x00;

                               display_address = rx_struct_msg->device_address;

                               char tx_message[sizeof(RegularMsg)];
                               memcpy(tx_message, &displayMsg, sizeof displayMsg);

                               nrf24_send_message(tx_message);
                        }

                        // Получен ответ о подтверждении инициализации дисплея
                        if(rx_struct_msg->function_code == FUNCTION_DISPLAY_SEND_HUB_STATE) {
                               printf("Display (%#04x) initialized\n", rx_struct_msg-
>device_address);

                        }

                }

                _delay_ms(10);

        }

        return 0;
}

void init_hub()
{

        RegularMsg displayMsg;
        displayMsg.function_code = FUNCTION_HUB_REQUEST_RESET;
        displayMsg.device_address = 0x01;
        displayMsg.hub_address = hub_address;
        displayMsg.data = 0x00;

        char tx_message[sizeof(RegularMsg)];
        memcpy(tx_message, &displayMsg, sizeof displayMsg);

        nrf24_send_message(tx_message);

}

void update_display()
{

        // Отправка кол-ва свободных мест на табло
```

```
if(display_address != 0x00) {

	RegularMsg displayMsg;

	displayMsg.function_code = FUNCTION_HUB_SEND_DISPLAY_COUNT;
	displayMsg.device_address = display_address;
	displayMsg.hub_address = hub_address;
	displayMsg.data = (2 - (parking_place_1_state + parking_place_2_state));

	char tx_message[sizeof(RegularMsg)];
	memcpy(tx_message, &displayMsg, sizeof displayMsg);
	nrf24_send_message(tx_message);
}
}
```

ПРИЛОЖЕНИЕ В
ИСХОДНЫЙ КОД ПРОГРАММЫ ИНФОРМАЦИОННОГО ТАБЛО

Листинг В.1 – Исходный код программы информационного табло

```
// Частота процессора 16 мГц
#define F_CPU 16000000UL

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdbool.h>
#include <stdio.h>
#include <string.h>

//     Include nRF24L01+ library
#include "nrf24l01.h"
#include "nrf24l01-mnemonics.h"
#include "spi.h"

#define SEGMENTS_COUNT 4

unsigned short displayStep = 0;
unsigned int displayCount = 0;


const int digits[] = {
       0b00111111, 0b00000110, 0b01011011, 0b01001111,
       0b01100110, 0b01101101, 0b01111101, 0b00000111,
       0b01111111, 0b01101111, 0b00111111
};

// Список функций для пеередачи данных
#define FUNCTION_HUB_REQUEST_RESET                                  0x20
#define FUNCTION_HUB_SEND_DISPLAY_COUNT                     0x24
#define FUNCTION_HUB_SEND_ADDRESS_DISPLAY                   0x25

#define FUNCTION_DISPLAY_GET_HUB_ADDRESS              0x31
#define FUNCTION_DISPLAY_SEND_HUB_STATE                     0x32

// Уникальный адрес информационного табло
const uint8_t display_address = 0xA1;
// Адрес хаба, к которому назначен дисплей
uint8_t linked_hub_address = 0x00;

// Структура пакета данных
typedef struct {
       uint8_t function_code;
       uint8_t device_address;
```

```
	uint8_t hub_address;
	uint8_t data;
} RegularMsg;

// Функция подтверждения инициализации дисплея
void apply_init(void);

int main (void) {

	/* ************************************** */
	//     Инициализация приемопередатчика

	nrf24_init();
	nrf24_start_listening();

	/* ************************************** */
	// Инициализация семисегментного индикатора

	// Настройка портов на выход
	DDRD = 0xFF;
	DDRC = 0xFF;
	PORTD = 0xFF;

	// Ининциализация первого таймера

	TCCR1B |= (1<<WGM12);	// сброс по совпадению
	TCCR1B |= (1<<CS11); // предделитель

	OCR1A = 1000;
	TIMSK1 |= (1<<OCIE1A);

	TCNT1 = 0;

	displayCount = 8888;

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

	sei();

	while(1) {

		// Если диспелей не подключен к хабу, отправляется запрос на инициализацию
		if(linked_hub_address == 0x00) {

			RegularMsg hubMsg;
			hubMsg.function_code = FUNCTION_DISPLAY_GET_HUB_ADDRESS;
			hubMsg.device_address = display_address;
			hubMsg.hub_address = 0x00;
			hubMsg.data = 0x00;

			char tx_message[sizeof(RegularMsg)];
			memcpy(tx_message, &hubMsg, sizeof hubMsg);


			nrf24_send_message(tx_message);

			_delay_ms(1000);
		}
```

```
		// Обработка поступающих запросов
		if(nrf24_available()) {

			const char *rx = nrf24_read_message();

			RegularMsg *rx_struct_msg = (RegularMsg *) rx;

			// Получен запрос сброса от хаба
			if(rx_struct_msg->function_code == FUNCTION_HUB_REQUEST_RESET
			&& rx_struct_msg->device_address == 0x01
			&& rx_struct_msg->hub_address == linked_hub_address) {
					linked_hub_address = 0x00;
					displayCount = 8888;
			}

			// Получен адрес хаба
			if(rx_struct_msg->function_code == FUNCTION_HUB_SEND_ADDRESS_DISPLAY) {
				if(rx_struct_msg->device_address == display_address) {
					linked_hub_address = rx_struct_msg->hub_address;
					displayCount = rx_struct_msg->hub_address;
					apply_init();
				}
			}

			// Получено новое значение свободных мест
			if(rx_struct_msg->function_code == FUNCTION_HUB_SEND_DISPLAY_COUNT) {
				displayCount = rx_struct_msg->data;
			}


		}

	}

	return 0;
}


// Прерывание работы семисегментного индикатора
ISR (TIMER1_COMPA_vect)
{

	PORTD = 0x00;
	PORTC |= (_BV(0) | _BV(1) | _BV(2) | _BV(3));

	PORTC &= ~_BV(displayStep);

	int digit = displayCount;
	for(int i = 0; i < displayStep; i++) {
		digit /= 10;
	}

	digit %= 10;


	PORTD = digits[digit];
```

```
        displayStep++;
        if(displayStep == SEGMENTS_COUNT) {
                displayStep = 0;
        }

        TCNT1 = 0; // Сброс счетчика / рестарт таймера

}

void apply_init() {
        RegularMsg hubMsg;
        hubMsg.function_code = FUNCTION_DISPLAY_SEND_HUB_STATE;
        hubMsg.device_address = display_address;
        hubMsg.hub_address = linked_hub_address;
        hubMsg.data = 0x00;

        char tx_message[sizeof(RegularMsg)];
        memcpy(tx_message, &hubMsg, sizeof hubMsg);


        nrf24_send_message(tx_message);
        }
```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД ПРОГРАММЫ МОДУЛЯ ESP8266

Листинг Г.1 – Исходный код программы модуля ESP8266

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>

#include <ESP8266HTTPClient.h>

#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

#include <WebSocketsServer.h>

const char* ssid = "*";
const char* password = "*";

ESP8266WebServer server(80);
WebSocketsServer webSocket = WebSocketsServer(81);

unsigned long loopTimeMain = 0;

String serverPath = "http://singularity172-001-
site1.dtempurl.com/Device/UpdateLot?lotJson=";

void handleRoot() {

  String indexWebPage;
  indexWebPage += "<link rel='stylesheet'
href='https://vtyumencev.github.io/ParkingSystem/ESP-WEB/main.css'></link>";
  indexWebPage += "<script src='https://vtyumencev.github.io/ParkingSystem/ESP-
WEB/main.js'></script>";
  indexWebPage += "<div class='log'></div>";
  server.send(200, "text/html", indexWebPage);

}

void setup(void) {

  Serial.begin(57600);
  //Serial.setTimeout(2000);

  WiFi.mode(WIFI_STA);
  WiFi.begin(ssid, password);

  Serial.println("");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
```

```
  Serial.println("");
  Serial.print("Connected to ");
  Serial.println(ssid);
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  if (MDNS.begin("esp8266")) {
    Serial.println("MDNS responder started");
  }

  server.on("/", handleRoot);

  server.begin();
  webSocket.begin();

  Serial.println("WebSocket & HTTP server started");

}

void loop(void) {

  server.handleClient();
  MDNS.update();
  webSocket.loop();

  WiFiClient client;

  HTTPClient http;

  if (Serial.available()){

    String str = Serial.readStringUntil('\n');
    String wsMsg;
    wsMsg += "[Serial Hub] -> ";
    wsMsg += str;


    webSocket.broadcastTXT(wsMsg.c_str(), wsMsg.length());

    if(str.charAt(0) == '$') {
      String strPlace = str.substring(2,11);
      String strState = str.substring(13,15);

      uint32_t place_id = strPlace.toInt();
      uint8_t parking_state = (strState == "11" ? 1 : 0);

      char sendCh[128];
      sprintf(sendCh, "[Web Server] -> Data: Place id: %d, State: %d", place_id,
parking_state);
       String sendStr = sendCh;

      webSocket.broadcastTXT(sendStr.c_str(), sendStr.length());

      String response = "[Web Server] -> Response: ";
      response += pushToServer(3, place_id, parking_state);


      webSocket.broadcastTXT(response.c_str(), response.length());
```

```
        }
      }

      if ((millis() - loopTimeMain) > 5000) {

        loopTimeMain = millis();
      }



    }

    String pushToServer(uint16_t parking_id, uint32_t place_id, uint8_t place_state) {

      if(WiFi.status()== WL_CONNECTED) {

          HTTPClient http;
          String httpQuery = serverPath;
          char params[128];

          sprintf(params,
"{'parking_id':%d,'ParkingPlaces':[{'place_id':%d,'place_is_free':%s,'park_id':3}]}",
            parking_id,
            place_id,
            (place_state == 1 ? "false" : "true"));

          httpQuery += params;

          Serial.println(httpQuery);

          http.begin(httpQuery.c_str());
          int httpResponseCode = http.GET();

          if (httpResponseCode > 0) {
            Serial.print("HTTP Response code: ");
            Serial.println(httpResponseCode);
            String payload = http.getString();
            Serial.println(payload);
            return payload;
          }
          else {
            Serial.print("Error code: ");
            Serial.println(httpResponseCode);
            return "Error code: " + httpResponseCode;
          }
          http.end();
        }
        else {
          Serial.println("WiFi Disconnected");
        }

        return "Error";
    }
```