Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Е.В. Рогожников, А.Коновальчиков, Е.В. Ломаков

ПРОГРАММИРОВАНИЕ ВСТРАИВАЕМЫХ СИСТЕМ

Методические указания для выполнения практических работ и самостоятельной работы для студентов технических направлений подготовки и специальностей

Томск
2026

УДК 681.3.068
ББК 32.973.2
Р 59

Рецензент:
Дмитриев Э., доцент кафедры телекоммуникаций и основ радиотехники ТУСУРа, кандидат технических наук

Р 59 Программирование встраиваемых систем: Методические указания для выполнения практических работ и самостоятельной работы / Е.В. Рогожников, А. Коновальчиков, Е.В. Ломаков – Томск: Томск. гос. ун-т систем управления и радиоэлектроники, 2026. – 185 с.

Настоящее учебно-методическое пособие содержит указания по выполнению практических работ и самостоятельной работе. Данный практикум имеет целью закрепить и расширить теоретические знания студентов в области цифровой обработки сигналов путем обеспечения работы студентов с реальными сигналами, полученными из радиоэфира, что также позволит применить на практике имеющиеся знания.

Одобрено на заседании кафедры ТОР, протокол № 4 от 26 декабря 2025 г.

УДК 681.3.068
ББК 32.973.2

Оглавление

ВВЕДЕНИЕ

Практикум имеет целью закрепить и расширить теоретические знания студентов в области программирования микроконтроллеров, дать навыки по созданию собственного проекта и по представлению технических проектов.

Практикум содержит описание следующих работ:

1) Введение в программирование микроконтроллеров;
2) Введение в булеву алгебру;
3) Основы программирования микроконтроллеров (часть 1);
4) Основы программирования микроконтроллеров (часть 2);
5) Обработка данных с кнопок и энкодеров;
6) Изучение работы аналого-цифрового преобразователя (ацп);
7) Датчик жестов. Датчик освещенности;
8) Работа с микрофоном;
9) Изучение интерфейса UART;
10) Работа с экраном;
11) Изучение интерфейса I2C;
12) Автоматическое управление микроклиматом на базе датчика температуры;
13) Изучение интерфейса SPI;
14) Реализация кодового замка с применением RFID;
15) Сохранение и чтение файлов с внешнего носителя;
16) Беспроводные модули интернета вещей (часть 1);
17) Беспроводные модули интернета вещей (часть 2);
18) Беспроводные модули интернета вещей (часть 3);
19) Беспроводные модули интернета вещей (часть 4);
20) Разработка проекта «Умный дом» на отладочной плате.

Практические и самостоятельные работы данного перечня выполняются на базе микроконтроллеров STMElectronics и в среде разработки Arduino IDE. Среда разработки Arduino IDE – это ПО, необходимое для программирования плат Arduino. Но поддержка данной среды разработки ушла настолько далеко, что данная платформа позволяет программировать и другие микроконтроллеры, используя язык программирования Wiring, что упрощает работу с данными. Данную программу можно скачать с официального сайта https://www.arduino.cc/en/software.

Работа № 1

«Введение в программирование микроконтроллеров»

За последние 20 лет области применения микроконтроллеров значительно увеличились и продолжают расширяться. Сейчас сложно представить что-либо из окружения, что не содержало бы в себе микроконтроллер. Микроконтроллеры применяются в различных механизмах и устройствах. Основными областями их применения являются: авиационная промышленность, робототехника, промышленное оборудование, железнодорожный транспорт, автомобили, электронные детские игрушки, автоматические шлагбаумы, светофоры, компьютерная техника, электронные музыкальные инструменты, средства связи, медицинское оборудование, бытовая техника.

Цель работы: ознакомить студентов с понятием микроконтроллера, принципе работы и ключевыми различиями между микроконтроллером и микропроцессором, ознакомить со средой разработки Arduino® IDE и программированием микроконтроллеров.

Задачи работы:

1) Получить знания о принципах работы микроконтроллера и сферах его применения.
2) Ознакомиться с понятием программирования микроконтроллеров.
3) Ознакомиться со средой разработки Arduino® IDE.
4) Получить первичные навыки программирования микроконтроллера.

1. Теоретическая часть

1.1 Что такое микроконтроллер?

Микроконтроллер – это специальная микросхема, которая предназначенная для управления различными электронными устройствами. Микроконтроллеры во многом являются основной частью систем «Интернета Вещей», так как в свою очередь принимают и обрабатывают данные с датчиков и отправляют их на сервер.

Вид микроконтроллера представлен на рисунке 1.1:

Рисунок 1.1 – Вид микроконтроллера

Часто путают определения «микроконтроллер» и «микропроцессор». Но действительно ли это название одного и того же устройства?

Микропроцессор – это центральное устройство любой ЭВМ, выполненное по интегральной технологии. Само название говорит о том, что именно в нем происходят вычислительные процессы, пускай они и микро (небольшие). Чтобы из него получилась ЭВМ, пусть даже не очень современная и мощная, его надо дополнить внешними устройствами. В первую очередь, такими как оперативная память и порты ввода и вывода информации.

Микроконтроллер имеет внутри себя процессор, оперативную память, память программ, а кроме этого целый набор периферийных устройств ввода и вывода, которые превращают процессор в полнофункциональную ЭВМ (рисунок 1.2).

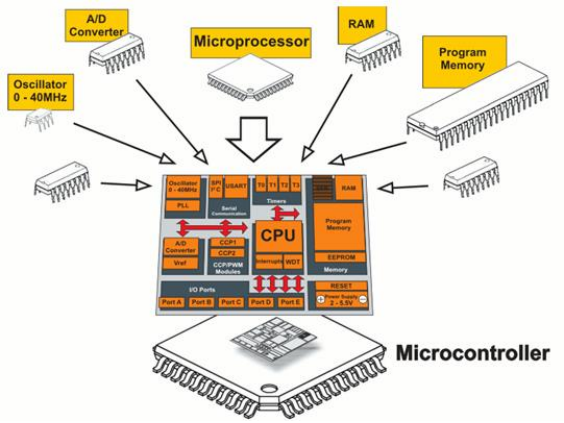


Рисунок 1.2 – Вид микроконтроллера

Каждый микроконтроллер имеет ножки (пины, от англ. pin - штырь). Ножки, это некоторые органы чувств устройства, с помощью которых они взаимодействуют с окружающими устройствами. Каждый датчик, светодиод и прочие периферийные устройства подключаются к ножкам. У каждой ножки микроконтроллера есть свой номер или название. По номеру или названию можно обращаться к ножке, считывая или отправляю на нее данные.

В данном курсе основное взаимодействие будет проводиться с микроконтроллером от «STMElectronics©» под названием STM32F103C8. Распиновка данного микроконтроллера выглядит следующим образом (рисунок 1.3):

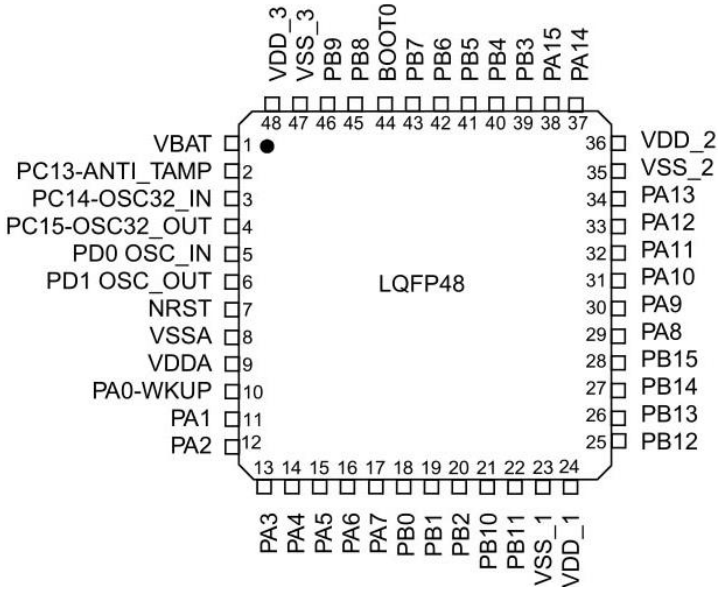


Рисунок 1.3 – Распиновка микроконтроллера STM32F103C8

В процессе выполнения лабораторных работ будет необходимо часто взаимодействовать с пинами микроконтроллера, к которым будут подключены разнообразные периферийные устройства.

1.2 Программирование микроконтроллеров

Программы для микроконтроллеров пишутся на тех же языках, что и программы для компьютеров. Основное отличие заключается лишь в том, что структура программы выстроена таким образом, чтобы работать с периферийными устройствами, которые подключены к микроконтроллеру.

Программы для микроконтроллеров можно писать на различных языках начиная от языков низкого уровня как Assembler, и заканчивая более высокоуровневыми языками: C++, Python, Java и другие. Но чаще всего пользуются языками Си и C++ из-за их скорости работы и читаемости кода.

Язык программирования – инструмент, позволяющий написать команды на понятном для человека языке. Но данный код необходимо перевести в язык, который будет понятен цифровому устройству, и для этого понадобится еще один инструмент, который называется средой разработки.

Среда разработки – это программа, которая состоит из текстового редактора, где вы можете набрать свой программный код, и чаще всего приятным дополнением является то, что такие программы подсвечивают синтаксис написанной программы, чтобы он становился более читаемым, а также компилятора (от англ. compilation - трансляция), который компилирует (транслирует) написанную программу из исходного написанного кода в машинный код, понятный устройству.

Среда разработки для программирования микроконтроллера также поставляется с загрузчиком, который позволяет загрузить машинный код в плату, что иным языком называется - «прошить устройство».

2. Среда разработки ARDUINO® IDE

Среда разработки Arduino® IDE – это ПО необходимое для программирования плат Arduino®. Поддержка данной среды разработки ушла настолько далеко, что данная платформа позволяет программировать и другие микроконтроллеры, используя высокоуровневый язык программирования C++, что упрощает работу с данными.

Основным преимуществом платформы Arduino® является активная поддержка сообществом, что позволяет найти практически любой написанный код, который подходит для решения той или иной задачи, а также написанные библиотеки, которые содержат в себе готовые функции, которые в несколько раз упрощают процесс написания кода программы.

Программа, написанная на платформе Arduino®, называется скетч (от англ. sketch - набросок).

Основной интерфейс программы представлен на рисунке 2.1.

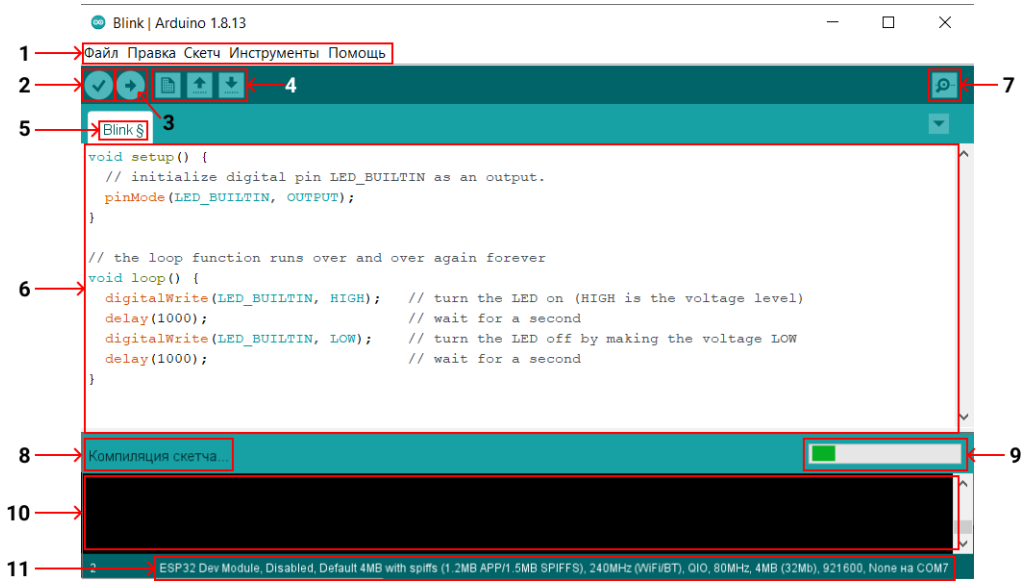


Рисунок 2.1 – Интерфейс программы

Для начала рассмотрим верхнюю часть окна (рисунок 2.2).

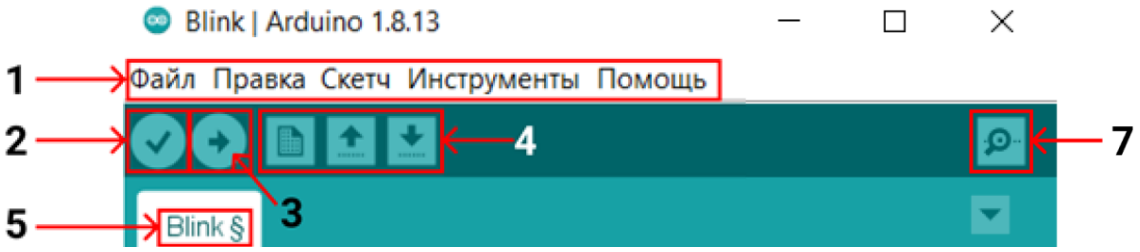


Рисунок 2.2 – Верхняя часть окна

1. Дополнительные команды (настройка среды, выбор контроллера, получение помощи, настройка скетча и другие);

2.  – кнопка, которая запускает процесс проверки кода на наличие синтаксических ошибок.

3.  – кнопка, которая запускает процесс загрузки кода в память микроконтроллера.

4.  – кнопки, которые позволяют взаимодействовать со скетчем (создать новый, загрузить другой, сохранить имеющийся).

5. Вкладка с отображением файла проекта (скетча), количество файлов в проекте (может быть большое количество).

6. Текстовый редактор кода – в данном окне необходимо писать программный код, как вы можете видеть, текстовый редактор подсвечивает ключевые слова, что делает код более читабельным.

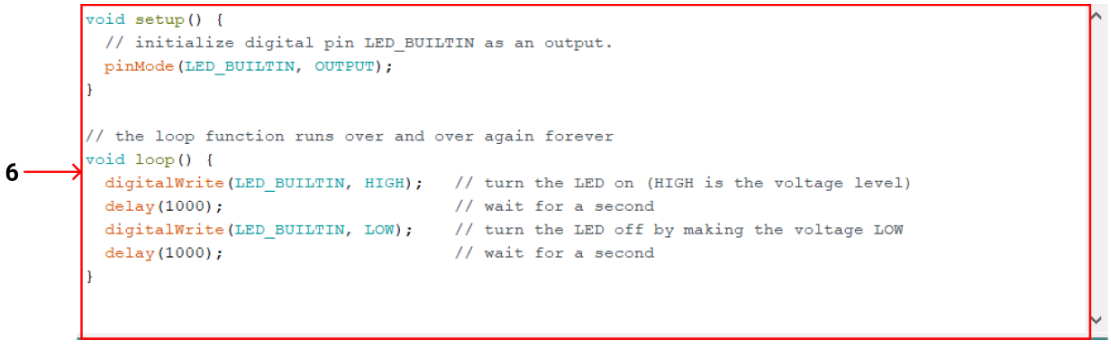


Рисунок 2.3 – Текстовый редактор кода

7.  – кнопка, которая позволяет зайти пользователю в монитор порта.

Монитор порта – отображает данные, которые отправляет плата по интерфейсу UART. Также с помощью монитора порта можно передать некоторые данные в плату в виде строки.

Далее рассмотрим нижнюю часть окна (рисунок 2.4):

Рисунок 2.4 – Нижняя часть программы

8. Строка состояния – в данной строке отображается действие, которая среда разработки выполняет в данный момент.

9. Прогресс бар – указывает прогресс выполнения действия из строки состояния.

10. Консоль программы – в ней указана информация, которую выводит программа для того, чтобы дать больше информации о выполняемом процессе. Также в консоли отображаются ошибки (рисунок 2.5), которые появились при компиляции.

11. Параметры микроконтроллера – в данной строке указан микроконтроллер и его параметры, под который пишется программа.

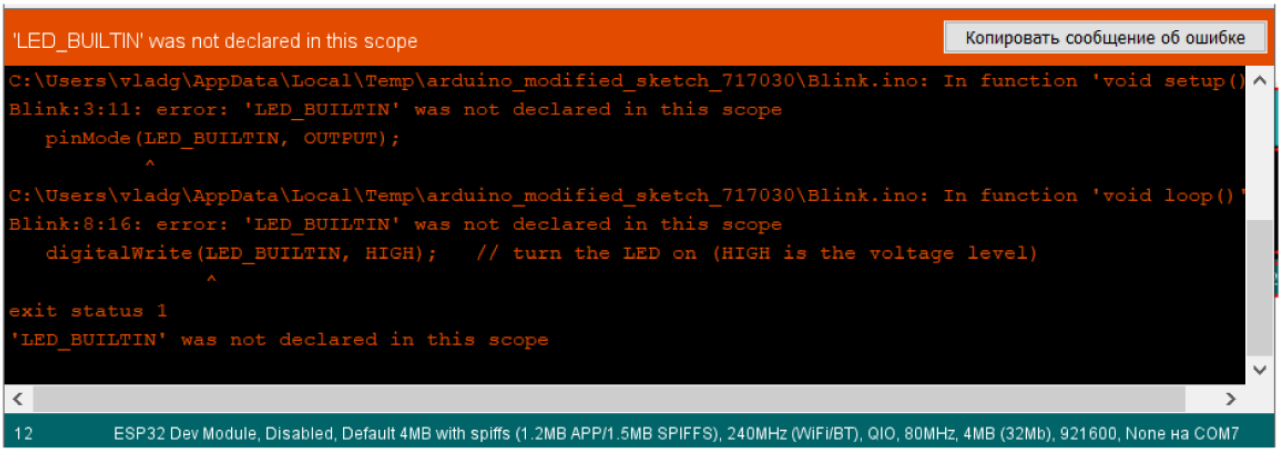


Рисунок 2.5 – Ошибка при компиляции скетча

Ошибки – неотъемлемая часть написания программного кода. Так как код пишется на определённом языке (C\C++), то у него есть свой собственный синтаксис**,** которому необходимо следовать.

Синтаксис **–** это некоторый набор правил языка, на котором вы пишите. Если не следовать синтаксису – программный код работать не будет.

В течение курса постепенно будет изучена часть структуры языка Си, которая потребуется нам для написания кода программы, но этого будет недостаточно для того, чтобы уверено понимать, как работает язык программирования.

Поэтому крайне рекомендуется изучить синтаксис языка Си самостоятельно!

3. Знакомство с платой TUSUR IOT BOARD и её возможностями

Плата TUSUR IoT Board представляет собой отладочную плату на базе двух микроконтроллеров: STM32F103C и ESP-WROOM-32.

В свою очередь, контроллер ESP-WROOM-32 является вспомогательным контроллером, который позволяет управлять подключаемой периферией и экраном.

Благодаря контроллеру ESP-WROOM-32 на плате осуществляется:

– Взаимодействие с экраном;

– Работа с отключением\включением периферии;

– Возможность изучать передаваемые данные в плате с помощью осциллографа (аналоговые сигналы) и логического анализатора (SPI, I2C, UART и другие цифровые сигналы)

– Возможность автоматической прошивки контроллера STM32F103C без установки пинов в нужное положение (осуществляется автоматически).

Контроллер STM32F103C является основным программируемым контроллером, к которому подключена большая часть периферии. Список подключенных модулей и датчиков указан в таблице 3.1.

Таблица 3.1 – Список подключенных модулей и датчиков к плате TUSUR IoT Board

ВВОД	ВЫВОД	ВВОД/ ВЫВОД
Датчик температуры	Вентилятор	NRF
Датчик давления и температуры	Светодиоды	WI-FI (ESP)
RFID	Экран (ESP)	Bluetooth (ESP)
IR приемник	Зуммер	Flash-карта
Кнопка, переключатели	RGB светодиод	
Encoder	IR передатчик	
Микрофон	Динамик (ESP)	
Переменный резистор		
Фоторезисторы		

К элементам, у которых рядом в скобках указано - (ESP), можно получить доступ только через контроллер ESP-WROOM-32.

Для того, чтобы получить информацию о том, к каким пинам подключены все модули и датчики, необходимо воспользоваться пунктом меню «Распиновка» на плате. Как это сделать рассмотрим ниже.

Главное меню прошивки для взаимодействия с платой представленно следующим образом (рисунок 3.1):

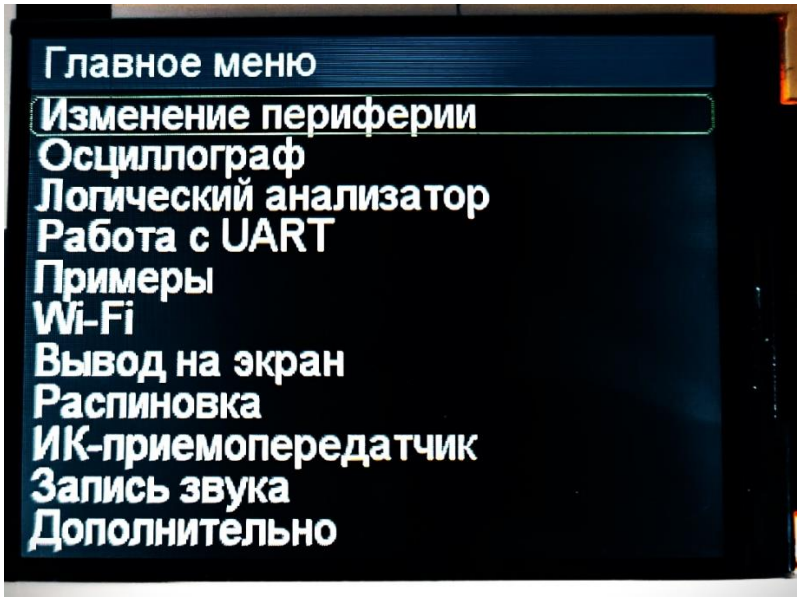


Рисунок 3.1 – Меню для взаимодействия с платой

Меню может немного отличаться в зависимости от прошивки.

Для навигации в меню необходимо использовать нижний энкодер (от англ. encoder – кодирующее устройство), предназначенный для преобразования угла поворота вращающегося объекта в цифровые или аналоговые сигналы, который расположен недалеко от экрана. Расположение энкодера представлено на рисунке 3.2.

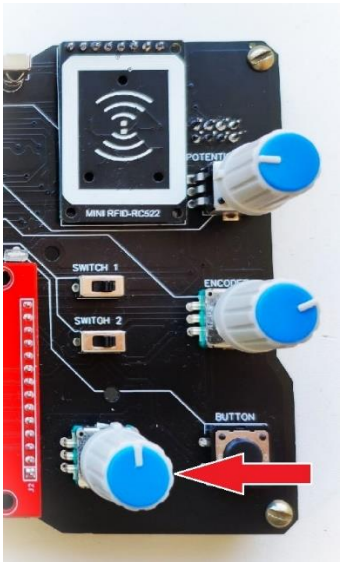


Рисунок 3.2 – Энкодер, использующейся для навигации

3.1 Устройство главного меню

Главное меню платы TUSUR IoT Board содержит довольно большое количество инструментов, позволяющих настроить плату под конкретную задачу, а также получить необходимую текстовую или графическую информацию. Главное меню представлено на рисунке 3.1 и содержит:

- Изменение периферии – меню, которое позволяет подключать\отключать модули, подключенные к STM (вентилятор, RGB-светодиод и пр.);

• Осциллограф – прибор, который позволяет вывести сигналы, которые подаются на один из пинов ESP;

• Логический анализатор – позволяет увидеть, какие сигналы передаются при использовании интерфейсов I2C, SPI, UART;

• Работа с UART – позволяет вывести на экран платы «монитор порта» или «плоттер» с необходимой скоростью;

• Примеры – позволяет запустить заранее написанные примеры для демонстрации взаимодействия с периферией платы. Обязательно должен быть залит скетч *STM_EXAMPLES.INO* на микроконтроллер STM;

• Wi-Fi – позволят включить/выключить Wi-Fi модуль;

• Вывод на экран – отображение команд для вывода на экран различных фигур и определенной палитрой цветов;

• Распиновка – отображает список «пинов» с подключенной к ним периферией;

• ИК-приемопередатчик – позволяет формировать или принимать команду по ИК порту;

• Запись звука – позволят записать фрагмент или выполнять потоковое воспроизведение звука с микрофона;

Дополнительно – содержит в себе перезагрузку платы или можно включить генератор QR-кодов.

4. Практическая часть

4.1 Разработка своего первого проекта

Для того чтобы увидеть сигнал, поданный на ножку микроконтроллера, мы можем к ножке подключить светодиод, который при наличии сигнала будет включен, и тогда мы с легкостью увидим наличие или отсутствие сигнала.

Для того чтобы начать проект, запустите программу Arduino® IDE (рисунок 4.1):

Рисунок 4.1 – Иконка программы Arduino® IDE

Далее в открывшемся окне нажмите на кнопку «Новый проект» (рисунок 4.2):

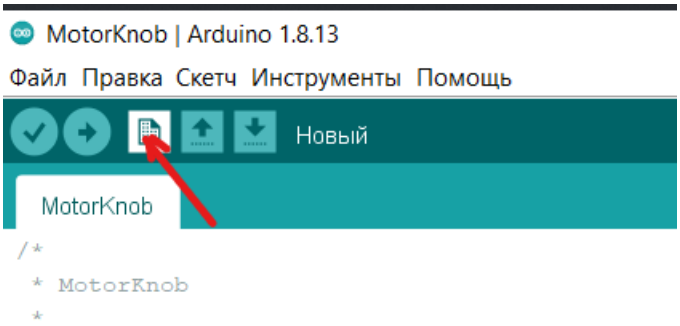


Рисунок 4.2 – Создание проекта

После создания нового проекта откроется следующее окно (рисунок 4.3):

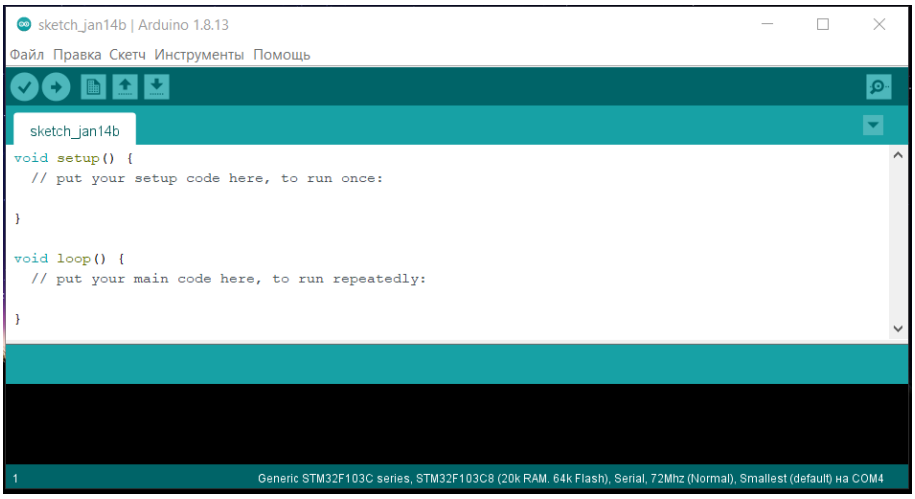


Рисунок 4.3 – Новый проект

После того как проект был создан, необходимо дать ему название и сохранить его, для этого необходимо нажать кнопку сохранить (рисунок 4.4) или нажать сочетание клавиш «*ctrl + s*»:

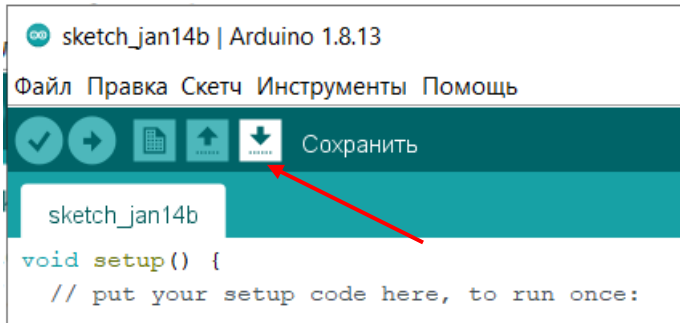


Рисунок 4.4 – Сохранение проекта

Сохраните проект в папке Arduino® (рисунок 4.5):

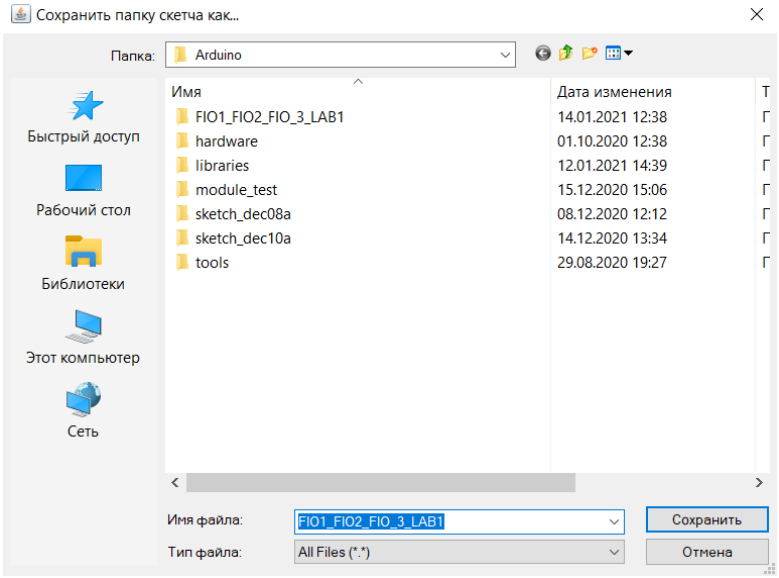


Рисунок 4.5 – Сохранение проекта

Для названия проекта следует выбрать следующую формулу:

SNM1_WORKN

где SNM1 – инициалы автора, который выполняет работу;

WORKN – где N – номер работы.

Далее необходимо выбрать контроллер, который вы хотите запрограммировать и выставить его настройки. Для этого перейдите во вкладку «Инструменты» (рисунок 4.6):

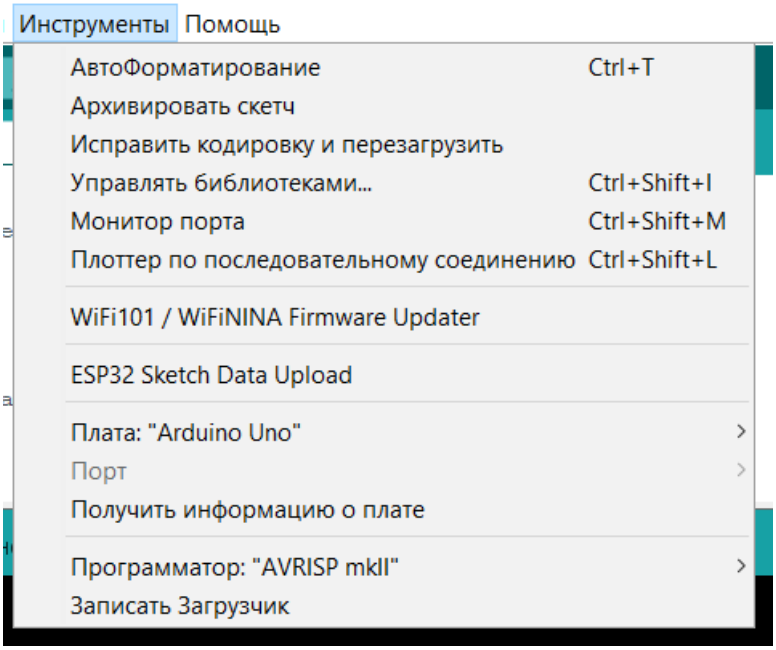


Рисунок 4.6 – Вкладка инструменты

Далее выберите микроконтроллер Generic STM32F103C series. Процесс выбора микроконтроллера представлен на рисунке 4.7:

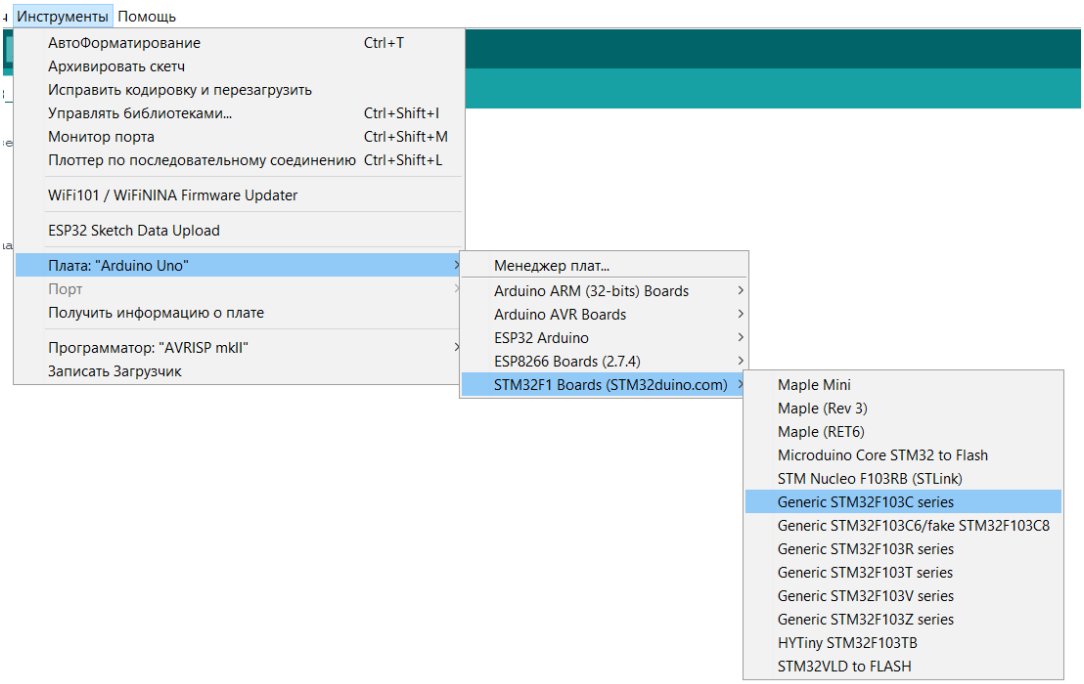


Рисунок 4.7 – Процесс выбор платы

Далее необходимо выставить настройки платы согласно рисунку 4.8. Важно всегда выбирать «Serial», поскольку связь между компьютером и платой TUSUR IoT Board осуществляется по последовательному порту.

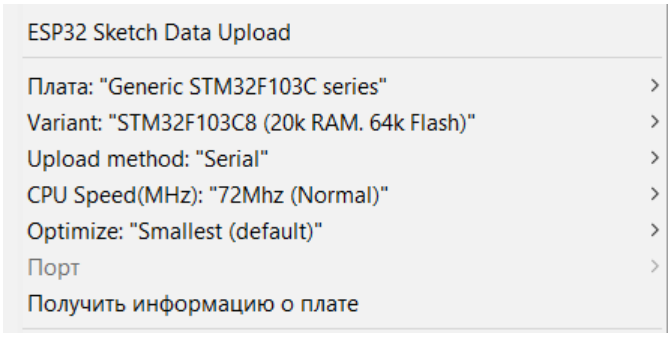


Рисунок 4.8 – Настройка платы

Далее необходимо выбрать COM-порт, к которому подключена плата (рисунок 4.9):

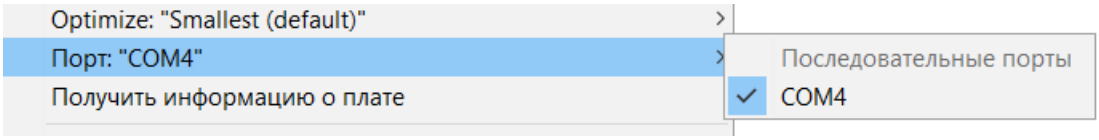


Рисунок 4.9 – Выбор COM порта

Номер COM-порта может отличаться на разных компьютерах. Для того, чтобы выбрать верный, необходимо выбрать порт отличный от COM-1.

После того, как проект был создан, можно приступить к его разработке. В текстовом редакторе мы можем увидеть следующие строки:

```
void setup() {
  // Таким образом указываются строчные комментарии к вашему коду
}
void loop() {
}
```

`void setup()` – это функция, которая выполняется один раз при запуске контроллера. При запуске микроконтроллера код, который находится в этой функции, выполнится единожды и будет выполняться каждый раз при включении микроконтроллера.

Обычно в данной функции происходит настройка микроконтроллера, к примеру:

– настройка пинов;

– инициализация устройств.

`void loop()` – это функция, которая будет непрерывно вызываться каждый раз, после функции `setup()`, то есть код, который содержится в данной функции, будет выполняться циклически без перерыва.

Так как нашей задачей является поморгать светодиодом, то нам необходимо в функции `setup()` настроить пин, к которому подключен светодиод на выход.

Для этого нужно воспользоваться функцией `pinMode(…)`:

```
void setup() {
  // put your setup code here, to run once:
  pinMode(PC13, …);  //Перевод пина PC13 в "выходной" пин
}
```

Здесь в функцию `pinMode(…);` передается 2 параметра:

PC13 – название ножки, к которой подключен тестовый светодиод светодиод.

OUTPUT – состояние пина на выход (от англ. Output – выход). Это означает, что пин переходит в выходное состояние, что, в свою очередь, означает, что пин будет посылать, а не принимать сигнал. Существует еще также состояние пина INPUT (от англ. Input – вход), то есть режим пина на вход, на принятие сигнала.

Также в коде вы можете заметить следующие строки:

```
// put your setup code here, to run once:
//Перевод пина PC13 в "выходной" пин
```

Данные строки называются комментариями, и они никак не влияют на работу программы, но позволяют программисту делать необходимы пометки в коде для того, чтобы понять, какие действия выполняет программа.

Комментарии могут выглядеть следующим образом:

```
//Строчный комментарий
/*Комментарий
Который содержит
Несколько строк*/
```

ВАЖНО!

Используйте комментарии в своем коде для того, чтобы как Вы, так и преподаватель могли лучше понимать ход Ваших мыслей.

После того, как пин был настроен, нам необходимо разработать логику поведения работы программы.

Для того чтобы последовательно включать и выключать светодиод, нам необходимо включить диод, подождать какое-то время, выключить диод, и снова подождать какое-то время.

Для этого воспользуемся функциями `digitalWrite(…)` и `delay(…)`:

```
void loop() {
  // put your main code here, to run repeatedly:
  digitalWrite(PC13, …); //Подача высоко напряжения на пин
  delay(…); // Задержка на … мс
  digitalWrite(PC13, …); // Подача низкого напряжения на пин
  delay(…); // задержка на … мс
}
```

`digitalWrite(PC13, HIGH);` – функция, которая подает на определенный пин (в нашем случае PC13) напряжение HIGH (от англ. High - высокий) или LOW (от англ. Low - низкий), то есть либо высокое напряжение (логическая единица), либо низкое напряжение (логический ноль).

`delay(…);` – (от англ. delay – задерживать) функция, которая дает команду контроллеру «подождать» некоторое время. То есть данная функция принимает значение времени в мс., которое контроллер будет «простаивать\ожидать».

Допустим, для того, чтобы контроллер простаивал 2 сек, необходимо ввести функцию `delay(2000);`

$2000\ мс\ =\ 2\ сек$

После того, как все настройки будут выполнены, необходимо скомпилировать программу и загрузить ее в контроллер, для это необходимо нажать кнопку *«Загрузка»* (рисунок 4.10):

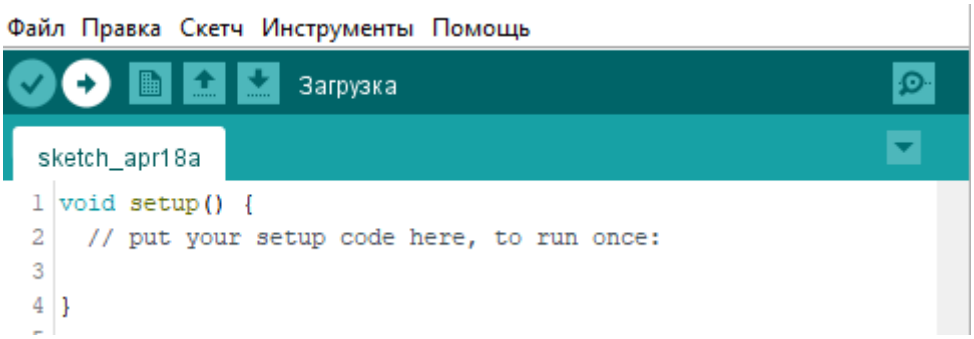


Рисунок 4.10 – Загрузка скетча в микроконтроллер

После нажатия кнопки начнется процесс загрузки кода в микроконтроллер (рисунок 4.11):

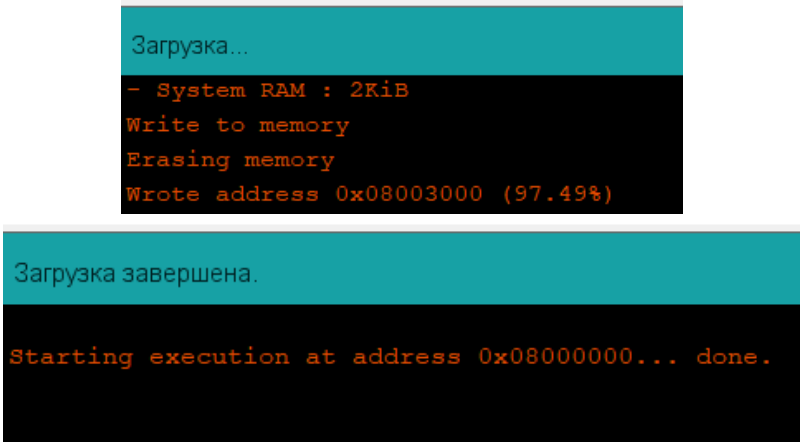


Рисунок 4.11 – Загрузка программы в микроконтроллер

Если программа была написана верно, то вы увидите мигающий светодиод на плате.
Далее модернизируйте программу, чтобы:
1) Светодиод был включен на протяжении 500 мс и выключен на протяжении 500 мс;
2) Светодиод был включен на протяжении 2 с и выключен на протяжении 1 с;
3) Чтобы светодиод мигал с частотой 5 Гц.

Работа № 2

«Введение в булеву алгебру»

Методическое пособие содержит теоретический материал по изучению двоичной системы счисления, исчерпывающую информацию о переводе чисел из десятичной системы счисления в двоичную, а также информацию о логических операциях, выполняемых над двоичными числами.

Цель работы: ознакомиться с двоичной системой счисления и основными логическими операциями.

Задачи работы:

1) Научиться переводить числа из 10-чной системы счисления в 2-чную и наоборот;
2) Познакомиться с выражениями True, False;
3) Познакомиться с логическими операторами AND, OR, XOR, NOT.

1. Теоретическая часть

1.1 Двоичная система счисления

Почему нужно знать двоичную систему счисления для программирования микроконтроллеров?

Дело в том, что 2-чная система счисления – это язык вычислительной техники.

Любая программа, текст или число должны храниться в памяти компьютера. Для хранения данных в компьютере используется двоичная система счисления.

Допустим, у нас есть десятичное число 52, которое требуется сохранить в компьютерной памяти. Мы задействуем участок памяти, в данном случае состоящий как минимум из двух элементов, отводимых под разряды. В одном из разрядов мы сохраняем десятичное число 5, в другом – число 2.

Элемент памяти – это физическое электронное устройство. Если проектировать его для хранения десятичной цифры, потребуется создать такое устройство, которое может находиться в десяти разных состояниях и способно переключаться между ними. Каждое из этих состояний будет соответствовать числу от 0 до 9 и иметь ровное 10 состояний (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).

Создать такой элемент памяти возможно, однако сложнее и дороже, чем создать элемент, способный находиться только в двух состояниях. Одно состояние сопоставить нулю, второе – единице. То есть либо наличие, либо отсутствие напряжения.

Поэтому оказалось проще перевести число 52 в двоичную систему счисления, получив число 110100, и именно его сохранить в памяти. И пусть даже при этом будут задействованы не два, а шесть разрядов, то есть шесть единиц памяти.

Единицей памяти в информатике называется бит, то есть хранение числа 52 занимает 6 бит памяти (1 1 0 1 0 0).

В компьютере мы привыкли видеть информацию в байтах:

В 1 байте содержится 8 бит.

В 1 Кбайте содержится 1024 байта

В 1 Мбайте содержится 1024 Кбайта.

В 1 Гбайте содержится 1024 Мбайта и т.д.

Но что нужно сделать для того, чтобы превратить десятичное число в двоичное?

Для это нужно провести ряд вычислений.

Одним из алгоритмов перевода десятичного числа в двоичное является деление нацело на два с последующим «сбором» двоичного числа из остатков. Переведем таким образом число 52 в двоичное представление.

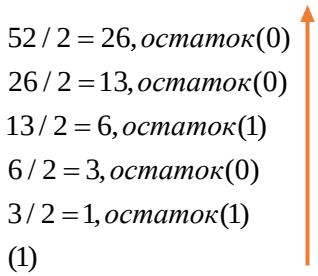


Рисунок 1.1 – Перевод числа 52 в двоичную систему счисления

Собирать остатки надо с конца, то есть с последнего деления. Получаем 110100.

Для того чтобы перевести обратно в десятичный, необходимо каждый разряд умножить на 2 в степени номера разряда (самый крайний правый имеет номер 0). Номер разряда увеличивается с права на лево.

Таблица 1.1 – Преобразование 2-чного числа в 10-чное

Значение разряда	1	1	0	1	0	0
Номер разряда	5	4	3	2	1	0
2 в степени номера разряда	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
2 в степени номера разряда * значение разряда	$32 \cdot 1 = 32$	$16 \cdot 1 = 16$	$8 \cdot 0 = 0$	$4 \cdot 1 = 4$	$2 \cdot 0 = 0$	$1 \cdot 0 = 0$
Сумма всех полученных результатов	$32 + 16 + 4 = 52$					

1.2 Логические выражения

Одним из преимуществ двоичной системы счисления является то, что с помощью неё можно организовывать логические операции в программе.

То есть это сравнения каких-либо величин, логические операции и многое другое.

Для логических операций используются выражения «True» (от англ. true – истина) и «False» (от англ. false – ложь). Для представления в двоичной системе их можно записать как: 1 – «True», а 0 – «False».

Данные выражения используются для того, чтобы показать результат некоторой логической операции, допустим, операций сравнения двух переменных.

Для сравнения двух переменных используются операторы сравнения, в языке Си они представлены следующим образом (таблица 1.2):

Таблица 1.2 – Операторы сравнения в языке Си

Равенство	==	$a == b$
Неравенство	!=	$a != b$
Больше	>	$a > b$
Меньше	<	$a < b$
Больше или равно	>=	$a >= b$
Меньше или равно	<=	$a <= b$
Неравенство	!=	$a != b$

Результатом сравнения двух переменных может быть выражение либо «True», либо «False».

Для примера сравним несколько выражений (таблица 1.3):

Таблица 1.3 – Примеры использования операторов сравнения

25	==	43	=	False
34	!=	53	=	True
2	>	21	=	False
3	<	23	=	True
4	>=	4	=	True
5	<=	10	=	True

Также в программировании присутствуют логические операторы, такие как «OR» (от англ. or – или), «AND» (от англ. and – и), «NOT» (от англ. not – не). В языке СИ они представлены следующим образом (таблица 1.4):

Таблица 1.4 – Логические операторы в СИ

Логическое «ИЛИ»	||	$a \,\|\| b$
Логическое «И»	&&	$a \,\&\& b$
Логическое «НЕ»	!	$!a$

Как вы можете заметить, операции логических «И» и «ИЛИ» выполняются между двумя аргументами, в то время как операция логического «НЕ» выполняется по отношению к одному аргументу.

Работу логических операторов удобнее представлять в виде таблицы истинности.

Рассмотрим работу оператора «OR» (таблица 1.5):

Таблица 1.5 – Таблица истинности оператора «OR»

False	||	False	=	False
False	||	True	=	True
True	||	False	=	True
True	||	True	=	True

Можно легко заметить, что истинными результатами при использовании логического оператора «OR» будут являться случаи, когда одно или оба из выражений истинны.

Рассмотрим работу оператора «AND» (таблица 1.6):

Таблица 1.6 – Таблица истинности оператора «AND»

False	&&	False	=	False
False	&&	True	=	False
True	&&	False	=	False
True	&&	True	=	True

Можно легко заметить, что истинным результатом при использовании логического оператора «AND» будет являться только один случай – когда оба из выражений истинны.

Рассмотрим работу оператора «NOT» (таблица 1.7):

Таблица 1.7 – Таблица истинности оператора «NOT»

!False	=	True
!True	=	False

Из таблицы видно, что оператор «NOT» просто инвертирует полученный аргумент, если «НЕ» ложь, то истина, если «НЕ» истина, то ложь.

2. Практическая часть. Проверка работы логических операторов и операторов сравнения в Arduino IDE

2.1 Подключение кнопок и переключателей

В прошлой лабораторной работе мы использовали проект, который задействует только 1 устройство вывода – светодиод. Но часто происходит так, что нужно подключать к плате некоторые вводные устройства, при взаимодействии с которыми на плату будут поступать данные.

В данной лабораторной работе мы научимся подключать такие типы устройств, как переключатель и кнопку.

Как переключатель, так и кнопка могу находиться в двух состояниях ВКЛ. (0) и ВЫКЛ, (1). Данная особенность является очень удобной для цифровых систем, так как мы можем интерпретировать состояние кнопки или переключателя, используя двоичную систему счисления.

Для начала необходимо:

1) Создать новый проект;

2) Сохранить его;

3) Настроить плату.

Согласно тому, как это происходило в 1-ой лабораторной работе

После того, как проект будет создан и настроен, в текстовом редакторе будет находиться знакомая картина:

```
void setup () {
  // put your setup code here, to run once:
}
void loop () {
  // put your main code here, to run repeatedly:
}
```

Далее в поле `setup ()` добавьте инициализацию пина к которому подключен светодиод на выход!

(Можно посмотреть в 1-ой лабораторной работе)

Пином, к которому подключен светодиод является PC13

```
void setup () {
        ***ВАШ КОД ИНИЦИАЛИЗАЦИИ ПИНА PC13 НА ВЫХОД***
}
```

Далее необходимо подключить кнопку, с помощью которой будет осуществляться управление светодиодом.

И так как нам нужно будет считать значение с кнопки, то необходимо настроить пин на вход. Для этого необходимо добавить следующую команду:

`pinMode(PB4, INPUT);` – инициализация пина на вход, где:

PB4 – это название пина на микроконтроллере, к которому подключена кнопка.

INPUT – назначение пина на вход.

```
void setup() {
    ***ВАШ КОД ИНИЦИАЛИЗАЦИИ ПИНА PC13 НА ВЫХОД***
    pinMode(PB4, INPUT); // ИНИЦИАЛИЗАЦИИ ПИНА PB4 НА ВХОД
 }
```

Далее в поле `loop ()` необходимо создать переменную, которая будет хранить в себе значение, которое будет считываться с кнопки.

```
 void loop () {
    bool but = digitalRead(…); // Запись цифрового значения с пина PB4 в
переменную but
 }
```

Здесь:

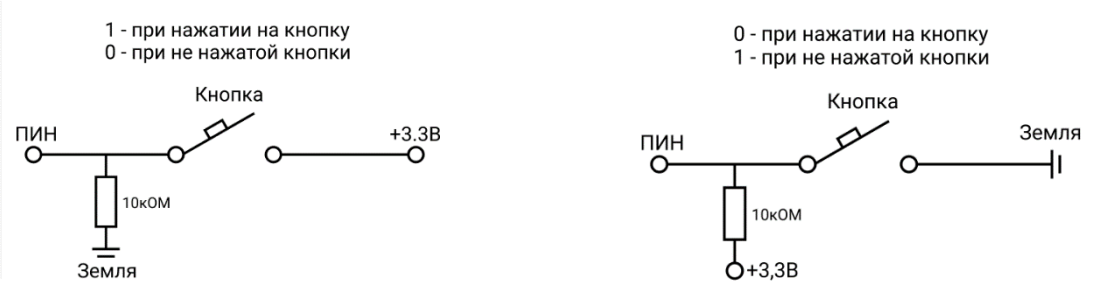
`bool` – тип данных, которой может хранить в себе лишь два значения информации. То есть либо 0(False), либо 1(True). В языке СИ существует большое множество типов данных. В данном курсе они будут рассмотрены чуть позже.

but – название переменной, которая будет хранить в себе данные, считанные с кнопки.

`digitalRead(…)` – функция, которая считывает значение с пина. Дословно можно перевести данную функцию как «цифровое считывание». С помощью данной функции можно считать два состояния высокое напряжение (логическая 1) и низкое напряжение (логический 0). Значение высоко напряжения определяется логикой микроконтроллера. На микроконтроллере STM, которым мы пользуемся, используется 3,3В логика. То есть 3,3В – высокое напряжение. 0В – низкое напряжение.

В зависимости от значения на пине, данная функция вернет значение 0 или 1.

Так как кнопка сама по себе не может генерировать сигнал и представляет собой всего лишь ключ, следовательно, нужно построить схему включения данной кнопки (рисунок 2.1):



а) б)

Рисунок 2.1 – Схема подключения кнопки к пину, где, а) первый тип, б) второй тип

Второй тип подключения – за счет бесконечного сопротивления при разжатой кнопки на вход пина, будет подаваться напряжение в 3,3 В, что соответствует логической единице. Как только кнопка будет нажата, между землей и пином будет нулевой напряжение, следовательно, на пине будет 0 В, что соответствует логическому нулю.

На плате TUSUR IoT Board реализован 1 тип подключения кнопки PB4.

Основное различие между 1 и 2 типом подключения заключается в значение, которое будет при нажатии на кнопку. При 1 типе подключения на пине при нажатии на кнопку будет подано высокое напряжение.

Далее нам необходимо на пин с диодом подать напряжение в соответствие с напряжением, которое мы получили с пина, на котором установлена кнопка.

Для этого в поле `loop ()` необходимо написать следующий код:

```
void loop() {
   bool but = digitalRead (PB5);
    digitalWrite (PC13, but); //Подача напряжения, снятого с PB4 на пин PC13.
 }
```

Помните, что переменная but хранит в себе значение, считанное с пина к которому подключена кнопка.

То есть если кнопка не будет нажата, то в переменной but будет хранится значение (0), согласно рисунку 2.1. Данное значение передается в функцию, которая подаст на пин, к которому подключен светодиод, напряжение 0 В (пин подключен к земле), что соответствует логическому нулю и светодиод будет выключен.

Если же нажать на кнопку, то на пине PB4 будет высокое напряжение, и переменная but будет хранить значение 1. Следовательно, на светодиоде будет напряжение 3,3 В, что соответствует логической единице и светодиод будет включен.

Завершенный программный код, выглядит следующим образом:

```
void setup() {
    ***ВАШ КОД ИНИЦИАЛИЗАЦИИ ПИНА PC13 НА ВЫХОД***
    ***ВАШ КОД ИНИЦИАЛИЗАЦИИ ПИНА PB4 НА ВХОД***
 }
 void loop () {
    bool but = «команда на чтение» (…); // Запись цифрового значения с пина PB4 в
переменную but
    «команда на запись» (…, …); //Подача напряжения, снятого с PB4 на пин PC13.
 }
```

Загрузите программный код в микроконтроллер, согласно тому, как вы это делали в ЛР№1.

Протестируйте программу!

При нажатии на кнопку – светодиод должен гореть.

Если все работает верно, то:

1) Модифицируйте программный код таким образом, чтобы при нажатии на кнопку, светодиод выключался, а при выключенной кнопке горел. (Программно реализовать 2 тип подключения);

Подсказка: воспользуйтесь оператором НЕ (!), который описан в теоретическом материале.

2) Вместо кнопки подключите на вход переключатель (на плате их 2, и они расположены на пинах PC14 и PC15. Нужно выбрать один).

Переключатель работает по такому же принципу, что и кнопка, принципиальное различие лишь в том, что переключатель сохраняет свое состояние за счет своего строения. То есть для удержания его во включенном состоянии не нужно постоянно держать переключатель в данном положении, достаточно только перевести его в это состояние.

2.2 Изучение работы логических операторов

В прошлой работе мы научились подключать кнопки и переключатели. Теперь на основе полученных знаний, необходимо изучить работу логических операторов. С одним логическим оператором вы уже познакомились. Это оператор НЕ (!).

Особенность оператора заключается в том, что для его работы нужен лишь один операнд.

Операнд – это переменная или число, над которым выполняется операция.

То есть в случае !but:

! – операция логического отрицания «НЕ»;

but – операнд над которым выполняется операция.

Далее напишите программу, которая:

1) Настраивает пин со светодиодом на выход (PC13);

2) Настраивает пины с двумя переключателями на вход (PC14, PC15).

Далее в поле `loop ()` необходимо создать переменные, которые будут хранить в себе состояние переключателей 1 и 2:

```
void loop () {
   // put your main code here, to run repeatedly:
   bool switch1 = «команда на чтение» (…);
   bool switch2 = «команда на чтение» (…);
   bool result = switch1 лог. операция switch2
 }
```

Далее необходимо выполнить логическую операцию между данными переменными и результат операции направить на **пин PC13**, к которому подключен светодиод, точно также, как мы это делали в предыдущей работе.

Далее:

1) Вместо поля «лог. операция» вставьте логическую операцию «ИЛИ», загрузите программу в микроконтроллер, запишите результаты работы в таблицу 2.1;

2) Вставьте логическую операцию «И», загрузите программу в микроконтроллер, запишите результаты работы в таблицу 2.1.

Таблица 2.1 – Результаты работы программы

Логическая операция ИЛИ		
Состояние 1-го переключателя	Состояние 2-го переключателя	Состояние светодиода
Выкл.	Выкл.	
Вкл.	Выкл.	
Выкл.	Вкл.	
Вкл.	Вкл.	

Логическая операция И		
Состояние 1-го переключателя	Состояние 2-го переключателя	Состояние светодиода
Выкл.	Выкл.	
Вкл.	Выкл.	
Выкл.	Вкл.	
Вкл.	Вкл.	

Сравните полученные результаты с таблицами 1.5 и 1.6.

Работа № 3

«Основы программирования микроконтроллеров часть 1»

В языке СИ очень важно правильно указать тип переменной, т.к. от этого будет зависеть работоспособность программы. Если тип данных выбран неправильно, то и значения в переменную запишутся неверно, или будет выделено необоснованно большее количество памяти для их хранения.

Наряду с важностью знания типов переменных и умения их применять для программирования необходимо знать условные и циклические операторы. Они необходимы для того, чтобы для определенных условий выполнить отдельный участок программы, и для того, чтобы отдельный участок программы выполнить несколько раз в ходе одного выполнения программы.

Цель работы: получить базовые навыки по программированию микроконтроллеров.

Задачи работы:

1) Изучить основные типы данных языка СИ, используемых при программировании микроконтроллеров.

2) Изучить работы условных операторов в языке СИ.

3) Изучить принципы работы циклических операторов в языке СИ.

1. Теоретическая часть

1.1 Типы данных языка СИ

Для хранения каких-либо данных требуется выделение определенного места в области памяти. Так для хранения булевой переменной потребуется всего один бит (0 или 1) (False или True), в то время как для хранения числа 9 потребуется 4 бита информации (1001).

И для того чтобы под каждую переменную в коде не было необходимости указывать количество выделяемой памяти в языках программирования со строгой типизацией, к такому языку программирования относится язык СИ, разработаны определенные типы данных, которые занимают фиксированное значение занимаемой памяти.

При разработке проектов мы уже пользовались объявлением переменной с указанием типа данных:

```
void loop() {
    bool but = digitalRead(PB4); // Запись цифрового значения с пина PB4 в
переменную but
 }
```

Здесь, когда мы создали переменную `but`, мы перед ней указали тип данных bool

`bool` – тип данных, которой может хранить в себе лишь два значения информации. То есть либо 0, либо 1.

При создании переменной bool в памяти устройство выделяется память размером 1 байт.

И тут сразу же назревает вопрос: почему для хранения всего двух значений (True, False) необходимо выделять 1 байт памяти. Напомним, в 1 байте (8 бит) памяти может хранится $2^8 = 256$ значений.

Все дело в том, что архитектура процессоров выполнена таким образом, что они могу передавать данные, вес которых равен минимум 1 байту.

Типов данных переменных в языке Си большое множество и при создании переменной на это необходимо обращать внимание!

Допустим, для хранения состояние True или False. Мы знаем, что достаточно будет выделить 1 байт памяти, используя тип данных bool.

Но что, если мы в памяти устройства захотим записать число, к примеру, год, когда появилась первая возможность программировать микроконтроллеры, используя Arduino IDE – 2005 год. Для хранения такой переменной понадобится больше, чем 1 байт, а если быть точнее, то 11 бит [11111010101]. Получим данное число в байтах, для этого разделим на 8: $\frac{11}{8}$ = 1,375 байт. Округляем в большую сторону и получаем 2 байта.

Какой тип данных использовать для хранения переменой, которая занимает 2 байта?

Воспользуемся таблицей основных типов данных, используемых при программировании микроконтроллеров в Arduino IDE (таблица 1.1):

Таблица 1.1 – Основные типы данных в Arduino IDE

Название	Альт. название	Вес	Диапазон	Особенность
boolean	bool	1 байт	0 или 1	Логическая переменная
char	–	1 байт	-128… 127	Хранит номер символа из таблицы символов ASCII
–	int8_t	1 байт	-128… 127	Целочисленный тип
byte	uint8_t	1 байт	0… 255	Целочисленный тип
int	int16_t, short	2 байта	-32 768… 32 767	Целочисленный тип
unsigned int	uint16_t, word	2 байта	0… 65 535	Целочисленный тип
long	int32_t	4 байта	-2 147 483 648… 2 147 483 647	Целочисленный тип
unsigned long	uint32_t	4 байта	0… 4 294 967 295	Целочисленный тип
float	–	4 байта	-3.4028235E+38… 3.4028235E+38	Хранит числа с плавающей точкой (десятичные дроби). Точность: 6-7 знаков

При выборе типа данных переменных нужно внимательно смотреть на диапазон значений, которые можете хранить в себе переменная. Диапазон значений некоторых переменных может включать значения меньше 0, то есть отрицательные числа.

1.2 Условные операторы

При программировании микроконтроллеров часто возникают ситуации, при которых необходимо выполнить некоторый фрагмент программы при определенном условии.

К примеру, можно отнести ситуации, когда при нажатии на кнопку необходимо поменять некоторые характеристики работы микроконтроллера.

Для этого нужно создать условие, которое проверит: нажата ли кнопка? А далее должна выполниться часть кода, которая меняет параметры основной программы.

Для создания условия в языке Си используется уловный оператор if. Условный оператор if может использоваться в форме полной или неполной развилки.

Фрагмент кода для неполной развилки выглядит следующим образом:

```
if (Условие)
{
```

```
  1_Фрагмент_Кода;
}
```

Фрагмент кода для полной развилки выглядит немного иначе:

```
if (Условие)
{
  1_Фрагмент_Кода;
}
else
{
  2_Фрагмент_Кода;
}
```

Если представить данный код в виде блок-схем, то они будут выглядеть следующим образом (рисунок 1.1):

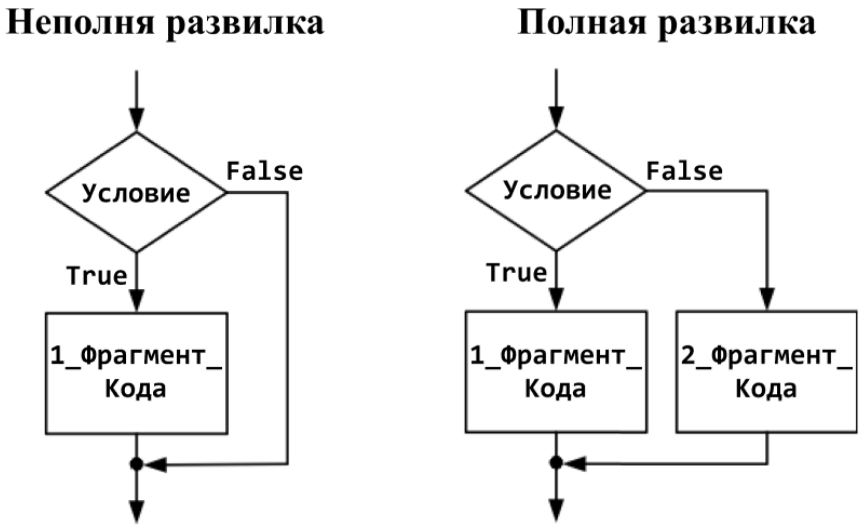


Рисунок 1.1 – Блок-схемы

Работает оператор if следующим образом: если условие в круглых скобках истинно, то выполняется программный код, заключенный в фигурных скобках; если же условие не верно, то будет выполнен код, представленный после оператора else, заключенный в фигурных скобках, при условии, что оператор else существует. При условии, что оператора else нет, при неверном условии в операторе if ничего выполняться не будет.

Причем если перед else несколько операторов if, то оператор else будет относиться к последнему if.

В качестве условия могут быть абсолютно разные комбинации.

Чаще всего используется сравнение чисел:

Представим, что num1, num2 – некоторые числа, то тогда можно записать большое количество сравнений, используя логические операторы сравнения.

```
if(num1 < num2) //программный код выполнится при условии что num1 меньше num2
{
  //программный код
}
if(num1 >= num2) //программный код выполнится при условии что num1 больше или
равен num2
```

```
{
  //программный код
}
if(num1 == num2) //программный код выполнится при условии, что num1 равен num2
{
  //программный код
}
else //к какому из if относится данный else?
{
 //программный код
}
```

Можно использовать любые операторы сравнения, которые были представлены в 2-3 лабораторных работах.

Если условие не одно, а несколько, то следует использовать логические операторы («ИЛИ» или «И»). Пример:

```
if((num1 < num2) || (num1 == 3)
{
  //программный код
}
```

В данном случае программный код выполнится при условии, если первое число меньше второго или если первое число равно 3.

То есть даже если число один будет больше второго числа, но при этом равно 3, код все равно выполнится.

Но если же нам нужно сделать условие, при котором код должен выполнится только в том случае, если первое число меньше второго и равно 3, то нам нужно использовать логический оператор «И»:

```
if((num1 < num2) && (num1 == 3))
{
  //программный код
}
```

1.3 Циклические операторы

Во время написания программ мы уже встречались с фрагментом кода, который выполняется циклически бесконечное число раз, пока включен микроконтроллер – loop().

Но что, если надо будет создать цикл внутри программы, который должен будет выполняться некоторое число раз. Например, для реализации подсчёта суммы алгебраической прогрессии (к этому примеру мы еще вернемся позднее).

Для таких задач используются циклические операторы, которые существенно позволяют сократить код.

Существуют два вида циклических операторов while и for.

Синтаксис циклического оператора for выглядит следующим образом:

```
for (Инициализация; Условие; Модификация)
{
  Фрагмент_кода;
```

```
}
```

for – параметрический цикл (цикл, который повторяется фиксированное число раз).

Для того чтобы данный цикл заработал, необходимо осуществить 3 операции:

Инициализация – присваивание параметру цикла начального значения;

Условие – проверка условия повторения цикла, чаще всего - сравнение величины параметра с некоторым граничным значением;

Модификация – изменение значения параметра для следующего прохождения тела цикла.

Но как же применять данный оператор на практике?

Вспомним пример, о котором говорилось выше. Допустим, нам нужно подсчитать сумму всех чисел арифметической прогрессии до 5.

Выглядит это примерно так:

$$1+2+3+4+5=15$$

Для реализации данного алгоритма воспользуемся циклом for:

```
int sum = 0;
for (int i=0; i<=5; i++)
{
  sum = sum + i;
}
```

Пронумеруем действия, которые будет выполнять программа при использовании данного цикла (рисунок 1.2):

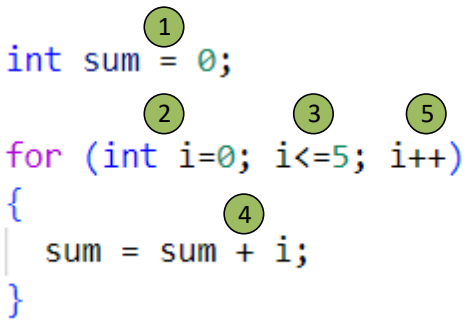


Рисунок 1.2 – Цикл for

При выполнении данного фрагмента программы будет выполнен ряд следующих действий:

1) Сначала создается переменная под названием sum, которая приравнивается к 0;

2) Далее в блоке инициализация создается переменная с названием i, которая также приравнивается к нулю;

3) Для того чтобы запустил фрагмент кода внутри оператора, оператор в блоке условия проверяет условие i<=5. Если оно верно, тогда выполняется фрагмент кода внутри цикла;

4) Выполняет фрагмент кода внутри цикла. К переменной sum прибавляется переменная i и результат сложения записывается в переменную sum;

5) В конце каждого цикла выполняется инструкция, которая указана в блоке модификация. В данной записи используется операция инкрементирования.

Инкремент – операция, которая выполняется для одного операнда, увеличивающая его значение на 1. (Данная операция доступна только для целочисленных данных). Запись i++ эквивалентна i = i + 1

6) Далее операции под номером 3,4,5 будут выполняться циклически до тех пор, пока будет справедливо условие i<=5.

Таким образом, мы реализуем ряд операций, которые позволяют посчитать сумму арифметической прогрессии с 0 до 5.

Цикл for не единственный циклический оператор в языке Си. В языке Си также присутствует циклический оператор под названием while.

Синтаксис циклического оператора while выглядит следующим образом:

```
while (Условие)
{
  Фрагмент_кода;
}
```

Синтаксис циклического оператора while очень похож на синтаксис условного оператора if.

Принцип работы также похож на принцип работы оператора if.

Различие заключается в том, что при выполнении условия в операторе if фрагмент кода выполнится единожды, в то время как в операторе while фрагмент кода будет выполняться до тех пор, пока условие будет справедливо.

Если написать программу для подсчёта суммы арифметической прогрессии от 0 до 5, используя цикл while, то код будет выглядеть следующим образом:

```
int sum = 0;
int i = 0;
while (i <= 5)
{
  sum = sum + i;
  i++;
}
```

Принцип работы данной программы следующий:

1) Изначально создается переменная sum и приравнивается к 0;

2) Далее создается переменная I и также приравнивается к 0;

3) После происходит проверка условия i <= 5, так как i = 0, то условие будет истинным, следовательно, фрагмент программы будет выполнен;

4) Далее переменная i добавится к переменной sum и результат запишется в переменную sum;

5) После переменная i увеличится на 1;

6) Далее произойдет проверка условия i <= 5, и до тех пор, пока данное условие будет выполняться, цикл будет выполнять фрагмент кода.

Существует еще один циклический оператор do while. Но, как правило, на практике он применяется реже. Его работу нужно изучить самостоятельно.

2. Практическая часть. Применение условных и циклических операторов в Arduino IDE

2.1 Плавное изменение яркости свечения светодиода

В первой работе мы написали программу, которая позволяет мигать светодиодом, используя лишь 2 состояния вкл/выкл (HIGH/LOW). А что, если мы захотим модернизировать процесс и сделать так, чтобы свечение светодиода было более плавным.

Для этого нужно воспользоваться ШИМ (Широтно-импульсная модуляция) сигналом (что это такое мы более подробно разберем в дальнейших работах).

Простыми словами, ШИМ сигнала позволяет имитировать аналоговый сигнал, который обладает несколькими уровнями. В нашем случае мы можем сгенерировать ШИМ сигнал, который содержит в себе 255 уровней. Т.е. мы сможем подать сигнал, который будет имитировать сигнал от 0В (LOW) до 3.3В (HIGH) с шагом $\frac{3.3В}{255} = 0{,}013В$

Для начала необходимо:

1. Создать новый проект;
2. Сохранить его;
3. Настроить плату.

Согласно тому, как это происходило в 1-ой лабораторной работе

Далее в поле setup() добавьте инициализацию пина, к которому подключен светодиод на выход!

(Можно посмотреть в 1-ой лабораторной работе)

Пином, к которому подключен светодиод, является PB1

```
void setup() {
   // put your setup code here, to run once:
   ***ВАШ КОД ИНИЦИАЛИЗАЦИИ ПИНА PB1 НА ВЫХОД***
}
```

Мы подключаем светодиод PB1, так как нам необходим светодиод, который подключен к пину, у которого есть возможность генерировать ШИМ сигнал. Таким и является пин PB1.

Но данный пин подключен через мультиплексор, следовательно, в главном меню необходимо будет выбрать RGB светодиод. Для этого необходимо в главном меню выбрать раздел «изменение периферии», в меню раздела «изменение периферии» выбрать «RGB светодиод». Убедитесь, что состояние диода изменилось из состояния «[выкл]» в состояние «[вкл]».

Перед прошивкой в данной лабораторной работе всегда необходимо проверять состояние RGB светодиода. Он должен быть во включенном положении.

Далее необходимо воспользоваться функцией, которая позволит нам выдавать на пин ШИМ сигнал определённого уровня.

Данной функцией является `analogWrite();`

Для того чтобы ей воспользоваться нужно написать следующий код:

```
void setup() {
 ***ВАШ КОД ИНИЦИАЛИЗАЦИИ ПИНА PB1 НА ВЫХОД***
}
void loop() {
   analogWrite(«Пин светодиода», «Уровень ШИМ сигнала»);
   Задержка;
}
```

Попробуйте вместо надписи `{Уровень ШИМ сигнала}` вставить своё значение от 0 до 255.

К примеру, 50, 100, 150, 200, 250.

Загрузите код в контроллер. И посмотрите, как изменится яркость свечения светодиода. Для того чтобы обеспечить плавное включение диода нужно сделать плавное изменение подаваемого ШИМ сигнала от 1 до 255. Для того чтобы это сделать, воспользуемся циклом:

```
void setup() {
  ***ВАШ КОД ИНИЦИАЛИЗАЦИИ ПИНА PB1 НА ВЫХОД***
}
void loop() {
  for (Инициализация; Условие; Модификация)
  {
    Подача ШИМ сигнала («Пин светодиода», «Уровень ШИМ сигнала»);
    Задержка;
  }
}
```

В цикле переменная i будет принимать значения от 1 до 255 с шагом 1, при этом мы увидим каждое значение с задержкой в 5 миллисекунд (это сделано с той целью, чтобы была возможность увидеть результат работы программы).

Загрузите полученный код в программу.

Как вы можете заметить, при выполнении данного кода светодиод плавно включается, но резко выключается.

Для того чтобы обеспечить плавное выключение светодиода напишите свой цикл, который позволит вам выполнить данную операцию.

Цикл должен обеспечить изменение переменной i от 255 до 1 с шагом 1 и записывать её в аргументы функции `analogWrite(PB1,i);`. Не забудьте про задержку!

```
void setup() {
   ***ВАШ КОД ИНИЦИАЛИЗАЦИИ ПИНА PB1 НА ВЫХОД***
}
void loop() {
 for (Инициализация; Условие; Модификация)
  {
    Подача ШИМ сигнала («Пин светодиода», «Уровень ШИМ сигнала»);
    Задержка;
  }
{Ваш цикл, который плавно выключает светодиод}
}
```

После того, как задача будет выполнена, продемонстрируйте работу вашей программы преподавателю.

Дополнительное задание:

Далее попробуйте изменить один из циклов for на цикл while, сохранив функционал программы* (Описание цикла while находится в теоретическом материале).

2.2 Изменение яркости свечения светодиода с помощью кнопки

В работе выше мы изменяли яркость свечения светодиода с помощью программного кода, но что если нам понадобится влиять на яркость свечения светодиода извне, не изменяя программу?

Для этого модернизируем код, который позволит нам менять значения, подаваемое в функцию `analogWrite(…,…);` с помощью оператора условия if:

Для этого инициализируйте в функции `setup()`:

Пин диода PB1 на выход. Пин кнопки PB4 на вход.

Далее в цикле loop напишите следующий программный код:

```
int i = 0;
```

```
void loop() {
  int but = «Считать с кнопки»;
  if(… == …) // если кнопка нажата
  {
    «здесь увеличение»; // Увеличиваем переменную i на 5
  }
  if(… >= …) // если i больше 255
  {
    i = 0; // обнуляем
  }
  Задержка;
  Подача ШИМ сигнала («Пин светодиода», «Уровень ШИМ сигнала»);
}
```

Данный код считывает цифровое значение с кнопки, которая подключена к пину PB4, и записывает в переменную but. Переменная but может принимать значения 0 и 1, в зависимости от того, нажата кнопка или нет.

1 – для нажатой кнопки.

0 – для не нажатой кнопки.

Далее мы создаем условие, при котором переменная i увеличивает свое значение на 5, при условии, что кнопка нажата.

Так функция `analogWrite();` может принимать значение от 0 до 255, следовательно, сделаем проверку, которая обнулит переменную I при преодолении значения 255.

В конце программы записываем полученное значение в функцию, и делаем небольшую задержку для того, чтобы отследить изменения.

2.3 Дополнительное задание

Реализуйте изменение свечения светодиод с помощью свитча, который подключен к пину PC14.

То есть когда свитч включен, светодиод меняет яркость, в ином случае – сохраняет свое состояние.

Работа № 4

«Основы программирования микроконтроллеров часть 2»

В языке СИ для удобства и облегчения программирования используются функции. Если в программе встречается некоторый фрагмент кода, который используется большое количество раз и лишь с небольшими изменениями (например, используется лишь другое значение), то можно данный фрагмент кода превратить в функцию и пользоваться ей внутри своей программы.

Функции делятся на два типа: бывают функции с возвратом значения и функции без возврата значения.

Ранее в работах уже были использованы такие функции, как `pinMode()`, `digitalRead()`, `digitalWrite()`, `analogWrite()` и другие. Отличие в типе функции можно рассмотреть на их примере, так `pinMode()` инициализирует тип пина и ничего не возвращает, а `digitalRead()` считывает цифровое значение с указанного пина и возвращает считанное значение. Также с `digitalWrite()` и `analogWrite()` устанавливает логическую единицу или нуль, или подает ШИМ сигнал на выбранный пин соответственно и в них нет возвращаемого значения.

Цель работы: Получить базовые навыки написания функций и работы с ними.

Задачи:

1) Изучить принцип работы функций;
2) Научиться передавать данные в функцию и принимать возвращаемое значение;
3) Научиться использовать библиотеки в Arduino IDE;
4) Научиться пользоваться директивой `#define`.

1. Теоретическая часть

1.1 Принцип работы функций

Существует два типа функций и бывают: функции с возвратом значения и функции без возврата значения. Также для работы некоторых функций им необходимо сообщить входные параметры. Количество необходимых параметров указывается при объявлении функции. Из этого можно сделать вывод, что функции принимают в себя входные параметры для того, чтобы внутри себя с ними взаимодействовать и вернуть или не вернуть результат этого взаимодействия. Например, для того, чтобы считать цифровое значение с определенного пина микроконтроллера, необходимо передать в функцию адрес пина, с которого необходимо считать и вернуть значение.

Аргументы, которые необходимо сообщить функции, указываются в круглых скобках, например, `digitalRead(PB4)`, где PB4 – пин, с которого нужно считать значение.

Также существуют функции, которые принимают в себя несколько значений аргументов. К примеру, функция для записи цифрового значения. Важной запомнить, что если функция принимает два и более значения аргументов, то необходимо их указывать через запятую, например, `digitalWrite(PB4, HIGH)`, где PB4 – пин, на который необходимо записать цифровое значение, а `HIGH` – значение, которое необходимо записать (в данном случае логическая единица).

Узнать тип и количество аргументов, которое необходимо сообщить функции, можно, воспользовавшись встроенным в Arduino IDE справочником. Доступ к справочнику можно получить следующим образом: перейти в меню «Помощь» и выбрать «Справочник». На рисунке 1.1 показано, как это можно сделать.

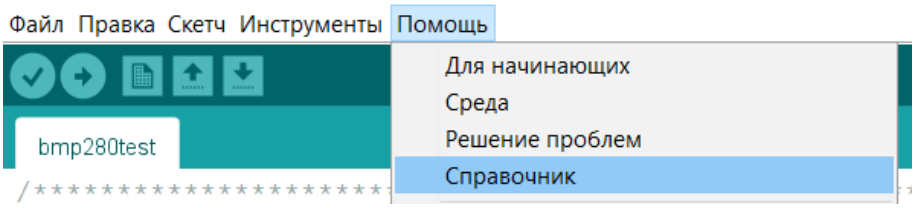


Рисунок 1.1 – Доступ к справочнику в Arduino IDE

Стоит отметить, что в справочнике вы получите информацию только о стандартных функциях для среды разработки Arduino IDE.

Часто для программирования микроконтроллеров необходимо воспользоваться уже существующими библиотеками, которые были написаны пользователями из сообщества Arduino программистов. Набор существующих функций для решения тех или иных задач записываются в библиотеку.

Библиотека – это некоторый фрагмент кода, в котором описаны функции и алгоритмы, которые доступны для использования пользователю Arduino IDE. Чаще всего в библиотеках написаны функции, которые позволяют взаимодействовать с модулями, которые подключены к микроконтроллеру. Для того чтобы подключить библиотеку, необходимо использовать директиву #include <Название библиотеки>.

Часто для решения задачи необходимо написать свои собственные функции, для этого необходимо воспользоваться следующим синтаксисом:

```
<тип данных, который возвращает функция> Имя_функции(тип_1_аргумента
название_1_аргумента, тип_2_аргумента название_2_аргумента и т.д.)
{
  //Фрагмент кода
  return возвращаемое значение
}
```

Рассмотрим написание функции на примере. В предыдущей работе был реализован код, позволяющий плавно включать и выключать светодиод. Напишем функцию, которая выполняет данную операцию.

Перед началом написания функции необходимо знать ответ на ряд вопросов:

1. Какое количество и тип значений функция будет принимать?
2. Будет ли функция возвращать значения, если да, то какие?

В случае нашего примера, функция выполняет алгоритм, и необходимости возвращать значение нет.

Ответом же на первый вопрос будет то, что для плавного включения и выключения светодиода необходимо принять адрес пина, к которому подключен светодиод, следовательно, нам, как минимум необходимо, передать в аргументы функции адрес пина, к которому подключен светодиод. Для этого необходимо воспользоваться типом данных void. Это тип данных, который является «пустым». Он ничего в себе не хранит и используется для создания функций, которые не возвращают значений.

Пример написания такой функции представлен ниже.

```
void smooth_blink(int pin) { // создание функции, которая принимает 1 аргумент
  // Реализуем алгоритм плавной подачи ШИМ
   for(int i=1;i<255;i++){
    analogWrite(pin, i);
    delay(5);}
```

```
  for(int i=255;i>1;i--){
    analogWrite(pin, i);
    delay(5);}
}
void setup() {
  pinMode(PB1, OUTPUT);
}
void loop() {
  smooth_blink(PB1);
}
```

Здесь можно увидеть, что функция объявляется всегда, прежде чем она будет использоваться. Это связано с тем, что в память устройства переменные и имена функций добавляются последовательно. То есть у вас не получится сначала воспользоваться переменной, а потом объявить ее – это же правило справедливо и для функций.

Загрузите данный код на плату и проверьте его работоспособность.

Для закрепления навыка работы с функциями, добавьте в функцию аргумент, который будет изменять задержку в каждом из циклов.

2. Практическая часть

2.1 Написание функции изменения цвета свечения RGB светодиода

Принцип работы RGB светодиода заключается в том, что в одном корпусе такого светодиода присутствует 3 светодиода (красный, зеленый, синий). лишнее для того, чтобы добиться определенного цвета, необходимо включать один из светодиодов с определенной интенсивностью.

Перед выполнением работы в меню платы «Изменение периферии» включите RGB светодиод.

RGB светодиод подключен к пинам PB1, PB9, PB8.

Но какой из цветов, подключен именно к этому пину?

Вам предстоит это выяснить. Для этого напишите следующую программу:
Последовательная подача сначала высокого, потом низкого напряжения на каждый диод. Вы должны запомнить порядок, в котором будете их включать и выключать, для того, чтобы запомнить к какому пину подключен светодиод с определенным цветом.

Основа для вашей программы должны выглядеть следующим образом:

```
#define diod PC13
#define R PB9
#define G PB8
#define B PB1
void setup() {
  pinMode(R, OUTPUT);
  pinMode(G,OUTPUT);
  pinMode(B, OUTPUT);
  pinMode(diod,OUTPUT);
}
void loop() {
  /*Для начала установим высокое напряжение на пин,
   где установлен обычный светодиод,
```

```
   для того, чтобы отследить начало программы */
  digitalWrite(diod, HIGH);
  delay(1000);
  digitalWrite(diod, LOW);
  delay(1000);
  /* Далее последовательно подаем напряжение (и убираем его) на каждый пин
   К которому подключен RGB светодиод */
   analogWrite(R,255);
   delay(500);
   analogWrite(R,0);
   delay(50);
   analogWrite(G,255);
   delay(500);
   analogWrite(G,0);
   delay(50);
   analogWrite(B,255);
   delay(500);
   analogWrite(B,0);
}
```

Здесь вы можете заметить пример использования директивы `#define`. `#define` – работает по принципу «найти и заменить», то есть перед выполнением программы если в тексте встретится *diod*, то он заменится на PC13. Данная конструкция удобна тем, что, переименовав пин в удобное нам название, мы будем использовать его, а при компиляции кода наше название заменится на номер пина, и программа выполнится корректно. Попробуйте написать код без переименования пинов RGB светодиода. Вы уже знаете, какой пин соответствует нужному цвету RGB-светодиода.

Проделанную работу необходимо продемонстрировать преподавателю.

Далее необходимо написать функцию, которая будет включать определенный цвет. Для основы можете воспользоваться следующим шаблоном.

```
void RGBgo (byte R_pin, byte G_pin, byte B_pin, byte R, byte G, byte B){
   analogWrite(R_pin,R);
   analogWrite(G_pin,G);
   analogWrite(B_pin,B);
}
void setup() {
  pinMode(R_pin, OUTPUT);
  pinMode(G_pin,OUTPUT);
  pinMode(B_pin, OUTPUT);
  pinMode(PC13,OUTPUT);
}
int R = 125;
int G = 255;
int B = 25;
void loop() {
RGBgo(R_pin, G_pin,B_pin,R,G,B);
}
```

Дополните программный код таким образом, чтобы у вас работала функция, и с ее помощью последовательно отображалось 3 цвета: красный, зеленый, синий с перерывом в одну секунду.

Если это задание выполнено верно, то сформируйте цвет одежды каждого участника группы также с перерывом в 1 секунду, используя вашу функцию.

Для определения уровня ШИМ сигнала можете воспользоваться программой Paint (рисунок 2.1).

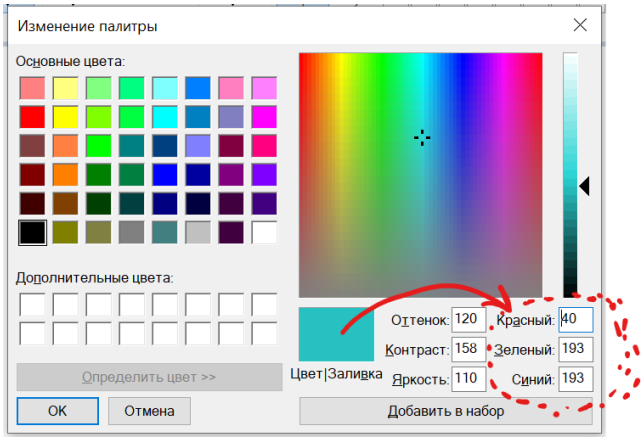


Рисунок 2.1 – Получения значений RGB с помощью Paint

2.2 Работа с библиотеками

Как говорилось ранее, для того чтобы расширить функционал работы вашей программы, не всегда обязательно самостоятельно писать код, описывающий этот функционал. Зачастую, для этого уже создана библиотека. Библиотеки могут быть представлены как встроенными библиотеками Arduino, так и пользовательскими, которые создаются пользователями Arduino IDE. Для того чтобы воспользоваться встроенными библиотеками Arduino IDE, их необходимо загрузить. Через вкладку «инструменты» необходимо выбрать пункт контекстного меню «Управлять библиотеками». Выглядит это следующим образом:

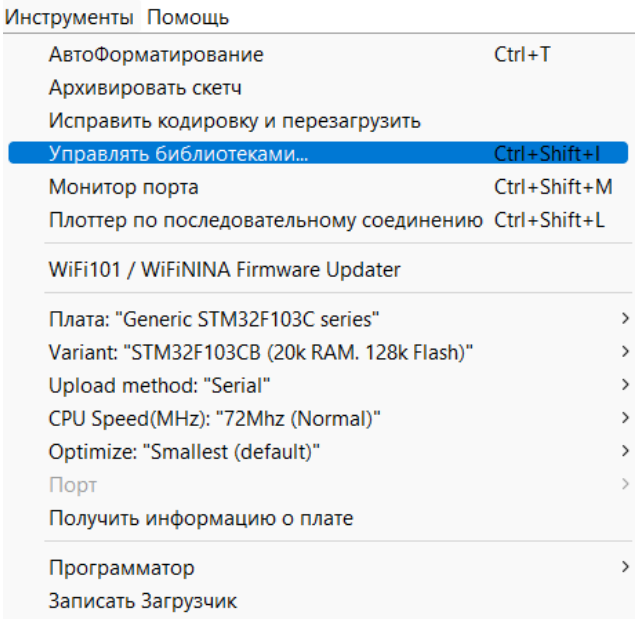


Рисунок 2.2 – Переход к управлению библиотеками

Далее откроется окно менеджера библиотек, которое выглядит следующим образом:

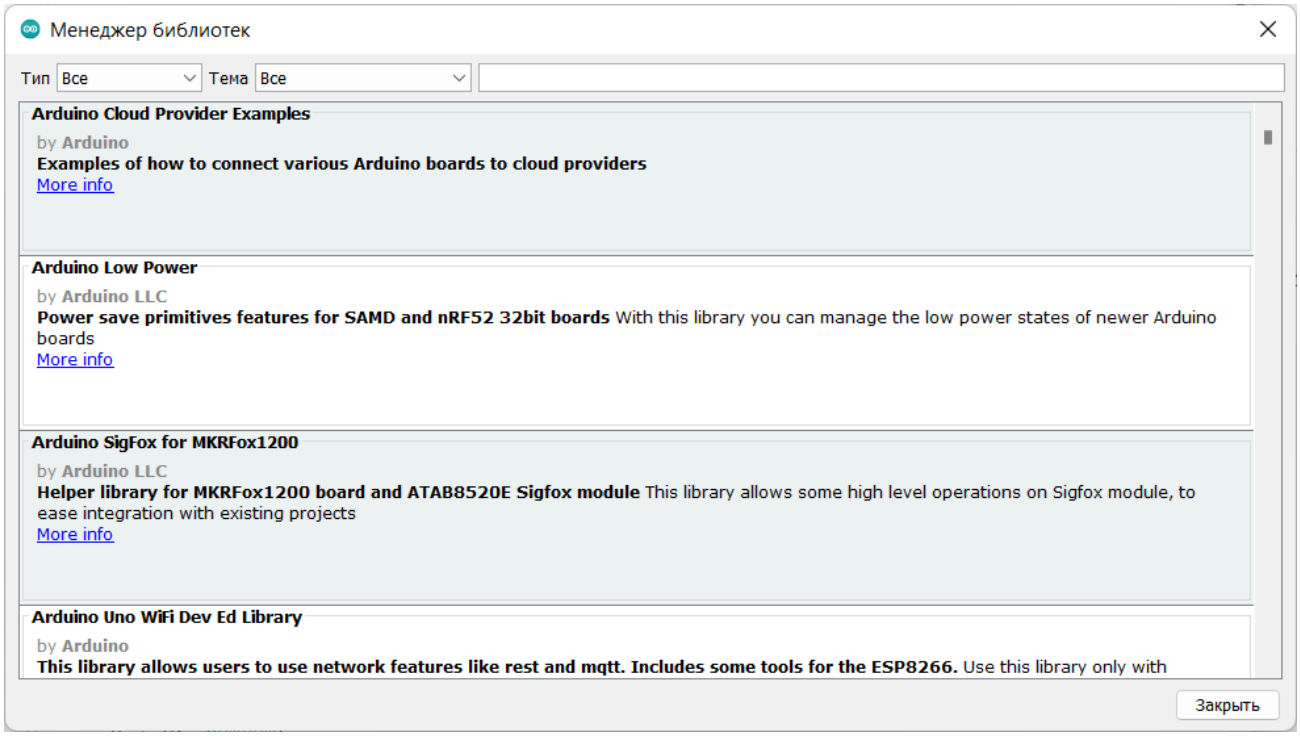


Рисунок 2.3 – Окно менеджера библиотек

В данном окне можно просмотреть как установленные библиотеки с их версией и обновлениями, так и осуществить поиск необходимой библиотеки для реализации работы.

В случае, если интересующей библиотеки не нашлось в списке стандартных библиотек, можно поискать библиотеку на сторонних ресурсах. Если следовать по данному сценарию, то скачанный файл с библиотекой необходимо поместить в папку вашего проекта и перезапустить Arduino IDE. В таком случае рядом с вкладкой вашего основного скетча появится вкладка скетча используемой библиотеки. Выглядит это следующим образом:

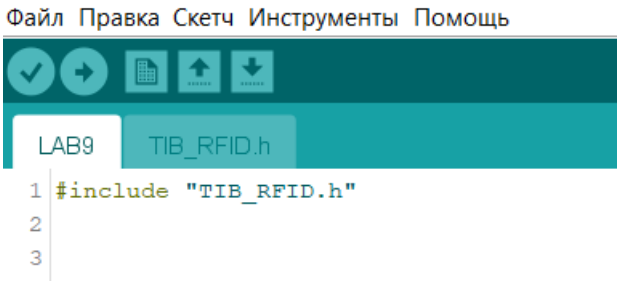


Рисунок 2.4 – Использование сторонней библиотеки

Как видно из рисунка 2.4, справа от основного скетча появилась вкладка со скетчем библиотеки.

Для использования библиотеки недостаточно того, что она добавлена в папку с проектом или скачана через «Менеджер библиотек». Чтобы библиотека заработала, и ваш основной скетч ее использовал, необходимо «подключить» библиотеку к основному скетчу. Подключение выполняется командой `#include “Название библиотеки”`, как это показано на рисунке 2.4. После чего в основном скетче можно описывать взаимодействие с библиотекой.

Рассмотрим вариант, в котором вынесем переименование пинов и написанную ранее функцию в отдельную библиотеку таким образом, чтобы библиотека имела следующий вид:

```
#define RGB_B PB1
#define RGB_R PB9
#define RGB_G PB8
#define but PB4
#define diod PC13

void RGBto(byte R_pin, byte G_pin, byte B_pin, byte R,byte G,byte B){
  analogWrite (R_pin, R);
  analogWrite (G_pin, G);
  analogWrite (B_pin, B);
  }
```

Данная библиотека уже создана и находится в папке с работами под названием «RGB.h». Вам необходимо скопировать ее в папку с вашим проектом и перезапустить Arduino IDE.

Далее модернизируйте код основного скетча таким образом, чтобы убрать из него все `#define` и саму функцию RGBto, при этом использовав подключение библиотеки. Также используйте полученные навыки в написании циклов, написав код, который будет в случайном порядке изменять цвет RGB светодиода. В качестве дополнительного задания реализуйте проверку на истинность цвета, используя условный оператор.

Конечный код программы будет выглядеть примерно так:

```
#include "RGB.h"
void setup() {
  pinMode(RGB_R, OUTPUT);
  pinMode(RGB_G, OUTPUT);
  pinMode(RGB_B, OUTPUT);
  pinMode(diod, OUTPUT);
  Serial.begin(115200);
}
void loop() {
    for (int i=1;i=100;i++){
      int R = random(191,193);
      int G = random(0,2);
      int B = random(191,193);
      RGBto(RGB_R, RGB_G, RGB_B, R, G, B);
      if(R==192&&G==0&&B==192){
        Serial.println("СОВПАЛО!!!");
        digitalWrite(diod,HIGH);
        delay(1000);
      }
      else{
        Serial.println("0");
        digitalWrite(diod,LOW);
      }
      delay(500);
    }
}
```

В данном коде используется функция `random`, которая принимает в себя первым аргументом наименьшее значение диапазона, а вторым аргументов – наивысшее, и возвращает псевдослучайное значение в заданном диапазоне. Важно знать, что данная функция возвращает псевдослучайное число в диапазоне от минимального (включительно) до максимального (не включительно).

Таким образом, если передать в функцию `random` аргументы (0, 1), то случайное число, возвращаемое функцией, всегда будет равняться 0, а если передать аргументы (-N, N) то функция будет возвращать псевдослучайное число от –N до N-1.

Работа № 5

«Обработка данных с кнопок и энкодеров»

Ранее на практических занятиях уже проводилась работа с кнопкой. Все мы интуитивно понимаем, что происходит при замыкании и размыкании кнопки. Кнопка – это ключ, состоящий из разорванного проводника и токопроводящей пластинки над ним. При нажатии, пластинка прижимается к проводникам и замыкает цепь, при отпускании – отжимается и размыкает. Однако если немного углубиться, то можно понять, что данная процедура не происходит мгновенно и равномерно. Это означает, что существует такой момент времени, когда кнопка пребывает в состоянии неопределенности или меняет состояние из разомкнутого в замкнутое и наоборот несколько раз за этот момент времени. Такие эффекты вызваны тем, что на поверхности контактов присутствуют микронеровности, не позволяющие им сомкнуться мгновенно.

Энкодер или ДУП (датчик угла поворота) – устройство, позволяющее преобразовывать угловое положение механизма в цифровой сигнал. Чаще всего энкодер применяется как элемент управления интерфейсом, в частности, как и на плате TUSUR IoT Board, где выполняет роль инструмента навигации по меню. Как и кнопка, ДУП подвержен дребезгу контактов.

Цель работы: Получить базовые навыки обработки данных с кнопок и энкодеров.

Задачи:

1) Изучить принцип работы кнопок и энкодеров;
2) Изучить принцип работы ползунковых переключателей;
3) Закрепить полученные знания с предыдущих работ.

1. Теоретическая часть

1.1 Обработка данных с кнопок и переключателей

Как вы можете помнить, в предыдущих работах мы уже обращались к кнопке. Важно напомнить, что у кнопки существует два типа подключения, изображенных на рисунке 1.1.

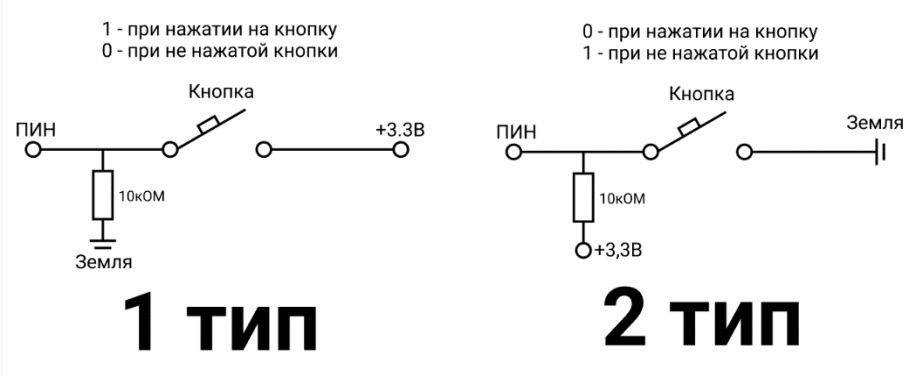


Рисунок 1.1 – Типы подключения кнопки

Исходя из рисунка 1.1, можно увидеть, что кнопка является ключом, размыкающим и замыкающим цепь. Также для изменения состояния замкнутости цепи используются и ползунковые переключатели. Ползунковый переключатель или «свитч» (от англ. switch - переключатель) – это та же кнопка, только с двумя фиксированными состояниями. Если кнопка имеет только одно фиксированное состояние и изменяет его только на момент нажатия, то свитч будет сохранять любое свое состояние (0 или 1) без необходимости его удержания. Внешний вид свитча представлен на рисунке 1.2.

Рисунок 1.2 – Внешний вид ползункового переключателя

Схематично устройство свитча не многим отличается от кнопки и выглядит следующим образом:

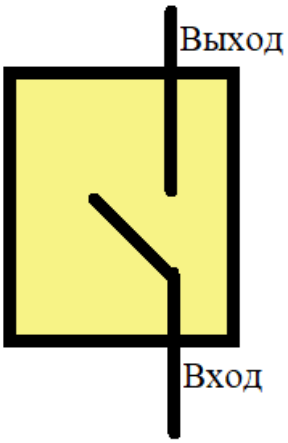


Рисунок 1.3 – Устройство двухпозиционного переключателя

В момент нажатия на кнопку или переключения переключателя изменяется состояние сигнала. Графически такое изменение представлено на рисунке 1.4.

Рисунок 1.4 – Графическое представление изменения сигнала при нажатии на кнопку или переключении свитча

Соответственно, как видно из рисунка, при нажатии на кнопку или переключении состояния свитча выход изменяет свое состояние из логического 0 в логическую 1. Это изменение позволяет выстраивать на его основе логику работы программы, используя операторы ветвления или, используя пару кнопок или свитчей для выполнения логических операций.

1.2 Обработка данных с энкодера

Энкодер – устройство, которое позволяет преобразовывать угловое положение механизма в цифровой сигнал. Чаще всего он применяется для управления интерфейсом устройств. Ранее вы уже встречались с энкодером в моментах взаимодействия с меню платы IoT TUSUR Board. Визуально энкодер выглядит следующим образом:

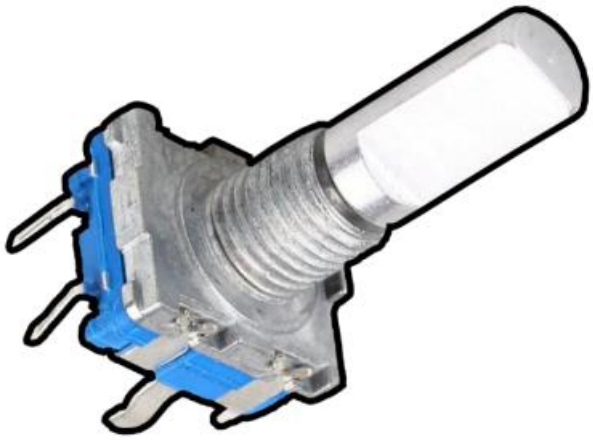


Рисунок 1.5 – Датчик угла поворота (энкодер)

Датчик угла поворота, зачастую, имеет следующую структуру, включающую в себя 2 ключа и одну кнопку:

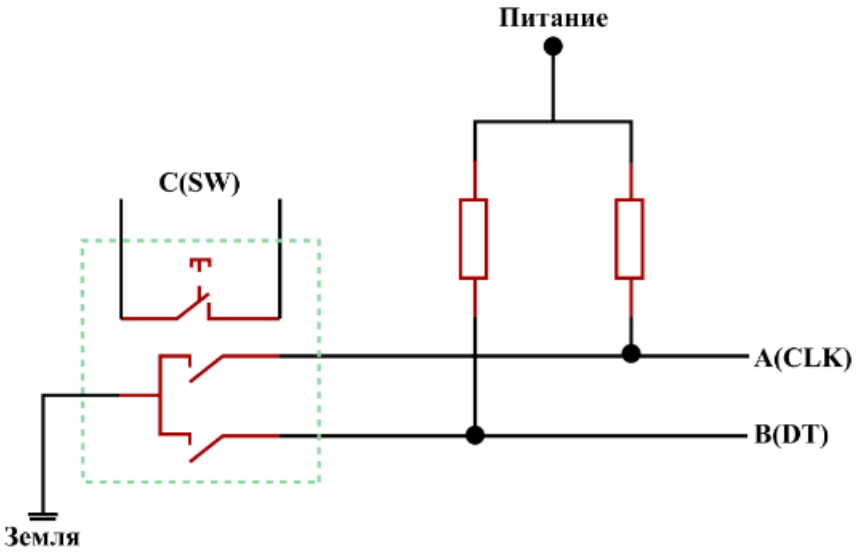
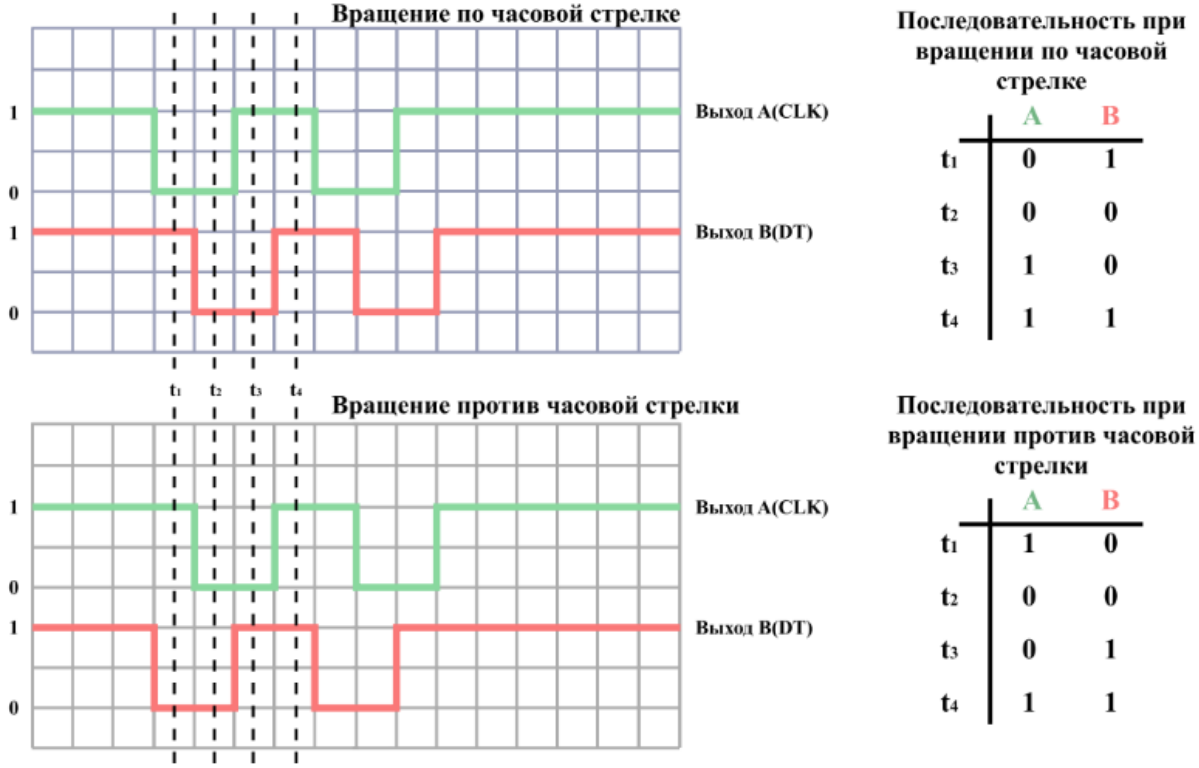


Рисунок 1.6 – Схема датчика угла поворота (энкодера)

Наличие двух ключей позволяет определять: в какую сторону был повернут датчик угла поворота (ДУП). При повороте ручки замыкается сначала ключ (А), а следом, через определенный промежуток времени, ключ (В), следовательно, при повороте в другую сторону ключи замыкаются в обратном порядке. Данная закономерность позволяет определить: в какую именно сторону был повернут ДУП в данный момент времени и использовать эту информацию для взаимодействия с системами. Сигнал, формирующийся при повороте ручки ДУП, представлен на рисунке 1.7.



Последовательность при вращении по часовой стрелке

	A	B
t_1	0	1
t_2	0	0
t_3	1	0
t_4	1	1

Последовательность при вращении против часовой стрелки

	A	B
t_1	1	0
t_2	0	0
t_3	0	1
t_4	1	1

Рисунок 1.7 – Вид сигнала при повороте ручки

Таким образом, зная последовательность, которая поступает на вход микроконтроллера при повороте ручки ДУП, можно ее декодировать для распознавания сигнала.

2. Практическая часть

2.1 Обработка данных с энкодера

В данной работе необходимо реализовать работу звукового маячка («зуммера») с изменением частоты звука при помощи энкодера. Для данной работы понадобится следующая периферия:

1. Энкодер (обозначения пинов: SW, CLK, DT);
2. Зуммер (обозначение пина: zoom);
3. Кнопка (обозначение пина: but).

Для удобства написания кода, воспользуемся уже известным скриптом `#define` и переименуем все пины в их обозначения:

```
#define CLK PB13
#define DT PB12
#define SW PA15
#define zoom PA8
#define but PB4
```

В данной работе уже используется функция, которая позволяет быстро обработать данные с энкодера при помощи таблиц, представленных на рисунке 1.7. Данная функция выглядит следующим образом:

```
long pos = 0;
byte lastState = 0;
long last_pos = 0;
/*Таблица по которой происхожит чтение данных с энкодера*/
const int8_t increment[16] = {0, -1, 1, 0, 1, 0, 0, -1, -1, 0, 0, 1, 0, 1, -1,
0};
/*Функция которая возвращает 3 значения*/
/* 1 если ручка повернута по часовой стрелке*/
/* 2 если ручка повернута против часовой стрелки*/
/* 0 если никаких изменений не произошло*/
uint8_t enc_read(uint8_t clk, uint8_t dt) {
byte state = digitalRead(clk) | (digitalRead(dt) << 1);
if (state != lastState) {
pos += increment[state | (lastState << 2)];
lastState = state;
if (last_pos < pos){
last_pos = pos;
return 1;
}
if (last_pos > pos){
last_pos = pos;
return 2;
}
```

```
}
return 0;
}
```

Вставьте код с переименованием пинов и данную функцию в ваш скетч, а затем инициализируйте пины:

```
void setup() {
pinMode(CLK, ???);
pinMode(DT, ???);
pinMode(SW, ???);
pinMode(zoom, ???);
pinMode(but, ???);
}
```

Вместо вопросительных знаков вставьте верное значение. А затем в loop () необходимо написать следующий программный код:

```
uint16_t val = 0;
void loop() {
/*Получем значение с энкодера*/
uint8_t enc_state = enc_read(CLK, DT);
/*Если по час. стр. то увеличиваем значение переменной*/
if (???){
val++;
}
/*Если против час. стр. то уменьшаем*/
else if (???){
???
}
/*Если нажата кнопка энкодера или просто кнопка, то*/
if(???){
/*Подаем значение на зуммер*/
tone(zoom, val*25, 1000);

8

?Задержка на 5 миллисекунд?
}
/*В ином случае заглушаем зуммер*/
else{
???
}
/*Для конечного проекта можно реализовать систему, которая позволит менять
множитель октавы в зависимости от свитчей*/
/*Или сделать гитару используя фоторезисторы*/
}
```

Завершите данный код, заполнив пробелы, обозначенные знаками вопроса. Загрузите скетч на плату и проверьте его работоспособность. Примерный конечный работающий код программы представлен ниже:

```
#define CLK PB13
#define DT PB12
#define SW PA15
#define zoom PA8
#define but PB4
/*----------------------------*/
/*Глобальные переменные для обработки данных с энкодера*/
long pos = 0;
byte lastState = 0;
long last_pos = 0;
/*Таблица по которой происхожит чтение данных с энкодера*/
const int8_t increment[16] = {0, -1, 1, 0, 1, 0, 0, -1, -1, 0, 0, 1, 0, 1, -1,
0};
/*Функция которая возвращает 3 значения*/
/* 1 если ручка повернута по часовой стрелке*/
/* 2 если ручка повернута против часовой стрелки*/
/* 0 если никаких изменений не произошло*/
uint8_t enc_read(uint8_t clk, uint8_t dt) {
    byte state = digitalRead(clk) | (digitalRead(dt) << 1);
  if (state != lastState) {
    pos += increment[state | (lastState << 2)];
    lastState = state;
    if (last_pos < pos){
      last_pos = pos;
      return 1;
    }
    if (last_pos > pos){
      last_pos = pos;
      return 2;
    }
  }
  return 0;
}

void setup() {
  /*Объявление пинов*/
  pinMode(CLK, INPUT);
  pinMode(DT, INPUT);
  pinMode(SW, INPUT);
  pinMode(zoom, OUTPUT);
  pinMode(but, OUTPUT);
}
uint16_t val = 0;

void loop() {
```

```
  /*Получем значение с энкодера*/
  uint8_t enc_state = enc_read(CLK, DT);
  /*Если по час. стр. то увеличиваем значение переменной*/
 if (enc_state==1){
val++;
}
  /*Если против час. стр. то уменьшаем*/
else if (enc_state==2){
val--;
}
  /*Если нажата кнопка энкодера или просто кнопка, то*/
  if(digitalRead(but)||!digitalRead(SW)){
    /*Подаем значение на зуммер*/
    tone(zoom, val*25, 1000);
    delay(5);
  }
  /*В ином случае заглушаем зуммер*/
  else{
    tone(zoom,0);
}
}
```

3. Последовательный порт

3.1 Работа с последовательным портом

Работа с последовательным портом связана с интерфейсом UART. С интерфейсом UART знакомство будет несколько позже, но работа с последовательным портом необходима уже сейчас. Для работы с последовательным портом используются следующие команды:

Таблица 3.1 – Перечень и значение команд для работы с последовательным портом

Serial.begin(speed)	Запускает связь по Serial на скорости speed (измеряется в baud, бит в секунду). Скорость можно поставить любую, но есть несколько “стандартных” значений.
Serial.end()	Прекращает связь по Serial. Также освобождает пины RX и TX.
Serial.available()	Возвращает количество байт, находящихся в буфере приёма и доступных для чтения.
Serial.availableForWrite()	Возвращает количество байт, которые можно записать в буфер последовательного порта, не блокируя при этом функцию записи.
Serial.write(val), Serial.write(buf, len)	1) Отправляет в порт val, численное значение или строку, 2) Отправляет количество len байт из буфера buf. Отправляет данные по таблице ASCII
Serial.print(val), Serial.print(val, format)	Отправляет в порт значение val – число или строку, фактически “печатает”. В отличие от write выводит именно текст. format позволяет настраивать систему счисления для вывода данных: BIN, OCT, DEC, HEX Цифра после вывода

	float позволяет настраивать выводимое количество знаков после точки
Serial.println(), Serial.println(val), Serial.println(val, format)	Полный аналог print(), но автоматически переводит строку после вывода. Позволяет также вызываться без аргументов (с пустыми скобками) просто для перевода курсора на новую строку.
Serial.flush()	Ожидает окончания передачи данных.
Serial.peek()	Возвращает текущий байт с края буфера, не убирая его из буфера. При вызове Serial.read() будет считан тот же байт, но из буфера уже уберётся.
Serial.read()	Читает и возвращает крайний символ из буфера.
Serial.setTimeout(time)	Устанавливает time (миллисекунды) таймаут ожидания приёма данных для следующих ниже функций.
Serial.find(target), Serial.find(target, length)	Читает данные из буфера и ищет набор символов target (тип char), опционально можно указать длину length. Возвращает true, если находит указанные символы. Ожидает передачу по таймауту.
Serial.findUntil(target, terminal)	Читает данные из буфера и ищет набор символов target (тип char) либо терминальную строку terminal. Ожидает окончания передачи по таймауту, либо завершает приём после чтения terminal.
Serial.readBytes(buffer, length)	Читает данные из порта и закидывает их в буфер buffer (массив char[] или byte[]). Также указывается количество байт, который нужно записать – length (чтобы не переполнить буфер).
Serial.readBytesUntil (character, buffer, length)	Читает данные из порта и закидывает их в буфер buffer (массив char[] или byte[]), также указывается количество байт, который нужно записать – length (чтобы не переполнить буфер) и терминальный символ character. Окончание приёма в buffer происходит при достижении заданного количества length, при приёме терминального символа character (он в буфер не идёт) или по таймауту
Serial.readString()	Читает порт, формирует из данных строку String, и возвращает её (урок про стринги). Заканчивает работу по таймауту.
Serial.readStringUntil(terminator)	Читает порт, формирует из данных строку String, и возвращает её. Заканчивает работу по таймауту или после приёма символа terminator (символ char).
Serial.parseInt(), Serial.parseInt(skipChar)	Читает целочисленное значение из порта и возвращает его (тип long). Заканчивает работу по таймауту. Прерывает чтение на всех знаках, кроме знака – (минус). Можно также отдельно указать символ skipChar, который нужно пропустить, например кавычку-разделитель тысяч (10’325’685), чтобы принять такое число
Serial.parseFloat()	Читает значение с плавающей точкой из порта и возвращает его. Заканчивает работу по таймауту.

Работа № 6

«Изучение работы аналого-цифрового преобразователя (АЦП)»

У каждого на слуху «цифровое ТВ», «цифровой сигнал». Как было сказано ранее, в цифровых системах передается цифровой сигнал. Давайте разберемся с тем, что же такое «аналоговый сигнал» и «цифровой сигнал». Аналоговый сигнал — сигнал данных, у которого каждый из представленных параметров описывается функцией времени и непрерывным множеством возможных значений. Цифровой сигнал – это такой сигнал, который является счетным (дискретным) как по времени, так и по амплитуде. Отсюда можно сделать вывод, что аналоговый сигнал непрерывен и бесконечен по времени и вмещает в себя множество возможных значений амплитуды. В то время как цифровой сигнал дискретен, конечен во времени и имеет определенный набор значений амплитуды. Микроконтроллер не может воспринимать аналоговый сигнал как таковой, он воспринимает только битовую последовательность. Для того чтобы преобразовать аналоговый сигнал в цифровой, используют аналогово-цифровой преобразователь. Аналогово-цифровой преобразователь (АЦП) – устройство, преобразующее входной аналоговый сигнал в цифровой (дискретный код). После прохождения аналогово сигнала через АЦП он преобразуется в битовую последовательность понятную ЭВМ.

Цель работы: изучения принципа работы аналогово-цифрового преобразователя и его применение на практике.

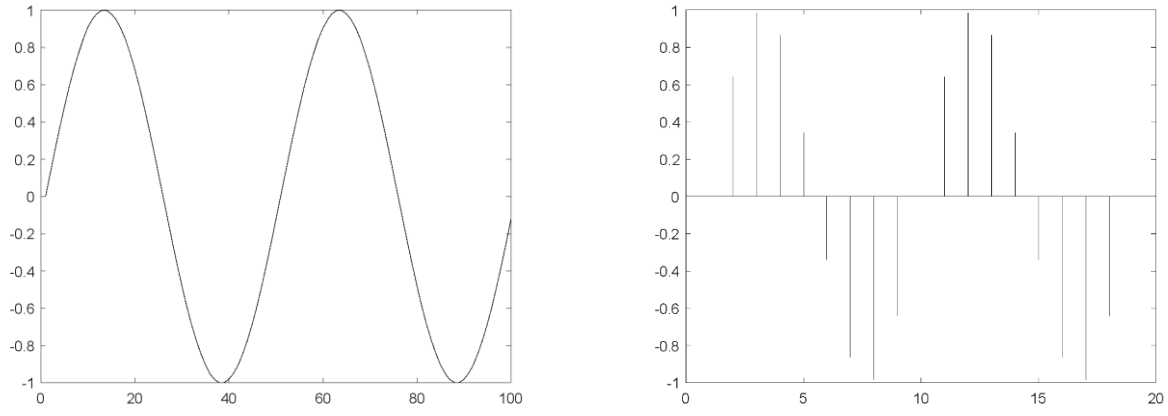
Задачи:

1) Изучить принцип работы АЦП;
2) Изучить реализацию работы АЦП в микроконтроллерах;
3) Получить навыки по обработке данных с аналоговых датчиков;
4) Получить и обработать данные с переменного резистора и микрофона.

1. Теоретическая часть

1.1 Принцип работы аналого-цифрового преобразователя

Снимая напряжение с некоторых датчиков, мы получаем непрерывный сигнал, что, в свою очередь, означает, что данный сигнал определен в каждый момент времени его существования, в то время как дискретный сигнал известен только в определенные промежутки времени (отсчеты). Графическое представление аналогового и цифрового сигналов представлены на рисунке 1.1.



а) б)

Рисунок 1.1 – Графическое представление а) – аналогового (непрерывного) и б) – дискретного сигнала

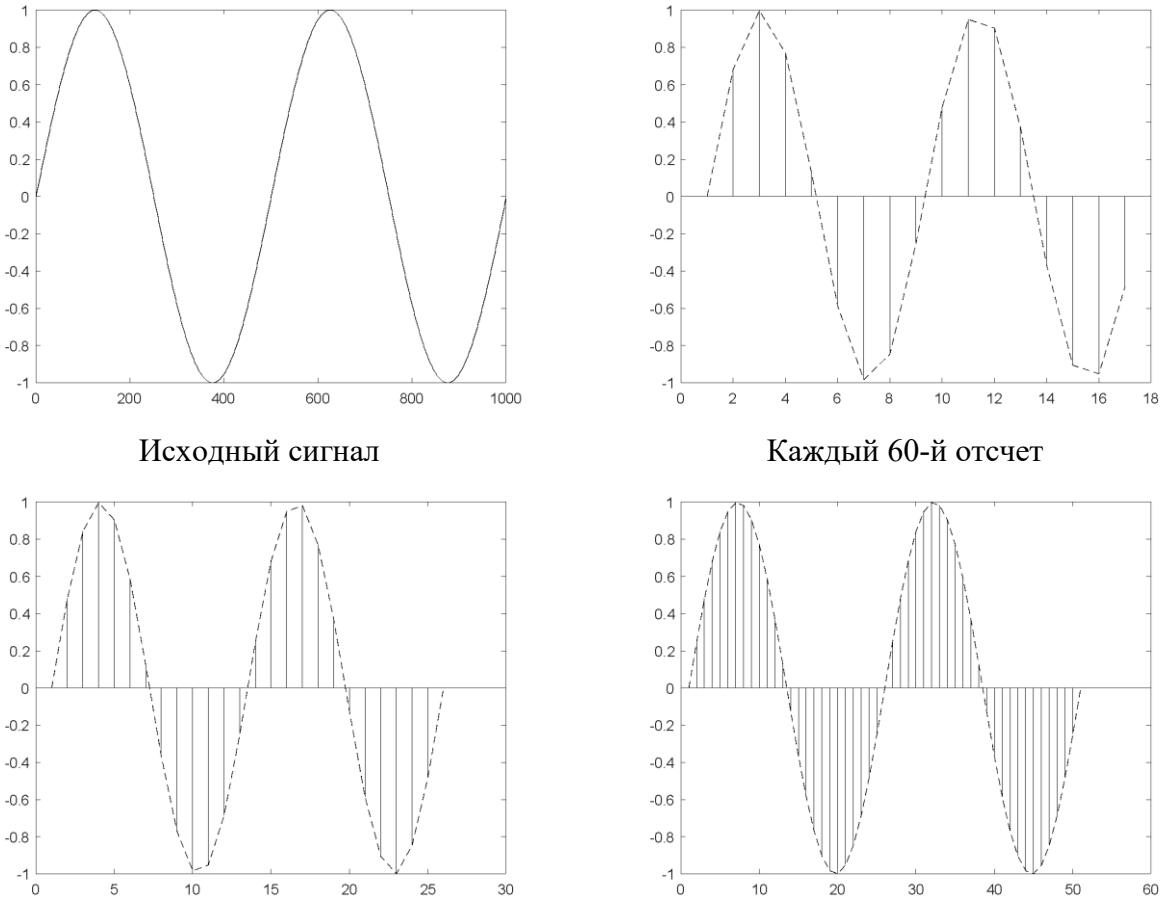
Для того чтобы из аналогового сигнала получить цифровой (дискретный) сигнал, необходимо выполнить следующие операции над ним:

1. Дискретизация по времени;
2. Квантование по уровню;
3. Кодирование.

1.1.1 Дискретизация по времени

Дискретизация по времени – это процесс выбора отсчетов аналогового сигнала с заданной частотой, то есть при поступлении аналогового сигнала можно снимать значения каждый определенный промежуток времени.

На рисунке 1.2 представлен пример дискретизации сигнала по времени со следующим набором отсчетов (каждый 60, 40 и 20-й отсчет), также это может быть определенная единица времени, например, каждые 20, 40, 60 мс.



Исходный сигнал | Каждый 60-й отсчет

Каждый 40-й отсчет | Каждый 20-й отсчет

Рисунок 1.2 – Пример дискретизации сигнала по времени

Из рисунка 1.2 можно заметить, что чем чаще берутся отсчеты, тем более точный сигнал мы получаем на выходе. Но чем больше отсчетов мы берем, тем больше отсчетов сигнала придется хранить в памяти устройства. Поэтому всегда нужно соблюдать баланс между точностью сигнала и количеством места, которое будет занимать оцифрованный сигнал.

Важно помнить, что минимальная частота дискретизации, согласно теореме Котельникова:

$$f_D \geq 2 \cdot f_S ,$$

где f_D – частота дискретизации;

f_S – частота сигнала.

1.1.2 Квантование по уровню

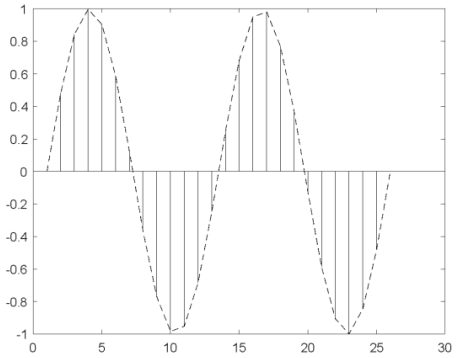
После дискретизации по времени сигнал стал счетным во временной области, то есть стал существовать только в определенные моменты времени. Однако, значение данных отсчетов до сих пор являются непрерывной величиной, следовательно, нам нужно добавить уровни, к которым мы будем записывать сигнал в память устройства.

Количество уровней определяется разрядностью АЦП. Допустим, мы захотим воспользоваться 16-ти уровневым АЦП. Это означает, что наш сигнал (который в данный момент может времени может принимать любое значение от -1 до 1 включительно) может принадлежать только одному из 16-ти уровней в зависимости от того, к какому уровню отсчет ближе всего. То есть сигнал может принимать значения от -1 до 1 с шагом:

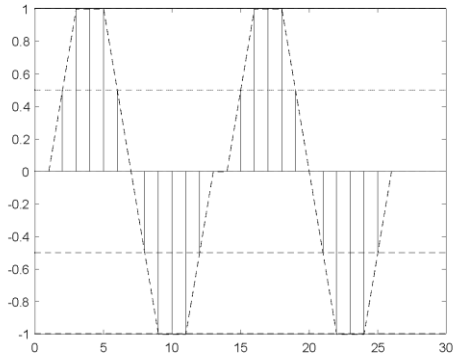
$$U_n = \frac{(1-(-1))}{16} = 0{,}125\,\text{В}.$$

Т.е. ряд уровней по напряжению будет выглядеть так: -1, -0.875, -0,750, -0.625, -0.500, -0.375, - 0,250, - 0.125, 0, 0,125, 0,250, 0.375, 0.500, 0.625, 0,750, 0.875, 1.

Для того чтобы хранить 16 уровней, необходимо использовать $\log_2(16) = 4$ бита.

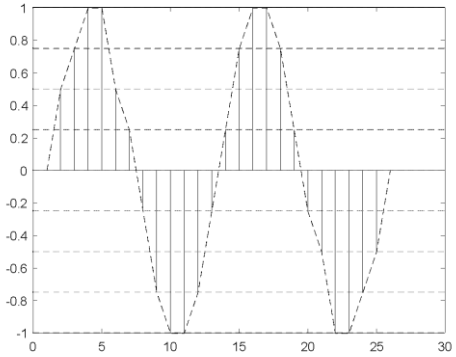


Дискретный во времени сигнал

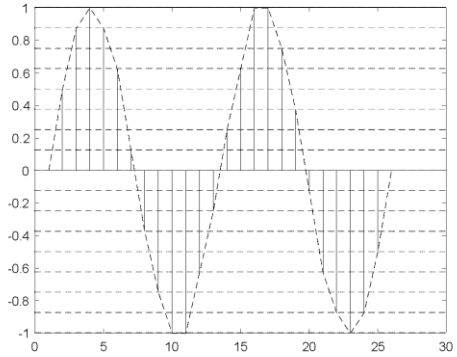


Четыре уровня квантования

Рисунок 1.3 – Квантование сигнала



Восемь уровней квантования



Шестнадцать уровней квантования

Рисунок 1.4 – Влияние уровней квантования на точность сигнала

Как видно из рисунков 1.3 и 1.4, чем больше количество уровней квантования, тем больше точность сигнала на выходе с АЦП. В то же время, как и с частотой дискретизации сигнала при дискретизации по времени, так и с количеством уровней квантования, чем

больше значений хранится, тем больше места занимает оцифрованный сигнал в памяти микроконтроллера.

1.1.3 Кодирование

При получении оцифрованного сигнала, который является счетным по времени и по амплитуде, остается вопрос – а как передать эти данные микроконтроллеру? Данную проблему решает кодирование.

Получив сигнал, у которого есть определенное количество уровней по амплитуде, нетрудно получить количество бит, которых потребуется для передачи данных уровней.

Например, для 8 уровней квантования потребуется $\log_2 8 = 3$ бит информации, то есть уровни будут представлять собой последовательность нулей и единиц по три бита каждый. Пример для восьмиуровневого кодирования представлен в таблице 1.1.

Нетрудно заметить, что полученные результаты могут представлять собой:

– последовательность нулей и единиц;

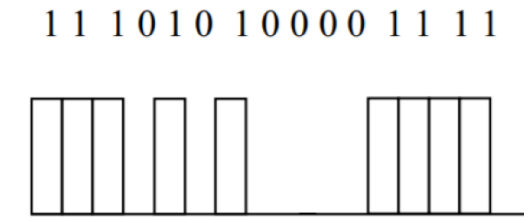
– последовательность только нулей;

– последовательность только единиц.

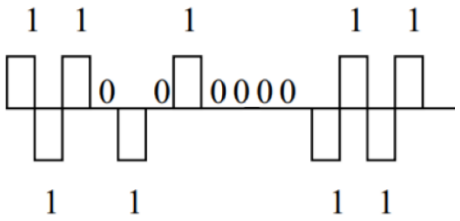
Таблица 1.1 – Кодирование 8-ми уровней

Уровень	Код
1	000
2	001
3	010
4	011
5	100
6	101
7	110
8	111

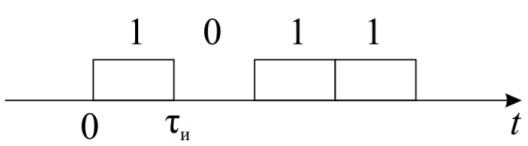
Передавать данные последовательности можно используя один из видов кодирования. К примеру, на рисунке 1.5 представлены самые известные виды кодирования.



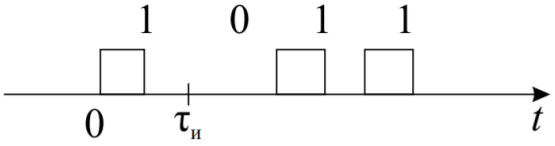
Однополярный код ($0 \to 0{,}1 \to 1$)



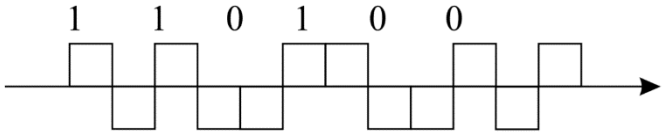
Биполярный код ($0 \to -1{,}1 \to 1$)



NRZ код (Нет возврата к нулю)



RZ код (Есть возврат к нулю)



Абсолютный биимпульсный код

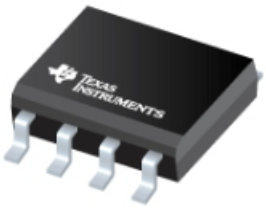
Рисунок 1.5 – Примеры кодирования

1.2 Датчик и обработка данных с датчика

Датчик – устройство, предназначенное для измерения какой-либо величины и обработки результата измерения. Схема датчика генерирует сигнал в удобной для передачи форме, дальше сигнал обрабатывается или хранится. Датчики используются повсеместно: от вашего смартфона до сфер медицины и промышленности.

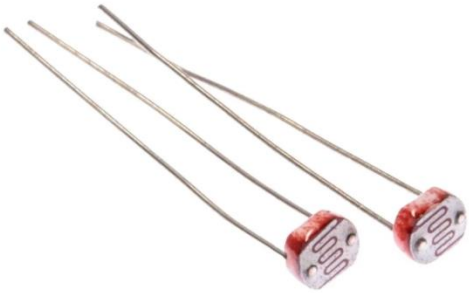
Датчик содержит в своей конструкции чувствительный элемент и преобразовательную часть. Главными характеристиками электронных датчиков являются их чувствительность и погрешность измерения. В зависимости от преобразовательной части существует два вида датчиков: аналоговые датчики и цифровые датчики.

На плате TUSUR_IOT_BOARD присутствуют следующие датчики и модули для изменения напряжения, представленные на рисунке 1.6.



Датчик температуры (LM74A)

Конденсаторный микрофон(Датчик звука)



Фоторезистор (Датчик света)

Потенциометр (Делитель напряжения)

Рисунок 1.6 – Датчики и модули на плате TUSUR_IOT_BOARD

Аналоговый датчик генерирует на выходе аналоговый сигнал, значение уровня которого получается функцией времени, и изменение такого сигнала происходит непрерывно. Для того чтобы считать данное значение микроконтроллером, необходимо воспользоваться АЦП, принцип работы которого был описан выше.

2. Практическая часть

2.1 Получение данных с аналоговых датчиков TUSUR_IOT_BOARD

В Arduino IDE получить данные с аналоговых датчиков достаточно просто. Необходимо подключить их к пинам, у которых есть возможность отправить полученный сигнал на АЦП.

На плате TUSUR_IOT_BOARD все датчики подключены к пинам, у которых есть доступ к АЦП.

Для оцифровки аналогового сигнала используется функция analogRead().

Перед выполнением работы в меню платы “Изменение периферии” включите переменный резистор (ПР). Определите, к какому пину подключен ПР, микрофон, и фоторезисторы. Определить пин к которому подключено устройство можно при помощи главного меню платы. Для этого перейдите в раздел «Распиновка».

Также перед началом работы запишите в тетради или создайте таблицу 2.1.

Таблица **Ошибка! Текст указанного стиля в документе отсутствует.**.1 – Таблица значений для заполнения

Максимальное значение, полученное с АЦП при использовании ПР	
Количество уровней АЦП	уровней
Разрядной АЦП	бит
Значение микрофона в спокойном состоянии	
Минимальное значение микрофона при громких звуках	
Значение освещения с 1-го фоторезистора	
Значение освещения со 2-го фоторезистора	
Значение освещения с 3-го фоторезистора	
Усредненное значение с 3-х фоторезисторов	

Создайте новый проект и объявите необходимые переменные, они все понадобятся нам в ходе работы:

```
void setup() {
      // put your setup code here, to run once:
      /* Назначение всех пинов для исследования на ввход*/
      Serial.begin(9600);
      /*-----------------------------------------------*/
      /* Назначение свитча на вход*/

      /*Инициализация пинов с RGB светодиодом*/

      /*--------------------------------------*/
```

В данной работе появляется незнакомая функция `Serial.begin(9600);`

Данная функция инициализирует определенный порт для передачи данных по UART. Это необходимо для того, чтобы отправлять полученные данные на компьютер для их анализирования.

Также с помощью UART можно отправлять команды с компьютера на микроконтроллер.

Подробнее работу UART мы изучим в следующих работах.

Далее необходимо написать программу, которая будет отправлять данные в UART, которые будут считанные с пина с АЦП, к которому подключен ПР.

Для того чтобы контролировать поток данных, добавим условие if, которое будет считывать значение со свитча, и программа будет работать только при включенном положении свитча.

```
void loop() {
    if () { //Начало обработки данных при включенном свитче
      /*Сбор данных с переменного резистора*/
        float " " = ; Создать переменную и записать в нее данные с ПР
            Serial.println(" ");
        /*---------------------------*/
```

```
        Задержка;
    }
}
```

Напишите программный код.

Перед загрузкой кода убедитесь в том, что в меню периферии включен переменный резистор!

Как только код будет загружен, выберите в выпадающем меню «Инструменты» вкладку «Плоттер по последовательному соединению» (рисунок 2.1) или нажмите комбинацию клавиш Ctr+Shift+L:

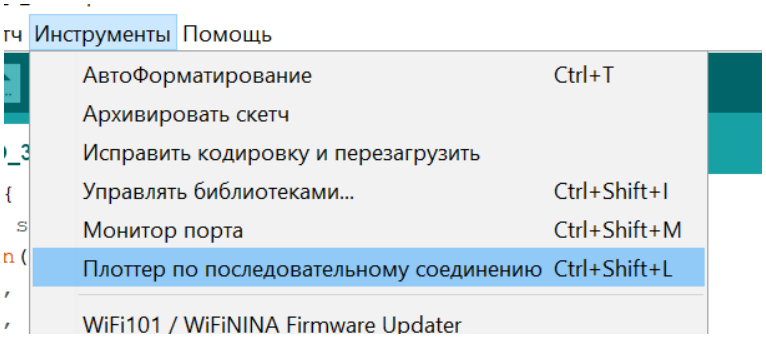


Рисунок 2.1 – Запуск плоттера

После выполнения у вас должно открыться следующее окно:

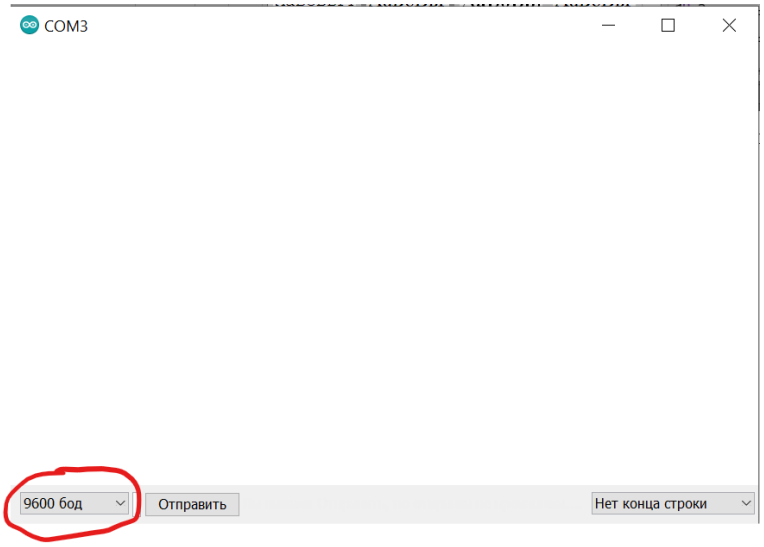


Рисунок 2.2 – Окно плоттера

Важно! Проверьте, что скорость, указанная в коде, совпадает со скоростью считывания, указанной в плоттере! В Arduino IDE плоттер имеет особенность автоматического масштабирования. Поэтому если процесс переходит в состояние покоя, то масштаб становится максимальным, и на выходных данных можно увидеть ошибки АЦП.

Для того чтобы пошли данные, надо установить свитч во включенное положение, и должна появится подобная картина (рисунок 2.3):

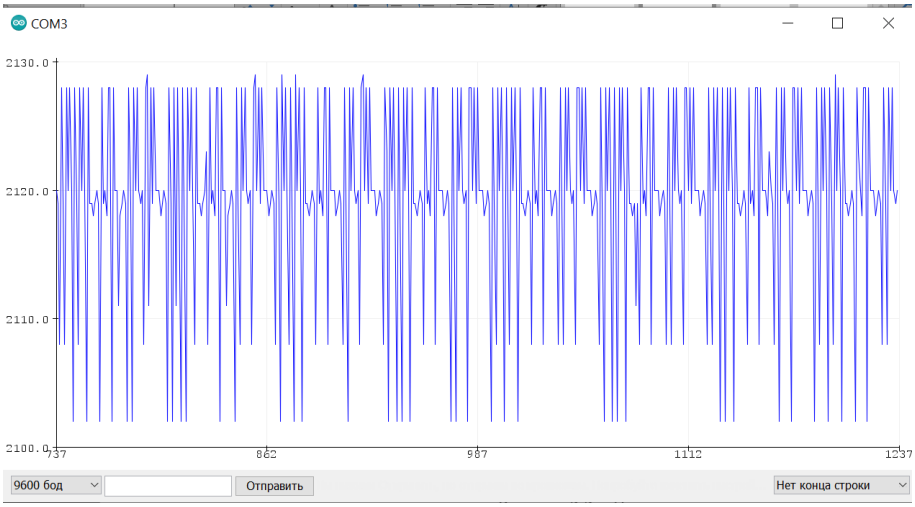


Рисунок 2.3 – Снятие данных с ПР

Но это не единственный способ того, как можно получить данные с COM порта в Arduino IDE. Также можно посмотреть эти данные в численном представлении в мониторе COM-порта. Для этого нажмите клавиши Ctrl+Shift+M или нажмите данную иконку  в правом верхнем углу (плоттер должен быть закрыт!). Откроется подобное окно, где можно увидеть цифровые данные (рисунок 2.4):

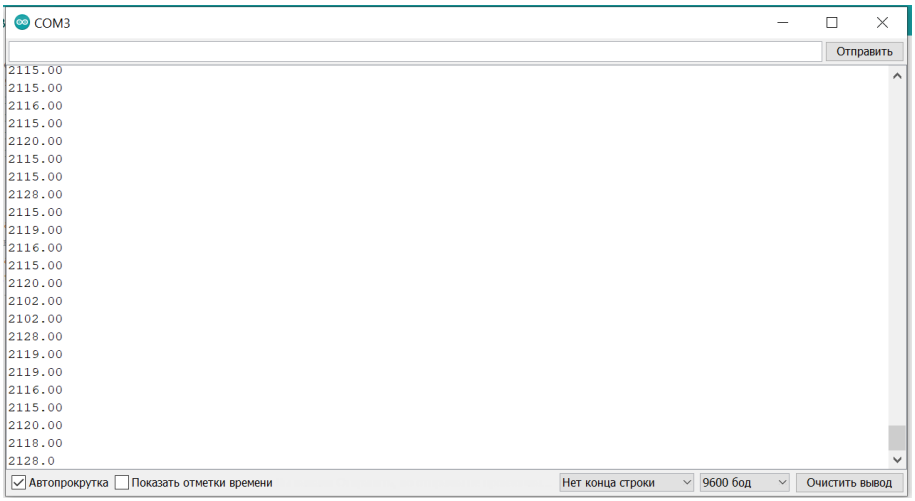


Рисунок 2.4 – Монитор COM порта

Далее изменяя скетч, заполните таблицу 2.1! Не забудьте переключить в меню периферии значение микрофона на «вкл» при надобности.

Подсказки:

– Для того чтобы снять данные сразу с 3 фоторезисторов для их усреднения, необходимо создать 3 переменные, которые будут хранить данные значения.

– Записывайте усредненное значение с 3 фоторезисторов в отдельную переменную – это понадобится в следующем задании.

После того как таблица будет заполнена, покажите данные преподавателю.

2.2 Создание системы автоматического освещения с подстройкой чувствительности

Используя скетч, в котором были получены данные с 3 фоторезисторов, можно создать систему автоматического освещения.

Принцип работы данной системы очень прост:

1. Получаем данные с фоторезисторов;

2. Подаем ШИМ сигнала на светодиод, который будет завесить от значения, полученного с фоторезисторов.

Для основы программы можно использовать данный код, фрагмент setup() остается прежним (из пункта 2.1):

```
void loop() {
    if (…) { //Начало обработки данных при включенном свитче
      /*Сбор данных с фоторезисторов*/
        float val1 = analogRead(…);
        /*----------------------------*/
        float avrg = { Расчет среднего значения } // Среденее значение с 3
датчиков

        uint8_t SignalToLed = ((… / 4096.0) * 255; // Расчёт сигнала для отправки

        Подача ШИМ сигнала на RGB светодиод
            Задержка;
    }
}
```

При загрузке скетча убедитесь, что в меню периферии включены: переменный резистор, фоторезисторы, светодиод RGB.

Как только скетч будет запущен выкрутите ручку переменного резистора на максимальное значение. При максимальном значении светодиод должен гореть ярче всего.

Теперь настройте с помощью переменного резистора работу системы таким образом, чтобы яркость светодиода зависела пропорционально от изменения падающего света на него.

Продемонстрируйте работу преподавателю.

Для получения дополнительного балла объясните: по какому принципу формируется значение, подаваемое на светодиод.

Домашнее задание:

Изучить принципы работы массивов на языке СИ и принцип их работы в Arduino IDE.

Что такое ASCII таблица и как буквы и цифры интерпретируются программой?

Работа № 7

«Датчик жестов. Датчик освещенности»

Тенденция такова, что происходит отказ от физических кнопок и переключателей в сторону сенсоров. Сейчас мало кого удивляют сенсорные панели и включение/отключение чего-либо простым касанием или жестом. Наличие сенсора позволяет создать более лаконичное устройство без утери его функционала. Один из способов реализации сенсорного управления – реализация на фотоэлементах. Так, например, на плате TUSUR IoT Board присутствуют, уже знакомые вам, 3 фоторезистора. И в этой работе предлагается реализовать сенсорное управление и управление жестами светом.

Цель работы: Научиться реализовывать сенсорное и жестовое управление на основе фоторезисторов.

Задачи:

1) Повторить принцип работы фоторезисторов;
2) Получить навыки по обработке данных с аналоговых датчиков;
3) Получить и обработать данные фоторезисторов;
4) Создать программу, выполняющую работу по сенсорному и жестовому управлению.

1. Теоретическая часть

1.1 Датчик уровня света

В прошлой работе вы уже сталкивались с фоторезисторами или же датчиками освещенности. Фоторезистор – датчик, изменяющий величину своего сопротивления в зависимости от интенсивности падающего света. Это полупроводниковый прибор, не имеющий p-n перехода и за счет этого обладающий одинаковой проводимостью для тока, протекающего в любом направлении. Явление изменения электрического сопротивления осуществляется за счет действия излучения и называется фоторезистивным эффектом.

Принцип работы фоторезистора заключается в том, что между двумя проводящими электродами (ножками) находится полупроводниковый элемент, изменяющий свое сопротивление в зависимости от количества падающего на него света. Неосвещенный фоторезистор может обладать сопротивлением до единиц МОм, а освещенный – единицами Ом. Таким образом, ток, протекающий через фоторезистор, увеличивается вместе с интенсивностью света. Внешний вид фоторезисторов производства СССР представлен на рисунке 1.1. Современные фоторезисторы представлены на рисунке 1.2.

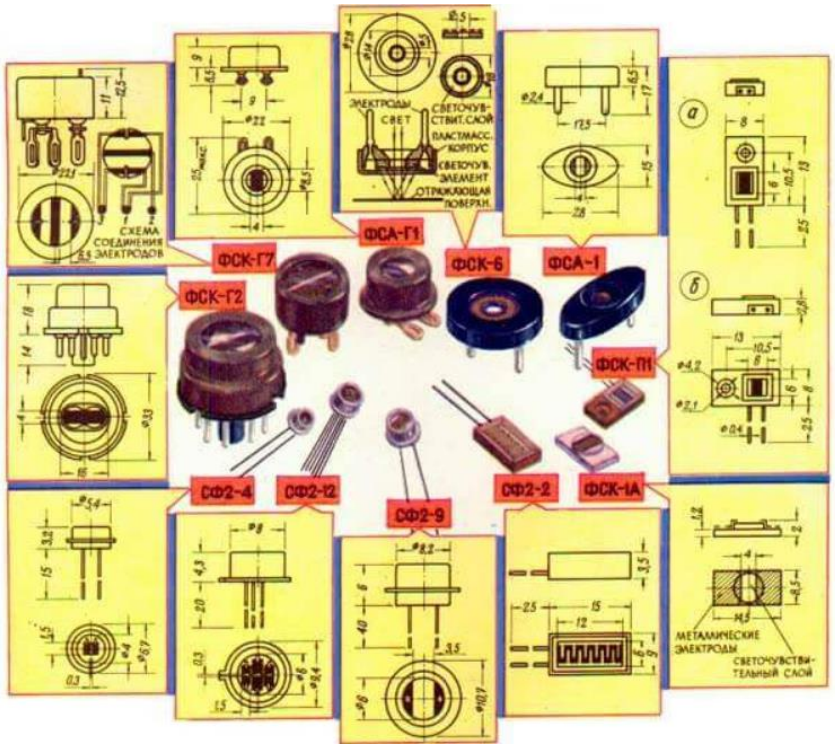


Рисунок 1.1 – Внешний вид фоторезисторов производства СССР

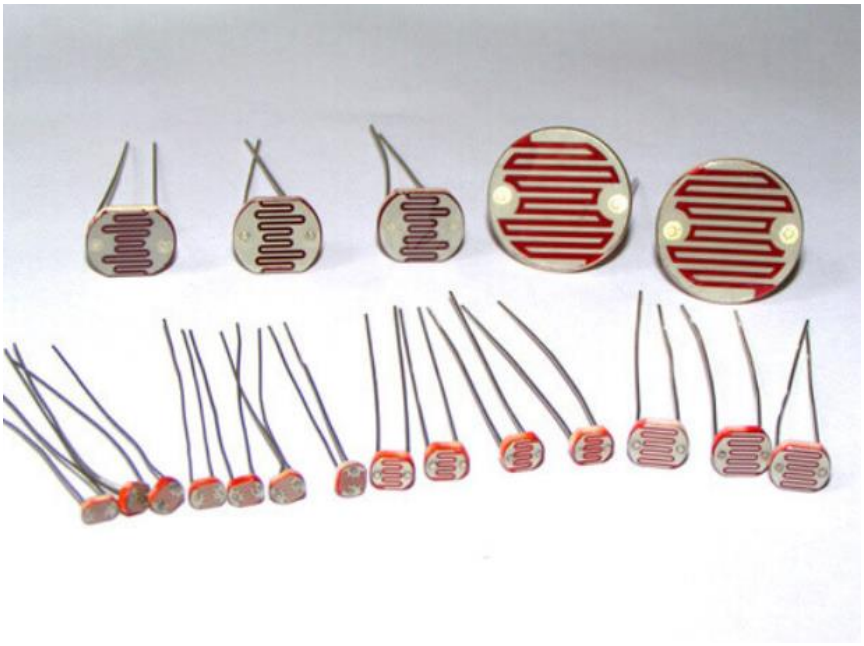


Рисунок 1.2 – Внешний вид современных фоторезисторов

Основные характеристики фоторезисторов: темновое сопротивление – сопротивление фоторезистора в темноте; интегральная фоточувствительность – изменение протекающего тока в зависимости от изменения светового потока. Ток, протекающий через фоторезистор, называется «фототок». Фототок – это разница между теневым током и током освещенного элемента, т.е. та часть, которая возникла из-за эффекта фотопроводимости.

На рисунке 1.3 представлен график изменения сопротивления резистора в зависимости от освещенности.

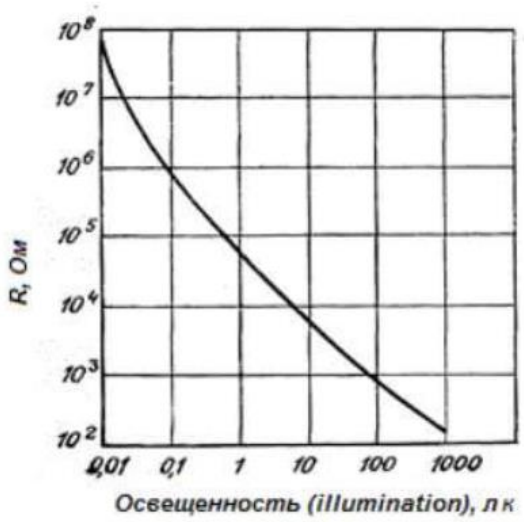


Рисунок 1.3 – График изменения сопротивления в зависимости от освещенности

Важно знать и помнить, что фоторезисторы обладают инерционностью, заключающейся в том, что изменение сопротивления происходит не мгновенно после облучения световым потоком, а с некоторой задержкой. Параметр, описывающий эту инерционность – граничная частота. Граничная частота – частота синусоидального сигнала, модулирующего световой поток через элемент, при которой чувствительность элемента снижается в корень из 2 раз (т.е. в 1,41 раза). Быстродействие фоторезисторов обычно лежит в пределах десятков миллисекунд (10^{-5} с.), что, в свою очередь, накладывает ограничение на область применение фоторезисторов и делает их не пригодными для применения в быстродействующих системах с минимальной задержкой реакции на изменение внешних факторов.

1.2 Области применения фоторезисторов

В прошлой работе вы проделали работу, используя фоторезисторы, и ознакомились с работой АЦП. Фоторезисторы применяются в сумеречных реле или фотореле – устройствах автоматического включения света по приходу сумерек. Вами же была реализована работа несколько более сложной логики, включающая в себя логику работы фотореле и подстройки интенсивности света в зависимости от освещенности окружения. Также фоторезисторы применяются в датчиках освещенности для измерение светового потока.

Еще одна область применения фоторезисторов – сигнализации. Обычно для этого используются фоторезисторы чувствительные к ультрафиолетовому излучению. Фоторезистор освещается излучателем, а в момент, когда луч пересекается, срабатывает сигнализация или исполнительный механизм. Так, например, устроены датчики охранной сигнализации, датчики пожарной сигнализации (датчики задымленности), турникеты в метро или других местах.

В промышленности фоторезисторы применяются как датчики наличия чего-либо. Так, например, контролируется целостность бумажной ленты на линии при ее изготовлении.

2. Практическая часть

2.1 Кнопка на основе датчика освещенности

С применением фоторезистора можно реализовать работу бесконтактной кнопки, т.е. кнопки, не требующей физического нажатия на нее. Например, применение бесконтактной кнопки может пригодиться там, где нельзя нарушать герметичность или целостность конструкции из стекла или другого светопроводящего материала, а также исключить касание поверхности. Например, из близкого к нам, зеркало со встроенной подсветкой под управлением бесконтактной кнопки, находящейся на передней панели. Это также позволяет сделать изделие более компактным, ведь нет необходимости встраивать механизм переключения. Согласитесь, никому не хочется пачкать зеркало и жертвовать его портативностью. На рисунке 2.1. представлен принцип внедрения кнопки, построенной на фоторезисторе, в зеркало.

Рисунок 2.1 – Бесконтактная кнопка на основе фоторезистора

В данной работе вам предлагается реализовать работу кнопки на плате TUSUR IoT Board для управления включением и выключением света.

Для этого нам понадобятся один фоторезистор и RGB светодиод, а также свитч для начала работы. Подготовьте плату и ваш скетч к началу работы. Переименуйте пины необходимой периферии так, как вам удобно, например, так:

```
#define FR1 PA1 // Подключение фоторезистора
#define SW1 PC15 // Подключение свитча
#define RGB_B PB1 // Подключение B
#define RGB_R PB9 // Подключение R
#define RGB_G PB8 // Подключение G
```

Далее необходимо создать переменную, которая будет выполнять роль счетчика касаний для того, чтобы одной кнопкой мы могли как включать, так и выключать подсветку. Также инициализируйте пины надлежащим образом.

```
int g = 0; // счетчик касаний
void setup() {
      Serial.begin();// Инициализация UART
      … (FR1, ); Инициализация фоторезистора на вход
      … (SW1, ); Инициализация свитча на вход

      … (RGB_R, ); //Инициализация R на выход
      … (RGB_G, ); //Инициализация G на выход
```

```
        … (RGB_B, ); //Инициализация B на выход
}
```

После чего необходимо описать работу программы. Для этого в `void loop()` необходимо считать информацию с фоторезистора так, как вы это делали в прошлой работе, по средством считывания аналогового сигнала с пина.

После этого необходимо выполнить ветвление для того, чтобы наша программа работала только тогда, когда этого хотим мы. Для этого мы и добавили свитч в работу. Используйте условный оператор полной развилки для выполнения кода при условии включенного и выключенного свитча. В теле условия включенного свитча создайте еще одно условие, которое будет отвечать за пороговое нажатие на кнопку. Чем выше порог, тем более чувствительной будет ваша бесконтактная кнопка. На данном этапе у вас должен получаться код, похожий на этот:

```
void loop() {
       float val1 = … (…);// Считывание данных с фоторезистора
       if (… (…)) { //Условие включенного свитча
             if (…  < 500) {// Если уровень ниже 500
```

Далее необходимо описать логику, выполняемую при достижении данных условий. Так как мы выполняем бесконтактную кнопку включения и выключения света, то, соответственно, при нажатии на кнопку, нам необходимо подать высокий уровень на RGB светодиод и выполнить отсчет касания. Также необходимо описать логику работы программы, если счетчик насчитает два касания (значит, мы собираемся выключить подсветку), а также необходимо обнулить счетчик. Примерно так выглядит код, реализующий такую логику работы:

```
       if (… (…)) {
             if (… < 500) {

                    … (…, HIGH); // Включаем R
                    … (…, HIGH); // Включаем G
                    … (…, HIGH); // Включаем B

                    g ++; // Инкремент счетчика касаний
                    delay(…); // Задержка
                    Serial.println(g); // Выводим в монитор порта значение счетчика
                    if (…) { // Если счетчик равен 2
                    … (…, HIGH); // Включаем R
                    … (…, HIGH); // Включаем G
                    … (…, HIGH); // Включаем B
                          g = …; // Обнуляем счетчик
                    }
             }
       }
```

Далее необходимо реализовать работу программы при выключенном свитче. Обычно, при отключении свитча должно происходить выключение всех огней, и управление платой становится невозможным. Таким образом, реализуйте эту часть кода и загрузите его на плату. Подумайте, как можно модернизировать код и сделать его более лаконичным, не теряя функциональности.

Основа кода программы представлена ниже:

```
#define FR1 PA1
#define SW1 PC15
#define RGB_B PB1
#define RGB_R PB9
```

```
#define RGB_G PB8

int g = 0;
void setup() {
	Serial.begin(9600);
	pinMode(FR1, …);
	pinMode(SW1, …);

	pinMode(RGB_R, …);
	pinMode(RGB_G, …);
	pinMode(RGB_B, …);
}
void loop() {
	float val1 = analogRead(FR1);
	if (digitalRead(SW1)) {
		if (val1 < 500) {

			digitalWrite(RGB_R, …);
			digitalWrite(RGB_G, …);
			digitalWrite(RGB_B, …);

			g = g + 1;
			delay(250);
			Serial.println(g);
			if (g == 2) {
				digitalWrite(RGB_R, …);
				digitalWrite(RGB_G, …);
				digitalWrite(RGB_B, …);
				g = 0;
			}
		}
	}
	else {
		digitalWrite(RGB_R, LOW);
		digitalWrite(RGB_G, LOW);
		digitalWrite(RGB_B, LOW);
	}
}
```

Продемонстрируйте работу преподавателю.

2.2 Управление жестами

Для реализации управления жестами нам необходимо использовать уже 2 фоторезистора. Используя уже существующий код, добавьте в него второй фоторезистор и примените к условным операторам следующие условия: при пересечении сначала первого фоторезистора, а затем второго (движение слева направо), необходимо включить свет; при пересечении сначала второго, а затем первого фоторезистора (движение справа налево) – выключить. Конечный код программы представлен ниже:

```
#define FR1 PA1 // Подключение первого фоторезистора
#define FR2 PA2 // Подключение второго фоторезистора
#define SW1 PC15 // Подключение свитча
#define RGB_B PB1 // Подключение B
#define RGB_R PB9 // Подключение R
#define RGB_G PB8 // подключение G

void setup() {
  Serial.begin(9600); // Инициализация UART
  pinMode(FR1, INPUT); // Настройка ФР1 на вход
  pinMode(FR2, INPUT); // Настройка ФР2 на вход
  pinMode(SW1, INPUT); // Настройка свитча на вход
```

```
  pinMode(RGB_R, OUTPUT); // Настройка R на вход
  pinMode(RGB_G, OUTPUT); // Настройка G на вход
  pinMode(RGB_B, OUTPUT); // Настройка B на вход
}
void loop() {
  float val1 = analogRead(FR1); // Считывание данных с ФР1
  float val2 = analogRead(FR2); // Считывание данных с ФР2

  if (digitalRead(SW1)) { // Если свитч включен

    if (val1 < 1000&&val2>1300) { // Проверяем условие жеста слева направао
      digitalWrite(RGB_R, HIGH); // Включаем свет
      digitalWrite(RGB_G, HIGH); //
      digitalWrite(RGB_B, HIGH); //
      delay(500); // Задержка для корректности жеста
    }
    if (val1 > 1300&&val2<1000) { // Проверяем условие жеста справа налево
      digitalWrite(RGB_R, LOW); // Выключаем свет
      digitalWrite(RGB_G, LOW); //
      digitalWrite(RGB_B, LOW); //
      delay(500); // Задержка для корректности жеста
    }
  }
  else { // Если свитч выключен
      digitalWrite(RGB_R, LOW); // Выключаем свет
      digitalWrite(RGB_G, LOW); //
      digitalWrite(RGB_B, LOW); //
  }
}
```

Выполните работу и продемонстрируйте ее преподавателю.

Дополнительное задание:

Дополнительное задание заключается в том, чтобы реализовать в коде кнопку, как в первой части работы, но при этом освещение должно загораться и тухнуть плавно, а не скачкообразно.

Работа № 8

«Работа с микрофоном»

Микрофон – устройство, преобразующее звуковые волны в электрический сигнал. Современные микрофоны способны сразу выдавать цифровой сигнал, но чаще высего они аналоговые. Данная работа посвящена работе с микрофоном. Думаю, многие из вас встречались или видели в кино управление светом через хлопки. В это работе мы реализуем такую систему включения света.

Цель работы: Научиться обрабатывать данные с микрофона и реализовать механизмы управления.

Задачи:

1) Обработать данные с микрофона;
2) Реализовать управление светом хлопками.

1. Теоретический материал

1.1 Принцип работы микрофона

Микрофон – устройство, преобразующее звуковые волны в электрический сигнал.

Принцип работы микрофона заключается в том, что давление звуковых волн действует на тонкую мембрану микрофона, колебания которой возбуждают электрические импульс. В зависимости от типа микрофона используется явление электромагнитной индукции, изменение емкости конденсаторов или пьезоэлектрический эффект. Типы микрофонов отличаются усилением, диапазоном частот и пр. Однако, сейчас нам важно лишь то, что на выходе микрофона получается сигнал, который необходимо обработать.

Хлопки характеризуются короткими пикообразными скачками. Если подключиться к микрофону на плате TUSUR IoT Board и считать его выходной сигнал, то мы увидим следующую картину (рисунок 1.1).

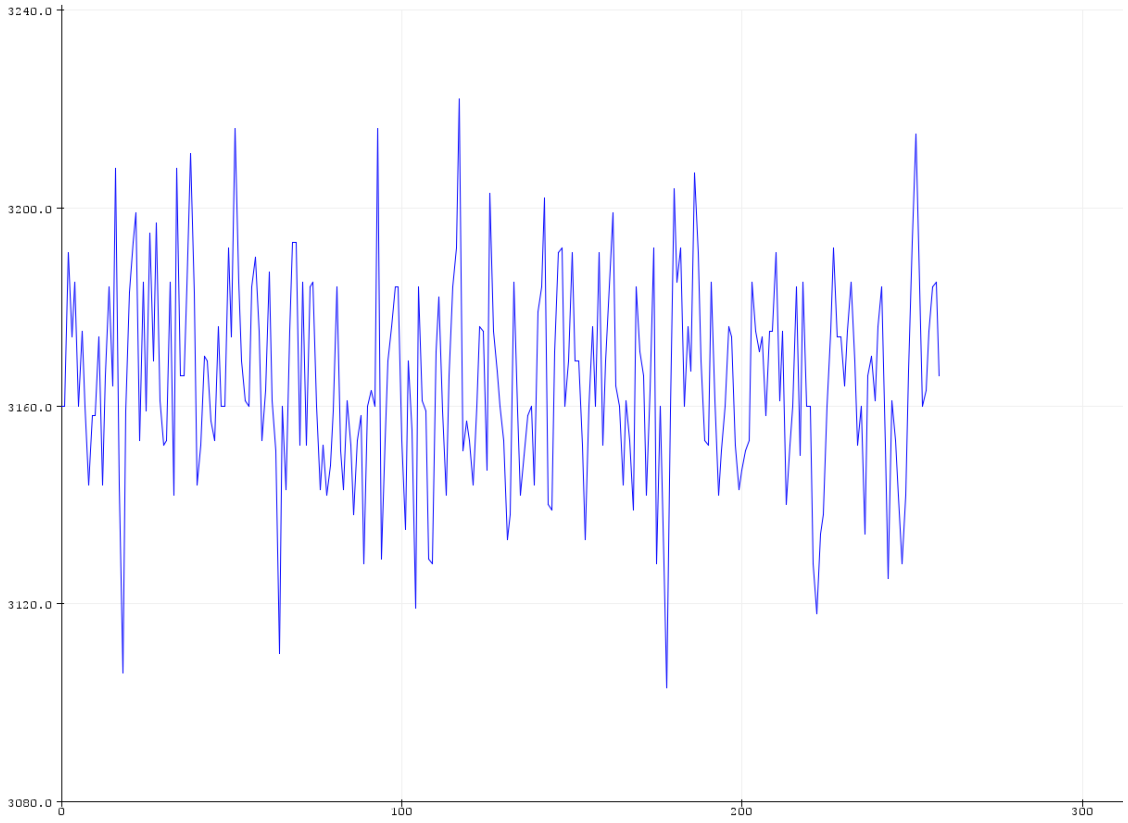


Рисунок 1.1 – Выход с микрофона TUSUR IoT Board

Однако, когда мы начинаем, например, издавать хлопающие звуки, выход микрофона изменит свой вид и будет примерно похож на зависимость с рисунка 1.2.

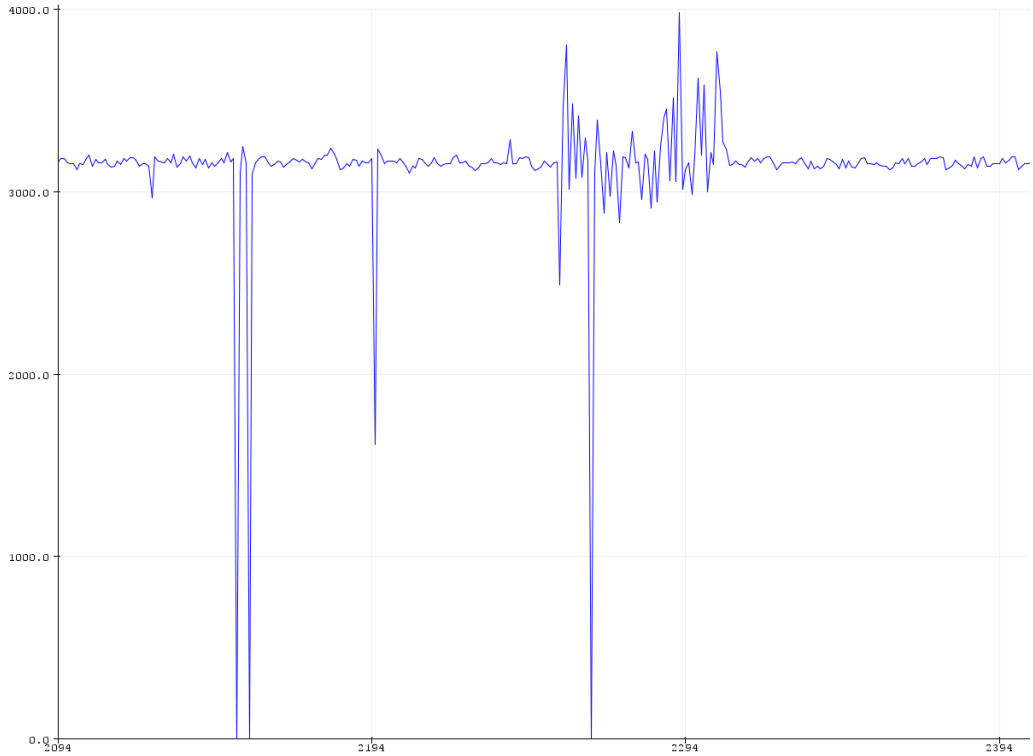


Рисунок 1.2 – Выход с микрофона в момент хлопающих» звуков

Как видно из рисунка 1.1, в момент покоя выход с микрофона обладает уровнем амплитуды 3160 с флуктуациями в +-100. В момент хлопков скачки амплитуд достигают значения от 200 до нескольких тысяч. Это и позволит в будущем создать программу, реагирующую на эти скачки.

Как мы помним из предыдущих работ, микроконтроллер не воспринимает аналоговый сигнал как таковой. Поэтому работа с микрофоном подразумевает и работу с АЦП. Путь, который сигнал проходит от хлопка до исполнения кода, будет иметь следующий вид: Хлопок, генерирующий звуковые волны; микрофон, преобразующий звуковые волны в электрический аналоговый сигнал; аналоговый сигнал преобразуется в АЦП; передача оцифрованного сигнала в микроконтроллер; принятие решения, основываясь на полученных значениях; исполнение нужной части кода.

2. Практическая часть

2.1 Применение микрофона

В данной работе будет рассмотрен принцип выключателя света по хлопкам, а также реализована система управления светом.

Для начала необходимо настроить периферию платы под текущую работу. Для этого необходимо перейти в раздел меню «Изменение периферии» и перевести «RGB-светодиод» и «Микрофон» во включенное состояние. Далее, переходим к написанию кода программы для считывания данных с микрофона. Необходимо создать переменную, в которую будет записываться выход с микрофона, а также инициализировать пин микрофона на вход и задать скорость работы интерфейса UART для дальнейшего вывода значений с микрофона. Пример написания такого кода представлен ниже:

```
#define mic PA3
int Count = 0; //Переменная для хранения значений с датчика звука
void setup() {
	pinMode(mic, INPUT); //Переводим пин 3 в  режим выхода
	Serial.begin(9600); // Задаем скорость работы UART
}
```

Далее необходимо считать данные с микрофона и поместить их в созданную ранее переменную. Данное действие выполняется уже в `void loop()`, так как считывание данных с микрофона должно происходить постоянно. Также необходимо отобразить полученные значения, для этого воспользуйтесь командой `Serial.println(Count)`. Обязательно после команды вывода информации в плоттер поставьте задержку! Таким образом, вы должны получить код, которым нужно прошить микроконтроллер для считывания данных с микрофона.

Ваш код может выглядеть иначе, но должен выполнять следующий функционал:

```
#define mic PA3
int Count = 0; //Переменная для хранения значений с датчика звука
void setup() {
	pinMode(mic, INPUT); //Переводим пин 3 в  режим выхода
	Serial.begin(9600);  // Задаем скорость работы UART
}
void loop() {
	Count = analogRead(mic); //читаем значения
	Serial.println(Count);
	delay(5);
}
```

Загрузите код в микроконтроллер и по выполнению загрузки откройте «Плоттер по последовательному соединению» через вкладку «Инструменты» или воспользуйтесь комбинацией клавиш «Ctrl+Shift+L». Откроется окно плоттера где вы должны наблюдать примерно следующую картину, представленную на рисунке 2.1. Важно выбрать соответствующую скорость работы плоттера. Она должна совпадать с скоростью, указанной в коде! Как com-порт, так и уровень сигнала у всех может отличаться!

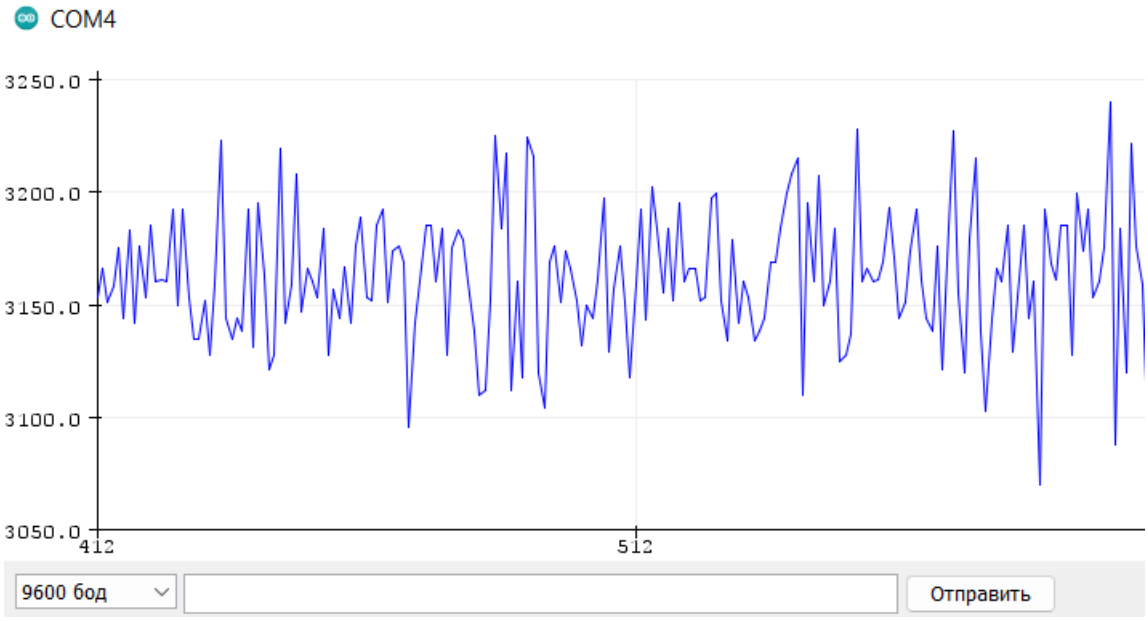


Рисунок 2.1 – Выход с микрофона

Теперь начните издавать хлопки и пронаблюдайте поведение выхода с микрофона. Присутствуют ли пики? Какой амплитуды? Посмотрите схож ли характер поведения выхода и рисунком 1.2.

Для удобства можно ввести константу, которая будет равняться среднему значению уровня сигнала на выходе микрофона в покое, а затем от этой константы отнимать считанный сигнал с микрофона и взять это значение в модуль. Введите константу `int const Etalon = 3200;` где `const` – обозначение константы, значение которой невозможно будет изменить, а 3200, это средний уровень сигнала с микрофона (у вас он может отличаться!). Далее введите переменную, которая будет хранить в себе значения разности амплитуд, например, `int res = 0`, таким образом создалась пустая переменная, а далее в `void loop()` добавьте разность «эталонного» значения и принимаемого сигнала с микрофона, а также возьмите модуль. Такая команда выглядит следующим образом: `res = abs(Etalon - Count)`. Выведите полученное значение в плоттер. Итоговый код такой программы представлен ниже:

```
#define mic PA3
int Count = 0; //Переменная для хранения значений с датчика звука
int const Etalon = 3200;
int res = 0;
void setup() {
      pinMode(mic, INPUT); //Переводим пин 3 в  режим выхода
      Serial.begin(9600);  // Задаем скорость работы UART
}
void loop() {
      Count = analogRead(mic); //читаем значения
      res = abs(Etalon - Count);
      Serial.println(res);
      delay(5);
```

```
}
```

Результат выполнения программы, который выводится в плоттер, должен иметь следующий вид:

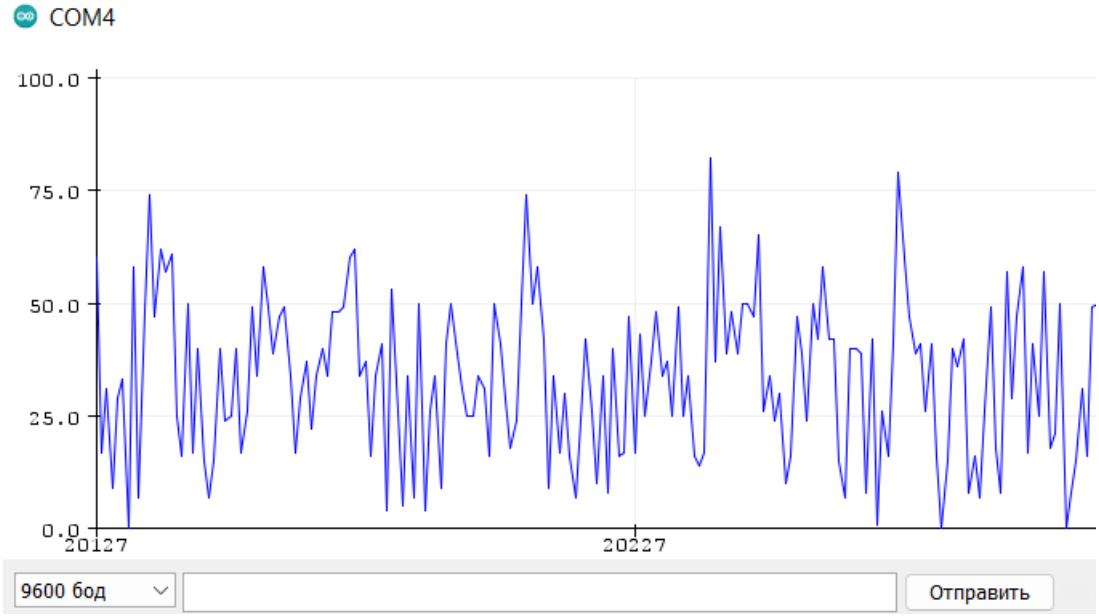


Рисунок 2.2 – Флуктуации сигнала в режиме покоя

Выполните несколько хлопков и оцените диапазон разлета амплитуды вашего сигнала, запишите или запомните его для дальнейшего применения.

2.2 Реализация «хлопкового» выключателя

Начните реализовывать логику работы программы. Так как мы хотим сделать выключатель, который срабатывал тогда и только тогда, когда мы этого хотим, то необходимо выполнить логику через два быстрых хлопка, иначе любой стук активировал бы переключатель, и свет самопроизвольно загорался и тух. Поэтому необходимо реализовать проверку наличия хлопка, и в течении некоторого времени после первого хлопка ожидать второй. Если второго хлопка не последовало, то состояние света изменяться не должно.

Для начала работы введем еще одну переменную, которая будет хранить в себе состояние переключателя. Подумайте, какой тип переменной лучше всего создать. Добавьте инициализацию пинов RGB светодиода. Далее воспользуйтесь условным оператором ветвления с условием, в котором укажите диапазон значений хлопка, который вы записали: `if (res > ?? && res < ??)` (укажите ваш диапазон). Далее необходимо выставить задержку db 200 мс. для ожидания второго хлопка. Затем откройте цикл `for`, который будет искать второй хлопок, и если его обнаружит, то изменит состояние выключателя. Цикл `for` должен содержать данные условия: `for (int t = 0; t <= 500; t++)`. До 500 – чтобы точно захватить оба хлопка. Подумайте, как можно это сделать, и реализуйте данную работу.

Если у вас все получилось, то подумайте, как можно улучшить данный код.

Примерный финальный код программы представлен ниже:

```
#define mic …
#define RGB_B …
#define RGB_R …
#define RGB_G …

int Count = 0; //Переменная для хранения значений с датчика звука
```

```
int const Etalon = 3200; //Переменная для хранения порогового уровня
int res = 0; //Переменная, хранящая в себе разность порогового и считанного
сигнала
bool Relay = 0; //Переменная для хранения состояния выключателя

void setup() {
      pinMode(RGB_R, …); //переводим пин R в  режим выхода
      pinMode(RGB_G, …); //переводим пин G в  режим выхода
      pinMode(RGB_B, …); //переводим пин B в  режим выхода
      pinMode(mic, …); //переводим пин mic в  режим выхода
      Serial.begin(…);
}

void loop() {
      Count = …Read(…); //читаем значения
      res = abs(… - …); //Расчет разности

      if (res > 200 && res < 1000)
      {
            delay(…); //ожидаем 250 милисекунд для повторного хлопка

            for (int t = 0; t <= 500; t++)
            {
                  delay(1);
                  Count = …Read(…); //считываем значение
                  res = abs(Etalon - Count); //Расчет разности

                  if (res > 200 && res < 1000)
                  {
                        Relay = !Relay; //Изменение состояния выключателя
                        break; //Выходим из цикла после второго хопка
                        delay(…); //Пауза
                  }
            }
      }
      Serial.println(…); //Вывод данных
      delay(5);
      …Write(RGB_R, …); //Подача состояния переключателя на R
      …Write(RGB_G, …); //Подача состояния переключателя на G
      …Write(RGB_B, …); //Подача состояния переключателя на B
}
```

Работа № 9

«Изучение интерфейса UART»

Все интерфейсы передачи данных служат для одной цели – передачи данных. Данными с микроконтроллером могут обмениваться датчики, исполнительные устройства, дисплеи, индикаторы, другие контроллеры и др. Передача данных бывает как двусторонняя, так и односторонняя. Под односторонней связью подразумевается, например, передача информации датчиком температуры на микроконтроллер без ожидания обратного ответа.

Универсальный асинхронный приемопередатчик (УАПП, англ. Universal Asynchronous Receiver-Transmitter, UART) - узел вычислительных устройств, предназначенный для организации связи с другими цифровыми устройствами.

Протокол UART является старейшим и самым простым физическим протоколом передачи данных. Наиболее известным из семейства UART является протокол RS-232 – так называемый COM-порт.

Цель работы: изучить способы передачи информации от контроллера к контроллеру/компьютеру

Задачи:

1) Изучить понятие интерфейса;
2) Изучить принцип работы UART – последовательного интерфейса;
3) Получить навыки по работе с монитором порта в Arduino IDE.

1. Теоретическая часть

1.1 Интерфейсы

Интерфейс – это некоторое аппаратное или программное решение, которое позволяет обмениваться данными между разными типами ЭВМ. К примеру: контроллер – контроллер, контроллер – компьютер и т.д.

Существует большое количество способов, с помощью которых мы можем передавать большое множество данных, и есть большое количество нюансов, которые можно поменять. Например, частота передаваемого сигнала, форма, количество проводов.

Но для того, чтобы производство было гораздо комфортнее и продуктивнее, в мире используются стандарты (набор правил), по которым осуществляется передача данных между контроллерами – интерфейсы.

В данном курсе мы рассмотрим основные типы интерфейсов, которые используются при программировании встраиваемых систем, а именно:

- UART;
- I2C;
- SPI.

1.2 Последовательный интерфейс UART

Последовательный UART интерфейс – один из первых интерфейсов, с которым знакомятся программисты микроконтроллеров, так как часто возникает такая ситуация, когда необходимо вывести информацию с микроконтроллера на компьютер для анализа и обработки данных.

Вспомните прошлую работу, где данные с АЦП мы передавали в последовательный порт, чтобы вывести данные в плоттер.

UART интерфейс является асинхронным интерфейсом. Это означает, что для передачи данных не используется сигнал синхронизации.

Для передачи данных по последовательному интерфейсу достаточно 1 провода, по которому передаются сигналы следующего характера, представленного на рисунке 1.1.

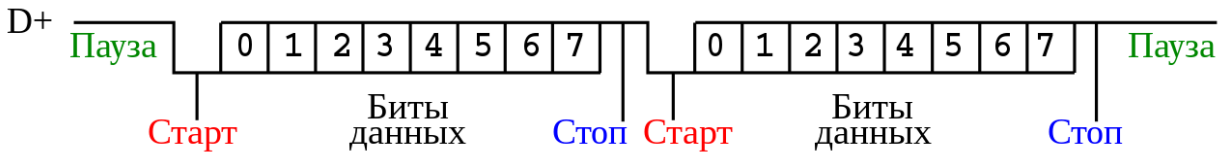


Рисунок 1.1 – Представление пакета данных для передачи по последовательному интерфейсу

Как мы можем видеть из рисунка, в момент времени, когда ничего не передается, на линии UART'а сохраняется состояние логический единицы.

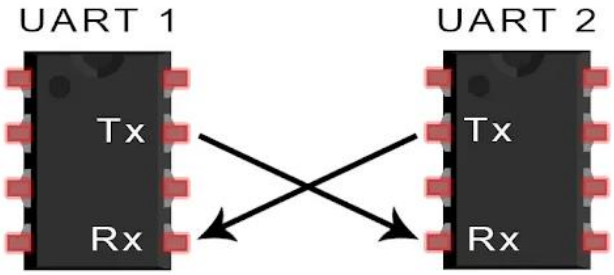
Перед началом передачи идёт старт бит (логический ноль), который говорит о том, что далее будут передаваться 8 бит данных.

После того, как 8 бит данных будут переданы, отправляется стоп бит (логическая единица), и линия становится доступной для следующих данных.

Период времени на один бит зависит от скорости передачи информации. Для данного интерфейса можно выбрать ряд следующих скоростей: 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 бит/с.

Чем больше скорость передачи, тем меньше период (так как за меньшее количество времени нужно передать большее количество информации), что в свою очередь повышает вероятность ошибки.

UART интерфейс может работать как в одном направлении (либо прием, либо передача), так и в двух направления сразу, только являться как приемником, так и передатчиком, но для этого понадобится использовать 2 провода (рисунок 1.2):



Двунаправленная передача данных интерфейса UART

Рисунок 1.2 – Подключение последовательного интерфейса между двумя устройствами для приема/передачи

На плате TUSUR_IOT_BOARD связь между контроллерами ESP-32(отвечает за экран и меню) и STM (контроллер для программирования) устанавливается по UART, при использовании меню «Примеры».

При входе в данное меню ESP отправляет запрос на плату STM, который способен разобрать скетч с примерами, и который способен отправить ответ, благодаря которому ESP понимает, что установлен верный скетч, и с ним можно работать. И далее при выборе примера на ESP, на STM отправляется сигнал, который запускает определенную инструкцию для воспроизведения примера.

2. Практическая часть

2.1 Изучение работы UART интерфейса

Для того чтобы увидеть форму сигнала при передаче информации с использованием последовательного интерфейса, необходимо воспользоваться логическим анализатором.

Логический анализатор – это измерительное устройство, которое позволяет анализировать сигнал, который поступает на его вход по его логическим значениям.

Логический анализатор присутствует на плате TUSUR_IOT_BOARD. Для доступа к нему необходимо воспользоваться меню и выбрать пункт «Логический анализатор» (рисунок 2.1), выбрать необходимый вид протокола для передачи данных (UART, I2C, SPI), а также выбрать необходимую скорость передачи данных.

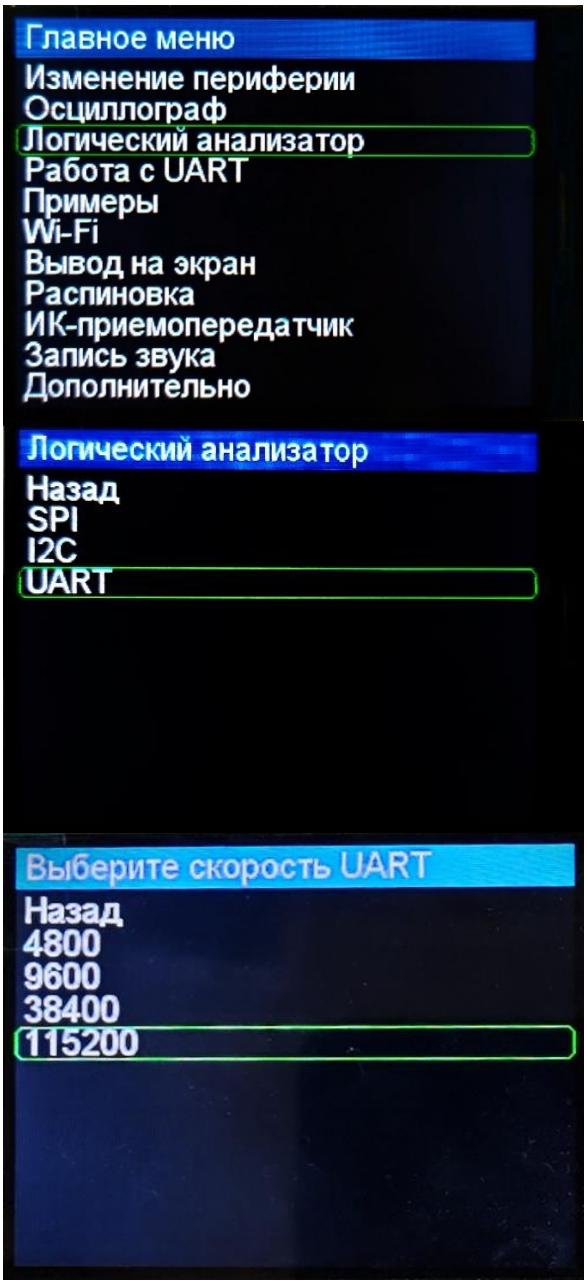


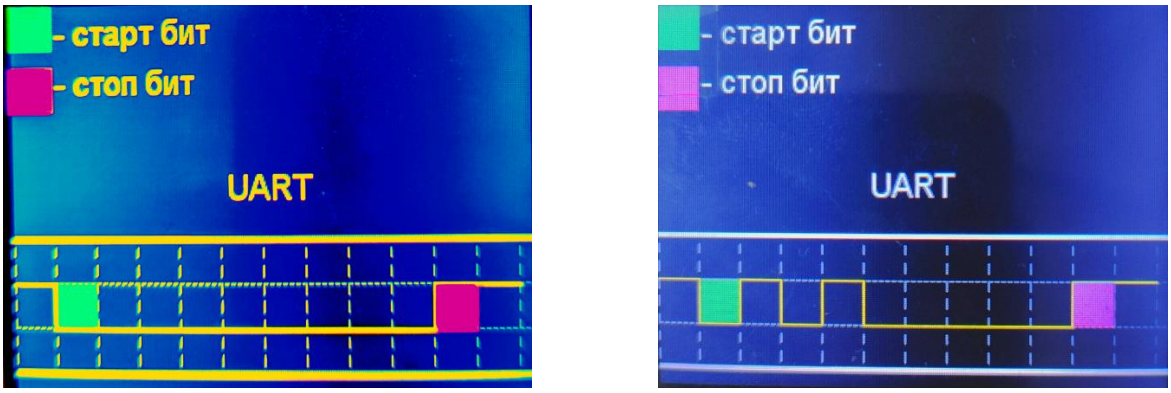
Рисунок 2.1 – Главное меню платы и логического анализатора

Для того чтобы воспользоваться последовательным интерфейсом для передачи данных нужно воспользоваться следующим кодом:

```
void setup() {
    Serial.begin(9600); // Инициализация передачи по последовательному интерфейсу
со скоростью 115200 байт\с
}

uint8_t i = 0; //Создание глобальной переменной i
void loop()
{
    //Serial.write(i); // Отправка значения, которое содержит переменная i по UART
    for (…) { // Цикл, который меняет значение i от 0 до 255
        delay(…); // Ожидание в течении 3 секунд (3000 миллисекунд)
        Serial.write(…);// Отправка значения, которое содержит переменная i по
UART
    }
}
```

Допишите данный код и загрузите на плату. Вы уже использовали данный код в работе №5 «Обработка данных с кнопок и энкодеров», поэтому допишите его или возьмите с той работы и вспомните структуру посылки в UART.

После того, как скетч будет запущен, попробуйте воспользоваться логическим анализатором, выбрав его для протокола UART. Посмотрите форму сигнала и сравните полученные результаты с теоретическим материалом. В случае правильной работы кода вы будете наблюдать следующую картину, при i=0 и i=5:



i=0 i=5

Рисунок 2.2 – Показания анализатора спектра при i=0 и i=5

Попробуйте передать свое собственное значение, для этого:

Закомментируйте цикл *for* с его фрагментом программы;

Снимите комментарий с *Serial.write(i);* над циклом и добавьте задержку в 3 секунды;

В качестве числа можете использовать любое число от 0 до 255 (к примеру, номер вашей группы) и посмотрите его представление в логическом анализаторе

Докажите преподавателю, что передано именно то самое число, которые было введено вами.

2.2 Написание программы, которая имитирует работу кодового замка

Перед написанием программы определим, какие компоненты необходимы для того, чтобы данная программа заработала:

- Диоды (красный, зеленый, синий) для того, чтобы отображать статус замка;

• Вентилятор, имитирующий работу замка;

• Последовательный порт, с помощью которого будет производиться взаимодействие с контроллером;

• Глобальная переменные (массивы и переменные для счетчика), которые будут хранить в себе значение верного пароля и вводимого пароля на разных этапах его жизни, а также для счетчика неверных попыток.

Для того чтобы инициализировать данные компоненты нам понадобится реализовать следующий код:

```
#define LED1 PB9 // красный светодиод
#define LED2 PB8 // зеленый светодиод
#define LED3 PB1 // синий светодиод
#define Cool PA1 // реле на замок
#define NUM_KEYS 5 // количество знаков в пароле

char myarraw[NUM_KEYS] = { '1','2','3','4','5' }; // массив с верным паролем
char button_pressed1[NUM_KEYS]; //Переменная хранящая считанный пароль
char button_pressed; // Переменная для считывания пароля из монитора порта

int k = 0; // Переменная для счетчика неправильных попыток

void setup() {
  pinMode(LED1, …); // красный светодиод
  pinMode(LED2, …); // зеленый светодиод
  pinMode(LED3, …); // зеленый светодиод
  pinMode(Cool, …); // реле управления замком
  Serial.begin(…);
}
```

Serial.begin(115200); – функция, которая принимает аргумент 115200 (скорость последовательного интерфейса) и запускает работу последовательного интерфейса.

Для того чтобы выводить данные в последовательный интерфейс и видеть их с компьютера можно воспользоваться двумя функциями:

Serial.println() – выводит значение и в конце добавляет возврат коретки (перенос строки);

Serial.print() – выводит значение.

Далее, воспользовавшись данными функциями, выведите сообщение «Для доступа в систему необходимо ввести 5-и значный пароль».

Для этого вставьте строку «Для доступа в систему необходимо ввести 5-и значный пароль» в круглые скобки в функции.

Запустите программу и с помощью монитора порта (Ctrl + Shift + M) убедитесь в том, что вы видите приветственное сообщение.

ВАЖНО!

Необходимо поставить скорость монитора порта в соответствии с той, что указана при инициализации последовательного интерфейса. Как это сделать описано в прошлой работе!

Далее реализуем процесс, который запустится при первом включении платы и попросит ввести пароль для доступа к ней.

Для этого воспользуйтесь следующим программным кодом.

```
void setup() {
  pinMode(LED1, …); // красный светодиод
  pinMode(LED2, …); // зеленый светодиод
  pinMode(LED3, …); // зеленый светодиод
  pinMode(Cool, …); // реле управления замком
  Serial.begin(…);
  Serial.println("Для доступа в систему необходимо ввести 5-и значный пароль.");
```

```
  delay(1000);
}
void loop() {
    Serial.begin(…);
    digitalWrite(LED3, …); // Включаем синий светодиод
    while (!(Serial.available() > NUM_KEYS)) {} // Ожидаем NUM_KEYS байт ввода

    for (int i = 0; i < NUM_KEYS; i++) { // Цикл выбирающий по 1 байту

        button_pressed = Serial.read(); // Выбираем по 1 байту
        button_pressed1[i] = button_pressed; // Формируем массив пароля
    }
    Serial.println(button_pressed1); // Выводим введенный пароль
}
```

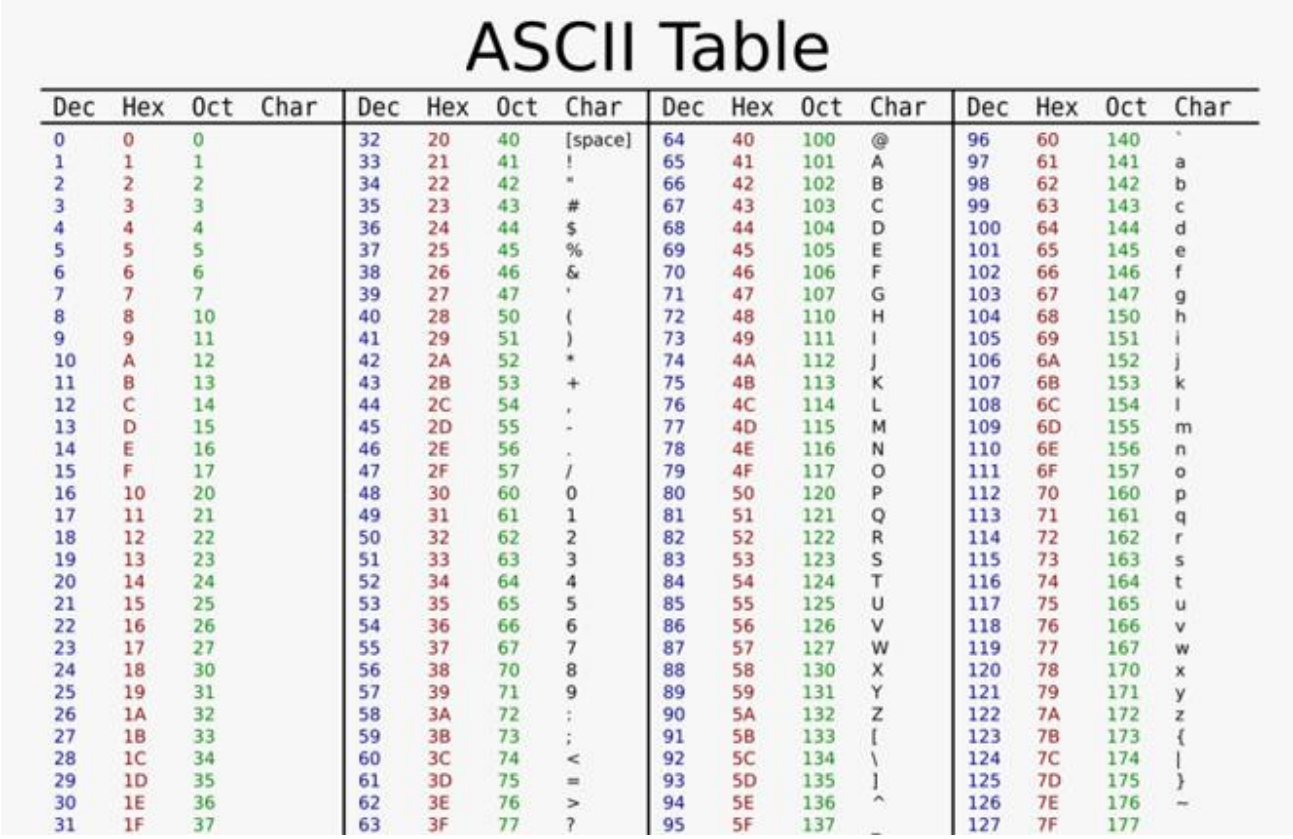
Важно! В данной работе использовался несколько нестандартный подход к включению UARTa. Как вы могли заметить, его включение происходит как в `void setup()`, так и в `void loop()`. Это очень важно, поскольку будет необходимо неоднократно включать интерфейс UART. Позже вы поймете зачем.

Здесь используется функция Serial.available() – данная функция хранит в себе значение количества байт, которое хранится в буфере последовательного порта. Так как наш пароль состоит из 5 цифр, следовательно, нам необходимо получить значение равное 5 байтам.

Следовательно, сделаем цикл, который будет «бесконечно крутиться» до тех пор, пока функция не вернет значение больше 4, которое будет означать, что в буфере хранится значение, которое занимает 5 байт (5 цифр), а это как раз то, что нам необходимо.

Далее нам необходимо считать из буфера значение, которые мы ввели. Для этого необходимо воспользоваться функцией Serial.read(). Данная функция возвращает значение, которое пришло в последовательный порт в виде типа данных byte, то есть она возвращает каждый байт информации. Для того чтобы считать все 5 байт, нам нужно повторить эту операцию 5 раз. Для этого воспользуемся циклом.

ВАЖНО! Функция возвращает свои значения согласно ASCII таблице (рисунок 2.3):



ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Рисунок 2.3 – ASCII таблица

Если ввести 0, то функция Serial.read() вернет значение 30 (согласно таблице), которое соответствует 48 в десятичном представлении.

Для того чтобы получить именно значение 0, которое мы запишем в функцию, воспользуемся следующей хитростью. Допустим, мы получили значение 1, которое соответствует 49. Отнимем от 49 значение 0 в символьном представлении (‘0’). Получится 49 – 48 = 1. То, что нам и нужно. Так работает с любыми введенными числами. Проверьте!

Однако, в нашем коде мы воспользовались типом переменной `char`, что позволяет миновать таблицу ASCII, так как в таком случае нет необходимости интерпретировать ввод, а каждый символ воспринимается, как символ. И дальше мы работаем уже с массивом `char`. Т.е. если вы ввели «4», то в коде «4» - не число, а символ. Это значительно упрощает работу с такой переменной в нашей работе.

Для того чтобы отправить данные по последовательному UART интерфейсу нужно ввести данные в данном поле (рисунок 2.3) и нажать кнопку отправить (ENTER).

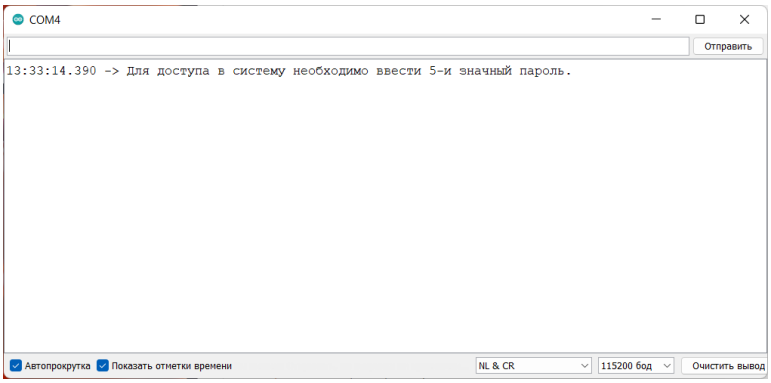


Рисунок 2.4 – Монитор порта

Загрузите программу и попробуйте ввести пароль и проверьте, что пароль, который вы ввели действительно, верно выводится на экран.

Покажите работу программы преподавателю.

Далее во фрагменте loop() реализуем процесс проверки пароля и отображения состояния проверки на RGB светодиоде и включении вентилятора, имитирующего работу замка. Для этого нам нужно написать функцию, выполняющую проверку двух массивов типа `char` на совпадение длин и содержимого. Для этого необходимо проверить длину каждого массива. Сделать это можно используя цикл `while`. Запись будет выглядеть так:

```
int i = 0; // Создаем глобальную переменную
while (str1[i] != '\0' && str2[i] != '\0') { // Открываем цикл сравнения двух массивов
```

Эта запись означает, что цикл работает до тех пор, пока не найдется конец строки. Т.е. пока не будет найдет символ `'\0'`, означающий конец строки. И именно его можно заметить, если передать информацию через монитор порта и вывести его на экран. Можно увидеть следующую картину:

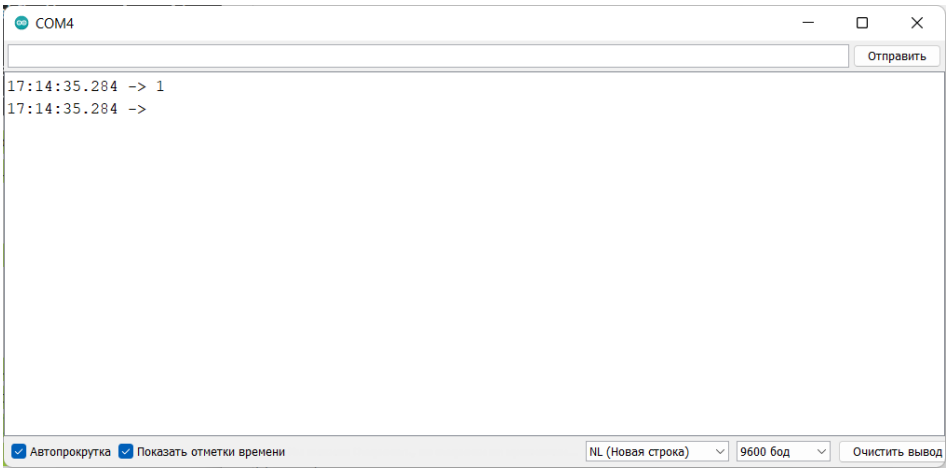


Рисунок 2.5 – Монитор порта с отображением конца символа

После того, как будет найден конец строки, необходимо будет сравнить их длины и выполнить поэлементное сравнение. Если длины и элементы в массивах совпали, то вернуть значение True, а если хоть что-то не совпало, то False. Функция будет выглядеть следующим образом:

```
bool strcheck(char* str1, char* str2) {
    int i = 0;
    while (...[...] != '\0' && ...[...] != '\0') { // Цикл выполняется до тех пор, пока
                                                   //не найдется хоть один конец строки

        if (...[...] != ...[...]) // Сравниваем каждый i элемент массивов друг с другом
            return …; // Если хоть один не совпал возвращаем False
        i++;
    }
    //Если мы вышли из цикла значит одну из строк мы перебрали до конца
    if (...[...] == '\0' && ...[...] == '\0') // Проверяем на соответствие i элемента
                                              //каждого массива элемента концу строки
        return 1; // Если совпало, возвращаем True
```

```
    return 0; // Не совпало, возвращаем False
}
```

Допишите код и переходите к дальнейшему выполнению работы.

После того, как была создана функция проверки подлинности кода, необходимо ее вызвать в `void loop()` после того, как вы сформировали массив введенных данных. Результат, возвращенный функцией, необходимо использовать для ветвления программы. Как вы помните, функция возвращает всего два значения True и False, True – если пароль совпал и False – если не совпал. Таким образом, воспользуйтесь оператором ветвления и для возвращенного значения True создайте следующую логику в его теле:

1. Обнуление счетчика k;
2. Вызов функции вывода информации в монитор порта с сообщением о том, что пароль введен корректно;
3. Синий светодиод - отключение;
4. Зеленый светодиод – включение;
5. Вентилятор – включение;
6. Задержка в 1с;
7. Вентилятор – отключение;
8. Цикл бесконечной истинности;

```
    if (... == ...) {// Условие выполняется при возврате True
        k = ...; // Обнуление счетчика
        Serial.println(); // Переход на новую строку
        Serial. ...(...); // Вывод сообщения о правильности пароля
        digitalWrite(...); // Выключаем синий светодиод
        ... (..., ...); // Включаем зеленый светодиод
        ... (Cool, ...); // Включаем мотор замка
        ... (...); // Задержка в 1 секунду
        ... (..., ...); // Выключаем мотор замка
        while (...); // Цикл бесконечной истинности
    }
```

Допишите данный код согласно описанию логике работы. Загрузите на плату и проверьте, работает ли ваша программа. Перед этим убедитесь, что в меню выбора периферии у вас включен RGB светодиод и вентилятор.

Следующим этапом необходимо реализовать полное ветвление условного оператора, в теле которого необходимо реализовать следующую логику:

1. Инкремент счетчика;
2. Условный оператор с условием достижения определенного количества попыток (значений счетчика):
   a. Вывод сообщения о том, что превышен лимит попыток;
   b. Синий светодиод – отключение;
   c. Зеленый светодиод – отключение;
   d. Красный светодиод – включение;
   e. Цикл бесконечной истинности;
3. Полное ветвление условного оператора, в теле которого:
   a. Вывод сообщения о неправильно введенном пароле;
   b. Синий светодиод – отключение;
   c. Зеленый светодиод – отключение;
   d. Красный светодиод – включение;
   e. `Serial.end();`;
   f. Задержку в 2 секунды.

```
    else { // Иначе
```

```
        k++; // Инкремент счетчика
        if (... == ...) { // Если счетчик равен "", то
            Serial.println("… "); // Выводим сообщение о запрете доступа
            ... (..., ...); // Выключаем синий светодиод
            ... (..., ...); // Выключаем зеленый светодиод
            ... (..., ...); // Включаем красный светодиод
            while (...); // Цикл бесконечной истинности
        }
        else Serial.println("..."); // Иначе выводим сообщение о повторе попытки
        ... (..., ...); // Выключаем синий светодиод
        ... (..., ...); // Выключаем зеленый светодиод
        ... (..., ...); // Включаем красный светодиод
        Serial.end(); // Очищаем буффер UARTa
        ...(...); // Задержка в 2 секунды
    }
```

Команда `Serial.end()` необходима для того, чтобы выключить UART и очистить буфер. Это нам нужно для того, чтобы удалить все ненужные байты информации, и чтобы при вводе пароля в следующий раз ваш пароль вновь был первым в очереди, а также не случилось ситуации, что вы ввели правильный пароль, но из-за того, что в UARTe в очереди уже стоял байт информации, оставшийся с прошлого раза. После команды `Serial.end()` необходимо вновь запустить UART. Именно для этого в самом начале `void loop()` вы и написали команду на включение UARTa.

Далее перед закрытием цикла `void loop()` необходимо прописать команды на отключение всех светодиодов RGB светодиода. Это необходимо для того, чтобы при условии полного выполнения цикла `void loop()` программа вернула плату в то состояние, в котором она была до выполнения `void loop()`.

```
    ... (..., ...); // Выключаем синий светодиод
    ... (..., ...); // Выключаем зеленый светодиод
    ... (..., ...); // Выключаем красный светодиод
}
```

Допишите все участки кода и объедините их в один. Прошейте плату. Результатом работы такой программы будет предоставление n количества попыток на введение пароля, что позволит обладателю кодового замка не переживать о блокировке двери при первом же неправильно введенном коде, а иметь несколько попыток для его ввода. При правильном вводе выполнить открытие двери и более не ожидать ввода пароля.

Конечный код программы представлен ниже:

```
#define LED1 PB9 // красный светодиод
#define LED2 PB8 // зеленый светодиод
#define LED3 PB1 // синий светодиод
#define Cool PA1 // вентилятор
#define NUM_KEYS 5 // количество знаков в пароле

char myarraw[NUM_KEYS] = { '1','4','2','-','2' }; // массив с верным кодом
char button_pressed1[NUM_KEYS]; // Переменная хранящая считанный пароль
char button_pressed; // Переменная для считывания пароля из монитора порта

int k = 0; // Переменная для счетчика неправильных попыток

void setup() {
  pinMode(LED1, OUTPUT); // красный светодиод
  pinMode(LED2, OUTPUT); // зеленый светодиод
  pinMode(LED3, OUTPUT); // зеленый светодиод
  pinMode(Cool, OUTPUT); // реле управления замком
  Serial.begin(115200);
  Serial.println("Для доступа в систему необходимо ввести 5-и значный пароль.");
```

```
  delay(1000);
}

bool strcheck(char* str1, char* str2) {
    int i = 0;
    while (str1[i] != '\0' && str2[i] != '\0') { // Цикл выполняется до тех пор,
пока
                                                 //не найдется хоть один конец
строки

        if (str1[i] != str2[i]) // Сравниваем каждый i элемент массивов друг с
другом
            return 0; // Если хоть один не совпал возвращаем False
        i++;
    }
    //Если мы вышли из цикла значит одну из строк мы перебрали до конца
    if (str1[i] == '\0' && str2[i] == '\0') // Проверяем на соответствие i
элемента
                                            //каждого массива элемента концу
строки
        return 1; // Если совпало, возвращаем True
    return 0; // Не совпало, возвращаем False
}

void loop() {
    Serial.begin(115200);
    digitalWrite(LED3, HIGH);
    while (!(Serial.available() > NUM_KEYS)) {}

    for (int i = 0; i < NUM_KEYS; i++) {

        button_pressed = Serial.read();
        button_pressed1[i] = button_pressed;

    }
    Serial.println(button_pressed1);
    bool c =strcheck(myarraw, button_pressed1);

    if (c == 1) { // Условие выполняется при возврате True
        k = 0; // Обнуление счетчика
        Serial.println(); // Переход на новую строку
        Serial.println("Пароль верный."); // Вывод сообщения о правильности пароля
        digitalWrite(LED3, LOW); // Выключаем синий светодиод
        digitalWrite(LED2, HIGH); // Включаем зеленый светодиод
        digitalWrite(Cool, HIGH); // Включаем мотор замка
        delay(1000); // Задержка
        digitalWrite(Cool, LOW); // Выключаем мотор замка
        while (1); // Цикл бесконечной истинности
    }

    else { // Иначе
        k++; // Инкремент счетчика
        if (k == 3) { // Если счетчик равен "", то
            Serial.println("В доступе отказано."); // Выводим сообщение о запрете
доступа
            digitalWrite(LED3, LOW); // Выключаем синий светодиод
            digitalWrite(LED2, LOW); //  Выключаем зеленый светодиод
            digitalWrite(LED1, HIGH); // Включаем красный светодиод
            while (1); // Цикл бесконечной истинности
        }
        else Serial.println("Пароль не верный. Попробуйте снова."); // Иначе
        digitalWrite(LED3, LOW); // Выключаем синий светодиод
        digitalWrite(LED2, LOW); // Выключаем зеленый светодиод
        digitalWrite(LED1, HIGH); //Включаем красный светодиод
        Serial.end(); // Очищаем буффер UARTa
```

```
        delay(2000);  // Задержка в 2 секунды
    }


    digitalWrite(LED3, LOW); // Выключаем синий светодиод
    digitalWrite(LED2, LOW); // Выключаем зеленый светодиод
    digitalWrite(LED1, LOW); // Выключаем красный светодиод
}
```

Домашнее задание:

- Найти все функции для работы с Serial в Arduino IDE и сформировать таблицу, где будет написано пояснение для использования каждой из них.
- Модернизировать код. Найти более простые или изящные решения. Реализовать не полную блокировку двери при ошибке, а на какое-то время, после чего разрешить ввод пароля.

Дополнительное задание:

- Реализуйте функцию сигнализации при определенном количестве неверных попыток.

Работа № 10

«Работа с экраном»

Экран – один из важных способов вывода информации, обладающий большим потенциалом в частности объема выводимой информации, ее читабельности и органичности. С применением экранов и графики стало возможным получать полный объем необходимой информации за короткий промежуток времени. Данное свойство экрана и будет реализовано в данной работе.

В данной работе вы научитесь реализовывать простейший графический интерфейс, показывающий уровень освящения в комнатах и включение света в каждой из них. Для этого нам понадобится обратиться к предыдущей работе с АЦП, где были задействованы датчики освещенности. Перед выполнением данной работы обратитесь к работе с АЦП и освежите в памяти выполняемые действия.

Цель работы: изучение принципов построения изображения на электронных дисплеях

Задачи:

1) Изучить алгоритмы, по которым строятся изображения на электронных дисплеях;

2) Написать программу, которая выводит информацию на дисплей платы TUSUR IoT Board.

1. Вводная часть

1.1 Что такое экран

В течение жизни мы часто сталкиваемся с примерами использования дисплея в электронике. Будь то электронные часы, дисплей на микроволновке или стиральной машине, который показывает оставшееся время. Такое широкое распространение дисплеи получили в связи с тем, что являются очень удобным средством вывода информации.

Дисплеи, используемые в электронике, можно разделить на:

- Сегментные;
- Алфавитно-цифровые;
- Графические.

Сегментные используются для индикации простых величин, например, температуры, времени, количества оборотов. Такие используются в калькуляторах и на бюджетной бытовой технике и по сей день. Информация выводится путем засвечивания определенных символов.

Чаще всего такие дисплеи состоят из множества семи-сегментных индикаторов (рисунок 1.1):

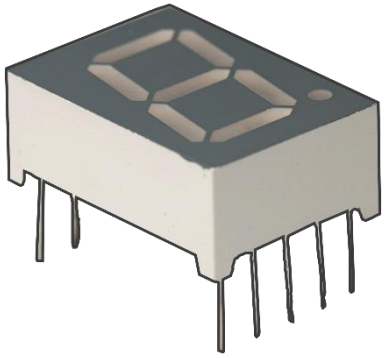


Рисунок 1.1 – Семи-сегментный индикатор

С помощью засвечивания определенного сегмента формируются цифры, символы и буквы. Они могут быть как жидкокристаллическими, так и светодиодными.

Алфавитно-цифровые дисплеи можно встретить на старой бытовой технике, игрушках, промышленной технике. Их еще называют знакосинтезирующими, текстовыми, символьными. Состоят из набора крупных пикселей (рисунок 1.2):

Рисунок 1.2 – Алфавитно-цифровой дисплей

Основное отличие данного дисплея от сегментного заключается в том, что на один символов приходится большее количество пикселей, что позволяет выводить практически любые символы.

К графическим дисплеям можно отнести даже монитор или экран смартфона. Их основное отличие заключается в том, что они представляют собой поверхность, на которой в той или иной пропорции рассыпаны пиксели (набор элементов, которые могу засвечиваться, чаще всего разными цветами).

Пример данного экрана можно наблюдать и на самой плате TUSUR_IOT_BOARD (рисунок 1.3):

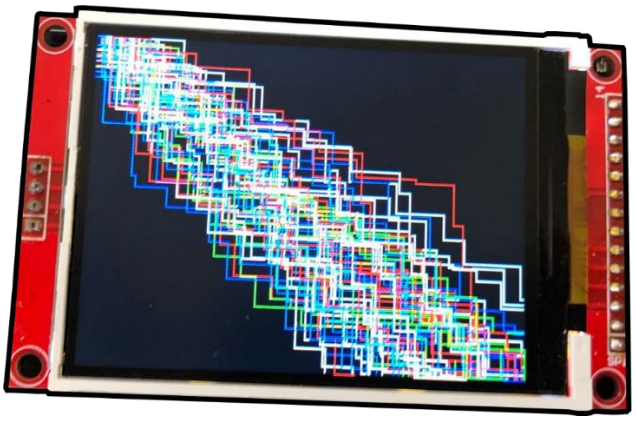


Рисунок 1.3 – Дисплей на плате TUSUR IoT Board

Формирование изображения на данных дисплеях представляет собой закрашивание ячеек матрицы. Дисплей на плате TUSUR_IOT_BOARD обладает разрешением 320x240 пикселей, то есть экран представляет собой матрицу размером 320x240.

Давайте разберем процесс построения геометрических фигур на матрице размером 32x24 (рисунок 1.4):

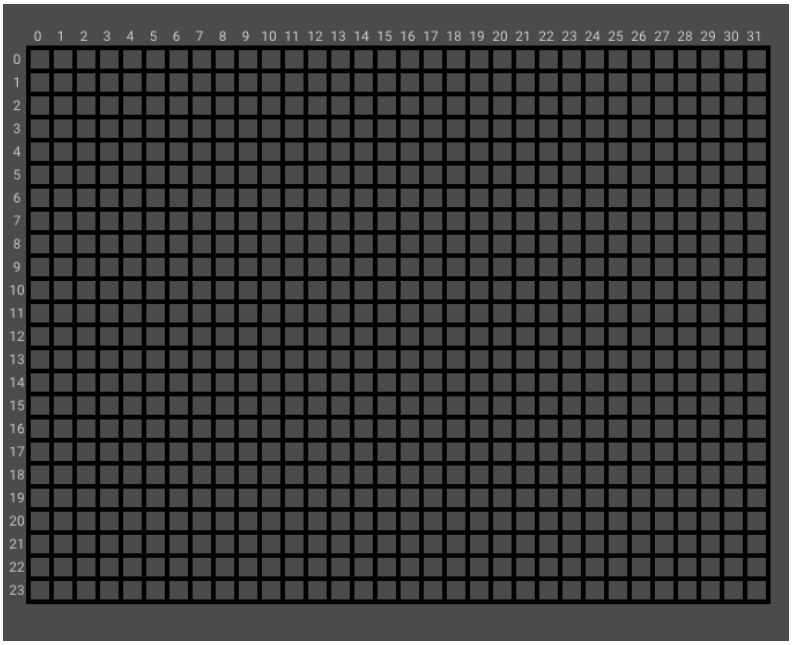


Рисунок 1.4 – Матрица для построения изображений

Чаще всего нумерация каждой ячейки начинается с левого верхнего уровня, то есть самый левый верхний элемент обладает индексом [0, 0].

Для начала давайте попробуем нарисовать линию, которая начинается из координат [3, 3] до координаты [28, 3].

Здесь мы сможем увидеть одну закономерность, что в данной линии у нас будет изменяться лишь одна координата – по горизонтали, по вертикали координата всегда будет оставаться прежней.

То есть, по сути, нам нужно закрашивать пиксель, меняя координату по горизонтали от 3 до 28.

Давайте проследим, как это будет выглядеть (рисунок 1.5):

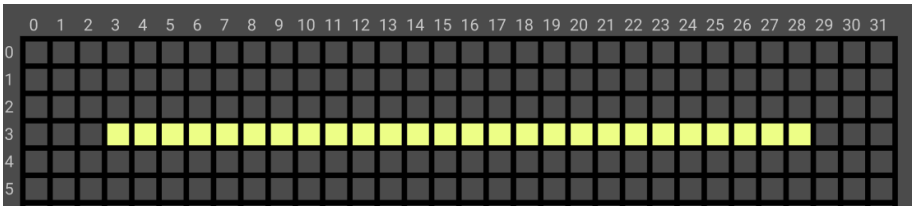


Рисунок 1.5 – Отображение линии

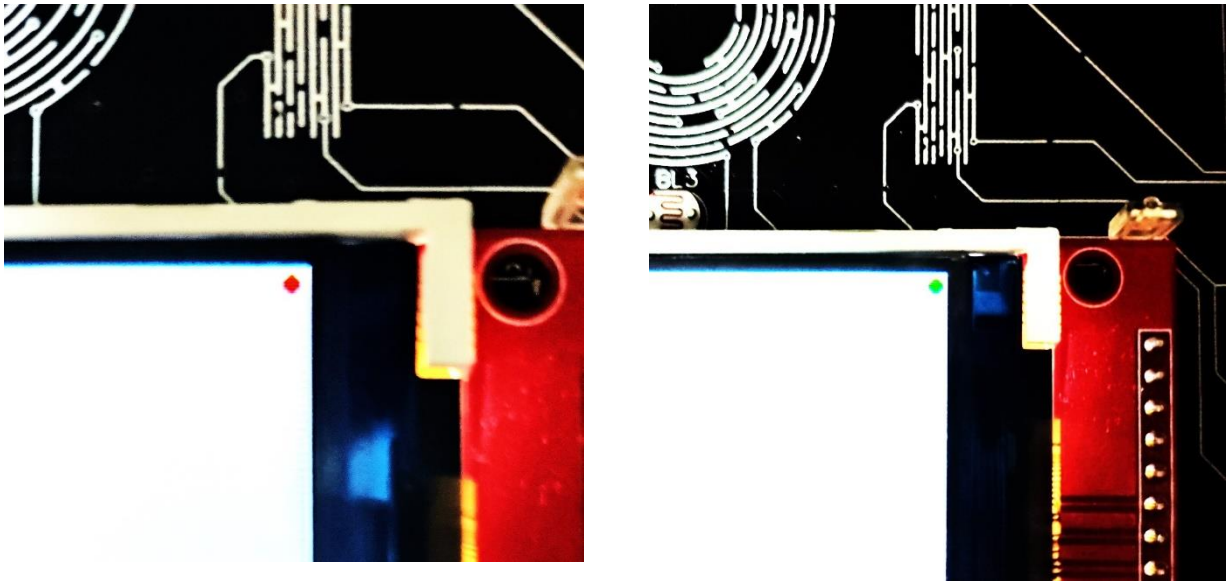
Здесь мы можем увидеть, что каждый закрашиваемый пиксель имеет следующие значения: [3, 3] [4, 3] [5, 3] [6, 3] … [28, 3] → [3-28, 3].

Для того чтобы выполнить рисование данной линии в программе с использованием Arduino IDE, необходимо ознакомиться со списком команд, которыми можно оперировать при взаимодействии с экраном. Для этого на экране главного меню выберете пункт «Вывод на экран». В открывшемся окне вы увидите все возможные команды для взаимодействия с экраном, а также палитру доступных цветов. На рисунке 1.6 представлено окно с командами и палитрами цветов для взаимодействия с экраном.

Рисунок 1.6 – Команды для взаимодействия с экраном

Далее для того, чтобы устройство было готово отображать ваши команды, необходимо еще раз нажать на энкодер. У вас должен измениться экран на полностью белый цвет.

Важно! В верхнем правом углу экрана располагается индикатор, отображающий корректность работы. Если вы посылаете неправильные команды, то данный индикатор будет гореть красным цветом, если все работает корректно, то индикатор будет гореть зеленым цветом, как показано на рисунке ниже.



а) б)

Рисунок 1.7 – Индикация корректности отправленной команды: а) – не корректно, б) – корректно

Для отображения на экране линии необходимо воспользоваться командой «tftDrawLine(x1,y1,x2,y2,цвет)». В данной команде x1 и y1 являются начальными координатами, а x2 и y2 – конечными. Для передачи такой команды необходимо передать такую строку через монитор порта. Откройте монитор порта в Arduino IDE, установите скорость равную 115200 бод и отправьте туда следующую команду: «tftDrawLine(3,3,30,3,5)» Проверьте, действительно ли у вас в верхнем левом углу экрана появилась линия красного цвета.

Важно! Все команды, которые содержат в себе «Set», необходимо указывать перед тем, как будет применяться команда, на которую ориентирована команда «Set». Так, например, если текст необходимо вывести по центру экрана, то сначала необходимо установить в это место курсор при помощи команды «tftSetCursor(x,y)», а уже за ней выводить текст при помощи команды «tftPrintln(Hi)». Попробуйте сначала ввести команду на вывод текста. Вы увидите, что текст появится в случайном месте. Теперь укажите координаты курсора, где бы вам хотелось вывести текст, а затем выведете текст. Вы заметите, что он переместился в интересующее вас место.

Важно! Все, что выводится на экран, остается на экране до момента стирания. Стирание осуществляется путем заливки интересующей области ранее установленным там цветом. Если цвет белый, то в аргументе «цвет» указывается 1, если зеленый – 3 и т.д. Полный список команд с пояснениями представлен в таблице 1.1.

Таблица **Ошибка! Текст указанного стиля в документе отсутствует.**.2 – Описание списка команд вывода на экран

Команда	Назначение
tftSetCursor(x,y)	Установка курсора в координату X,Y
tftFillRect(x1,y1,a,b,цвет)	Прямоугольник с полной заливкой от координаты верхнего левого угла (x1,y1) со сторонами a и b
tftDrawRect(x1,y1,x2,y2,цвет)	Прямоугольник от координаты верхнего левого угла (x1,y1) со сторонами a и b
tftDrawLine(x1,y1,x2,y2,цвет)	Линия с началом в (x1,y1) и концом в координате (x2,y2)
tftDrawPixel(x,y,цвет)	Изменить цвет пикселя в координате (x,y)
tftPrint(текст), tftPrintln(текст)	Вывести «текст» без переноса строки, вывести «текст» с переносом строки
tftFillScreen(цвет)	Залить весь экран определенным цветом
tftSetTextColor(цвет)	Установить цвет текста
tftDrawCircle(x,y,r,цвет)	Вывести круг с центром в координате (x,y) радиусом r с определенным цветом
tftFillCircle(x,y,r,цвет)	Залить область в центром в координате (x,y) радиусом r определенным цветом

Ниже приведена таблица цветов, доступных для использования:

Таблица 1.2 – Таблица доступных цветов для вывода на экран

Номер цвета	Цвет
1	Белый
2	Черный
3	Зеленый
4	Синий
5	Красный
6	Желтый
7	Оранжевый

Давайте попробуем собственноручно сделать меню, например, отображающее работу турбин ректора. Начнем с того, что весь экран необходимо залить синим цветом, для этого необходимо воспользоваться командой tftFillScreen(4).

Важно! Все координаты описаны для примера, вы можете использовать свои.

Далее необходимо отчеркнуть заглавие нашего меню, для этого введите команду tftDrawLine(1,20,320,20,1), для того, чтобы линия стала выразительнее, добавьте еще одну линию на один пиксель ниже. Установите курсор tftSetCursor(84,4) и цвет текста tftSetTextColor(1) (по умолчанию цвет текста - черный), выведите текст заглавия меню tftPrintln(Состояние турбин). Промежуточный результат должен выглядеть следующим образом:

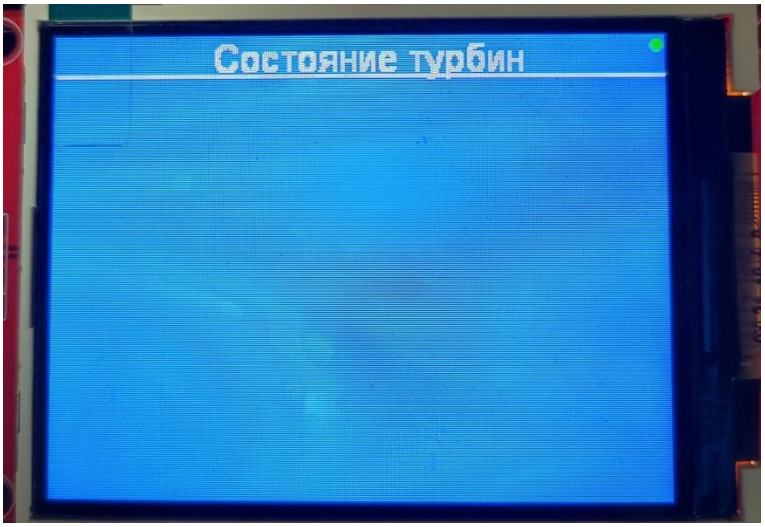


Рисунок 1.8 – Изображение на экране после первого этапа

Далее начнем выводить окна турбин. Для того чтобы задействовать как можно больше команд, предположим, что у нас 2 турбины. Выведете на экран два залитых прямоугольника, используя команду tftFillRect(20,25,280,100,6) для первого и ее же, с другими координатами, для второго прямоугольника. Далее необходимо создать рамку вокруг нижнего прямоугольника, выполнив имитацию последнего изменения в данном окне для этого воспользуйтесь командой tftDrawRect(20,135,280,100,5) и для выразительности рамки выведете еще один прямоугольник, но уменьшив его грани на 2, как бы вложив 1 прямоугольник в другой.

После данного этапа у вас должна получиться следующая картина:

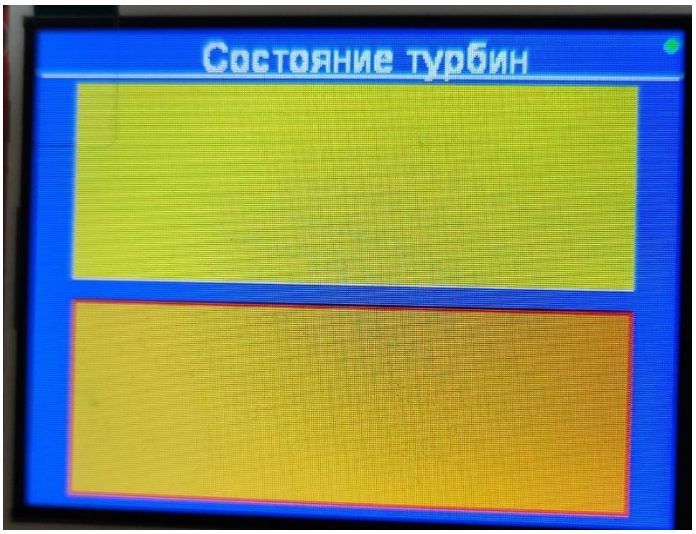


Рисунок 1.9 – Изображение на экране после второго этапа

Начнем выводить текст. Установите курсор в область первого прямоугольника и установите цвет текста на черный. Выведите текст «Турбина», пользуясь командой tftPrintln(Турбина), установите курсор на 20 пикселей ниже и выведете текст «Первого реактора», используя такую же команду.

Установите курсор в область второго прямоугольника и, пользуясь командой tftPrint(Текст), выведете текст «Турбина реактора 2».

После данного этапа у вас должна получиться следующая картина:

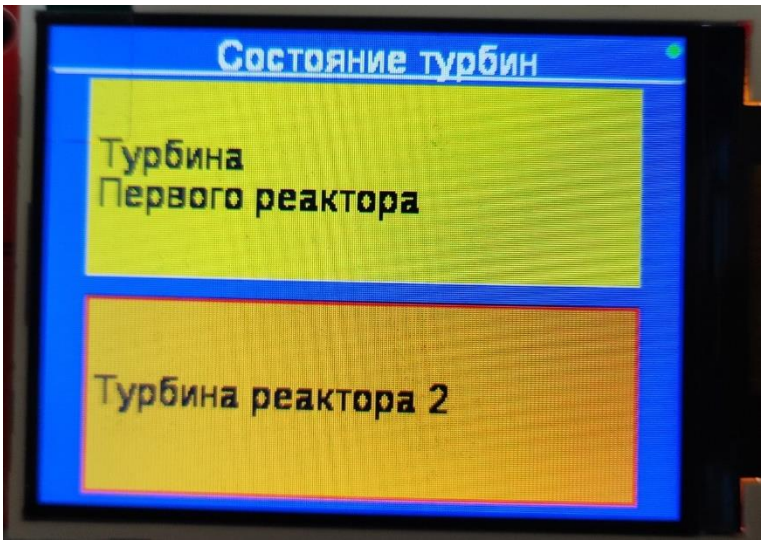


Рисунок 1.10 – Изображение на экране после третьего этапа

Далее выведем индикацию состояние турбины. Для этого выведете на экран в область каждого прямоугольника два круга, воспользовавшись командой tftDrawCircle(x,y,r,цвет), а затем область круга индикации состояния второй турбины необходимо залить зеленым цветом, показывая, что турбина находится в рабочем состоянии. Для этого воспользуйтесь командой tftFillCircle(x,y,r,цвет), указав координаты центра такие же, как и у круга, но радиусом на 1 меньшем.

После того, как вы проделаете всю работу, у вас на экране должно быть примерно следующее изображение:

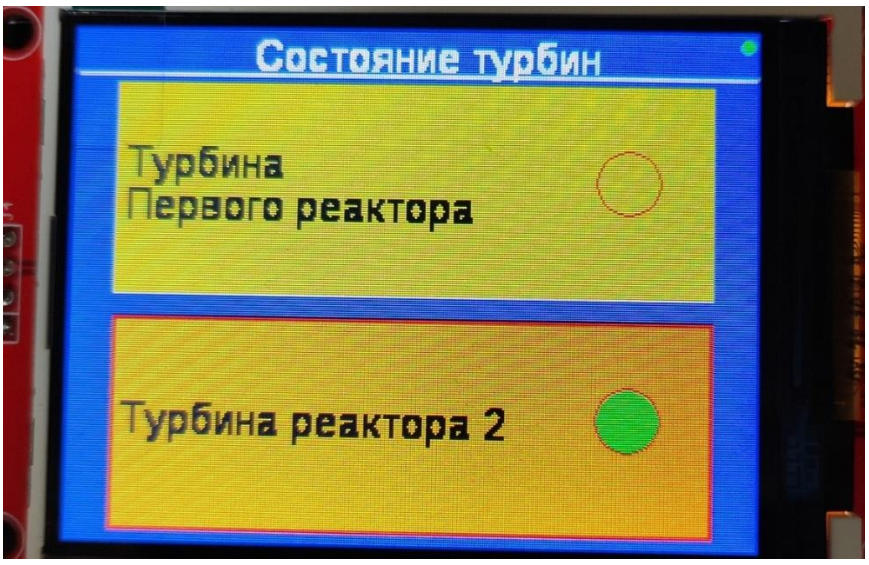


Рисунок 1.11 – Изображение на экране по окончанию работы

1.2 Вывод информации на экран в коде

Для того чтобы каждый раз не писать команды отрисовки вручную, что довольно долго и неудобно, данные команды можно вводить при помощи кода, заложив их логику на этапе программирования.

Для передачи такой команды необходимо создать строку с содержанием необходимой на данный момент команды. Таким образом, код вывода линии будет выглядеть следующим образом:

```
void setup()
{
	Serial.begin(115200); // Инициализируем UART (обязательно 115200)

}
void loop()
{
	String line = "tftDrawLine(3,3,28,3,5)"; // Создаем строку команды
	 Serial.println(line); //
	 delay(1000); //
}
```

Однако, если необходимо выполнить динамическое изменение координат, то код несколько преобразится. Динамическое изменение координат позволит не писать команды для отрисовки различных линий, а выполнить это при помощи одной команды, изменяя только координаты. Данный способ будет использоваться и для вывода динамических данных, например, температуры, скорости вращения вентилятора, уровня освещенности и т.д.

Код для вывода линии с динамическими координатами будет выглядеть так:

```
int x = 3; // создаем начальную координату X
int y = 3; // создаем начальную координату Y
int a = 28; // Создаем конечную координату X
int b = 3; // Создаем конечную координату Y
int color = 5; // Задаем цвет линии
void setup()
{
	Serial.begin(115200); // Инициализируем UART (обязательно 115200)

}
void loop()
{
	for (int i = 1; i < 8; i++) {
		x = 3*i;  // изменяем начальную координату X
		y = 3*i; // изменяем начальную координату Y
		a = 28*i; // изменяем конечную координату X
		b = 3*i; // изменяем конечную координату Y
		String line; // Создаем строку
		line += "tftDrawLine("; // Первый компонент строки
		line += x; // Нач.коорд. X
		line += ","; // Разделение
		line += y; // Нач.коорд. Y
		line += ","; //
		line += a; //
		line += ","; //
		line += b; //
		line += ","; //
		line += color; //
		line += ")"; // Закрываем скобки.
		Serial.println(line); //
		delay(1000); //
```

```
        }
}
```

При данном написании команды вывода информации на экран программист имеет возможность изменять координаты на необходимые значения и отрисовать линию. В данном коде представлена отрисовка 7 линий с разными координатами.

Важным является тот факт, что все команды необходимо отправлять последовательно!

1.3 Создание графических комнат

Как вы можете помнить, на плате TUSUR IoT Board имеется 3 фоторезистора. Значит, есть возможность присвоить каждой комнате по одному фоторезистору и отслеживать уровень освещенности в каждой из комнат.

На экране необходимо графически отобразить 3 комнаты. Для простоты изобразим их простыми геометрическими фигурами – квадратами. Так как координаты комнат изменяться не будут, то нет необходимости в динамическом изменении координат. Создадим 3 команды, которые будут при запуске рисовать границы комнат:

```
#define but PB4 // Вводим имя кнопки
// Границы комнат
String boxRed = "tftDrawRect(10,10,100,100,5)"; // Красная комната
String boxGreen = "tftDrawRect(111,120,100,100,3)"; // Зеленая комната
String boxBlue = "tftDrawRect(212,10,100,100,4)"; // Синяя комната
void setup()
{
       Serial.begin(115200); // Инициализируем UART (обязательно 115200)
        pinMode(but, INPUT); // Инициализируем кнопку, как ввод

       while (digitalRead(but) < 1) {} // Ожидаем нажатия кнопки для запуска

       Serial.println(boxRed); // Рисуем красную комнату
       delay(5);
       Serial.println(boxGreen); // Рисуем зеленую комнату
       delay(5);
       Serial.println(boxBlue); // Рисуем синюю комнату
}
```

Загрузите данный код на плату, перейдите в «Вывод на экран» и нажмите на кнопку. На белом фоне должны изобразиться 3 квадрата в позициях, отображающих позиции фоторезисторов за них отвечающих. Выглядеть это должно примерно так, как это показано на рисунке 1.7.

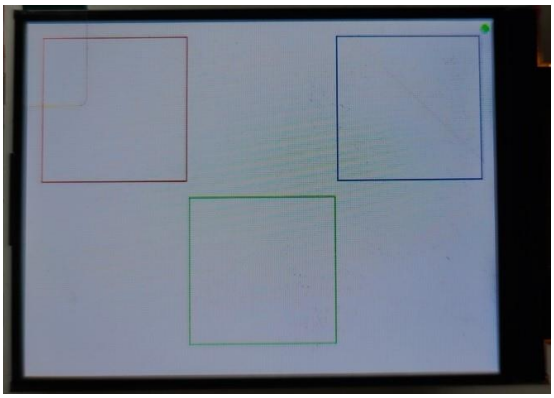


Рисунок 1.12 – Отображение 3-х комнат

2. Основная часть

2.1 Подготовка к реализации основной логики кода

Далее данные квадраты необходимо заполнить информацией. Начнем с того, что подпишем каждую из комнат. Для примера подпишем их как «Room 1» … «Room 3». Для этого необходимо создать команды для установки курсора в места, где будут осуществляться подписи, а также создать команды, выводящие текст подписи. Данные команды необходимо передать в микроконтроллер при запуске программы для того, чтобы при отрисовке комнат сразу вывелись их подписи.

Однако вывод подписей комнат необходимо осуществлять всегда, позже вы поймете для чего это необходимо.

Создайте 3 переменных типа «строка» для установки курсоров для вывода информации о номерах комнат и следом 3 команды с подписями комнат.

```
//Курсоры номеров комнат
String cursBoxRedNuber = "… (12,12)"; // Курсор номера красной комнаты
String cursBoxGreenNuber = "… (113,122)"; // Курсор номера зеленой комнаты
String cursBoxBlueNuber = "… (214,12)"; // Курсор номера синей комнаты

//Текст номеров комнат
String textBoxRedNuber = "… (Room 1)"; // Подпись красной комнаты
String textBoxGreenNuber = "… (Room 3)"; // Подпись зеленой комнаты
String textBoxBlueNuber = "… (Room 2)"; // Подпись синей комнаты
```

Теперь вам необходимо вызвать данные команды при запуске программы. Добавьте их ввод в `void setup()`. Загрузите код на плату и проверьте, действительно ли при запуске программы на экран выводятся 3 комнаты и их подписи. Если координаты оставались неизменными, то подпись должна находиться в верхнем левом углу каждой комнаты.

Теперь, когда у вас имеются комнаты, можно переходить к дальнейшей работе. Так как данная работы подразумевает вывод информации на экран о включении и выключении света, то необходимо подключить датчики освещенности, а также подключить RGB светодиод, который будет означать включение света в той или иной комнате, а информация на экране будет ее дублировать. Свитч необходим для его дальнейшего применения.

```
#define but PB4
#define SW1 PC15

#define FR1 PA1
#define FR2 PA2
#define FR3 PA3

#define RGB_B PB1
#define RGB_R PB9
#define RGB_G PB8
```

Таким образом выглядят все необходимые для работы подключения. Далее инициализируйте пины в `void setup()` для дальнейшей работы с ними.

2.2 Реализация логики работы программы

Далее начинается самая интересная часть работы. В `void loop()` реализуйте ожидание включения свитча, без которого выполнение программы не пойдет дальше. Далее введите глобальную задержку в 100 мс, чтобы система обладала небольшой инерционностью. Теперь необходимо считать данные с датчиков освещенности таким образом, как это осуществлялось в работе с АЦП.

Создайте строковые переменные, в которых будут храниться команды на вывод уровня освещенности. Однако, теперь данная команда будет хранить в себе динамическую переменную освещенности, что означает необходимость применения второго (составного) способа написания команды. Пример такого написания представлен в конце раздела 1.2 и ниже:

```
// Уровень освещенности
String boxRedLightLevel = "tftPrintln(";
boxRedLightLevel += val1Lux;
boxRedLightLevel += " Lux";
boxRedLightLevel += ")";
```

Напишите оставшиеся две команды для других комнат. Добавьте команды установки курсора для вывода информации о уровне освещенности.

Вы уже знаете, что информация, выведенная на экран, не стирается самостоятельно, поэтому ее необходимо стирать самостоятельно. Для этого создайте команды, заливающие каждый квадрат белым цветом – стирание информации, а также добавьте команды для каждой из комнат такого же размера, заливающие комнаты соответствующим им цветом. Это позволит «включать» индикацию включения света в комнате на экране и «тушить» свет в графической комнате.

Все настройки и подготовки к реализации логики работы программы выполнены, теперь можно переходить к ее написанию.

Дальнейший код будет состоять из 3-х блоков для каждой комнаты. Блок состоит из условного оператора с полным ветвлением. Вам необходимо реализовать следующую логику:

- Если уровень освещенности упал ниже порогового значения, то:
  - Включить соответствующий цвет светодиода цвету комнаты;
  - «Включить» индикацию включения света на дисплее;
  - Обновить вывод информации о номере комнаты (так как после «включения» индикации вся площадь квадрата зальется и текст пропадет);
  - Вывести информацию о уровне освещенности.
- Иначе:
  - Выключаем светодиод;
  - «Выключаем» индикацию включенного света;
  - Остальные пункты повторяются.

Важно, что после каждого вывода информации необходимо устанавливать небольшую задержку для того, чтобы микроконтроллер не «захлебывался» от данных по UART. Как вы помните, данный интерфейс асинхронный, а, значит, часть информации, которую содержит в себе команды, потеряется и произойдет сбой. Пример написания логики вывода информации для одной комнаты представлены ниже.

```
      //Red
      if (val1 < 150) {
            analogWrite(RGB_R, …);
            Serial.println(boxRedLight);
            delay(7);
            Serial.println(cursBoxRedNuber); // Устанавливаем курсор подписи
            Serial.println(textBoxRedNuber); // Выводим имя комнаты
            delay(10);
            Serial.println(cursBoxRedLight); // Устанавливаем курсор освещенности
            Serial.println(boxRedLightLevel); // Выводим значение освещенности
            delay(35);
      }
      else {
            analogWrite(RGB_R, …);
```

```
            Serial.println(boxRedOff);
            delay(7);
            Serial.println(cursBoxRedNuber); // Устанавливаем курсор подписи
            Serial.println(textBoxRedNuber); // Выводим имя комнаты
            delay(10);
            Serial.println(cursBoxRedLight); // Устанавливаем курсор освещенности
            Serial.println(boxRedLightLevel); // Выводим значение освещенности
            delay(35);
      }
```

Допишите данный код и загрузите на плату. Проверьте, что работа выполняется корректно и ваша плата реагирует на закрытие фоторезистора тем, что включается красный светодиод, «включается» красная комната, информация о комнате и уровне освещенности в ней. При открытии фоторезистора все должно возвращаться к начальному состоянию.

Продемонстрируйте работу преподавателю.

Завершите код для других комнат и продемонстрируйте работу преподавателю.

Итоговый код программы представлен ниже:

```
#define but PB4
#define SW1 PC15
#define beep PA8

#define FR1 PA1
#define FR2 PA2
#define FR3 PA3

#define RGB_B PB1
#define RGB_R PB9
#define RGB_G PB8

// Границы комнат
String boxRed = "tftDrawRect(10,10,100,100,5)"; // Красная комната
String boxGreen = "tftDrawRect(111,120,100,100,3)"; // Зеленая комната
String boxBlue = "tftDrawRect(212,10,100,100,4)"; // Синяя комната

//Индикация включенного света
String boxRedLight = "tftFillRect(11,11,98,98,5)"; // Включение Красная
String boxGreenLight = "tftFillRect(112,121,98,98,3)"; // Включение Зеленая
String boxBlueLight = "tftFillRect(213,11,98,98,4)"; // Включение Синяя

//Индикация выключенного света
String boxRedOff = "tftFillRect(11,11,98,98,1)"; // Выключение Красная
String boxGreenOff = "tftFillRect(112,121,98,98,1)"; // Выключение Зеленая
String boxBlueOff = "tftFillRect(213,11,98,98,1)"; // Выключение Синяя

//Курсоры номеров комнат
String cursBoxRedNuber = "tftSetCursor(12,12)"; // Курсор номера красной комнаты
String cursBoxGreenNuber = "tftSetCursor(113,122)"; // Курсор номера зеленой
комнаты
String cursBoxBlueNuber = "tftSetCursor(214,12)"; // Курсор номера синей комнаты

//Курсоры освещенности
String cursBoxRedLight = "tftSetCursor(12,42)"; // Курсор освещенности Красной
String cursBoxGreenLight = "tftSetCursor(113,152)"; // Курсор освещенности Зеленой
String cursBoxBlueLight = "tftSetCursor(214,42)"; // Курсор освещенности Синей

//Текст номеров комнат
String textBoxRedNuber = "tftPrintln(Room 1)"; // Подпись красной комнаты
String textBoxGreenNuber = "tftPrintln(Room 3)"; // Подпись зеленой комнаты
String textBoxBlueNuber = "tftPrintln(Room 2)"; // Подпись синей комнаты
void setup()
{
```

```
	Serial.begin(115200); // Инициализируем UART (обязательно 115200)
	pinMode(FR1, INPUT);
	pinMode(FR2, INPUT);
	pinMode(FR3, INPUT);

	pinMode(but, INPUT);
	pinMode(SW1, INPUT);

	pinMode(RGB_R, OUTPUT);
	pinMode(RGB_G, OUTPUT);
	pinMode(RGB_B, OUTPUT);


	while (digitalRead(but) < 1) {} // Ожидаем нажатия кнопки для запуска
		tone(beep, 100 * 25, 50);
		delay(200);
		tone(beep, 100 * 25, 50);

	Serial.println(boxRed); // Рисуем красную комнату
	delay(5);
	Serial.println(boxGreen); // Рисуем зеленую комнату
	delay(5);
	Serial.println(boxBlue); // Рисуем синюю комнату
	delay(15);
	Serial.println(cursBoxRedNuber); // Курсор номера Красной
	Serial.println(textBoxRedNuber); // Номер Красной
	delay(15);
	Serial.println(cursBoxGreenNuber); // Курсор номера Зеленой
	Serial.println(textBoxGreenNuber); // Номер Зеленой
	delay(15);
	Serial.println(cursBoxBlueNuber); // Курсор номера Синей
	Serial.println(textBoxBlueNuber); // Номер Синей
	delay(15);
}
void loop()
{

	while (digitalRead(SW1) < 1) {} // Ожидаем нажатия кнопки для запуска
	delay(200);
	// Считываем значения с фоторезисторов
	float val1 = analogRead(FR1)/8;
	float val2 = analogRead(FR2)/8;
	float val3 = analogRead(FR3)/8;
	float val1Lux = val1 / 6;
	float val2Lux = val2 / 6;
	float val3Lux = val3 / 6;
	float shim1 = 270 - val1;
	float shim2 = 270 - val2;
	float shim3 = 270 - val3;

	// Уровень освещенности
	String boxRedLightLevel = "tftPrintln(";
	String boxGreenLightLevel = "tftPrintln(";
	String boxBlueLightLevel = "tftPrintln(";
	boxRedLightLevel += val1Lux;
	boxRedLightLevel += " Lux";
	boxRedLightLevel += ")";
	boxGreenLightLevel += val3Lux;
	boxGreenLightLevel += " Lux";
	boxGreenLightLevel += ")";
	boxBlueLightLevel += val2Lux;
	boxBlueLightLevel += " Lux";
	boxBlueLightLevel += ")";

	//Red
```

```
      if (val1 < 150) {
            analogWrite(RGB_R, shim1);
            Serial.println(boxRedLight);
            delay(7);
            Serial.println(cursBoxRedNuber); // Устанавливаем курсор подписи
            Serial.println(textBoxRedNuber); // Выводим имя комнаты
            delay(10);
            Serial.println(cursBoxRedLight); // Устанавливаем курсор освещенности
            Serial.println(boxRedLightLevel); // Выводим значение освещенности
            delay(35);
      }
      else {
            analogWrite(RGB_R, LOW);
            Serial.println(boxRedOff);
            delay(7);
            Serial.println(cursBoxRedNuber); // Устанавливаем курсор подписи
            Serial.println(textBoxRedNuber); // Выводим имя комнаты
            delay(10);
            Serial.println(cursBoxRedLight); // Устанавливаем курсор освещенности
            Serial.println(boxRedLightLevel); // Выводим значение освещенности
            delay(35);
      }
      //Blue
      if (val2 < 150) {
            analogWrite(RGB_B, shim3);
            Serial.println(boxBlueLight);
            delay(7);
            Serial.println(cursBoxBlueNuber); // Устанавливаем курсор подписи
            Serial.println(textBoxBlueNuber); // Выводим имя комнаты
            delay(10);
            Serial.println(cursBoxBlueLight); // Устанавливаем курсор освещенности
            Serial.println(boxBlueLightLevel); // Выводим значение освещенности
            delay(35);
      }
      else {
            analogWrite(RGB_B, LOW);
            Serial.println(boxBlueOff);
            delay(7);
            Serial.println(cursBoxBlueNuber); // Устанавливаем курсор подписи
            Serial.println(textBoxBlueNuber); // Выводим имя комнаты
            delay(10);
            Serial.println(cursBoxBlueLight); // Устанавливаем курсор освещенности
            Serial.println(boxBlueLightLevel); // Выводим значение освещенности
            delay(35);
      }
      //Green
      if (val3 < 80) {
            analogWrite(RGB_G, shim2);
            Serial.println(boxGreenLight);
            delay(7);
            Serial.println(cursBoxGreenNuber); // Устанавливаем курсор подписи
            Serial.println(textBoxGreenNuber); // Выводим имя комнаты
            delay(10);
            Serial.println(cursBoxGreenLight); // Устанавливаем курсор
освещенности
            Serial.println(boxGreenLightLevel); // Выводим значение освещенности
            delay(35);
      }
      else {
            analogWrite(RGB_G, LOW);
            Serial.println(boxGreenOff);
            delay(7);
            Serial.println(cursBoxGreenNuber); // Устанавливаем курсор подписи
            Serial.println(textBoxGreenNuber); // Выводим имя комнаты
            delay(10);
```

```
            Serial.println(cursBoxGreenLight); // Устанавливаем курсор
освещенности
            Serial.println(boxGreenLightLevel); // Выводим значение освещенности
            delay(35);
      }
}
```

Подумайте, как можно упростить данный код.

Работа № 11

«Изучение интерфейса I2C»

Синхронный интерфейс I2C разработан фирмой Philips Semiconductors в начале 1980-х как простая 8-битная шина для внутренней связи управляющей электроники и рассчитан на частоту равную 100 кГц. Данный интерфейс был стандартизирован в 1992 году и, в дополнение к режиму со скоростью 100 кбит/с, был добавлен еще один скоростной режим, так называемый Fm (Fast-mode, Fm) со скоростью до 400 кбит/с. В 1998 году был введен режим Hs (High-speed mode, HS), что позволяло увеличить скорость до 3,4 Мбит/с. В 2007 и 2012 были добавлены режимы Fm+ (Fast-mode plus) со скоростью передачи 1 Мбит/с и механизмом идентификации устройств (ID), UFm (Ultra Fast-mode, UFm) – однонаправленный режим со скоростью 5 Мбит/с с применением двухтактной логики без подтягивающих резисторов.

Цель работы: Изучить способы передачи данных от контроллера к контроллеру/компьютеру.

Задачи

1) Изучить принцип работы интерфейса I2C;

2) Получить навыки по работе с интерфейсом I2C в среде Arduino IDE.

1. Теоретическая часть

1.1 Интерфейс I2C

Интерфейс I2C является синхронным интерфейсом. Для подключения используются 2 проводника: последовательная линия данных SDA (Serial Data, SDA) и последовательная линия тактирования SCL (Serial Clock, SCL) с напряжением 5 или 3,3В. На шине обязательно должен располагаться ведущий (Master) и ведомый (Slave). По линии тактирования передается синхронизирующий сигнал, который генерирует только Master. Поэтому интерфейс I2C и называется синхронным, поскольку присутствует линия синхронизации. Всего на одной двупроводной шине может располагаться до 127 устройств. На рисунке 1.1 представлено графическое представление подключения устройств по интерфейсу I2C.

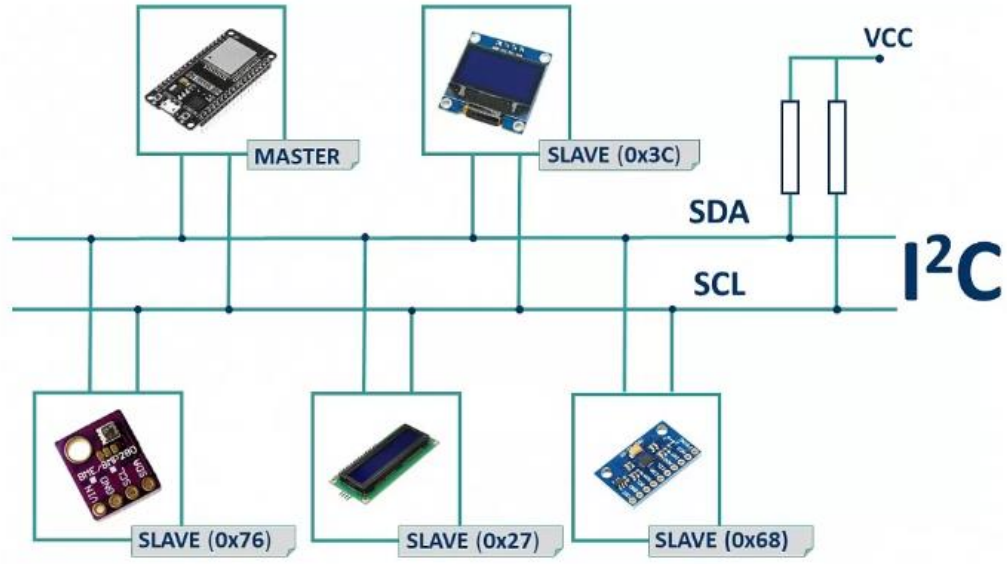


Рисунок 1.1 – Подключение устройств по I2C интерфейсу

Как видно из рисунка 1.1, все ведомые устройства имеют свой уникальный адрес, даже если такое устройство всего одно, ему все равно будет присвоен уникальный адрес. Однако, мастер не имеет адреса, что обуславливается тем, что все ведомые его знают. Ключевым в интерфейсе I2C является то, что все ведомые слушают или молчат до тех пор, пока к ним не обратится мастер по их уникальному адресу. Благодаря этому в сети царит порядок.

1.2 Физический уровень передачи информации по шине I2C

Как говорилось во введении, в начальный момент времени на шине SDA и SCL установлен высокий уровень (логическая единица), равный 5 или 3,3В за счет подтягивающих резисторов. Передача или прием сигналов осуществляется при «прижимании» линии к логическому 0. Важно оптимально подобрать подтягивающие резисторы. Чем больше номинал резистора, тем больше паразитная емкость между проводниками, а значит, будет больше времени уходить на восстановление логической 1. Это, в свою очередь, ведет к увеличению фронта нарастания импульса и, как следствие, к снижению скорости передачи информации. Наиболее распространенные скоростные режимы работы интерфейса I2C – это 10 и 100 кбит/с.

Передача данных состоит из старт-бита, бит информации и стоп-бита. На рисунке 1.2 представлено графическое представление передачи данных по интерфейсу I2C.

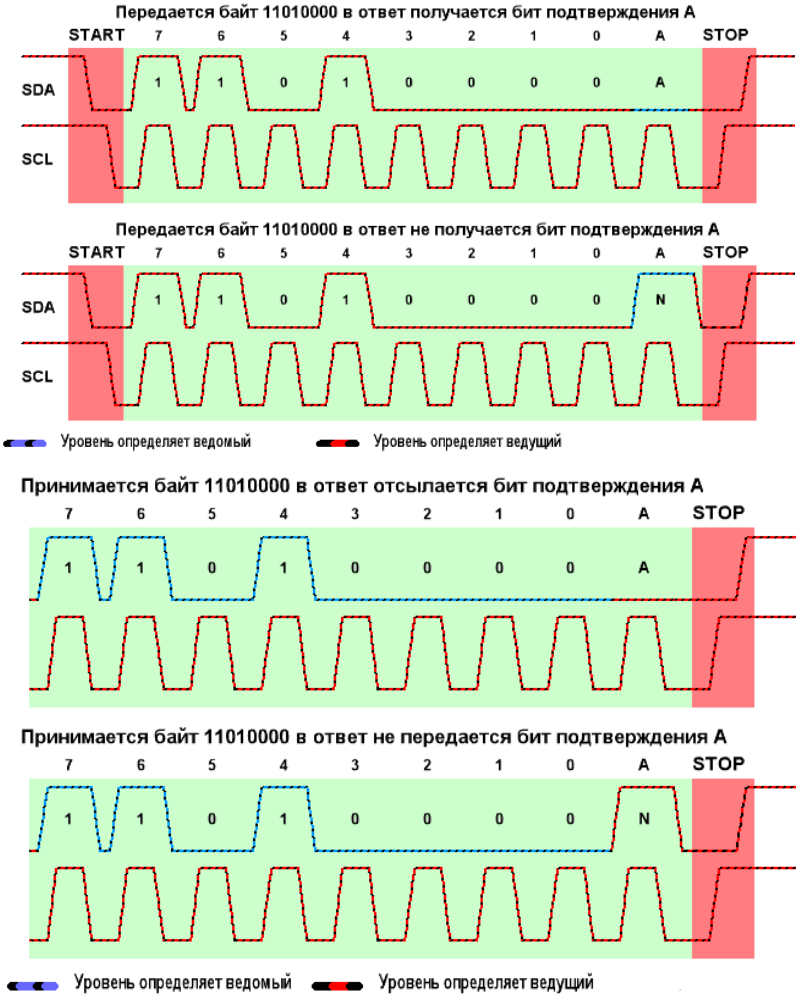


Рисунок 1.2 – Графическое представление передачи данных интерфейса I2C

Старт передачи начинается с того, что при высоком уровне на линии SCL линия SDA опускается в 0. После старта передачи данных передача одного бита данных идет по тактовому импульсу, что означает, когда на линии SCL выставляется логический 0, то Master или Slave выставляют бит на SDA («прижимают» линию, если передается 0 или оставляют высокий уровень – если передают 1), затем линия SCL возвращается в состояние 1, и Master/Slave считывают бит. Это позволяет передаче данных не зависеть от временных интервалов, а только от частоты тактовых бит. Стоп передачи осуществляется при высоком уровне SCL, линия SDA переходит в состояние 1.

Резюмируя вышеописанное:

1. Изменение на шине данных в момент приема может быть только при низком уровне SCL;
2. Когда SCL вверху – идет чтение;
3. При изменении SDA, при высоком уровне SCL, передается служебная информация (старт или стоп-бит).

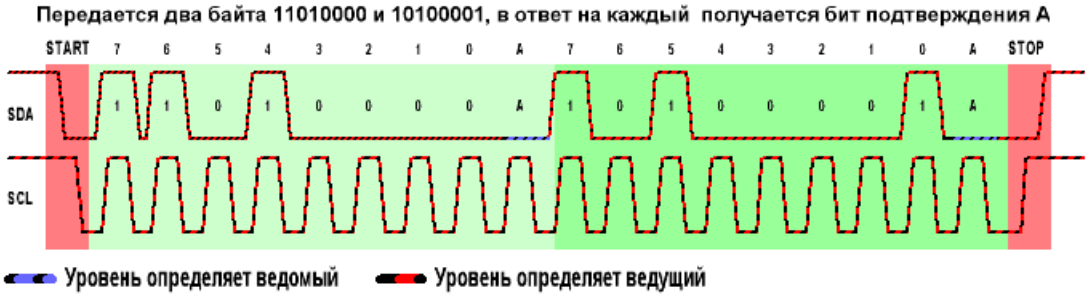


Рисунок 1.3 – Графическое представление передачи двух байт информации

Ранее говорилось, что только Master может управлять линией SCL, однако, это не совсем верно. В интерфейсе I2C есть ситуация, когда Slave может изменять состояние линии синхронизации. Это происходит тогда, когда Slave не успевает записать данные, которые посылает Master. В таком случае, Slave зажимает линию SCL и не дает мастеру генерировать биты до тех пор, пока Slave не закончит записывать предыдущие. Мастеру, в свою очередь, необходимо понять, что линия не поднялась, и дальше генерировать нельзя, а после того, как Slave закончил запись, продолжить с того момента, где передача данных прервалась. Графическое представление принудительного удержания SCL Slave(ом) представлено на рисунке 1.4.

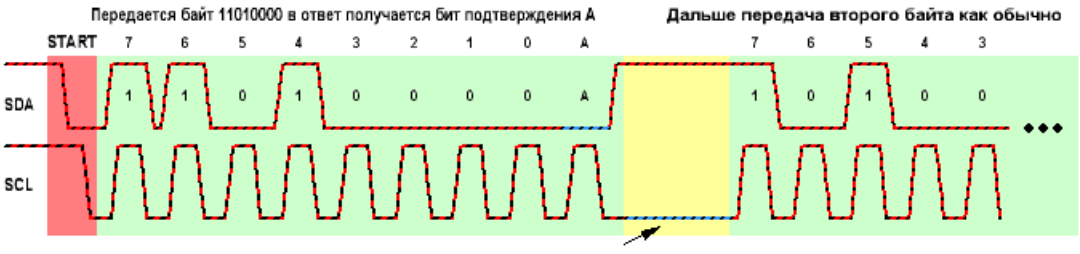


Рисунок 1.4 – Графическое представление принудительного удержания линии SCL Slave(ом)

1.3 Логический уровень передачи информации по шине I2C

В предыдущем разделе было рассказано, как передаются отдельные биты. Теперь необходимо разобраться, что они означают. Как говорилось ранее, в интерфейсе I2C имеется структура адресации. Данные шлются пакетами. Каждый пакет включает в себя девять бит: весь бит данных и один бит подтверждения или не подтверждения приема.

Первый пакет данных имеет уникальную структуру и шлет его Master, и он содержит адрес устройства и бит того, что должно сделать это устройство. На рисунке 1.5 представлено графическое представление первого пакета.

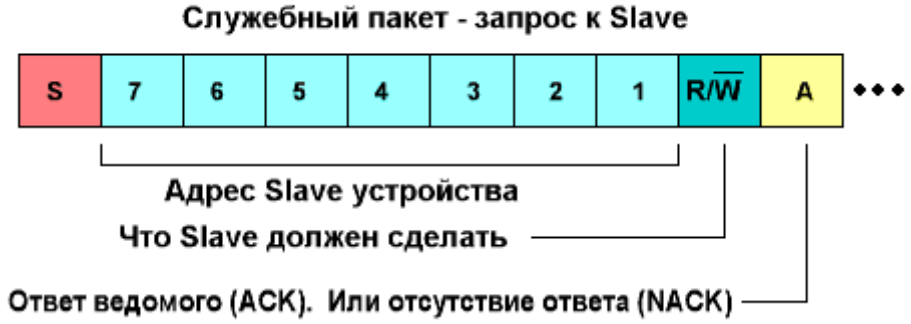


Рисунок 1.5 – Структура первого пакета

Как видно из рисунка 1.5, пакет состоит из адреса устройства длиной в 7 бит (именно поэтому существует ограничение в количестве подключаемых устройств), восьмой бит указывает, что необходимо сделать этому устройству на следующем байте – принимать или передавать данные, а девятый бит – бит подтверждения. Если адрес Slave совпал с переданным, то в момент бита подтверждения он опустит линию SDA в 0, тем самым дав понять Master(у), что он готов выполнять дальнейшую передачу или прием данных. Если же Slave не откликнулся на свой адрес по любой из возможных причин (неправильно принял посылку, сгорел и т.д.), то линию SDA некому будет переключить в состояние 0. Мастер поймет, что что-то не так и прекратит дальнейшую передачу данных к этому устройству до устранения неполадок.

После адресного пакета идут пакеты с данными от ведущего к ведомому или в обратную сторону, в зависимости от переданной в первом пакете команды. На рисунке 1.6 показано графическое представление следования пакетов от ведущего к ведомому в случае команды «запись». В случае «чтения» посылки будут выглядеть несколько иным образом. Их графическое представление представлено на рисунке 1.7.

Рисунок 1.6 – Графическое представление следования пакетов данных от ведущего к ведомому в I2C

Рисунок 1.7 – Графическое представление следования пакетов данных от ведомого к ведущему в I2C

Разница между посылками обуславливается тем, что изначально мастер передает посылку с адресом опрашиваемого устройства, команду на чтение, а дальше начинает

принимать данные от ведомого, давая на каждый байт ответ о приеме. Важно, что после приема последнего байта ведущий обязательно должен дать ответ о том, что он не принял пакет. Это необходимо для того, чтобы дать ведомому понять, что далее от него ничего не ждут и остановить передачу данных.

2. Практическая часть

2.1 I2C не на бумаге, как это выглядит

На плате TUSUR IoT Board содержится два датчика, подключенных к шине I2C: датчик атмосферного давления (BMP280) и датчик температуры (LM75A). Для того чтобы пронаблюдать сигнал на шине I2C необходимо воспользоваться логическим анализатором, как вы это делали в предыдущей работе с UART. Зайдите в подменю логического анализатора и выберите интерфейс I2C:

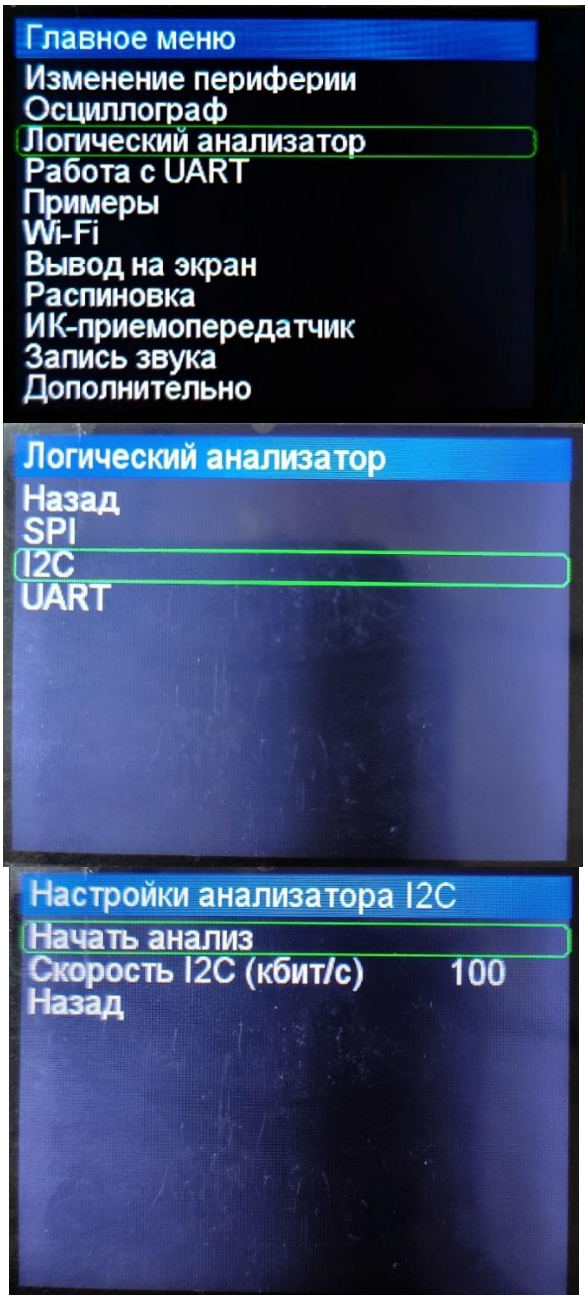


Рисунок 2.1 – Настройка логического анализатора для работы с I2C

Для того чтобы инициализировать работу I2C необходимо подключить библиотеку *Wire.h*. Библиотека *Wire.h* применяется для связи микроконтроллера с модулями и датчиками, подключенных к интерфейсу I2C. Данная библиотека является стандартной и устанавливается вместе с Arduino IDE.

Рассмотрим основные команды для работы с данной библиотекой, которые будут использованы в данной работе:

• Wire.begin(address) – инициализирует библиотеку *Wire* и подключается к шине I2C как ведущий или ведомый. Если указать адрес, то плата подключается как ведомый, если адрес не указывать – как мастер.

• Wire.requestFrom(address, quantity, stop) – обращаемся к ведомому устройству и запрашивает определенное количество байт. Address – адрес ведомого устройства, к которому отправляется запрос. Quantily – количество запрашиваемых байт от ведомого. Stop – необязательный параметр, позволяющий либо очистить шину I2C при значении True, либо не освобождать шину, а отправить дополнительный запрос при значении False.

• Часто используют команду while(Wire.available()), позволяющую ведомому устройству передать меньше байт информации, чем у него запросили.

После того, как вы ознакомились с библиотекой *Wire.h*, перейдем к написанию программы. В данной работе логический анализатор запускается в тот момент, когда ведущий начинает свою работу. Таким образом, вам необходимо подключить библиотеку *Wire.h* и инициализировать ее в `void setup()`, подключившись к шине I2C как мастер. Следующим шагом необходимо в `void loop()` отправить запрос к ведомому устройству и запросить у него один байт информации, поставить задержку. Рекомендуется поставить задержку в 5 с для того, чтобы была возможность увидеть как первую посылку с адресом от ведущего, так и байт информации, полученный от ведомого.

```
#include <...>

void setup()
{
      Wire.begin();          // подключиться к шине i2c (адрес для мастера не
обязателен)
}

void loop()
{
            Wire.requestFrom(..., ...,...);     // запросить 1 байт от ведомого
устройства 0x76
            delay();
}
```

Допишите данный код и продемонстрируйте результат своей работы. Результат, который вы должны увидеть в логическом анализаторе представлен ниже:

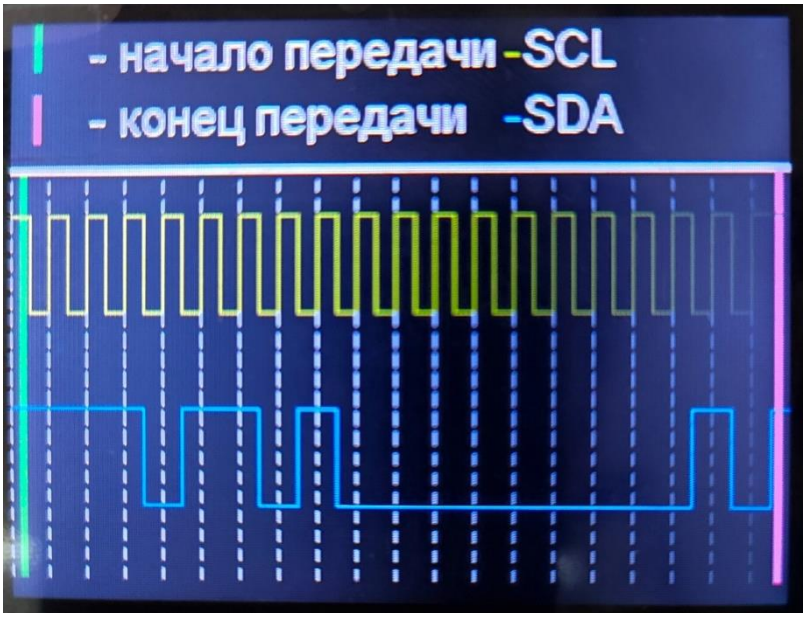


Рисунок 2.2 – Запрос ведущего и ответ ведомого размеров в 1 байт

Объясните, какой бит за что отвечает, и правильно ли работает ваша программа.
Домашнее задание:
Найдите и опишите все команды библиотеки *Wire.h*.

3. Работа с датчиком температуры

3.1 Датчик температуры LM75A

Датчик температуры LM75A широко распространен в системах термозащиты. Данный датчик оснащен выходом с открытым коллектором, который в даташите называется «O.S.», что позволяет по достижению заданной температуры, записанной в регистр датчика, самостоятельно выполнять функцию включения или отключения чего-либо, например, управлять заслонкой термостата или же включать и отключать вентилятор. Датчик подключается по интерфейсу I2C. На рисунке 1.3 представлены обозначения ввода/вывода датчика LM75A.

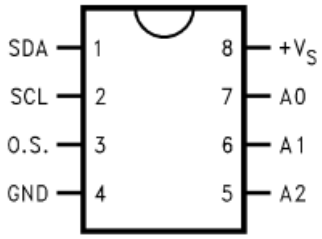


Рисунок 1.3 – Обозначения ввода/вывода датчика LM75A

Датчик может быть представлен в качестве готового модуля, а также и просто микросхемой, при установке которой необходимо учесть необходимость в подтягивающем резисторе для работы интерфейса I2C, конденсатора для фильтрации входа. На готовом модуле установлен светодиод с токоограничивающим резистором для него. Внешний вид датчика представлен на рисунках 1.4-1.5.

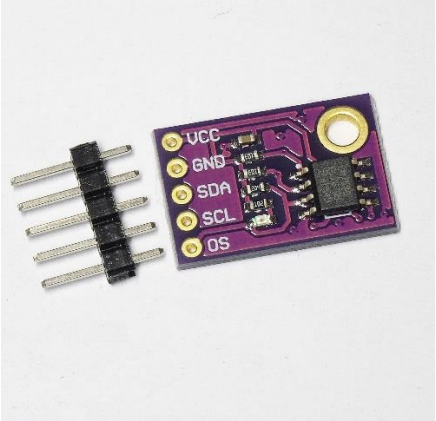


Рисунок 1.4 – Внешний вид модуля датчика температуры LM75A

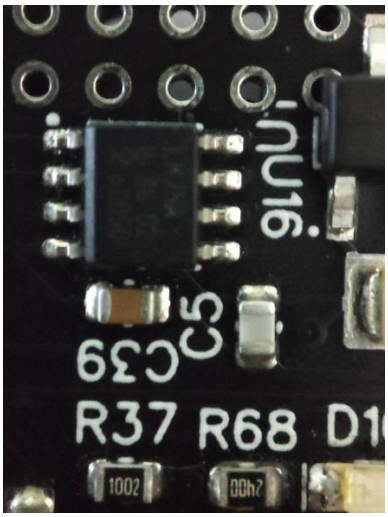


Рисунок 1.5 – Датчик температуры LM75A на плате TUSUR IoT Board

Основные технические характеристики представлены в таблице 1.2.

Таблица **Ошибка! Текст указанного стиля в документе отсутствует.**.3 – Технические характеристики датчика температуры LM75A

Характеристика	Значение
Напряжение питания	3,0-5,5 V
Рабочий ток питания	250 µA, 1 mA(макс), 4 µA (выкл.)
Погрешность измерения температуры	-25-+100 ˚C ±2˚C(max), −55˚C to 125˚C ±3˚C(max)
Выход с открытым коллектором	O.S.
Интерфейс подключения	I2C
Максимально подключаемых устройств на шину	8

Для работы с датчиком необходимо использовать библиотеку «M2M_LM75A.h». Если данная библиотека отсутствует, то ее необходимо установить через управление библиотеками в Arduino IDE.

После того, как библиотека была установлена, переходим к проверке работы датчика температуры. Создайте скетч и подключите библиотеку для работы с датчиком LM75A. Выполните подключение класса датчика для доступа ко всем функциям из библиотеки, используя команду: `M2M_LM75A lm75a`. Затем в `void setup()` необходимо инициализировать работу датчика и инициализировать работу UARTa. В `void loop()` создайте переменную типа `double` и считайте в нее данные с датчика температуры, воспользовавшись функцией `lm75a.getTemperature()`. Выведете считанные данные в монитор порта.

Если ваш код написан верно, и библиотечный адрес датчика совпадает с его физическим адресом, то в монитор порта должна выводиться информация о текущей температуре. Если информация не выводится, то проверьте адрес датчика температуры в библиотеке. Библиотека находится по пути C:\Users\UserName\Documents\Arduino\libraries\LM75A_Arduino_library\src и откройте файл M2M_LM75A с расширением «Исходный файл C Header» или .h. Найдите строку `#define LM75A_DEFAULT_ADDRESS 0x48 // Address is configured with pins A0-A2` и установите в ней адрес 0x48. Сохраните изменения и вернитесь к вашему скетчу, загрузите повторно. Если после этого вам не удалось получить данные с датчика, проверьте правильность вашего кода или обратитесь к преподавателю. Код для считывания данных с датчика LM75A представлен ниже:

```
#include <M2M_LM75A.h>
M2M_LM75A lm75a; // Подключение класса датчика

void setup() {
    // put your setup code here, to run once:
    lm75a.begin();
    Serial.begin(…);
    Serial.println(F("M2M_LM75A - Basic usage"));
    Serial.println(F("========================================"));
    Serial.println("");
}

void loop() {
    // put your main code here, to run repeatedly:
    double temp = lm75a.getTemperature();// Считывание температуры с LM75A
    Serial.print(F("Температура: "));
    Serial.print(…);
    Serial.println(F(" *C"));
    delay(1000);
}
```

Работа № 12

«Автоматическое управление микроклиматом на базе датчика температуры»

В предыдущей работе вы ознакомились с синхронным интерфейсом I2C, который будет задействован в данной работе. Данная работа посвящена автоматическому управлению микроклиматом в помещении, что является весьма актуальной задачей. Поскольку всегда хочется находиться в комфортных условиях и, желательно, не затрачивать время и силы для их создания.

В данной работе вы научитесь принимать и обрабатывать данные с датчика температуры, а также создадите программу, позволяющую включать и выключать вентилятор и изменять его скорость в зависимости от значения температуры.

Цель работы: создать систему автоматического управления климатом.

Задачи:

1) Научиться принимать данные с датчика температуры;
2) Научиться работать с данными с плавающей точкой.

1. Теоретическая часть

1.1 Модуль BMP280

Датчик BMP280, на самом деле, является высокоточным датчиком атмосферного давления с возможностью измерения температуры. Данный датчик возможно подключить по любому из двух возможных интерфейсов: SPI и I2C. На плате TUSUR IoT Board данный датчик подключен с использованием интерфейса I2C. На рисунке 1.1 представлена электрическая схема модуля BMP280.

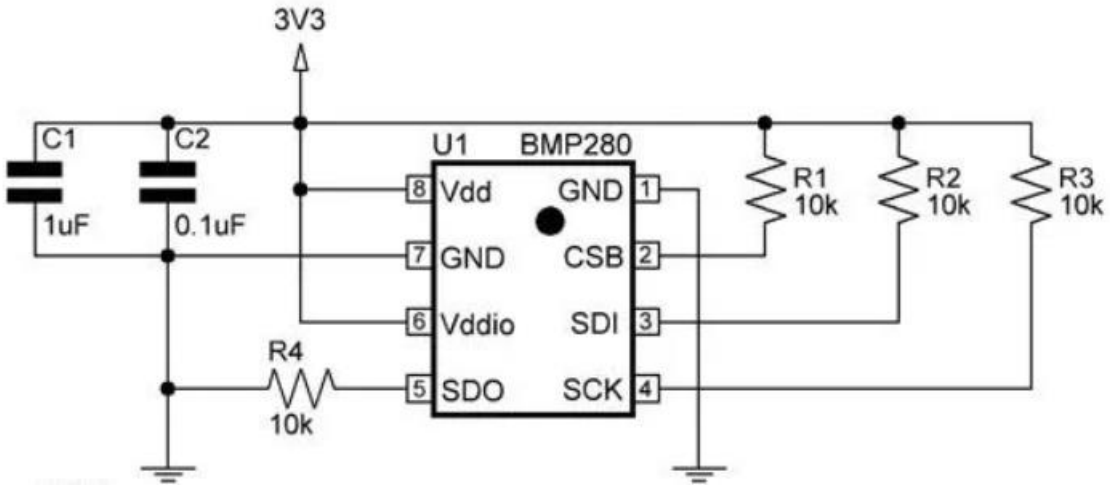


Рисунок 1.1 – Схема электрическая модуля BMP280

Как видно из схемы, в модуле предусмотрены фильтрующие конденсаторы по питанию и подтягивающие резисторы для интерфейсов ввода/вывода. Графический вид модуля представлен на рисунке 1.2.

Рисунок 1.2 – Внешний вид модуля BMP280

Данный модуль обладает следующими техническими характеристиками, представленными в таблице 1.1.

Таблица **Ошибка! Текст указанного стиля в документе отсутствует.**.4 – Технические характеристики модуля BMP280

Характеристика	Значение
Напряжение питания	1,71-3,6 V
Интерфейс обмена данными	I2C или SPI
Максимально подключаемых устройств на шину	8
Ток потребления в рабочем режиме	2,7 µA при частоте опроса 1 Hz
Диапазон измерения атмосферного давления	300-1100, hPa (±0,12), что эквивалентно -500-9000 м над уровнем моря
Диапазон измерения температуры	-40-85 ºC (±0,01)
Максимальная частота работы интерфейса I2C	3,4 MHz
Максимальная частота работы интерфейса SPI	10 Mhz
Размер модуля	21x18 mm

Для датчика BMP280 можно найти более десятка библиотек, однако библиотека «Adafruit_BMP280.h» является наиболее оптимальной для получения навыков в работе с данным модулем. Данная библиотека позволяет как выбирать способ подключения модуля (SPI или I2C), так и настраивать периодичность и точность измерений. Воспользуемся данной библиотекой и реализуем небольшую программу, которая позволит нам пронаблюдать выходные значения с модуля.

Для этого необходимо подключить две библиотеки: это *Whire.h* и *Adafruit_BMP280.h*. Если у вас отсутствует библиотека *Adafruit_BMP280.h*, то ее необходимо установить, как вы это делали ранее, через управление библиотеками. Выполнить подключение класса датчика для доступа ко всем функциям из библиотеки. Подключение класса выполняется следующим образом – `Adafruit_BMP280 bmp`. Таким образом на данном этапе ваш код должен содержать три команды:

```
#include <Wire.h> // Библиотека для работы с шиной I2C
#include <Adafruit_BMP280.h> // Библиотека для работы с датчиком BMP280

/* Подключение экземпляра класса Adafruit_BMP280
*/Adafruit_BMP280 bmp; /*для доступа к функциям датчика BMP280*/
```

Теперь ваш скетч готов к тому, чтобы начать взаимодействовать с модулем. Далее необходимо в `void setup()` инициализировать работу UARTa для того, чтобы выводить информацию с датчика в монитор порта и инициализировать модуль. Инициализацию лучше проводить с уловным оператором и информацией об ошибке с циклом бесконечной истинности. Это позволит при запуске программы удостовериться в том, что датчик подключен и действительно находится на правильном адресе. На плате TUSUR IoT Board датчик находится на 0x76 адресе. Это важно учитывать, поскольку иногда в библиотеке указан 0x77 адрес, и в таком случае ваша программа сразу даст вам знать, что что-то идет не так. Данный код указан ниже:

```
void setup() {
    Serial.begin(9600); // Для вывода отладочной информации в терминал
    if (!bmp.begin()) { // Если датчик BMP280 не найден
        Serial.println("Датчик BMP280 не найден"); // Выводим сообщение об ошибке
        while (1); // Переходим в бесконечный цикл
    }
}
```

Загрузите полученный код на плату и проверьте вывод в монитор порта. Если в монитор порта выводится сообщение об ошибке, то вам необходимо открыть библиотеку. Обычно путь к библиотеке это: C:\Users\User_Name\Documents\Arduino\libraries\Adafruit_BMP280_Library (по данному пути будет располагаться два файла с названием Adafruit_BMP280. Вам необходимо выбрать файл с расширением «Исходный файл C Header» или же .h), открыть его с помощью Visual Studio, Visual Studio Code или же блокнота, и в 32 строке изменить адрес с 0x77 на 0x76:

```
#define BMP280_ADDRESS (0x77) /**< The default I2C address for the sensor. */
```

Если адрес уже указан как 0x76, то необходимо искать проблему в вашем скетче!

После того, как проблема была решена, переходите к дальнейшему написанию программы.

Далее в `void loop()` необходимо реализовать вывод информации с модуля. Как мы помним, модуль может измерять атмосферное давление и температуру. На самом деле, на основе модуля BMP280 можно проводить также расчет высоты над уровнем моря. Для считывания температуры необходимо воспользоваться функцией `bmp.readTemperature()`. Данная функция считывает и возвращает значения температуры с модуля. Таким образом, для вывода температуры в монитор порта необходимо написать следующее:

```
void loop() {

    // Выводим значение температуры
    Serial.print(F("Температура = "));
    Serial.print(bmp.readTemperature()); // Функция измерения температуры
    Serial.println(" *C");

    delay(5000);
}
```

Вы можете заметить, что первая строка `Serial.print(F())` содержит в себе конструкцию `F()`. данная конструкция позволяет хранить данный текст в постоянной памяти контроллера и не использовать оперативную память для данного текста. Однако злоупотреблять подобным нельзя, поскольку это сокращает объем самой программы, которую вы можете написать, но, при нехватке оперативной памяти контроллера всегда можно воспользоваться таким методом, и тем самым разгрузить оперативную память.

Загрузите данный скетч на плату и проверьте его работоспособность.

Для вывода информации о давлении необходимо воспользоваться функцией `bmp.readPressure()` - выводит значение в паскалях. Для удобства восприятия можно перевести данное значение в привычные гектопаскали (hPa) или же в мм.рт.ст (mmHg). Для перевода в hPa необходимо поделить выходное значение на 100, а для перевода в mmHg необходимо поделить на 133,322.

Для вывода информации о высоте над уровнем моря необходимо воспользоваться функцией `bmp.readAltitude(1006.3).` В данную функцию передается значение об атмосферном давлении на уровне моря для текущего места. Например, для Томска это значение примерно равно 1006,3 hPa. Исходя из заданного давления на уровне моря и текущего давления, функция вернет значение высоты.

По аналогии с температурой выведете все остальные показания с модуля BMP280.

1.2 Датчик температуры LM75A

Датчик температуры LM75A широко распространен в системах термозащиты. Данный датчик оснащен выходом с открытым коллектором, который в даташите называется «O.S.», что позволяет по достижению заданной температуры, записанной в регистр датчика, самостоятельно выполнять функцию включения или отключения чего-либо, например, управлять заслонкой термостата или же включать и отключать вентилятор. Датчик подключается по интерфейсу I2C. На рисунке 1.3 представлены обозначения ввода/вывода датчика LM75A.

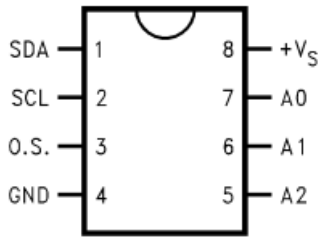


Рисунок 1.3 – Обозначения ввода/вывода датчика LM75A

Датчик может быть представлен в качестве готового модуля, а также и просто микросхемой, при установке которой необходимо учесть необходимость в подтягивающем резисторе для работы интерфейса I2C, конденсатора для фильтрации входа. На готовом модуле установлен светодиод с токоограничивающим резистором для него. Внешний вид датчика представлен на рисунках 1.4 – 1.5.

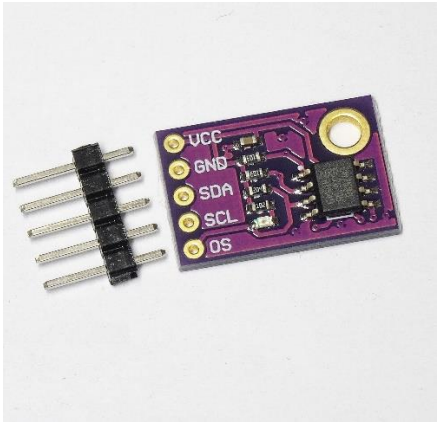


Рисунок 1.4 – Внешний вид модуля датчика температуры LM75A

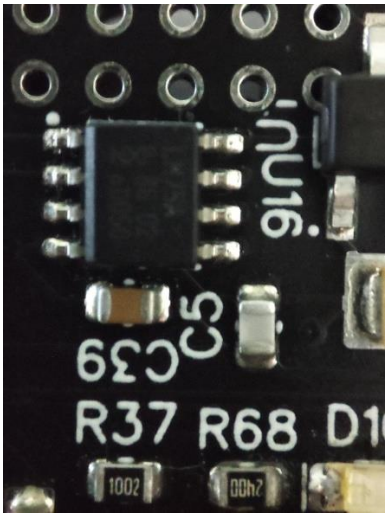


Рисунок 1.5 – Датчик температуры LM75A на плате TUSUR IoT Board

Основные технические характеристики представлены в таблице 1.2.

Таблица **Ошибка! Текст указанного стиля в документе отсутствует.**.5 – Технические характеристики датчика температуры LM75A

Характеристика	Значение
Напряжение питания	3,0-5,5 V
Рабочий ток питания	250 µA, 1 mA(макс), 4 µA (выкл.)
Погрешность измерения температуры	-25-+100 ˚C ±2˚C(max), −55˚C to 125˚C ±3˚C(max)
Выход с открытым коллектором	O.S.
Интерфейс подключения	I2C
Максимально подключаемых устройств на шину	8

Для работы с датчиком необходимо использовать библиотеку «M2M_LM75A.h». Если данная библиотека отсутствует, то ее необходимо установить через управление библиотеками в Arduino IDE.

После того, как библиотека была установлена, переходим к проверке работы датчика температуры. Создайте скетч и подключите библиотеку для работы с датчиком LM75A. Выполнить подключение класса датчика для доступа ко всем функциям из библиотеки, используя команду: `M2M_LM75A lm75a`. Затем в `void setup()` необходимо инициализировать работу датчика и инициализировать работу UARTa. В `void loop()` создайте переменную типа `double` и считайте в нее данные с датчика температуры, воспользовавшись функцией `lm75a.getTemperature()`. Выведете считанные данные в монитор порта.

Если ваш код написан верно и библиотечный адрес датчика совпадает с его физическим адресом, то в монитор порта должна выводиться информация о текущей температуре. Если информация не выводится, то проверьте адрес датчика температуры в библиотеке. Библиотека находится по пути C:\Users\UserName\Documents\Arduino\libraries\LM75A_Arduino_library\src и откройте файл M2M_LM75A с расширением «Исходный файл C Header» или.h. Найдите строку `#define LM75A_DEFAULT_ADDRESS 0x48 // Address is configured with pins A0-A2` и установите в ней адрес 0x48. Сохраните изменения и вернитесь к вашему скетчу, загрузите повторно. Если после этого вам не удалось получить данные с датчика, проверьте правильность вашего кода или обратитесь к преподавателю. Код для считывания данных с датчика LM75A представлен ниже:

```
#include <M2M_LM75A.h>
M2M_LM75A lm75a; // Подключение класса датчика

void setup() {
    // put your setup code here, to run once:
    lm75a.begin();
    Serial.begin(115200);
    Serial.println(F("M2M_LM75A - Basic usage"));
    Serial.println(F("========================================="));
    Serial.println("");
}

void loop() {
    // put your main code here, to run repeatedly:
    double temp = lm75a.getTemperature();// Считывание температуры с LM75A
    Serial.print(F("Температура: "));
    Serial.print(temp);
    Serial.println(F(" *C"));
    delay(1000);
}
```

2. Практическая часть

2.1 Создание программы автоматического управления микроклиматом

Как говорилось ранее, автоматическое управление микроклиматом – удобный инструмент для создания комфортных условий без участия в нем человека. Данная работа посвящена управлению вентиляционной установкой для поддержания комфортной температуры в помещениях с растущей температурой или серверных комнат.

Для данной работы будем применять датчик температуры LM75A. У вас уже готов код, позволяющий считывать данные температуры с датчика. Значит, можно переходить к написанию самой программы.

Как всегда, начинаем писать код с введением имен пинов, с которыми будет осуществляться работа. Для работы нам потребуются вентилятор, RGB-светодиод, свитч, бипер. Введите имена и далее в `void setup()` инициализируйте пины.

```
// Глобальные переменные
#define fan ... // PA1
#define R ... // PB9
#define G ... // PB8
#define B ... // PB1
#define SW1 ... // PC14 или PC15
#define beep ... // PA8
```

Далее переходим в `void loop()` и работаем там. Создайте переменную, которая будет хранить в себе ШИМ сигнал для подачи его на вентилятор. При этом уровень ШИМ сигнала должен изменяться в зависимости от температуры. Такую запись можно представить в следующем виде: `int rpm = (125*(temp-35))`, где temp – температура с датчика, а остальные значения – коэффициенты, служащие для калибровки скорости вращения вентилятора.

Реализуйте условный оператор, который будет обнулять скорость вращения вентилятора, если она ниже 55. Это необходимо для того, что для вентилятора уровня ШИМ сигнала в 55 будет недостаточно для запуска, однако, его подача будет осуществляться, имитируя заклинивание вентилятора, что не очень хорошо. Введите глобальную задержку в 1 с. Реализуйте условный оператор, работающий по включению свитча, внутри которого и будет выполняться основная логика работы программы.

При срабатывании условия включенного свитча необходимо:

• Включить светодиод, находящийся на пине PC13, для индикации начала работы системы охлаждения.

```
if(...){ //Если свитч включен
    digitalWrite (...,...); // Включаем PC13
```

• Откройте условный оператор с условием критической температуры, например, `temp >= 37`. При срабатывании данного условия необходимо, чтобы вентилятор вращался с максимальной (ШИМ=250) скоростью, Включить красный светодиод, подавать звуковые сигналы о аварийной ситуации, Выключить синий и зеленый светодиоды, вывести сообщение о перегреве в монитор порта, а также скорость вращения вентилятора.

```
    if (...){ // Если t>=37
          ...; // Устанавливаем максимальную скорость
          ...; // Включаем красный светодио
          ...; // Сигнализация
          ...; // Выключаем синий
          ...; // Выключаем зеленый
          ...; // Подаем ШИМ на вентилятор
          Serial.print(F("Скорость вентилятора: "));
          Serial.print(...);
          Serial.println(F(" RPM"));
    }
```

• Откройте условный оператор `else if` с условием не критической температуры, например, `temp < 37`. При срабатывании данного условия необходимо Выключить синий и красный светодиоды, Включить зеленый и подать на вентилятор рассчитанный ШИМ сигнал. Откройте еще один условный оператор `if` с условием низкой температуры, например, `temp <=35`. В данном случае вентилятор не должен работать, а гореть должен только синий светодиод. Закройте условие. Выведите информацию о скорости вращения вентилятора в монитор порта. Закройте условный оператор `else if`. Закройте условный оператор, работающий по включенному свитчу.

```
else if (...){ // Если t<37
          ...; // Выключаем синий
          ...; // Выключаем красный
          ...; // Включаем зеленый
          ...; // Подаем ШИМ на вентилятор


       if (...) {// Если t<=35
                ...; // ШИМ на 0
                ...; // Подаем ШИМ на вентилятор
                ...; // Выключаем красный
                ...; // Включаем синий
                ...; // Выключаем зеленый

       }
       Serial.print(F("Скорость вентилятора: "));
       Serial.print(...);
```

```
        Serial.println(F(" RPM"));
    }

}
```

• Откройте оператор полного ветвления. В нем необходимо отключить все светодиоды, отключить вентилятор и вывести сообщение об отключении системы охлаждения. Поставьте задержку в 50 мс. Закройте оператор ветвления.

```
else{ // Если свитч отключен
...; //Выключаем вентилятор
...; //Выключаем красный
...; // Выключаем синий
...; // Выключаем зеленый
...; // Выключаем PC13
    Serial.print(F("Система охлаждения отключена "));
    Serial.print("");
    Serial.println(F(" "));
    delay (50); // Обязательная задержка
  }
```

• Выведете сообщения о текущей температуре и закройте `void loop()`.

```
    Serial.print(F("Температура: "));
    Serial.print(temp);
    Serial.println(F(" *C"));
}
```

Загрузите код на плату и проверьте его работоспособность. Если все сделано правильно, то при отключенном свитче у вас должна вводиться только информация о текущей температуре и предупреждение о том, что система охлаждения отключена. При включенном свитче и температуре ниже 35 градусов – гореть светодиод PC13, гореть синий светодиод и выводиться сообщение о текущей скорости вращения вентилятора (должна равняться 0) и текущей температуре. При температуре от 35 до 35 должен гореть светодиод PC13, зеленый светодиод, вентилятор должен вращаться с изменяющейся скоростью в зависимости от температуры, выводиться сообщения о текущей скорости вращения вентилятора и температуре. При температуре выше 37 должен гореть светодиод PC13, красный светодиод, издаваться, сигнализирующий о перегреве, звук, вентилятор должен крутиться с максимальной скоростью (ШИМ=255), выводиться сообщения о перегреве, скорости вращения вентилятора и текущей температуре.

Конечный код программы представлен ниже:

```
// Библиотеки
#include <M2M_LM75A.h> // Подключаем библиотеку для работы с LM75A

// Глобальные переменные
#define fan PA1 // PA1
#define R PB9 // PB9
#define G PB8 // PB8
#define B PB1 // PB1
#define SW1 PC15 // PC14 или PC15
#define beep PA8 // PA8
#define but PB4 // PB4
#define SW2 PC14 // PC14 или PC15

M2M_LM75A lm75a; // Подключение класса датчика
```

```
void setup()
{
    tone(beep, 100 * 25, 50); // Сигнализация
    delay(200);
    tone(beep, 100 * 25, 50); // Сигнализация
    lm75a.begin(); // Инициализация работы датчика
    Serial.begin(115200); // Инициализация работы UART
    Serial.println(F("M2M_LM75A - Basic usage"));
    Serial.println(F("========================================="));
    Serial.println("");
    pinMode(R, OUTPUT); // Красный на выход
    pinMode(G, OUTPUT); // Зеленый на выход
    pinMode(B, OUTPUT); // Синий на выход
    pinMode(fan, OUTPUT); // Вентилятор на выход
    pinMode(beep, OUTPUT); // Бипер на выход
    pinMode(PC13, OUTPUT); // PC13 на выход

    pinMode(SW1, INPUT); // Свитч на вход
}

void loop(){

    double temp = lm75a.getTemperature(); // Считывание температуры с LM75A
    int rpm = (125*(temp-35)); // Задаем уровень ШИМ для вентилятора

    if (rpm<55){ // Если ШИМ ниже 55
        rpm=0; // Обнуляем ШИМ
    }

    delay (1000); // Глобальная задержка в 1 секунду

    if(digitalRead (SW1)){ //Если свитч включен
        digitalWrite (PC13, HIGH); // Включаем PC13

        if (temp >= 37){ // Если t>=37
            rpm=250; // Устанавливаем максимальную скорость
            analogWrite(R, 250); // Включаем красный светодио
            tone(beep, 100*25, 100); // Сигнализация
            analogWrite(B, 0); // Выключаем синий
            analogWrite(G, 0); // Выключаем зеленый
            analogWrite(fan, rpm); // Подаем ШИМ на вентилятор
            Serial.print(F("Скорость вентилятора: "));
            Serial.print(rpm);
            Serial.println(F(" RPM"));
        }

        else if (temp < 37){ // Если t<37
            analogWrite(B, 0); // Выключаем синий
            analogWrite(R, 0); // Выключаем красный
            analogWrite(G, 250); // Включаем зеленый
            analogWrite(fan, rpm); // Подаем ШИМ на вентилятор

            if (temp <=35) { // Если t<=35
                rpm=0; // ШИМ на 0
                analogWrite(fan, rpm); // Подаем ШИМ на вентилятор
                analogWrite(R, 0); // Выключаем красный
                analogWrite(B, 250); // Включаем синий
                analogWrite(G, 0); // Выключаем зеленый
            }

            Serial.print(F("Скорость вентилятора: "));
            Serial.print(rpm);
            Serial.println(F(" RPM"));
        }
```

```
    }
    else{ // Если свитч отключен
        analogWrite(fan, 0); //Выключаем вентилятор
        analogWrite(R, 0); //Выключаем красный
        analogWrite(B, 0); // Выключаем синий
        analogWrite(G, 0); // Выключаем зеленый
        digitalWrite(PC13, LOW); // Выключаем PC13
        Serial.print(F("Система охлаждения отключена "));
        Serial.print("");
        Serial.println(F(" "));
        delay (50); // Обязательная задержка
    }

    Serial.print(F("Температура: "));
    Serial.print(temp);
    Serial.println(F(" *C"));
}
```

Домашнее задание:

- Изучить библиотеку *Adafruit_BMP280*;
- Изучить библиотеку *M2M_LM75A.h*.

Дополнительное задание:

Реализовать программу с применением модуля BMP280.

Работа № 13

«Изучение интерфейса SPI»

Последовательный синхронный интерфейс передачи данных в режиме полного дуплекса – SPI (serial peripheral interface, SPI). Данный интерфейс отлично подходит для высокоскоростного сопряжения микроконтроллера с периферийными устройствами. Интерфейс SPI обладает намного большей скоростью передачи и простотой на физическом уровне, чем UART или I2C, а также все линии шины SPI являются однонаправленными. Однако у интерфейса есть и существенные недостатки: большее количество проводов для подключения периферии; нет возможности мультимастерной работы; менее стандартизирован, чем интерфейс I2C и UART.

Цель работы: Познакомиться с интерфейсом SPI и научиться им пользоваться.

Задачи:

1) Изучить принципы работы SPI интерфейса;
2) Получить навыки по работе с SPI интерфейсом в Arduino IDE.

1. Вводная часть

1.1 SPI интерфейс

В прошлых работах мы уже изучили 2 интерфейса для передачи данных между несколькими устройствами: I2C и UART. Но существует также еще один популярный интерфейс SPI.

Из расшифровки аббревиатуры можно понять, что одно из главных назначений интерфейса SPI – это связь ведущего с периферийными устройствами (ведомыми). Ведущий в интерфейсе SPI всегда один и именного он управляет всеми процессами, генерируя тактовые импульсы. В нашем случае ведущий – микроконтроллер, а ведомыми могут выступать дисплеи, датчики, микросхемы ЦАП/АЦП, RFID-ридеры, модули беспроводной связи и т.д. Данный интерфейс используется там, где необходима высокая скорость передачи данных и высокая надежность. SPI обладает данными качествами, поскольку он быстрее рассмотренных ранее интерфейсов, а также легче с точки зрения необходимых для его работы ресурсов.

Платой за скорость, надежность и легкость стало большое количество проводных соединений, необходимых для работы данного интерфейса, а именно:

- SCLK или SCK – Serial Clock (тактовый сигнал);
- MISO – Master Input Slave Output (ведущий принимает, ведомый передает);
- MOSI – Master Output Slave Input (ведущий передает, ведомый принимает);
- SS – Slave Select (выбор ведомого устройства).

Как можно заметить, интерфейс SPI взял от UARTa независимые шины передачи данных, что обеспечивает одновременную передачу и прием данных, от I2C – тактовый сигнал для синхронизации и повышения надежности передачи сигнала. Однако, есть некоторая сложность. В интерфейсе нет адресности, как, например, в I2C, а устройств к шине можно подключить несколько, поэтому необходимо как-то разделить общение с ведомыми устройствами. Для этого в SPI и существует подключение SS. У каждого ведомого устройства присутствует такой пин, а у ведущего должно быть столько выводов, сколько и ведомых устройств. Для того чтобы обратиться к конкретному датчику, ведущий опускает соответствующую линию SS в низкий уровень, ведомый это замечает, и они начинают общаться между собой.

Существует 3 способа подключения. Первый способ: один ведущий и один ведомый. Такой способ подключения самый простой и представлен на рисунке 1.1.

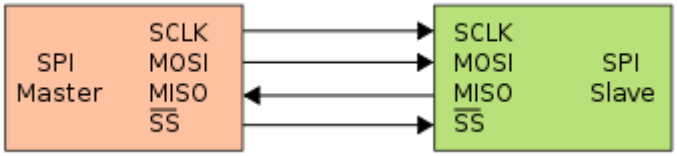


Рисунок 1.1 – Графическое представление первого способа подключения

Для подключения двух и более устройств используются два способа: классический и цепочка (кольцо) Классическое подключение заключается в том, что шины синхросигнала MISO и MOSI подключены параллельно, в шина SS ведет к каждому устройству индивидуально, соответственно, проводных соединений требуется: $x = 3 + n$, где n – количество ведомых устройств. Графическое представление такого подключения изображено на рисунке 1.2. Третий же тип подключения заключается в том, что SS для всех ведомых устройств один, но в таком случае данные передаются сквозь устройства. Плюсом такого способа подключения является уменьшение количества проводов, однако очень большой минус такого подключения – не все устройства поддерживают сквозную передачу данных. Третий способ подключения представлен на рисунке 1.3.

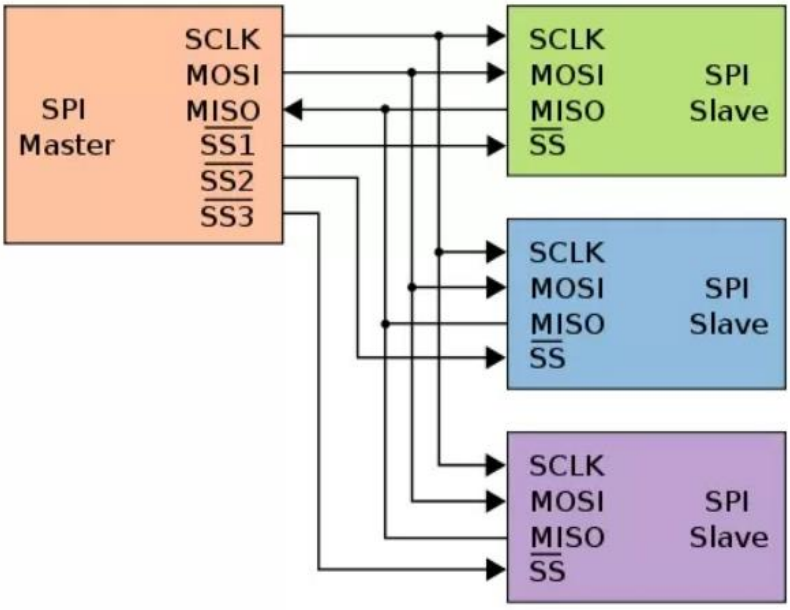


Рисунок 1.2 – Графическое представление второго способа подключения

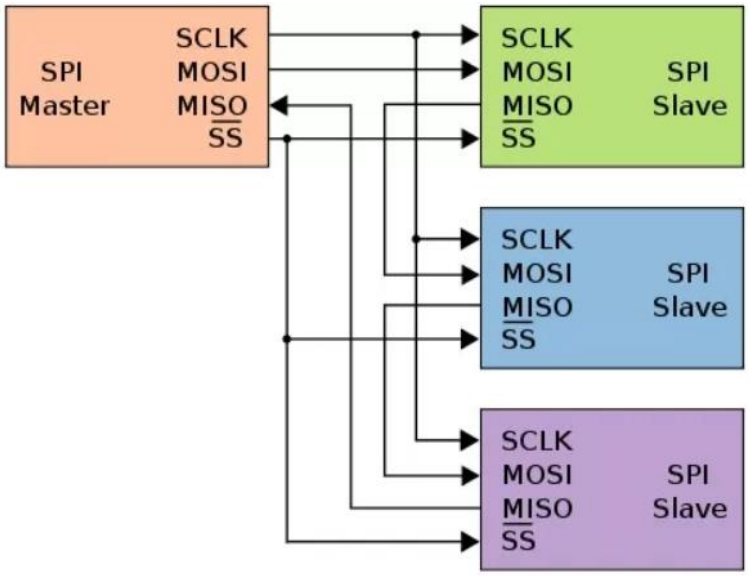


Рисунок 1.3 – Графическое представление третьего способа подключения

Протокол SPI имеет несколько режимов работы, которые описываются двумя параметрами:

- CPOL – исходный уровень сигнала синхронизации, низкий – 0 или высокий – 1;
- CPHA – фаза синхронизации, определяющий в какой момент импульса будет выполнена установка и считывание данных. По переднему фронту считывание, а по заднему установка – 0, наоборот – 1.

Естественно, очень важно, чтобы все устройства, подключенные к одной шине, работали на одинаковых режимах SPI. Существует 4 модификации режимов работы SPI со следующими комбинациями, приведенными в таблице 1.1.

Таблица **Ошибка! Текст указанного стиля в документе отсутствует.**.6 – Режимы работы интерфейса SPI

Режим	CPOL	CPHA
SPI_MODE0	0	0
SPI_MODE1	0	1
SPI_MODE2	1	0
SPI_MODE3	1	1

Для того чтобы лучше понимать, как влияет выбор режима работы непосредственно на работу интерфейса, посмотрим на графическое представление режимов работы интерфейса SPI, приведенного на рисунке 1.4, где R – чтение, W –запись.

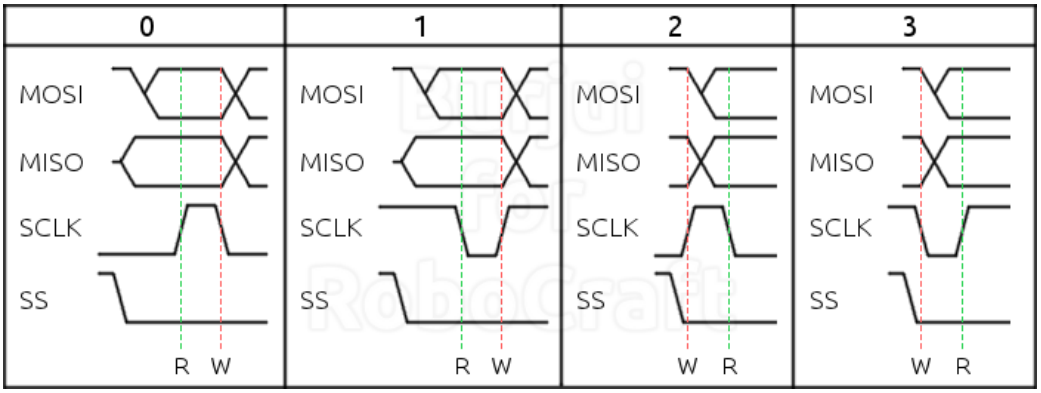


Рисунок 1.4 – Графическое представление режимов работы интерфейса SPI

Сама же приемопередача осуществляется по следующему принципу:

- Установка низкого уровня на SS нужного ведомого устройства;
- По низкому уровню SS схемотехника ведомого устройства переводится в активное состояние;
- Вывод MISO ведомого переходит в режим работы на выход;
- По тактовому сигналу происходит считывание на входе MOSI ведомого и сдвиг регистра.

Вся информация передается через сдвиговый регистр. Ведущий и ведомый выставляют биты в сдвиговый регистр и по синхросигналу начинают передавать биты. Биты передаются бит за битом и осуществляется сдвиг регистра. Передача осуществляется, начиная со старшего бита, но иногда производители делают возможным передачу бит, начиная с младшего разряда программными методами. После того, как пакет, состоящий из 1 байта, был передан, в целях синхронизации ведущий может перевести шину SS в высокое состояние.

Сигнал, передаваемый по интерфейсу SPI, выглядит следующим образом (рисунок 1.5):

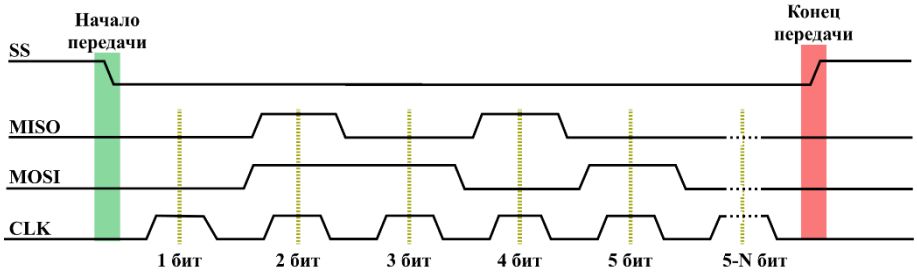


Рисунок 1.5 – Сигнал, передаваемый по интерфейсу SPI

Вычисление значения бита происходит по сигналам CLK. Если на линии CLK и на линии MOSI или MISO 1, значит, в данный момент времени передали 1 (единицу). Если же на CLK 1, а на лини MOSI или MISO 0, значит, в данный момент передали 0 (нуль).

2. Практическая часть

2.1 Знакомство с библиотеки SPI

Для изучения работы SPI интерфейса нам понадобится воспользоваться библиотекой <SPI.h>. Данная библиотека позволяет осуществлять работу с устройствами, работающими по SPI интерфейсу. Давайте разберем некоторые основные функции, которые в себе содержит библиотека, необходимые для нашей работы:

1. begin() и end() – инициализация и завершение работы SPI;
2. setBitOrder(order) – устанавливается порядок посылки бит данных:
   a. MSBFIRST – первым идет старший бит посылки (параметр по умолчанию);
   b. LSBFIRST – первым идет младший бит;
3. setClockDivider(divider) – установка делителя тактов для SPI относительно основной частоты микроконтроллера. Доступны делители: 2, 4, 8, 16, 32, 64 и 128. Для того чтобы установить соответствующий делитель, необходимо передать в функцию следующую переменную: SPI_CLOCK_DIVn, где n – делитель, например, SPI_CLOCK_DIV32. Если оставить этот параметр по умолчанию, то делитель будет равен 4;
4. setDataMode(mode) – задается режим работы SPI, используя имена режимов из таблицы 1.1. По умолчанию установлен нулевой режим работы;
5. SPI.transfer(data) – функция отправки данных через SPI.

2.2 Использование библиотеки SPI

Создайте скетч и подключите библиотеку «*SPI.h*». Если данная библиотека отсутствует, то добавьте ее через управление библиотеками. Далее необходимо написать программу, которая позволит пронаблюдать сигналы, передаваемые по всем шинам, задействованным в SPI. Создайте глобальную переменную, которая будет хранить данные, передаваемые по SPI. В `void setup()` инициализируйте работу UART, инициализируйте пин SS в состояние выход (SS находится на пине PA4), инициализируйте работу SPI. Далее необходимо выполнить настройку интерфейса SPI. Для этого задайте нулевой режим работы, укажите, что младший бит должен идти первым, а также установите делитель равный 64, что соответствует низкой скорости работы. Шаблон кода, который должен получиться на данный момент представлен ниже:

```
// Подключения
#include<SPI.h>

// Глобальные переменные
... i = 0; // Переменная, хранящая в себе данные для передачи по SPI

//Setup
void setup()
{
	...(...); // Инициализация UART
	...(..., ...); // Инициализация пина шины SS на выход
	SPI. ...(); // Инициализация 1 SPI порта
	SPI. ...(...); // Младший бит идет первым
	SPI. ...(...); // Установка SPI в нулевой режим
	SPI. ...(...); // Режим низкой скорости (72/64 = 1.125 МГц SPI_1 speed)
}
```

Далее необходимо создать функцию, которая будет отправлять данные по SPI. Данная функция, как следует из ее действий, не возвращает значений. В функции необходимо реализовать следующую логику:

- Прижать линию SS в 0, чем инициализировать передачу сообщения по SPI;
- Выполнить передачу информации;
- Отпустить линию SS в 1, чем прекратить передачу сообщения.

Шаблон кода функции представлен ниже:

```
... sendSPI(... ...)
{
	...(..., LOW); // Прижим SS к 0, начало передачи данных
	SPI. ...(...); // Функция передачи данных
	...(..., ...); // Отпускание SS в 1, конец передачи данныъ
}
```

После того, как функция была создана, переходим к написанию кода в теле `void loop()`. Создайте счетчик глобальной переменной, затем реализуйте задержку в 3-5 сек. После задержки вызовите функцию и передайте в нее вашу переменную, выведите значение переменной в монитор порта. Шаблон кода `void loop()` представлен ниже:

```
void loop()
{
	i...; // Инкриммент i

	...(...); // Задержка в 3 секунду
	...(...); // Вызов функции
	Serial. ...(...);
```

```
}
```

Для того чтобы посмотреть, как работает интерфейс SPI на логическом уровне, необходимо воспользоваться логическим анализатором, который необходимо запустить из меню платы по аналогии с предыдущими интерфейсами. Установите скорость в 1000 кбит/с и режим работы SPI – 0. Пронаблюдайте картину изменения передаваемого сигнала на логическом анализаторе. Самостоятельно задайте константное значение, передаваемой переменной, проверьте, действительно ли ее вы видите в анализаторе спектра?

Измените в анализаторе режим работы SPI на другой, посмотрите, что происходит в таком случае. Измените режим работы SPI в коде, посмотрите, как это влияет на его работу, продемонстрируйте разницу в режимах работы. Объясните эту разницу.

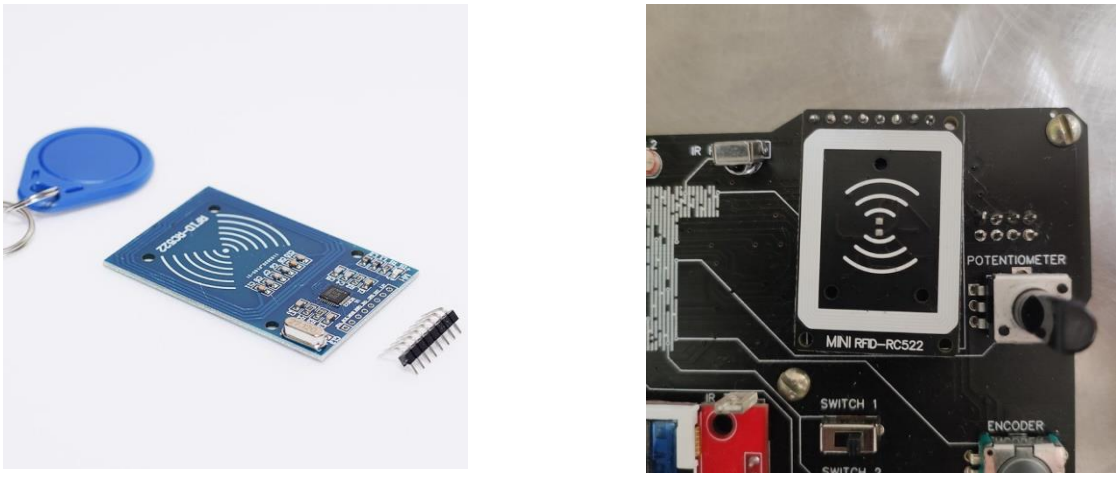
Итоговый код программы представлен ниже:

```
// Подключения
#include<SPI.h>

// Глобальные переменные
uint8_t i = 0; // Переменная, хранящая в себе данные для передачи  по SPI

// sendSPI
void sendSPI(uint8_t i)
{
	digitalWrite(PA4, LOW); // Прижим SS к 0, начало передачи данных
	SPI.transfer(i); // Функция передачи данных
	digitalWrite(PA4, HIGH); // Отпускание SS в 1, конец передачи данныъ
}

//Setup
void setup()
{
	Serial.begin(115200); // Инициализация UART
	pinMode(PA4, OUTPUT); // Инициализация пина шины SS на выход
	SPI.begin(); // Инициализация 1 SPI порта
	SPI.setBitOrder(MSBFIRST); // Младший бит идет первым
	SPI.setDataMode(SPI_MODE3); // Установка SPI в нулевой режим
	SPI.setClockDivider(SPI_CLOCK_DIV64); // Режим низкой скорости (72/64 =
1.125 МГц SPI_1 speed)
}

// Loop
void loop()
{
	i++; // Инкриммент i

	delay(3000); // Задержка в 3 секунду
	sendSPI(i); // Вызов функции
	Serial.print(i);
}
```

2.3 Работа с модулем RFID

На плате TUSUR IoT Board присутствует RFID-модуль (от англ. Radio Frequency IDentification, RFID (радиочастотная идентификация). Данный модуль подключен к контроллеру с использованием интерфейса SPI. Модуль RFID позволяет считывать и записывать RFID-метки. Сейчас RFID метки очень распространены от ключей домофона, до меток на одежде, служащих защитой от краж. Данные метки служат способом автоматической идентификации объектов посредством радиосигналов.

Для работы с RFID-модулем необходима соответствующая библиотека. В нашем случае используется RFID- модуль RC522, для которого существует библиотека «*MFRC522*». Внешний вид модуля RFID представлен на рисунке 2.1(а) и модуль RFID, установленный на плату 2.1(б).

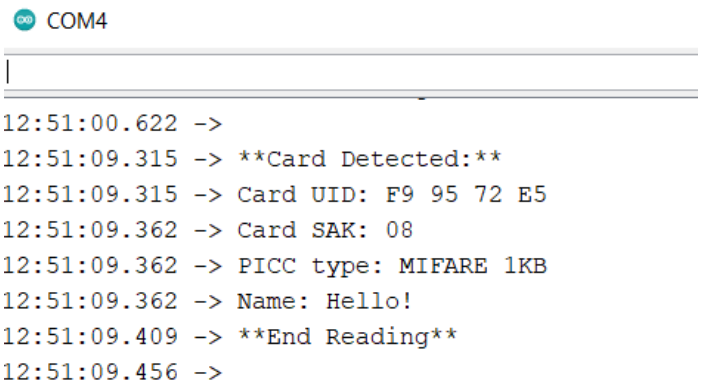


а) б)

Рисунок 2.1 – Внешний вид RFID-модуля RC522

Для того чтобы понять, как устроена память RFID-метки, необходимо воспользоваться встроенным примером из библиотеки. Для этого перейдите во вкладку «Файл» > «Примеры» > «MFRC522» > «rfid_read_personal_data»

В открывшимся скетче необходимо поменять назначение пинов: `RST_PIN` должен быть PB5, а `SS_PIN` – PB14. После чего загрузите данный скетч в память микроконтроллера. Откройте монитор порта и приложите RFID-метку к модулю. В монитор порта должна будет вывестись следующая информация, как показано на рисунке 2.2.



Из всей информации, что была считана с RFID, нам нужна только информация о имени (Name).

Соответственно, у каждой метки должно быть свое имя, ведь именно по нему происходит оценка свой/чужой. Попробуйте просмотреть записанные имена меток на нескольких картах.

Следующая работа будет посвящена созданию кодового замка на основе RFID метки.

Работа № 14

«Реализация кодового замка с применением RFID»

RFID в последнее время является одной из распространенных систем пропусков и безопасности: домофоны, турникеты, системы защиты от краж и системы присутствия (например, шахтера в шахте). Еще одна область применения RFID – банковские карты с бесконтактной оплатой. Для различных целей и задач используются разные типы RFID, и их характеризуют следующие параметры и вариации: активные, пассивные, высокочастотные и низкочастотные. Все это влияет на дальность связи, точность и объем информации, хранящейся на метке. Например, для банковских карт используются высокочастотные метки, так как на ней должно располагаться много информации, и скорость обмена должна быть также высока для реализации криптографии и иных шифровании для защиты информации.

В данной работе будет реализована пропускная система, которую можно применить на любом посте охраны, где необходимо разделение на «свой-чужой».

Цель работы: Реализация системы пропусков с применением RFID.

Задачи:

1) Реализовать систему записи данных на RFID метку;
2) Реализовать проверку данных, считанных с RFID метки.

1. Теоретическая часть

1.1 Введение в структуру памяти метки

В предыдущей работе вы уже познакомились с RFID модулем и RFID метками. Вы помните, что RFID метка содержит в себе UID. Для каждой карты UID уникален, однако сам UID не является уникальным по существу. Из-за того, что количество байт ограничено и весьма мало, то в конечном итоге UID повторяется, что может привести к несанкционированным действиям с применением карты «близнеца». Однако такое происходит очень редко и для небольшого предприятия достаточно выполнять проверку только UID карты, а значит, каждая карта уже сама по себе является ключом. Таким образом, для того, чтобы понимать, какая карта сейчас приложена к модулю, необходимо считать ее «дампа» (от англ. dump – свалка).

На прошлом занятии вы установили и провели небольшое знакомство с библиотекой «MFRC522.h». Вместе с данной библиотекой предоставляется большое множество примеров, и одним из них вы уже пользовались. Для того чтобы узнать, какую информацию несет в себе RFID метка, необходимо воспользоваться примером под названием «DumpInfo». Данный пример позволит вам считать всю информацию, что хранится на метке. Для этого перейдите к данному примеру, откройте его и загрузите на плату. Откройте монитор порта и приложите карту к считывателю. Подождите до тех пор, пока не будет считана вся информация.

Пример для считывания «дампа» с карты содержит в себе ряд функций, давайте рассмотрим их назначение:

- `mfrc522.PCD_Init()` – инициализация модуля RFID;
- `mfrc522.PCD_DumpVersionToSerial()` - вывод подробной информации о модуле;
- `mfrc522.PICC_IsNewCardPresent()` – проверка наличия карты в пределах считывания;
- `mfrc522.PICC_ReadCardSerial()` – чтение одной из карт и проверка ее корректности;

- `mfrc522.PICC_DumpToSerial(&(mfrc522.uid))` - чтение дампа карты.

Если все сделано правильно, и ваша карта считалась, то вы должны увидеть в мониторе порта примерно следующую картину (представлены начало и конец считывания):

```
10:26:08.874 -> Firmware Version: 0x12 = counterfeit chip
10:26:08.921 -> Scan PICC to see UID, SAK, type, and data blocks...
10:26:12.505 -> Card UID: 09 07 41 E9
10:26:12.505 -> Card SAK: 08
10:26:12.505 -> PICC type: MIFARE 1KB
10:26:12.505 -> Sector Block   0  1  2  3   4  5  6  7   8  9 10 11  12 13 14 15  AccessBits
10:26:12.646 ->   15     63   00 00 00 00  00 00 FF 07  80 69 FF FF  FF FF FF FF  [ 0 0 1 ]
10:26:12.693 ->          62   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
10:26:12.786 ->          61   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
10:26:12.880 ->          60   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
10:26:12.974 ->   14     59   00 00 00 00  00 00 FF 07  80 69 FF FF  FF FF FF FF  [ 0 0 1 ]
10:26:17.430 ->           4   31 31 33 30  33 32 31 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
10:26:17.477 ->    0      3   00 00 00 00  00 00 FF 07  80 69 FF FF  FF FF FF FF  [ 0 0 1 ]
10:26:17.571 ->           2   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
10:26:17.665 ->           1   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  [ 0 0 0 ]
10:26:17.712 ->           0   09 07 41 E9  A6 08 04 00  62 63 64 65  66 67 68 69  [ 0 0 0 ]
10:26:17.805 ->
```

Рисунок 1.1 – Считанный «дамп» с RFID

Как видно из сообщений, в первой строке указывается версия модуля. В моем случае версию модуля определить не удалось и вывелось сообщении о том, что чип поддельный. В третьей строке выводится информация о UID карты, который содержится в первых 4 байтах 0 блока 0 сектора, а далее идет информация о производителе и версии чипа. Данный срез памяти является не перезаписываемым. В четвертой и пятой строках выводится информация о производителе, версии чипа и объеме памяти EEPROM. В памяти также имеется служебная информация, и сектора с ней также являются не перезаписываемыми. Отследить их можно по тому, что в них при считывании уже содержится информация, исключением из примера с рисунка 1.1 является блок 4, поскольку в нем уже записана пользовательская информация.

Получив информацию о UID карты уже можно переходить к реализации пропускной системы. Однако если во всех картах, используемых на предприятии, будет иметься хотя бы две с идентичным UID, то необходимо усложнить процедуру идентификации карты.

Мы будем рассматривать простой случай, когда UID карт не повторяется, и необходимо проверять и выполнять логику работы замка только основываясь на нем.

Теперь, когда мы знаем UID, как можно посмотреть UID метки, мы можем переходить к написанию кода для кодового замка.

2. Практическая часть

2.1 Создание кодового замка на основе RFID

Для создания кодового замка на основе RFID на плате TUSUR IoT Board вам понадобится воспользоваться двумя библиотеками: «SPI.h» и «MFRC522.h». Подключите их, как вы делали в предыдущей работе.

Создай переменные, отвечающие за пины. Вам потребуется: SS вывод модуля RC522, RST вывод модуля RC522, бипер, RGB светодиод, вентилятор, имитирующий работу электрозамка.

Создайте переменную, которая будет хранить в себе данные о UID карты, которую мы будем расценивать, как «свой». Рекомендую использовать тип переменной с указателем «`char*`», что позволит обращаться к памяти, в которой будет располагаться ваша переменная с первой ячейки памяти. Запись будет выглядеть примерно так: `char* keys[] = { "202 168 186 128","73 36 43 229" }`, где переменная *keys* будет масштабироваться в зависимости от объема переданной в нее информации, а информация указывается в фигурных скобках блоками по 4 байта (размерность UID одной карты), помещенными в двойной апостроф. UID карт разделяются запятыми. Выполните подключение модуля, как делали это в предыдущей работе.

Далее в `void setup()` выполните инициализацию интерфейса UART, SPI и модуля RFID. Инициализировать все используемые пины и в конце вывести сообщение в монитор порта о том, что необходимо поднести карту.

Таким образом, вы должны написать код примерно следующего содержания:

```
#include <…>
#include <…>

#define SS_PIN  …       // SS вывод модуля RC522
#define RST_PIN  …      // RST вывод модуля RC522

#define BUZZER  …       //Пин для Пищалки
#define LED_R  …        //Пин для Красного цвета
#define LED_G  …        //Пин для Зелёного цвета
#define LED_B  …        //Пин для Голубого цвета
#define LOCK  …         // подключаем электрозамок


char* keys[] = { "202 168 186 128","73 36 43 229" }; // Создание переменной с
указателем в которой хранятся UID «своих» карт

MFRC522 mfrc522(…, …);   // Create MFRC522 instance.

void setup()
{
    …;              // Инициализация UART
    …;              // Инициализация SPI
    …;       // Инициализация MFRC522
    pinMode(…, …);   //объявляем пин как выход.
    …;   //объявляем пин как выход.
    …;   //объявляем пин как выход.
    …;  //объявляем пин как выход.
    …;    //объявляем пин как выход.
    …; // Выключение бипера
    …;  //Выключаем замок
    …("…");//Вывод сообщения о необходимости поднести карту
}
```

Допишите данный код и загрузите его на плату. Проверьте его правильность компиляции и вывод сообщения в монитор порта.

Далее, по аналогии с примером для считывания «дампа», необходимо выполнить проверки наличия карты в области считывателя и корректность работы карты. Для этого необходимо воспользоваться условными операторами и функциями `mfrc522.PICC_IsNewCardPresent()` и `mfrc522.PICC_ReadCardSerial()`. И если любое из этих условий не выполняется, то необходимо прервать работу. Структура каждой из таких проверок будет выглядеть так:

```
    if (!Условие) {
        return;
```

```
    }
```

Далее необходимо вывести сообщение о том, что сейчас здесь будет выводиться UID. Создайте переменную строкового типа, которая будет хранить в себе данные о считанном UID.

Далее необходимо открыть цикл от 0 до длины UID. Для того чтобы узнать длину UID можно воспользоваться командой «`mfrc522.uid.size`». Далее выведем в монитор порта данные, считанные с UID. Для этого необходимо пробежаться по байтам памяти и считать их. Для этого необходимо воспользоваться функцией «`mfrc522.uid.uidByte[i]`» и если значение меньше <0x10 – вывести «” 0“», иначе вывести «“ ”», а следующей строкой необходимо выполнить вывод считанных данных, преобразованных в десятичную систему счисления. Для этого необходимо написать следующее: «`Serial.print(mfrc522.uid.uidByte[i], DEC);`». через запятую указывается система счисления, «DEC» означает десятичную систему счисления. Далее в этом же цикле необходимо выполнить заполнение переменной для хранения UID данными, считанными с карты. Это необходимо выполнить с помощью конкатенации. Конкатенация - процесс «склеивания» строк. Следовательно, необходимо выполнить считывание данных с карты, преобразовать их в строковый тип, перевести их в десятичную систему счисления и выполнить «склейку». Конкатенация выполняется следующим образом: «имя переменной.concat()» Следовательно, описанный ранее цикл и процессы в нем будут выполняться следующим образом:

```
    for (byte i = 0; i < mfrc522.uid.size; i++)
    {
        if ((mfrc522.uid.uidByte[i] < 0x10)
        {
            Serial.print(" 0");
        }
        else
        {
            Serial.print(" ");
        }
        Serial.print(mfrc522.uid.uidByte[i], DEC);
        content.concat(String(… < 0x10 ? " 0" : " "));
        content.concat(String(…, DEC));
    }
```

Пожалуй, необходимо объяснить, что происходит в строке «`content.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));`». Начнем объяснение изнутри. В скобках описан процесс выбора байта из UID, и если он меньше «`0x10`», то присвоить значение 0 и присвоить ему строковый тип, если значение больше, что вывести это значение и присвоить ему строковый тип. Далее полученные строки «склеиваются» в одну при помощи конкатенации.

Далее необходимо выполнить перенос кареток для вывода информации через UART и перенос каретки для записи данных в переменную:

```
    Serial.println();
    content.toUpperCase();
```

Создайте переменную, которая внесет вариативность логики работы. Например, «`int x = 1;`». Далее необходимо создать цикл, в котором будет перебираться по 4 байта из массива с сохраненными UID как «свой» и сравниваться с считанным UID с карты. Таким образом, аргументы цикла будут выглядеть следующим образом:

```
for (int i = 0; i < sizeof(keys) / sizeof(char*); i++)
```

где «`sizeof(keys) / sizeof(char*)`» будет отвечать за количество сохраненных UID.

Подсчитывается так: если в массиве «`keys`» находится 8 байт данных, то «`sizeof`» вернет значение 8, а «`eof(char*)`» вернет значение 4. Таким образом, поделив 8 на 4 мы получим 2, что соответствует количеству UID, прописанных в массиве «`keys`». Значит, цикл будет иметь всего 2 итерации, поскольку в большем количестве нет необходимости выполнить проверку считанных 4 байт UID из карты и сравнить их с массивом «`keys`», выбирая оттуда по 4 байта данных. Для этого необходимо в аргументе условного оператора написать следующее: «`content.substring(1) == keys[i]`». Что же такое «`substring(1)`»? Данная функция получает подстроку из переменной типа «*String*». В нашем случае будет браться подстрока от первого элемента переменной «`content`» до конца этой переменной, поскольку не указан второй аргумент, который является необязательным и показывает, до какого элемента необходимо взять подстроку. Таким образом, мы берем все элементы массива «`content`» – 4 байта, и сравниваем их с каждыми 4-мя байтами из массива «`keys`». Выполняется логика работы самого замка. Важно, что после того, как выполнилась логика работы, необходимо прекратить работу цикла, ведь дальше перебирать UID нет смысла. В случае, если ни один из прописанных UID не совпал, то необходимо переменную «*x*» обнулить и выполнить логику работы программы, исходя из того, что приложенная карта неверна. Шаблон того, как должен выглядеть ваш код, реализующий ход действий, описанных выше, представлен ниже:

```
    int x = 1;
    for (int i = 0; i < sizeof(keys) / sizeof(char*); i++) {
        if (content.substring(1) == keys[i]) {
            …        // Сообщение о том, что доступ открыт
            …       // Включаем Замок
            …      // Включаем Зелёный светодиод
            …     // Красный светодиод выключен
            …    // Синий светодиод выключен
            …   // 5 секунд дверь открыта
            … // Гасим Зелёный светодиод
            …// Выключаем Замок
            return; // Выключаем Замок

        }
        else  x = 0;
    }
```

Код для карты, не обозначенной как «свой», представляет собой простой условный оператор с аргументом «`x == 0`», а в теле необходимо включить красный светодиод, вывести сообщение о том, что доступ запрещен, и снять питание с замка, раздать звуковой сигнал о несанкционированной попытке входа.

Когда ваш код будет готов, загрузите его на плату, проверьте работоспособность и продемонстрируйте работу преподавателю. Конечный код программы представлен ниже:

```
// карта - 73 36 43 229
// брелок - 202 168 186 128

#include <SPI.h>
#include <MFRC522.h>

#define SS_PIN  PB14       // SS вывод модуля RC522
#define RST_PIN  PB5       // RST вывод модуля RC522

#define BUZZER  PA8        //Пин для Пищалки
```

```
#define LED_R   PB9         //Пин для Красного цвета
#define LED_G   PB8         //Пин для Зелёного цвета
#define LED_B   PB1         //Пин для Синего цвета
#define LOCK   PA1          // подключаем электрозамок


char* keys[] = { "202 168 186 128","73 36 43 229" }; // ,"73 36 43 229"

MFRC522 mfrc522(SS_PIN, RST_PIN);   // Create MFRC522 instance.

void setup()
{
    Serial.begin(9600);        // Initiate a serial communication
    SPI.begin();               // Initiate  SPI bus
    mfrc522.PCD_Init();        // Initiate MFRC522
    pinMode(LED_G, OUTPUT);    //объявляем пин как выход.
    pinMode(LED_R, OUTPUT);    //объявляем пин как выход.
    pinMode(LED_B, OUTPUT);    //объявляем пин как выход.
    pinMode(BUZZER, OUTPUT);   //объявляем пин как выход.
    pinMode(LOCK, OUTPUT);     //объявляем пин как выход.
    noTone(BUZZER);
    analogWrite(LOCK, 0);  //Выключаем замок
    Serial.println("Поднесите карту");
}
void loop()
{
    //Выполняем проверку наличия карты и корректность ее работы
    if (!mfrc522.PICC_IsNewCardPresent()) {
        return;
    }
    if (!mfrc522.PICC_ReadCardSerial()) {
        return;
    }

    Serial.print("UID:");
    String content = "";
    for (byte i = 0; i < mfrc522.uid.size; i++)
    {
        Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
        Serial.print(mfrc522.uid.uidByte[i], DEC);
        content.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
        content.concat(String(mfrc522.uid.uidByte[i], DEC));
    }
    Serial.println();
    content.toUpperCase();

    int x = 1;
    for (int i = 0; i < sizeof(keys) / sizeof(char*); i++) {
        if (content.substring(1) == keys[i]) {
            Serial.println("Доступ разрешён");
            analogWrite(LOCK, 255);
            digitalWrite(LED_G, HIGH);
            digitalWrite(LED_R, LOW);
            digitalWrite(LED_B, LOW);
            delay(5000);
            digitalWrite(LED_G, LOW);
            analogWrite(LOCK, 0);
            return;
        }
        else  x = 0;
    }

    if (x == 0) {
        Serial.println("Доступ запрещён");
        digitalWrite(LED_R, HIGH);
```

```
        digitalWrite(LED_G, LOW);
        digitalWrite(LED_B, LOW);
        tone(BUZZER, 300);
        delay(1000);
        digitalWrite(LED_R, LOW);
    }   noTone(BUZZER);

}
```

Работа № 15

«Запись и чтение файлов с внешнего носителя»

Хранение данных – важная часть любой системы, осуществляющей контроль и мониторинг внутри себя или окружающей среды. Важность наличия записей в период работы системы обуславливается тем, что вывод информации в реальном времени – недостаточен и неисчерпывающий. Всегда необходимо хранить данные о событиях, ЧП (чрезвычайных происшествиях) и действиях персонала, чтобы иметь возможность в полной мере восстановить картину происходящего, выстраивать подробные графики изменения параметров системы и производить глубокий анализ происходящего.

Черные ящики, самописцы, резервные копии и т.д. – типы записи и хранения информации для различных сфер, устройств и назначений. Черные ящики применяются от самых глубинных проходческих комбайнов в горнодобывающей промышленности до орбитальной МКС.

Цель работы: Реализация системы чтения и записи файлов с внешнего носителя.

Задачи:

1) Реализовать систему чтения файлов с внешнего носителя;
2) Реализовать систему записи файлов на внешний носитель.

1. Вводная часть

1.1 Чтение типа карты

На плате TUSUR IoT Board для чтения и записи данных на внешний носитель используется модуль для носителей данных формата SD.

Для работы с данным модулем используется интерфейс SPI и библиотека «SD.h». Создайте новый скетч и подключите необходимые библиотеки.

Далее напишите следующие команды:

```
Sd2Card card; // Инициализируем работу SD
SdVolume volume;
SdFile root;
const int chipSelect =PB8; // Пин чип селекта
```

В «`void setup()`» инициализируйте работу интерфейса UART со скоростью 9600. Выведете сообщение о том, что выполняется инициализация SD карты, создайте условный оператор с полным ветвлением условием которого будет «`!card.init(SPI_HALF_SPEED, chipSelect)`», что проверит правильность подключения и если это условие выполняется, то не удалось подключиться к карте и выведите сообщение о неудаче. Иначе, выведете сообщение, о том, что карта инициализированная. Основа кода, описанных действий будет выглядеть следующим образом:

```
void setup() {
Serial.begin(…);
Serial.print("\nИнициализация SD карты");
  if(!card.init(SPI_HALF_SPEED, chipSelect)){
    Serial.println("…");
    return;
  }
```

```
        else{
          Serial.println("… /");
        }
```

Далее выводим сообщение «Тип карты» и реализуем конструкцию «switch, case». Эта конструкция новая, поэтому разберем ее немного подробнее. Данная конструкция является аналогом конструкции «if, else», позволяющая сократить строки кода, повысить скорость его работы и, в целом, сделать ваш код более продвинутым и читабельным. Конструкцию открывает команды «switch(){». В аргумент свитча мы можем передавать любую переменную. Нам потребуется передать туда функцию, которая будет возвращать тип карты. Данная функция, взятая из библиотеки, выглядит следующим образом «`card.type()`». Соответственно, открытие конструкции будет выглядеть так: «`switch(card.type()){`» Далее нам необходимо описать наши кейсы, которые являются аналогом «if». Для того, чтобы вам долго не разбираться в библиотеках и не искать типы карт, они приведены ниже:

Тип	Синтаксис
1	`SD_CARD_TYPE_SD1`
2	`SD_CARD_TYPE_SD2`
3	`SD_CARD_TYPE_SDHC`

Разберем синтаксис написания кейса на примере первого типа. Объявим, что сейчас начинается кейс: «`case`», далее укажем при каком условии будет выполняться данный кейс, и установим двоеточие, показываем тем самым, что дальше пойдет тело кейса. После выполнения кейса необходимо прервать работу конструкции «switch, case» командой «break». Код будет выглядеть так:

```
        case SD_CARD_TYPE_SD1:
        Serial.println("SD1");
        break;
```

По данному примеру опишите оставшиеся два кейса. Чаще всего, «свитч» заканчивается условием, что, если ни один из кейсов не был выполнен, выполняется «дефолтное» действие, например, вывод сообщения о том, что ни одно из условий в свитче не было выполнено. Но наличие «дефолтного» действия является необязательным, но желательным. После чего, «свитч» закрывается фигурной скобкой. Основа написания конструкции «switch, case» представлена ниже:

```
switch(card.type()){
   case SD_CARD_TYPE_SD1:
   Serial.println("SD1");
   break;
   case …:
   Serial.println("…");
   ...;
   case …:
   Serial.println("…");
   …;
   default:
   Serial.println("Такого типа не существует");
  }
```

Далее осуществляется проверка файловой системы (формата памяти карты). Для этого, воспользовавшись библиотечной функцией проверки типа файловой системы «volume.init(card)» создайте условие проверки типа файловой системы с прекращением работы программы и выводом сообщения, что формат памяти не соответствует.

```
if (!volume.init(card)) {
      // неверная файловая система
      Serial.println("Тип формата памяти неизвестен.");
      return;
   }
```

Далее необходимо считать тип памяти и вычислить размер раздела памяти. Тип может быть «FAT16» или «FAT32». Для того, чтобы считать тип памяти необходимо воспользоваться библиотечной командой «volume.fatType()». Для этого выведете 3 сообщения: первое – («Тип файловой системы FAT»), второе – («volume.fatType()»,DEC) и третье сообщение пустое, т.е. – (). Далее создайте переменную тип данных которой «uint32_t» в которую будет записываться считанные параметры памяти. Первый шаг, это приравнять переменную значению, возвращаемому функцией «volume.blocksPerCluster()», что вернет количество блоков, находящихся в одном кластере памяти. Второй шаг, полученное значение необходимо умножить на значение, возвращаемое функцией «volume.clusterCount()», что вернет количество кластеров. Третий шаг рассчета объема памяти в байтах, это умножить получившийся результат на 512 – столько байт находится в одном блоке. Далее необходимо вывести в монитор порта сообщение об объеме памяти, вмещающей в себя карты. Здесь можно вывести только объем в байтах или Мб. И в завершении необходимо вывести информацию о имеющихся файлах на карте их типе, дате и размере. Для этого выведете сообщение, о том, какая информация будет выведена ниже, напишите команду, которая «откроет» карту – «root.openRoot(volume)», а затем команду, которая прочтет и выведет всю необходимую информацию в монитор порта – «root.ls(LS_R | LS_DATE | LS_SIZE)». На этом этапе заканчивается «void setup()», а «void loop(void)» остается пустым. Основа описанного кода представлена ниже:

```
// считываем тип и вычисляем размер первого раздела
   uint32_t volumesize;
   Serial.print("\nVolume type is FAT");
   Serial.println(volume.fatType(), DEC);
   Serial.println();

   volumesize = volume.blocksPerCluster(); // блоков на кластер
   volumesize *= volume.clusterCount(); // кластеров
   volumesize *= 512; // 512 байтов в блоке, итого байт..
   Serial.print("Volume size (bytes): ");
   Serial.println(volumesize);
   Serial.print("Volume size (Kbytes): ");
   volumesize /= 1024;
   Serial.println(volumesize);
   Serial.print("Volume size (Mbytes): ");
   volumesize /= 1024;
   Serial.println(volumesize);

   Serial.println("\nFiles found on the card (name, date and size in bytes): ");
```

```
   root.openRoot(volume);
   // выводим список файлов
   root.ls(LS_R | LS_DATE | LS_SIZE);
}
void loop(void) {
}
```

Итоговый код программы представлен ниже:

```
#include<SPI.h>
#include<SD.h>

Sd2Card card; // Инициализируем работу SD
SdVolume volume;
SdFile root;
const int chipSelect =PB8; // Пин чип селекта

void setup() {
  // put your setup code here, to run once:
Serial.begin(9600);
Serial.print("\nInitialization SD card..");
  if(!card.init(SPI_HALF_SPEED, chipSelect)){
    Serial.println("Initialization failed");
    return;
  }
  else{
    Serial.println("Wiring is correct and a card is present/");
  }
  Serial.print("\nCard type");
  switch(card.type()){
    case SD_CARD_TYPE_SD1:
    Serial.println("SD1");
    break;
    case SD_CARD_TYPE_SD2:
    Serial.println("SD2");
    break;
    case SD_CARD_TYPE_SDHC:
    Serial.println("SDHC");
    break;
    default:
    Serial.println("Unknown");
  }
 if (!volume.init(card)) {
        // неверная файловая система
        Serial.println("Could not find FAT16/FAT32 partition.");
        return;
    }

    // считываем тип и вычисляем размер первого раздела
    uint32_t volumesize;
    Serial.print("\nVolume type is FAT");
```

```
  Serial.println(volume.fatType(), DEC);
  Serial.println();

  volumesize = volume.blocksPerCluster(); // блоков на кластер
  volumesize *= volume.clusterCount(); // кластеров
  volumesize *= 512; // 512 байтов в блоке, итого байт..
  Serial.print("Volume size (bytes): ");
  Serial.println(volumesize);
  Serial.print("Volume size (Kbytes): ");
  volumesize /= 1024;
  Serial.println(volumesize);
  Serial.print("Volume size (Mbytes): ");
  volumesize /= 1024;
  Serial.println(volumesize);

  Serial.println("\nFiles found on the card (name, date and size in bytes): ");
  root.openRoot(volume);
  // выводим список файлов
  root.ls(LS_R | LS_DATE | LS_SIZE);
}

void loop(void) {
}
```

1.2 Чтение данных с карты

Реализация кода для считывания информации с SD карты довольна простая. Выполните подключение необходимых библиотек, как вы это делали ранее, а также укажите пин, который отвечает за чип-селект.

В «void setup()» выполните инициализацию работы UART на скорости 9600. Выполните проверку подключения к SD карте, использовав условный оператор и команду «!SD.begin(chipSelect)». Данная команда проверит возможность подключения к карте по данному адресу пина чип-селекта, и, если подключиться не удастся, то выведите сообщение о ошибке инициализации, после чего прервите работу программы. Данный участок кода будет выглядеть следующим образом:

```
#include <SPI.h>
#include <SD.h>
const int chipSelect = …;
void setup() {
  Serial.begin(…);

 if( !SD.begin( chipSelect )){
    Serial.println("…");
    return;
  }
```

После того, как была выполнена проверка подключения к SD карте необходимо открыть файл, который мы собираемся прочесть. Для этого необходимо воспользоваться библиотечной командой «File myFile = SD.open("Имя файла.txt")», что создаст переменную с именем «myFile» через которую будет продолжаться дальнейшая работа. Далее создайте условие, что при успешном создании переменной «myFule» запустится цикл «while» с условием инициализации данных в данной переменной «myFile.available()» и в данном цикле необходимо выполнить запись в com-порт, считанных байт информации из файла «myFile.read()» и после того, как файл был прочитан полностью, его необходимо закрыть командой «myFile.close()». Основа описанного кода будет выглядеть следующим образом:

```
// открываем файл для чтения
File myFile = SD.open("Имя файла.txt");
if (myFile) {
  // считываем все байты из файла и выводим их в COM-порт
  while (…()) {
    Serial.write(…());
  }
  // закрываем файл
  myFile.close();
}
```

Далее реализуйте полное ветвления, что если не удалось открыть файл, то выводится сообщение о ошибке открытия файла, а «void loop()» остается пустым:

```
else {
    // выводим ошибку если не удалось открыть файл
    Serial.println("Ошибка открытия файла Имя файла.txt");
  }
}
void loop() {
}
```

Так как на карте на данный момент нет ни одного файла с данными в расширении «.txt», то данный код не сработает, но он нам потребуется для дальнейшей работы, когда вы запишите на карту свой первый файл с данными.

2. Основная часть

2.1 Создание кода для записи данных на SD карту

Для удобства и скорости написания кода с записью данных на SD карту, воспользуемся кодом, который был написан для работы автоматического управления климатом.

Откройте данный код и скопируйте его в новый скетч. В начале кода добавьте подключение библиотеки для работы с SD картой, введите переменную «chipSelect» и присвойте ей значение пина «PB8». В конец «void setup()» добавьте следующий код:

```
if (!SD.begin(chipSelect)) {
Serial.println("Card failed, or not present");
return;
}
```

В конец кода добавьте следующий код:

```
File dataFile = SD.open("test.txt", FILE_WRITE);
   if (dataFile) {
      // записываем строку в файл
      String temperatura;
      temperatura += "Temp= ";
      temperatura += temp;
       dataFile.println(temperatura);
       dataFile.close();
       Serial.println("Success!");
   } else {
      // выводим ошибку если не удалось открыть файл
       Serial.println("Не удалось открыть файл.");
   }
```

Данный код создает файл с именем «test.txt» и записывает в него информацию о текущей температуре, которая будет накапливаться с течением времени работы данной программы.

Загрузите код на плату и запустите, подождите некоторое время, пока ваша программа запишет объем информации о текущей температуры, при выключенном вентиляторе, включенном и критической температурой.

Затем загрузите код, для считывания информации с файла. Считайте информацию, через COM-порт. Продемонстрируйте работу ваших программ.

Дополнительное задание! Добавьте в информацию, что записывается в файл? счетчик времени, чтобы понимать в какой момент времени случилась та или иная ситуация.

Работа № 16

«Беспроводные модули интернета вещей. часть 1»

Передача данных с использованием IR (Infrared) приемопередатчика, является одной из простейших систем передачи данных беспроводным путем. Обмен данными осуществляется при помощи световых волн инфракрасного диапазона. Из-за своей простоты такая передача данных получила широкое распространение: телевизоры, музыкальные центры, DVD-проигрыватели, кондиционеры, телефоны и ноутбуки. В последних двух устройствах данный вид передачи данных, в современном мире, был вытеснен современными Bluetooth и Wi-Fi. Это связано с тем, что IR порт обладает рядом недостатков:

1. Передача данных возможна только на близкие расстояния и в прямой видимости луча;
2. Флуоресцентные лампы и яркий солнечный свет вносят значительные помехи в канал вплоть до прекращения передачи данных;
3. Низкий потенциал скорости передачи информации;
4. Усложнение сборки устройств из-за проводящего окна.

Однако, IR порты до сих пор широко применяются в пультах дистанционного управления (ПДУ), поскольку для этого не требуется передавать большое количество информации.

Цель работы: изучить инфракрасный приемопередатчик и реализовать программу с управлением пульта ДУ.

Задачи:

1) Изучить теоретический материал;
2) Реализовать программу IR передатчика;
3) Реализовать программу IR приемника;
4) Реализовать программу с дистанционного управления.

1. Теоретическая часть

1.1 Историческая справка

Пульт дистанционного управления (ПДУ) был изобретен 29 мая 1964 года Робертом Адлером. В 1979 году компания «Hewlett-Packard» объявила о старте продаж калькулятора с IR для вывода информации на печать. В течении следующий нескольких лет было разработано не мало устройств с передачей информации по открытому оптическому каналу IR, однако данные устройства не получили широкого распространения из-за несовместимости между собой. Этот факт стал причиной основания Infrared Data Association (IrDA) – международная некоммерческая организация, цель которой: разработка единых стандартов инфракрасных линий передачи информации. Результатом работы IrDA стал первый стандарт под названием Serial Infrared standard (SIR), обеспечивающий скорость передачи информации составляла 115,2 кбит/с.

В 1995 году IrDA была выпущена вторая версия стандарта IrDA 2.0, обратно совместимая с первой версией, увеличив скорость передачи данных с 115,2 кбит/с до 1,142-4 Мбит/с с длинной волны от 850 до 880 нанометров и дистанцией связи до 1 метра.

Для ПДУ нет необходимости в такой скорости передачи данных. Поэтому устройства для дистанционного управления устройствами развивались параллельно.

Типы ПДУ различаются по ряду характеристик:

1) Питание:
   a. Автономное;
   b. Проводное.

2) Мобильности:
   a. Стационарный;
   b. Носимый.
3) Функциональности:
   a. С фиксированным набором команд;
   b. С переключаемым набором команд (универсальный);
   c. С обучением набору команд (обучаемый).
4) Каналу связи:
   a. Механический;
   b. Проводной;
   c. Радиоканал;
   d. Ультразвуковой;
   e. Инфракрасный.

1.2 Протокол NEC

В инфракрасном приемопередатчике, применяемом в данной работе, используется протокол «кодирования длиной паузы» фирмы NEC. Протокол NEC работает таким образом, что начало каждого бита определяется импульсом длиной 560 мкс и, он же является концом предыдущего бита. Длина следующей за данным импульсом паузы и определяет логическое значение бита. В конце передачи данных так же отправляется импульс длинной 560 мкс., означающий конец передачи данных. Внешний вид сигнала логического 0 и логической 1, имеет следующий вид:

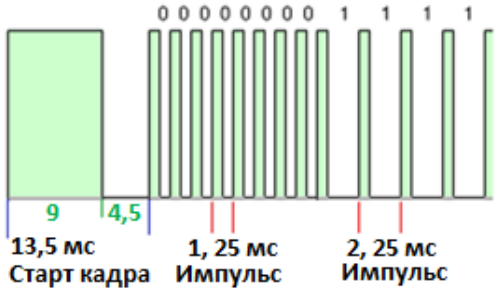


Рисунок 1.1 – Внешний вид логических уровней протокола NEC

Команда по каналу отправляется цельным пакетом, размер которого составляет 32 бита (4 байта). Старт передачи пакета характеризуется импульсом длинной 9 мс. и паузой длинной в 4,5 мс., затем следует пакет с данными. Пакет данных содержит в себе адрес устройства и команду. Каждый байт пакета передается младшим битом вперед.

Существует две версии протокола NEC: стандартная и расширенная. Структура пакета в стандартной версии протокола состоит из 8-ми битного адреса, 8-ми битного инвертированного адреса, 8-ми битной команды и 8-ми битной инвертированной команды. Структура посылки, которая отправляется протоколом NEC при нажатии кнопки ПДУ будет иметь следующий вид:

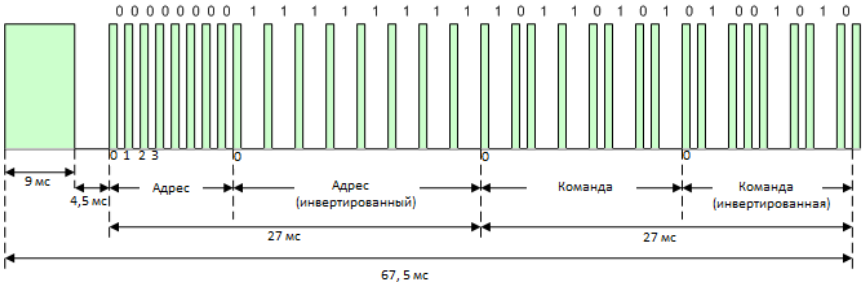


Рисунок 1.2 – Внешний вид пакета стандартной версии протокола NEC

В данной версии протокола пакет всегда имеет одинаковую длительность, поскольку команда и адрес передаются как в прямом, так и в инверсном виде. В качестве несущей частоты для сигнала используется частота в 38 кГц.

Важно знать, что преемники зачастую подтянуты на высокий уровень, соответственно, когда передающее устройство, например, ПДУ ничего не передает, то на принимающей стороне установлен высокий уровень сигнала. Когда с ПДУ передается сигнал, что на стороне приемника данный сигнал будет инвертирован.

2. Практическая часть

2.1 Написание кода передатчика

ИК приемопередатчик, как исходит его названия, включает в себя приемник и передатчик, как любая система беспроводной связи. Для начала, на плате TUSUR IoT Board необходимо реализовать передатчик. Схемотехнически уже все реализовано, необходимо только написать программу передатчика.

Для работы будет необходима библиотека «IRremote.h». Для этого перейдите во вкладку инструменты и через управление библиотеками загрузите необходимую библиотеку в Arduino IDE.

Создайте новый скетч и подключите библиотеку. Далее создайте константу, которая будет хранить в себе информационный кадр сигнала. Используйте тип переменной «long», а в переменную поместите данные в шестнадцатеричной системе счисления и размером 4 байта, например, «0x10AA20DF».

Инициализируйте работу IR передатчика, воспользовавшись библиотечной функцией «IRsend irsend()», где в скобках укажите пин, к которому подключен передатчик – PB9. Данная функция определяет тип работы девайса, подключенного устройства. В данном случае, это передатчик. Далее воспользуйтесь еще одной библиотечной функцией «decode_results results» которая создает объект «results», необходимый для работы IR канала.

```
#include <...>
const long Heal = 0x10AA20DF;

IRsend irsend(...);

decode_results ...;
```

Затем, необходимо перейти к написанию «`void setup()`» в теле которого инициализируйте пин светодиода на выход и инициализируйте работу интерфейса UART со скоростью 115200.

```
void setup() {
	pinMode(PC13,...);
	Serial.begin(...);
}
```

Далее в «void loop()» необходимо включить светодиод. Передать посылку, используя библиотечную команду «irsend.sendNEC(Heal, sizeof(Heal) * 8);» здесь указывается протокол передачи «sendNEC» в скобках передается информация о посылке и длительности посылки. Затем выключаем светодиод и устанавливаем задержку в 5 секунд.

```
void loop() {
	digitalWrite(..., ...);
	irsend.sendNEC(Heal, sizeof(Heal) * 8);
```

```
        digitalWrite(..., ...);
        delay(5000);
}
```

Допишите все фрагменты кода, заполнив пропущенные места, и загрузите его на плату. Индикатором, что передатчик отправляет команду будет выступать моргающий раз в 5 секунд зеленый светодиод. Однако, необходимо убедиться, правильно ли приемник интерпретирует переданную на него команду. Для этого в разделе главного меню платы TUSUR IoT Board необходимо выбрать пункт «ИК-приемопередатчик», а в нем подпункт «ИК-приемник». На рисунке 2.1 представлена графическая инструкция, описанных действий.

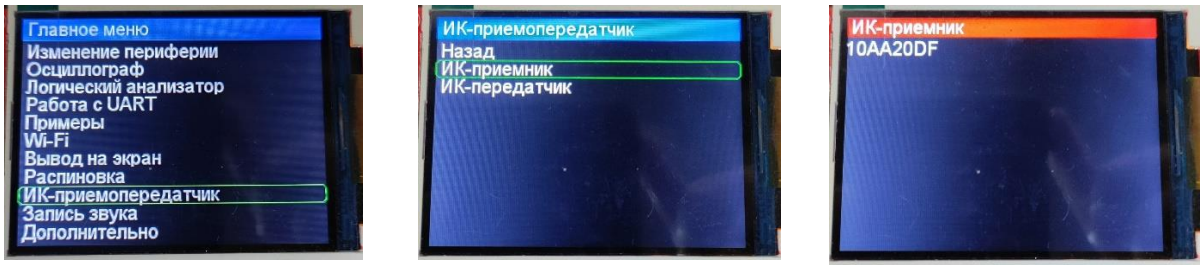


Рисунок 2.1 – Путь к выводу команды с приемника

В открывшемся окне мы сможете пронаблюдать какую команду считывает приемник. Необходимо убедиться, что команда, отправляемая вами через передатчика, верно интерпретируется приемником. Продемонстрируйте свою работу.

2.2 Написание кода приемника

Написание кода для приемника во многом схоже с написанием кода для передатчика. Главными отличиями являются функции, которыми вы будете пользоваться.

Также, как в случае с передатчиком, подключите библиотеку, вызовите функцию «IRrecv irrecv(…)». Воспользуйтесь пунктом «Распиновка» на плате и определите пин, к которому подключен приемник и укажите его в функции. Далее воспользуйтесь еще одной библиотечной функцией «decode_results results» которая создает объект «results», необходимый для работы IR канала.

В «`void setup()`» инициализируйте пин светодиода и пин PB9 на выход. Инициализируйте работу UART на скорости 115200, запустите прием инфракрасного сигнала при помощи команды «`irrecv.enableIRIn()`» и включите светодиод, сигнализирующий о выполненной передаче посылки, командой «`irrecv.blink13(true)`», чтобы видеть, что передача посылок осуществляется. Основа кода будет выглядеть следующим образом:

```
#include <...>

IRrecv irrecv(...);
decode_results results;

void setup() {
    pinMode(..., ...);
    pinMode(..., ...);
    Serial.begin(...);
    irrecv.enableIRIn();
    irrecv.blink13(true);
}
```

Далее в «`void loop()`» необходимо принудительно установить значение на пине PB9 в низкий уровень. Это необходимо для корректности работы. Если вы помните, то в теоретическом разделе мы говорили о том, что на приемнике изначально установлен высокий уровень, что приводит к инверсии входной команды. Чтобы избежать этого, вам необходимо принудительно установить низкий уровень на стороне приемника.

Создайте условие, аргументом которого будет проверка наличия посылки на стороне приемника. За это отвечает библиотечная функция «`irrecv.decode(&results)`» которая возвращает статус сигнала на приемнике. В теле условия выведете в монитор порта сообщение о том, что была принята команда, затем необходимо вывести саму принятую команду. Команду возвращает библиотечная функция «`results.value`», которая возвращает принятый сигнал, о чем и говорит само название функции. Важно, что, когда вы будете выводить принятый сигнал в монитор порта, необходимо указать, что его нужно выводить в шестнадцатеричной системе счисления. Затем, воспользуйтесь библиотечной функцией «`irrecv.resume()`», которая позволит далее опрашивать приемник на наличие сигнала на входе. Основа описанного кода будет выглядеть следующим образом:

```
void loop() {
    digitalWrite(..., ...);
    if (irrecv.decode(&results)) {
        Serial.println("Принятая команда:");
        Serial.println(results.value, HEX);
        irrecv.resume();
    }
}
```

Допишите все участки кода, соедините их в единое целое и загрузите на плату.

Как вы могли заметить, в вашем коде нет никакой последовательности, отвечающей за передаваемую в приемник команду. Для того, чтобы осуществить передачу команды, для ее приема, необходимо воспользоваться функционалом платы TUSUR IoT Board. Для этого перейдите в раздел меню «ИК-приемопередатчик», в нем выберете «ИК-передатчик». После выбора передатчика у вас откроется окно с настройками команды, которую вы будете отправлять, и выглядит окно следующим образом:

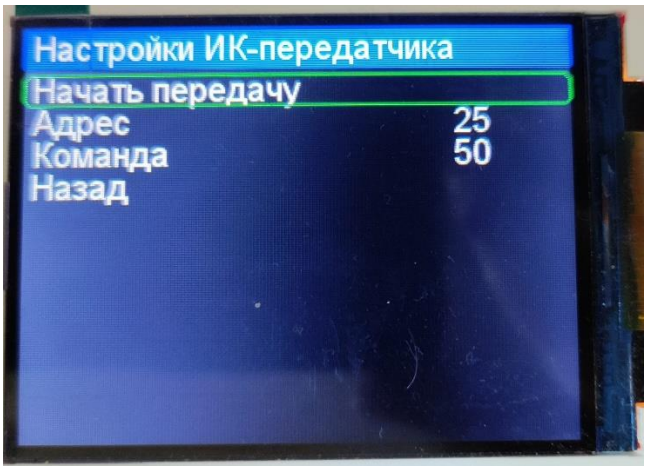


Рисунок 2.2 – Окно настройки пакета передачи

Как видно из рисунка 2.2 у вас есть возможность настроить адрес и команду. Вспомнив стандарт NEC вы легко можете понять за что отвечают данные настройки. После того, как вы выберете интересующие вас параметры выберете пункт «Начать передачу». Начнется передача выбранной вами посылки и откроется окно с графическим представлением кадра, как это показано на рисунке 2.3.

Рисунок 2.3 – Графическое представление кадра стандарта NEC

Откройте монитор порта в Arduino IDE и установите скорость работы в 115200. В монитор порта должна начать выводиться информация о принятом пакете в шестнадцатеричной системе. Проверьте правильность выводимой команды. Важно! На этапе настройки пакета номер канала и команды указываются в десятичной системе счисления, а вывод в монитор порта – шестнадцатеричный.

3. Создание вентилирующей системы с управлением по ИК каналу

3.1 Реализация команд включения и выключения вентиляции

В данном разделе будет реализована система вентиляции с управлением ее включения и выключения по ИК каналу. Как вы понимаете, для этого необходимо задействовать приемник на плате TUSUR IoT Board и сторонний ПДУ.

Для начала, как и в предыдущем примере, представленном в разделе 2.2, где вами создавался приемник, необходимо подключить библиотеку, инициализировать работу ИК приемника и объявить несколько констант, хранящих в себе несколько команд с пульта. Также необходимо объявить переменную счетчика, переменную, в которую будет записываться считанная команда и переменную, хранящую в себе уровень ШИМ сигнала.

Для начала нам потребуется всего одна константа с командой на включение, переменная счетчика и переменная с ШИМ сигналом, которая будет хранить фиксированную скорость вращения вентилятора. Остальные объявленные переменные и константы будут нужны для второй части работы, поэтому можете создать их сейчас или далее. Скелет описанного кода будет выглядеть следующим образом:

```
#include <...> // подключаем библиотеку для IR приемника

IRrecv irrecv(...); // указываем пин, к которому подключен IR приемник
decode_results results;
int c = 0; // Создаем счетчик
const uint32_t ON_OFF = ...; // Создаем константу для вкл/откл
const uint32_t PLUS_SPEED = ...; // Создаем константу для увеличения скорости
const uint32_t MINUS_SPEED = ...; // Создаем константу для снижения скорости
uint64_t f; // Создаем переменную в которую записывается считанная команда с
приемника
```

```
uint16_t speed = от 60 до 255; // Создаем переменную с любым ШИМ сигналом для вентилятора
```

Далее в «`void setup()`» необходимо запустить прием инфракрасного сигнала, как это было сделано в разделе 2.2. Инициализируйте работу UART со скоростью 115200 и инициализируйте пины светодиода и вентилятора на выход, а пин приемника ИК сигнала, на вход. Скелет описанного кода будет выглядеть следующим образом:

```
void setup() {
    irrecv.enableIRIn();  // запускаем прием инфракрасного сигнала
    Serial.begin(...); // Инициализируем работу UART

    pinMode(..., ...); // пин PC13 будет выходом
    pinMode(..., ...); // пин PB0 будет входом
    pinMode(..., ...); // пин PA1 будет выходом

}
```

Далее в «`void loop()`» создайте условие наличия сигнала на приемнике, как вы это делали в разделе 2.2. Далее, уже в теле данного условия будет писаться весь код работы вашей программы. Первым делом, когда ваша программа заходит в условие наличия на приемнике сигнала, то она должна вывести в монитор порта, принятый сигнал. Для этого воспользуйтесь уже известной вам «`Serial.println(results.value, HEX)`». Выведите 3 команды с пульта, которыми хотите оперировать в работе и впишите их в ваши константы.

Теперь, когда вы определили, какими кнопками вы будете манипулировать, создайте условие проверки нажатия кнопки включения, в теле которого необходимо включить светодиод и запустить вентилятор.

```
void loop() {

    if (irrecv.decode(&results)) // если данные пришли выполняем команды
    {
        Serial.println(results.value);
        if (results.value == ON_OFF) {
            digitalWrite(..., ...); // включаем светодиод
            analogWrite(..., ...); // включаем вентилятор

        }
        irrecv.resume(); // принимаем следующий сигнал на ИК приемнике
    }
}
```

Допишите все участки кода, описанные выше, соедините их и загрузите на плату. Продемонстрируйте свою работу.

Как вы можете заметить, на данном этапе реализации программы, все заканчивается только на включении, а выключить вы ничего не можете. Для этого есть два пути решения: создать отдельную кнопку для выключения или реализовать программный счетчик количества нажатий, по которому будет срабатывать выключение. Чаще всего в пультах ДУ используется именно второй подход. Зачем создавать дополнительную команду на отключение, если это повлечет дополнительное нагромождение кнопок ПДУ, да и решается программно.

Если вы помните, то в самом начале вы создавали счетчик. Воспользуемся им для того, чтобы считывать количество нажатий на кнопку включения и реализации программного отключения устройства. Для этого в условии нажатия на кнопку напишите инкремент вашего счетчика. Далее создайте условие того, что если ваш счетчик равен 1, то необходимо включить светодиод и включить вентилятор. Затем укажите второе условие, что если счетчик равен 2, то необходимо выключить светодиод, выключить вентилятор и обнулить счетчик.

```
void loop() {

    if (irrecv.decode(&results)) // если данные пришли выполняем команды
    {
        Serial.println(results.value);
          // включаем и выключаем светодиод, в зависимости от полученного
сигнала
        if (results.value == ON_OFF) {
            ...; //Инкремент счетчика

        }
        if (c ... ...) { //Если первый раз нажали, то
            digitalWrite(..., ...); //Включаем вентилятор
            analogWrite(..., ...); //Включаем светодиод
        }

        if (c ... ...) { //Если второй раз нажали, то
            digitalWrite(...,...);//Выключаем светодиод
            analogWrite(...,...);//Выключаем вентилятор
            c = ...; //Обнуляем счетчик
        }
        irrecv.resume(); // принимаем следующий сигнал на ИК приемнике
    }
}
```

Допишите данный код и загрузите его на плату. Проверьте работоспособность и продемонстрируйте ее.

3.2 Управление скоростью вращения вентилятора при помощи ПДУ

В предыдущем разделе вы реализовали код, при помощи которого управляли включением и выключением вентилятора при помощи ПДУ. Однако, скорость его вращения всегда была константой, которую вы заложили в самом начале. Но, что, если эта скорость будет не оптимальной в том или ином случае? Для этого необходимо реализовать управление скоростью вращения вентилятора при помощи ПДУ.

После условия включения необходимо создать еще одно условие, аргументами которого будут выступать проверки принятой команды с приемника с двумя вашими массивами с логическим оператором или. Т.е. команда, пришедшая на приемник, совпадает либо с одним массивом, либо с другим. В теле условия присвойте переменной, созданной для хранения считанной команды с приемника, считанную команду. Выведете в монитор порта считанную команду. Установите задержку в 50мс. Приравняйте вашу переменную скорости результату, возвращаемому функцией управления скорости, в которую передайте считанную команду, выглядеть это должно, например, так: «`speed = SpeedFunction(f)`». Здесь f – хранит в себе команду с приемника, а «`SpeedFunction()`» – имя функции (у вас оно может быть другим). Далее выведете в монитор порта текущую скорость вращения вентилятора.

Основа описанного выше кода будет выглядеть следующим образом:

```
if (results.value ... ... || results.value ... ...) {
```

```
            f = ...;
            Serial.println(...);
            delay(...);
            speed = SpeedFunction(...);
            Serial.println(...);
        }
```

Теперь, начинается самая интересная часть данной работы – создание функции управления скоростью вращения вентилятора. Переместитесь в начало кода и перед «`void setup()`» объявите функцию. Как вы помните, функции должны быть сначала объявлены, а потом вызваны.

Помните, что данная функция возвращает значение, а значит тип функции должен быть отличным от типа «`void`». Подумайте, какой тип возвращаемых значений должен быть у функции. Ваша функция должна принимать в себя команду, считанную с приемника и выполнять ряд действий:

1) При условии, что пришла команда на увеличение скорости вращения, необходимо к уже имеющейся скорости прибавить число – шаг с которым вы хотите увеличивать скорость. Также, условие ограничения ШИМ сигнала. Как вы помните, ШИМ сигнал не может быть выше 255. Возврат скорости.

2) При условии, что пришла команда на уменьшение скорости вращения, необходимо от уже имеющейся скорости отнять число – шаг с которым вы хотите уменьшать скорость. Также, условие ограничения ШИМ сигнала. В данном случае, не давать ему уменьшатся ниже 60 (при меньшем значении вентилятор просто не будет крутиться). Возврат скорости.

Основа функции, которая описана ранее будет выглядеть следующим образом:

```
... SpeedFunction(... ...) {
    if (f ... ...) {
        speed = speed + ...;
        if (...) {
            speed = ...;}
        return ...;}
    if (f ... ...) {
        speed = speed - ...;
        if (...) {
            speed = ...;}
        return ...;}
}
```

Допишите код, загрузите его на плату и продемонстрируйте его работу. Если все работает правильно, что при первом нажатии на кнопку включения система должна запуститься, при нажатии на кнопки «+» и «–» увеличиваться или уменьшаться скорость вращения, при втором нажатии на кнопку включения система должна выключиться и сбросить настройки счетчика и скорости к изначально заданным.

Конечный код программы, представлен ниже:

```
#include <IRremote.h> // подключаем библиотеку для IR приемника

IRrecv irrecv(PB0); // указываем пин, к которому подключен IR приемник
decode_results results;
int c = 0;
const uint32_t ON_OFF = 0x10E7C23D;
const uint32_t PLUS_SPEED = 0x10E78877;
const uint32_t MINUS_SPEED = 0x10E7B847;
uint32_t f;
uint16_t aspeed = 150;
uint16_t SpeedFunction(uint32_t f) {
    if (f == PLUS_SPEED) {
```

```
        //Serial.println(aspeed);

        aspeed = aspeed + 10;
        //Serial.println(aspeed);
        if (aspeed > 250) {
            aspeed = 250;
        }
        return aspeed;
    }
    if (f == MINUS_SPEED) {
        aspeed = aspeed - 10;
        // Serial.println(aspeed);
        if (aspeed < 60) {
            aspeed = 60;
        }
        return aspeed;
    }
}
void setup() {
    irrecv.enableIRIn();  // запускаем прием инфракрасного сигнала
    Serial.begin(115200); // подключаем монитор порта

    pinMode(PC13, OUTPUT); // пин PC13 будет выходом
    pinMode(PB0, INPUT); // пин PB0 будет входом
    pinMode(PA1, OUTPUT); // пин PA1 будет выходом

}
void loop() {

    if (irrecv.decode(&results)) // если данные пришли выполняем команды
    {
        if (results.value == ON_OFF) {
            digitalWrite(PC13, HIGH);
            c++;

        }
        if (results.value == PLUS_SPEED || results.value == MINUS_SPEED) {
            f = results.value;
            Serial.println(f, HEX);
            delay(50);
            aspeed = SpeedFunction(f);
            Serial.println(aspeed);
        }

        if (c == 1) {

            analogWrite(PA1, aspeed);
            //irrecv.resume();
        }

        if (c == 2) {
            digitalWrite(PC13, LOW);
            analogWrite(PA1, 0);
            c = 0;
            aspeed = 150;
        }
        irrecv.resume(); // принимаем следующий сигнал на ИК приемнике
    }
}
```

Дополнительное задание.

Подумайте, что можно улучшить в данном коде или как его можно усложнить с точки зрения увеличения функционала.

Работа № 17

«Беспроводные модули интернета вещей. часть 2»

В предыдущей работе была рассмотрена беспроводная система передачи данных по ИК каналу. Вы можете помнить, что дистанция передачи информации по ИК каналу ограничена несколькими метрами, а то и непосредственной близостью, а также прямой видимостью между приемником и передатчиком, низкой скоростью передачи. Для выполнения каких-либо простейших задач такое решение может подойти, но как только речь заходит о более сложных системах с применением сложных датчиков (датчиков температуры, влажности, давления), приходится искать новые пути решения передачи информации беспроводным способом.

Отличным решением является радиосвязь. Одним из устройств, позволяющих реализовать радиоканал между двумя и более платами – модуль NRF, в частности, модуль NRF2104, использующейся на плате TUSUR IoT Board. Данный модуль работает на частотах диапазона 2,4ГГц, использует GFSK модуляцию, обладает 126 каналами (возможно объединить до 126 устройств). Также модуль поддерживает несколько скоростей передачи данных: 250 кбит, 1 Мбит и 2 Мбит и обладает программируемой мощностью передатчика и настраиваемым усилителем приемника. Сам модуль к плате подключен по интерфейсу SPI и обладает трехуровневыми буферами по 32 байта для приема и передачи.

В данной работе будет реализован программный код передатчика информации по радиоканалу, а также программный код приемника информации по радиоканалу.

Цель работы: изучить принцип работы радиомодуля NRF.

Задачи:

- Изучить теоретический материал;
- Реализовать программу NRF передатчика.

1. Теоретическая часть

1.1 Интернет Вещей

Интернет вещей (IoT) – понятие, которое описывает взаимодействия множества систем, которые обмениваются данными с друг другом. Точно также, как и люди обмениваются данными с друг другом в сети, устройства могут делится своей информацией для выстраивания «умных» систем, которые позволяют существенно облегчить жизнь человека.

Стандартными примерами систем интернета вещей являются «умные» системы.

Система «умного» полива огорода, система «умный» дом. Возможно, вы уже встречались с такими системами.

Основным отличием таких систем является автономность и полная автоматизация, так как данные системы способны работать без вмешательства человека лишь изредка принимая от него команды, для изменения параметров и состояния системы.

Как правило данные системы состоят из нескольких датчиков и актуаторов.

С датчиками мы знакомы.

Датчики – устройства, которые преобразуют некую физическую характеристику в напряжение для её анализа.

Актуатор – устройство, которое преобразует напряжение в некоторый сигнал или физическую силу для взаимодействия с окружением.

Датчики и актуаторы подключены к контроллерам. Где в свою очередь контролеры обмениваются данными с центральным звеном, который обрабатывает все данные и отправляет команды для изменения состояния системы другим контроллерам.

Между контроллерами может быть два вида связи:

– проводная;

– беспроводная.

В данном курсе уже было изучено, как информация передаются по проводам используя такие интерфейсы, как:

– UART;

– I2C;

– SPI.

Существуют и другие интерфейсы для передачи информации по проводам, но основное принцип остается тем же. Между двумя устройствами должно быть устойчивое соединение проводов для организации связи.

В беспроводной связи устройства обмениваются между собой с помощью радиосигнала в том или ином виде. Разные виды связи (сотовая, WiFi, Bluetooth, радио) используют разные принципы построения радиосигнала и его передачи в эфире.

1.2 Введение в беспроводную связь

Для передачи данных в беспроводных системах используется радиосигнал. Но что из себя представляет радиосигнал? Часто мы можем увидеть, что радиосигнал изображается в виде синусоиды (рисунок 1.1):

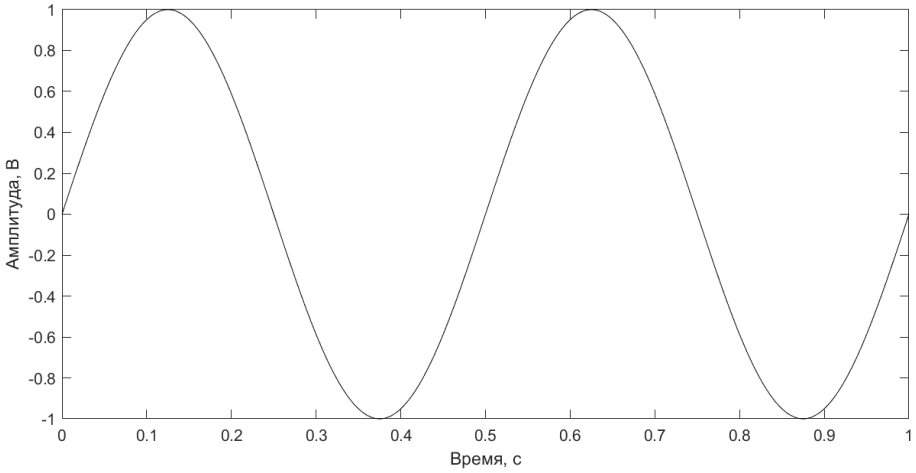


Рисунок 1.1 – Синусоида

Сигнал, которые выстраивается по закону синуса или косинуса – называется гармоническим и записывается по закону синуса или косинуса:

$$A \cdot \sin(2 \cdot \pi \cdot t \cdot f + \varphi_o),$$

где A – Амплитуда сигнала;

f – Частота сигнала;

φ_o – Фаза сигнала.

Попробуйте проверить свои знания и скажите какими параметрами обладает сигнал на рисунке 1.1.

Но сам по себе гармонический сигнал не передаёт какой-либо информации. Для передачи информации в сигнале меняют в течении времени, какой-либо из его параметров (фаза, частота, амплитуда).

Процесс изменения этих составляющих в ходе передачи информации, называется модуляцией, следовательно, модуляция может быть:

– Амплитудной;

– Фазовой;
– Частотной;
– И комбинацией модуляций.

Но слово модуляция применимо, в основном, только к аналоговым сигналам (голос, записанный с микрофона, данные с аналоговых датчиков).

Для примера давайте попробуем выполнить модуляцию над сигналом, используя данные 3 вида модуляции и с помощью синусоиды передать значение 010110 (рисунок 1.2):

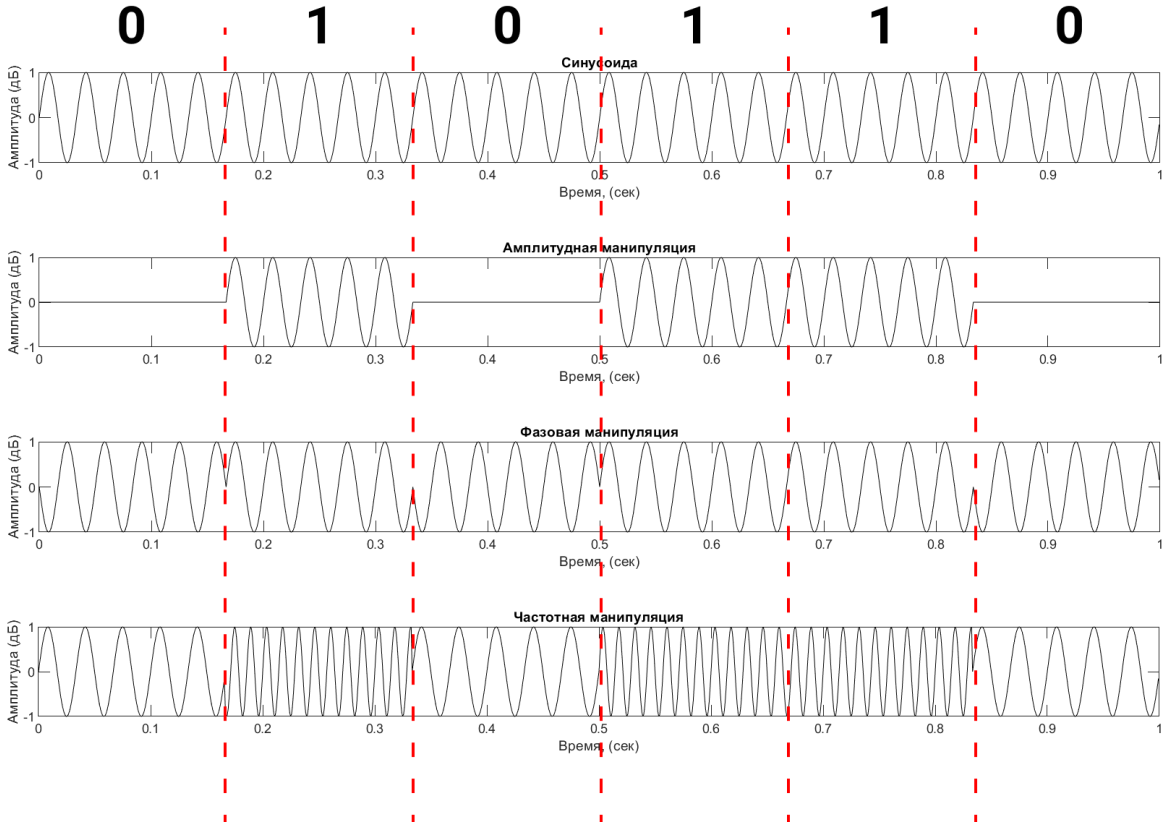


Рисунок 1.2 – Передача данных с использованием разных видов манипуляций

Здесь мы можем увидеть, как с течением времени в сигнале меняются его параметры и по данным параметрам мы можем определить какую информацию передаёт передатчик.

Таким образом передаются данные в беспроводных системах связи. Правда в настоящее время используются более сложные виды манипуляций, которые представлены комбинацией тех, что мы видим на рисунке 1.2.

Для организации беспроводной связи между контроллерами используются специальные модули. В практической части мы познакомимся с самым распространенным модулем NRF. И используя беспроводной канала передачи, попробуем принять и обработать значение.

2. Практическая часть

2.1 Реализация передатчика

В данной работе мы будем использовать беспроводной модуль для передачи информации, который расположен в правом верхнем углу платы (рисунок 2.1):

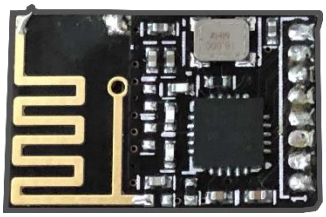


Рисунок 2.1 – Модуль NRF

Данный модуль передает сигнала в нелицензируемой полосе частот 2.4 ГГц, что дает возможность строить свои системы беспроводной передачи данных без проблем с законодательством. В данной полосе частот также работает Wi-Fi и некоторые другие технологии беспроводной передачи данных.

Для работы с данным модулем используется библиотека RF24.

После того, как библиотека будет установлена, создайте новый скетч и подключите необходимые библиотеки. Как вы помните, модуль NRF подключен по интерфейсу SPI. Значит, необходимо подключить библиотеку для работы с данным интерфейсом, а также библиотеки для работы с NRF: «`<nRF24L01.h>`» и «`<RF24.h>`».

Когда были подключены все необходимые библиотеки, начните работу по созданию передатчика. Для этого определите и переименуйте необходимые для работы пины и инициализируйте работу NRF модуля, также создайте массив, который будет хранить передаваемую информацию:

```
#define PIN_POT PB1   // Номер пина Arduino, к которому подключен потенциометр
#define PIN_CE PB2    // Номер пина Arduino, к которому подключен вывод CE радиомодуля
#define PIN_CSN PB15 // Номер пина Arduino, к которому подключен вывод CSN радиомодуля
RF24 radio(PIN_CE, PIN_CSN); // Создаём объект radio с указанием выводов CE и CSN
int potValue[1]; // Создаём массив для передачи значений потенциометра
```

Далее в «`void setup()`» необходимо инициализировать работу UART со скоростью 115200, определить пин потенциометра в соответствии с его назначением. Затем инициализируйте работу модуля NRF2301, используя команду «`radio.begin()`».

Далее необходимо настроить NRF модуль. У модуля NRF, как говорилось ранее настраивается канал передачи, например, пятый – «`radio.setChannel(5)`», скорость обмена данными, например, 1 Мбит/с – «`radio.setDataRate (RF24_1MBPS)`», мощность передатчика, например, высокая – «`radio.setPALevel(RF24_PA_HIGH)`». Также необходимо открыть уникальную нить общения, используя ID устройств, например, `0x7878787878LL` – «`radio.openWritingPipe(0x7878787878LL)`». Основа кода, описанного выше, представлена ниже:

```
void setup() {
  Serial.begin(...);
  pinMode(PIN_POT, ...);
  radio.begin();  // Инициализация модуля NRF24L01
  radio.setChannel(5); // Обмен данными будет вестись на пятом канале (2,405 ГГц)
  radio.setDataRate (...); // Скорость обмена данными 1 Мбит/сек
  radio.setPALevel(...); // Выбираем высокую мощность передатчика (-6dBm)
  radio.openWritingPipe(...); // Открываем трубу с уникальным ID
}
```

Далее, в «void loop()» необходимо считать данные с потенциометра и записать их в созданную переменную. Выведете значение потенциометра в монитор порта, чтобы выполнить проверку передаваемых и принятых значений и, воспользовавшись командой «radio.write(potValue, 1)», отправить показания потенциометра по первому радиоканалу.

Основа описанного кода, представлена ниже:

```
void loop() {
  potValue[0] = (analogRead(PIN_POT)/4096.0)*255; // Считываем показания потенциометра
  Serial.println(potValue[...]);
  radio.write(«имя переменной», «канал передачи»); // Отправляем считанные показания по
радиоканалу
}
```

Допишите код и загрузите его на плату. Предупредите окружающих, что вы начали передачу данных в вашем канале и сообщите ID вашего канала. Необходимо проверить, что отправляемые вами данные совпадают с принятыми данными на плате приемника и, реализованный в ней функционал работает.

2.2 Реализация приемника

Переходя к написанию кода приемника, необходимо выполнить те же подключения библиотек, определение пинов CE и CSN радиомодуля, а также пин к которому подключен вентилятор или один из пинов RGB светодиода. Создайте переменную, которая будет хранить в себе принятые значения по радиоканалу. Основа кода, выполняющего описанные дейтсвия, представлена ниже:

```
#include <...> // Подключаем библиотеку для работы с SPI-интерфейсом
#include <...> // Подключаем файл конфигурации из библиотеки RF24
#include <...> // Подключаем библиотеку для работы для работы с модулем NRF24L01

#define PIN_FAN PA1 // Номер пина Arduino, к которому подключен вентилятор
#define PIN_CE PB2 // Номер пина Arduino, к которому подключен вывод CE радиомодуля
#define PIN_CSN PB15 // Номер пина Arduino, к которому подключен вывод CSN радиомодуля
RF24 radio(..., ...); // Создаём объект radio с указанием выводов CE и CSN
... potValue[1]; // Создаём массив для приёма значений потенциометра [0-255]
```

Далее, в «void setup()» необходимо настроить пин вентилятора или светодиода, в соответствии с его назначением. Затем инициализируйте работу модуля NRF2301, используя команду «radio.begin()».

Далее необходимо настроить NRF модуль. У модуля NRF, как говорилось ранее настраивается канал передачи, например, пятый – «radio.setChannel(5)», скорость обмена данными, например, 1 Мбит/с – «radio.setDataRate (RF24_1MBPS)», чувствительность приемника, например, высокая – «radio.setPALevel(RF24_PA_HIGH)». Также необходимо открыть уникальную нить общения, используя ID устройств, например, 0x7878787878LL – «radio.openWritingPipe(0x7878787878LL)» и дать модулю понять, что ему необходимо прослушивать информацию в радиоканале, используя команду – «radio.startListening()». Описанный здесь фрагмент кода, полностью повторяет «void setup()» из части передатчика за дополнением его командой, начинающей прослушивание радиоканала.

Далее в «void loop()» необходимо наложить условие проверки наличия на приемнике информации, воспользовавшись библиотечной функцией «radio.available()», и, если это условие выполняется, то необходимо прочесть эту информацию и записать ее в переменную, которую вы создали. Для этого используется команда «radio.read(&potValue, sizeof(potValue))». Затем, записанное значение в переменную, необходимо передать на пин вентилятора или светодиода.

Основа кода, описанного выше будет выглядеть следующим образом:

```
void loop() {
  if(radio. ...) {// Если в буфер приёмника поступили данные
    radio.read(&potValue, sizeof(potValue)); // Читаем показания потенциометра
    analogWrite(..., potValue[...]); // Регулируем яркость диода
  }
}
```

Допишите код, загрузите на плату и проверьте реакцию вашей платы на радиосигнал, идущий с передающей платы.

Дополнительное задание! Реализуйте прием данных от передающей платы, где вам по радиоканалу будет передаваться температура окружающей. Исходя из полученной температуры реализуйте логику работы вентилятора и свечения RGB светодиода, который бы сигнализировал о том, что включена отопительная система или охлаждающая.

Работа № 18

«Беспроводные модули интернета вещей. часть 3»

Под топологией (компоновкой, конфигурацией, структурой) компьютерной сети обычно понимается физическое расположение компьютеров сети один относительно одного и способ соединения их линиями связи. Важно отметить, что понятие топологии относится, в первую очередь, к локальным сетям, в которых структуру связей можно легко проследить. В глобальных сетях структура связей обычно спрятана от пользователей не слишком важная, потому что каждый сеанс связи может выполняться по своему собственному пути. Топология определяет требования к оборудованию, тип используемого кабеля, возможные и наиболее удобные методы управления обменом, надежность работы, возможности расширения сети.

Цель работы: реализовать сеть используя радио модуль NRF24.

Задачи:

1) Изучить теоретический материал;
2) Реализовать программу NRF передатчика;
3) Реализовать программу NRF приемника;
4) Реализовать взаимодействие между передатчиком и приемником.

1. Теоретическая часть

Топология – это схема соединения каналами связи компьютеров или узлов сети между собой.

Сетевая топология может быть

- физической – описывает реальное расположение и связи между узлами сети.
- логической – описывает хождение сигнала в рамках физической топологии.
- информационной – описывает направление потоков информации, передаваемых по сети.
- управления обменом – это принцип передачи права на пользование сетью.

Существует множество способов соединения сетевых устройств. Выделяют следующие топологии:

- ячеистая
- общая шина
- звезда
- кольцо

Существует три основные топологии сети, рассмотрим каждую из них по подробнее.

1.1 Топология шина

Топология типа шина, представляет собой общий кабель , к которому подсоединены все рабочие станции (рисунок 1.1).

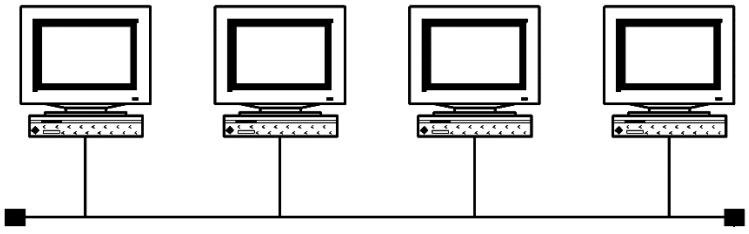


Рисунок 1.1 – Сетевая топология «шина»

Отправляемое рабочей станцией сообщение распространяется на все компьютеры сети. Каждая машина проверяет – кому адресовано сообщение и если ей, то обрабатывает его. Для того, чтобы исключить одновременную посылку данных, применяется либо «несущий» сигнал, либо один из компьютеров является главным и «даёт слово» остальным станциям.

Шина самой своей структурой допускает идентичность сетевого оборудования компьютеров, а также равноправие всех абонентов. При таком соединении компьютеры могут передавать только по очереди, потому что линия связи единственная. В противном случае переданная информация будет искажаться в результате наложения (конфликту, коллизии). Таким образом, в шине реализуется режим полудуплексного.

В топологии «шина» отсутствует центральный абонент, через которого передается вся информация, которая увеличивает ее надежность (ведь при отказе любого центра перестает функционировать вся управляемая этим центром система). Добавление новых абонентов в шину достаточно простое и обычно возможно даже во время работы сети. В большинстве случаев при использовании шины нужно минимальное количество соединительного кабеля по сравнению с другой топологией. Правда, нужно учесть, что к каждому компьютеру (кроме двух крайних) подходит два кабеля, что не всегда удобно.

Шине не страшны отказы отдельных компьютеров, потому что все другие компьютеры сети могут нормально продолжать обмен. Может показаться, что шине не страшен и обрыв кабеля, поскольку в этом случае остаются две полностью работоспособных шины. Однако из-за особенности распространения электрических сигналов по длинным линиям связи необходимо предусматривать включение на концах шины специальных устройств – терминаторов.

Достоинства:

- Небольшое время установки сети;
- Дешевизна (требуется меньше кабеля и сетевых устройств);
- Простота настройки;
- Выход из строя рабочей станции не отражается на работе сети.

Недостатки:

- Любые неполадки в сети, как обрыв кабеля, выход из строя терминатора полностью уничтожают работу всей сети;
- Сложная локализация неисправностей;

С добавлением новых рабочих станций падает производительность сети.

1.2 Топология звезда

Звезда – базовая топология компьютерной сети, в которой все компьютеры сети присоединены к центральному узлу, образуя физический сегмент сети. Подобный сегмент сети может функционировать как отдельно, так и в составе сложной сетевой топологии (как правило «дерево»). Весь обмен информацией идет исключительно через центральный компьютер, на который таким способом ложится очень большая нагрузка, потому ничем другим, кроме сети, оно заниматься не может. Как правило, именно центральный компьютер является самым мощным, и именно на него возлагаются все функции по управлению обменом. Никакие конфликты в сети с топологией звезда в принципе невозможные, потому что управление полностью централизовано (рисунок 1.2).

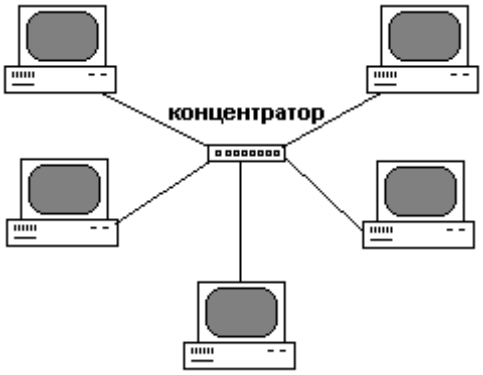


Рисунок 1.2 – Сетевая топология «звезда»

Активная звезда – В центре сети содержится компьютер, который выступает в роли сервера.

Пассивная звезда – В центре сети с данной топологией содержится не компьютер, а концентратор, или хаб (hub), что выполняет ту же функцию, что и репитер. Он возобновляет сигналы, которые поступают, и пересылает их в другие линии связи

Достоинства:

- выход из строя одной рабочей станции не отражается на работе всей сети в целом;
- хорошая масштабируемость сети;
- лёгкий поиск неисправностей и обрывов в сети;
- высокая производительность сети (при условии правильного проектирования);
- гибкие возможности администрирования.

Недостатки:

- выход из строя центрального концентратора обернётся неработоспособностью сети (или сегмента сети) в целом;
- для прокладки сети зачастую требуется больше кабеля, чем для большинства других топологий;
- конечное число рабочих станций в сети (или сегменте сети) ограничено количеством портов в центральном концентраторе.

1.3 Топология кольцо

Кольцо – это топология, в которой каждый компьютер соединен линиями связи только с двумя другими: от одного он только получает информацию, а другому только передает. На каждой линии связи, как и в случае звезды, работает только один передатчик и один приемник (рисунок 1.3).

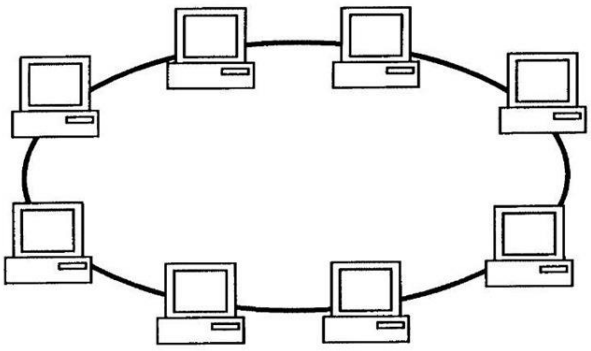


Рисунок 1.3 – Сетевая топология «кольцо»

Важна особенность кольца заключается в том, что каждый компьютер ретранслирует (возобновляет) сигнал, то есть выступает в роли ретронслятора, потому затухание сигнала во всем кольце не имеет никакого значения, важно только затухание между соседними компьютерами кольца. Четко выделенного центра в этом случае нет, все компьютеры могут быть одинаковыми. Однако достаточно часто в кольце выделяется специальный абонент, который управляет обменом или контролирует обмен. Понятно, что наличие такого управляющего абонента снижает надежность сети, потому что выход его из строя сразу же парализует весь обмен.

Компьютеры в кольце не являются полностью равноправными (в отличие, например, от шинной топологии). Одни из них обязательно получают информацию от компьютера, который ведет передачу в этот момент, раньше, а другие – позже. Именно на этой особенности топологии и строятся методы управления обменом по сети, специально рассчитанные на «кольцо».

Подключение новых абонентов в «кольцо» обычно совсем безболезненно, хотя и требует обязательной остановки работы всей сети на время подключения. Как и в случае топологии «шина», максимальное количество абонентов в кильке может быть достаточно большая (до тысячи и больше). Кольцевая топология обычно является самой стойкой к перегрузкам, она обеспечивает уверенную работу с самими большими потоками переданной по сети информации, потому что в ней, как правило, нет конфликтов (в отличие от шины), а также отсутствует центральный абонент (в отличие от звезды).

Достоинства:

- Простота установки;
- Практически полное отсутствие дополнительного оборудования;
- Возможность устойчивой работы без существенного падения скорости передачи данных при интенсивной загрузке сети, поскольку использование маркера исключает возможность возникновения коллизий.

Недостатки:

- Выход из строя одной рабочей станции, и другие неполадки (обрыв кабеля), отражаются на работоспособности всей сети;
- Сложность конфигурирования и настройки;

Сложность поиска неисправностей.

1.4 Ячеистая топология

В ячеистой сети один узел может напрямую взаимодействовать с несколькими другими узлами. Существует два типа ячеистых сетей: полная и частичная. В полносвязной топологии каждый узел может напрямую взаимодействовать с любым другим узлом в сети. В частично ячеистой сети, показанной на рисунке 1.4, каждый узел может напрямую подключаться к одному или нескольким другим узлам в сети, но не обязательно к каждому другому узлу в сети.

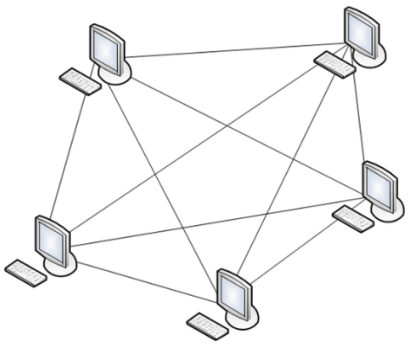


Рисунок 1.4 – ячеистая сетевая топология

Приложения IoT обычно используют топологию частичной сетки для расширения диапазона сети. В ячеистой сети узлы могут действовать как повторители, направляя данные по сети. В результате между каждыми двумя узлами существует несколько различных путей. Эта избыточность повышает устойчивость сети, и в случае сбоя одного пути можно использовать альтернативный путь для распространения данных по сети. Поскольку узлы могут действовать как ретрансляторы, узлы, которые не находятся в пределах прямого радиодиапазона друг друга, могут по-прежнему обмениваться данными через узлы маршрутизатора. Это главное преимущество ячеистой сети в приложениях IoT, поскольку она позволяет пользователю расширить диапазон сети за пределы одной радиосистемы.

Недостатком является то, что многоскачковая природа связи может увеличить задержку распространения пакета данных по сети. Количество переходов и, следовательно, сетевая задержка зависит от количества маршрутизаторов, через которые проходит пакет данных. Это усложняет оценку производительности сети по сравнению с простой структурой, такой как описанная выше топология «звезда».

2. Практическая часть

Один модуль NRF24L01 может активно прослушивать до 6 других модулей одновременно.

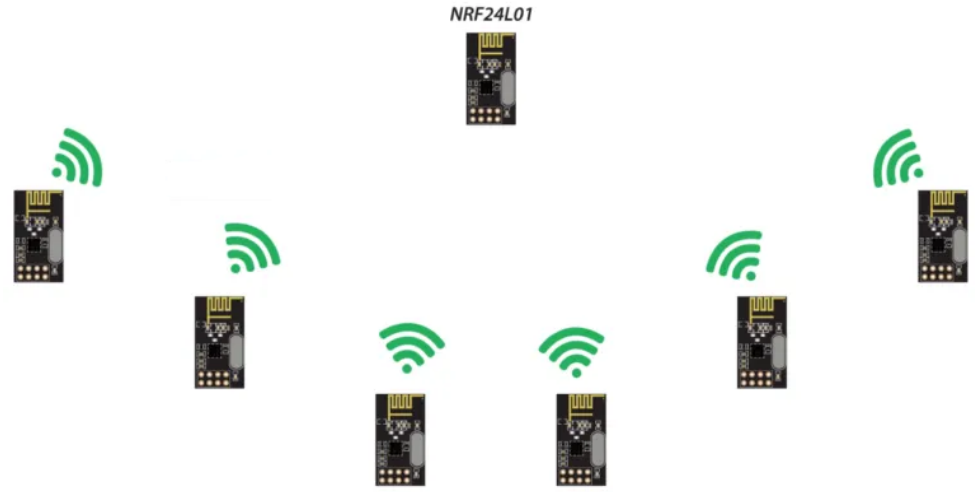


Рисунок 1.5 – Количество подключаемых модулей

Эта возможность используется библиотекой RF24Network для создания сети, расположенной в древовидной топологии, где один узел является базовым, а все остальные узлы являются дочерними либо от этого узла, либо от другого. Каждый узел может иметь до 5 дочерних узлов, и это может идти на 5 уровней вглубь, что означает, что мы можем создать сеть из 3125 узлов. Каждый узел должен быть определен с 15-битным адресом, который точно описывает положение узла в дереве.

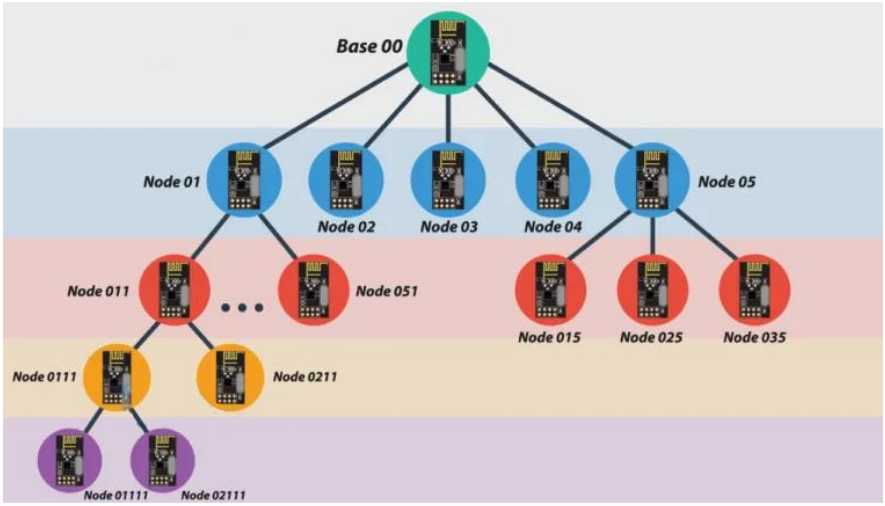


Рисунок 1. 6 – Принцип построения сети с использованием NRF24

Адреса узлов определяются в восьмеричном формате. Так, адрес мастера или базы равен 00, дочерние адреса базы – от 01 до 05, дочерние адреса узла 011 – от 011 до 051 и так далее.

Обратите внимание, что, если узел 011 хочет взаимодействовать с узлом 02, связь должна проходить через узел 01 и базовый узел 00, поэтому эти два узла должны быть активны все время, чтобы связь была успешной.

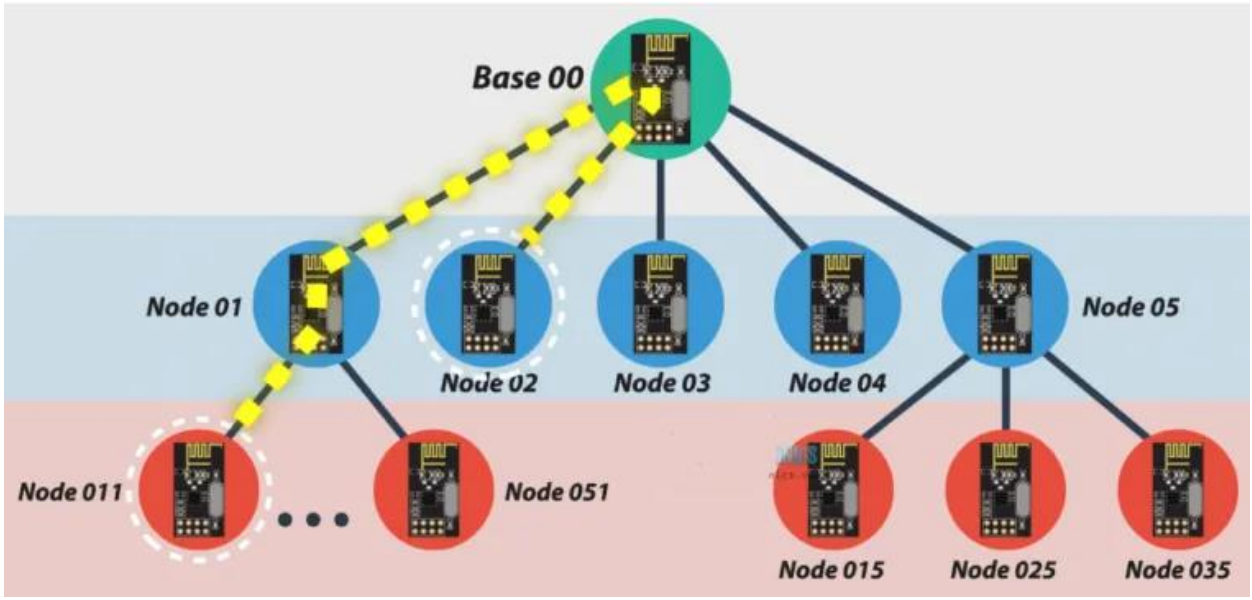


Рисунок 1.7 – Передача данных по сети

2.1 Написание кода приемо-передатчика

Подключите библиотеки необходимые для работы с радио модулем NRF24, а также библиотеку для организации сети. Так же необходимо настроить взаимодействие с датчиком температуры, LM75A, с которым вы работали в предыдущих работах.

```
#include <RF24.h> // Библиотека для работы с NRF 24
#include <RF24Network.h> // Библиотека для организации сети
#include <… … …> // Библиотека для работы с датчиком температуры
#include <… … … > // Библиотека SPI
```

Объявите пины радиомодуля к которым подключен передатчик и приемник по аналогии с предыдущей работой.

```
      #define PIN_CE  …….  // Номер пина Arduino, к которому подключен вывод CE
радиомодуля
      #define PIN_CSN …… // Номер пина Arduino, к которому подключен вывод CSN
радиомодуля
```

Затем нам нужно создать объект RF24 и включить его в объект RF24Network. Дальше нужно определить адреса узлов в восьмеричном формате, или 00 для главного узла, который у преподавателя и 01 для вашего узла.

```
      RF24 radio(PIN_CE, PIN_CSN);  // С оздаём объект radio с указанием выводов CE
и CSN
      RF24Network network(radio);        // Добавить объект радио в сеть
      const uint16_t this_node = 01;    // Адресс данного узла в восьмиричном формате
      const uint16_t master = 00;      // Адресс узла куда передаем информацию
```

Создайте экземпляр класса M2M_LM75A для работы с датчиком температуры по аналогии с предыдущими работами.

```
      M2M_LM75A lm75a;
```

В разделе настройки нам нужно инициализировать сеть, установив канал и адрес этого узла. Задать скорость serial порта 115200. Далее необходимо инициализировать пин PB14 как OUTPUT и перевести его в состояние HIGH, так как для работы шины SPI SS должно быть в нулевом состоянии, а все остальные пины в высоком.

```
      void setup()
      {
          Serial.begin(……);
          pinMode(PB14, ……);
          digitalWrite(PB14, ……);
          SPI.begin();
          radio.begin();
          network.begin(90, this_node);  //(channel, node address)
          lm75a.begin();

      }
```

В разделе цикла нам постоянно нужно вызывать функцию update(), через которую происходят все действия в сети.

Считайте данный с датчика температуры по аналогии с предыдущими работами в переменную temp:

```
      void loop()
      {
          network.update();
          double temp = ……;
```

Значения температуры мы получаем в формате Double, чтобы передать их по каналу связи необходимо преобразовать их в массив символов char и объединить с сообщением, которое необходимо передать. Создайте массив char с именем temp1 что бы в дальнейшем записать туда значения температуры. Для этого необходимо использовать функцию dtostrf (floatvar, StringLengthIncDecimalPoint, numVarsAfterDecimal, charbuf), где floatvar – преобразуемая переменная типа double, StringLengthIncDecimalPoint – длина получаемого символьного значения, numVarsAfterDecimal – количество символов после запятой, charbuf – символьный массив для сохранения результата преобразования.

```
char temp1[4];
dtostrf(temp, 4, 2, arr);
```

Далее необходимо создать два массива типа char. В одно мы запишем сообщение, которое хотим передать, во втором будет храниться финальная посылка, состоящая из сообщения и показаний с датчика температуры. Обратите внимание что размер финальной посылки не должен превышать 32 байта

```
char msg[14] = "Hello my temp:";
char fin[32];
```

Что бы объеденить два массива в один будем использовать следующие функции:

1) char * strcpy(char * destptr, const char * srcptr);

Функция копирует Си-строку *srcptr*, включая завершающий нулевой символ в строку назначения, на которую ссылается указатель *destptr*.

2) char * strcat(char * destptr, const char * srcptr);

Объединение строк. Функция добавляет копию строки *srcptr* в конец строки *destptr*. Нулевой символ конца строки *destptr* заменяется первым символом строки *srcptr*, и новый нуль-символ добавляется в конец уже новой строки, сформированной объединением символов двух строк в строке *destptr*.

```
strcpy(fin, msg);
strcat(fin, arr);
```

В цикле while постоянно проверяем нашу сеть на доступность. Создадим экземпляр класса RF24NetworkHeader необходимый для определения с какой ноды приходит сообщение.

```
while (network.available()) // проверяем доступность сети
{
    RF24NetworkHeader header;
```

Создадим переменную len и массив типа char для того, чтобы хранить принятые сообщения.

```
int len = 0;
char gotmsg[32];
```

Функция radio.getDynamicPayloadSize динамически определяет длинну принятого пакета.

Далее, используя функцию network.read(header, &gotmsg, len), будем принимать отправленные сообщения, где Header – переменная где будет храниться адрес того кто отправил сообщения, &gotmsg – указатель на массив, где будет храниться принятое сообщение, Len – длина принятого сообщения.

```
      len = radio.getDynamicPayloadSize();
      network.read(header, &gotmsg, len); // считывает принятые данные.
```

Создадим переменную ID в которую при помощи функции header. from_node номер ноды с которой пришло сообщение. Далее используя функцию *prinln* выведите сообщение (NODE: номер ноды с которой получили сообщение), и далее само сообщение.

```
      int ID = header.from_node;
            Serial.print("…… ");
            Serial.println(……);
            Serial.print(……);
            delay(1000);
            Serial.println();


          }
```

Для передачи сообщения для начала необходимо создать заголовочный файл с именем того устройства которому необходимо передать сообщение используя функцию RF24NetworkHeader header2(node01), где node01 адрес устройства которому необходимо передать сообщение.

Для самой передачи сообщения используется функция network.write(header2, &fin, sizeof(fin), где header2 – созданный ранее заголовочный файл, &fin указатель на массив с сообщением, sizeof(fin), – размер передаваемого массива.

```
      //=========================TXdata=============================================
======
          RF24NetworkHeader header2(node01);     // (Address where the data is going)
          bool ok = network.write(header2, &fin, sizeof(fin)); // Send the data
          //=====================================RXdata=========================
      }
```

Работа № 19

«Беспроводные модули интернета вещей. часть 4»

Передача данных с помощью Wi-Fi – технологии беспроводной локальной сети с устройствами на основе стандартов IEEE 802.11. Основными диапазонами частот Wi-Fi стандартизированы 2.4 ГГц, 5 ГГц и 6 ГГц. Передавать сигнал можно на большие расстояний, однако, в таком случае, для приема потребуется антенна с большим коэффициентом усиления. Отличительной особенность Wi-Fi перед другими стандартами беспроводной связи является высокая скорость передачи. Вышедший в 2019 году стандарт подразумевает скорость передачи до 11 Гбит/с и называется Wi-Fi 6.

В данной работе будет рассмотрено и реализовано управление платой по Wi-Fi при помощи веб-интерфейса, позволяющего управлять платой со смартфона и получать информацию с платы, непосредственно, на него.

Цель работы: изучение способа создания системы дистанционного управления устройством с использованием технологии Wi-Fi и WEB-сервера.

Задачи:

1) Изучить принцип работы Wi-Fi;

2) Изучить принципы работы системы дистанционного управления устройствами;

3) Изучить один из способа реализации дистанционного управления устройствами с использованием WEB-сервера.

1. Теоретическая часть

1.1 Описание технологии Wi-Fi

Стандарт беспроводной передачи данных Wi-Fi был создан специально для объединения нескольких компьютеров в единую локальную сеть. Обычные проводные сети требуют прокладки множества кабелей через стены, потолки и перегородки внутри помещений. Также имеются определенные ограничения на расположение устройств в пространстве. Беспроводные сети Wi-Fi лишены этих недостатков: можно добавлять компьютеры и прочие беспроводные устройства с минимальными физическими, временными и материальными затратами. Для передачи информации беспроводные устройства Wi-Fi используют радиоволны из спектра частот, определенных стандартом IEEE 802.11. Существует четыре разновидности стандарта Wi-Fi (таблица 1.1). 802.11n поддерживает работу сразу в двух частотных диапазонах одновременно на четыре антенны. Суммарная скорость передачи данных при этом достигается 150-600 Мбит/с.

Таблица 1.1 – Разновидности стандарта Wi-Fi

Стандарт	802.11b	802.11g	802.11a	802.11n	802.11ac	802.11ax
Частотный диапазон, ГГц	2,4	2,4	5	2,4/5	5	2,4/5
Максимальная скорость передачи данных в радиоканале, Мбит/с	11	54	54	150–600	до 6,77 Гбит/с 8x MU-MIMO	до 11 Гбит/с 8x MU-MIMO

Сформулируем некоторые ключевые особенности стандарта Wi-Fi. К его преимуществам относятся:

– высокая скорость передачи данных;

– компактность;

– большое разнообразие модулей под разные задачи;

– высокий уровень стандартизации и совместимость между устройствами Wi-Fi разных производителей;

– защита передаваемых данных.

Основной недостаток:

– большое энергопотребление и невозможность работы в течение длительного времени от автономных источников питания.

1.2 Система дистанционного управления на плате TUSUR_IOT_BOARD

Часто, когда мы сталкиваемся с «умными» системами, чаще всего, они имеют некоторый интерфейс для взаимодействия с ними.

К примеру, меню платы на плате TUSUR_IOT_BOARD. Данное меню позволяет настроить периферию платы и поменять некоторые параметры всей системы. Но данное меню обрабатывается прям на плате, с помощью вычислительных мощностей контроллера, который установлен на нем.

Для организации дистанционного управления нам может понадобится другое устройство, которое будет подключено по беспроводному каналу с управляемым устройством.

В данной работе мы будем использовать сеть Wi-Fi и наш смартфон с браузером, который позволит подключиться к WEB-серверу платы и отправлять команды на контроллер ESP, который по UART каналу будет ретранслировать их в контроллер STM.

Посредник в виде UART канала добавлен, так как у студентов есть возможность программировать только контроллер STM, а Wi-Fi модуль установлен в контроллере ESP.

Если представить данную систему связи, то получится следующая схема (рисунок 1.1).

Особенностью данной системы дистанционного управления будет тот факт, что данная система управления будет графической.

Это значит, что взаимодействия пользователя с данной системой будет осуществляться за счет графического интерфейса. Который в данной системе будет отрисован в браузере.

Существуют и другие системы дистанционного управления устройствами. К примеру:

– консольные (где команды отправляются с использованием консоли);

– использующие специальные (сетевые) протоколы передачи данных по типу MQTT, FTP;

– и другие.

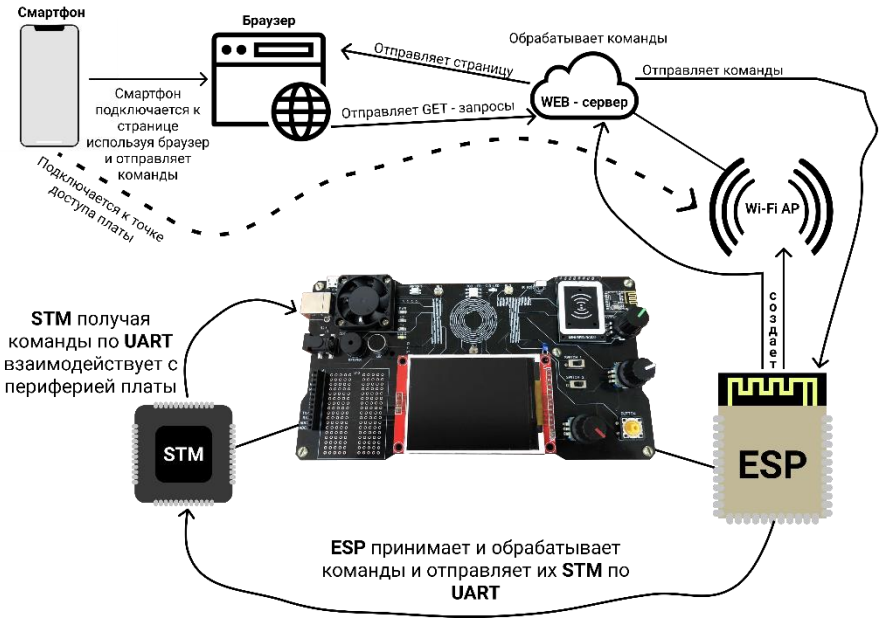


Рисунок 1.1 – Схема подключения

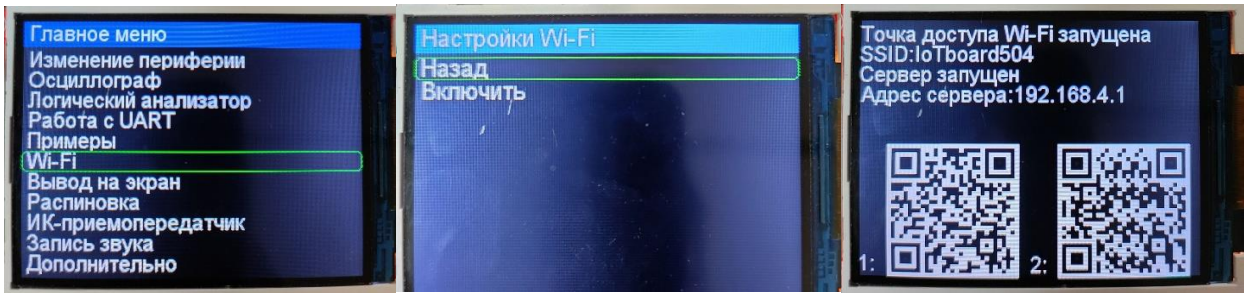
2. Практическая часть

2.1 Запуск сервера и подключения к веб-интерфейсу

В данной работе вам необходимо будет создать систему, которая будет управляться с помощью дистанционной системы управления.

Для настройки данной системы выполните следующие инструкции:

1. Включите Wi-Fi в меню платы. Для это зайдите в «Wi-Fi»-«Включить»:



2. Убедитесь, что Wi-Fi включился (должно открыться окно с информацией о том, что точка доступа запущена, имя сети, состояние сервера и IP сервера):

3. Далее воспользуйтесь своим смартфоном для того, чтобы подключиться к сети с именем, которое написано в SSID. Важно! Мобильная сеть должна быть отключена;

4. После того, как вы подключитесь к своей точке доступа зайдите в любой браузер и введите адрес из пункта -IP.

Рисунок 2.1 – Графический интерфейс вывода информации и удаленного управления платой TUSUR_IOT_BOARD

Если у вас отобразился графический интерфейс, представленный на рисунке 2.1 – значит вы все выполнили верно. Информация в графе данных будет отсутствовать. Система удаленного управления платой TUSUR_IOT_BOARD способна отправлять 4 к 2 кнопок, соответственно, «вкл» и «выкл». Команды, которые отправляют кнопки: 1: «вкл» – 0, «выкл» – 1; 2: «вкл» – 2, «выкл» – 3.

2.2 Написание программы управления платой через Wi-Fi

Данная работа позволит написать код, с помощью которого будет осуществляться удаленное управление платой, а также мониторинг параметров платы.

Для работы необходимо подключить подключить библиотеки для работы с SPI, I2C, датчиком температуры – LM75A, и датчиком давления BMP280. Определите пины RGB светодиода, вентилятора, свитча и кнопки при помощи директивы препроцессора.

В «`void setup()`» инициализируйте работу датчиков, как это осуществлялось в предыдущих работах, инициализируйте работу последовательного порта со скоростью 115200, и объявите все необходимые пины с определением их типа.

В «void loop()» считайте данные с датчиков и поместите их в переменные: «Temp LM», «Temp BMP», «Pressure». Объявите строки, отвечающие за передачу информации на сервер: «ledOn», «ledOff», «ledOnMassage», «ledOffMassage», «fanOn», «fanOff», «fanOnMassage»,«fanOffMassage», «tempFromLM», «tempFromBMP»,«pressFromBMP», «», «tempFromLmMassage»,« tempFromBmpMassage», «pressFromBmpMassage». Необходимо заполнить данные строки информацией, которую вы хотите вывести, как это выполнялось на предыдущих занятиях. Например:

```
String Press;
  Press = "Давление: ";
  Press+= Pressure;
  Press+= " мм рт. ст.";
```

После того, как были сформированы строки, несущие в себе информацию, необходимо сформировать строки с командами, позволяющими отправлять данные команды на сервер. Такие строки пишутся очень просто:

```
PressPrint = "wifiSend(";
  PressPrint+= 3;
  PressPrint+=",";
  PressPrint+= Press;
  PressPrint+=")";
```

Здесь «wifiSend» – команда, отправляющая данные на сервер, «3» – номер строки вывода, «Press» – строка с информацией. Всего можно вывести 7 строк информации.

Кнопки управления веб интерфейса Wi-Fi отправляют в последовательный порт значения от 0 до 3, соответствующих номеру кнопки и ее состоянию. Соответственно, для осуществления работы необходимо распознать наличие информации в последовательном порте и считать ее:

```
if(Serial.available()){
    int PR = Serial.read();
```

Следующим шагом является написание конструкции «switch/case» с 4 кейсами, в которых будет осуществляться управление функциями платы и вывод соответствующей информации о изменении состояния переключателя. Пример написания конструкции «switch/case» для данной работы приведен ниже:

```
switch (PR){
      case 0:
        analogWrite(R,250);
        analogWrite(G,250);
        analogWrite(B,250);
        ledOn = "Освещение ";
        ledOn+= " включено";
        ledOnMassage = "wifiSend(";
        ledOnMassage += 5;
        ledOnMassage +=",";
        ledOnMassage += ledOn;
        ledOnMassage +=")";
```

```
        delay(100);
        Serial.println(ledOnMassage);
        delay(100);
      break;
```

Вывод информации о температуре с датчика LM75A, датчика BMP, а также давления должен осуществляться в независимости от наличия информации в последовательном порте.

Основа конечного кода программы, представлена ниже:

```
#include "…"
#include "…"
#include <…> // Подключаем библиотеку для работы с LM75A
#include <…>// Библиотека, которая хранит в себе функции для взаимодействия с
датчиком BMP280.
Adafruit_BMP280 …; // Создание объекта bmp280 – температурный датчик
M2M_LM75A …; // Подключение класса датчика
#define fan PA1 // PA1
#define R PB9 // PB9
#define G PB8 // PB8
#define B PB1 // PB1
#define SW1 PC15 // PC14 или PC15
#define beep PA8 // PA8
#define but PB4 // PB4


void setup() {
  tone(beep, 100 * 25, 50); // Сигнализация
  delay(200);
  tone(beep, 100 * 25, 50); // Сигнализация
  lm75a.begin(); // Инициализация работы датчика
  while (!bmp280.begin(…)) { // Цикл для проверки подключения температурного
датчика по I2C;
  }
  Serial.begin(…); // Инициализация работы UART
  pinMode(R, …); // Красный на выход
  pinMode(G, …); // Зеленый на выход
  pinMode(B, …); // Синий на выход
  pinMode(fan, …); // Вентилятор на выход
  pinMode(beep, …); // Бипер на выход
  pinMode(PC13, …); // PC13 на выход
  pinMode(SW1, …); // Свитч на вход
}

void loop() {
  double temp = lm75a.getTemperature(); // Считывание температуры с LM75A
  double tempBMP = bmp280.readTemperature(); // Записываем значение с датчика
температуры в переменную temp
  double Pressure = (bmp280.readPressure())/100*0.750062;
  String ledOn;
```

```
String ledOff;
String ledOnMassage;
String ledOffMassage;

String fanOn;
String fanOff;
String fanOnMassage;
String fanOffMassage;
String b;
String a;
a = "Текущая температура: ";
a+=temp;
a+=" C";
b = "Текущая температура BMP: ";
b+=tempBMP;
b+=" C";
String as;
as = "wifiSend(";
as+= 1;
as+=",";
as+= a;
as+=")";
String bs;
bs = "wifiSend(";
bs+= 2;
bs+=",";
bs+= b;
bs+=")";
String Press;
Press = "Давление: ";
Press+= Pressure;
Press+= " мм рт. ст.";
String PressPrint;
PressPrint = "wifiSend(";
PressPrint+= 3;
PressPrint+=",";
PressPrint+= Press;
PressPrint+=")";

Serial.println(as);
delay(100);
Serial.println(bs);
delay(100);
Serial.println(PressPrint);
if(Serial.available()){
  int PR = Serial.read();
  switch (PR){
    case 0:
      analogWrite(R,250);
```

```
        analogWrite(G,250);
        analogWrite(B,250);
        ledOn = "Освещение ";
        ledOn+= " включено";
        ledOnMassage = "wifiSend(";
        ledOnMassage += 5;
        ledOnMassage +=",";
        ledOnMassage += ledOn;
        ledOnMassage +=")";
        delay(100);
        Serial.println(ledOnMassage);
        delay(100);
      break;
      case 1:
        analogWrite(R,0);
        analogWrite(G,0);
        analogWrite(B,0);
        ledOff = "Освещение ";
        ledOff += " выключено";
        ledOffMassage = "wifiSend(";
        ledOffMassage += 5;
        ledOffMassage +=",";
        ledOffMassage += ledOff;
        ledOffMassage +=")";
        delay(100);
        Serial.println(ledOffMassage);
        delay(100);
      break;
      case 2:
        analogWrite(fan,250);
        fanOn = "Вентилятор ";
        fanOn+= " включен";
        fanOnMassage = "wifiSend(";
        fanOnMassage += 6;
        fanOnMassage +=",";
        fanOnMassage += fanOn;
        fanOnMassage +=")";
        delay(100);
        Serial.println(fanOnMassage);
        delay(100);
      break;
      case 3:
        analogWrite(fan,0);
        fanOff = "Вентилятор ";
        fanOff += " выключен";
        fanOffMassage = "wifiSend(";
        fanOffMassage += 6;
        fanOffMassage +=",";
        fanOffMassage += fanOff;
```

```
        fanOffMassage +=")";
        delay(100);
        Serial.println(fanOffMassage);
        delay(100);
        break;
      default:
        analogWrite(R,0);
        analogWrite(G,0);
        analogWrite(B,0);
        analogWrite(fan,0);
        ledOff = "Освещение ";
        ledOff += " выключено";
        ledOffMassage = "wifiSend(";
        ledOffMassage += 5;
        ledOffMassage +=",";
        ledOffMassage += ledOff;
        ledOffMassage +=")";
        delay(100);
        Serial.println(ledOffMassage);
        delay(100);
        fanOff = "… ";
        fanOff += " …";
        fanOffMassage = "wifiSend(";
        fanOffMassage += …;
        fanOffMassage +=",";
        fanOffMassage += fanOff;
        fanOffMassage +=")";
        delay(100);
        Serial.println(fanOffMassage);
        delay(100);
      break;
  }
  }
}
```

Работа № 20

«Разработка проекта «умный дом» на отладочной плате»

Цель работы: Закрепление знаний, полученных в ходе выполнения предыдущих лабораторных работ. Выполнение итогового проекта.

Задачи:

1) Изучить работу модели «умного дома», написать программу, выполняющую основные сценарии, предусмотренные в комнате 1.

2) Написать программу, выполняющую основные сценарии, предусмотренные в комнате 2.

3) Написать программу, выполняющую основные сценарии, предусмотренные в комнате 3.

1. Изучение модели «умного дома»

Во встроенном ПО платы реализована модель «умного дома». Данная модель состоит из трех комнат виртуального дома, в каждой из которых можно реализовать различный набор сценариев Интернета вещей используя реальные модули на плате и виртуальное окружение комнаты.

Для того чтобы запустить данную модель в меню платы нужно выбрать пункт «Дополнительно», затем пункт «Модель умного дома». После этого на экране платы будет представлено меню для выбора комнат умного дома с различным функционалом.

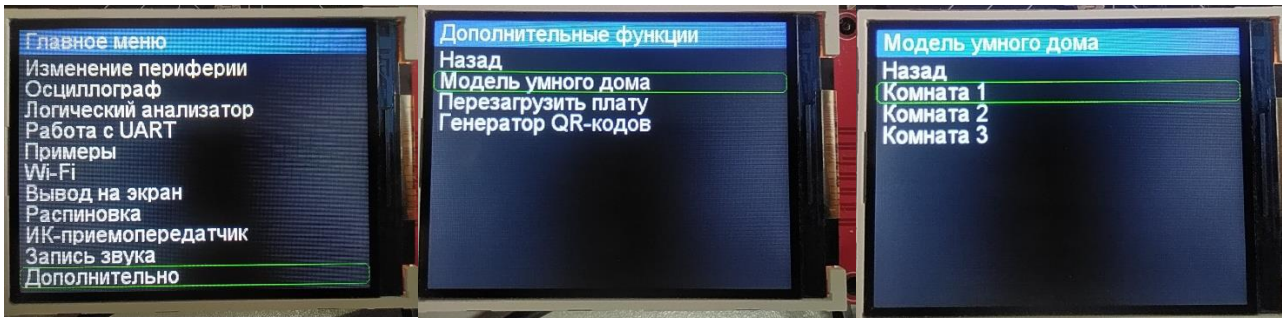


Рисунок 1.1 – Меню платы

Для начала выберите «Комната 1». Вы увидите меню со списком поддерживаемых в данной комнате команд, список виртуальных датчиков, и краткий список периферии платы, которую можно использовать в данной комнате.

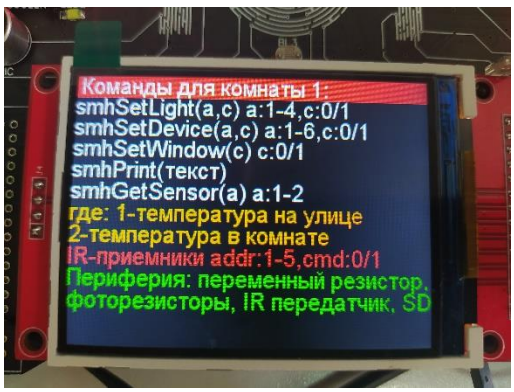


Рисунок 1.2 – Команды комнаты 1

После ознакомления с данной информацией, нажмите на энкодер платы, чтобы переместиться далее. Далее на экране будет представлена модель комнаты 1.

Рисунок 1.3 – Модель комнаты 1

Как представлено на рисунке 3 данная комната представляет из себя кабинет, в котором расположено множество устройств, групп освещения, которыми вы можете управлять. Управление происходит при помощи текстовых команд, которые необходимо отправлять по UART. Аналогичным образом вы выводили графическую информацию в лабораторной работе с экраном.

Список поддерживаемых команд в комнате 1 представлен в таблице 1.1.

Таблица **Ошибка! Текст указанного стиля в документе отсутствует.**.1 – Описание списка команд для комнаты 1

Команда	Назначение
smhSetLight(a,c)	Управление элементом освещения где a – номер элемента освещения (от 1 до 4), c – команда включения (1) или выключения (0).
smhSetDevice(a,c)	Управление устройствами в комнате где a – номер устройства (от 1 до 6), c – команда включения (1) или выключения (0).
smhSetWindow(c)	Проветривание окна где c – команда для открывания (1) или закрывания (0) окна.
smhPrint(текст)	Вывод текстовой информации в нижней части экрана где текст – выводимый текст на экран. Кавычки для вывода текста в данной команде не требуются.
smhGetSensor(a)	Команда для получения информации с виртуальных датчиков где a – номер датчика (1 или 2), 1 – виртуальный датчик температуры на улице, 2 – виртуальный датчик температуры в комнате.

Также к данной комнате подключен реальный ИК-приёмник сигнала, расположенный на плате. Это позволяет управлять устройствами в комнате при помощи инфракрасного пульта, который вы можете реализовать при помощи ИК-передатчика на плате. Прием сигналов ведется по протоколу NEC, как и в лабораторной работе по передаче команд по ИК-каналу.

Как было описано в предыдущих лабораторных по работе с ИК-каналом, управляющий по протоколу NEC состоит из адреса и команды. Соответственно, для управления устройствами в виртуальной комнате по ИК-каналу нужно указать адрес устройства (от 1 до 5) и команду на включение (1) и выключение (0) устройства. Например, для включения устройства 3 нужно отправить команду 0x03FC01FE.

Из-за конструкционных особенностей платы не все модули на плате могут быть задействованы в разработке проекта для автоматизации комнаты 1. Периферия, которую можно использовать в комнате 1: переменный резистор, фоторезисторы, ИК – передатчик, SD карта, модуль RFID, модуль NRF24L01, датчик BMP280, датчик LM75A, beeper, кнопка, переключатели, светодиод PC13.

Для того, чтобы протестировать работу команд откройте Arduino IDE, убедитесь, что в менеджере плат выбрана плата STM32F103C и выбранный COM порт соответствует плате. Затем запустите монитор порта. На плате загорится белый светодиод, как показано на рисунке ниже. Теперь команды из монитора порта ПК будут подаваться напрямую в модель «умного дома».

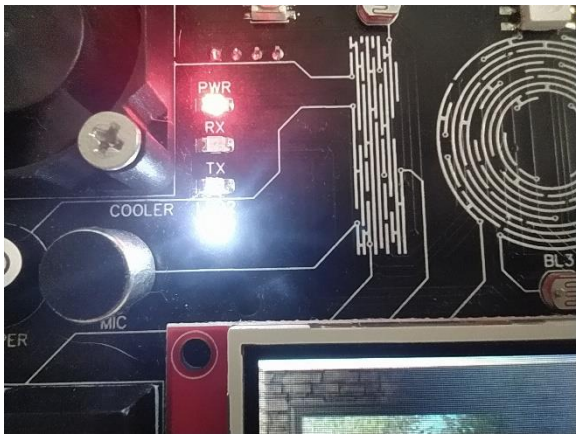


Рисунок 1.4 – Светодиод на плате

Установите скорость 115200 в мониторе порта ArduinoIDE. Модель «умного дома настроена на эту скорость UART».

Затем в поле ввода информации введите команду smhSetLight(1,1) и нажмите кнопку «отправить».

Если команда была введена верно, то один виртуальный светильник должен был включиться. Также в правом верхнем углу экрана появится зеленый круг, означающий правильный ввод команды. Если команда введена неверно, то круг будет окрашен в красный цвет.

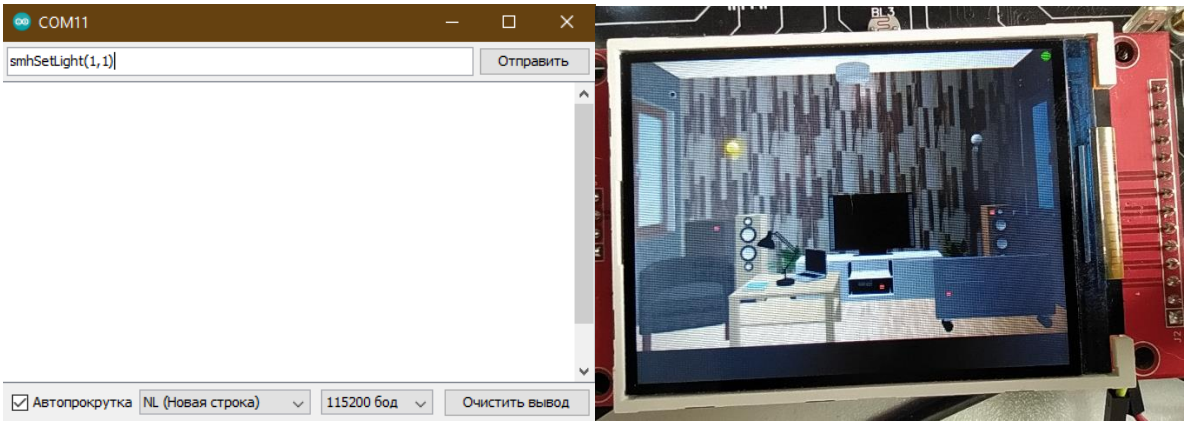


Рисунок 1.5 – Отправка команды

После этого введите команду smhSetLight(1,0) и светильник 1 выключится.

Следующим шагом введите команду smhSetDevice(3,1) и виртуальная приставка в центре комнаты будет включена, её индикация изменит цвет на зеленый.

Если введете команду smhPrint(t1=25), то в нижней части экрана появится надпись «t1=25». Эта команда будет полезна для вывода необходимой вам информации на экран.

После того как вы изучили принцип ввода команд протестируйте все команды из таблицы 1 и определите какое устройство в комнате соответствует определенному номеру в командах.

Внимание: при вводе команды smhGetSensor(a) с ПК, вы не сможете увидеть ответ от датчика, так как вторая линия UART, используемая в модели, соединена только с микроконтроллером STM32F103C, а не с ПК. Как использовать эту команду будет описано далее.

2. Задание для комнаты 1

Задание 1 заключается в обеспечении автоматизации в виртуальной комнате 1 при помощи реальных модулей на плате. Для этого создайте новый проект в ArduinoIDE. В качестве примера рассмотрим код, который будет считывать уровень освещенности на фоторезисторе и в зависимости от этого включать и выключать соответствующий светильник в комнате 1.

```
bool sensorValueflag = 0;

void SendMSG(String MSG){
  Serial.println(MSG);
  delay(500);
}

void setup() {
 Serial.begin(115200);
}

void loop() {
 int sensorValue = analogRead(PA1);
 if(sensorValue<800 && sensorValueflag){
   SendMSG("smhSetLight(1,1)");
   sensorValueflag = 0;
 }
 else if(sensorValue>800 && !sensorValueflag){
   int exampleVAR1 = 1;
   int exampleVAR2 = 0;
   String ExampleMGG = "smhSetLight(" + (String)exampleVAR1 + "," + (String)exampleVAR2 + ")"; // строка
"smhSetLight(1,0)"
   SendMSG(ExampleMGG);
   sensorValueflag = 1;
 }
}
```

Как видно в представленном коде, в Setup мы инициализируем UART с необходимой скоростью 115200 бод/с. Также мы создаем функцию SendMSG(), которая принимает в качестве аргумента строку типа String и отправляет ее по UART, после этого установлена задержка в 500 миллисекунд. Она обязательна для предотвращения переполнения буфера UART. Поэтому настоятельно рекомендуется использовать для отправки всех команд именно такую функцию.

В основном цикле программы происходит считывание показаний АЦП с PA1 (Данный пин подключен к фоторезистору BL1). Далее если значение станет ниже порога 800, в виртуальную комнату будет отправлена команда smhSetLight(1,1), а если значение будет выше то команда smhSetLight(1,1). Флаг sensorValueflag используется для того, чтобы команда отправлялась только один раз после пересечения установленного порога на АЦП, так как иначе команды на включение и выключение отсылались бы непрерывно, а такого допускать нельзя.

Как показано в коде строку для отправки на UART можно формировать как из нескольких переменных в единую строку String, так и непосредственно вписывать необходимую команду в кавычках в аргумент функции.

Загрузите данный код в микроконтроллер и запустите модель комнаты 1, если ее выключили. Если закрыть рукой фоторезистор BL1, то светильник в комнате 1 должен включаться. Если же руку убрать – выключаться.

Внимание! проверьте что монитор порта на ПК выключен и белый светодиод LED2 не горит. Так как при подключенном мониторе порта ПК связь между микроконтроллером и виртуальной комнатой будет отключена.

Далее рассмотрим код, который получает информацию от виртуального датчика температуры модели.

```
uint8_t t1 =0;

void SendMSG(String MSG){
  Serial.println(MSG);
  delay(500);
}

void setup() {
  Serial.begin(115200);
  pinMode(PB4,INPUT);
}

void loop() {
   int buttonValue = digitalRead(PB4);
   if(buttonValue){
   SendMSG("smhGetSensor(1)");
}

if(Serial.available()){
     t1 = Serial.read();
     String ExampleMGG = "smhPrint(t1=" + (String)t1 + ")";
     SendMSG(ExampleMGG);
  }
}
```

Аналогично первому коду мы инициализируем UART и создаем функцию для отправки команды. Также мы подключаем кнопку и в основном цикле программы отправляем команду на считывание показаний датчика при нажатии кнопки. Далее при наличии данных в приемном буфере UART мы записываем их в переменную t1 и отправляем в текстовом виде значение на экран.

Датчик отправляет ответ через секунду после принятия команды. Температура на датчике 1(температура на улице) изменяется периодически, а температура на датчике 2 (температура) изменяется в зависимости от нее, также на изменение температуры в виртуальной комнате влияет открытие окна и включение обогревателя.

Рисунок 1.6 – Вывод температуры на экран

После того как вы ознакомились с работой модели виртуальной комнаты напишите код, выполняющий две из представленных автоматизаций в комнате 1 (для успешной сдачи проекта нужно выполнить по 2 любых автоматизации в каждой комнате, рекомендуется только после этого писать код для остальных автоматизаций в качестве дополнения):

1) При изменении освещенности на фоторезисторах BL1, BL2, BL3 должны включаться и выключаться светильники 1, 2, 3.

2) При переключении SWITCH_1 должен включаться и выключаться светильник 4.

3) При переключении SWITCH_2 настройте включение и выключение телевизора при помощи ИК-передатчика. (например, команда по протоколу NEC для включения телевизора 0x01FE01FE).

4) При помощи модуля RFID обеспечьте открытие виртуального сейфа по выбранной RFID метке и закрытие при прикладывании неверной метки.

5) Обеспечьте в комнате 1 автоматическое удержание температуры 25 градусов Цельсия (показание датчика 2) при помощи включения и выключения виртуального обогревателя и открытия окна в комнате 1. Температуру в комнате и на улице выводите на экран платы.

6) Обеспечьте автоматизацию нескольких устройств в комнате по любому придуманному вами сценарию.

3. Задание для комнаты 2

Для того чтобы запустить данную модель виртуальной комнаты 2 в меню платы нужно выбрать пункт «Дополнительно», затем пункт «Модель умного дома», «Комната 2».

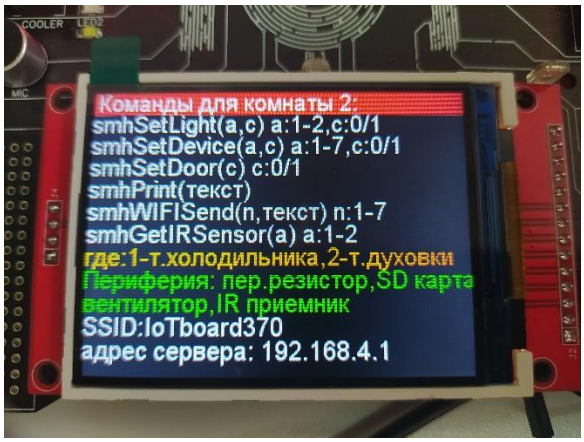


Рисунок 3.1 – Команды комнаты 2

Вы увидите меню со списком поддерживаемых в данной комнате команд, список виртуальных датчиков, и краткий список периферии платы, которую можно использовать в данной комнате.

Также в данном меню вы увидите SSID точки доступа Wi-Fi, запущенной в данной модели, и адрес сервера к которому можно подключиться при помощи Вашего смартфона.

После ознакомления с данной информацией, нажмите на энкодер платы, чтобы переместиться далее. Далее на экране будет представлена модель комнаты 2.

Рисунок 3.2 – Модель комнаты 2

Как представлено на рисунке 8 данная комната представляет из себя кухню, в котором расположено множество устройств, групп освещения, которыми вы можете управлять.

Список поддерживаемых команд в комнате 2 представлен в таблице 3.1.

Таблица 3.1 – Описание списка команд для комнаты 2

Команда	Назначение
smhSetLight(a,c)	Управление элементом освещения где a – номер элемента освещения (от 1 до 2), c – команда включения (1) или выключения (0).
smhSetDevice(a,c)	Управление устройствами в комнате где a – номер устройства (от 1 до 7), c – команда включения (1) или выключения (0).
smhSetDoor(c)	Открытие двери где c – команда для открывания (1) или закрывания (0) двери.
smhPrint(текст)	Вывод текстовой информации в нижней части экрана где текст – выводимый текст на экран. Кавычки для вывода текста в данной команде не требуются.
smhGetIRSensor(a)	Команда для получения информации с виртуальных датчиков где a – номер датчика (1 или 2), виртуальный датчик температуры холодильника, виртуальный датчик температуры духовки. Ответ с датчиков приходит по ИК каналу
smhWIFISend(n,текст)	Команда для отправки текстовой информации на сервер по Wi-Fi где n – номер строки, текст – выводимый текст на сервер.

Из-за конструкционных особенностей платы не все модули на плате могут быть задействованы в разработке проекта для автоматизации комнаты 2. Периферия, которую можно использовать в комнате 2: переменный резистор, вентилятор, ИК – приемник, SD карта, модуль RFID, модуль NRF24L01, датчик BMP280, датчик LM75A, beeper, кнопка, переключатели, светодиод PC13.

К данной комнате подключен реальный ИК-передатчик сигнала, расположенный на плате. Это позволяет управлять получать данные с датчиков при помощи инфракрасного приемника, который вы можете реализовать при помощи ИК-приемника на плате. Прием сигналов ведется по протоколу NEC, как и в лабораторной работе по приему команд по ИК-каналу.

Команды комнаты 2 аналогичны командам в комнате 1 за некоторым исключением.

При отправке команды smhGetIRSensor(a) ответ от датчика придет не по UART, а по ИК-каналу через 1 секунду после принятия команды. Для того чтобы получить эту информацию вы должны реализовать ИК-приемник на плате, как вы делали это в соответствующей лабораторной работе. Полученный с датчика пакет будет выглядеть следующим образом: 1 байт – адрес датчика (1 или 2), 2 байт – инвертированное значение адреса, 3 байт – значение с датчика (значение температуры), 4 байт – инвертированное значение. Например, ответ с датчика 2, если его значение будет равно 25 градусов Цельсия, следующий: 0x02FD19E6 02 – адрес датчика, 19 – значение температуры (25) в шестнадцатеричном виде.

При нажатии на кнопку на сервере, к которому вы можете подключиться при помощи смартфона, по UART на микроконтроллер будет отправлен соответствующий байт информации (0 и 1 для кнопки 1, 2 и 3 для кнопки 2).

Внимание! при работе с сервером проверьте, что монитор порта на ПК выключен и белый светодиод LED2 не горит. Так как при подключенном мониторе порта ПК связь между микроконтроллером и виртуальной комнатой будет отключена.

Перед тем как писать код подключите монитор порта ПК к плате и проверьте работу всех поддерживаемых в комнате 2 команд и установите соответствие устройств в виртуальной комнате их номерам в командах.

После того как вы ознакомились с работой модели виртуальной комнаты напишите код, выполняющий две из представленных автоматизаций в комнате 2 (для успешной сдачи проекта нужно выполнить по 2 любых автоматизации в каждой комнате, рекомендуется только после этого писать код для остальных автоматизаций в качестве дополнения):

1) Обеспечьте управление двумя любыми виртуальными электроприборами с Вашего смартфона. На сервере, к которому Вы можете подключиться по Wi-Fi, имеются две кнопки для включения и выключения. Работа с ними аналогична лабораторной работе с Wi-Fi.

2) Выведите на сервер, к которому Вы можете подключиться по Wi-Fi, показания с датчика BMP280, а также состояние двери. Открывать и закрывать дверь нужно по переключению SWITCH_1 на плате.

3) При помощи ИК-приемника считайте показания температуры с виртуальных датчиков холодильника и духовки и выведите их на экран по нажатию на кнопку платы. Определите, как изменяется температура при включении и выключении приборов.

4) При помощи ИК-приемника считайте показания температуры с виртуального датчика духовки и выведите ее на экран. Обеспечьте удержание температуры 150 градусов в духовке путем ее автоматизации.

5) Настройте автоматическое открывание двери при помощи модуля RFID.

4. Задание для комнаты 3

Для того чтобы запустить данную модель виртуальной комнаты 3 в меню платы нужно выбрать пункт «Дополнительно», затем пункт «Модель умного дома», «Комната 3».

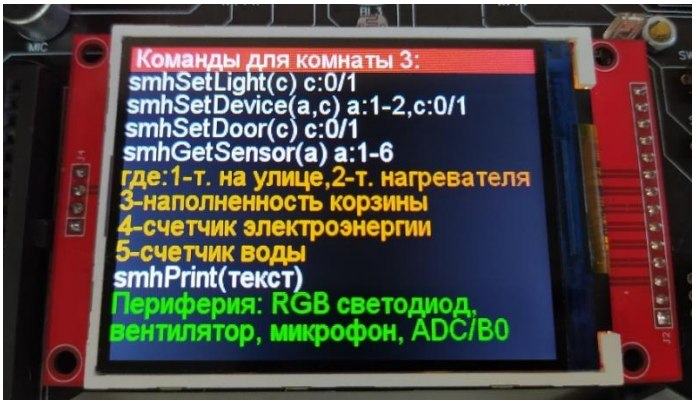


Рисунок 4.1 – Команды комнаты 3

Вы увидите меню со списком поддерживаемых в данной комнате команд, список виртуальных датчиков, и краткий список периферии платы, которую можно использовать в данной комнате.

После ознакомления с данной информацией, нажмите на энкодер платы, чтобы переместиться далее. Далее на экране будет представлена модель комнаты 3.

Рисунок 4.2 – Модель комнаты 3

Список поддерживаемых команд в комнате 3 представлен в таблице 3.

Таблица 4.1 – Описание списка команд для комнаты

Команда	Назначение
smhSetLight(c)	Управление элементом освещения где c – команда включения (1) или выключения (0).
smhSetDevice(a,c)	Управление устройствами в комнате где a – номер устройства (от 1 до 2), c – команда включения (1) или выключения (0).
smhSetDoor(c)	Открытие двери где c – команда для открывания (1) или закрывания (0) двери.
smhPrint(текст)	Вывод текстовой информации в нижней части экрана где текст – выводимый текст на экран. Кавычки для вывода текста в данной команде не требуются.
smhGetSensor(a)	Команда для получения информации с

	виртуальных датчиков где a – номер датчика (от 1 до 6), 1 – температура на улице, 2 – температура нагревателя, 3 – наполненность корзины, 4 – счетчик электроэнергии, 5 – счетчик воды, 6 – датчик воды (протечки).

Из-за конструкционных особенностей платы не все модули на плате могут быть задействованы в разработке проекта для автоматизации комнаты 3. Периферия, которую можно использовать в комнате 3: RGB светодиод, вентилятор, модуль RFID, микрофон, модуль NRF24L01, датчик BMP280, датчик LM75A, beeper, кнопка, переключатели, светодиод PC13.

В комнате 3 расположено большое количество датчиков. Ответ от них поступает на микроконтроллер по UART. Эти виртуальные датчики показывают следующие значения: температура на улице, которая периодически изменяется, температура водонагревателя, которая изменяется при его включении, наполненность мусорной корзины в процентах, которая возрастает при ее наполнении и обнуляется после ее очистки, показания счетчиков воды и электроэнергии.

Перед тем как писать код подключите монитор порта ПК к плате и проверьте работу всех поддерживаемых в комнате 2 команд и установите соответствие устройств в виртуальной комнате их номерам в командах.

После того как вы ознакомились с работой модели виртуальной комнаты напишите код, выполняющий две из представленных автоматизаций в комнате 3 (для успешной сдачи проекта нужно выполнить по 2 любых автоматизации в каждой комнате, рекомендуется только после этого писать код для остальных автоматизаций в качестве дополнения):

1) Настройте автоматическое открывание входной двери по RFID метке.
2) Считайте показания с виртуальных счетчиков и отправьте их на плату преподавателя при помощи модуля NRF24L01.
3) Настройте цветовую индикацию температуры на улице при помощи RGB светодиода, где цвет будет перетекать от синего к красному.
4) Настройте автоматическую очистку мусорной корзины при ее заполнении. Заполненность корзины нужно выводить на экран платы.
5) Настройте автоматизацию в комнате так, чтобы при изменении температуры на улице ниже 15 градусов включался водонагреватель, а при изменении её выше 15 градусов запускался вентилятор на плате.
6) Настройте поддержание температуры водонагревателя на отметке 80 градусов. Температуру водонагревателя нужно выводить на экран.
7) Настройте автоматизацию, в которой при обнаружении протечки датчиком воды (6) отправлялся сигнал на плату преподавателя по NRF24L01 и RGB светодиод загорался красным.

ЗАКЛЮЧЕНИЕ

Успешно пройденный курс позволяет приобрести навыки программирования микроконтроллеров на языке Wiring, теоретические знания и практический навык работы с датчиками и исполнительными устройствами, знания и навыки работы с интерфейсами взаимодействия периферийных устройств с микроконтроллером, такие как: UART, I2C и SPI. При успешном усвоении курса студентами также будут получены знания и практические навыки по таким беспроводным интерфейсам таким, как Wi-Fi, NRF и IR.

Курс дает исчерпывающие знания по программированию встраиваемых систем, а также навыки их проектирования, что позволяет при успешном усвоении материала позволит самостоятельно проектировать и программировать встраиваемые системы.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. Макаров, С. Л. Arduino Uno и Raspberry Pi 3: от схемотехники к интернету вещей: руководство / С. Л. Макаров. — Москва: ДМК Пресс, 2018. — 204 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/11613 (дата обращения 07.09.2025).

2. Антти, С. Интернет вещей: видео, аудио, коммутация / С. Антти. — Москва: ДМК Пресс, 2019. — 120 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/123717 (дата обращения 07.09.2025).