Министерство науки и высшего образования Российской Федерации

Томский государственный университет систем управления и радиоэлектроники

Е. В. Рогожников
К. В. Савенко
В. Гмырь

ПРОГРАММИРОВАНИЕ ВСТРАИВАЕМЫХ СИСТЕМ

Методические указания для выполнения лабораторных работ для студентов направления подготовки 11.03.01 и 11.03.02 по дисциплине «Программирование встраиваемых систем»

Томск

2021

УДК 681.3.068

ББК 32.973.2

Р 598

Рецензент:

Абенов Р. Р., доцент кафедры телекоммуникаций и основ Радиотехники ТУСУРа, канд. техн. наук

Рогожников, Евгений Васильевич

Р 598 Программирование встраиваемых систем: Методические указания для выполнения работ для студентов направления подготовки 11.03.01 и 11.03.02 по дисциплине Программирование встраиваемых систем / Е. В. Рогожников, К. В. Савенко, В. Гмырь. – Томск: Томск. гос. ун-т систем управления и радиоэлектроники, 2021. − 102 с.

Настоящие учебно-методическое пособие содержит методические указания по выполнению лабораторных работ по дисциплине Программирование встраиваемых систем. Данный лабораторный практикум по данной дисциплине имеет целью закрепить полученные в ходе курса навыки программирования микроконтроллеров, получить навыки по созданию собственного проекта и по представлению технических проектов.

Одобрено на заседании кафедры ТОР, протокол № 8 от 29 апреля 2021 г.

УДК 681.3.068
ББК 32.973.2

Оглавление

ВВЕДЕНИЕ

Лабораторный практикум по курсу “Программирование встраиваемых систем” имеет целью закрепить и расширить теоретические знания студентов в области программирования микроконтроллеров, дать навыки по созданию собственного проекта и по представлению технических проектов.

Первая часть лабораторного практикума, предназначенного для студентов направления подготовки 11.03.02, содержит описание следующих работ:

1) Введение в программирование микроконтроллеров;
2) Введение в булеву алгебру;
3) Основы программирования микроконтроллеров;
4) Работа с АЦП;
5) Изучение интерфейсов UART, I2C, SPI;
6) Введение в интернет вещей;

Лабораторные 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 высокоуровневый язык программирования C++, что упрощает работу с данными. Данную программу можно скачать с официального сайта https://www.arduino.cc/en/software.

Лабораторная работа №1.

Введение в программирование микроконтроллеров

Цель работы: Ознакомить студентов с программированием микроконтроллеров

Задачи лабораторной 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.

Теоретическая часть

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

Микропроцессор – это центральное устройство любой ЭВМ, выполненный 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).

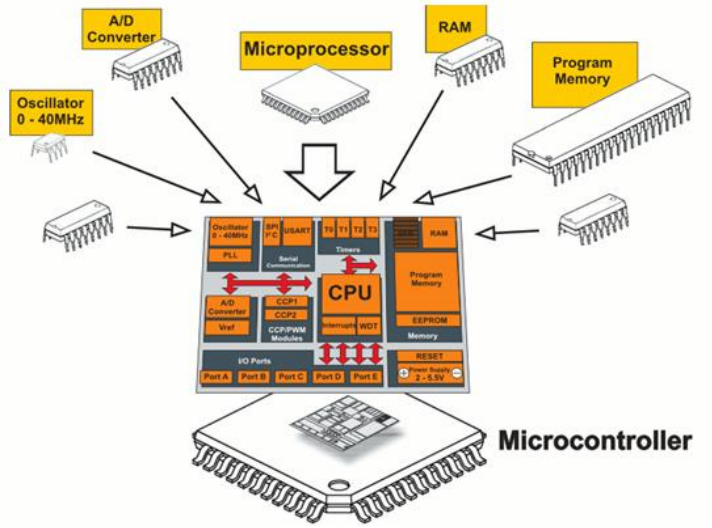


Рисунок 1.2 – Пример строения микроконтроллера

Каждый микроконтроллер имеет ножки (пины). Ножки – это некоторые органы чувств устройства, с помощью которых они взаимодействуют с окружающими устройствами. Каждый датчик, светодиод и многое другое 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):

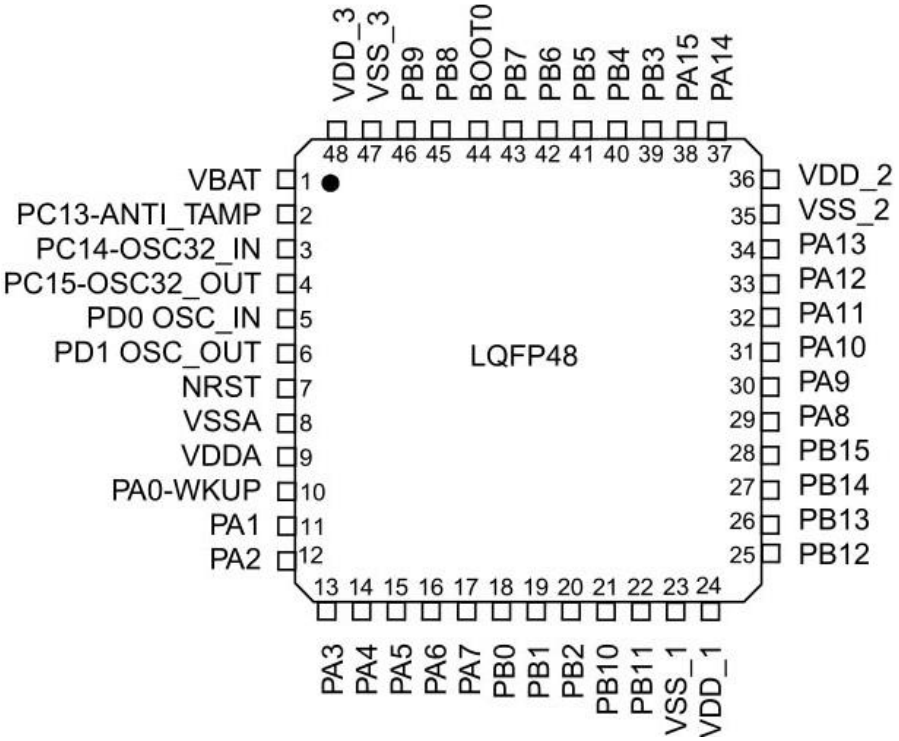


Рисунок 1.3 – Распиновка микроконтроллера STM32F103C8

В процессе выполнения лабораторных работ мы часто будем взаимодействовать с пинами микроконтроллера, к которым будут подключены разнообразные периферийные устройства

Сфера применения микроконтроллеров

Сфера их использования постоянно расширяется. 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;

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 (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 – это лишь инструмент, который позволяет 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 называет 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, который компилирует(переводит) написанную программу в машинный код, понятный устройству.

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

Среда разработки Arduino IDE

Среда разработки Arduino IDE – это ПО необходимое для программирования плат Aduino**.** Но 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**.**

Основной интерфейс программы представлен на рисунке 1.4:

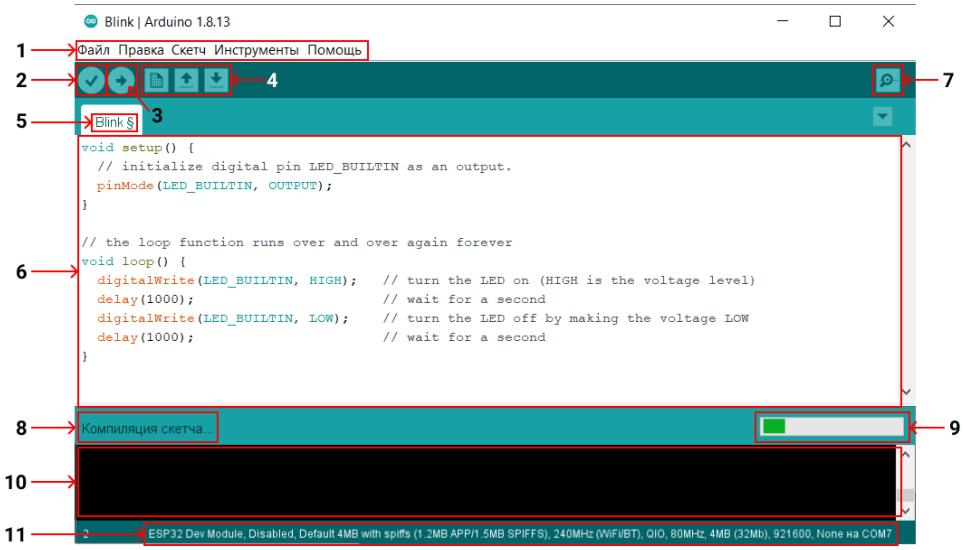


Рисунок 1.4 – Интерфейс программы

Для начала рассмотрим верхнюю часть окна (рисунок 1.5):

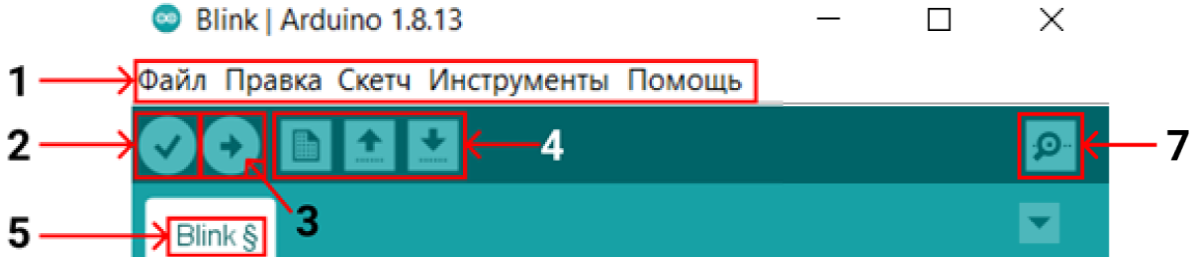


Рисунок 1.5 – Верхняя часть окна

Верхняя часть окна включает в себя следующие блоки:

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 позволяет 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 – 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.

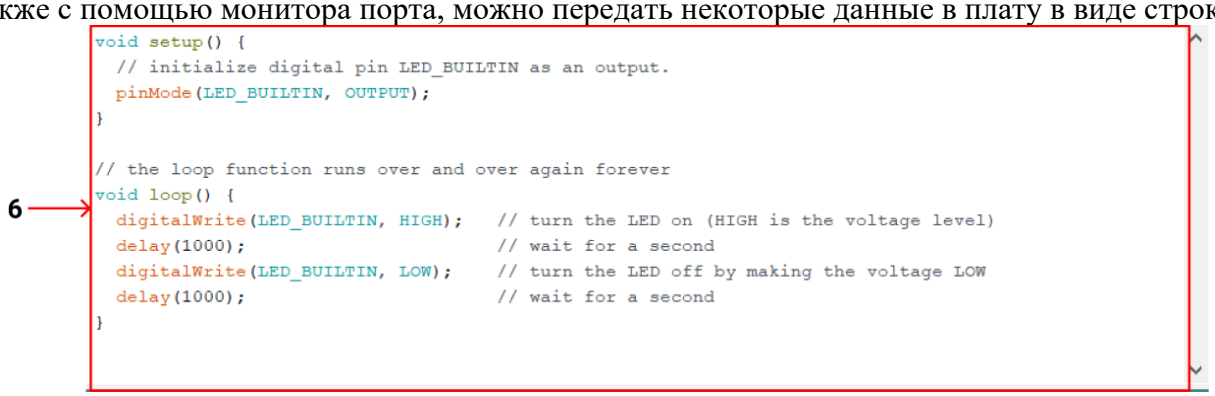


Рисунок 1.6 – Текстовый редактор кода:

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.

Далее рассмотрим нижнюю часть окна (рисунок 1.7).

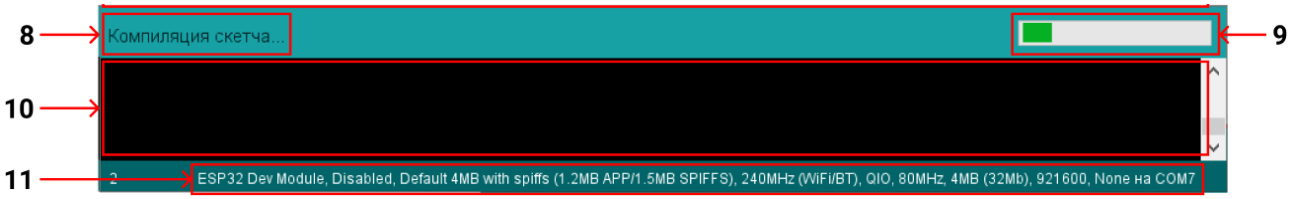


Рисунок 1.7 – Нижняя часть программы

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 1.8), которые появились при компиляции;

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

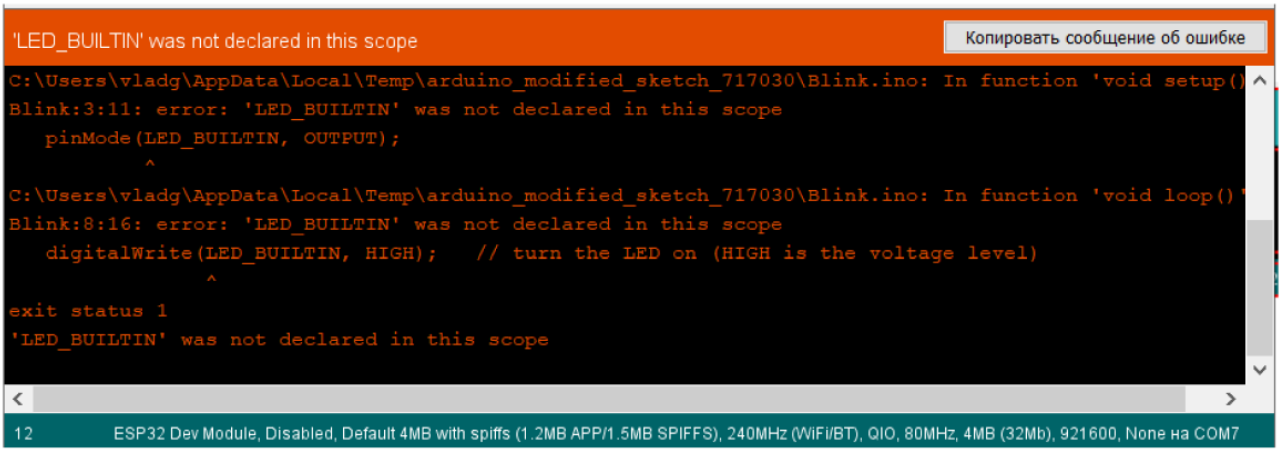


Рисунок 1.8 – Ошибка при компиляции скетча

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 течении курса, мы немного будем изучать структуру языка Си, которые потребуются 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!

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 нужное положение (осуществляется автоматически).

Контроллер STM32F103C является основным программируемым контроллером, к которому подключена большая часть периферии. Список подключенных модулей и датчик указан в таблице 1.1:

Таблица 1.1 – Список подключенных модулей и датчиков к плате TUSUR IoT Board

ВВОД	ВЫВОД
Датчик температуры	Вентилятор
Датчик давления	Светодиоды
RFID	Экран (ESP)
IR приемник	Зуммер
Кнопка, переключатели	RGB сетодиод
Encoder	IR передатчик
Микрофон	Динамик (ESP)
Переменный резистор	
Фоторезисторы	
ВВОД\ ВЫВОД	
NRF	
WI-FI (ESP)	
Bluetooth (ESP)	
Флешка	

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 открыть файл TIB_PinOut.pdf [1].

Главное меню прошивки для взаимодействия с платой представлено следующим образом (рисунок 1.9):

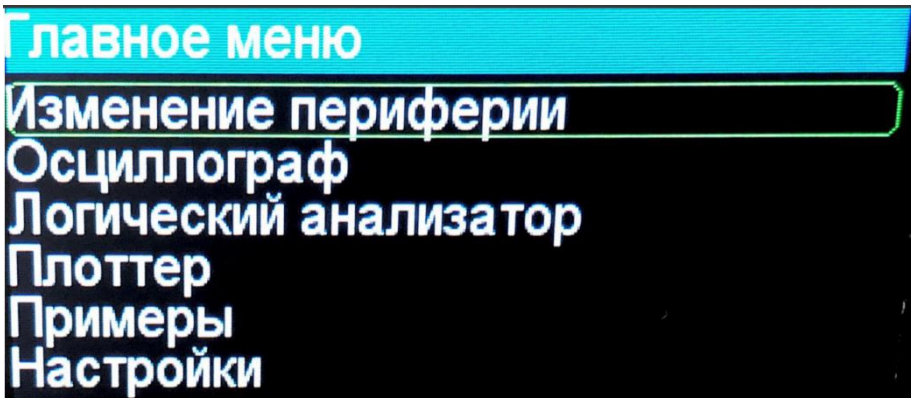


Рисунок 1.9 – Меню для взаимодействия с платой

Меню может быть представлено в другом виде. Зависит от прошивки. В момент написания методического пособия была установлена 6 версия прошивки.

Для навигации в меню необходимо использовать красный энкодер, который расположен недалеко от экрана.

Изменение периферии – меню которое позволяет подключать\отключать модули, подключенные к STM;

Осциллограф – осциллограф, которые позволяет вывести сигналы, которые подаются на один из пинов ESP;

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

Плоттер – выводит на экран значение, считанное с пина ESP в реальном времени;

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

Настройки – меню, которое позволяет осуществить некоторые настройки платы ESP.

Практическая часть

Разработка своего первого проекта

В самом начале практически каждый программист на любом языке начинает свой путь с программы, которая выводит “Hello World”.

Вот таким образом выглядит программа, выводящая “Hello World” в консоль, которая написана на языке Си

```
#include <stdio.h> //Подключение библиотеки вывода
int main (void) // Основная функция
{
puts("Hello, World!"); //Вывод текста
}
```

Так как на самом микроконтроллере нет возможности вывести какой-либо текст или изображение, потому что это просто набор полупроводниковых элементов, то воспользуемся одним из его пинов для того, чтобы подать нам сигнал о том, что он работает.

А 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 1.10):

Рисунок 1.10 – Иконка программы Arduino IDE

Далее в открывшемся окне нажмите на кнопку «Новый проект» (рисунок 1.11):

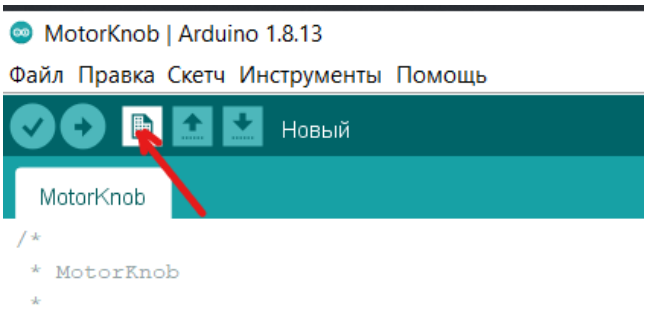


Рисунок 1.11 – Создание проекта

После создания нового проекта откроется следующее окно (рисунок 1.12):

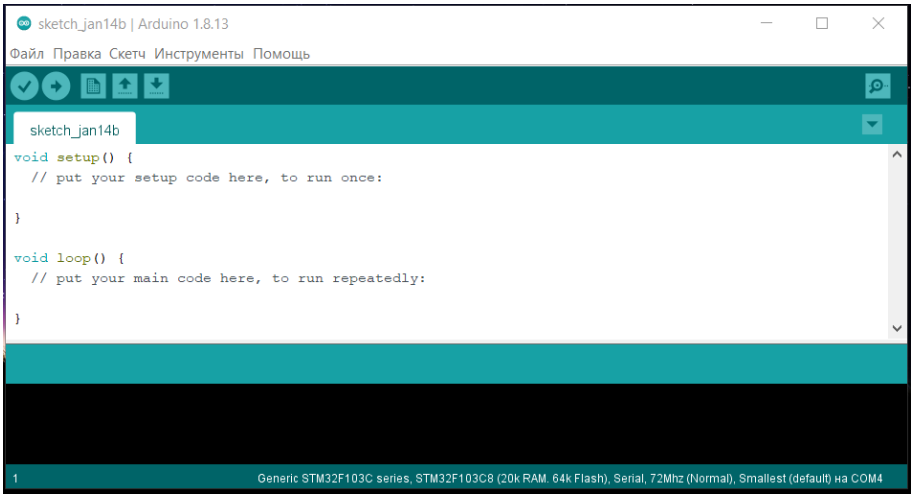


Рисунок 1.12 – Новый проект

После того как проект был создан, необходимо дать ему название и сохранить его, для этого необходимо нажать кнопку сохранить (рисунок 1.13) или нажать сочетание клавиш Ctrl + S:

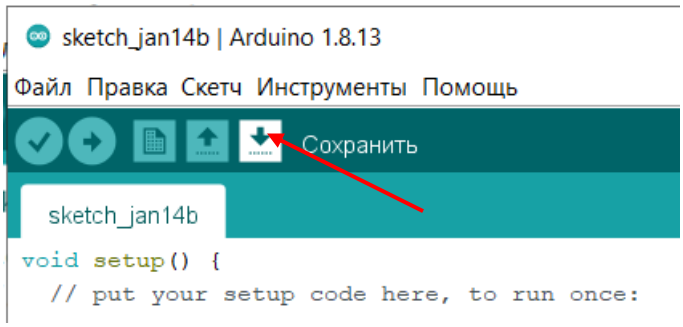


Рисунок 1.13 – Сохранение проекта

Сохраните проект в папке Arduino (рисунок 1.14):

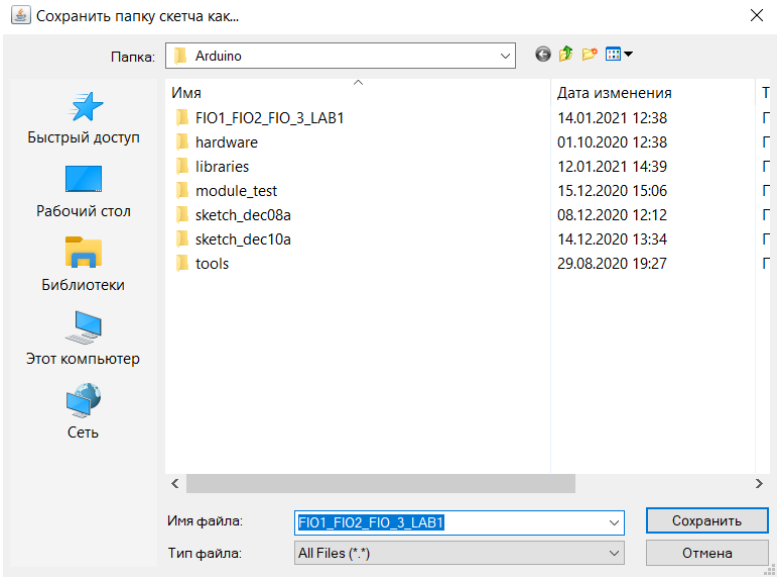


Рисунок 1.14 – Сохранение проекта

Для названия проекта можно следует выбрать следующую формулу: ФИО1_ФИО2_ФИО3_LABN

Где:

ФИО1, ФИО2, ФИО3 – инициалы студентов, которые выполняют лабораторную работу;

LABN – где N – номер лабораторной работы

Далее необходимо выбрать контроллер, который вы хотите запрограммировать и выставить его настройки. Для этого перейдите во вкладку «Инструменты» (рисунок 1.15).

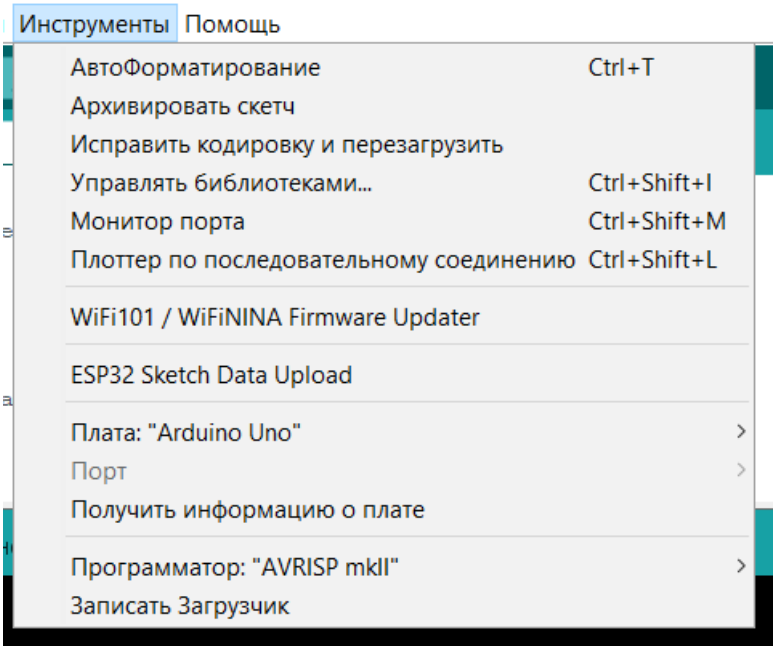


Рисунок 1.15 – Вкладка инструменты

Далее выберите плату Generic STM32F103C series. Процесс выбора платы представлен на рисунке 1.16.

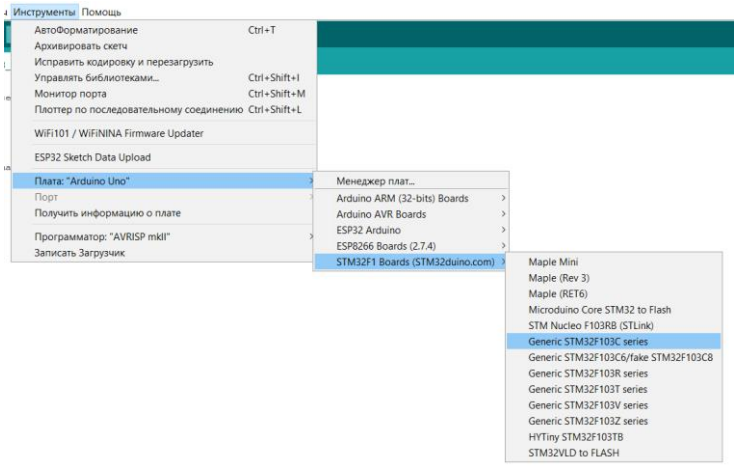


Рисунок 1.16 – Процесс выбор платы

Далее необходимо выставить настройки платы согласно рисунку 1.17:

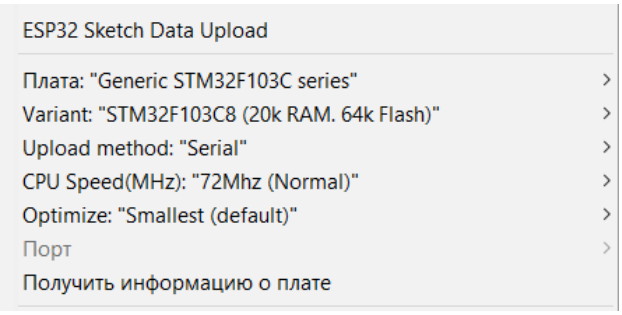


Рисунок 1.17 – Настройка платы

Далее необходимо выбрать COM-порт к которому подключена плата (рисунок 1.18).

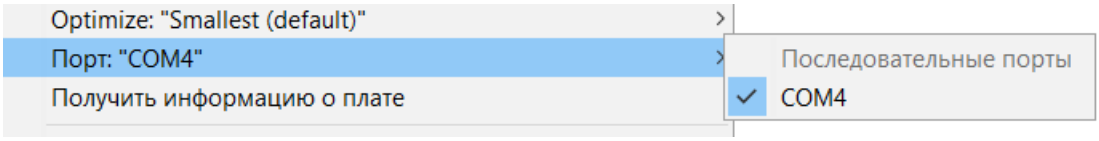


Рисунок 1.18 – Выбор COM порта

Номер COM-порта может отличаться на разных компьютерах, для того чтобы выбрать верный, обратитесь к преподавателю

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

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

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, OUTPUT);  //Перевод пина PC13 в "выходной" пин
}
```

Здесь в функцию pinMode()передается 2 параметра:

PC13 – название ножки к которой подключен светодиод. На плате TUSUR IoT Board светодиоды подключены к нескольким ножкам, но для теста будем использовать светодиод, который подключен к ножке PC13.

OUTPUT – состояние пина. Output с английского выхол. Это означает что пин переходит в выходной состояние, что в свою очередь означает, что пин будет **посылать**, а не принимать сигнал. Существует еще также состояние пина INPUT, 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*/
```

Используйте комментарии в своем коде для того, чтобы лучше понимать работу программы в данный момент и через несколько недель или месяцев!

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

Для того, чтобы маргать диодом, 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, HIGH); //Подача высоко напряжения на пин
  delay(1000); // Задержка на 1000 мс
  digitalWrite(PC13, LOW); // Подача низкого напряжения на пин
  delay(1000); // задержка еа 1000 мс
}
```

digitalWrite(PC13, HIGH); - функция, которая подает на определенный пин (в нашем случае PC13) напряжение HIGH или LOW, то есть либо высокое напряжение (логическая единица), либо низкое напряжение (логический ноль).

delay(1000); - функция, которая дает команду контроллеру «подождать» некоторое время (в нашем случа 1000мс). 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 1.19):

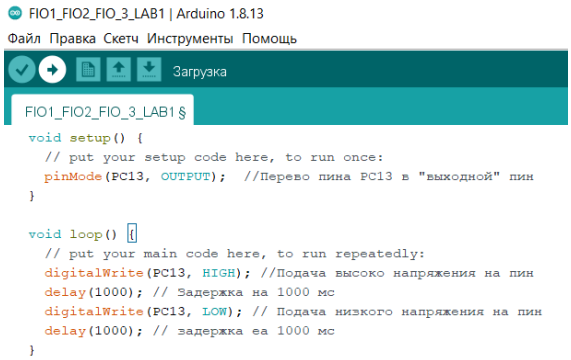


Рисунок 1.19 – Загрузка скетча в микроконтроллер

После нажатия кнопки начнется процесс загрузки кода в микроконтроллер (рисунок 1.20):

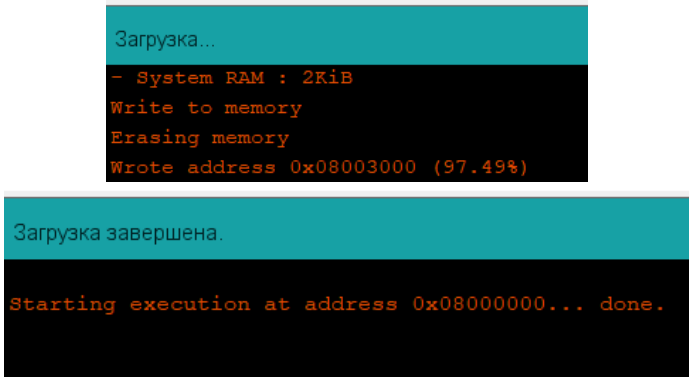


Рисунок 1.20 – Загрузка программы в микроконтроллер

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

Лабораторная работа № 2, 3
Введение в булеву алгебру

Цель**:** Познакомиться с двоичной системой счисления и основными логическими операциями

Задачи:

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.

Теоретическая часть

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. И 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.

Единица 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.

$52 / 2 = 26, остаток(0)$

$26 / 2 = 13, остаток(0)$

$13 / 2 = 6, остаток(1)$

$6 / 2 = 3, остаток(0)$

$3 / 2 = 1, остаток(1)$

(1)

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 2.1 – Преобразование двоичного числа в десятичное

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	$32 + 16 + 4 = 52$					

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 и False.

Для представления в двоичной системе их можно записать как:

1 – True.

0 – False.

True означает Правда.

False означает Ложь.

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 2.2):

Таблица 2.2 – 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	<=	$a <= b$

Результатом сравнения двух переменных может быть выражение либо True(1), либо False(0).

Для примера сравним несколько выражений (таблица 2.3).

Таблица 2.3 – Примеры использования операторов сравнения

25	==	43	=	False(0)
34	!=	53	=	True(1)
2	>	21	=	False(0)
3	<	23	=	True(1)
4	>=	4	=	True(1)
5	<=	10	=	True(1)

Также в программировании присутствуют логические операторы, такие как OR(ИЛИ), AND(И), NOT(НЕ).

В языке СИ они представлены следующим образом (таблица 2.4):

Таблица 2.4 – 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(ИЛИ) (таблица 2.5):

Таблица 2.5 – Таблица истинности оператора OR

False(0)	||	False(0)	=	False(0)
False(0)	||	True(1)	=	True(1)
True(1)	||	False(0)	=	True(1)
True(1)	||	True(1)	=	True(1)

Можно легко заметить, что истинным(True) результатом при использовании логического оператора OR будет являться случаи, когда одно или оба из выражений равняются 1 (True).

Рассмотрим работу оператора AND(И) (таблица 2.6):

Таблица 2.6 – Таблица истинности оператора OR

False(0)	&&	False(0)	=	False(0)
False(0)	&&	True(1)	=	False(0)
True(1)	&&	False(0)	=	False(0)
True(1)	&&	True(1)	=	True(1)

Можно легко заметить, что истинным (True) результатом при использовании логического оператора AND будет являться только один случай, когда **оба** из выражений равняются 1 (True). То есть И первое И второе выражения

Рассмотрим работу оператора NOT(НЕ) (таблица 2.7):

Таблица 2.7 – Таблица истинности оператора OR

!	False(0)	=	TRUE(1)
!	True(1)	=	False(0)

Из таблицы видно, что оператор NOT, просто инвертирует полученный аргумент, Если НЕ False, то True. Если не True, то False.

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

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.

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

Как переключатель, так и кнопка могу находится 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, то 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() {
  // put your setup code here, to run once:
  ***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:

```
void setup() {
  // put your setup code here, to run once:
  pinMode(PC13, OUTPUT);  //Перевод пина PC13 в "выходной" пин
  pinMode(PB4, INPUT);    //Перевод пина PB4 во "входной" пин
}
```

INPUT – в переводе с английского «Вход».

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

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

```
void loop() {
  // put your main code here, to run repeatedly:
  bool but = digitalRead(PB4); // 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(PB4) – функция, которая считывает значение с пина PB4 (к нему подключена кнопка). Дословно можно перевести данную функцию, как «цифровое считывание». С помощью данной функции можно считать два состояния высокое напряжение(1) и низкое напряжение(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 с собой всего лишь ключ, следовательно, нужно построить схему включения данной кнопки (рисунок 2.1):

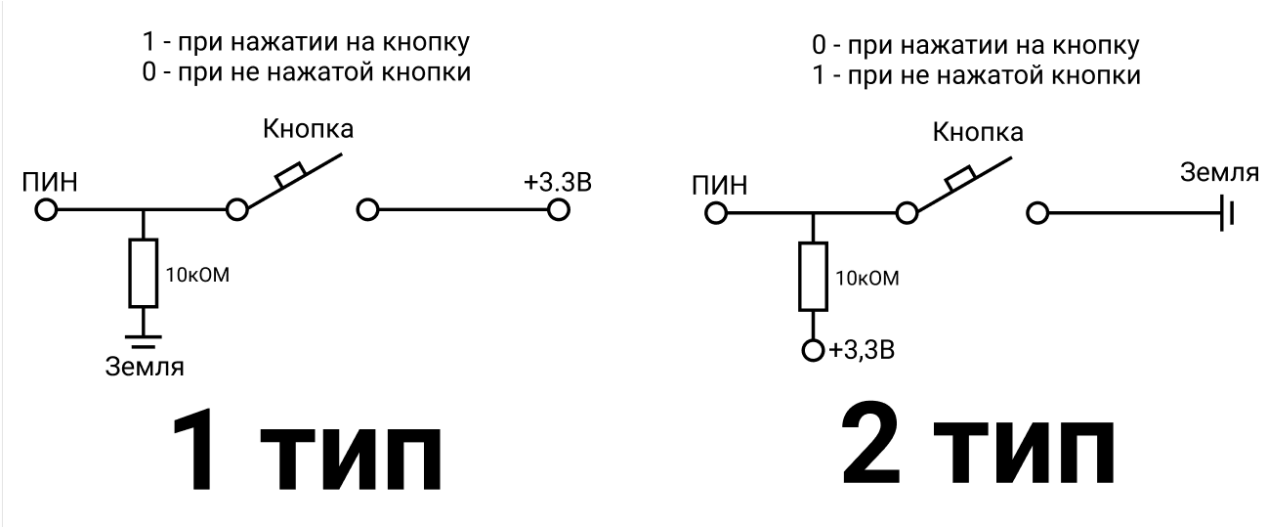


Рисунок 2.1 – Схема подключения кнопки к пину

Существует 2 типа (варианта) подключения кнопки к пину.

Рассмотрим 2 тип:

За счет бесконечного сопротивления при разжатой кнопки на вход пина, будет подаваться напряжение в 3,3В, что соответствует логической единице (1). Как только кнопка будет нажата, то между землей и пином будет нулевой напряжение, следовательно, на пине будет 0В, что соответствует логическому (0).

На плате TUSUR IoT Board реализован 1 тип подключение.

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 выставлено высокое напряжение (1).

Далее нам необходимо на пин с диодом (PC13) подать напряжение в соответствие с напряжением, которое мы получили с пина, на котором установлена кнопка (PB4).

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

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

  bool but = digitalRead(PB4);
  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 (0) и светодиод будет выключен.

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

Завершенный программный код, выглядит следующим образом:

```
void setup() {
  // put your setup code here, to run once:
  pinMode(PC13, OUTPUT);  //Перевод пина PC13 в "выходной" пин
  pinMode(PB4, INPUT);    //Перевод пина PB4 во "входной" пин
      }

void loop() {
  // put your main code here, to run repeatedly:
  bool but = digitalRead(PB4); // Запись в переменную but значения с пина PB4
  digitalWrite(PC13, but); //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 работают 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.

Изучение работы логических операторов

В прошлой работе мы научились подключать кнопки и переключатели. Теперь на основе полученных знаниях, 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.

Также в теоретическом материале вы можете увидеть еще 2 логических оператора:

И(&&) и ИЛИ(||)

Для данных операторов необходимо минимум два операнда. Пример: 1||0=1. Между операндами 1 и 0 выполняется операция ИЛИ результатом которой, является 1

То есть записи &&1 приведет к ошибке!

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 = digitalRead(PC14);
  bool switch2 = digitalRead(PC15);
  bool result = switch1 {лог. операция} switch2
 }
```

Далее необходимо выполнить логическую операцию между данными переменами 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 2.8;

2) Вставьте логическую операцию И, загрузите программу в микроконтроллер, запишите результаты работы в таблицу 2.8.

Таблица 2.8 – Результаты работы программы

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

Сравните полученные результаты с таблицами 2.5 и 2.6.

Лабораторная работа №4
Основы программирования микроконтроллеров

Цель работы: Получить базовые навыки по программированию микроконтроллеров

Задачи:

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.

Теоретическая часть

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() {
  // put your main code here, to run repeatedly:
  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(False), либо 1(True).

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 равняется минимум 1 байт.

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 возможность программировать первые микроконтроллеры, используя Arduino IDE – 2005 год. Для хранения такой переменной понадобится больше, чем 1 байт, а если быть точнее, то 11 бит [11111010101]. Получим данной число в байтах, для это разделим на 8:

$$\frac{11}{8} = 1,325 \text{ байт.}$$

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 4.1):

Таблица 4.1 – Основные типы данных в Arduino IDE

Название	Альт.название	Вес	Диапазон	Особенность
boolean	bool	1 байт	0 или 1, **true** или **false**	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.

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 выполнится часть кода, которая меняет параметры основной программы.

Для создания условия в языке Си использует 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 4.1):

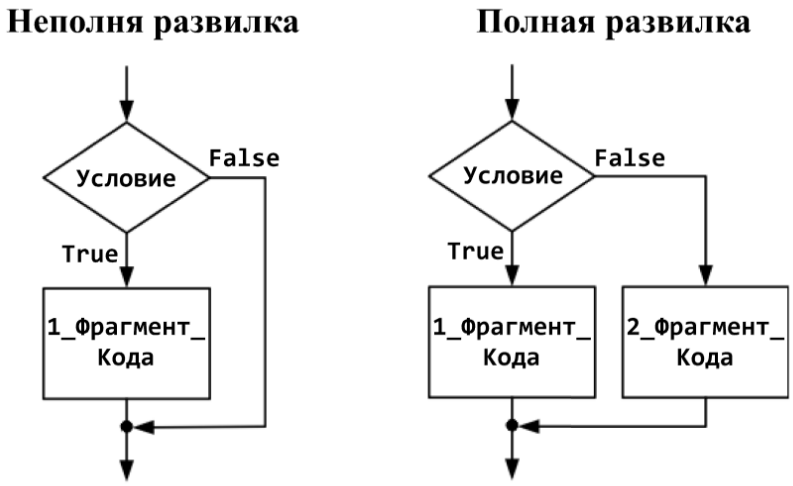


Рисунок 4.1 – Блок схемы

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, то тогда при неверном условии в операторе if, ничего выполняться не будет.

При чем, если перед else несколько операторов if, то оператор else будет относится 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 условий 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
        }
```

Циклические операторы

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

Называется он loop().

Но, что, если понадобится создан 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, который будет выполнять программа при использовании данного цикла (рисунок 4.2):

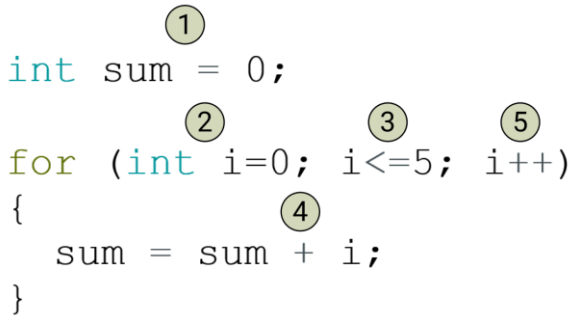


Рисунок 4.2 – цикл for

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 использует операция инкрементирования.

Инкремент – операция которая выполняет 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 позволяет 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. При желании его работу можно изучить самостоятельно.

Практическая часть.
Применение условных и циклических операторов в Arduino IDE

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 сигналом (что это такое мы более подробно разберем в дальнейших работах).

Простыми словами ШИМ сигнала позволяет имитировать аналоговый сигнал, которые 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

После того как проект будет создан и настроен, то 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 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, которые подключен к пину, у которого есть возможность генерировать ШИМ сигнал. Таким и является пин PB1.

Но данный пин подключен через мультиплексор, следовательно, в меню периферии необходимо будет выбрать RGB светодиод (рисунок 4.3):

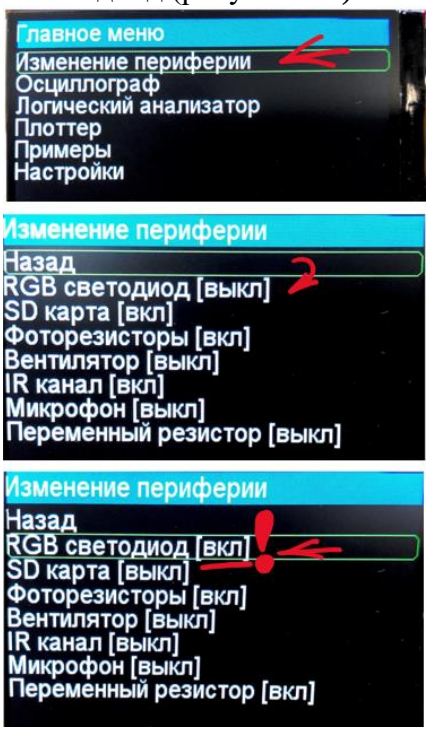


Рисунок 4.3 – Выбор RGB светодиода

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() {
```

```
 pinMode(PB1, OUTPUT);
}
void loop() {
  analogWrite(PB1, {Уровень ШИМ сигнала});
  delay(1000);
}
```

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() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(PB1, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
  for(int i=1;i<255;i++)
  {
    analogWrite(PB1, i);
    delay(5);
  }
}
```

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() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(PB1, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
  for(int i=1;i<255;i++)
  {
    analogWrite(PB1, i);
    delay(5);
  }
```

```
{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)

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 (PB1, i); 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 = digitalRead(PB4);
  if(but == 1) // если кнопка нажата
  {
    i = i + 5; // Увеличиваем переменную i на 5
  }
  if(i >= 255) // если i больше 255
  {
    i = 0; // обнуляем
  }
  delay(100);
  analogWrite(PB1, i);
}
```

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 отличие 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 – 255, следовательно сделаем проверку, которая обнулит переменную I при преодолении значения 255.

В конце программы записываем полученное значение в функцию: analogWrite(PB1, i);

И делаем небольшую задержку для того, чтобы отследить изменения.

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.

Лабораторная работа №5

Основы программирования микроконтроллеров

Цель работы: Получить базовые навыки по программированию микроконтроллеров

Задачи:

1) Изучить принципы работы функций
2) Научиться передавать данные в функцию и принимать от нее значение
3) Понять каким образом работает подключение библиотек в Arduino IDE
4) Изучить работу скрипта #define

Теоретическая часть

Принцип работы функций

В ходе предыдущих лабораторных работ мы уже сталкивались с функциями.

К примеру:

```
pinMode();
digitalRead();
digitalWrite();
analogWrite();
//и другие…
```

Функции – это неотъемлемая часть программирования на языке Си. Так как:

Функции созданы для того, чтобы упростить работу программиста.

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.

Но функции бывают разные. Бывают функции, которые возвращают значение, а бывают функции без возврата значений.

Можно посмотреть данное отличие на примере функций, которые мы уже использовали:

pinMode(); - Инициализирует тип пина. Ничего не возвращает

digitalRead(); - Считывает цифровое значение с указанного пина. И возвращает считанное значение

digitalWrite(); - Устанавливает либо высокое либо низкое напряжение на пине. Ничего не возвращает

analogWrite(); - посылает ШИМ сигнал на выбранный пин, с выбранным величиной рабочего цикла. Ничего не возвращает

Также для работы некоторых функций им необходимо сообщить некоторые параметры. Количество необходимых параметров указывается при объявлении функции. Синтаксис объявления функции рассмотрим чуть позднее.

То есть существуют функции, которые принимают некоторые параметры для того, чтобы внутри себя с ними как-либо взаимодействовать.

Например, для того чтобы считать цифровое значение с определенного пина микроконтроллера, необходимо сообщить его функции для того, чтобы ей был известен адрес пина, с которого нужно считать и вернуть значение.

Аргументы, которые необходимо сообщить функции, указываются в круглых скобках:

digitalRead(PB4);

где PB4 – это пин, с которого нужно считать значение.

Также существуют функции, которые принимают несколько значений аргументов.

К примеру, функция для записи цифрового значения:

Если функция принимает несколько аргументов, то необходимо их указывать через запятую.

```
digitalWrite(PB4, HIGH);
```

Где:

PB4 – это пин, на который нужно записать цифровое значение,

HIGH – это значение, которое необходимо записать (1).

Каким образом узнать какое количество аргументов и какого типа нужно сообщать функции?

1) Изначально в курсе лабораторных работ мы использовали стандартные функции Arduino IDE. К данным функциям предоставляется стандартная документация Arduino IDE. В данной документации описаны все функции. Значения, которые они принимают и значения, которые они возвращают. Стандартный справочник можно найти в интерфейсе Arduino IDE, для это необходимо перейти в меню «Помощь» и выбрать «Справочник» (рисунок 5.1);

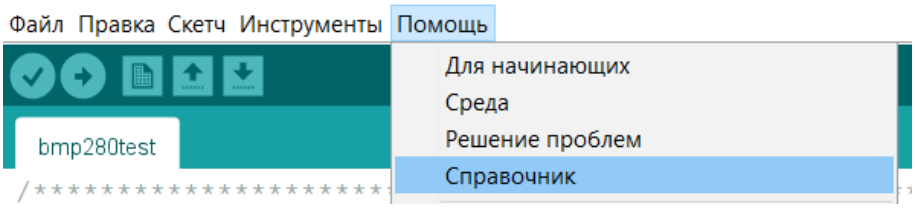


Рисунок 5.1 – Справочник в Arduino IDE

2) Часто при программировании микроконтроллеров необходимо воспользоваться уже существующими функциями, 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 подключения библиотеки используется скрипт #include <название библиотеки>

Но помимо использования уже существующих можно писать свои собственные функции, для этого надо 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
 }
```

Давайте разберем на примере.

В прошлой лабораторной работе мы написали код, который позволяет плавно включать и выключать диод. Напишем функцию, которая выполняет данную операцию.

Что нужно знать, перед тем как написать функцию?

1)Какое количество и каких значений она будет принимать?

2) Будет ли она возвращать значения, если да то какие?

Так как мы пишем функцию, которая выполняет алгоритм, то нет нужды возвращать некоторое значение.

1) Для того, чтобы плавно включать и выключать светодиод, нам необходимо принять пин, к которому подключен светодиод, следовательно, нам как минимум придется сообщить функции к какому пину подключен светодиод;

2) Для это воспользуемся типом данных void. Это тип данных «пустой», который не хранит в себе ничего и используется для создания функций, которые не возвращают значений.

Приступим к написанию функции:

```
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() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(PB1, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
  smooth_blink(PB1);
}
```

Здесь мы можем видеть, что функция объявляется всегда прежде, чем она будет использоваться. Связанно это с тем, что в память устройства переменные и имена функций добавляются последовательно. То есть у вас не получится сначала воспользоваться переменной, а потом объявить её, это же правило справедливо и для функций.

Можете попробовать загрузить данный код на плату и проверить его работоспособность.

Для того, чтобы закрепить навык написания функций. Добавьте в функцию аргумент, который будет менять задержку в каждом из циклов.

Практическая часть

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). И 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 это напишите программу, которая:

Последовательно будет подавать сначала высокое, потом низкое напряжение 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 RGB_R PB1
#define RGB_G PB9
#define RGB_B PB8
#define diod PC13
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(RGB_R, OUTPUT);
  pinMode(RGB_G,OUTPUT);
  pinMode(RGB_B, OUTPUT);
  pinMode(diod,OUTPUT);
}
// the loop function runs over and over again forever
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 убераем его) на каждый пин
  // К которому подключен RGB светодиод
  {ВАШ КОД}
}
```

Здесь вы можете заметить пример использования скрипта #define

#define – Работает по принципу «Найти и заменить». То есть перед выполнением программы в данном случае:

#define RGB_R PB1

Если в тексте кода встретиться RGB_R то эта часть кода заменится на PB1

Использовать данную конструкцию очень удобно, так как если поменять что-то в данном скрипте он изменит это во всем коде. Сейчас мы с вами в этом убедимся.

Напишите в поле остальную часть программы, которая подает напряжение на каждый из диодов RGB светодиода.

Вместо написания своего пина в функцию digitalWrite();

Используйте имена из скрипта #define

После написания кода, запустите программу и запишите к какому пину подключен какой светодиод, далее исправьте пины в верху в #define и запустите код. Только теперь вы уже заранее должны знать порядок включения диодов.

После выполнения данной работы продемонстрируйте её преподавателю.

Далее необходимо написать функцию, которая будет включать определенный цвет

Для основы своей программы можете взять следующий код:

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

void RGBgo(byte R_pin, byte G_pin, byte B_pin, byte R, byte G, byte B)
{
{ВАШ КОД, КОТОРЫЙ ПОДАЕТ ШИМ СИГНАЛ СО ЗНАЧЕНИЯ R, G, B на три пина}

}
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(RGB_R, OUTPUT);
  pinMode(RGB_G,OUTPUT);
  pinMode(RGB_B, OUTPUT);
  pinMode(diod,OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
   RGBgo({АРГУМЕНТЫ ФУНКЦИИ});
}
```

Дополните программный код, чтобы у вас работала функция и с помощью нее попробуйте отобразить 3 цвета. Красный, Зеленый, Синий последовательно с перерывом в 1 секунду.

Если все получилось верно, то сформируйте цвет одежды, каждого участника группы также с перерывом в 1 секунду на данном светодиоде, с помощью вашей функции.

Подсказка: можно воспользоваться паинтом, сформировав там нужный цвет и посмотрев его код (рисунок 5.2):

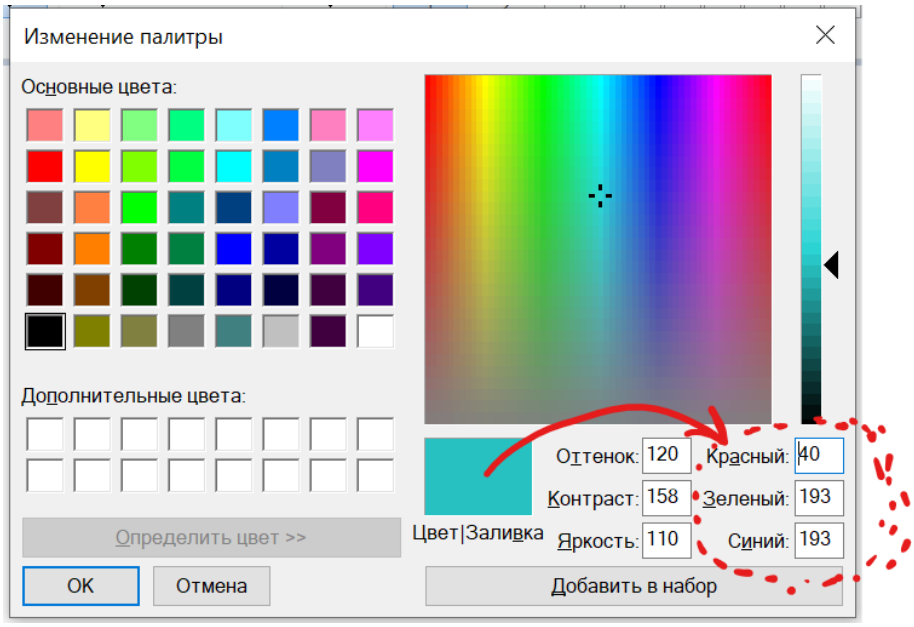


Рисунок 5.2 – Получение значений RGB с помощью Paint

Лабораторная работа №6
Изучение работы АЦП (Аналогово-Цифровой-Преобразователь)

Цель работы: Изучение принципов работы Аналогово Цифрового Преобразователя и его применение на практике

Задача:

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.

Теоретическая часть

Принцип работы аналогового-цифрового преобразователя

Как мы с вами выяснили ранее в цифровых системах передается цифровой сигнал.

Но, что значит цифровой сигнал?

При каких условиях сигнал можно назвать цифровым**?**

Цифровой сигнал – это такой сигнал, который является счетным (дискретным) как по времени, так и по амплитуде.

Снимая напряжение с некоторых датчиков, мы получим непрерывный сигнал, что в свою очередь означает, что данный сигнал определен в каждый момент времени его существования, в то время как дискретный сигнал, определен только в определенные промежутки времени (отсчёты).

Графически данные сигналы можно представить следующим образом (рисунок 6.1):

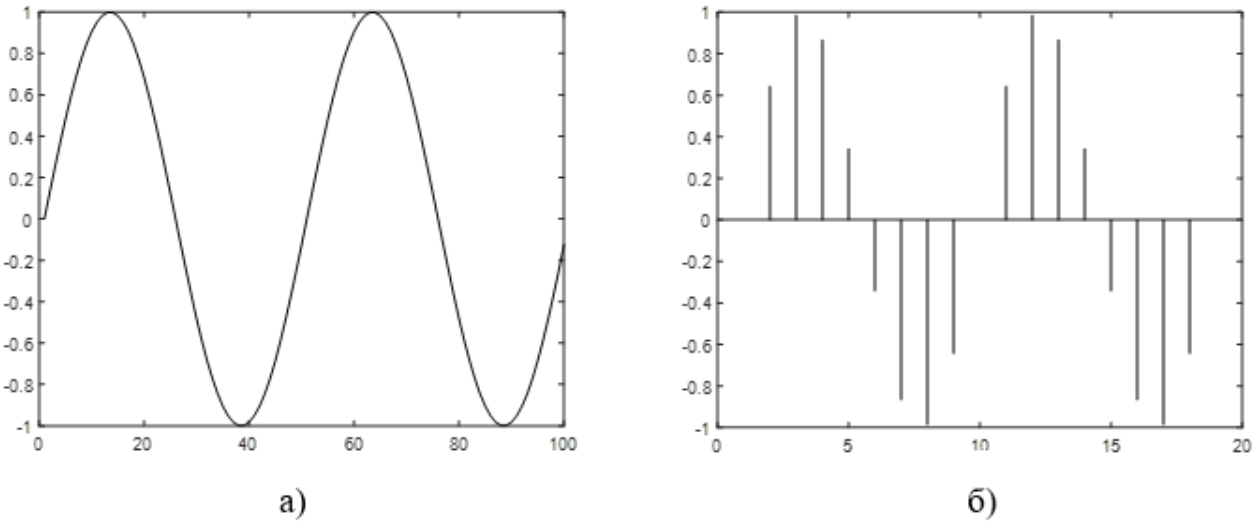


Рисунок 6.1 – а) Непрерывный (аналоговый сигнал); б) Дискретный сигнал

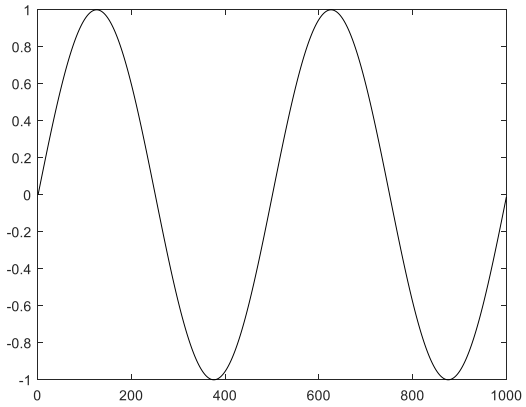
Для того, чтобы получить цифровой (дискретный) сигнал из аналогово (непрерывного) сигнала необходимо выполнить следующие операции:

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

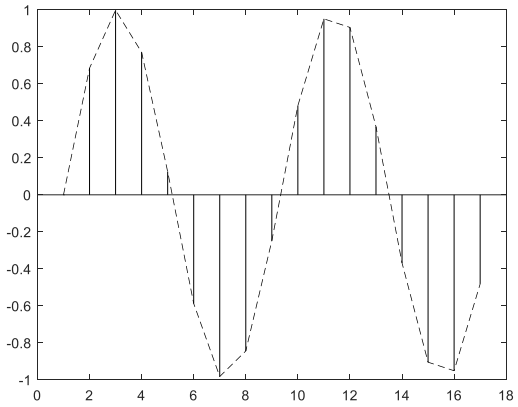
Дискретизация по времени

Даная операция представляет собой выбор отсчетов аналогового сигнала с заданной частотой, то есть при поступлении аналогового сигнала можно снимать значение каждый определенный отсчет.

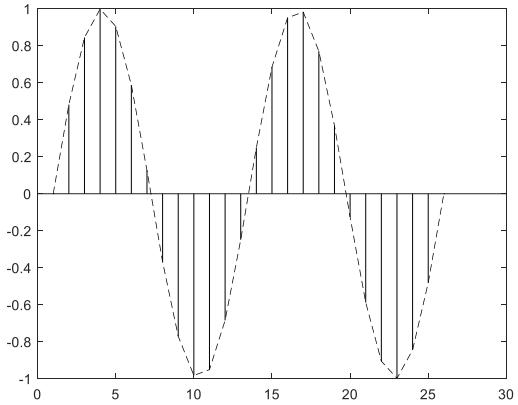
На рисунке 6.2 представлен пример дискретизации сигнала по времени со следующими выборами отсчетов (каждый 60, 40, 20 отсчёт). В качестве отсчёта может быть определенная единица времени. (Пример: каждый 20 мс, 40мс, 60мс).



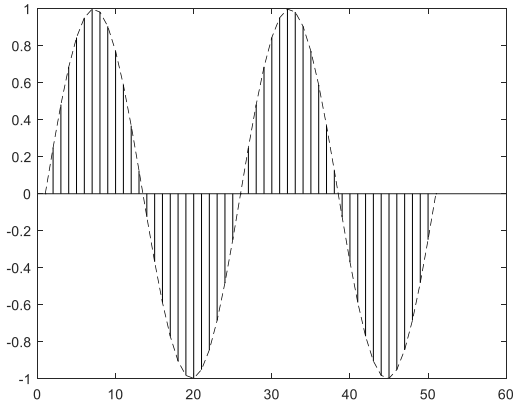
Исходный сигнал



Каждый 60 отсчёт



Каждый 40 отсчёт



Каждый 20 отсчёт

Рисунок 6.2 – 1Пример дискретизации сигнала по времени

Здесь мы можем заметить, что чем чаще мы берем отсчёты, тем более точный сигнал мы получаем на выходе.

Но чем больше отсчётов мы берём, тем больше отсчетов сигнала придется хранить в памяти цифрового устройства. Поэтому всегда нужно соблюдать баланс между точностью сигнала и количеством места, которое будет занимать оцифрованный сигнал.

Квантование по уровню.

После дискретизации во времени наш сигнал стал счетным во временной области, то есть стал существовать только в определенные моменты времени.

Но значение данных отсчетов до сих пор является непрерывной величиной, следовательно, нам нужно добавит уровни, по которым мы будем записывать сигнал в память устройства.

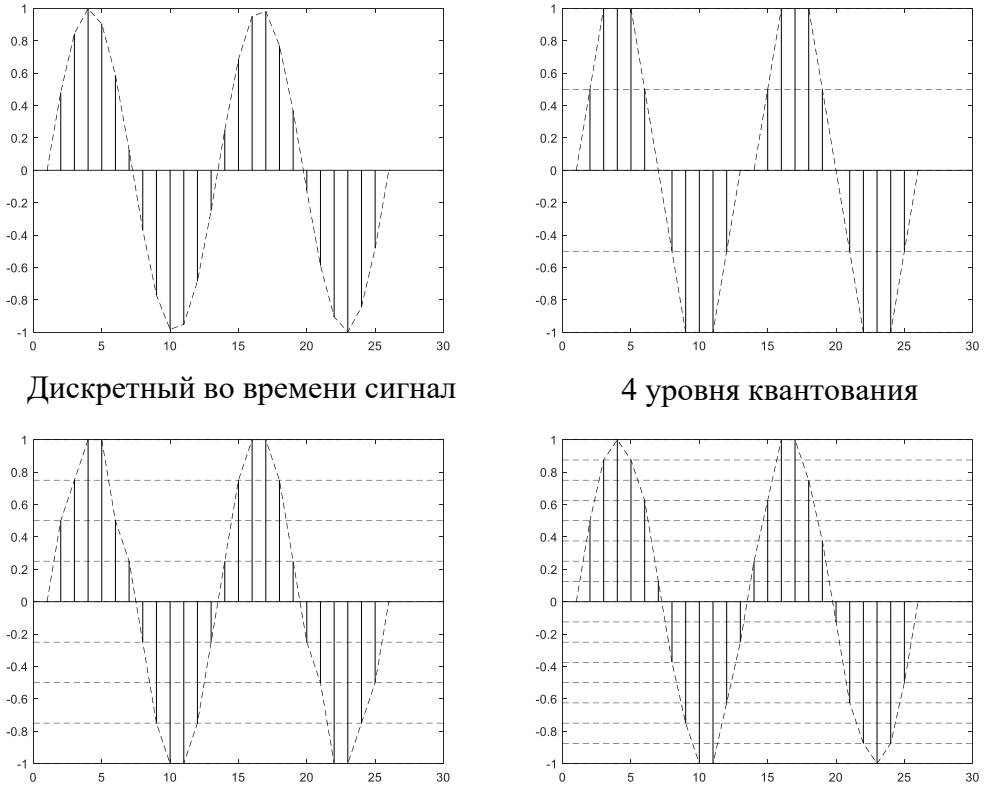
Количество уровней определяется разрядностью АЦП. Допустим мы захотим сделать 16 уровневый АЦП. Что это будет означать.

Это означает, что наш сигнал (который на данный момент может принимать **любое** значения от 0 до 1 включительно) будет иметь возможность принадлежать только одному из 16 уровней в зависимости от того к какому уровню отсчёт ближе всего. То есть сигнал сможет принимать значения от 0 до 1 с шагом $(1-(-1))/16 = 2/16 = 1/8 = 0{,}125$, а именно:

-1 -0.875 -0.75 -0.625 -0.5 -0.375 -0.25 -0.125 0 0.125 0.25 0.375 0.5 0.625 0.75 0.875 1

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

Рассмотрим, как влияет количество уровней на точность сигнала (рисунок 6.3):



Дискретный во времени сигнал | 4 уровня квантования

8 уровней квантования | 16 уровней квантования

Рисунок 6.3 – Влияние количества уровней на точность сигнала

Как мы можем видеть из рисунка выше, чем больше количество уровней, тем больше точность сигнала на выходе АЦП. Но в тоже время, как и с частотой дискретизации сигнала при дискретизации во времени, так и с количеством уровней, чем больше значений мы храним, тем больше места занимает оцифрованный сигнал в памяти микроконтроллера.

Кодирование.

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

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

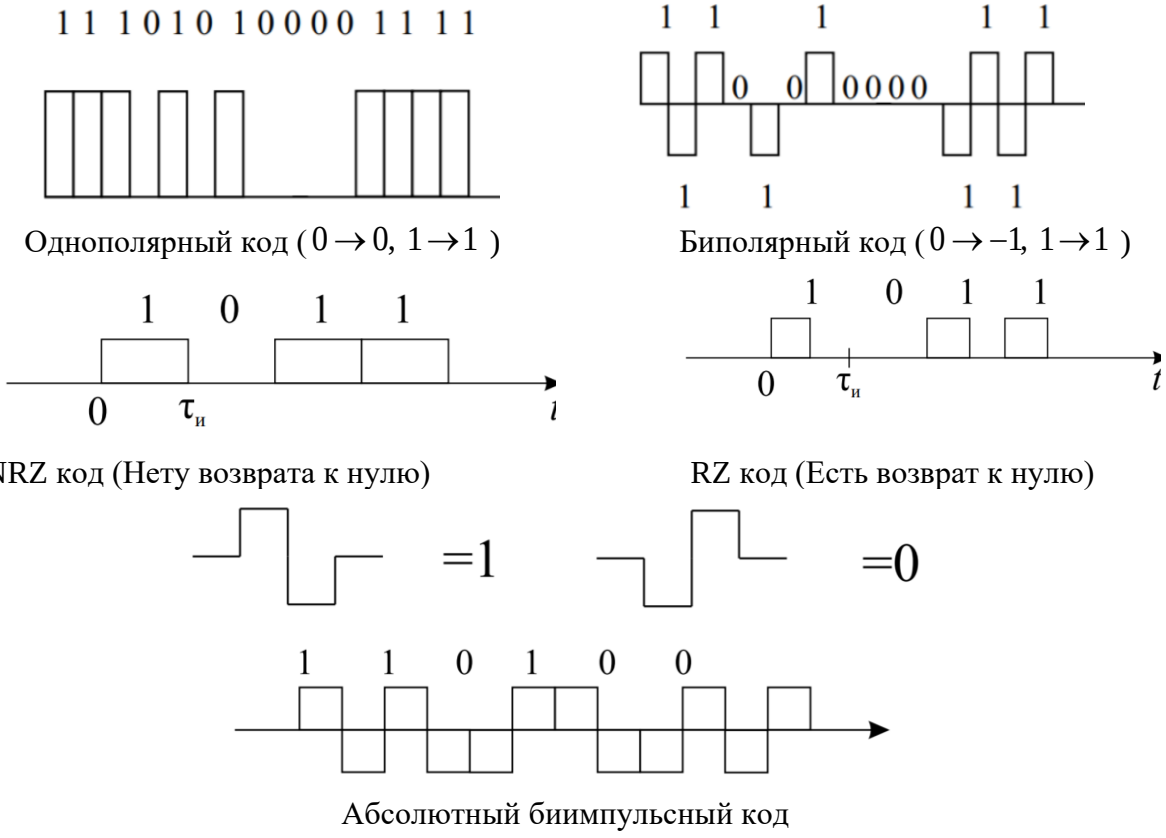
Для 8 уровней квантования потребуется $\log_2 8 = 3$ бит информации, то есть уровни будет представлять собой следующие последовательности 0 и 1 (таблица 6.1):

Таблица 6.1 – Кодирование 8 уровней

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

Нетрудно заметить, что получение результаты могу представлять собой
- чередование нулей и единиц;
- последовательность одних единиц;
- последовательность одних нулей.

А передавать данные последовательности можно используя один из видов кодирования. К примеру, здесь представлены самый известные виды кодирования (рисунок 6.4):



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

Биполярный код ($0 \to -1,\ 1 \to 1$)

NRZ код (Нету возврата к нулю)

RZ код (Есть возврат к нулю)

Абсолютный биимпульсный код

Рисунок 6.4 – Примеры кодирования

Датчики и обработка данных с датчиков

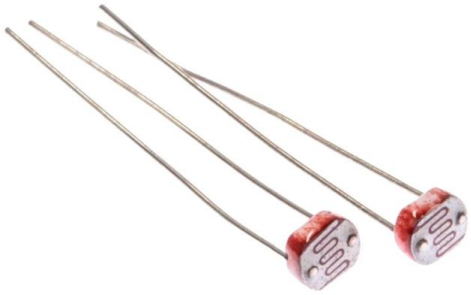
Датчик – это механизм, предназначенный для измерения какой-нибудь величины и обработки результата измерения. Схема датчика генерирует сигнал в удобной для передачи форме, дальше сигнал обрабатывается или хранится. Сегодня датчики применяются везде, от вашего смартфона, до разных сфер медицины и промышленности.

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 (рисунок 6.5):

Конденсаторный микрофон (Датчик звука)



Фоторезистор (Датчик света)

Переменный резистор (Делитель напряжения)

Рисунок 6.5 – Датчики и модули для преобразования напряжения 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.

Практическая часть

Получение данных с аналоговых датчиков в Arduino IDE

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

Переменный резистор (ПР) подключен к пину PB0.

Также перед началом работы запишите в тетради или создайте таблицу 6.2:

Таблица 6.2 – Экспериментальные данные

Максимальное значение полученное с АЦП при использовании ПР (PB0)	
Количество уровней АЦП	уровней
Разрядность АЦП	бит
Значение микрофона (PA3) в спокойном состоянии	
Минимальное значение микрофона (PA3) при громких звуках	
Значение освещения с 1 фоторезистора (PA1)	
Значение освещения с 2 фоторезистора (PA2)	
Значение освещения с 3 фоторезистора (PA3)	
Усредненное значение с 3 фоторезисторов	

Создайте новый проект и объявите необходимые переменные они все понадобятся нам входе работы:

```
 void setup() {
  // put your setup code here, to run once:
  /* Назначение всех пинов для исследования на ввход*/
Serial.begin(9600);
  pinMode(PA1, INPUT);
  pinMode(PA2, INPUT);
  pinMode(PA3, INPUT);
  pinMode(PB0, INPUT); //Переменный резистор
  /*-------------------------------------------*/
  /* Назначение свитча на вход*/
  pinMode(PC15, INPUT);

  /*Инициализация пинов с RGB светодиодом*/
  pinMode(PB1, OUTPUT);
  pinMode(PB9, OUTPUT);
  pinMode(PB8, OUTPUT);
 }
```

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

Данная функция инициализирует определенные порт для передачи данных по UART. Это необходимо для того, чтобы отправлять полученные данные на компьютер для того, чтобы их анализировать и делать выводы.

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 которые будет считывать значение со свитча. И программа будет работать только при включенном положении свитча.

```
void loop() {
 if (digitalRead(PC15)){ //Начало обработки данных при включенном свитче
  /*Сбор данных с переменного резистора*/
  float PR = analogRead(PB0);
  Serial.println(PR);
```

```
  /*--------------------------*/
  delay(5);
 }
}
```

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 6.6) или нажмите комбинацию клавиш Ctr+Shift+L:

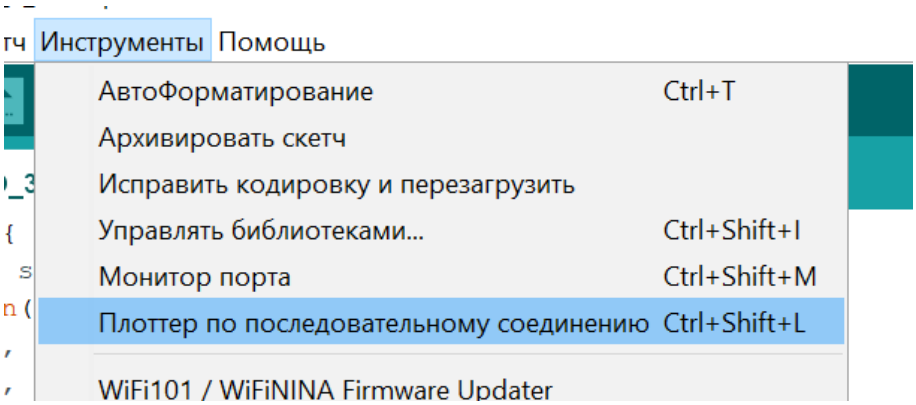


Рисунок 6.6 – Запуск плоттера

У вас должно открыться похожее окно (рисунок 6.7):

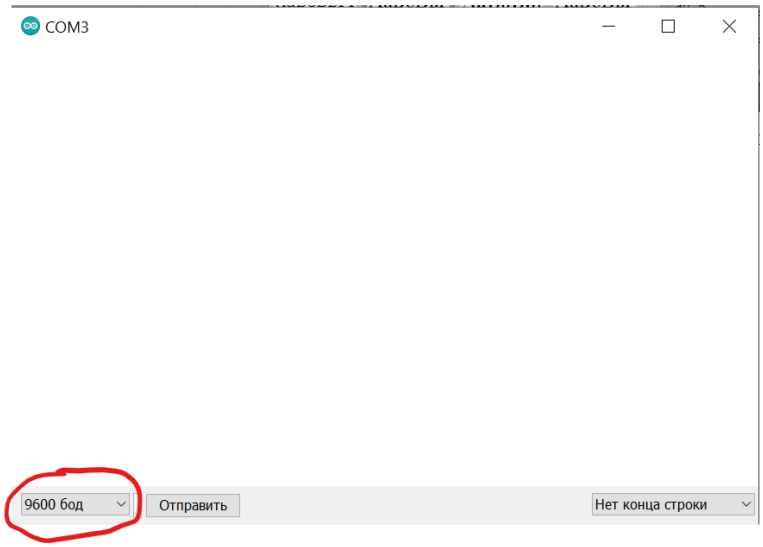


Рисунок 6.7 – Окно плоттера

Проверьте, скорость считывания должна быть 9600 бод!

Для того, чтобы пошли данные, надо установить свитч во включенное положение и должна появится подобная картина (рисунок 6.8):

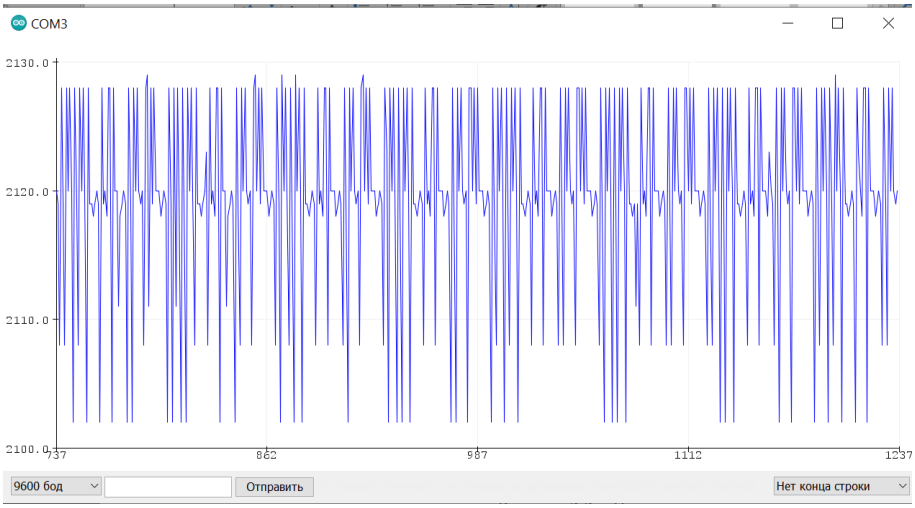


Рисунок 6.8 – Снятие данных с ПР

Но это не единственные способ того, как можно получить данные с COM порта в Arduino IDE. Также можно посмотреть эти данные в численном представление в мониторе COM-порта для этого нажмите клавиши Ctrl+Shift+M либо же нажмите на иконку лупы в правом верхнем углу (плоттер должен быть закрыт!). Откроется подобное окно, где можно увидеть цифровые данные (рисунок 6.9):

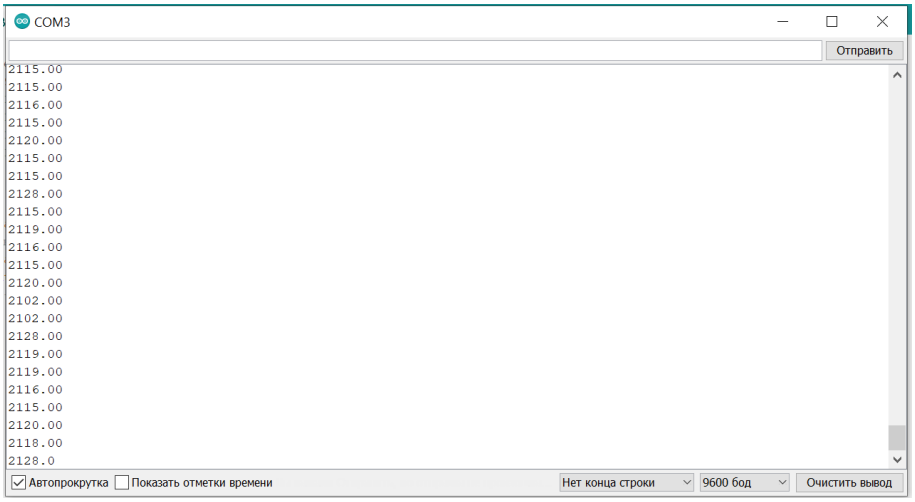


Рисунок 6.9 – Монитор COM порта

Далее изменяя скетч – заполните таблицу 6.2!

НЕ ЗАБУДЬТЕ ПЕРЕКЛЮЧИТЬ В МЕНЮ ПЕРЕФЕРИИ ЗНАЧЕНИЕ МИКРОФНА НА ВКЛ ПРИ НАДОБНОСТИ

Подсказки:

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

2) Записывайте усредненное значение с 3 фоторезистор в отдельную переменную – это понадобится в следующем задании.

После того как таблица будет заполнена, покажите данные преподавателю.

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 (digitalRead(PC15)){ //Начало обработки данных при включенном свитче
    /*Сбор данных с фоторезисторов*/
    {Считывание данных с 3-ех фоторезисторов и запись их в переменные}
   /*---------------------------*/
    float avrg = {Расчет среднего значения}; // Среденее значение с 3 датчиков

    uint8_t SignalToLed = ((analogRead(PB0) - avrg)/4096.0)*255; // Расчёт сигнала для
отправки

    analogWrite(PB1, SignalToLed);
    analogWrite(PB8, SignalToLed);
    analogWrite(PB9, SignalToLed);
    delay(5);
  }
}
```

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.

Лабораторная работа №7
Изучение интерфейсов. UART

Цель работы: изучить способы передачи информации от контроллера контроллеру\компьютеру.

Задача:

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

1 Теоретическая часть

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.

Почему необходимо пользоваться именно интерфейсами для передачи данных?

Существует большое количество способов с помощью, которых мы можем передавать данные большое множество 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 **с**тандарты (набор правил) по которым осуществляется передача данных между контроллерами – интерфейсы.

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

- UART;
- I2C;
- SPI.

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 (рисунок 7.1):

Рисунок 7.1 – 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 линии становится доступной для следующих данных

Период сигнала T 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 направление (либо прием, либо передача), так и в двух направления сразу только являться как приемником, так и передатчиком, но для это понадобится использовать 2 провода (рисунок 7.2):

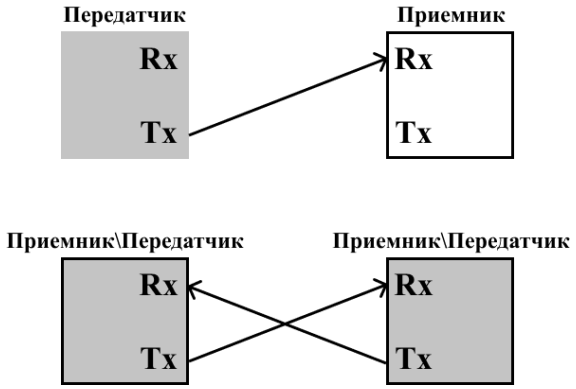


Рисунок 7.2 – Подключение последовательного интерфейса между двумя устройствами для приема\передачи

На плате TUSUR_IOT_BOARD связь между контроллерами ESP-32(отвечает за экран и меню) и STM (контроллер для программирования) также общаются с друг другом 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.

Практическая часть

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 значения (0 и 1).

Логический анализатор присутствует на плате TUSUR_IOT_BOARD для доступа к нему необходимо воспользоваться меню и выбрать пункт «Логический анализатор» (рисунок 7.3) и выбрать необходимый вид протокола для передачи данных (UART, I2C, SPI).

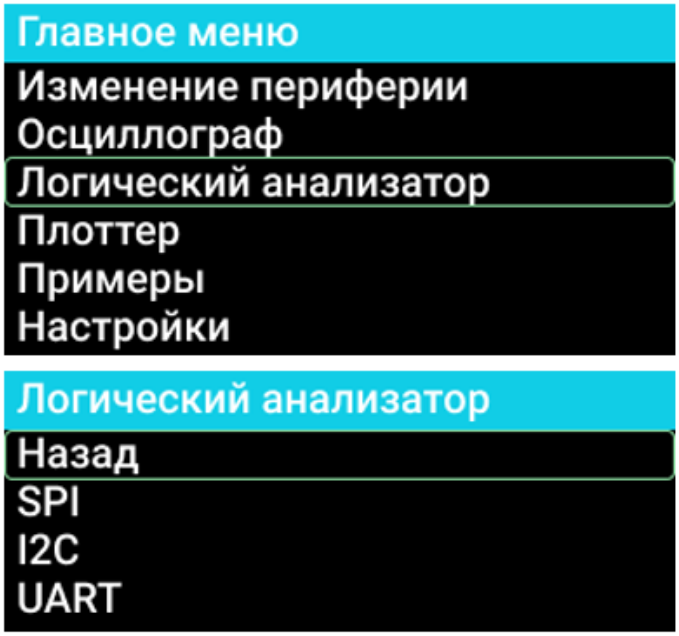


Рисунок 7.3 – Главное меню платы и логического анализатора

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

```
void setup() {
  Serial.begin(115200); // 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(i = 0; i <= 255; i++){ // Цикл, который меняет значение i от 0 до 255
     delay(3000); // Ожидание в течении 3 секунд (3000 миллисекунд)
     Serial.write(i);// Отправка значения, которое содержит переменная i по UART
  }
}
```

Скопируйте данный скетч и загрузите его в плату.

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. Действительно ли на устройство приходит сигнал, который меняет свое значение от 0 до 255 с шагом в 3 секунды?

Попробуйте передать свое собственное значение, для этого:

1) Закомментируйте цикл for с его фрагментом программы

2) Раскомментируйте строку 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.

Написание программы, которая имитирует работу «кодового» замка

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

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

2) Последовательный порт. С помощью которого будет производиться взаимодействие с контроллером;

3) Глобальная переменная (массив), которая будет хранить в себе значение пароля;

Для того, чтобы инициализировать данные компоненты нам понадобится реализовать следующий код.

```
/* Объявление скрипта define для записи данных пинов RGB светодиода*/
#define RGB_G PB1
#define RGB_R PB9
#define RGB_B PB8
/*--------------------------------------------------------------------*/
uint8_t password[8] = {0}; // Переменная, которая будет хранить в себе пароль

void setup() {
  Инициализируйте 3 пина RGB_G, RGB_R, RGB_B на выход
   Serial.begin(115200); // Инициализация передачи данных в UART интерфейс со скоростью 115200 кбит\с
   }
```

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 «Hello world».

Для этого вставьте строку «Hello world» 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 запуститься при первом включении платы и попросить ввести пароль для доступа к ней.

Для этого воспользуйтесь следующим программным кодом.

```
/* Объявление скрипта define для записи данных пинов RGB светодиода*/
#define RGB_G PB1
#define RGB_R PB9
#define RGB_B PB8
/*--------------------------------------------------------------------*/
uint8_t password[8] = {0}; // Переменная, которая будет хранить в себе пароль

void setup() {
  Инициализируйте 3 пина RGB_G, RGB_R, RGB_B на выход
  // put your setup code here, to run once:
```

```
        Serial.begin(115200); // Инициализация передачи данных в UART интерфейс со скоростью 115200 кбит\с

        Serial.println("Вы включили плату в первый раз. Введите пароль для доступа [8 цифр]:");

        while(!(Serial.available() > 7) ){}// Данный цикл позволит "ожидать" до тех пор пока в буфере не накопится 8 байт

        Serial.println("Ваш пароль:");

        /*Цикл который вычитывает 8 байт из буфера обмена*/
        for(Цикл от 0 до 8 (не включительно)){
          password[i] = Serial.read() - '0';
          Serial.print(password[i]);
        }
        Serial.read(); // Функция вызывается для считывания последнего символа \n - возврат каретки
        /*-----------------------------------------*/
        Serial.println(); // Так как предыдущая функция вывода не вставляет возврат коретки воспользуемся данной функцией
         }
```

Здесь используется функция Serial.available() – данная функция хранит в себе значения количества байт, которое хранится в буфере последовательного порта. Так как наш пароль состоит из 8 цифр, следовательно, нам необходимо получить значение равное 8 байтам.

Следовательно, сделаем цикл, который будет «бесконечно крутиться» до тех пор, пока функция не вернет значение больше 7, которое будет означать, что в буфере хранится значение, которое занимает 8 байт (8 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 8 байт, нам нужно повторить эту операцию 8 раз. Для этого воспользуемся циклом.

ВАЖНО ПОНИМАТЬ

Функция возвращает свои значения согласно ASCII таблице (рисунок 7.4):

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Рисунок 7.4 – ASCII таблица

То есть, если ввести 0, то функция Serial.read() вернет значение 30_{16} (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!

Именно эта конструкция и реализована в коде выше!

Также в коде мы выводим значение, которое поступило к нам в монитор порта. Для того, чтобы удостовериться в том, что то, что мы вели действительно верно запишется в память устройства.

Для того, чтобы отправить данные по последовательному UART интерфейсу нужно ввести данные в данном поле (рисунок 7.5) и нажать кнопку отправить (ENTER).

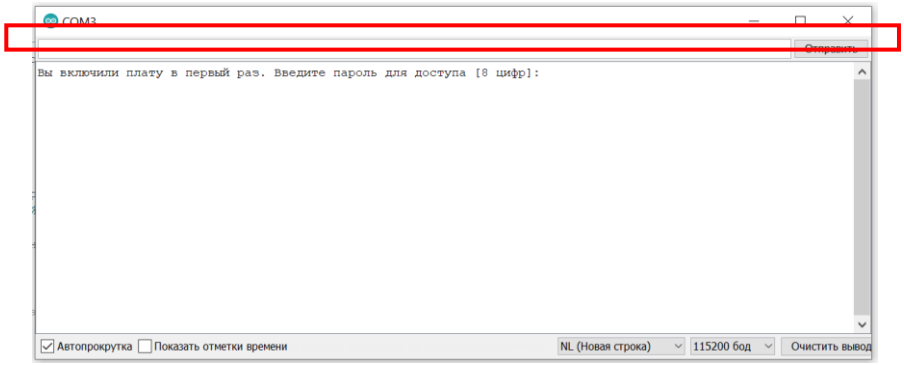


Рисунок 7.5 – Монитор порта

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.

Перед этим убедитесь, что в меню выбора периферии у вас включен RGB светодиод.

```
void loop() {
  // put your main code here, to run repeatedly:
  Serial.println("Для доступа в систему необходимо ввести пароль:");
   Включаем синий светодиод
  while(!(Serial.available() > 7) ){} // Данный цикл позволит "ожидать" до тех пор пока в буфере не накопится 8 байт
  /*Цикл который вычитывает 8 байт из буфера обмена*/
  for(int i = 0; i < 8; i++) {
    uint8_t current = Serial.read() - '0';
    if (password[i] != current){
      Serial.println("Пароль не верный. Доступ запрещен");
       Выключаем синий светодиод
       Включаем красный светодиод
      while(1) ; //Бесконечный цикл который остановит работу программы
    }
  }
   Выключаем синий светодиод
  Serial.println("Пароль верный! Доступ есть!");
  Включаем зеленый светодиод
  while(!(Serial.available() > 1) ){}
}
```

Скопируйте данный код и завершите его. Далее протестируйте его работу.

Дополнительные задания, которые необходимо выполнить, чтобы получить дополнительный балл:

1) Изменить длину пароля для ввода до 10 символов;

2) Объяснить для чего в коде присутствует данный фрагмент:

```
while(!(Serial.available() > 1) ){}
```

Домашнее задание

Найти все функции для работы с Serial в Arduino IDE и сформировать таблицу, где будет написано пояснение для использования каждой из них.

В дальнейшем здесь можно сделать систему управления, которая также будет считывать;

Данные с UART и выполнять некоторые действия (для конечного проекта);

Идея для конечного проекта. Сделать возможность сохранения пароля в памяти устройства. Которая не зависит от наличия энергии.

Лабораторная работа №8
Изучение интерфейсов. I2C

Цель работы: изучить способы передачи информации от контроллера контроллеру\компьютеру.

Задача:

1) Изучить принципы работы I2C интерфейса;

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

Теоретическая часть

I2C интерфейс

В прошлой работе мы изучили принцип работы последовательного интерфейса UART. Для реализации данного интерфейса понадобится всего лишь один провод для односторонней передачи. И два для двусторонней.

В случае интерфейса I2C для передачи информации между схемами по умолчанию требуются 2 провода. Для двух линий:

- Линия данных (SDA);
- Линия такта (SCL).

Устройство, которое является инициатором соединения называют Master (Ведущий). В случае платы TUSUR_IOT_BOARD Master'ом будет выступать контроллер.

Устройство, которое подключается к линии, управляемой Ведущим, называется Slave (Ведомым). Чаще всего в роли ведомого выступают датчики или другие контроллеры.

На плате TUSUR_IOT_BOARD на шине I2C расположенные 3 устройства. 2 микроконтроллера: ESP и STM. А также датчик температуры и давления BMP280.

Обмен между двумя ведомыми невозможен. Так как некому будет задавать синхросигналы.

Все на одной двухпроводной линии может быть до 127 устройств.

Процесс подключения устройств к шине I2C выглядит следующим образом (рисунок 8.1):

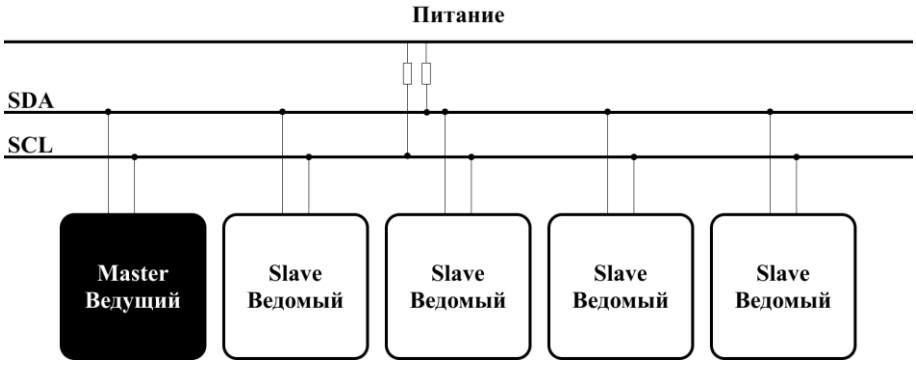


Рисунок 8.1 – Подключение устройств по шине I2C

Каждая из шин подтянута к логической единице (к питанию). Слово подтянута можно интерпретировать как подключена. То есть это означает, что при включении системы, когда шина не будет активна на ней будет присутствовать логическая единица.

Важно понимать, что каждое устройство подключается к шине I2C со своим собственным адресом. Для передачи адреса используется 7 бит.

$2^7 - 1 = 127$ возможных адресов, следовательно, 127 возможных устройств.

И тут можно сразу же увидеть преимущество шины I2C по сравнению с UART. Оно заключается в том, что для подключения большого количества устройств по UART для каждого устройства необходимо будет выделить один провод для передачи в одну сторону и

два для передачи в две стороны. В то время как для I2C будет достаточно 2 провода для подключения 127 устройств.

Процесс передачи информации по шине I2C

Сигнал, который передается по линии I2C выглядит следующим образом (рисунок 8.2):

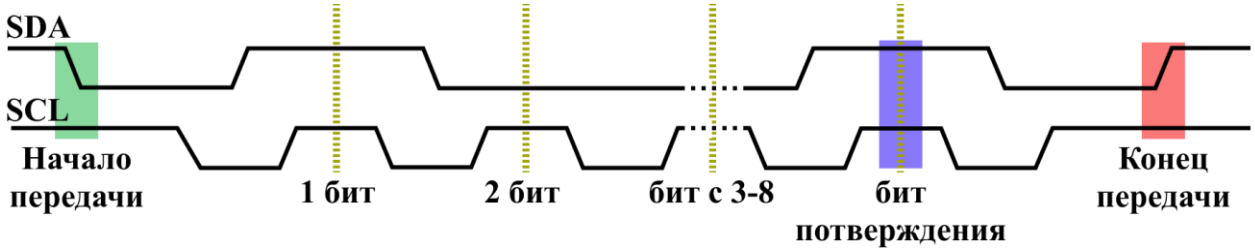


Рисунок 8.2 – Пример сигнала при передаче информации по шине I2C

Процедура обмена начинается с того, что ведущий формирует состояние старт: при высоком уровне на линии SCL он генерирует переход сигнала линии SDA из высокого состояния в низкое, этот переход воспринимается всеми устройствами, подключенными к шине, как признак начала процедуры обмена. Генерация синхросигнала — это всегда обязанность ведущего; каждый ведущий генерирует свой собственный сигнал синхронизации при пересылке данных по шине.

При передаче посылок по шине I2C каждый ведущий генерирует свой синхросигнал на линии SCL. После формирования состояния старт ведущий опускает состояние линии SCL в низкое состояние и выставляет на линию SDA старший бит первого байта сообщения. Данные действительны и должны оставаться стабильными только во время высокого состояния синхроимпульса. Для подтверждения приёма байта от ведущего-передатчика ведомым-приёмником в спецификации протокола обмена по шине I2C вводится специальный бит подтверждения, выставляемый на шину SDA после приёма 8 бит данных.

Процедура обмена завершается тем, что ведущий формирует состояние стоп — переход состояния линии SDA из низкого состояния в высокое при высоком состоянии линии SCL. Состояния старт и стоп всегда вырабатываются ведущим. Считается, что шина занята после фиксации состояния старт. Шина считается освободившейся через некоторое время после фиксации состояния стоп.

Практическая часть

Изучения работы I2C интерфейса

Как и в прошлой работе для изучения I2C интерфейса нам понадобится логический анализатор. Который можно найти в меню платы. В меню логического анализатора необходимо выбрать «I2C».

Далее загрузите следующий скетч на микроконтроллер:

```
#include <Wire_slave.h> // Библеотека для реализации передачи данных по I2C как slave

void setup()
{
  Wire.begin(8);                // Подключение к шине I2C с адресом 0x08
  Wire.onRequest(requestEvent); // Передача названия функции, которая будет срабатывать при поступлении запроса от мастера
}
uint8_t i = 0; // Переменная которая будет хранить значения для передачи по I2C
```

```
void loop()
{
  i++; // Увеличение переменной
  delay(3000); // Ожидание в течении 3 сек
}
void requestEvent() //Функция которая будет срабатывать при поступлении запроса от Master'a
{
  Wire.write(i); // Отправка значения i по I2C
}
```

В данном скетче по шине I2C каждый три секунды передается значение, которое увеличивается от 0 до 255. Проследите за изменением сигнала.

Далее в функции Wire.write(); замените переменную i на своё значение. Убедитесь в том, что на логическом анализаторе ваше значение отображается верно. Докажите это преподавателю!

Разработка системы контроля температурного режима

В ходе данной работы реализуем систему контроля температурного режима. Для реализации данной системы нам понадобятся следующие элементы:

1) RGB светодиод;
2) Вентилятор;
3) Подсветка вентилятора;
4) Датчик температуры;
5) Переменный резистор.

Данная система должна выполнять следующие действия:

1) Иметь возможность установить желаемую температуру;
2) Получать данные о текущей температуре с датчика;
3) На основании сравнения желаемой и текущей температурой с датчика. Включать\выключать «отопление\охлаждение».

Для того, чтобы ими воспользоваться – объявим пины к которым они подключены:

```
#include <Adafruit_BMP280.h>// Библиотека, которая хранит в себе функции для взаимодействия с датчиком BMP280.
Adafruit_BMP280 bmp280; // Создание объекта bmp280 – температурный датчик
#define fan PA1
#define led_fan PA2
#define PR PB0
#define RGB_G PB1
#define RGB_R PB9
#define RGB_B PB8
void setup() {
  Serial.begin(115200);
  /*Инициализация необходимых пинов*/
  pinMode(fan, OUTPUT);
  pinMode(led_fan, OUTPUT);
  pinMode(PR, INPUT);
  pinMode(RGB_R, OUTPUT);
  pinMode(RGB_G, OUTPUT);
  pinMode(RGB_B, OUTPUT);
  /*-------------------------------------*/
```

```
  while (!bmp280.begin(0x76)) { // Цикл для проверки подключения температурного датчика по I2C
// Где 0x76 – адрес температурного датчика
            Serial.println(F("Could not find a valid BMP280 sensor, check wiring!"));
            delay(2000);
          }
        }
```

Здесь мы подключаем датчик по шине I2C. И здесь вы не сможете увидеть явное объявление пинов I2C (SDA, SCL), так как к микроконтроллеру STM датчик BMP280 подключен, используя аппаратные пины I2C. Это означает, что изначально при проектировании микроконтроллера разработчиками изначально были выбраны пины для I2C, которые программа использует по умолчанию, когда объявляется устройство, использующее данную шину.

Далее напишем программный код, который позволяет получить значение с датчика, а также задать желаемую температуру, с помощью переменного резистора.

```
        void loop() {
          float temp = bmp280.readTemperature(); // Записываем значение с датчика температуры в переменную temp
          float our_temp = analogRead(PR)*30.0/4096.0+10.0; // С помощью преобразований значения полученного с ПР устанавливаем желаемую температуру.
          float difference = разница температур; // Разницу между желаемой и истинной температурой записываем в переменную difference
          Serial.println();
          Serial.print(temp);
          Serial.print("-");
          Serial.print(our_temp);
          Serial.print("=");
          Serial.print(разница температур);
          Serial.println();
          delay(3000);
        }
```

Вместо поля «разница температур» запишите разницу переменных, которые хранят в себе значения с датчика и значение, которое вы задаете с помощью ПР.

Далее ОБЯЗАТЕЛЬНО

Убедитесь в том, что в меню периферии включены:

- RGB светодиод;
- Вентилятор;
- Переменный резистор.

Запустите данный программный код и проверьте, что значения приходят верные.

Значения с переменного резистора должны быть в диапазоне от 10 до 40.

Значения с датчика температуры должны быть сопоставимы с температурой в комнате.

А теперь ответь на вопросы:

При какой разнице нужно включить охлаждение?

При какой разнице нужно включить отопление?

Соответственно полученной разнице наша программа должна реагировать определенным образом.

Когда требуется включить отопление мы запускаем вентилятор с включенным встроенным в вентилятор светодиодом. А также включаем красный светодиод яркость

которого пропорциональна разности температур. Чем больше разница, тем ярче горит красный светодиод. Также скорость включения вентилятора также должны зависеть от разницы между температурами.

Для сохранения пропорциональности примем разницу температур в 10 градусов максимальной. Следовательно, когда разница будет достигать значение в 10 градусов все наши системы должны работать с максимальной интенсивностью.

Для реализации данной задачи, воспользуемся следующим программным кодом:

```
 void loop() {
  float temp = bmp280.readTemperature(); // Записываем значение с датчика температуры в переменную temp
  float our_temp = analogRead(PR)*30.0/4096.0+10.0; // С помощью преобразований значения полученного с ПР устанавливаем желаемую температуру.
  float difference = разница температур; // Разницу между желаемой и истинной температурой записываем в переменную difference

  if (difference < -1){ // Если разница меньше нуля - то нужно нагреть
   digitalWrite(led_fan, HIGH); //Включаем индикатор отопления
   if (abs(difference) >= 10) difference = 10; // Если разница достигает больше 10 оставляем значение 10 (для пропорции)
   analogWrite(fan, 255*abs(difference)/10); // Крутим вентилятор с силой равной разнице между темп-ми
   digitalWrite(RGB_G, LOW); // Тушим зеленый светодиод
   analogWrite(RGB_B, 0); // Тушим синий светодиод
   analogWrite(RGB_R, 255*abs(difference)/10); // Светим красным с силой равной разнице между темп-ми
  }
  else if (difference > 1){ // Если разница больше нуля - то нужно остудить
   Выключаем индикатор отопления
   if (difference >= 10) difference = 10;  // Если разница достигает больше 10 оставляем значение 10 (для пропорции)
   analogWrite(fan, 255*abs(difference)/10); // Крутим вентилятор с силой равной разнице между темп-ми

   Тушим зеленый светодиод
   Тушим красный светодиод
   Светим синим светодиодом с силой равной разнице между темп-ми
  }
  else { // Если разница небольшая, то просто ожидаем изменение температуры
   Выключаем индикатор отопления

   Тушим синий светодиод
   Тушим красный светодиод
   Зажигаем зеленный светодиод
   Останавливаем вентилятор
  }
 }
```

Заполните поля, выделенные жёлтым.

После этого загрузите скетч в память устройства. Если все работает верно, можете взять дополнительное задание у преподавателя!

Лабораторная работа №9
Изучение интерфейсов. SPI

Цель работы: изучить способы передачи информации от контроллера контроллеру\компьютеру.

Задача:

1) Изучить принципы работы SPI интерфейса;

2) Получить навыки по работе с SPI интерфейсом в Arduino IDE;

Теоретическая часть

SPI интерфейс

В прошлых двух работах мы уже изучили 2 интерфейса для передачи данных между несколькими устройствами I2C и UART. Но существует также еще один популярный интерфейс SPI.

Для передачи по SPI интерфейсу нам понадобится 4 линии:

- SS выбор схемы;
- MISO (MasterInputSlaveOutput) вход ведущего выход ведомого;
- MOSI (MasterOutputSlaveInput) выход ведущего вход ведомого;
- CLK сигнал синхронизации.

Подключение устройств по SPI интерфейсу выглядит следующим образом (рисунок 9.1):

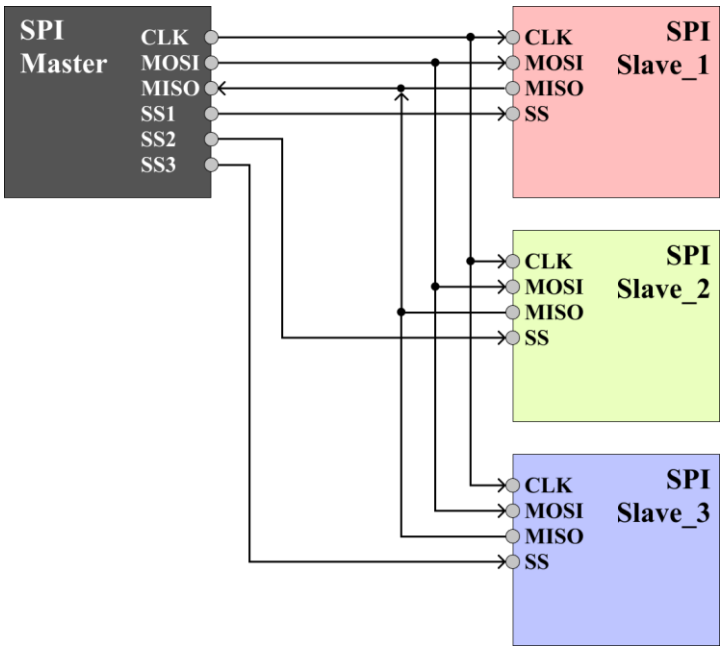


Рисунок 9.1 – Подключение устройств по SPI интерфейсу

Как мы можем видеть из рисунка для подключения нескольких устройств к одному мастеру понадобится одна шина для выбора схемы. Следовательно, для подключения, к примеру, 5-ти устройств понадобится 5 дополнительных пинов для SS. Это является небольшим недостатком использования интерфейса SPI.

К большому преимуществу следует отнести тот факт, что в SPI интерфейсе. Одновременно можно осуществить прием и передачу данных, в отличие от интерфейса I2C, где в один момент времени можно либо передавать данные для Slave'а либо же принимать от него данные.

Сигнал, передаваемый по интерфейсу SPI, выглядит следующим образом (рисунок 9.2):

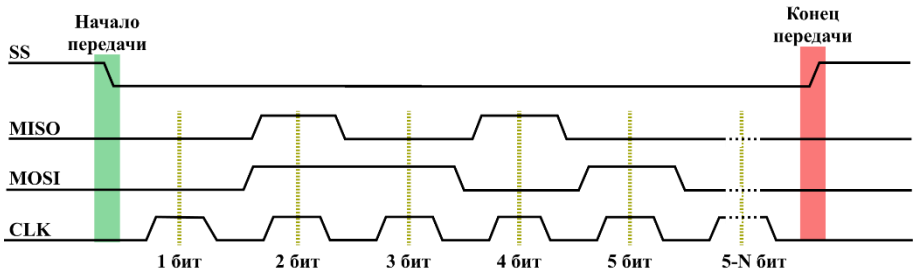


Рисунок 9.2 – Сигнал, передаваемый по интерфейсу SPI

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).

СТАРТ передачи легко определить по фронту линии SS. Если линия SS переходит из 1 в 0. Значит начался старт передачи данных. Если же линия переходит из 0 состояния в 1, то это означает КОНЕЦ передачи данных.

Важно знать, что SPI интерфейс встречается очень часто для подключения разного рода модулей и датчиков к плате. И при проектировании системы нужно всегда помнить, что SPI интерфейс занимает как минимум 4 пина для передачи данных и по 1 пину для каждого последующего подключенного устройства. Но при этом обеспечивает наилучшую пропускную способность среди предшествующих 2 интерфейсов.

Практическая часть

Изучение работы SPI интерфейса

Для изучения работы SPI интерфейса нам понадобится логический анализатор, который можно найти в меню платы.

Запустив логический анализатор, загрузите следующий код в микроконтроллер:

```
#include <SPI.h> // Библеотека для работы SPI

uint8_t i = 0; // Переменная которая хранит значение для передачи по SPI
void setup() {
  Serial.begin(115200);
  pinMode(PA4, OUTPUT);
  SPI.begin(); //Инициализация 1 SPI порта.
  SPI.setBitOrder(MSBFIRST); // Установка при котором младгий бит идет первым
  SPI.setDataMode(SPI_MODE0); //Установка SPI в 0 режиме
  SPI.setClockDivider(SPI_CLOCK_DIV64); // Режим низкой скорости (72 / 64 = 1,125
MHz SPI_1 speed)
}
void loop() {
  i++;
  delay(3000);
  sendSPI(i);
}
void sendSPI(uint8_t i) // Функция для отправки данных
{
 digitalWrite(PA4, LOW); // Отпускаем ножку SS в 0
 SPI.transfer(i); //Отправляем данные
 digitalWrite(PA4, HIGH); // Отпускаем ножку SS в 1
}
```

В данном скетче по шине SPI каждый три секунды передается значение, которое увеличивается от 0 до 255. Проследите за изменением сигнала.

Далее в функции SPI.transfer(i); замените переменную i на своё значение. Убедитесь в том, что на логическом анализаторе ваше значение отображается верно. Докажите это преподавателю!

Создание системы КПП с использованием RFID модуля

КПП (контрольно-пропускной пункт). Смысл данной системы заключается в том, чтобы осуществлять контроль при пропуске лиц в некоторое частное место. (К примеру, пропуск в общежитие). В качестве пропуска может выступать как документ, так и некоторые ключи.

В нашем случае будут использоваться RFID метки.

Данная система должна обладать следующими параметрами:

1) Уметь записывать данные на метку и считывать их;
2) Сравнять считанное значение;
3) В случае успешного прохождения теста – отдавать сигнала для пропуска.

Для создания данной системы нам понадобится:

1) RGB светодиоды и зуммер для оповещения;
2) Свитч для переключения записи\чтения данных с метки;
3) RFID метка;
4) RFID модуль;

ВАЖНО!

Для того, чтобы взаимодействовать с RFID модулем необходимо в папку с проектом добавить файл «TIB_RFID.h» в данном файле описаны функции и все необходимы параметры для того, чтобы взаимодействовать с RFID модулем.

Данный файл вы можете получить у преподавателя или на портале SDO.

Для того, чтобы добавить файл в своей проект нужно:

1) Создать проект и сохранить его в определенной папке (рисунок 9.3):

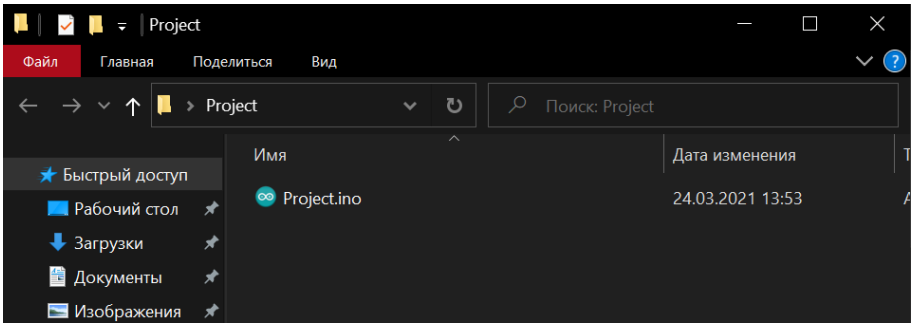


Рисунок 9.3 – Сохранение проекта

2) Далее в папку с проектом необходимо добавить полученный файл (рисунок 9.4):

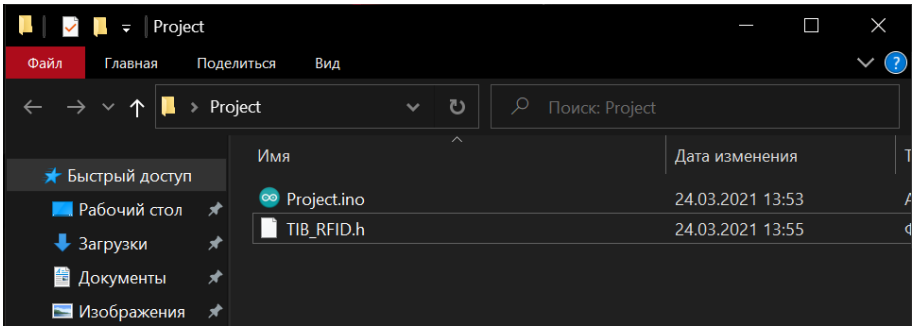


Рисунок 9.4 – Добавление файла в проект

3) Перезапустите проект. (Закрыв и открыв его заново);

4) Если все прошло успешно, данный файл должен появится у вас во вкладках программы (рисунок 9.5):

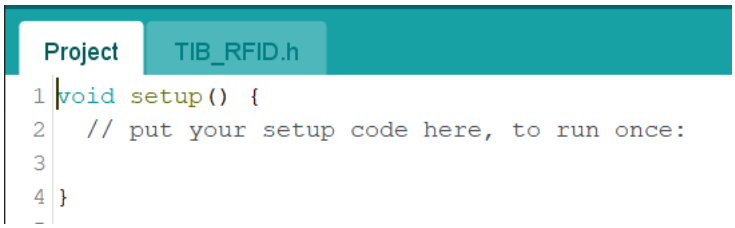


Рисунок 9.5 – Наличие файла в проекте

5) Чтобы воспользоваться кодом из данного файла необходимо вставить скрипт #include “TIB_RFID.h” (рисунок 9.6):

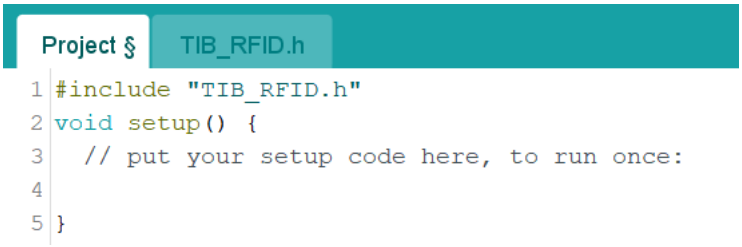


Рисунок 9.6 – Подключение файла

Далее приступим к написанию кода.

В начале сделаем необходимые настройки и объявим необходимые переменные:

```
#include "TIB_RFID.h"
String password = "TeST"; // 16 символов МАКС
void setup() {
  pinMode(RGB_R, OUTPUT); // Красный RGB
  pinMode(RGB_G, OUTPUT); //Зеленый RGB
  pinMode(RGB_B, OUTPUT); //Синий RGB
  pinMode(zoom, OUTPUT); //Зуммер
  pinMode(LED, OUTPUT); //Зеленый светодиод
  pinMode(switch1, INPUT); //Свитч1
  Serial.begin(9600); // Инициализация последовательного интерфейса
  SPI.begin();      // Инициализация SPI шины
  mfrc522.PCD_Init(); // Инициализация RFID модуля
  //Подготовка ключа для записи и чтения с RFID метки по умолчанию FFFFFFFFFFFFh
  for (byte i = 0; i < 6; i++) key.keyByte[i] = 0xFF;
 }
```

Далее давайте изучим работу функций из файла TIB_RFID.h. В данном файле присутствуют 2 функции:

- Одна создана для записи данных на метку: TIB_write_rfid();
- Другая для чтения данных с метки: TIB_read_rfid();

Функция для записи данных на метку принимает два аргумента: строку (класс String), а также длину данной строки (тип uint8_t).

Давайте попробуем записать свои первые данные на метку. Для это в переменную «`password`», запишите свое сообщение. НЕ БОЛЬШЕ 16 СИМВОЛОВ.

В цикле воспользуйтесь функцией TIB_write_rfid(password, password.length());
Данная функция отправит на запись ваше сообщение.
password.length() – это метод, который позволяет получить длину строки.

```
void loop() {
  TIB_write_rfid(password, password.length() + 1);
}
```

Запустите скетч, НО перед загрузкой убедитесь, что в меню периферии включени RGB светодиод.

При записи светодиод будет мигать поочередно то синий, то зеленый. При успешной записи вы услышите короткий звук, если запись прошла не успешна, то более длинный звук, а также RGB светодиод будет гореть красным цветом!

Далее давайте проверим сообщение записанное на метку, для этого воспользуемся функцией TIB_read_rfid();

Данная функция не принимает аргументов. Она возвращает строку считанную с RFID метки.

Для того, чтобы ей воспользоваться создайте переменную String answer;

И присвойте ей значение, возвращаемое функцией.

Самостоятельно напишите код, который выведет в последовательный порт значение, считанное с метки. Покажите преподавателю, что на метке находится именно то значение, которое было записано вами.

Далее реализуем свою систему КПП, для этого, воспользуемся следующим кодом:

```
void loop() {
  String answer;
  if (Свой код){ // Если свитч отключен начинаем запись на метку
    TIB_write_rfid(password, password.length() + 1);
    return;
  }

  if (Свой код){ // Если свитч включен начинаем считывать значение с метки
    answer = TIB_read_rfid();
    if (answer.length() != password.length()){ // Если размеры ключа и полученного значения не совпадают выводим сообщение об ошибке
      password_res(false);
      return;
    }
    for(int i = 0; i < answer.length(); i++){ // Если размеры равны, то сравниваем каждые символы
      if (answer[i] != password[i]){ // В случае если хоть один символ не совпал выводим сообщение об ошибке.
        password_res(false);
        return;
      }
    }
    password_res(true); // Если все тесты пройдены значит метки совпадают
  }
}
```

Здесь мы можем увидеть функцию с названием password_res();

Данная функция не объявлена в коде и вам предстоит ее написать, для этого вам нужно ее объявить. Данная функция принимает 1 аргумента типа bool. И выполняет следующие действия:

1) Если пришло false (метка не совпадает), то включается красный диод, горит на протяжение 2 секунд и потом выключается;

2) Если пришло true (метка совпала), то включается зеленый светодиод, вентилятор. Потом через 1 секунду выключается вентилятор и еще через 1 секунду зеленый светодиод.

После того, как весь код будет дописан. Запустите скетч и проверьте его работу.

Запишите новый код на метку и проверьте его.

Лабораторная работа №10
Обработка данных с кнопок и энкодеров

Цель работы**:** получение навыком по обработке данных кнопок и энкодеров с избеганием влияния «дребезга»

Задачи:

1) Изучить схемы подключения кнопок и энкодеров;

2) Изучить понятие «дребезг» и способы борьбы с ним;

3) Написать программу, которая позволяет программно предотвратить влияние дребезга на кнопку.

Теоретическая часть

Обработка данных с кнопок

Мы уже рассматривали схемы подключения кнопки (рисунок 10.1):

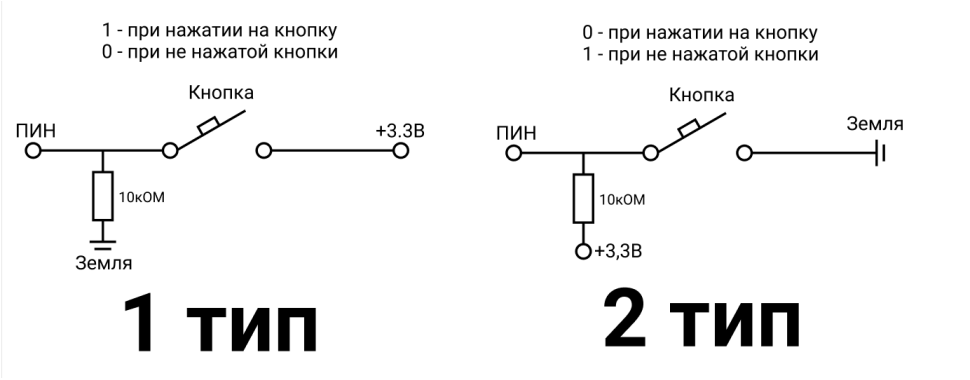


Рисунок 10.1 – Схемы подключения кнопки

Исходя из этих схем, можно легко предположить, что кнопка по своей сути является разводным мостом, который позволяет определенному току пройти\уйти до\от пина, где мы будем считывать значение напряжения.

Нажимая на любую кнопку, мы формируем определенное давление на внутренние механизмы, в результате чего происходит сближение или расхождение металлических пластин (проводов).

В идеальном мире при нажатии на кнопку сигнал сразу же достигает своего назначения.

В нашем неидеальном мире в момент нажатия на кнопку в месте соединения контакты не соприкасаются мгновенно, микронеровности на поверхности не позволяют пластинам мгновенно соединиться из-за чего мы можем наблюдать сигнал следующей формы (рисунок 10.2):

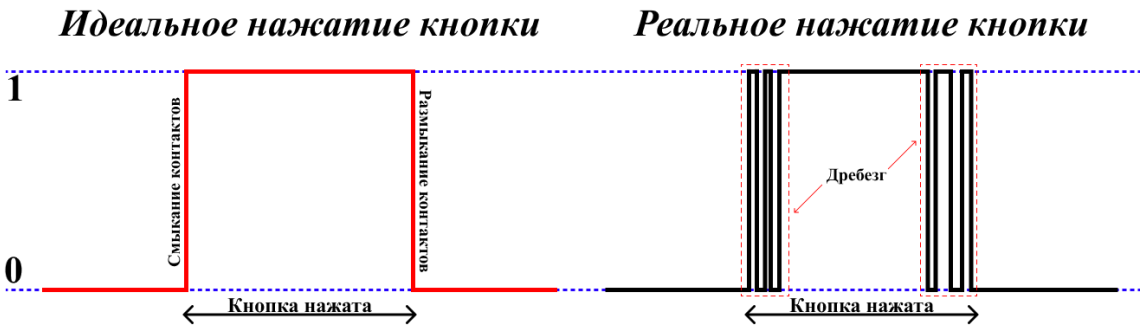


Рисунок 10.2 – Форма сигнала без дребезга и с дребезгом

Мы можем заметить, что за счёт неровностей, при нажатии кнопки возникает момент, когда она не сразу устанавливает плотный контакт, а как бы немного вибрирует и за счет этого мы можем видеть несколько, чаще всего, случайных попыток установки контакта. И в этот момент для контроллера это будет означать, что вы быстро нажали кнопку несколько раз подряд.

На первый взгляд может показать, что ничего страшного. Но проблема заключается в том, что мы можем ожидать нажатие кнопки в тот момент, когда там происходит дребезг, но контроллер может считать сигнал именно в тот момент, когда контакт еще не установился (рисунок 10.3):

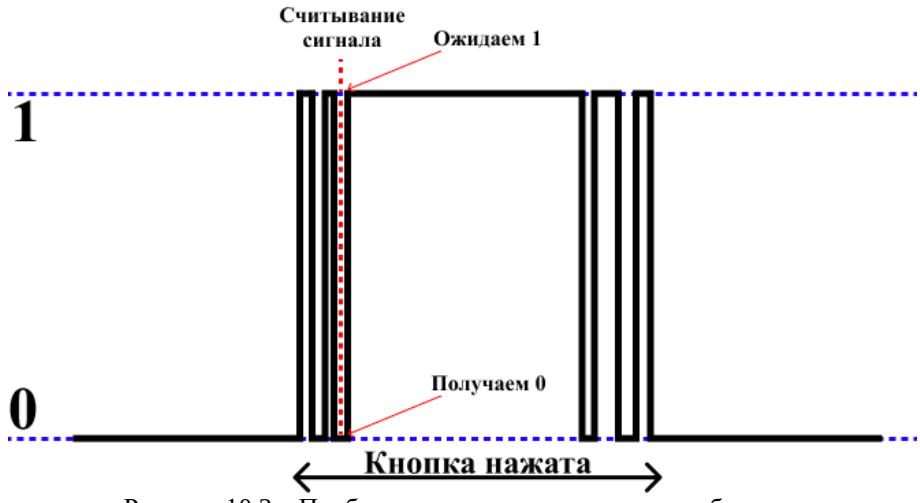


Рисунок 10.3 – Проблема считывания данных с дребезгом

В случае с диодом, который включается по кнопке, это не так критично, так как при неправильном считывании он просто не включится\выключится. Но, что, если придётся использовать кнопку для остановки какого-либо опасного процесса, где счёт времени идет на секунды и не будет времени, чтобы в случае неудачи нажать кнопку еще раз.

Когда в системе важна надежность используют алгоритмы и способы для минимизации дребезга.

Данную проблему можно решить двумя способами: программным и аппаратным.

Программный способ защиты от дребезга подразумевает использования некоторого алгоритма, который позволит обойти аппаратные ограничения подключенной к плате кнопки.

Самый простой способ реализации защиты от дребезга программным способом выглядит следующим образом:

```
int currentValue, prevValue;
void loop() {
  currentValue = digitalRead(BUTTON);
  if (currentValue != prevValue) {
    // Произошло изменение состояния
    // Далее может начаться эффект дребезга
    // Поэтому делаем задержку
    delay(10); // Можно изменять задержку для достижения более благоприятного результата [5-50 мс]
    // После задержки считываем значение, считая что эффект закончился
    currentValue = digitalRead(BUTTON);
  }
  prevValue = currentValue;
}
```

Данный алгоритм можно представить в виде рисунка 10.4:

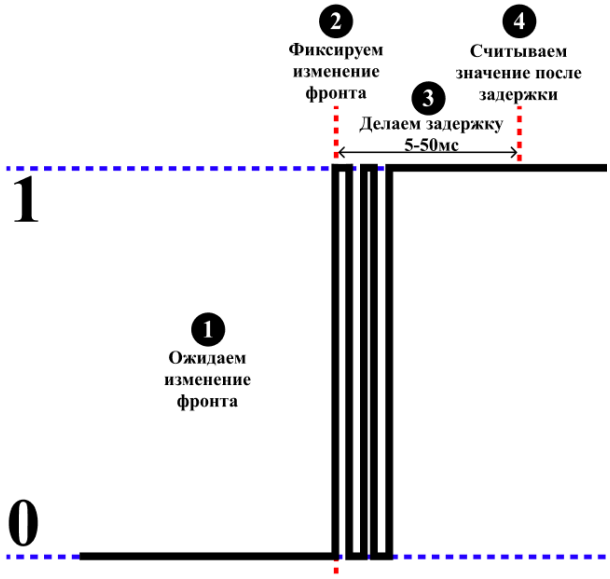


Рисунок 10.4 – Программное решение для устранения «дребезга»

Аппаратная защита от дребезга подразумевает собой использования улучшенной схемы подключения кнопки к пину за счёт использования фильтра. В качестве фильтра чаще всего используется конденсатор номиналом 100 нФ. Схема для подключения кнопки с фильтром будет выглядеть следующим образом (рисунок 10.5):

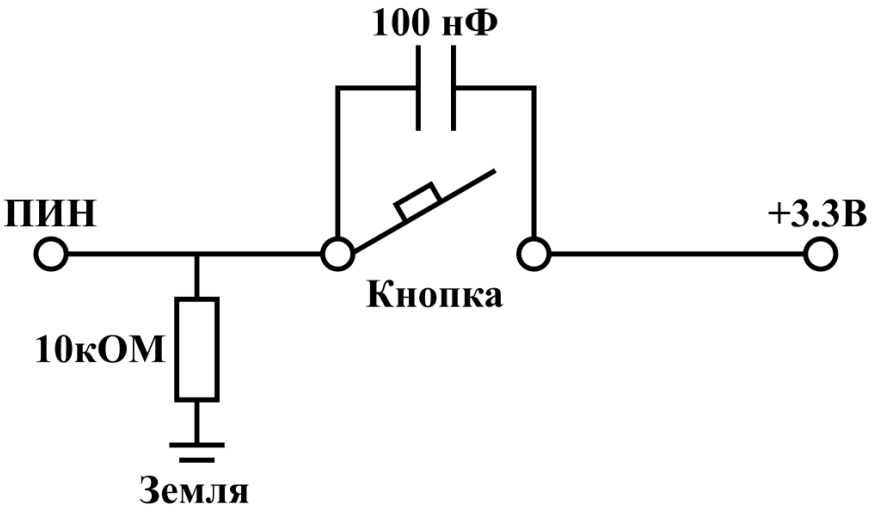


Рисунок 10.5 – Схема подключения кнопки с конденсаторным фильтром

Данная схема подключения позволяет воспользоваться переходными свойствами конденсатора. Он может “поглощать” все резкие пики, медленно накапливая и отдавая энергию, точно так же, как это делает пружина в амортизаторах. Если использовать данную схему подключения, то сигнал на входе будет выглядеть следующим образом (рисунок 10.6):

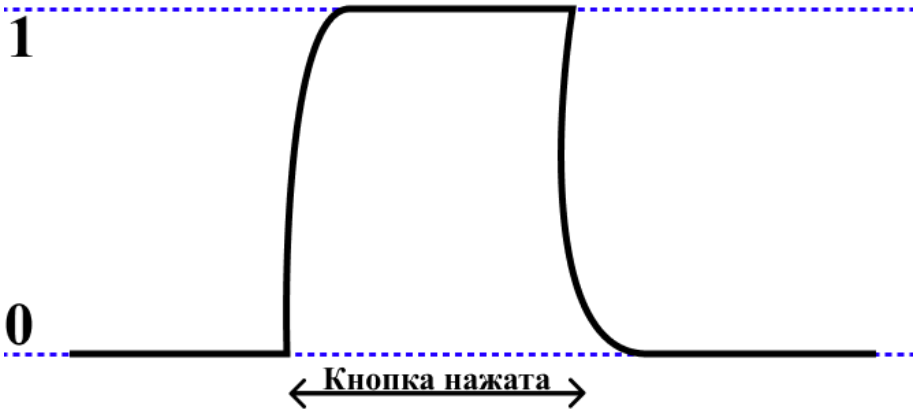


Рисунок 10.6 – Сигнал, принятый с кнопки при использовании фильтра

Как мы видим конденсатор позволяет сгладить фронты, что в свою очередь позволяет проще определить фронт сигнала и допустить меньше ошибок.

В качестве защиты от дребезга на плате TUSUR_IOT_BOARD используется аппаратное решение проблемы. Именно поэтому, в ходе работы с платой не возникает проблем при использовании кнопок.

Обработка данных с Энкодера

Энкодер (другими словами, ДУП – датчик угла поворота) представляет собой устройство, которое позволяет преобразовать угловое положение механизма в цифровой сигнал. Чаще всего энкодер использует как элемент управления интерфейсом в некоторых устройствах. Вы уже встречались с энкодером на плате TUSUR_IOT_BOARD. На данной плате **энкодер** используется для взаимодействия с интерфейсом меню платы.

ДУП выглядит следующим образом (рисунок 10.7):

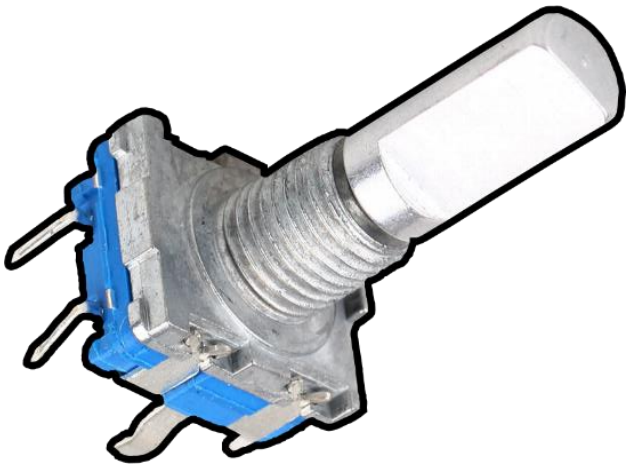


Рисунок 10.7 – Датчик угла поворота

По своей структуре датчик угла поворота, чаще всего, представляет из себя подключение 2 ключей и одной кнопки (рисунок 10.8):

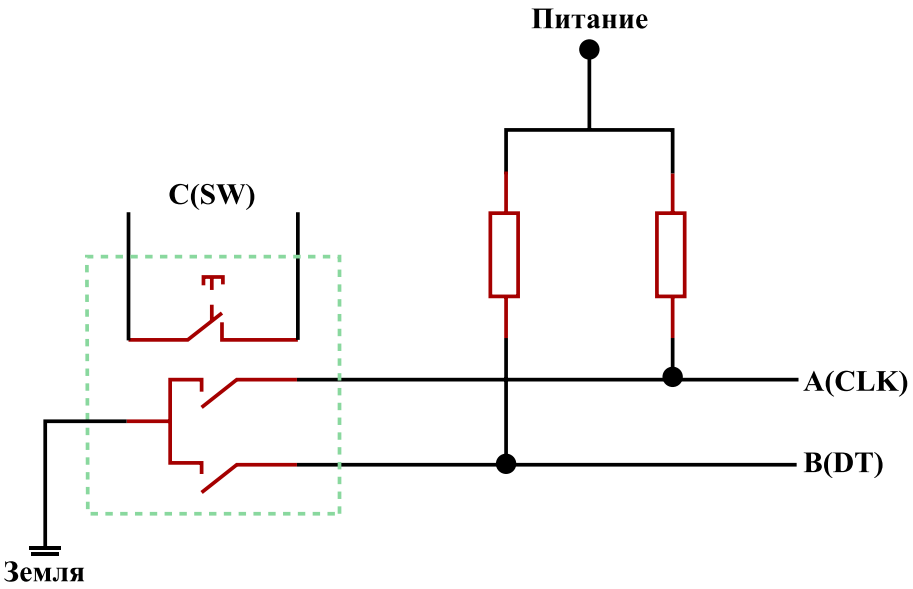
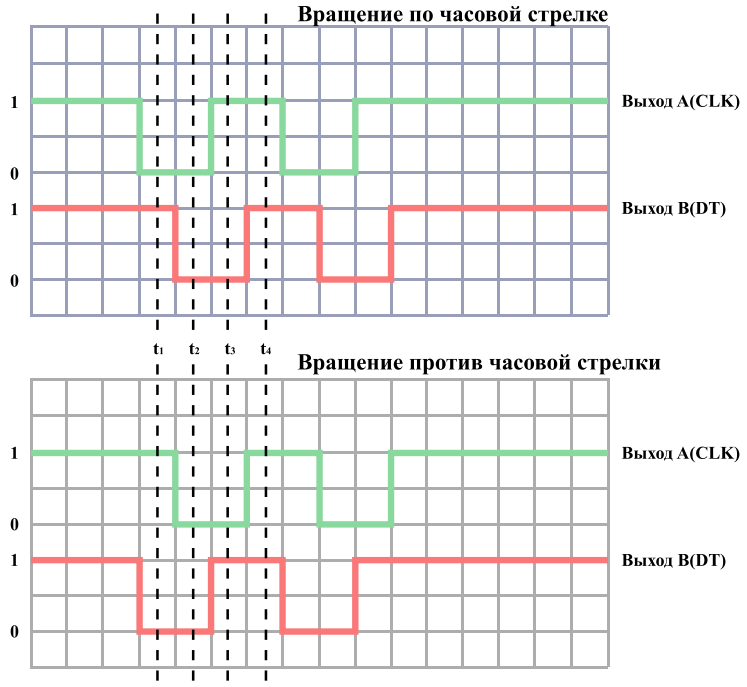


Рисунок 10.8 – Схема датчика угла поворота

Детектирование угла поворота осуществляется следующим образом. При повороте ручки замыкается сначала один ключ (A), а следом, через определенные промежуток времени, ключ (B), следовательно, при повороте ручки в другую сторону в обратном порядке. Данная закономерность позволяет определить в какую именно сторону была повернута ручка и использовать данное знание для взаимодействия с системами. Сигнал, которой формируется при повороте ручки, представлен на рисунке 10.9:



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 10.9 – Вид сигнала при повороте ручки

Зная последовательность, которая поступает на вход контроллера при повороте ручки, можно её декодировать для распознавания сигнала.

Практическая часть

Написание программы для обработки данных с энкодера.

В данной работе мы с вами попробуем реализовать функционал одного из примера, который находится в меню платы. А именно проигрыш звука, с помощью зуммера под управлением энкодера.

Для данной практики нам понадобятся следующая периферия:

1) Энкодер (SW, CLK, DT);
2) Зуммер (zoom);
3) Кнопка (but);

Для удобства можно воспользоваться скриптом #define:

```
/*Объявление скриптов для пинов*/
#define CLK PB13
#define DT PB12
#define SW PA15
#define zoom PA8
#define but PB4
/*----------------------------*/
```

В данный работе уже используется функция, которая позволяет быстро обработать данные энкодера с помощью таблицы (рисунок 10.9), данная функция выглядит следующим образом:

```
/*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;
}
```

Вставьте часть кода с #define и функцию в ваш проект.
Далее во фрагменте setup() выполните необходимые настройки.

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

Вместо «???» вставьте верное значение.
Далее во фрагменте loop() вам необходимо реализовать 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);
    ?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 в выделенных местах. Загрузите скетч на плату и проверьте работоспособность.

Лабораторная работа №11
Работа с экраном

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 геометрические фигуры на дисплей платы TUSUR_IOT_BOARD.

Теоретическая часть

Основные принципы построения изображений на электронных дисплеях

В течении 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: температура, время, количество 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 семисегментных индикаторов (рисунок 11.1):

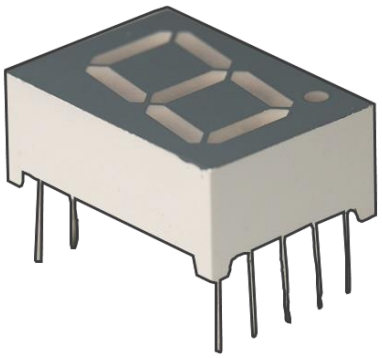


Рисунок 11.1 – Семисегментный индикатор

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 11.2):

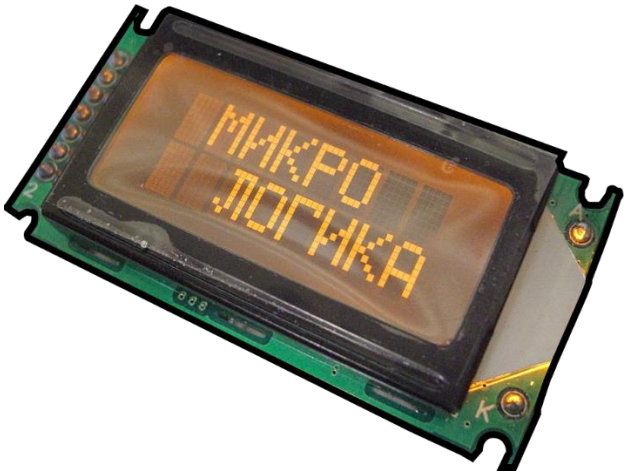


Рисунок 11.2 – Алфавитно-цифровой дисплей

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

К графическим дисплеям можно отнести даже монитор или экран смартфона. Их основное отличие заключается в том, что они представляют собой поверхность на котором 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 11.3):

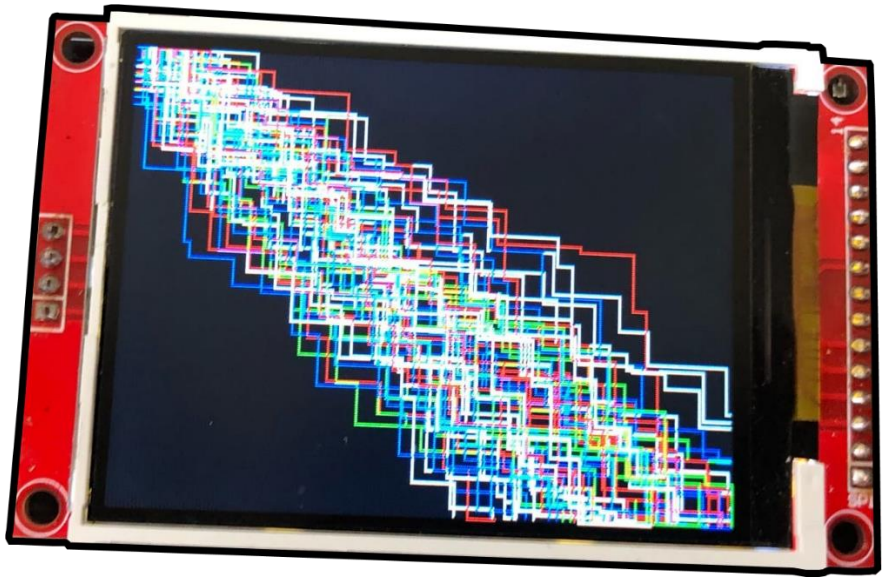


Рисунок 11.3 – Дисплей на плате TUSUR_IOT_BOARD

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.

Давай разберем процесс построения геометрических фигур на матрице размером 32x24 (рисунок 11.4):

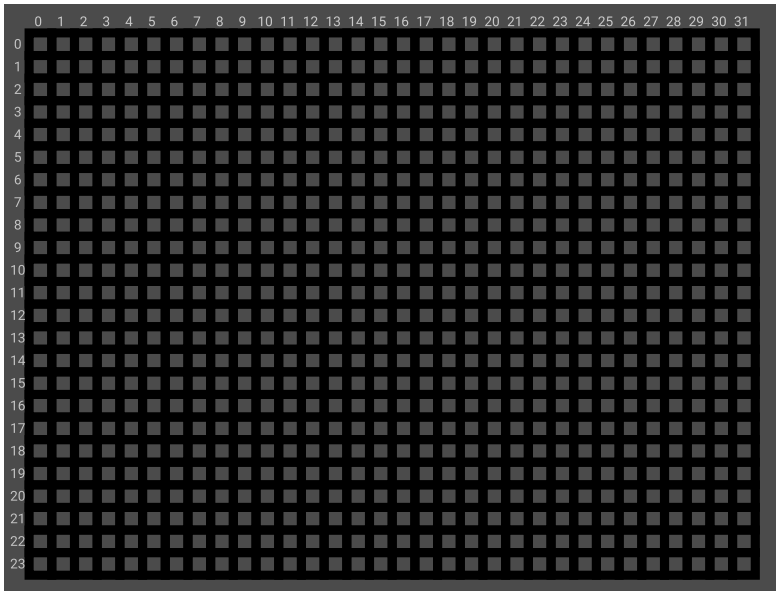


Рисунок 11.4 – Матрица для построения изображения

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 11.5):

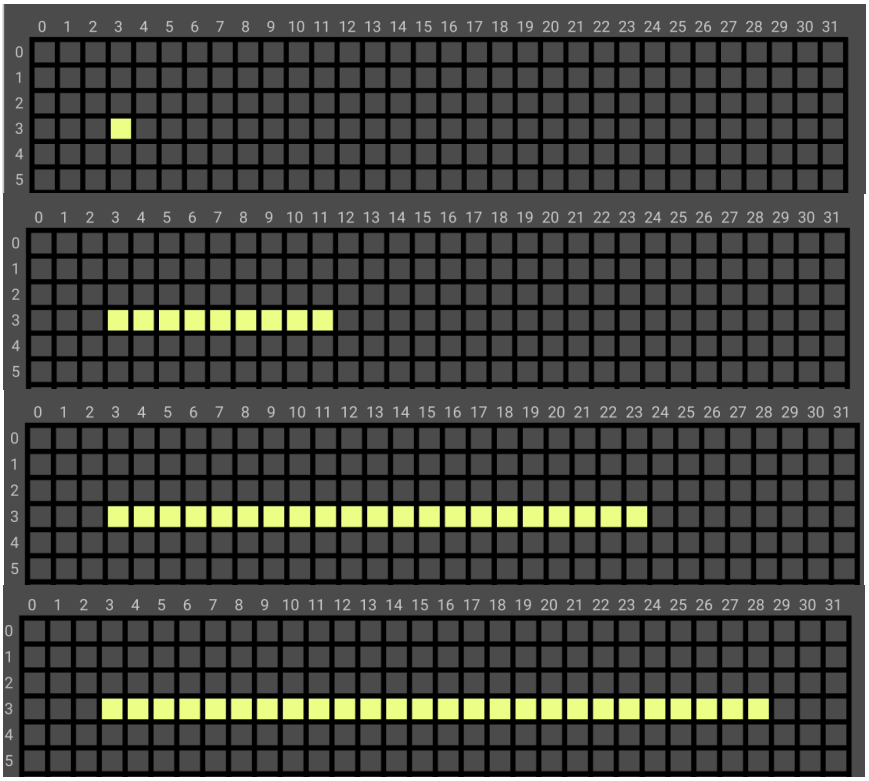


Рисунок 11.5 – Рисовании линии

Здесь мы можем увидеть, что каждый закрашиваемый пиксель принимал следующие значения:

[3, 3] [4, 3] [5, 3] [6, 3] . . . . [28, 3] → [3-28, 3]

Если бы нарисовать линию нужно было бы, используя программу, то программный код в Arduino IDE выглядел бы следующим образом:

```
uint16_t x = 3;
uint16_t y = 3;
uint16_t c = 2;
for(; x < 29; x++){
        disp_put_pixel(x, y, c);
        delay(2);
}
```

В данном примере используется функции disp_put_pixel(x, y, c); , которая находится в библиотеке TIB_display.h, которая в свою очередь специально разработана для платы TUSUR_IOT_BOARD. Она позволяет передавать команды по UART на контроллер ESP, так как дисплей подключен к нему, который в свою очередь формирует изображения используя функции другой библиотеки внутри своего программного кода.

Данная функция обладает 3 параметрами:

x – координата по горизонтали;

y – координата по вертикали;

c – значение от 1 – 4. Цвет (Зеленый, Красный, Синий, Белый).

То есть для того, чтобы закрасит точку с координатами [5, 7] в белый цвет, нужно вызвать функцию со следующими параметрами:

disp_put_pixel(5, 7, 4);

Функция delay(2); добавляется, так как отправка сообщения по UART занимает некоторое время. И если без остановки передавать данные, то могут возникнуть наложения, которые могут повлиять на работу программы.

Хорошо! Линия нарисована, но что нужно сделать для того, чтобы нарисовать прямоугольник?

Нарисовать 4 линии.

Две по горизонтали, две по вертикали.

То есть, если мы захотим сделать квадрат с вершинами (рисунок 11.6):

1) [3,3]

2) [28, 3]

3) [28 20]

4) [3, 20]

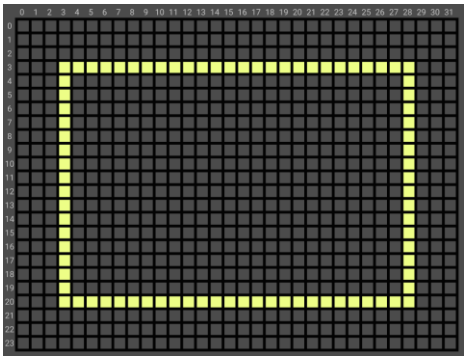


Рисунок 11.6 – Прямоугольник

Если проследить закономерность, то мы можем увидеть, что при переходе из одной вершины в другую у нас всегда сохраняется значение одной из координат, воспользуемся данной закономерностью и напишем следующий программный код:

```
uint16_t x = 3;
uint16_t y = 3;
uint16_t c = 2;
// Рисуем верхнюю горизонтальную линию [3-28, 3]
for(; x < 29; x++){
        disp_put_pixel(x, y, c);
        delay(2);
}
// Рисуем правую вертикальную линию [28, 3-20]
for(; y < 21; y++){
        disp_put_pixel(x, y, c);
        delay(2);
}
// Рисуем нижнюю горизонтальную линию [28 – 3, 20]
for(; x > 2; x--){
        disp_put_pixel(x, y, c);
        delay(2);
}
// Рисуем левую вертикальную линию [3, 20 - 3]
for(; y > 2; y--){
        disp_put_pixel(x, y, c);
        delay(2);
}
```

Таким образом у нас получится квадрат, указанный на рисунке 11.6.

С горизонтальными линиями вроде как все понятно. Одна координата сохраняется, другая же меняет свое значение.

Но что, если придется нарисовать не квадрат, а треугольник (рисунок 11.7).

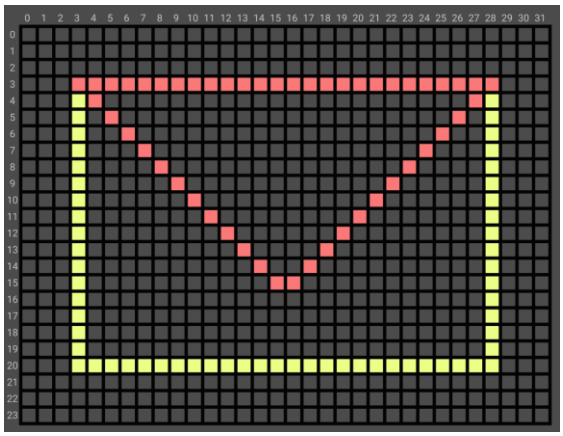


Рисунок 11.7 – Красный треугольник

Для начала, давайте разберём треугольник на три простые линии:

1 горизонтальная линия с вершинами [3, 3] [28, 3];

2 диагональная линия с вершинами [3,3] [15, 15];

3 диагональная линия с вершинами [16, 15] [28, 3].

С горизонтальной линией алгоритм не изменится. Одна координата меняется, другая же стоит на месте.

Но что делать с диагональной?

Со 2 диагональной линией мы сможем легко проследить изменения:

[3,3] [4, 4] [5, 5] . . . . [3-15, 3-15]

То есть в процессе отображения диагональной линии координаты меняются равномерно и на протяжении рисования остаются равными.

С 3 диагональной линией, процесс изменения будет также равномерным, просто координаты не будут равняться друг другу.

[16, 15] [17, 14] [18, 13] [19, 12] . . . . [16-28, 15-3]

То есть здесь мы можем увидеть, что x увеличивается на 1, в то время как y уменьшается на 1 с каждой итерацией.

Давайте напишем программный код, который позволит нам нарисовать данный треугольник:

```
x = 28;
y = 3;
// Рисуем горизонтальную линию [28-3, 3]
for(; x > 2; x--){
        disp_put_pixel(x, y, c);
        delay(2);
}
// Рисуем 1 диагональную линию [3-15, 3 – 15]
for(; x < 16; x++){
        y = x;
        disp_put_pixel(x, y, c);
        delay(2);
}
x++; // Так как значения x, y находится на [15, 15] увеличиваем x для [16, 15]
// Рисуем 2 диагональную линию [16-28, 15-3]
for(; x > 29; x++){
        y--;
disp_put_pixel(x, y, c);
        delay(2);
}
```

С помощью поиска данных закономерностей можно легко построить геометрическую фигуру любой сложности.

Для формирования круглых изображения используется зависимость cos и sin. Так как данную плоскость можно легко представить виде функции, где координата y зависит от координаты x.

Для того, чтобы рисовать линии толще, можно рядом с нарисованной линией нарисовать такую же, что зрительно увеличит толщину данной линии. К примеру:

```
disp_put_pixel(x, y, c);
delay(2);
disp_put_pixel(x+1, y, c);
```

Данная комбинация позволит нарисовать две точки, отличие между которыми будем лишь в том, что она будет дублировать справа на одну позицию.

Практическая часть

В практической работе необходимо будет сформировать изображение из теоретической части, но уже на экране платы TUSUR_IOT_BOARD.

Учитывая, что экран обладает разрешением 320x240 точек.

Если в процессе написания программы значение одной из координат выйдет за пределы значений [320, 240]. То экран загорится красным цветом и изображение удалится.

Для работы вам понадобится библиотека TIB_display.h. Которую вы можете получить у преподавателя.

```
#include "TIB_display.h"
#define but PB4
#define switch1 PC15
#define switch2 PC14
void setup() {
  pinMode(switch1, INPUT);
  pinMode(but, INPUT);
  pinMode(switch2, INPUT);
  Serial.begin(115200);
}
uint16_t x = 160;
uint16_t y = 120;
uint8_t c = 1;
void loop() {
  if (digitalRead(switch1) && !digitalRead(switch2)){
    ВАШ КОД ДЛЯ РИСОВАНИЯ КАРТИНКИ ИЗ ТЕОРИИ
 }
  if (digitalRead(switch2) && !digitalRead(switch1)){
    for(uint16_t i = x; x < i + random(0, 319); x++){
      if (x >= 319) {
        x = 0;
        y = 0;
        c = rand() % 4 + 1;
        break;
      }
      disp_put_pixel(x, y, c);
      delay(2);
    }
    for (uint16_t i = y; y <= i + random(0, 239); y++){
      if (y >= 239) {
        x = 0;
        y = 0;
        c = rand() % 4 + 1;
        break;
      }
      disp_put_pixel(x, y, c);
      delay(2);
    }
  }
  if(digitalRead(but)){
    disp_clear();
```

```
  }
}
```

Для начала работы вставьте следующий программный код:
Загрузите данный код в плату удалив строку, которая выделена жёлтым цветом.
УБЕДИТЕСЬ, что оба свитча выключены.
Далее выберите в меню платы пункт «Вывод на экран» (рисунок 11.8).

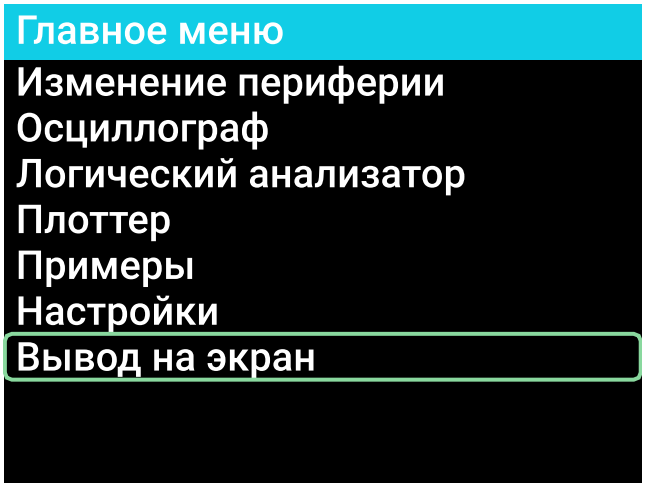


Рисунок 11.8 – Пункт меню «Вывод на экран»

Включите свитч под номером 2.

Так у вас должно выводиться изображение рандомных линий, выходящих из правого верхнего угла. Отключите свитч. Нажмите на жёлтую кнопку. Если экран очистился, значит все работает верно. Приступайте к работе, написав свой код во фрагменте:
«ВАШ КОД ДЛЯ РИСОВАНИЯ КАРТИНКИ ИЗ ТЕОРИИ»

Лабораторная работа №12
Введение в интернет вещей (беспроводную связь). Передача информации по радиоканалу

Цель работы: изучение понятия Интернета Вещей и беспроводной передачи данных и изучение реализации беспроводной передачи данных в ПВС.

Задачи:

1) Изучить понятие Интернет Вещей;

2) Изучить принципы работы беспроводной передачи данных;

3) Изучить возможные модули для реализации беспроводной сети передачи данных в ПВС.

Теоретическая часть

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 актуатором.

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.

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 12.1):

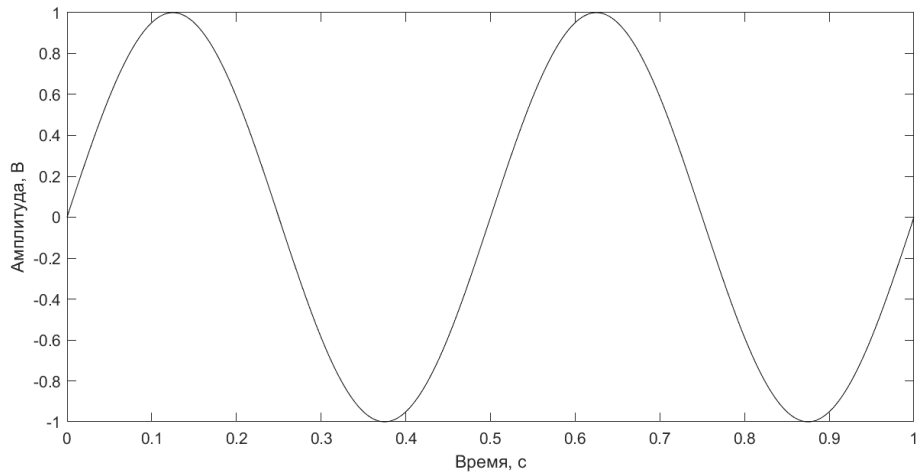


Рисунок 12.1 – Синусоида

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 12.1.

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).

Процесс изменения параметров сигнала для передачи цифрового сигнала называется манипуляцией.

Для примера давайте попробуем промодулировать сигнал используя данные 3 вида манипуляции и с помощью синусоиды передать значение 010110 (рисунок 12.2):

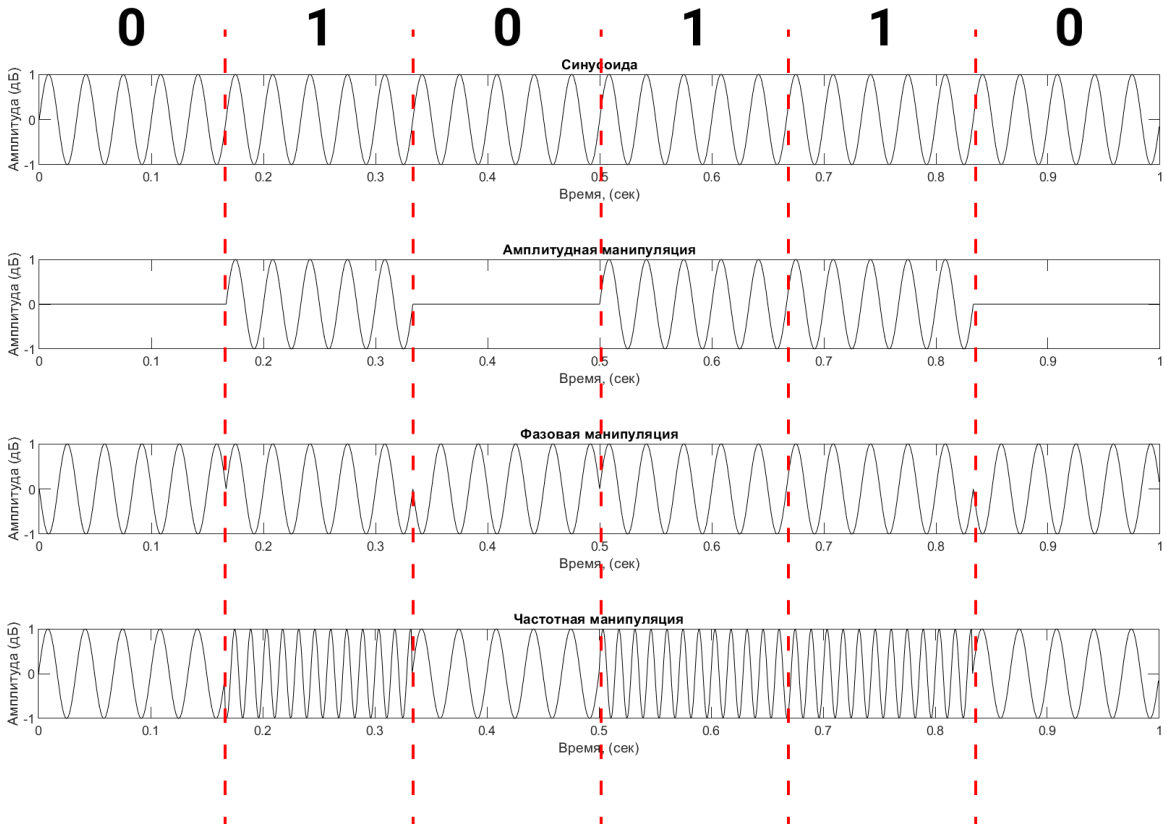


Рисунок 12.2 – Передача данных с использованием разных видов манипуляций

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 12.2.

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.

Практическая часть

Прием сигнала с использованием NRF

В данной работе мы будем использовать беспроводной модуль для передачи информации, который расположен в правом верхнем углу платы (рисунок 12.3):

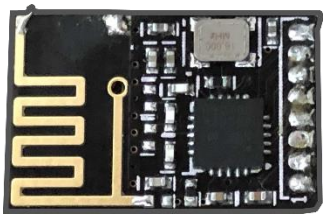


Рисунок 12.3 – Модуль NRF

Данный модуль передает сигнала в нелицензируемой полосе частот 2.4 ГГц, что дает возможность строит 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.

Для подключения библиотек для модулей нужно выполнить следующие задачи:

1) Найти Datasheet на приобретённый модуль;

2) В документации найти подходящую библиотеку и скачать её с официального источника;

3) Чаще всего библиотеки распространяются в сжатых zip папках и в Arduino IDE есть инструмент, который позволяет их удобно устанавливать.

Для начала проверьте. Есть ли на вашем компьютере библиотека RF24. Для этого зайдите в пункт: Скетч -> Подключить библиотеку. И в выпадающем списке попробуйте найти библиотеку RF24 (рисунок 12.4):

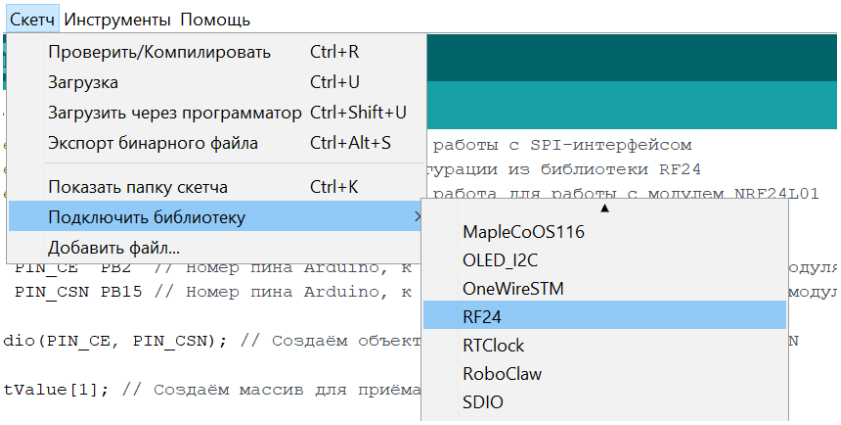


Рисунок 12.4 – Библиотека RF24

Если данная библиотека отсутствует, то воспользуйтесь следующей инструкцией для ее установки:

1) Получить zip архив с библиотекой у преподавателя;

2) Далее воспользуйтесь в меню скетч, воспользуйтесь функцией добавить .zip библиотеку (рисунок 12.5):

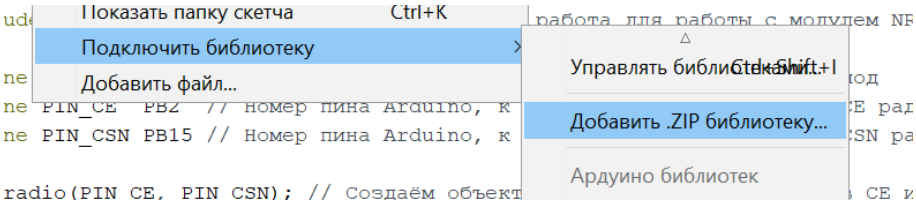


Рисунок 12.5 – Добавление .zip библиотеки

3) В появившемся меню выберите файл RF24–master.zip (рисунок 12.6):

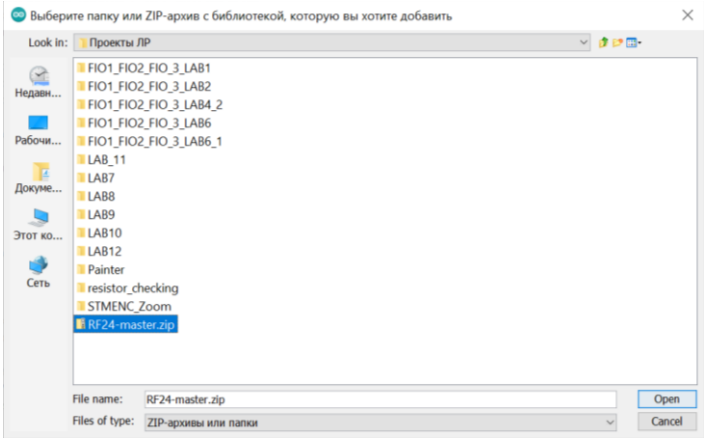


Рисунок 12.6 – Выбор библиотеки RF24

После того, как библиотека будет установлена воспользуйтесь следующим программным кодом для того, чтобы принять сигнал:

```
#include <SPI.h>  // Подключаем библиотеку для работы с SPI-интерфейсом
#include <nRF24L01.h> // Подключаем файл конфигурации из библиотеки RF24
#include <RF24.h> // Подключаем библиотеку для работы с модулем NRF24L01

#define PIN_LED PB8  // Номер пина Arduino, к которому подключен светодиод
#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
тип данных potValue[1]; // Создаём массив для приёма значений потенциометра [0-255]
void setup() {
  //Serial.begin(115200);
  pinMode(PIN_LED, OUTPUT); // Настраиваем на выход пин светодиода
  //KOSTIL
  pinMode(PB5, OUTPUT);
  digitalWrite(PB5, LOW);
  radio.begin();  // Инициализация модуля NRF24L01
  radio.setChannel(5); // Обмен данными будет вестись на пятом канале (2,405 ГГц)
  radio.setDataRate (RF24_1MBPS); // Скорость обмена данными 1 Мбит/сек
  radio.setPALevel(RF24_PA_MAX);  // Выбираем высокую мощность передатчика (0dBm)
  radio.openReadingPipe (1, 0x7878787878LL); // Открываем трубу ID передатчика
  radio.startListening(); // Начинаем прослушивать открываемую трубу
 }
void loop() {
  if(radio.available()){ // Если в буфер приёмника поступили данные
    radio.read(&potValue, sizeof(potValue)); // Читаем показания потенциометра
    //Serial.println(potValue[0]);
    Регулируйте яркость диода
  }
 }
```

Вставьте данный программный код в скетч и закончите его.

Выберите правильный тип данных для переменной, которая будет хранить необходимое значение. С передатчика от преподавателя будут приходить значения от 0-255.

Далее полученное значение будет храниться в переменной potValue[0]. Используя данное значение регулируйте яркость диода.

ВАЖНО! Для работы диода нужно включить его в меню периферии!

После того, как код будет закончен – загрузите его в плату. Если все было написано верно, преподаватель сможет регулировать яркость свечения вашего диода используя беспроводной канал передачи данных.

Для получения задания для дополнительного балла обратитесь к преподавателю.

Лабораторная работа №13
Введение в интернет вещей (беспроводную связь). Передача команд с использованием технологии WiFi

Цель работы: изучение способа создания системы дистанционного управления устройством с использованием технологии Wi-Fi и WEB-сервера

Задачи:

1) Изучить принципы работы Wi-Fi;

2) Изучить принципы работы системы дистанционного управления устройствами;

3) Изучить один из способа реализации дистанционного управления устройствами с использованием WEB-сервера.

Теоретическая часть

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 13.1). 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 13.1 - Разновидности стандарта Wi-Fi

Стандарт	802.11b	802.11g	802.11a	802.11n
Количество используемых неперекрывающихся радиоканалов	3	3	3	11
Частотный диапазон, ГГц	2,4	2,4	5	2,4/5
Максимальная скорость передачи данных в радиоканале, Мбит/с	11	54	54	150–600

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.

Основные недостатки таковы:

- большое энергопотребление и невозможность работы в течение длительного времени от автономных источников питания;

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.

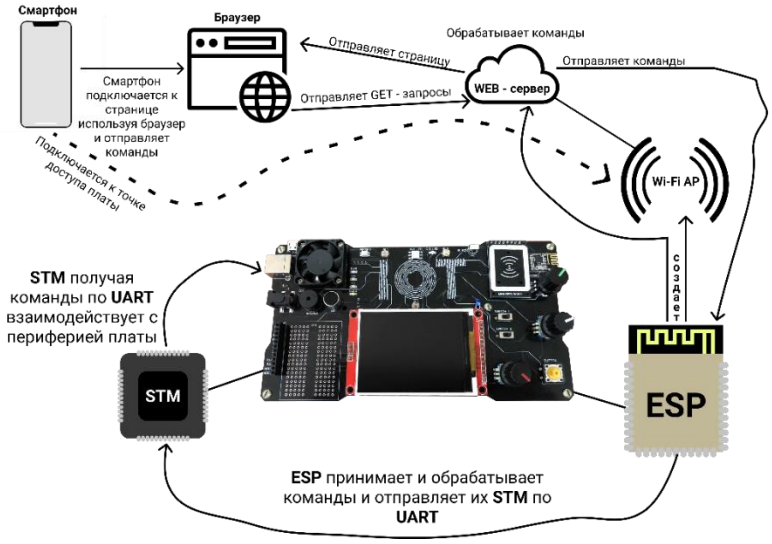


Рисунок 13.1 – Схема подключения

Практическая часть

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 «Настройки»-«Wi-Fi»-«Включить»:

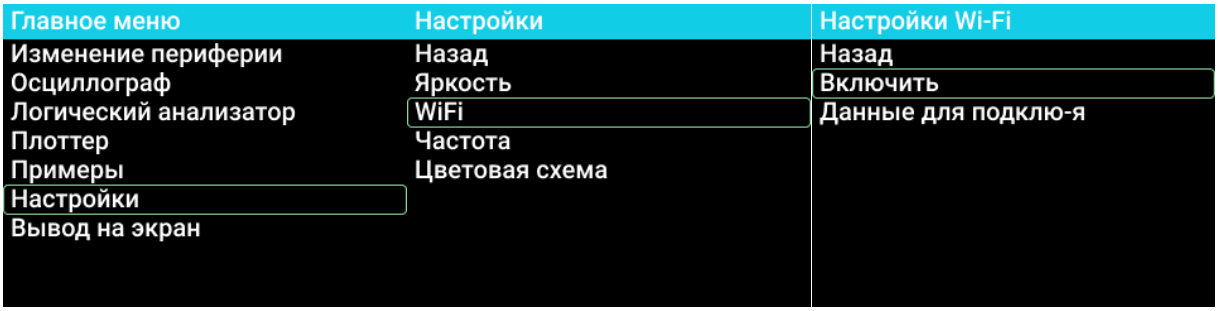


Рисунок 13.2 – Включения Wi-Fi

Убедитесь, что Wi-Fi включился (кнопка «Включить» должна превратиться в кнопку «Выключить»). Зайдите в меню данные для подключения и посмотрите информацию о своей плате. Данные на разных платах могут быть различны!

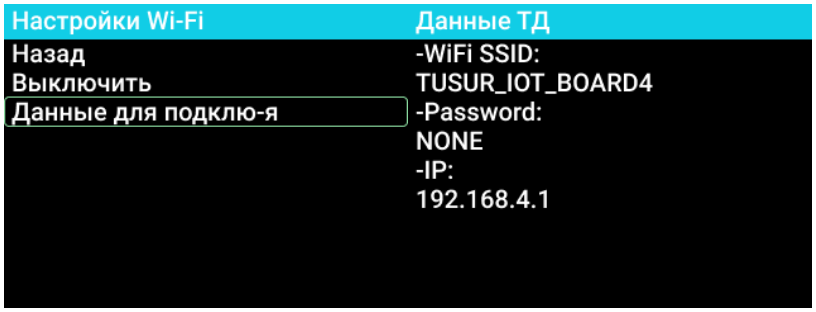


Рисунок 13.3 – Подключение к Wi-Fi

2) Далее воспользуйтесь своим смартфоном для того, чтобы подключиться к сети с именем, которое написано в -WiFi SSID.

3) После того, как вы подключитесь к своей точке доступа зайдите в любой браузер и введите адрес из пункта -IP.

Рисунок 13.4 – Графический интерфейс удаленного управления платой TUSUR_IOT_BOARD

Если у вас отобразился графический интерфейс, представленный на рисунке 13.2 – значит вы все выполнили верно.

Система удаленного управления платой TUSUR_IOT_BOARD способна отправлять 4 команды, которые уже прописаны в данном коде:

```
/*Пины которые можно использовать в проекте*/
#define LED PC13
#define fan  PA1
#define zoomer PA8
#define PR PB0
#define CLK PB12
#define DT PB13
#define SW PA15
#define RGB_G PB1
#define RGB_R PB9
#define RGB_B PB8
#define but PB4
#define switch1 PC15
#define switch2 PC14
#define web_button_1 101U
#define web_button_2 102U
#define web_button_3 103U
#define web_button_4 104U
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  pinMode(zoomer, OUTPUT);
  pinMode(LED, OUTPUT);
  pinMode(RGB_R, OUTPUT);
  pinMode(fan, OUTPUT);
}
void loop() {
  uint8_t value = 0;
  if (Serial.available()){
    value = Serial.read();
  }
  /*Если была нажата кнопка №1*/
  if (value == web_button_1){
    digitalWrite(LED, HIGH); //Включить диод
    delay(1000);
  }
  else{ // Иначе
    digitalWrite(LED, LOW);// Выключить диод
  }
    /*Если была нажата кнопка №2*/
  if (value == web_button_2){
    // Что-то сделать
  }
  else{
    // Что-то сделать
  }
  /*Если была нажата кнопка №2*/
  /*Если была нажата кнопка №3*/
  /*Если была нажата кнопка №4*/
}
```

Скопируйте и загрузите данный код в плату, ничего в нем не меняя.

Проверьте работу данной системы. При нажатии на 1-ую кнопку в интерфейсе на плате должен загореться зеленый светодиод и потом отключиться.

Если все выполнено верно, то закончите данный код, используя свои идеи.

Лабораторная работа №14
Введение в интернет вещей (беспроводную связь).
Инфракрасный приемник и передатчик.

Цель работы: изучение способа создания системы дистанционного управления устройством с использованием технологии передачи сигнала в инфракрасном диапазоне

Задачи:

1) Изучить принципы работы передачи сигнала в инфракрасном диапазоне с использованием ИК диода и приемника;

2) Создать систему распознавания значений отправленных с помощью ИК-пульта;

3) Создать собственный ИК пульт для передачи информации на основе платы TUSUR_IOT_BOARD.

Теоретическая часть

Принцип работы передачи сигнала в инфракрасном диапазоне

Основной принцип работы передачи сигнала в инфракрасном диапазоне очень похож на передачу данных по проводам.

В качестве 0 и 1 выступают отсутствие и наличие сигнала. Принцип передачи похож на передачу информации с помощью света, где существует светодиод, который генерирует лучи света и приемник, которые данные лучи света распознает.

Основное отличие заключается в том, что свет находится в инфракрасном диапазоне (рисунок 14.1):

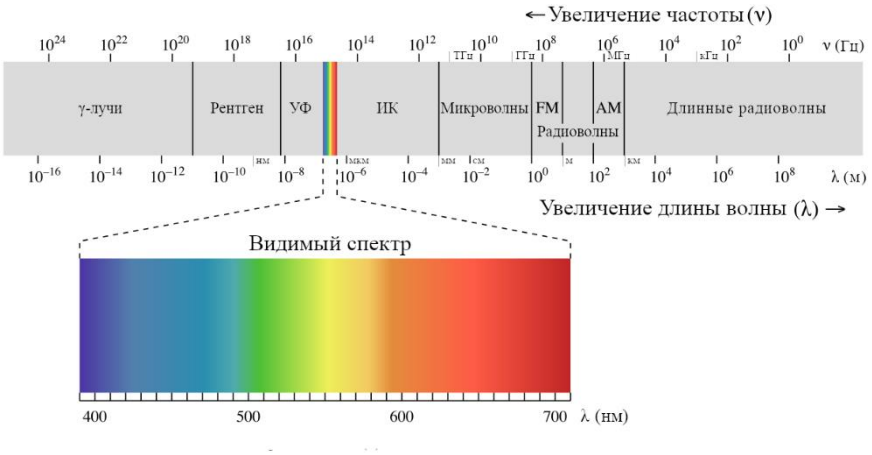


Рисунок 14.1 – Спектр света

Здесь мы можем увидеть, что инфракрасный диапазон не относится к видимому спектру света, который способен распознать человеческий глаз. Что и делает использование данных световых передатчиков комфортным для человека, так как никак не отражается на повседневной жизнедеятельности.

В качестве передатчика в ИК диапазоне используется ИК диод, способный генерировать ИК сигнал.

В качестве приемника используется ИК приемник (по типу фоторезистора для обычного света), который способен распознавать ИК сигналы.

Но в данном виде передачи существует проблема. Солнечный свет и некоторые лампы (рисунок 14.2), которые мы используем как источник света, также являются источниками инфракрасного излучения.

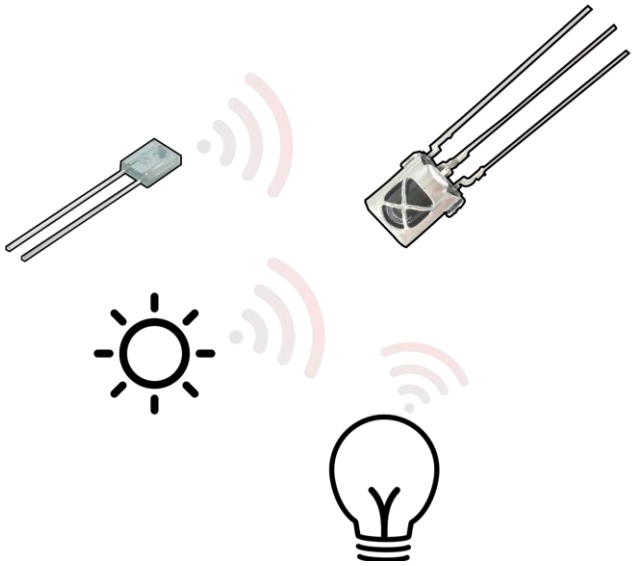


Рисунок 14.2 – Источники ИК излучения

И для уменьшения влияния в ИК приемниках используются фильтры, которые позволяют пропустить лишь сигнал с определенной частотой и тем самым убрать ИК помехи с постоянной составляющей (солнце, лампы и др.).

Данные фильтры уже не позволят ИК диоду генерировать постоянный сигнал и будут требовать генерацию сигнала с определенной частотой (~38 кГц). Данные отличия представлены на рисунке 14.3:

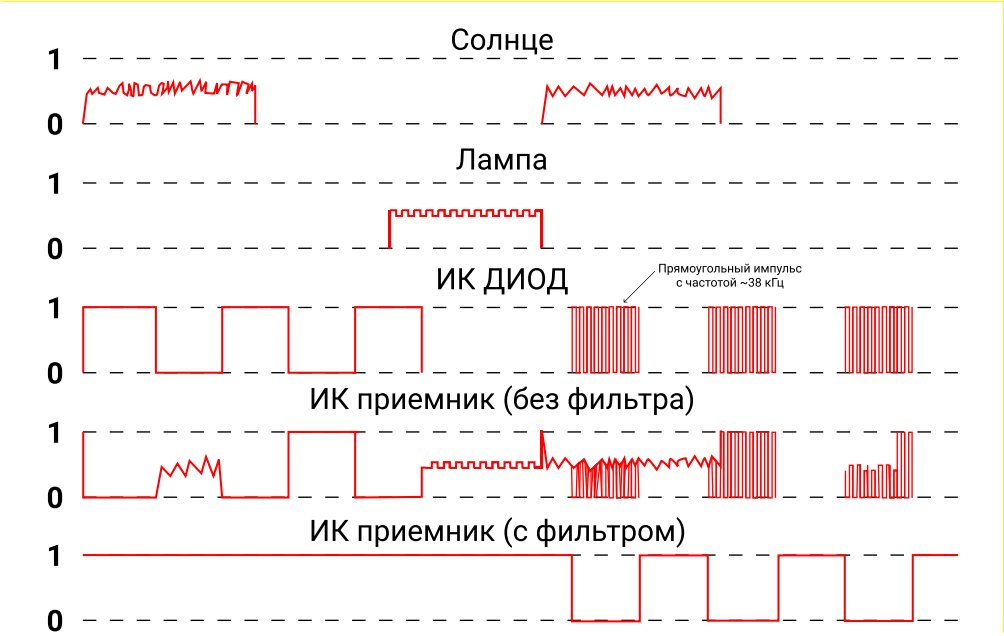


Рисунок 14.3 – Фильтрация ИК помех

С помощью данных действий мы сможем устанавливать верные уровни на ИК приемнике, минуя влияние ИК помех.

Но, как и в любой сети передачи данных нужен протокол, который позволит разработчикам других систем создавать устройства, который смогут взаимодействовать с друг другом.

Существуют несколько протоколов для передачи данных в ИК диапазоне. Чаще всего в простых системах встречается SIRC подобный протокол (рисунок 14.4):

Рисунок 14.4 – SIRC подобный протокол передачи данных в ИК диапазоне

Особенность данного вида протоколов заключается в том, что 0 и 1 кодируется длительностью импульса. Где,

1 – длинный импульс (>1200 мкс);

0 – короткий импульс (<1200 мкс).

Причем правильно считать длительность импульса от момента ухода в 0, до следующего момента ухода в 0.

Для платы TUSUR_IOT_BOARD уже существуют написанные функции, которые позволяют интерпретировать сигналы с ИК пультов для управления светодиодной лентой (рисунок 1.5), а также генерировать их.

Для этого в библиотеке TIB_IR.h присутствуют следующие функции:

void TIB_IR_impuls(uint16_t microsec); - Функция для формирования импульсов с частотой 38 кГц, которая принимает на вход один аргумент – длительность данных импульсов.

uint8_t TIB_IR_intrans(uint8_t value, int pin_tx, int pin_rx); - Функция, которая позволяет передавать данные, используя ИК приемник и передатчик, которые установлены на плате, с помощью UART подобного протокола. Принимает на вход значения и пины, к которым подключены ИК приемник и передатчик. Данная функция возвращает значение принятое ИК приемником.

uint32_t TIB_IR_read(int pin); - Функция, которая позволяет считать значение отправленное ИК пультом для управления RGB лентой. На вход подаётся пни, которому подключен вывод ИК приемника. Данная функция возвращает считанное значение с ИК канала. Если же значение не было получено она возвращает -1.

void TIB_IR_write(uint32_t value); - Функция, которая позволяет передать значение по ИК каналу, используя протокол для ИК пульта для управления RGB лентой.

Практическая часть

Передачи значений, используя ИК канал на плате TUSUR_IOT_BOARD

На плате TUSUR_IOT_BOARD присутствует, как и ИК передатчик, так и ИК приемник (рисунок 14.5):

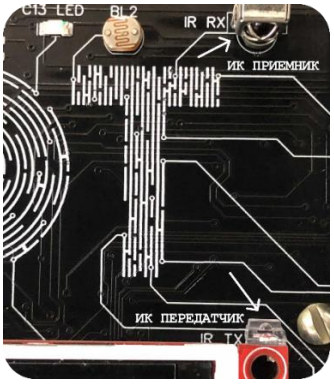


Рисунок 14.5 – Нахождение на плате ИК приемника и передатчика

Так как контроллер не может в один момент времени выполнять две команды, следовательно, нужно применить некоторую хитрость, которая позволит одновременно считывать и передавать значения на одной плате.

Данная хитрость реализована в функции:

```
uint8_t TIB_IR_intrans(uint8_t value, int pin_tx, int pin_rx);
```

Которая находится в библиотеке TIB_IR.h.

Следовательно, она понадобится для реализации данной работы. Добавьте данную библиотеку в свой проект, как вы это делали в прошлых работах.

И вставьте в программу следующий код:

```
#include "TIB_IR.h"
#define IR_RX PB1
#define IR_TX PB9
#define LED PC13
#define but PB4

void setup() {
  // put your setup code here, to run once: 4576 - dsrk 4560 d
  pinMode(IR_RX, INPUT);
  pinMode(but, INPUT);
  pinMode(IR_TX, OUTPUT);
  pinMode(LED, OUTPUT);
  Serial.begin(115200);
}

uint8_t value = 7; // Число, которое вы хотите отправить
void loop() {
  // put your main code here, to run repeatedly
  if(digitalRead(but)){
    uint8_t rec_value = TIB_IR_intrans(value, IR_TX, IR_RX); // В переменной rec_value будет хранится полученное значение
    // Выведите полученное значение на экран
    // Если оно совпадает с тем числом, которое вы хотели
    // отправить выведите на экран сообщение "Отлично!"
    // В ином случае "Ошибка!"
    delay(1000);
  }
}
```

Закончите программный код реализовав задачи выделенные желтым.

После того, как проект будет закончен, докажите, что передача происходит именно по ИК каналу!

Создание системы управления устройством по ИК каналу

В данной работе необходимо сделать систему, которая будет распознавать команды, отправленные ИК пультом.

Для этого необходимо будет воспользоваться функцией:

```
uint32_t TIB_IR_read(int pin);
```

Которая также находится в библиотеке TIB_IR.h.

Для начала выполнения работы воспользуйтесь, следующим программным кодом:

```
#include "TIB_IR.h"
#define IR_RX PB1
#define LED PC13
#define fan  PA1
#define zoomer PA8
void setup() {
  // put your setup code here, to run once: 4576 - dsrk 4560 d
  pinMode(IR_RX, INPUT);
  pinMode(fan, OUTPUT);
  pinMode(zoomer, OUTPUT);
  pinMode(LED, OUTPUT);
  Serial.begin(115200);
}
void loop() {
  // put your main code here, to run repeatedly
  uint32_t current = TIB_IR_read(IR_RX);

  if (current != 0xFFFFFFFF) { // Если что-то пришло
    //Выведите значение считанной переменной
 }
}
```

Скопируйте данный код и выполните задачу, выделенную желтым цветом.

Далее запишите значения двух команд с ИК пульта, которые для вас выберет преподаватель.

Одна из команд будет:

- Включать светодиод,
- Вентилятор,

Другая будет все отключать.

При этом каждое выполнение команды, должно сопровождаться звуком.

Для этого используйте функцию:

```
tone(zoomer, 3000, 10); - Для включения звука
tone(zoomer, 0, 0); - Для выключения звука
```

Для реализации задачи вам понадобится конструкция с условным оператором:

```
if (current == {Значение вашей команды}) {
            }
```

СПИСОК ЛИТЕРАТУРЫ

1. Программирование микроконтроллерных плат Arduino/Freeduino. – СПБ.: БХВ-Петербург, 2012. – 256 с.

2. Язык С в XXI веке / пер. с англ. А.А. Слинкина. – М.: ДМК Пресс, 2015. – 376 с.

3. Язык программирования Си.\Пер. с англ., 3-е изд., испр. – СПб.: “Невский Диалект”, 2001. – 352с.

4. Изучаем Arduino: инструменты и методы технического волшебства. 2-е изд.: пер. с англ. – СПб.: БХВ-Петербург, 2020. – 529 с.

ПРИЛОЖЕНИЕ А
(справочное)

Методическое пособие для создания собственного проекта

Цель работы: формирование базовых навыков создания проектов по теме «Программирование встраиваемых систем».

Задачи работы:

1) Закрепить полученные в ходе курса навыки программирования микроконтроллеров;

2) Получить навыки по созданию собственного проекта;

3) Получить навыки по представлению технических проектов.

Вспомогательный материал для реализации проекта

Для реализации проекта, необходимо подготовить:

1) Идею (название) проекта. Цели и задачи, решаемые проектом;

2) Решения, которые способны реализовать идею проекта, на базе платы TUSUR_IOT_BOARD;

3) Программный код для реализации проекта в среде программирования Arduino IDE;

4) Отчет, который будет отражать этапы подготовки к проекту и пояснения решений для его реализации;

5) Презентация и демонстрация проекта.

Создание идеи проекта

При создании проекта необходимо определить его идею, цели и задачи. В качестве примера можно взять идею данных проектов (Таблица А.1):

Таблица А.1 – Идеи готовых проектов

Идея проекта:	Цель проекта:	Задачи проекта
Система авторизации на базе UART интерфейса	Создание системы аутентификации и идентификации пользователя для доступа к некоторым параметрам системы	1. Создать систему проверки имени пользователя и пароля; 2. Создать систему разграничения доступа для пользователей; 3. Создать систему, которая позволяет сохранять пользователей в памяти устройства.
Система КПП с регистрацией пользователи на базе RFID считывателя	Создание системы способную выполнять функционал КПП с добавлением и удалением пользователей	1. Создать систему считывания данных с RFID метки 2. Создать систему записи данных на RFID метку 3. Создать меню для управления КПП системой
Создание «умного» дома с контролем температурного режима, включения и выключения света по хлопку и т.д.	Создание системы «умного» дома	1. Создать систему автоматической регулировки температурного режима; 2. Создать систему автоматического освещения; 3. Создать систему освещения по «хлопку»; 4. Связать все системы в одну.

Список используемой периферии на плате TUSUR_IOT_BOARD

Список периферии, подключенной к плате представлен в таблице А.2:

Таблица А.2 – Устройства подключенные к плате

ВВОД	ВЫВОД
Датчик температуры	Вентилятор
Датчик давления	Светодиоды
RFID	Экран
IR приемник	Зуммер
Кнопка, переключатели	RGB сетодиод
Encoder	IR передатчик
Микрофон	Динамик
Переменный резистор	
Фоторезисторы	
NRF	
WI-FI	
Bluetooth	
Флешка	

Библиотеки для взаимодействия с некоторым устройствами:
TIB_RFID.h – библиотека для взаимодействия с RFID модулем;
TIB_display.h – библиотека для взаимодействия с экраном с контроллера STM32;
TIB_IR.h – библиотека для взаимодействия ИК каналом.

Данные, которые необходимо отразить в отчёте и в презентации

1) Название проекта, цели и задачи проекта;
2) Ход работы;
3) Исходный программный код с подробным объяснением работы алгоритмов;
4) Тестирование результатов работы проекта.

Отчёт должен быть выполнен по стандарту ОС ТУСУР.

Во время демонстрации нужно также продемонстрировать устойчивость работы проекта к ошибкам и нежелательным воздействиям.

ПРИЛОЖЕНИЕ Б
(справочное)

Установка и настройка лабораторного макета

Установка среды разработки Arduino IDE

1) Переходите по ссылке на официальный сайт поставщика ПО Arduino IDE (Рисунок Б.1) https://www.arduino.cc/en/software;

Рисунок Б.1 – Сайт ардуино.

2) В данной окне выбираете версию своей ОС и скачиваете установщик;

3) Далее запускает установщик и следуя всем инструкциям устанавливаете ПО Arduino IDE.

Добавление платы для программирования контроллера STM в Arduino IDE

Для того, чтобы среда разработки понимала какие настройки необходимо применить для программирования определенного контроллера в нее нужно добавить плату. Для этого можно воспользоваться стандартным менеджером плат.

Но перед этим в настройках нужно указать источник откуда необходимо загрузить данные:

1) Для этого запустите Arduino IDE (Рисунок Б.2):

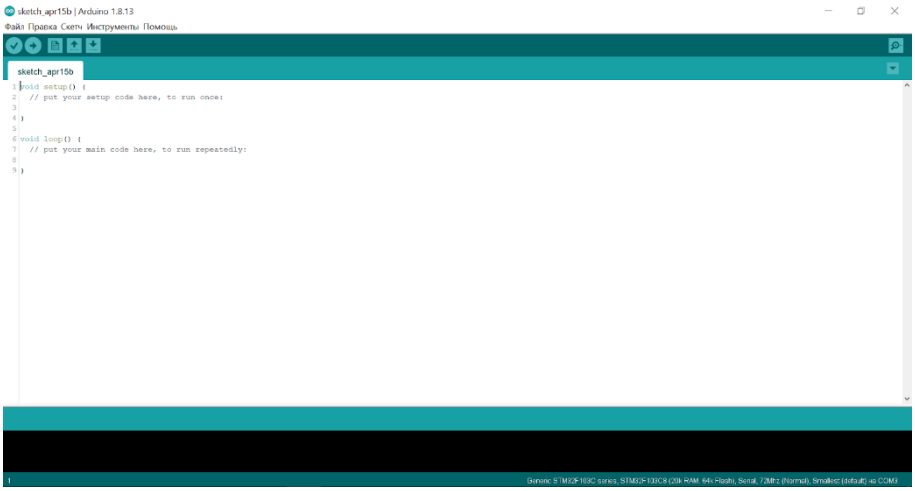


Рисунок Б.2 – Arduiono IDE

2) Перейдите во вкладку «Файл» -> «Настройка» (Рисунок Б.3):

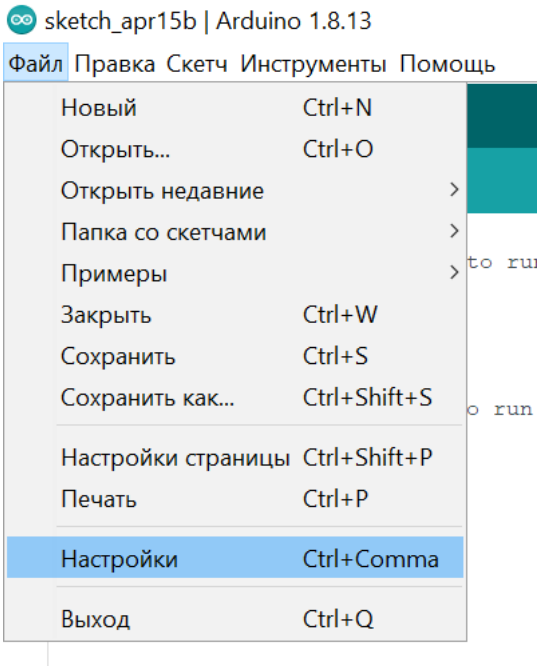


Рисунок Б.3 – Настройки Arduini IDE

3) Далее нажмите кнопку для редактирования дополнительных ссылок для менеджера плат (Рисунок Б.4):

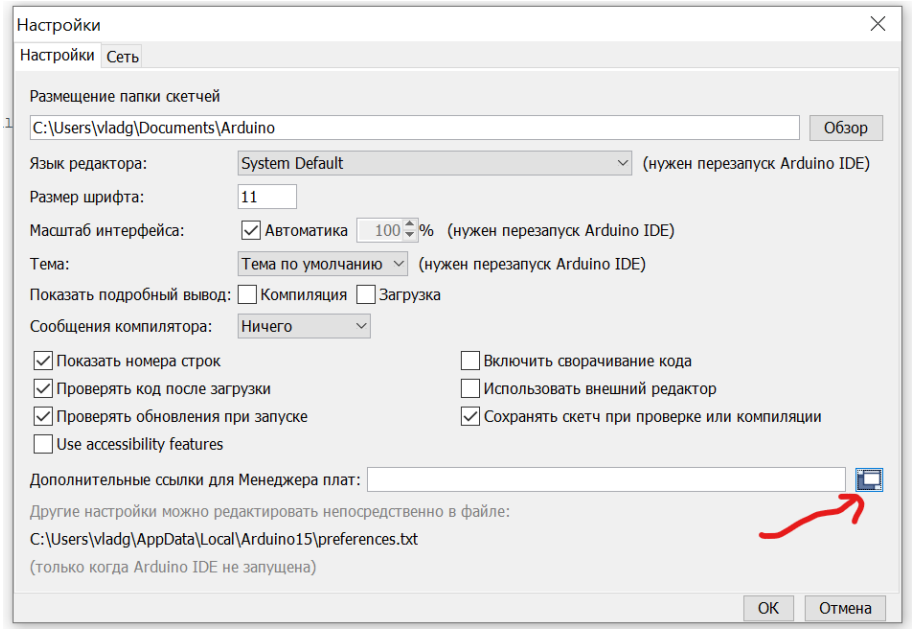


Рисунок Б.4 – Настройка Менеджера плат

4) И вставьте туда строки (Рисунок Б.5):

https://dl.espressif.com/dl/package_esp32_index.json

https://raw.githubusercontent.com/stm32duino/BoardManagerFiles/master/STM32/package_stm_index.json

http://dan.drown.org/stm32duino/package_STM32duino_index.json

http://arduino.esp8266.com/stable/package_esp8266com_index.json

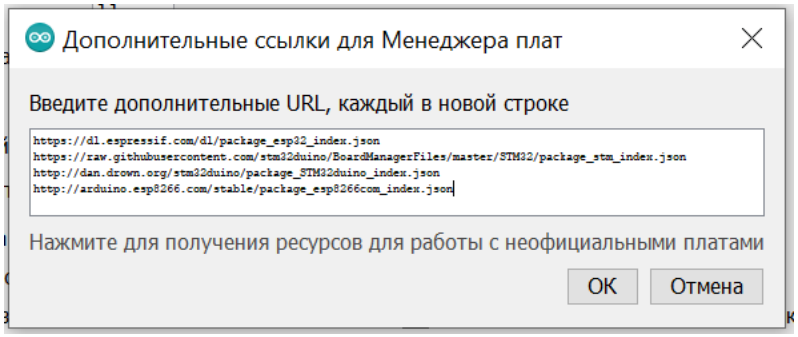


Рисунок Б.5 – Дополнительные ссылки для Менеджера плат

5) Далее перейдите в менеджер плат (Рисунок Б.6):

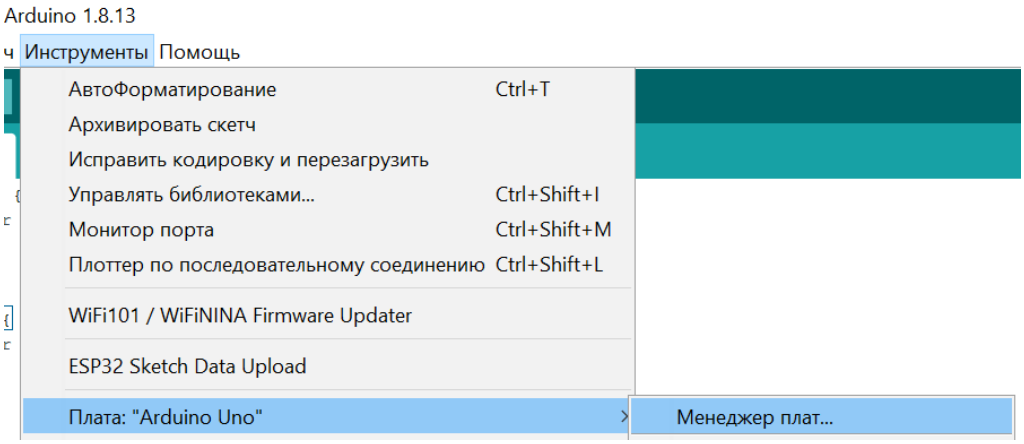


Рисунок Б.6 – Менджер плат

6) И скачайте следующие платы, введя их название в строке поиска (Рисунок Б.7):

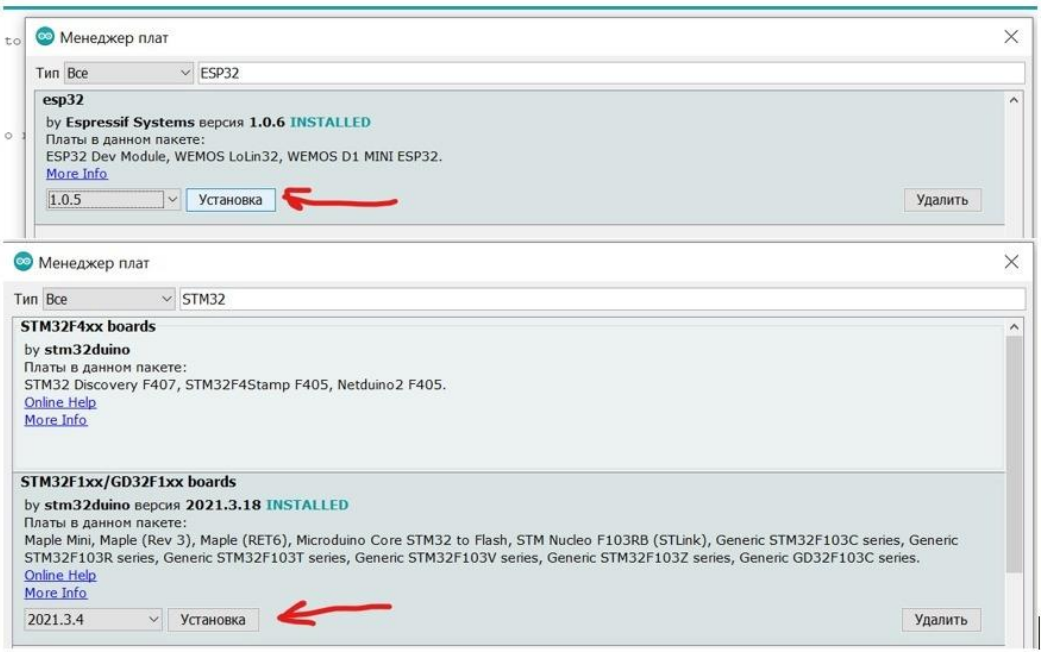


Рисунок Б.7 – Загрузка плат

Программирование STM контроллера

Для программирования STM контроллера в среде программирование ArduinoIDE должны быть установлены, следующие параметры:

Для выбора платы (Рисунок Б.8):

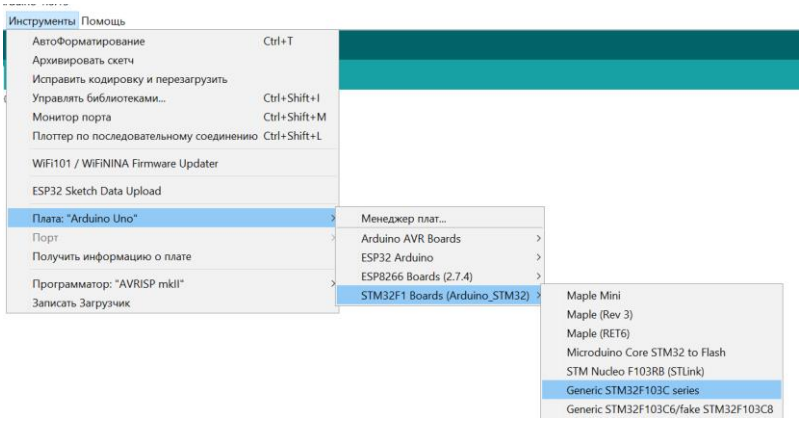


Рисунок Б.8 – Выбор платы

Параметры платы (Рисунок Б.9):

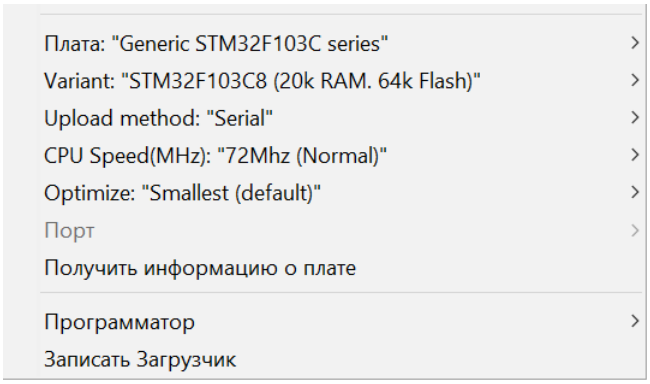


Рисунок Б.9 – Параметры платы

При этом важно проверить расположение перемычек для программирования STM микроконтроллера:

Расположение перемычек (Рисунок Б.10):

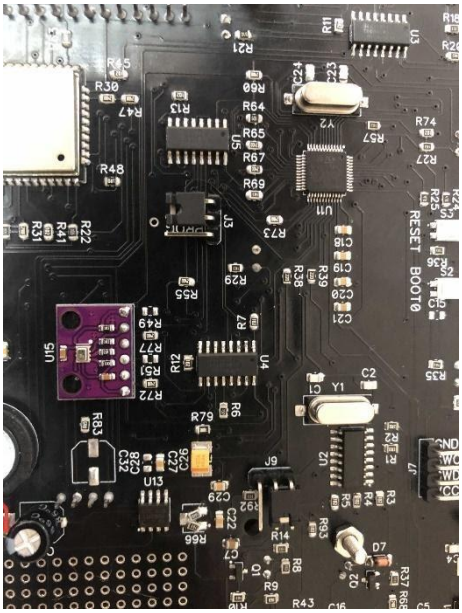


Рисунок Б.10 – Расположение перемычек

Для проверки работы контроллера, можно загрузить прошивку STM_EXAMPLES. При условии, что на контроллере ESP установлена прошивка для управления платой!

И далее в меню платы необходимо выбрать пункт «Примеры»

В ином случае можно загрузить обычный Blink. Для перевода STM в режим загрузки необходимо для начала нажать и удерживать кнопку boot0 и один раз нажать на кнопку RESET (Рисунок Б.11):

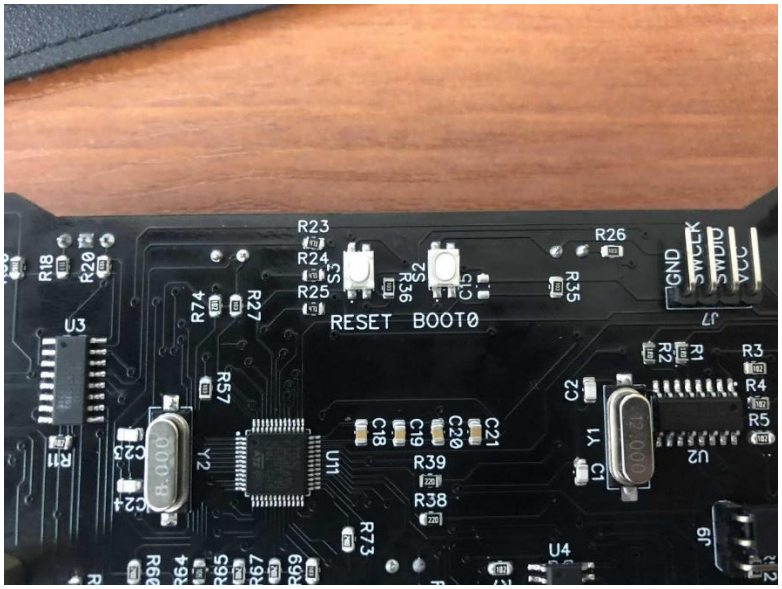


Рисунок Б.11 – Кнопки RESET и BOOT0