



Амперка

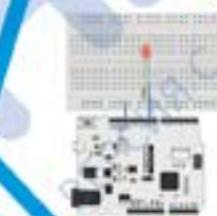
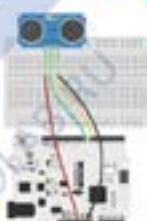
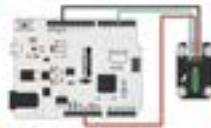
ОСНОВЫ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРОВ

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ
К ОБРАЗОВАТЕЛЬНОМУ НАБОРУ
ПО МИКРОЭЛЕКТРОНИКЕ
«АМПЕРКА»

1
ЧАСТЬ



12-15
ЛЕТ



(БАЗОВЫЙ УРОВЕНЬ)

ОСНОВЫ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРОВ

Артём Бачинин
Василий Панкратов
Виктор Накоряков
под редакцией
Сергея Косаченко

ОСНОВЫ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРОВ

Учебно-методическое пособие
к образовательному набору
по микроэлектронике

«Амперка»



Амперка

ОБРАЗОВАТЕЛЬНЫЙ
РОБОТОТЕХНИЧЕСКИЙ МОДУЛЬ

(БАЗОВЫЙ УРОВЕНЬ)

12 – 15 ЛЕТ



ЭКЗАМЕН
ТЕХНОЛАБ



Издательство
ЭКЗАМЕН®

МОСКВА
2017

УДК 372.8:004

ББК 32.816

Е72

Бачинин А., Панкратов В., Накоряков В., под.ред. Косаченко С.

E72 Основы программирования микроконтроллеров: Учебно-методическое пособие к образовательному набору по микроэлектронике «Амперка»: образовательный робототехнический модуль (базовый уровень): 12 – 15 лет/Артём Бачинин, Василий Панкратов, Виктор Накоряков, под редакцией Сергея Косаченко — М. : Издательство «Экзамен», 2017. — 184 с.

ISBN 978-5-377-10297-7

Учебное пособие является частью образовательного набора «Амперка», который позволяет освоить основы разработки собственных электронных устройств.

Пособие предназначено для проведения занятий среди учеников средних и старших классов. Оно включает как теоретический материал, так и практические упражнения по проектированию электрических схем, программированию микроконтроллеров.

За основу взята плата Iskra Neo, совместимая с популярной платформой для разработки Arduino Leonardo, которая позволяет быстро вникнуть в суть проектирования устройств и на практике разобраться с электронными компонентами и модулями.

Учебное пособие воедино собирает материал, преподаваемый в рамках уроков физики, информатики и технологии, и демонстрирует как можно применить полученные знания в реальном мире.

УДК 372.8:004

ББК 32.816

Подписано в печать с диапозитивов 25.01.2017.

Формат 60x90/8. Гарнитура «Calibri». Бумага офсетная.

Усл. печ. л. 23. Тираж 600 экз. Заказ №

ISBN 978-5-377-10297-7

© Бачинин А., Панкратов В., Накоряков В., Косаченко С., 2017

© Издательство «ЭКЗАМЕН», 2017

© ООО «ЭКЗАМЕН-ТЕХНОЛАБ», 2017

© ООО «АМПЕРКА», 2017

Содержание

1	ЧТО ТАКОЕ МИКРОКОНТРОЛЛЕР?	
1.1	Как научить электронную плату думать	10
1.2	Как сделать электронику проще: Iskra Neo	11
1.3	Как управлять Iskra Neo: среда разработки Arduino IDE	12
1.4	Как заставить Iskra Neo мигать лампочкой: светодиод	13
2	ОБЗОР ЯЗЫКА ПРОГРАММИРОВАНИЯ ARDUINO	
2.1	Процедуры <code>setup</code> и <code>loop</code>	16
2.2	Процедуры <code>pinMode</code> , <code>digitalWrite</code> , <code>delay</code>	19
2.3	Переменные в программе	21
3	ЭЛЕКТРОННЫЕ КОМПОНЕНТЫ	
3.1	Что такое электричество: напряжение и ток	27
3.2	Как укротить электричество: резистор, диод, светодиод	29
3.3	Как быстро строить схемы: макетная доска и мультиметр	30
3.4	Железнодорожный светофор	32
4	ВЕТВЛЕНИЕ ПРОГРАММЫ	
4.1	Условный оператор <code>if</code>	36
4.2	Оператор многозначного выбора <code>switch</code>	37
4.3	Операторы сравнения и логические значения	39
4.4	Цикл с условием <code>while</code> и цикл с параметром <code>for</code>	39
4.5	Как написать свою собственную функцию	41
4.6	Как упростить код: <code>SOS</code> при помощи процедур	43
5	МАССИВЫ И ПЬЕЗОЭЛЕМЕНТЫ	
5.1	Что такое массив	48
5.2	Строки: массивы символов	49
5.3	Воспроизведение произвольных слов на азбуке Морзе	51
5.4	Как пишать на Iskra Neo: пьезоэффект и звук	57
6	СОЕДИНЕНИЕ С КОМПЬЮТЕРОМ	
6.1	Последовательный порт, параллельный порт, UART	66
6.2	Как передавать данные с компьютера на Iskra Neo	67
6.3	Как научить компьютер говорить на азбуке Морзе	70
7	СЕМИСЕГМЕНТНЫЙ ИНДИКАТОР	
7.1	Как работает индикатор	75
7.2	Как включить индикатор	75
7.3	Как научить Iskra Neo считать до десяти	78

Содержание

8	МИКРОСХЕМЫ	
8.1	Зачем нужны микросхемы	82
8.2	Как упростить работу с индикатором: драйвер CD4026	83
8.3	Как сосчитать до 99 при помощи драйвера	84
8.4	Как вывести произвольное число	86
9	ШИМ И СМЕШЕНИЕ ЦВЕТОВ	
9.1	Понятие ШИМ и инертности восприятия	90
9.2	Управление яркостью светодиода	91
9.3	Смешение и восприятие цветов	94
9.4	Радуга из трёхцветного светодиода	95
10	СЕНСОРЫ	
10.1	Что такое сенсоры	100
10.2	Аналоговый и цифровой сигналы	100
10.3	Как распознать наклон: датчик наклона, digitalRead	103
11	КНОПКА — ДАТЧИК НАЖАТИЯ	
11.1	Как работает тактовая кнопка	107
11.2	Как при помощи кнопки зажечь светодиод	107
11.3	Как сделать кнопочный выключатель	111
11.4	Шумы, дребезг, стабилизация сигнала кнопки	113
12	ПЕРЕМЕННЫЕ РЕЗИСТОРЫ	
12.1	Как преобразовать сигнал: делитель напряжения	117
12.2	Как делить напряжение «на ходу»: потенциометр	119
12.3	Как Iskra Neo видит свет: фоторезистор	122
12.4	Как измерить температуру: термистор	124
13	ДАЛЬНОМЕРЫ	
13.1	Ультразвуковой дальномер	128
13.2	Характеристики ультразвукового дальномера HC-SR04	129
13.3	Выводы ультразвукового дальномера HC-SR04	129
13.4	Как это работает	132
14	ЖИДКОКРИСТАЛЛИЧЕСКИЕ ЭКРАНЫ	
14.1	Как работает текстовый дисплей	134
14.2	Как вывести приветствие: библиотека, класс, объект	137
14.3	Как вывести русскую надпись	141

Содержание

15 ДВИГАТЕЛИ	
15.1 Разновидности двигателей: постоянные, шаговые, серво	145
15.2 Как управлять серводвигателем с Iskra Neo	147
16 ТРАНЗИСТОРЫ	
16.1 Как управлять электричеством: транзистор	152
16.2 Разновидности транзисторов	152
16.3 Как вращать двигатель	154
16.4 Как управлять скоростью двигателя	157
17 ISKRA NEO И ИНТЕРНЕТ ВЕЩЕЙ	
17 ISKRA NEO и интернет вещей	161
17.1 Модуль ESP8266 с поддержкой WiFi	162
17.2 Как проверить модуль ESP8266	164
17.3 Управление модулем ESP8266	164
17.4 Как сделать умный датчик температуры	170
17.5 Как это работает	180
17.6 Перспективы использования модуля ESP8266	183



Введение

Ежедневно все мы встречаемся с электронными устройствами. Устройства эти очень разнообразны по своему назначению. Чуть ли не для каждого действия в нашей жизни существует устройство, которое его облегчает. Нужно с кем-то поговорить на расстоянии — мы берём телефон; постирать вещи — бросаем их в стиральную машину; узнать новости или развлечься — подключаемся к интернету или включаем телевизор.

Невероятное количество устройств работает на благо нам, не бросаясь при этом в глаза. Автомобили и самолёты нафаршированы электронной начинкой, обеспечивающей защиту от внештатных ситуаций. Светофоры контролируют движение без участия человека. Сортировочные машины в аэропорту автоматически отправляют наш багаж на нужный рейс по сложной системе конвейерных лент. Роботы-пылесосы стали уже обычным бытовым устройством. Список можно продолжать бесконечно.

Бессчётное количество электронных устройств уже было создано, но прогресс не стоит на месте, а потому они будут продолжать изобретаться, строиться, совершенствоваться. Как показывает нам история, это будет происходить всё более и более быстрыми темпами. Почему? Потому что человечество уже накопило достаточное количество наработок, чтобы изобретение новых устройств перестало быть уделом «безумных гениев»: теперь это стало настолько просто, что даже ребёнок может собрать что-то своё, уникальное.

Совместимая с платформой Arduino плата Iskra Neo — яркий пример шага в сторону упрощения разработки. Человек, который берёт её в качестве основы своего изобретения, может лучше сосредоточиться на самой сути своего устройства, на его функциональности, удобстве, надёжности, дизайне. При этом изобретатель может не сильно вдаваться в сложные аспекты схемотехники и программирования на низком уровне. Всё это было абсолютно необходимо раньше, но теперь достаточно базовых знаний, по крайней мере на этапе создания рабочего прототипа.

Данная книга в совокупности с «железом» из образовательного набора «Амперка» позволит новичку, не имевшему ранее опыта создания собственных электронных устройств, войти в этот мир и получить все необходимые знания, чтобы стать изобретателем.

Один параграф — одно занятие. По ходу его изучения своими руками будет создаваться и программироваться как минимум однорабочее электронное устройство. Объединив в конце все полученные знания, можно будет уверенно приступать к созданию чего-то большого, сложного, полезного, интересного и умного. Желаем в этом удачи!

Подготовка к работе

Платформа разработки Arduino, с которой мы и будем иметь дело на протяжении всего курса приобрела свою популярность прежде всего из-за простоты использования.

Технически для работы с ней нужен лишь обычный USB-кабель и компьютер с операционной системой Windows, Linux или Mac OS. Чтобы программировать микроконтроллер на плате и общаться с ней, необходимо произвести минимальное количество действий по подготовке компьютера.

Установка Arduino IDE

Для начала необходимо установить среду для программирования Arduino IDE. Это программа, которая распространяется производителем свободно и, на самом деле, её даже не нужно устанавливать в привычном смысле этого слова. Достаточно просто скопировать директорию с ней на жёсткий диск в любое место, например, на «рабочий стол».

Вы можете скачать последнюю версию Arduino IDE с официального сайта:

<http://arduino.cc/en/Main/Software>

Выберите версию, соответствующую вашей операционной системе и после окончания загрузки, распакуйте архив, как было сказано, в любую директорию.

Также вы можете взять Arduino IDE с прилагающегося к книге диска.

Установка драйверов

Если на рабочем компьютере установлена операционная система Linux или Mac OS, вам повезло, установка драйверов не требуется: всё заработает из коробки. Если у вас Windows:

1. Соедините компьютер с Iskra Neo через USB-кабель. На плате загорится зелёный светодиод ON
2. Откройте «диспетчер устройств»: Пуск → Панель управления → Система → Диспетчер устройств
3. В дереве устройств, в ветке «Порты (COM&LPT)» вы найдёте устройство «Arduino Leonardo (COMxx)»
4. В меню, выпадающем по правому клику на нём, выберите «Обновить драйвер»
5. В открывшемся окне укажите, что укажете расположение драйвера самостоятельно
6. В качестве местоположения укажите директорию Drivers, находящуюся внутри директории с Arduino IDE, которую вы только что установили. Вам нужна именно директория Drivers, а не её поддиректория FTDI.
7. Продолжите установку и дождитесь её окончания

Установка примеров

Все примеры, которые приводятся в учебнике, существуют в виде уже готовых отдельных файлов, которые можно просто открыть. Нелепо машинально перепечатывать программу с бумаги, лучше потратить это время, сосредоточившись на действительно важных вещах.

Чтобы примеры были доступны непосредственно из среды программирования Arduino IDE, их нужно установить. Если вы устанавливали Arduino IDE с прилагаемого диска, ничего дополнительно не нужно: они уже встроены.

Если вы брали среду с официального сайта, перейдите в директорию `libraries`, расположенную внутри директории с установленной Arduino IDE. Внутрь этой директории скопируйте директорию `Amperka`, которую вы можете найти на прилагаемом диске, внутри директории `examples` или скачать с сайта:

<http://teacher.amperka.ru/examples.zip>

В процессе экспериментов учениками, в зависимости от настроек файловой системы, текст программ может быть ими изменён или испорчен. Чтобы реанимировать изменения достаточно заново установить примеры: перезаписать директорию `Amperka` прямо поверх имеющейся.

Проверка работоспособности

1. Соедините компьютер с Iskra Neo через USB-кабель
2. Запустите Arduino IDE файлом `arduino.exe` (в Windows) или просто `arduino` (в Linux или Mac OS)
3. Откройте файл-пример через меню `File → Examples → Amperka → p02_blink`. Если подменю Amperka не оказалось, убедитесь, что вы внимательно соблюли инструкции по установке примеров
4. Нажмите на панели инструментов кнопку прошивки 
5. В нижней части окна через несколько секунд появится надпись «Done Uploading», а Iskra Neo будет мигать светодиодом, отмеченным символом L. Если произошла ошибка, убедитесь, что в меню `Tools → Serial port` выбран именно тот порт, на котором расположена Iskra Neo (Arduino Leonardo). Если меню не активно или порта с Iskra Neo (Arduino Leonardo) в нём нет, убедитесь, что установили драйвер правильно.

Теперь всё готово. Поехали!





§1. ЧТО ТАКОЕ МИКРОКОНТРОЛЛЕР?

- 1.1 Как научить электронную плату думать: микроконтроллеры и компьютеры
- 1.2. Как сделать электронику проще: Iskra Neo
- 1.3 Как управлять Iskra Neo: среда разработки Arduino IDE
- 1.4. Как заставить Iskra Neo мигать лампочкой: светодиод

ЧТО ТАКОЕ МИКРОКОНТРОЛЛЕР?

§1



§1. ЧТО ТАКОЕ МИКРОКОНТРОЛЛЕР?

1.1 Как научить электронную плату думать: микроконтроллеры и компьютеры

Это пособие позволит нам вместе с вами погрузиться в волшебный мир электроники. Мы будем зажигать лампочки, воспроизводить звук, отображать текст. Мы научим электронную плату реагировать на свет, тепло, наклон. Мы научимся подключать кнопки и регуляторы. Мы будем вращать двигатели и поворачивать их на строго заданный угол. И, наконец, соберём настоящего робота, который будет ездить вдоль нарисованной вами линии.

Всё это было бы довольно сложно или даже практически невозможно без одного очень важного компонента — микроконтроллера. Давайте, потому, для начала поймём, что это такое, и как это работает.

Каждый из вас, наверное, имеет представление о том, как работает компьютер. Он воспринимает информацию из внешнего мира, «обдумывает» её и как-то реагирует. Скажем, если вы нажмёте клавишу на клавиатуре, на экране компьютера может появиться определённая буква. Мы также можем подключить к компьютеру вентилятор и термометр и заставить его включать вентилятор, когда температура воздуха будет выше двадцати градусов. Современные компьютеры позволяют выполнять множество задач одновременно. При этом каждая задача может быть много сложнее перечисленных выше.

Однако компьютер часто бывает не лучшим решением поставленной задачи. Во-первых, из-за цены, а во-вторых, из-за размера. Сложно разместить компьютер на радиоуправляемом самолёте. А если мы хотим просто включать вентилятор в зависимости от температуры, то компьютер будет явно дорогим удовольствием. В таких ситуациях лучше всего использовать микроконтроллер. Он менее мощный, способен выполнять лишь одну задачу, но зато компактный и дешёвый.



Рис. 1.1: Популярный микроконтроллер ATmega328p

Микроконтроллер — это небольшая микросхема, в которой уже есть процессор, оперативная память и флеш-память. Они бывают разной формы, немного различаются возможностями и производительностью, но суть остаётся неизменной.

Микроконтроллер часто является мозговым центром платы, отвечающей за определённую задачу: на него приходят все сигналы, поступающие на плату, а он в свою очередь раздаёт команды всем устройствам, подключённым к ней.

Микроконтроллеры используются повсеместно: от бытовых кухонных устройств до элементов управления космическим кораблём, от домофонов до систем безопасности в автомобиле, от радиоуправляемых игрушек до роботов на конвейере завода.

Микроконтроллер, как и компьютер, можно программировать, т.е. задавать ему логику поведения. Скажем, мы можем запрограммировать микроконтроллер таким образом, чтобы он включал вентилятор, когда температура воздуха больше двадцати градусов, а можем сделать, наоборот, чтобы вентилятор включался, когда станет холоднее двадцати градусов. В конце этой главы мы запрограммируем нашу плату, чтобы она включала и выключала светодиод (лампочку на плате) каждую секунду.

1.2. Как сделать электроннику проще: Iskra Neo



Рис. 1.2: Плата Iskra NEO от фирмы Амперка совместимая с Arduino Leonardo

Итак, допустим, у нас есть микроконтроллер. Как, например, заставить лампочку мигать раз в секунду?

Для этого нам нужно написать сложный программный код, управляющий регистрами микроконтроллера. Затем через специальное устройство, называющееся программатором, загрузить этот код в микросхему. Потом спаять электронную плату со множеством компонентов, предварительно высчитав характеристики каждого, впаять в неё наш контроллер, подключить к сети. Если вдруг что-то не заработает, нужно будет снова всё выпаивать и делать заново.

Электронная плата *Iskra Neo* (рис. 1.2), совместимая с *Arduino Leonardo*, позволяет заметно упростить весь этот процесс. Все нужные компоненты на ней уже есть. Программный код получается намного проще из-за того, что большая его часть уже написана.

Для микроконтроллера не нужен специальный программатор, т.к. программы загружаются на микроконтроллер через USB при помощи специального приложения на компьютере. Для различных устройств есть специальные разъёмы, чтобы не нужно было ничего припаивать. К тому же Arduino — это мировой стандарт с открытыми спецификациями. А поэтому для неё существует большое множество уже готовых программ и различных дополнений, так называемых мезонинных плат, при соединении с которыми Iskra Neo получает новые возможности.

Таким образом, используя Iskra Neo, мы можем заставить мигать лампочку, просто подключив её к компьютеру через USB и залив на неё необходимый программный код.

1.3 Как управлять Iskra Neo: среда разработки Arduino IDE

Давайте разберёмся, как управлять Iskra Neo. Для этого нам понадобится специальная программа, которая называется средой разработки. Среда разработки нужна для удобства программирования.

Она уже должна быть установлена на вашем компьютере, если были выполнены все действия, описанные в введении к книге. Зайдите в директорию, где была установлена среда и запустите файл arduino.exe (или просто arduino, если у вас MacOS или Linux). Появится окно среды Arduino IDE (как на рис. 1.3).

В центральную часть этого окна пишется программный код для управления микроконтроллером. Такой код также называется скетчом. Программный код — это просто текст, написанный на специальном языке, который понятен и человеку, и компьютеру.

Нажав на кнопку «Verify»  , вы сможете проверить свой код на правильность. Если после нажатия на неё в нижней части окна появится надпись «Done compiling», значит, вы написали всё правильно (рис. 1.4). В противном случае вы увидите сообщение об ошибке.

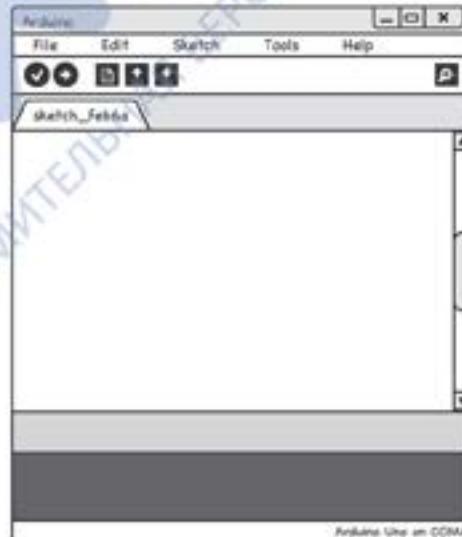
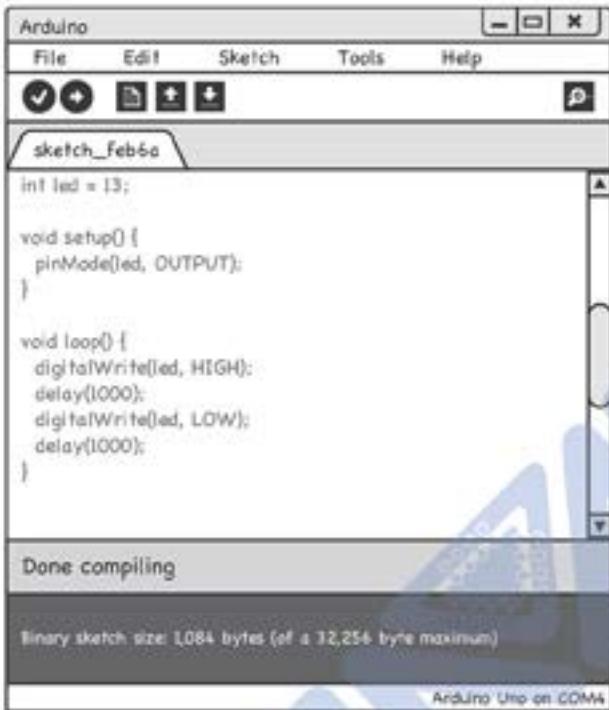


Рис. 1.3: Окно среды Arduino IDE при запуске



The screenshot shows the Arduino IDE interface with the following details:

- Title Bar:** Arduino
- Menu Bar:** File, Edit, Sketch, Tools, Help
- Toolbar:** Includes icons for Open, Save, Print, and Upload.
- Sketch Area:** Displays the code for "sketch_feb20" with the following content:


```
int led = 13;

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```
- Status Bar:** Done compiling
Binary sketch size: 1084 bytes (of a 32,256 byte maximum)
- Bottom Status:** Arduino Uno on COM4

Рис. 1.4: Информация об успешной компиляции отражается в нижней части окна

1.4. Как заставить Iskra Neo мигать лампочкой: светодиод

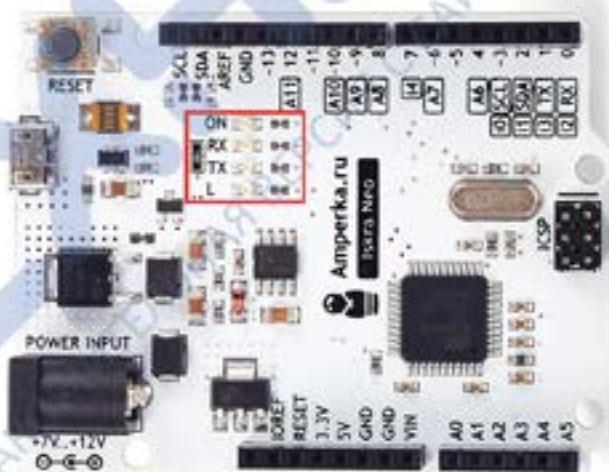


Рис. 1.5: Светодиоды на плате Iskra Neo

Чтобы написанный вами скетч перенёсся с компьютера на микроконтроллер Iskra Neo, нужно загрузить его. Процесс загрузки также называют прошивкой или заливкой. Это можно сделать при помощи кнопки «Upload» ●. Точно также при удачной загрузке внизу появится строка «Done uploading».

Если вы посмотрите на плату Iskra Neo, то увидите на ней несколько маленьких лампочек, которые называются светодиодами (см. рис. 1.5). Подробнее с ними мы познакомимся в последующих главах. А пока давайте заставим один из них мигать, т.е. включаться и выключаться каждую секунду.

Для этого подключите вашу плату через USB к компьютеру. На Iskra Neo загорится светодиод, сигнализирующий о том, что плата подключена и готова к работе.

Вставьте в среду разработки следующий код:¹

```
void setup()
{ pinMode(13, OUTPUT);
}
void loop()
{ digitalWrite(13, HIGH);
delay(1000);
digitalWrite(13, LOW);
delay(1000);
}
```

Нажмите кнопку Upload ●. Подождите некоторое время. Светодиод, подписанный буквой L, начнёт мигать.

Это успех, это ваша первая работающая программа для микроконтроллера!

В последующих главах мы подробно разберём, почему и как это работает.



¹ Код можно также найти через меню Arduino IDE: File → Examples → Amperka → p01_blink



ОБЗОР ЯЗЫКА ПРОГРАММИРОВАНИЯ ARDUINO

§2



```

Arduino
File Edit Sketch Tools Help
sketch_Feb06
int led = 13;

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}

Done compiling
Binary sketch size: 1084 bytes (of a 32,256 byte maximum)
Arduino Uno on COM4

```



2.1 Как правильно написать программу: процедуры `setup` и `loop`

2.2 Как управлять подключенным устройством: процедуры `pinMode`, `digitalWrite`, `delay`

2.3 Как сообщить о бедствии при помощи светодиода: переменные в программе

§2. ОБЗОР ЯЗЫКА ПРОГРАММИРОВАНИЯ ARDUINO

2.1 Как правильно написать программу: процедуры `setup` и `loop`

Первая программа

Давайте напишем простейшую программу на Arduino. Для этого откройте среду разработки и введите в открывшемся окне следующий текст:

```
void setup()
{
}
void loop()
{
}
```

Нажмите кнопку `Verify` и удостоверьтесь, что внизу появилась надпись «Done compiling». Она означает, что вы не допустили ошибок при написании кода (см. рис. 1.4).

Как это работает

Наша программа состоит из двух процедур, каждая из которых имеет следующий вид:

```
void названиеПроцедуры()
{
}
```

Одна процедура называется `setup` (перевод с английского “установка, настройка”) и выполняется один раз при включении платы, а другая — `loop` (перевод с английского “петля, цикл”) выполняется бесконечно. Эти процедуры обязательно должны быть в программе, иначе компилятор выдаст ошибку. В написанной программе процедуры пусты: нет ничего между фигурными скобками, поэтому программа не делает ничего. Это минимально возможная программа для Iskra Neo. Можете считать её «чистым листом», который нужно дополнять, чтобы получить нечто полезное.

Мигающий светодиод

Теперь давайте напишем более сложную программу, которая будет управлять включением и выключением светодиода на вашей плате Iskra Neo. Для этого вставьте в окно среды разработки следующий код:²

```
/*
Мигание
```

² Вы также можете открыть этот код через меню: File → Examples → Amperka → p02_blink

Повторяющееся включение и отключение светодиода
с периодом в одну секунду

```
/*
void setup()
{ pinMode(13, OUTPUT);
}
void loop()
{
digitalWrite(13, HIGH); // включаем светодиод
delay(1000); // ждём секунду
digitalWrite(13, LOW); // выключаем светодиод
delay(1000); // ждём секунду
}
```

Процедуры **setup** и **loop**

Снова нажмите Verify и убедитесь, что внизу появилась надпись «Done Compiling». После этого можно “прошивать” код на плату, т.е. загрузить программу в память контроллера. Для этого подключите плату Iskra Neo к компьютеру через USB-кабель и нажмите на кнопку Upload . Внизу должна появиться надпись «Done uploading». Это означает, что процесс прошивки прошёл успешно.

Если на этапе загрузки вы получили сообщение об ошибке, убедитесь, что в меню Tools → Board выбрана совместимая модель платы, с которой вы работаете: Arduino Leonardo. Проверьте, правильно ли выбран последовательный порт, к которому подключена Iskra Neo через меню Tools → Serial port: если вариантов несколько, попробуйте выбирать разные. После изменения настроек нужно вновь нажать Verify и Upload.

После того, как вы успешно залили код, вы увидите, что светодиод на вашей Iskra Neo начал мигать. Давайте поймём, почему это произошло.

Как это работает

Попробуем последовательно разобрать написанный код. В начале программы вы видите следующие строки:

```
/*
```

Мигание

Повторяющееся включение и отключение светодиода
с периодом в одну секунду

*/

Это так называемые комментарии, они имеют следующий вид:

/*

Здесь вы можете писать любой текст,
состоящий из любого количества строк

*/

или

// здесь вы можете писать любой текст в 1 строку

В комментариях вы можете писать всё, что угодно. Они никак не влияют на работу программы, компилятор их игнорирует. Комментарии используются для собственных пометок или пояснений. Это удобно для программиста.

Далее идёт процедура `setup`.³

```
void setup()
{
    pinMode(13, OUTPUT);
}
```

Эта процедура выполняется единожды в начале работы программы: сразу после перезапуска микроконтроллера, подачи питания или перепрошивки. Последовательно выполняется каждая команда, заключённая между фигурными скобками этой процедуры. Каждая команда должна завершаться символом точки с запятой «;».

Как мы видим, в нашем случае процедура `setup` содержит одну единственную команду — `pinMode(13, OUTPUT)`. Мы расскажем, зачем нужна эта команда, чуть ниже.

После процедуры `setup` выполняется процедура `loop`.⁴

```
void loop()
{
    digitalWrite(13, HIGH); // включаем светодиод
    delay(1000);           // ждём секунду
    digitalWrite(13, LOW);  // выключаем светодиод
    delay(1000);           // ждём секунду
}
```

В отличие от `setup`, процедура `loop` постоянно повторяется. То есть, как только последовательно выполнены все команды, которые содержатся в этой процедуре, она запускается снова. В нашем случае процедура `loop` состоит из четырёх команд:

³ `setup` — установка, настройка (англ.)

⁴ `loop` — петля, цикл (англ.)

- Сначала выполняется команда включения светодиода
- Затем — команда сна (ожидания) на одну секунду
- После чего выполняется команда выключения светодиода
- И в конце — сон (ожидание) на ещё одну секунду
- После выполнения всех четырёх команд, снова выполняется первая команда (включение светодиода), и так до тех пор, пока включена Iskra Neo.

2.2 Как управлять подключенным устройством: процедуры `pinMode`, `digitalWrite`, `delay`

Давайте ещё раз взглянем на программу, которую мы только что написали, и попытаемся в деталях понять, что в ней происходит. Начнём с процедуры `Setup`.

```
void setup()
{
    pinMode(13, OUTPUT);
}
```

Как мы видим, данная процедура содержит одну-единственную команду `pinMode(13, OUTPUT)`. Это вызов встроенной процедуры `pinMode` с аргументами `13` и `OUTPUT`. Аргумент — это то, что передаётся в процедуру для уточнения того, что мы хотим сказать командой.

Вызов любой процедуры в Arduino происходит следующим образом:

имяПроцедуры(Аргумент1, Аргумент2, Аргумент3, ...);

Количество аргументов может варьироваться в зависимости от процедуры. Некоторые процедуры вообще не требуют аргументов.

Такими, например, являются процедуры `setup` и `loop`: вы видели, круглые скобки рядом с их названием открываются и тут же закрываются без всяческого перечисления аргументов.

Прежде чем понять, что же делает процедура `pinMode`, давайте разберём общие принципы работы Iskra Neo. Если вы взглянете на плату, то увидите множество пронумерованных контактов. Эти контакты мы будем называть пинами.⁵ К каждому контакту может подсоединяться какое-нибудь устройство. В нашем случае — светодиод. Iskra Neo может общаться с этим устройством посредством подачи на него электрического напряжения. В таком случае говорят, что пин, к которому подсоединенено устройство, работает на выход и это обозначается словом `OUTPUT`.

Однако общение может происходить и в обратную сторону. То есть прибор (например, плата Iskra Neo) считывает напряжение. В таком случае говорят, что пин работает на вход и это обозначается словом `INPUT`.

⁵ От английского pin: иголка, ножка микросхемы.

Давайте ещё раз взглянем на команду `pinMode(13, OUTPUT)`:

- `pinMode` (от `pin` - «пин, контакт» и `Mode` - «режим») — это название процедуры, которая устанавливает определённый пин в соответствующий режим.

- `13` — номер пина, с которым мы будем работать.⁶

- `OUTPUT` — это режим «Выход» в который мы устанавливаем пин. То есть в нашем случае 13-й пин будет работать на выход. Это значит, что мы будем передавать напряжение от контроллера на светодиод.

Теперь посмотрим, что происходит дальше. После выполнения процедуры `setup`, микроконтроллер приступает к выполнению процедуры `loop`. И, как мы уже говорили, эта процедура выполняется постоянно, то есть после завершения запускается снова.

Процедура `loop` состоит из четырёх команд, которые выполняются последовательно:

```
digitalWrite(13, HIGH);
delay(1000);
digitalWrite(13, LOW);
delay(1000);
```

Процедура `digitalWrite` тоже является встроенной на Arduino. Она подаёт напряжение на заданный пин. `13` — номер пина, на который мы подаём напряжение, а `HIGH` (англ. «высокий, высокое») — величина напряжения, которое мы подаём на пин. `HIGH` — встроенная константа.

В данном случае она соответствует пяти вольтам. То есть мы подаём напряжение 5 вольт на 13-й пин. Соответственно, при выполнении команды `digitalWrite(13, HIGH)`, светодиод должен загореться. `LOW` (англ. «низкий, низкое») же в данном случае соответствует нулю вольт. То есть, при выполнении команды `digitalWrite(13, LOW)`, светодиод должен выключиться.

Ещё одна встроенная процедура — `delay`. Это процедура задержки, ожидания. После выполнения этой процедуры, программа на некоторое время «засыпает» т.е. перестаёт выполнять команды. В качестве аргумента процедура `delay` принимает значение времени задержки в миллисекундах. В нашем случае `delay(1000)` означает, что перед тем, как выполнять следующую команду, Iskra Neo должна подождать 1000 миллисекунд, что равно 1 секунде.

Итак, давайте ещё раз пробежимся по нашей программе, в ней:

1. Запускается процедура `setup`. Внутри этой процедуры выполняется одна команда:

- (а) выполняется процедура `pinMode(13, OUTPUT)`, которая устанавливает 13-й пин в режим работы на выход.

⁶ Несмотря на то, что в отверстие 13-го пина у нас не идёт ни одного устройства, к нему подсоединен светодиод, находящийся на плате

2. Запускается процедура `loop`. Внутри этой процедуры выполняются четыре команды:
 - (а) выполняется процедура `digitalWrite(13, HIGH)`, которая подаёт пять вольт на светодиод
 - (б) выполняется процедура `delay(1000)`, которая останавливает дальнейшее выполнение программы на одну секунду
 - (с) через секунду выполняется процедура `digitalWrite(13, LOW)`, которая подаёт на светодиод ноль вольт, то есть отключает его
 - (д) снова выполняется процедура `delay(1000)`, которая останавливает дальнейшее выполнение программы на одну секунду
3. Через секунду процедура `loop` снова выполняется сначала (пункт 2). И так до тех пор, пока на `Iskra Neo` подаётся питание

Так мы получаем мигающий светодиод. Попробуйте изменить программу так, чтобы мигание светодиода проходило в два раза чаще или в два раза реже.

2.3 Как сообщить о бедствии при помощи светодиода: переменные в программе

Итак, мы научились включать и выключать светодиод. Давайте теперь немного усложним нашу программу. Пускай теперь наш светодиод передаёт слова на азбуке Морзе.

Азбука Морзе (или «Морзянка») широко использовалась в радиосвязи в двадцатом веке и используется в любительской радиосвязи и по сей день. Информация в азбуке Морзе кодируется следующим образом: каждой букве соответствует комбинация из длительных сигналов, их называют «тире», и коротких сигналов, их называют точками.

Сигнал может подаваться в виде звука или света. За единицу длительности принимается длительность точки. Длительность тире равняется длительности трех точек. Пауза между знаками в букве — одна точка, а между буквами в слове — 3 точки. Пауза между словами — 7 точек. Таким образом, мы можем закодировать любое слово или предложение.

International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.

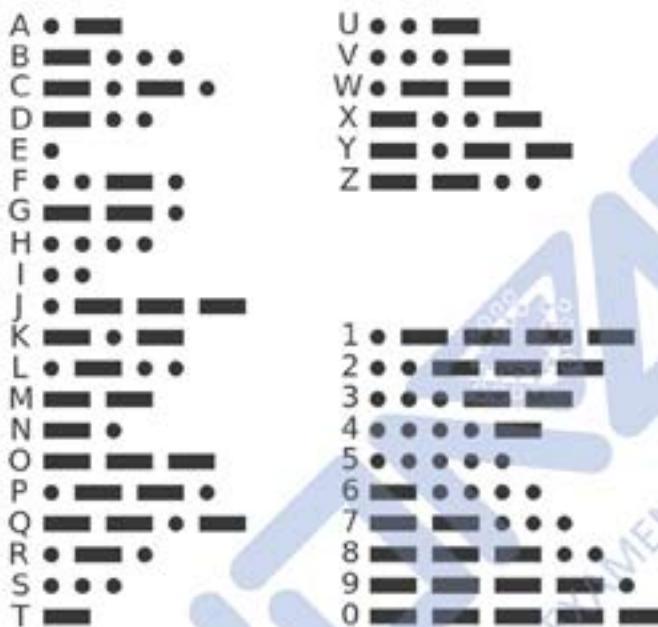


Рис. 2.1: Коды Азбуки Морзе для латинских букв (иллюстрация с сайта Wikipedia)

Во всём мире для обозначения бедствий на азбуке Морзе используют так называемый сигнал SOS.⁷ Его мы и воспроизведём на нашей Iskra Neo при помощи светоизлучателя.

Для этого загрузите на Iskra Neo следующий код:⁸

```
int ledPin = 13; // Номер пина, на котором
                  // расположен светодиод
int dotDelay = 200; // Время длительности
                  // символа «точка»

void setup()
{
  pinMode(ledPin, OUTPUT);
}

void loop()
```

⁷ SOS — это международный сигнал о помощи, состоящий из трёх последовательных букв S, O, S (три точки, три тире, три точки). ⁸ Вы также можете открыть этот код через меню: File → Examples → Amperka → p02_morse

```
{  
// == Буква S ==  
// Точка  
digitalWrite(ledPin, HIGH);  
delay(dotDelay);  
digitalWrite(ledPin, LOW);  
delay(dotDelay);  
// Точка  
digitalWrite(ledPin, HIGH);  
delay(dotDelay);  
digitalWrite(ledPin, LOW);  
delay(dotDelay);  
// Точка  
digitalWrite(ledPin, HIGH);  
delay(dotDelay);  
digitalWrite(ledPin, LOW);  
// Окончание буквы  
delay(3 * dotDelay);  
// == Буква O ==  
// Тире  
digitalWrite(ledPin, HIGH);  
delay(3 * dotDelay);  
digitalWrite(ledPin, LOW);  
delay(dotDelay);  
// Тире  
digitalWrite(ledPin, HIGH);  
delay(3 * dotDelay);  
digitalWrite(ledPin, LOW);  
delay(dotDelay);  
// Тире  
digitalWrite(ledPin, HIGH);  
delay(3 * dotDelay);
```

```
digitalWrite(ledPin, LOW);
// Окончание буквы
delay(3 * dotDelay);
// == Буква S ==
// Точка digitalWrite(ledPin, HIGH);
delay(dotDelay); digitalWrite(ledPin, LOW);
delay(dotDelay);
// Точка
digitalWrite(ledPin, HIGH);
delay(dotDelay);
digitalWrite(ledPin, LOW);
delay(dotDelay);
// Точка
digitalWrite(ledPin, HIGH);
delay(dotDelay);
digitalWrite(ledPin, LOW);
// Окончание слова
delay(7 * dotDelay);
}
```

Как это работает

А теперь давайте разберём, что в нашей программе происходит. В начале программы мы видим такие строчки:

```
int ledPin = 13; // Номер пина, на котором
                  // расположен светодиод
int dotDelay = 200; // Время длительности
                   // символа «точка»
```

Это так называемые переменные. Переменная — это некоторое имя, под которым скрывается какое-либо число или строка. Так, например, в русском языке мы можем сказать, что наш светодиод прикреплён к pinу с номером 13, а затем, вместо того чтобы говорить «pin номер 13», мы можем говорить «pin, к которому прикреплён светодиод». В программировании точно так же: вместо того, чтобы писать везде одно

и то же число, мы можем назначить ему некоторое имя и использовать его вместо самого числа. Таким образом, если мы захотим переподключить светодиод на другой пин, нам будет достаточно изменить программу всего в одном месте, а не выискивать все места, где мы обращались к 13-му pinу напрямую.

Общий вид инициализации или, как ещё говорят «объявления», переменной выглядит так:

```
типПеременной имяПеременной = значениеПеременной;
```

Как мы знаем, числа бывают разные (целые, дробные, действительные). В соответствии с этим существует множество различных типов, но с ними мы познакомимся позже. Слово `int` обозначает целые числа. Имя переменной может быть любым, если оно состоит из символов латиницы и цифр и не начинается с цифры, его мы придумываем и задаём сами. Значение переменной — это то, что мы будем подразумевать под именем переменной. В данном случае *везде*, где мы будем использовать переменную `ledPin`, мы будем подразумевать число 13.

Таким образом, команда `int ledPin = 13;` означает, что мы объявляем новую переменную с названием `ledPin`, которая обозначает целое число тринадцать.

С командами `digitalWrite` и `delay` мы уже знакомы. В данной программе мы периодически включаем и выключаем светодиод на определённое время.

Стоит лишь добавить, что:

- `delay(dotDelay)` означает, что мы ждём 200 мс, т.к. `dotDelay = 200`
- `delay(3 * dotDelay)` означает, что мы ждём 600 мс
- `delay(7 * dotDelay)` — ожидание 1400 мс или 1,4 с.

Символ «звёздочка» в программировании означает умножение. Мы можем свободно выполнять арифметические действия прямо на лету с использованием наших переменных и констант.

Поздравляем! Вы освоили основы программирования и сделали своё первое, пусть примитивное, но осмысленное, работающее устройство.



§3. ЭЛЕКТРОННЫЕ КОМПОНЕНТЫ

3.1 Что такое электричество:

напряжение и ток

3.2 Как укротить электричество:

резистор, диод, светодиод

3.3 Как быстро строить схемы:

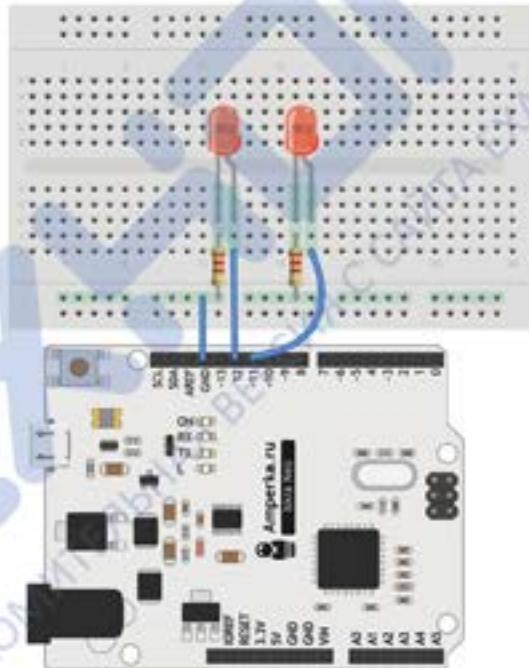
макетная доска и мультиметр

3.4 Железнодорожный светофор



ЭЛЕКТРОННЫЕ КОМПОНЕНТЫ

§3



§3. ЭЛЕКТРОННЫЕ КОМПОНЕНТЫ

3.1 Что такое электричество: напряжение и ток

В этом параграфе мы с вами поговорим о том, что такое ток и напряжение, почему к светодиоду нужен резистор, для чего нужны *макетные платы*, и соберём наше первое устройство с внешними компонентами.

Что такое заряд, и откуда он появляется, точно не знает даже Стивен Хокинг. Но обычно за заряд частицы принимается её способность участвовать в электромагнитном взаимодействии. Хотя, собственно, заряд частицы и определяется этим самым взаимодействием. Заряд измеряется в кулонах. 1 кулон — это заряд тел, которые на расстоянии в 1 метр притягиваются друг к другу с силой в 1 ньютон.

Уйдя от аксиом физики и понимания первооснов материи, вздохнём свободнее. Ток — это движение заряженных частиц. Легче всего это представить на примере водопроводной трубы. Провод — это своеобразная труба, по которой бежит «вода», состоящая из электронов. Силой тока в таком случае можно назвать объем воды, протекший через трубу за секунду. Сила тока измеряется в амперах. Если через провод за секунду протек один кулон — значит, сила тока в нём — 1 ампер.

Как силу тока можно увеличить? Увеличим трубу в диаметре и почистим стенки от ржавчины и накипи. Теперь, при той же скорости потока, через трубу проходит намного больше воды. Точно так же, как и труба, провод сопротивляется текущему через него потоку электронов. И так же, как трубу, для снижения сопротивления, провод надо увеличивать в диаметре и «чистить стенки» — выбирать материал с наименьшим удельным сопротивлением.⁹

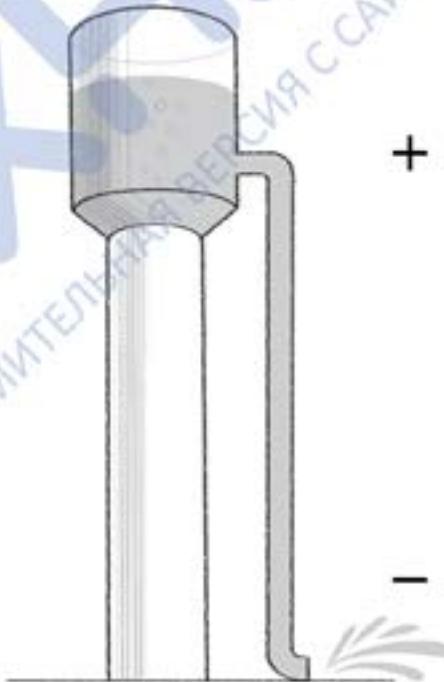


Рис. 3.1: Течение тока по аналогии с водонапорной башней

⁹ Лучшие в этом отношении — золото и серебро, но все провода из золота не сделаешь — поэтому большинство проводов делаются из меди и алюминия

Кроме того, можно увеличить напор воды. Поставить насос, к примеру. Напор электронов в электричестве называется напряжением, а насос — источником питания. Розетка в стене — источник питания. Батарейка — тоже источник питания. У источника питания всегда есть два полюса, которые называют «плюсом» и «минусом».

Принято, что электрический ток течёт от плюса к минусу. Представляйте плюс, как бочку в водонапорной башне, а минус — как кран на уровне земли.

На самом деле заряд переносится отрицательно заряженными электронами, поэтому формально ток бежит от минуса к плюсу. Дело в том, что электричество открыли раньше, чем разобрались, что именно переносит заряд, поэтому ошибочно приняли за направление движение тока путь от плюса к минусу. Эта историческая путаница продолжается уже несколько веков. Но вам важно запомнить одно: в разговорах, публикациях, книгах о практических электрических схемах считается, что ток бежит от плюса к минусу и совершенно неважно, что на физическом уровне всё происходит с точностью дооборот. Просто забудьте об электронах.

Напряжение измеряется в вольтах. Источник питания даёт 1 вольт напряжения, если при перемещении 1 кулона заряда между его полюсами совершается работа в 1 джоуль.

Ванну или бассейн в электронике заменяет конденсатор — устройство для накопления электрического заряда. Пока конденсатор не заполнился, через него течёт ток. Через заполненный конденсатор постоянный ток не течёт. Если же у нас поток электронов постоянно изменяет своё направление, такой ток называется переменным, и он проходит через конденсатор, как будто в ванну постоянно вливают и выливают воду. Наполните ванну и начните открывать и закрывать кран. Несмотря на то, что входящий поток воды прерывистый, вытекает вода равномерно. Точно так же конденсатор сглаживает пульсации напряжения.

Ёмкость конденсатора измеряется в фарадах. Но один фарад — ёмкость огромная, поэтому на практике ёмкость измеряют в пикофарадах (одна триллионная часть фарада), нанофарадах (одна миллиардная) и микрофарадах (одна миллионная).

Иногда нужно выдать строго определённый поток электронов, для этого применяются резисторы — сопротивления, сконцентрированные в одной детали (рис. 3.2)



(a) Внешний вид
резистора



(b) Обозначение на схемах
в Европе и России



(c) Обозначение на схемах
в Америке и Японии

Рис. 3.2: Резистор

Представьте себе их как тоненькую трубку. При заанее известном напоре воды через трубку за секунду пройдёт ровно столько, сколько позволяет ее диаметр. Чем больше сопротивление, тем меньше диаметр этой воображаемой трубы. Сопротивление измеряется в омах. Один ом — сопротивление такого резистора, в котором при подаче напряжения в один вольт течёт ток в один ампер. Как тонкая пластиковая труба разорвётся при подключении мощного насоса, так и тонкие провода при большой силе тока расплавятся, а при высоком напряжении многие элементы с треском, дыром и другими спецефектами сгорят. Будьте осторожны в первую очередь с сетевым напряжением!

Еще один важный элемент схемы — диод. Диод — это клапан, пропускающий ток только в одном направлении, от анода к катоду. Обычно катод диода помечен краской или точкой.

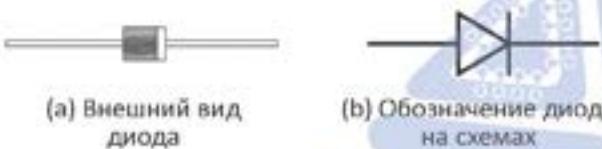


Рис. 3.3: Диод

3.2. Как укротить электричество: резистор, диод, светодиод

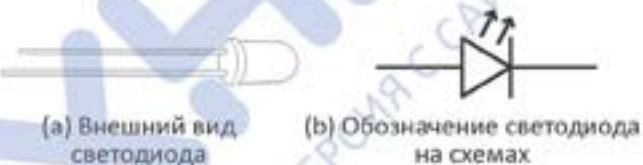


Рис. 3.4: Светодиод

Запомнить значение слов «анод» и «катод» достаточно просто. В слове «анод» столько же букв, сколько в слове «плюс», а в слове «катод» столько же букв, сколько в слове «минус». Ток течёт от плюса к минусу, от анода к катоду.

Также существуют *светодиоды*. Это такие диоды, которые светятся, когда через них проходит ток. Именно такой светодиод мы использовали в прошлом параграфе.

Светодиод — это токовый прибор. То есть такой прибор, чья работа определяется не напряжением на его выводах, а потоком электричества через него, током. Чем больше ток, тем ярче горит светодиод. Но слишком большой ток вызывает перегрев светодиода, и он перестаёт работать. Если же ток чересчур велик, то может даже треснуть корпус. Впрочем, у светодиода есть и пробойное напряжение, разрушающее его. Не превышайте его.

Поэтому светодиоды необходимо подключать вместе с токоограничивающим (или, ещё говорят, балластным) резистором. Дело в том, что светодиод практически не имеет сопротивления, поэтому даже небольшое напряжение создает большой ток. Из-за чего к светодиоду обязательно подключают дополнительное сопротивление, которое и определяет возможный максимум тока при заданном напряжении. Для большинства светодиодов достаточно резистора от 150 до 300 Ом, если он питается от стандартного напряжения в 5 вольт. И не забывайте о том, что светодиод — это диод, т.е. он пропускает ток только в одном направлении. Обычно анод, т.е. положительная ножка светодиода (та, к которой надо подключить «плюс» источника питания), длиннее.

Помимо простых пассивных электронных компонентов, тех, чье поведение полностью определено внешними условиями, существуют активные, то есть управляемые электрически компоненты. Думайте о первых, как о трубах, ваннах и турбинах, а о вторых — как о клапанах, кранах, переключателях. Из активных и пассивных электронных компонентов собираются любые устройства — от стабилизаторов напряжения до Hi-Fi аудиоусилителей и вычислительных машин.

Если же типовое устройство собрано в одном небольшом корпусе и залито пластиком или керамикой — это микросхема. Перечислять их можно бесконечно долго, их огромное множество. К примеру, на одной Iskra Neo стоит микросхема-стабилизатор напряжения, сглаживающая пульсации и из напряжения от 7 до 15 вольт, делающая стабильные 5 вольт, и, собственно, сам микроконтроллер.

3.3 Как быстро строить схемы: макетная доска и мультиметр

Для того, чтобы удобно и быстро собирать схемы без использования паяльника, лучше всего использовать доску для прототипирования. Её также называют *макетной доской*, *беспаечной макетной платой* или *bredboardом*.¹⁰

Устроена она следующим образом: справа и слева идут ряды, обозначенные как «+» и «-». Обычно к ряду с плюсом подключается напряжение питания 5 вольт, а к минусу — общий провод, он же — земля схемы. Пользуясь аналогиями из предыдущей главы, «земля» — это то море, в которое неизбежно впадают все реки, нулевой уровень электрического потенциала.

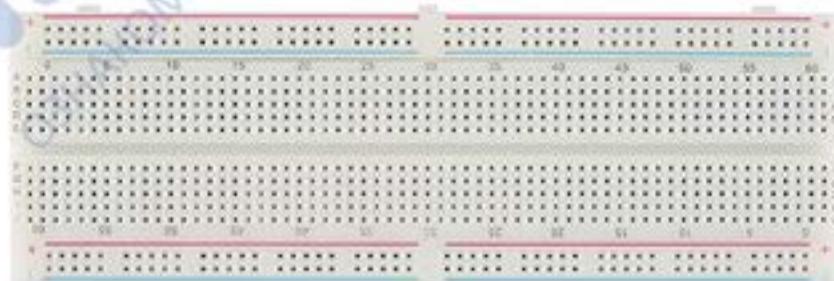


Рис. 3.5: Типичная макетная доска на 830 точек с рельсами для питания.

¹⁰ От английского breadboard, кухонная разделочная доска

Чаще всего уровень земли совпадает с уровнем минуса источника питания. Когда вы работаете с Iskra Neo, «минус» и «земля» означают одно и то же. Не стоит путать этот термин с названием нашей планеты или почвой под ногами: это совершенно разные понятия, просто обозначаемые одним и тем же словом. Земля обозначается аббревиатурой GND.¹¹

В центре платы — ряды пружинных контактов, легко зажимающие выводы деталей и провода. На нашей плате ряды контактов горизонтальны. В каждом ряду по пять контактов справа и слева соединены между собой. То есть, если вы подсоедините два компонента в один и тот же ряд, они будут соединены между собой.

Все контакты пронумерованы. Это позволяет нам обозначать контакты, как клетки в шахматах: буквами и цифрами. К примеру, A1 — левый верхний контакт на макетной плате.

Теперь о том, как искать ошибки в схеме при помощи мультиметра — устройства, сочетающего в себе вольтметр, амперметр и омметр. В электротехнике есть два вида типичных ошибок: есть контакт там, где он не нужен, и нет контакта там, где он нужен. Поэтому для начала пройдитесь по схеме, выставив на мультиметре режим «пищалки». «Пищалка» — режим мультиметра, при котором он издаёт звук при наличии контакта.¹² Процесс поиска замыканий в этом режиме называют прозвонкой. Для перехода в этот режим поверните ручку мультиметра в положение, обозначенное знаком. •))

Для того, чтобы опробовать пищалку на практике, возьмите макетную доску и попробуйте «позвонить» между контактами в различных отверстиях. Проверьте, есть ли контакт между отверстиями, расположенными на одной вертикали, горизонтали, в одной рельсе и т.д. Так вы научитесь пользоваться одним из режимов мультиметра и заодно поймёте устройство макетной доски.

Что ещё вы можете сделать с мультиметром, так это измерить напряжение между двумя точками. Для этого поверните ручку в сектор, отмеченный как DCV¹³ на отметку, значение которой заведомо превосходит напряжение, которое вы хотите измерить. Так, например, Iskra Neo выдаёт 5 вольт, поэтому логичнее всего установить ручку на отметку 20, что означает, что мы будем измерять напряжение в диапазоне от 0 до 20 вольт.

Подключите Iskra Neo к USB, а щупами мультиметра коснитесь контактов 5V и GND. Вы увидите значение, равное примерно 5 вольтам: это означает, что между этими контактами поддерживается указанное напряжение. Измерьте напряжение между контактом 3.3V и GND — получите 3,3 вольта. Измерьте напряжение между полюсами любой батарейки, вы будете удивлены: напряжение будет заметно ниже того, что написано на упаковке. Батарейки по мере разрядки теряют напряжение.

¹¹ От английского ground, земля. ¹² На самом деле — при низком сопротивлении, оно зависит от модели мультиметра

¹³ От английского Direct Current Voltage, напряжение постоянного тока.

3.4 Железнодорожный светофор

А теперь давайте соберём «железнодорожный светофор» — два светодиода, включающиеся попеременно.

Для этого нам потребуются:

1. Iskra Neo
2. макетная плата
3. два светодиода
4. два резистора на 220 Ом
5. соединительные провода и перемычки

Соберите схему так, как показано на рисунке 3.6.

Если вы с помощью мультиметра разобрались в устройстве макетной доски, вам должно быть понятно, что мы сделали. Мы подключили землю Iskra Neo к одной из горизонтальных рельс макетной доски. Для этого мы соединили проводом GND от Iskra Neo и одно из отверстий в этой рельсе. После чего на любом другом контакте этой рельсы также будет общая земля нашей схемы: ведь все контакты одной рельсы соединены проводником.

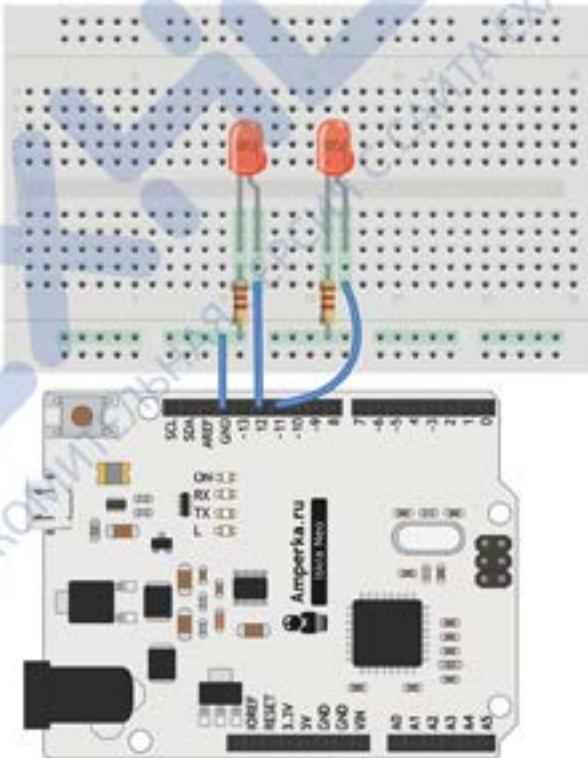


Рис. 3.6: Схема подключения «железнодорожного светофора» на макетной доске

Далее мы поставили светодиоды. Их катоды мы соединили с нашей общей землей через токоограничивающие резисторы. Резисторы с сопротивлением 180–240 Ом вполне подойдут. И, наконец, мы подключили к анодам светодиодов пины Iskra Neo номер 11 и 12. Можно выбрать любые другие пины, но для примера программы мы будем использовать именно эти.

Можно было собрать ту же самую схему ещё множеством способов: левее, правее, на верхней половине, без использования общей рельсы, с использованием обеих рельс и т.д. Главное, чтобы соблюдались основные принципы: контакты компонентов, которые должны быть соединены между собой, должны находиться в соединённой группе контактов на макетной доске, и наоборот: то, что не должно быть соединено электрически, не должно находиться в одной группе контактов на доске.

Не всегда схемы собираются на макетной доске, и к тому же иллюстрации сложных схем в таком виде могут оказаться крайне запутанными. Поэтому в схемотехнике чаще всего пользуются так называемыми принципиальными схемами. В них принята единая система

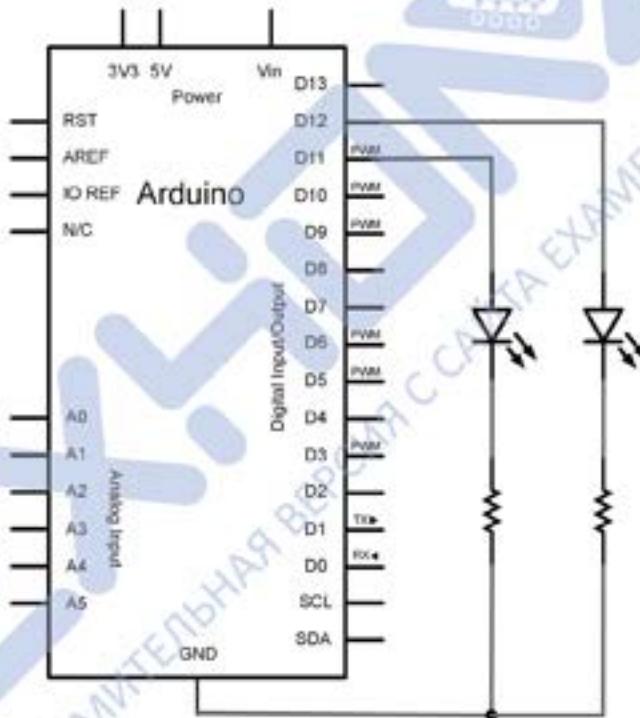


Рис. 3.7: Принципиальная схема «железнодорожного светофора» обозначений, пиктограмм, соединений и т.д. Принципиальная схема нашего устройства выглядит, как показано на рисунке 3.7.

Обратите внимание на то, как обозначены резисторы, светодиоды, микроконтроллер Iskra Neo. Достаточно просто и наглядно, не так ли?

Теперь заставим подключенные светодиоды мигать, как светофор. Для этого в программе мы попеременно будем подавать 5 В на 11-й и 12-й пины.

Код для этого приведён ниже:¹⁴

```
int pin1 = 11;  
int pin2 = 12;  
  
void setup()  
{  
    pinMode(pin1, OUTPUT);  
    pinMode(pin2, OUTPUT);  
}  
  
void loop()  
{  
    digitalWrite(pin2, LOW);  
    digitalWrite(pin1, HIGH);  
    delay(500);  
    digitalWrite(pin1, LOW);  
    digitalWrite(pin2, HIGH);  
    delay(500);  
}
```

Сможете разобраться со скетчем и объяснить, что происходит? Со всеми командами мы уже знакомы по второму параграфу.

Подключите Iskra Neo и прошейте на него этот скетч. Светодиоды мигают, как светофор?

Поздравляем — всё получилось. Вы собрали первое устройство с внешними компонентами!



¹⁴ File → Examples → Amperka → p03_railroad



§4. ВЕТВЛЕНИЕ ПРОГРАММЫ

- 4.1 Условный оператор if
- 4.2 Оператор многозначного выбора switch
- 4.3 Операторы сравнения и логические значения
- 4.4 Цикл с условием while и цикл с параметром for
- 4.5 Как написать свою собственную функцию
- 4.6 Как упростить код: SOS при помощи процедур

ВЕТВЛЕНИЕ ПРОГРАММЫ

§4



```

Arduino
File Edit Sketch Tools Help
sketch_Feb06
int led = 13;

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}

Done compiling
Binary sketch size: 1084 bytes (of a 32,256 byte maximum)
Arduino Uno on COM4
    
```



§4. ВЕТВЛЕНИЕ ПРОГРАММЫ

4.1 Условный оператор if

В жизни мы часто используем условную конструкцию «если ..., то ...». Например, мама может сказать своему сыну: «если ты будешь вести себя хорошо, то вечером мы пойдём в зоопарк». На языке Arduino эта фраза будет выглядеть так:

```
if (хорошееПоведение) {
    походВЗоопарк;
}
```

То есть, если условие, находящееся в круглых скобках после слова `if` выполняется, мы исполняем команды, которые находятся внутри фигурных скобок.

Описанную выше фразу можно расширить: «если ты будешь вести себя хорошо, то вечером мы пойдём в зоопарк, иначе весь вечер будешь стоять в углу». Вот так она будет выглядеть на языке Arduino:

```
if (хорошееПоведение) {
    походВЗоопарк;
} else {
    стояниеВУглу; }
```

В общем виде if-конструкция выглядит так:

```
if (Условие) {
    Команда1;
    Команда2; ...
} else {
    Команда3;
    Команда4;
...
}
```

Условная конструкция — это одна из основных конструкций в Arduino. Она позволяет нам отслеживать выполнение какого-либо условия, а в ответ мы можем выполнять какое-нибудь действие: например, зажигать светодиод, если кнопка нажата, и выключать его в противном случае. Более подробно с конструкцией `if` мы познакомимся, когда будем изучать работу с кнопкой.

4.2 Оператор многозначного выбора switch

Конструкция `if`, как мы видим, предполагает всего два возможных варианта: либо условие выполняется, либо нет. Однако бывают случаи, когда выбор несколько больше. Например, родители могут сказать школьнику: «если ты получишь два по математике, то будешь стоять в углу; если получишь три, останешься дома; в любом другом случае сможешь пойти гулять». В таком случае обычно используется конструкция `switch`:

```
switch (оценкаПоМатематике) {
    case 2:
        стояниееВУглу;
        break;
    case 3:
        сидениеДома;
        break;
    default: прогулкаПоУлице;
}
```

Как мы видим, после слова `switch` ставится переменная, значение которой мы оцениваем. Далее, после слов `case`, мы перебираем значения, которые она может принимать, и для каждого значения описываем команды, которые должны исполняться в случае, если значение переменной равно указанному значению. Каждый `case` должен заканчиваться командой `break`. Очень распространённая ошибка — по невнимательности забыть написать `break`. В этом случае продолжится исполнение кода, следующего после очередного `case`, что в большинстве случаев не предполагается и приводит к неожиданным результатам.

Метка `default` обозначает все случаи, которые не учтены ни в одном `case`. Выражение `default` можно и не использовать, если в случаях, когда ни один `case` не подошёл, ничего делать не нужно. В общем виде конструкция `switch` выглядит так:

```
switch(Переменная) {
    case Значение1:
        Команда1;
        Команда2;
        ...
        break;
    case Значение2:
        Команда3;
        Команда4;
```

```
...
break;
...
default:
    Команда5;
    Командаб;
...
}
```

Наш пример мы можем записать и через if-конструкцию:

```
if (оценкаПоМатематике == 2) {
    стояниеВУглу;
} else {
    if (оценкаПоМатематике == 3) {
        сидениеДома;
    } else {
        прогулкаПоУлице;
    }
}
```

4.3 Операторы сравнения и логические значения

Оператор	Синтаксис
Равно	$a == b$
Не равно	$a != b$
Больше	$a > b$
Меньше	$a < b$
Больше или равно	$a >= b$
Меньше или равно	$a <= b$

Таблица 4.1: Операторы сравнения

Знак «==» означает «равно». В программировании различают знак присваивания «=» от знака сравнения «==». Первый присваивает переменной некоторой зна-

чение. Например, `z = 2` будет означать, что под переменной `z` будет подразумеваться число 2. Второй сравнивает левую и правую части и возвращает истинное или ложное значение, в зависимости от того, равны они друг другу или нет.

Например, `z == 2` вернёт истину, в случае если `x` равняется двум и ложь в противоположном случае.

Ложь или истина являются результатом операторов сравнения (Таб. 4.1).

Истина и ложь в программировании — это не строки, не слова и не числа.

Это просто значения, которые могут принимать переменные специального типа: `boolean`. Подобно тому, как `int` означает целое число, `boolean` означает логическое значение: `true` — истина, `false` — ложь; третьего не дано. Когда вы пишете:

```
if (z == 2) ...
```

вы на самом деле неявно создаёте логическую переменную, которая не имеет имени и используется один раз. В явном виде вы могли бы написать:

```
boolean check = (z == 2);
if (check) ...
```

Если при выполнении условия необходимо выполнить всего одну команду, фигурные скобки можно не писать. Поэтому наш пример может быть написан в более компактной, наглядной форме:

```
if (оценкаПоМатематике == 2)
    стояниеВУглу;
else if (оценкаПоМатематике == 3)
    сидениеДома;
else прогулкаПоУлице;
```

4.4 Цикл с условием `while` и цикл с параметром `for`

А теперь давайте познакомимся с так называемыми циклами.

Некоторые процессы требуют продолжительных и повторяющихся действий. Например, путь от дома до школы — это многократное повторение одного и того же действия — шага. Т.е. мы делаем один шаг, потом ещё один, потом ещё и ещё. И так, пока не достигнем школы. Многократно повторяющийся процесс называется циклом. С одним важным циклом в языке Arduino мы уже знакомы — это процедура `loop`. Она беспрестанно повторяется, пока плата `Iskra Neo` подключена к сети.

Но в Arduino есть ещё и циклические конструкции. Таковыми являются конструкции `while` и `for`. Если бы мы описали наш путь от дома до школы на языке Arduino, он бы выглядел примерно так:

```
координаты = дом;  
while (координаты != школа) {  
    шаг;  
}
```

Знак `!=` означает «не равно».

Эта программа описывает алгоритм, при котором нужно шагать, пока мы не окажемся в школе. Пока истинно то, что находится в круглых скобках после слова `while`, нужно постоянно выполнять то, что находится между фигурными скобками.

В общем виде конструкция `while` выглядит так:

```
while (Условие) {  
    Команда1;  
    Команда2;  
    ...  
}
```

Пока выполняется условие, будут постоянно выполняться команды, одна за другой.

Ещё одной разновидностью цикла является конструкция `for`. В общем виде она выглядит так:

```
for (Инициализация; Условие; Инкремент) {  
    Команда1;  
    Команда2;  
    ...  
}
```

Инициализация — это команда, которая выполняется один раз в начале цикла. На каждом шаге цикла проверяется условие. Если условие выполняется («истина»), то выполняются команды в фигурных скобках и инкремент.

Инкремент — это тоже команда. Далее снова проверяется условие и, в случае его выполнения, запускаются команды в фигурных скобках и инкремент. И так до тех пор, пока выполняется условие. В противном случае, если условие не выполняется («ложь»), цикл завершается.

Наш пример с дорогой до школы можно описать при помощи конструкции `for` следующим образом:

```
for (координаты = дом; координаты != школа; шаг) {  
}
```

Самым распространённым применением цикла `for` является перебор всех целых чисел в нужном диапазоне. Например, если мы подключим к `Iskra Neo` несколько светодиодов к пинам с 4-го по 8й, мы можем включить все сразу, применив цикл по номеру пина, который на самом деле является просто целым числом:

```
for (int pin = 4; pin <= 8; ++pin) {  
    digitalWrite(pin, HIGH);  
}
```

В этом примере инициализация — это объявление переменной `pin`, которую мы будем использовать как номер пина со светодиодом. Сразу же её значение выставляется в 4: это значение оно будет содержать при первом прогоне цикла.

Цикл сработает один раз — зажжётся светодиод на пине 4. Затем будет выполнен инкремент. В данном случае — это `++pin`. Это выражение просто означает, что значение переменной `pin` необходимо увеличить на единицу. `++pin` — это более компактная запись выражения `pin = pin + 1`.

После инкремента, `pin` примет значение 5, и будет проверено условие. Условие истинно, т.к. $5 < 8$, поэтому цикл будет продолжен, но на этот раз `pin` будет равен не 4, а 5, и поэтому в результате включится светодиод на 5-м пине.

Цикл продолжится, включая светодиоды один за одним, пока `pin` не примет значение 9. При этом условие не выполнится, а потому выполнение самого цикла закончится, а программы — продолжится со следующей за циклом строки.

Вы могли подумать, что включать светодиоды один за одним — не всегда то, что нужно. Но на практике перебор значений происходит настолько быстро, что заметить разницу в моменте включения светодиодов, человеку абсолютно невозможно.

4.5 Как написать свою собственную функцию

В прошлых главах мы неоднократно использовали процедуры, такие как `pinMode()` или `digitalWrite()`. Давайте попробуем понять, что это такое, и как написать свою собственную процедуру.

Процедура — это часть кода, которому назначено какое-то имя. Она используется в первую очередь для того, чтобы существенно сократить код, а также для удобства и наглядности. Так, например, вместо того, чтобы сказать «зайди в ванную, включи кран, намыль руки, смой с них мыло под водой и выключи кран», мы просто говорим «помой руки». Это типичный пример процедуры, который на языке программирования может быть описан следующим образом:

```
void помытьРуки()
{
    зайтиВВанную();
    включитьКран();
    намылитьРуки();
    смытьСРукМыло();
    выключитьКран();
}
```

В данном случае наименование «помытьРуки» подразумевает под собой целый ряд действий, которые нужно произвести.

Также и в языке Arduino. Общий вид процедуры выглядит следующим образом:

```
void имяПроцедуры(ТипАргумента1 имяАргумента1,
                     ТипАргумента2 имяАргумента2, ... ) {
    команда1; команда2;
    ...
}
```

Аргументы — это некоторые переменные, которые передаются в процедуру. Их может и не быть. Тогда скобка открывается и тут же закрывается.

Если процедура в результате своей работы должна возвращать нам обратно какое-то значение, она называется функцией. Общий вид функции выглядит так:

```
ТипВозвращаемогоЗначения имяФункции(
    ТипАргумента1 имяАргумента1,
    ТипАргумента2 имяАргумента2, ... ) {
    команда1; команда2;
    ...
    return возвращаемаяПеременная;
}
```

Скажем, функцию мытья тарелки можно описать следующим образом:

```
ЧистаяТарелка помытьТарелку(ГрязнаяТарелка тарелка) {
    берёмГрязную(тарелка);
    наносимМоющееСредствоНа(тарелка);
    погружаемПодВоду(тарелка);
    return получаемЧистую(тарелка);
}
```

Здесь мы в качестве аргумента получаем грязную тарелку, а возвращаем чистую.

Как мы видим, функция отличается от процедуры лишь двумя моментами:

1. вместо `void` стоит ТипВозвращаемогоЗначения. `void`¹⁵ означает, что возвращаемой переменной нет, то есть ничего не возвращается.
2. в конце появляется выражение с `return`, которое и возвращает значение переменной

Самым простым примером функции является сумма двух чисел. Её можно описать на Arduino следующим образом:

```
int sum(int a, int b)
{
    int c = a + b;
    return c;
}
```

Данная функция означает, что мы должны передать в неё два целых числа `a` и `b`. Внутри этой функции создаётся переменная (тоже с целым типом) `c`. Переменной `c` присваивается сумма `a` и `b`. После чего значение переменной `c` возвращается.

После того, как мы описали эту процедуру, мы можем вызвать её в любом месте программы:

```
int myVariable = sum(2, 3);
```

В данном случае значение переменной `myVariable` целого типа будет равняться пяти.

Зачем писать функцию для сложения двух чисел, если мы просто можем написать «`+`» непосредственно там, где это нужно?! Да, для сложения делать это абсолютно не стоит, это был всего лишь пример.

На практике процедуры и функции существенно сокращают программу. А также делают её более удобной для восприятия и использования.

4.6. Как упростить код: SOS при помощи процедур

Итак, а теперь давайте попробуем при помощи процедур упростить тот код, который мы написали в третьем параграфе для подачи сигнала SOS.

Для этого загрузите на `Iskra Neo` следующий код:¹⁶

```
int ledPin = 13; // Номер пина со светодиодом
int dotDelay = 200; // Время длительности «точки»
// Точка
```

¹⁵ `void` — пустота (англ.). ¹⁶ File → Examples → Amperka → p04_morse

```
void dot()
{
    digitalWrite(ledPin, HIGH);
    delay(dotDelay);
    digitalWrite(ledPin, LOW);
    delay(dotDelay);
}
// Тире
void dash()
{
    digitalWrite(ledPin, HIGH);
    delay(3 * dotDelay);
    digitalWrite(ledPin, LOW);
    delay(dotDelay);
}
// Конец буквы
void letterEnd()
{
    delay(2 * dotDelay);
}
// Конец слова
void wordEnd()
{
    delay(6 * dotDelay);
}
// Буква S
void letterS()
{
    dot();
    dot();
    dot();
    letterEnd();
}
// Буква О
void letterO()
```

```

{
    dash();
    dash();
    dash();
    letterEnd();
}
void setup()
{
    pinMode(ledPin, OUTPUT);
}
void loop(){
    letterS(); // Показываем букву S
    letterO(); // Показываем букву O
    letterS(); // Показываем букву S
    wordEnd(); // Делаем паузу между словами
}

```

Как мы видим, сигнал SOS мигает точно также, однако код стал более удобен для восприятия и использования.

Как это работает?

А теперь давайте разберём, что в нашей программе происходит

```

// Точка
void dot()
{
    digitalWrite(ledPin, HIGH);
    delay(dotDelay);
    digitalWrite(ledPin, LOW);
    delay(dotDelay);
}

```

Процедура `dot()` включает светодиод и ждёт промежуток времени, равный 200 миллисекундам, которые обозначает переменная `dotDelay`. Затем она выключает светодиод и снова ждёт 200 миллисекунд. Так воспроизводится сигнал «точка».

Также в нашей программе есть процедура `dash()`, которая делает то же, что и `dot()`, но только оставляет светодиод включённым в три раза дольше, т.е. 600 мс.

```
// Тире
void dash()
{
    digitalWrite(ledPin, HIGH);
    delay(3 * dotDelay);
    digitalWrite(ledPin, LOW);
    delay(dotDelay);
}
```

Так воспроизводится сигнал тире.

- Процедура `letterEnd()` ждёт 400 мс. Она означает окончание буквы.
- Процедура `wordEnd()` ждёт 1200 мс. Она означает окончание слова.
- Процедура `letterS()` воспроизводит букву S, которая, как можно увидеть из рисунка 2.1, состоит из трёх точек. В конце мы вызываем процедуру окончания буквы.
- Процедура `letterO()` воспроизводит букву O, которая состоит из трёх тире, а затем также вызывает процедуру окончания буквы.
- В процедуре `loop()` мы вызываем последовательно процедуры `letterS`, `letterO` и `letterS`, а затем вызываем процедуру `wordEnd`. Таким образом, мы воспроизводим последовательно буквы S, O и S, после чего ждём некоторое время и снова воспроизводим буквы S, O, S. В итоге наша Iskra Neo непрерывно передаёт сигнал бедствия.

Код программы получился на порядок более понятным и выразительным, не правда ли? Теперь вы даже можете поделиться написанными функциями с другим человеком, чтобы он использовал их в своём проекте, и ему не пришлось бы писать их заново. А он в свою очередь может поделиться своими функциями, которые делают что-то другое полезное. Всё это — основа коллективной разработки, без которой невозможно было бы делать сложные программы и устройства.

Процедуры и функции — мощный инструмент. Поэтому крайне важно им овладеть. Пробуйте, экспериментируйте, ошибайтесь, исправляйтесь, творите!



§5. МАССИВЫ И ПЬЕЗОЭЛЕМЕНТЫ

5.1 Что такое массив

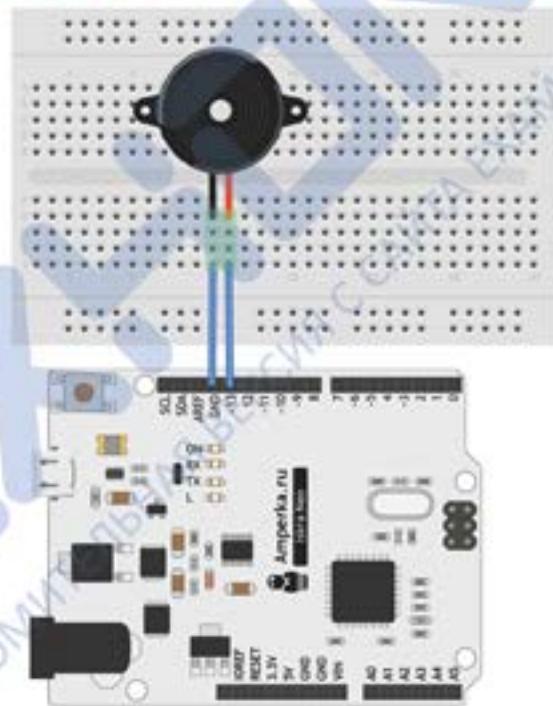
5.2 Строки: массивы символов

5.3 Воспроизведение произвольных слов на азбуке Морзе

5.4 Как пищать на Iskra Neo: пьезоэффект и звук



МАССИВЫ И ПЬЕЗОЭЛЕМЕНТЫ

§5

§5. МАССИВЫ И ПЬЕЗОЭЛЕМЕНТЫ

5.1 Что такое массив

Ещё одним важным элементом в программировании является **массив**. *Массив* — это индексированный, то есть пронумерованный, список элементов. Что это значит?

Представьте себе футбольную команду. Это набор людей, каждому из которых присвоен какой-то уникальный номер. Произнося, например, «номер шестьдесят восемь», мы имеем в виду какого-то конкретного футболиста. Точно также и на языке Arduino. Мы можем сохранять значения переменных в единый массив, а затем обращаться к ним по их номерам.

Сам массив — это такая же переменная, как и все те, с которыми мы уже встречались. Перед использованием, эту переменную-массив нужно точно так же объявить. В общем виде это происходит так:

```
типЭлементовМассива имяМассива[количествоЭлементов] =  
{Элемент1, Элемент2, ...};
```

Скажем, в случае с футболистами, это выглядело бы следующим образом:

```
Футболист команда[11] = {Иванов, Петров, Сидоров, ...};
```

В таком случае, если мы далее напишем команда[2], то мы будем иметь в виду третьего члена команды, то есть Сидорова.

Дело в том, что в программировании отсчёт начинается не с единицы, а с нуля — 0, 1, 2, 3, и т.д. Это может показаться странным на первый взгляд, но на практике, при решении реальных задач, оказывается, что так действительно удобнее. Кроме того, это удобнее и для процессора:

обращение по номерам, начинающимся с нуля, происходит быстрее.

Рассмотрим примеры определения или, как ещё говорят, инициализации массива:

```
int mySensVals[6] = {2, 4, -8, 3, 2, 1};  
int myPins[] = {2, 4, 8, 3, 6, 2};  
int myInts[6];
```

Первый случай является наиболее полным примером инициализации массива. Мы указываем, что хотим использовать массив с именем *mySensVals*, состоящий из шести элементов, а затем перечисляем, чему должны быть равны эти элементы в фигурных скобках.

Во втором случае мы инициализируем массив без указания длины. В этом случае длина массива (шесть) будет вычислена компилятором автоматически, исходя из количества элементов, записанных в фигурных скобках.

В третьем случае мы просто инициализируем массив, состоящий из шести целочисленных элементов. Обратите внимание, что мы не указываем начальные значения элементов. Такая короткая запись удобна, если мы знаем, что в последствии наполним массив данными самостоятельно. При создании же, в качестве значений шести элементов будет взят произвольный мусор: случайные числа, которые нельзя предугадать. Если далее в программе мы напишем `myPins[2]`, то будем иметь в виду третий элемент массива `myPins`, то есть 8. То, что записывается в квадратных скобках — номер элемента, в программировании называется индексом.

5.2 Строки: массивы символов

Частным случаем массива является *строка*. Страна — это массив букв. Скажем, слово «way» — это массив с типом элементов «буква». В программировании букву ещё называют словом «символ». Символ — это такой же тип данных, как `int` и всё остальное, и обозначается он в Arduino как `char`. Итак, массив-слово «way» состоит из трёх букв, первая из которых «w», вторая — «a», третья — «у».

В Arduino мы можем инициализировать строку следующим образом:

```
char theword[4] = {'w', 'a', 'y', 0x00};
```

Вы можете удивиться, откуда взялась странная четвёртая буква `0x00`. Дело в том, что этот символ используется для обозначения конца строки. Он не выводится на экран, не считается при вычислении длины строк, не участвует в других операциях. Это просто обязательный маркер, по которому Arduino может понять, что строка закончена. Даже если после него следуют другие символы.

Не стоит путать этот ноль с символом арабской цифры ноль, которая бы записывалась в одинарных кавычках, как и все другие буквы будь она в строке:

```
char thousand[5] = {'f', '0', '0', 'd', 0x00};
```

Использовать запись с побуквенным перечислением строки чаще всего неудобно, поэтому существует более естественная, короткая запись:

```
char theword[] = "way";
```

Обратите внимание, что мы не указали количество элементов букв: оно будет вычислено автоматически. Также мы использовали двойные кавычки вместо одинарных. Это имеет значение: в одинарных указывается одиничный символ, в двойных — строка. И ещё, мы явно не завершили строку терминальным нулем: при такой записи он будет добавлен компилятором автоматически, и мы получим всё тот же массив из четырёх элементов. Напомним, что строка — это обычный массив, поэтому мы можем менять отдельные символы, просто обращаясь к ним по индексам. Например, мы можем изменить слово «way» на «cat», заменив первый и третий символы:

```
char theword[] = "way";
theword[0] = 'c';
theword[2] = 't';
```

Вообще, у каждого символа есть свой код. Когда мы пишем букву в одинарных кавычках, компилятор за нас переводит эту букву в код.

!	33	0x21	A	65	0x41	a	97	0x61
"	34	0x22	B	66	0x42	b	98	0x62
#	35	0x23	C	67	0x43	c	99	0x63
\$	36	0x24	D	68	0x44	d	100	0x64
%	37	0x25	E	69	0x45	e	101	0x65
&	38	0x26	F	70	0x46	f	102	0x66
'	39	0x27	G	71	0x47	g	103	0x67
(40	0x28	H	72	0x48	h	104	0x68
)	41	0x29	I	73	0x49	i	105	0x69
*	42	0x2A	J	74	0x4A	j	106	0x6A
+	43	0x2B	K	75	0x4B	k	107	0x6B
,	44	0x2C	L	76	0x4C	l	108	0x6C
-	45	0x2D	M	77	0x4D	m	109	0x6D
.	46	0x2E	N	78	0x4E	n	110	0x6E
/	47	0x2F	O	79	0x4F	o	111	0x6F
0	48	0x30	P	80	0x50	p	112	0x70
1	49	0x31	Q	81	0x51	q	113	0x71
2	50	0x32	R	82	0x52	r	114	0x72
3	51	0x33	S	83	0x53	s	115	0x73
4	52	0x34	T	84	0x54	t	116	0x74
5	53	0x35	U	85	0x55	u	117	0x75
6	54	0x36	V	86	0x56	v	118	0x76
7	55	0x37	W	87	0x57	w	119	0x77
8	56	0x38	X	88	0x58	x	120	0x78
9	57	0x39	Y	89	0x59	y	121	0x79
:	58	0x3A	Z	90	0x5A	z	122	0x7A
;	59	0x3B	[91	0x5B	{	123	0x7B
<	60	0x3C	\	92	0x5C		124	0x7C
=	61	0x3D]	93	0x5D	}	125	0x7D
>	62	0x3E	^	94	0x5E	~	126	0x7E
?	63	0x3F	_	95	0x5F			
@	64	0x40	'	96	0x60			

Таблица 5.1: Таблица символов в кодировке ASCII:

символ / десятичный код / шестнадцатирический код

Символ	Код в десятичной системе	Код в шестнадцатиричной системе
Терминальный ноль	0	0x00
Табуляция	9	0x09
Перенос строки (LF)	10	0x0A
Возврат каретки (CR)	13	0x0D
Пробел	32	0x20

Таблица 5.2: Коды служебных символов

Это необходимо, потому что всё, с чем умеет работать процессор — это числа. Символы с их кодами представлены в таблице 5.1.

Таблица, ставящая в соответствие каждому символу свой код, называется кодировкой. Наиболее простой, старой и поддерживаемой кодировкой является ASCII.¹⁷ Она-то и представлена в таблице. Она не содержит символы из нелатинских алфавитов, то есть не содержит буквы русского языка, но тем не менее достаточна для многих задач. О том, как поддерживать кириллицу в строках, мы поговорим в одном из следующих параграфов.

Ещё вы могли заметить, что в таблице представлены символы, начиная с 33-го. Дело в том, что коды от 0 до 32 отведены под невидимые, служебные символы, но тем не менее некоторые из них не менее важны, чем привычные буквы, цифры и знаки препинания. Наиболее значимые из таких служебных символов представлены в таблице 5.2

Итак, с полученными знаниями, мы можем записать слово «way» напрямую, при помощи кодов символов. Это будет выглядеть так:

```
char theword[4];
theword[0] = 0x77;
theword[1] = 0x61;
theword[2] = 0x79;
theword[3] = 0x00;
```

5.3 Как общаться на азбуке Морзе: воспроизведение произвольных слов

А теперь давайте обобщим наши знания из предыдущей главы о конструкциях `for` и `switch` со знаниями о работе со строками. Для этого сделаем программу, которая умеет воспроизводить любое английское слово или последовательность слов на азбуке Морзе.

¹⁷ Аббревиатура от англ. American Standard Code for Information Interchange

Загрузите на skra Neo следующий код:¹⁸

```
int ledPin = 13; // Номер пина со светодиодом
int dotDelay = 200; // Время длительности «точки»
char theword[] = "hello, world";
// Точка
void dot()
{
    digitalWrite(ledPin, HIGH);
    delay(dotDelay);
    digitalWrite(ledPin, LOW);
    delay(dotDelay);
}
// Тире
void dash()
{
    digitalWrite(ledPin, HIGH);
    delay(3 * dotDelay);
    digitalWrite(ledPin, LOW);
    delay(dotDelay);
}
// Окончание буквы
void letterEnd()
{
    delay(2 * dotDelay);
}
// Окончание слова
void wordEnd()
{
    delay(6 * dotDelay);
}
// Слово
void morseWord(char theword[])
{
```

¹⁸ File → Examples → Amperka → p05_morse

```

{
    int len = strlentheword);
    for(int i = 0; i < len; ++i)
        morseLettertheword[i]);
    wordEnd();
}

// Буква void morseLetter(char c)
{
    switch(c) {
        case 'a':
            dot();dash();
            break;

        case 'b':
            dash();dot();dot();dot();
            break;

        case 'c':
            dash();dot();dash();dot();
            break;

        case 'd':
            dash();dot();dot();
            break;

        case 'e':
            dot();
            break;

        case 'f':
            dot();dot();dash();dot();
            break;

        case 'g':
            dash();dash();dot();
            break;

        case 'h':
            ...
    }
}

```

```
dot();dot();dot();
break;
case 'i':
    dot();dot();
    break;
case 'j':
    dot();dash();dash();dash();
    break;
case 'k':
    dash();dot();dash();
    break;
case 'l':
    dot();dash();dot();dot();
    break;
case 'm':
    dash();dash();
    break;
case 'n':
    dash();dot();
    break;
case 'o':
    dash();dash();dash();
    break;
case 'p':
    dot();dash();dash();dot();
    break;
case 'q':
    dash();dash();dot();dash();
    break;
case 'r':
    dot();dash();dot();
```

ОГЛАСОВАННАЯ ВЕРСИЯ С САЙТА EXAMEN-TECHNOLAB.RU

```

        break;
case 's':
    dot();dot();dot();
    break;
case 't':
    dash();
    break;
case 'u':
    dot();dot();dash();
    break;
case 'v':
    dot();dot();dot();dash();
    break;
case 'w':
    dot();dash();dash();
    break;
case 'x':
    dash();dot();dot();dash();
    break;
case 'y':
    dash();dot();dash();dash();
    break;
case 'z':
    dash();dash();dot();dot();
    break;
case '.':
    wordEnd();
    break;
}
letterEnd();
}

```

```
void setup()
{
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    morseWordtheword();
}
```

После прошивки светодиод должен сигнализировать предложение «hello world», что означает «Здравствуй, Мир».

Как это работает?

Давайте пробежимся по программе и поймём, что здесь происходит.

Вверху программы мы назначаем различным переменным их значения. В том числе массиву символов (или иначе: строке) `theword` мы присваиваем значение «hello world» (команда `char theword[] = "hello world";`). Эту фразу и будет воспроизводить `Iskra Neo`. Вы можете поменять её на любую другую фразу на английском языке.

Процедура `loop` вызывает процедуру `morseWord`, которой в качестве аргумента подаётся слово, записанное в переменной `theword`.

```
// Слово void morseWord(char theword[])
{
    int len = strlentheword;
    for(int i = 0; i < len; ++i)
        morseLettertheword[i];
    wordEnd();
}
```

Функция `morseWord`, как видно из кода выше, начинается с команды `int len = strlentheword;` которая присваивает переменной `len` значение, равное длине слова, записанного в переменной `theword`. Функция `strlen` – встроенная. Вы можете использовать её на любых строках; она подсчитывает количество символов до терминального нуля.

Далее следует цикл `for`. В этом цикле мы пробегаем по всем буквам слова. Мы инициализируем переменную-индекс `i` целого типа, присваивая ей значение 0, и прибавляем на каждом шаге единицу. Пока `i` не станет равным длине строки `theword`.

Внутри цикла, т.е. на каждом шаге цикла, мы берём *i*-ю букву из слова *theword* и подаём её в качестве аргумента в функцию *morseLetter*.

В этой процедуре — один единственный *switch*, который, в зависимости от поданной в него буквы, производит определённые действия со светодиодом. Например, если мы подаём в процедуру *morseLetter* символ 'a', то запускаются последовательно процедуры *dot* и *dash*, смысл которых мы разбирали в прошлом параграфе.

Таким образом, мы разбираем слово на буквы и каждую букву воспроизводим при помощи светодиода.

5.4 Как пищать на Iskra Neo: пьезоэффект и звук

А теперь давайте сделаем устройство, общающееся с нами звуками, а не миганием индикаторов. Для этого нам потребуется пьезоэлемент, также известный как «пищалка». Он изображён на рисунке 5.1.

Что такое пьезоэлемент? Это конденсатор, с одним интересным свойством. Он изменяет свой размер, когда на него подаётся напряжение и возвращается к первоначальному размеру, если напряжение снято. И, естественно, этот пьезоэлемент толкает воздух при своём расширении. Поскольку звук есть колебания воздуха, можно превратить пьезоэлемент в источник звука. Для этого лучше использовать «пищалки» со звукоусиливающим рупором, заранее установленным на фабрике. Естественно, как и положено конденсатору, постоянный ток пьезоэлемент не проводит, поэтому управление им несколько отличается от управления светодиодом, который работает как раз от постоянного тока.



Рис. 5.1: Пьезоэлемент

Подключите пьезоэлемент к плате так, как показано на схемах 5.2 и 5.3.

То есть один контакт пищалки должен быть соединён с 13-м пином, а другой — с землёй. Полярность не важна: ножки пьезоизлучателя равнозначны. Кроме того, у вас в руках скорее всего пьезоизлучатель, ножки которого достаточно далеко друг от друга, а потому не могут быть размещены на макетной доске в точности так, как показано. Но это не важно: главное, чтобы один контакт так или иначе был соединён с 13-м пином, а другой — с землёй.

Загрузите на Iskra Neo пример мигания из второго параграфа:¹⁹

```
void setup()
{
    pinMode(13, OUTPUT);
}

void loop()
{
    digitalWrite(13, HIGH);
    delay(1000);
    digitalWrite(13, LOW);
    delay(1000);
}
```

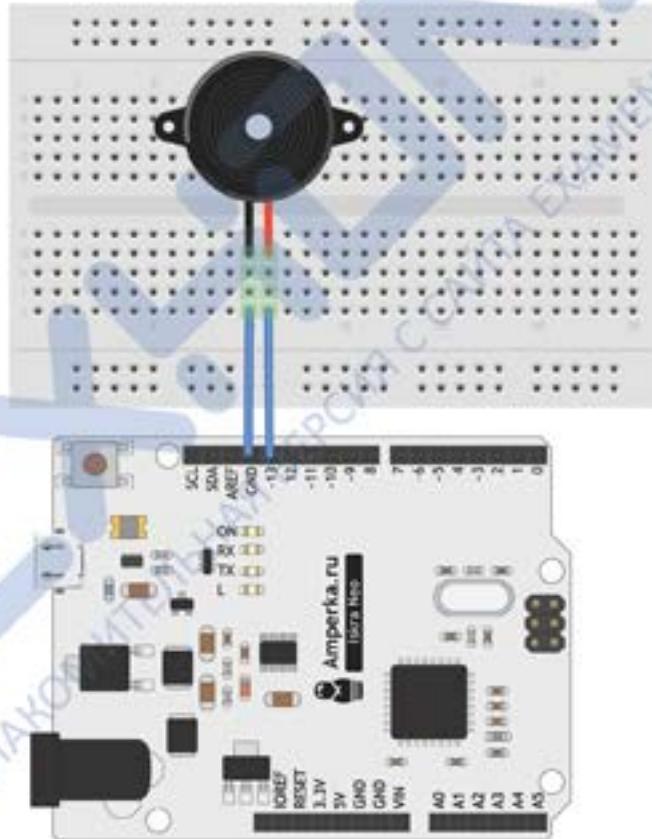


Рис. 5.2: Схема подключения пьезоэлемента на макетной доске

¹⁹ File → Examples → Amperka → p02_blink

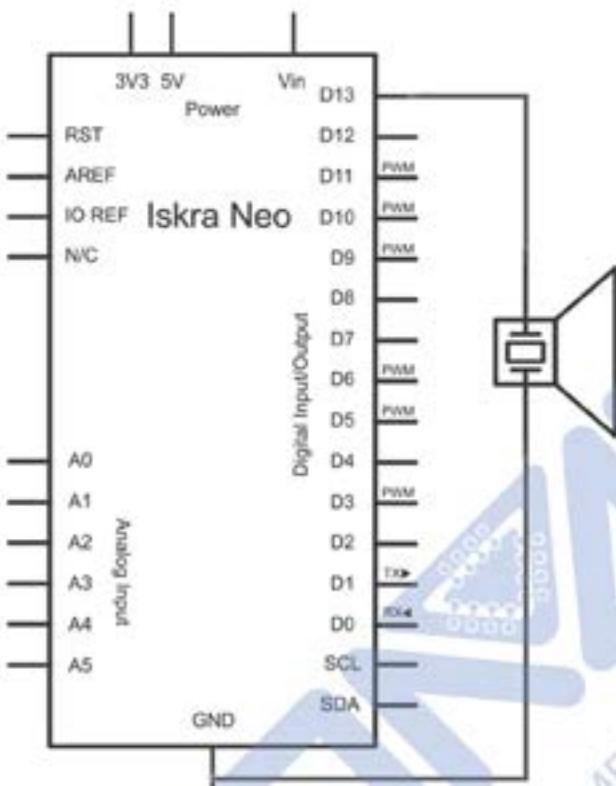


Рис. 5.3: Принципиальная схема подключения пьезоэлемента

Если вы услышите периодические тихие щелчки — поздравляем, всё правильно. Но тихие щелчки нас не устраивают. Нам нужен громкий писк.

Прежде чем его воспроизвести, расскажем немного о природе звука. Звук — это периодическое сгущение и разрежение воздуха. Высота звука — это частота этих сгущений и разрешений. Человеческое ухо слышит звуки от 20 колебаний в секунду — это низкие, басовые звуки, до 20 000 колебаний в секунду — это очень тонкий писк. Пьезоэлемент не умеет воспроизводить низкие звуки — вместо них он выдаёт щелчки.²⁰ Поэтому мы будем работать с высокими звуками. Подадим на пьезоэлемент переменное напряжение высокой частоты. К примеру — 500 герц, то есть 500 колебаний в секунду. Для этого на 1 миллисекунду мы подадим на пищалку 5 вольт и 1 миллисекунду позволим ёмкости пьезоэлемента разрядиться. Измените значение в предыдущем коде на 1 миллисекунду и загрузите программу в Iskra Neo:

```
void setup()
{
    pinMode(13, OUTPUT);
}

void loop()
{
```

²⁰ Вообще, низкие звуки хорошо умеют воспроизводить только акустические системы размером с небольшой холодильник.

```

        digitalWrite(13, HIGH); delay(1);
        digitalWrite(13, LOW);
        delay(1);
    }

```

Вы получите монотонный писк на частоте 500 Гц.

А теперь давайте изменим нашу программу для азбуки морзе таким образом, чтобы она воспроизводила звук.

Для этого загрузите в Iskra Neo следующий код:²¹

```

int soundPin = 13; // Номер пина с пищалкой
int dotDelay = 50; // Время длительности «точки»
char theword[] = "sos";
long time; // Вспомогательная переменная
int rate = 100; // Период колебания в микросекундах.

        // Чем больше период, тем ниже частота
void sound(int duration)
{
    time = millis();
    while (millis() - time < duration) {
        digitalWrite(soundPin, HIGH);
        delayMicroseconds(rate);
        digitalWrite(soundPin, LOW);
        delayMicroseconds(rate);
    }
}
// Точка
void dot()
{
    sound(dotDelay);
    delay(dotDelay);
}
// Тире
void dash()

```

²¹ File → Examples → Amperka → p05_morse_buzzer

```

{
    sound(3 * dotDelay);
    delay(dotDelay);
}

// Окончание буквы
void letterEnd()
{
    delay(2 * dotDelay);
}

// Окончание слова
void wordEnd()
{
    delay(6 * dotDelay);
}

// Слово
void morseWord(char theword[])
{
    int len = strlen(theword);
    for(int i = 0; i < len; ++i)
        morseLetter(theword[i]);
    wordEnd();
}

// Буква
void morseLetter(char c)
{
    // не красивая, но компактная запись уже
    // встречавшегося выражения для сопоставления
    // букв и их кодов в азбуке Морзе
    switch(c) {
        case 'a': dot();dash(); break;
        case 'b': dash();dot();dot();dot(); break;
}

```

```
        case 'c': dash();dot();dash();dot(); break;
        case 'd': dash();dot();dot(); break;
        case 'e': dot(); break;
        case 'f': dot();dot();dash();dot(); break;
        case 'g': dash();dash();dot(); break;
        case 'h': dot();dot();dot(); break;
        case 'i': dot();dot(); break;
        case 'j': dot();dash();dash();dash(); break;
        case 'k': dash();dot();dash(); break;
        case 'l': dot();dash();dot();dot(); break;
        case 'm': dash();dash(); break;
        case 'n': dash();dot(); break;
        case 'o': dash();dash();dash(); break;
        case 'p': dot();dash();dash();dot(); break;
        case 'q': dash();dash();dot();dash(); break;
        case 'r': dot();dash();dot(); break;
        case 's': dot();dot();dot(); break;
        case 't': dash(); break;
        case 'u': dot();dot();dash(); break;
        case 'v': dot();dot();dot();dash(); break;
        case 'w': dot();dash();dash(); break;
        case 'x': dash();dot();dot();dash(); break;
        case 'y': dash();dot();dash();dash(); break;
        case 'z': dash();dash();dot();dot(); break;
        case ' ': wordEnd(); break;
    }
    letterEnd();
}
void setup()
{
    pinMode(soundPin, OUTPUT);
```

```

}
void loop()
{
    morseWordtheword();
}

```

Итак, теперь ваше устройство воспроизводит сигнал SOS уже при помощи звука.

Как это работает?

Как мы видим, отличие от прошлой программы заключается лишь в том, что появилась ещё одна процедура — sound, а также две переменных: rate и time.

```

long time; // Вспомогательная переменная
int rate = 100; // Период колебания в микросекундах.
                // Чем больше период, тем ниже частота
void sound(int duration)
{
    time = millis();
    while(millis() - time < duration) {
        digitalWrite(soundPin, HIGH);
        delayMicroseconds(rate);
        digitalWrite(soundPin, LOW);
        delayMicroseconds(rate);
    }
}

```

rate — это периодичность подачи напряжения на пищалку. Как мы помним, пищалка воспроизводит звук, если на ней постоянно подавать и убирать напряжение. Что мы и делаем с периодом в

$100+100 = 200$ микросекунд. То есть, 5000 раз в секунду, то есть заставляем пищалку звучать на частоте 5 кГц. Если вы хотите, чтобы звук был более низким, вы можете изменить это значение, например, на 200, 1000 или 4000. Если же вы возьмёте очень большое значение (скажем, 50 000), то услышите просто последовательные щелчки: басы пьезоэлемент воспроизводят отвратительно.

Процедура sound принимает параметр duration. Этот параметр означает время, в течение которого должен воспроизводиться сигнал. Встроенная функция millis возвращает текущее время (время с момента включения Iskra Neo) в миллисекундах. Таким образом, мы сохраняем начальное время во вспомогательную переменную time. Затем всё время, пока текущее время в миллисекундах минус сохранённое вре-

мя не превысит значение `duration` (в случае точки, например, оно составляет 50 миллисекунд), мы подаём напряжение на 100 микросекунд (значение переменной `rate`) и убираем напряжение на 100 микросекунд. Иными словами, весь отрезок времени, равный `duration`, мы периодически подаём и убираем напряжение с пищалки.

Отдельно обратим внимание, что для переменной `time` мы использовали тип данных `long`, а не `int`. Это связано с тем, что время, измеряемое в миллисекундах, может принимать большие значения. Так, на Arduino `int` может содержать значения не более 32 767, а `long` — до 2 с небольшим миллиардов. Процессор работает с `long` несколько медленнее, чем с `int`, поэтому, если есть уверенность в том, что переменная не будет принимать большие значения, лучше использовать `int`.

Можете попробовать изменить значение `dotDelay`. Чем меньше это значение, тем быстрее проигрывается предложение.

Итак, вы узнали основы работы со звуком. Вероятно, вы занимаетесь музыкой, тогда вы знаете что такое нота, октава, длительность.

Пользуясь этими знаниями и тем, что вы сегодня узнали, вы наверняка сможете с помощью *Iskra Neo* проиграть мелодию из *Super Mario*, не так ли?





§6. СОЕДИНЕНИЕ С КОМПЬЮТЕРОМ

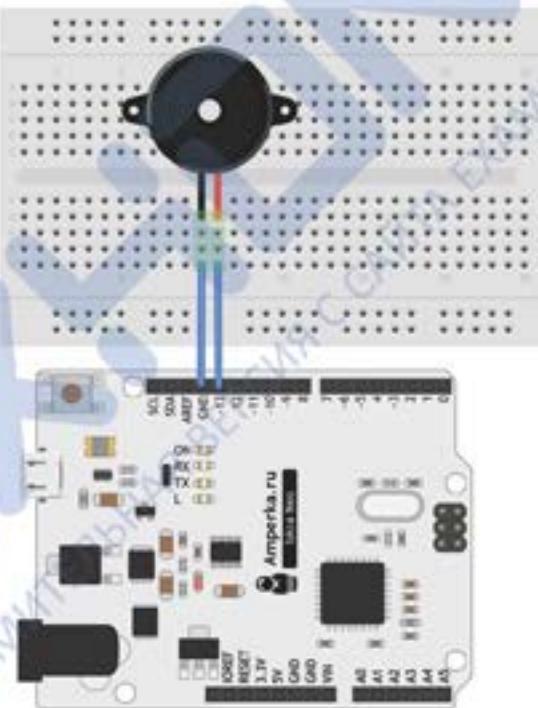
6.1 Последовательный порт, параллельный порт, UART

6.2 Как передавать данные с компьютера на Iskra Neo

6.3 Как научить компьютер говорить на азбуке Морзе

СОЕДИНЕНИЕ С КОМПЬЮТЕРОМ

§6



§6. СОЕДИНЕНИЕ С КОМПЬЮТЕРОМ

6.1 Как Iskra Neo общается с компьютером: последовательный порт, параллельный порт, UART

Iskra Neo может взаимодействовать со множеством различных устройств. Давайте разберемся, как Iskra Neo общается с компьютером. Для начала рассмотрим, как вообще происходит обмен данными между различными устройствами.

Вообще, данные могут передаваться по проводам или «по воздуху» (так называемая беспроводная связь. Например, Wi-Fi, Bluetooth, ИК). К тому же существует довольно большое количество разных способов передачи данных. Однако все их, как правило, можно разделить на четыре класса в зависимости от двух параметров:

- 1. Последовательные / Параллельные**
- 2. Синхронные / Асинхронные**

Последовательный способ передачи данных использует один канал связи для обмена данными. Под каналом понимается один или два провода: один для передачи данных (его обозначают символами Tx) и ещё один для приёма данных (его обозначают Rx). Если устройству требуется только передача или только приём, тогда второй провод не используется. Таким образом, вся информация, которая есть ни что иное, как последовательность битов, в одном направлении передаётся по одному проводу. Принимающее устройство постепенно считывает каждый бит и формирует из них байты, из байтов — килобайты, мегабайты и т.д. Такой способ связи сегодня очень распространён. Он используется в COM-портах, USB²² (универсальная последовательная шина данных), Ethernet (интернет-кабель), S-ATA (шина данных для жёстких дисков) и во многих других стандартах. Слово «последовательный» в этом смысле по-английски звучит, как «Serial». Запомните его — это слово будет вам встречаться очень часто при разработке под микроконтроллеры.

Параллельный способ передачи данных, в отличие от последовательного, передаёт информацию сразу по нескольким каналам. С таким типом передачи мы с вами познакомимся в параграфе, когда будем подключать LCD-дисплей. Там четыре провода отвечают за передачу данных. По всем этим проводам с Iskra Neo одновременно передается по одному биту: 0 (низкий уровень сигнала — 0 В) или 1 (высокий уровень сигнала — 5 В). Дисплей считывает все эти 4 бита одновременно и за два таких считывания собирает один байт (8 бит). Синхронная передача данных использует отдельный канал связи для передачи так называемого тактового сигнала. Он нужен для согласованного обмена данных между устройствами. Тактовый сигнал говорит о том, что «данные готовы, можно считывать!». При подключении экрана, мы будем использовать такой. То есть в итоге, ЖК-экран использует параллельный синхронный тип связи.

²² USB — англ. аббревиатура от Universal Serial Bus

Асинхронная коммуникация, в отличие от синхронной, не использует тактовый сигнал. Это позволяет менять скорость передачи данных и сэкономить на проводах. Скорость передачи данных измеряется в бодах. Бод — это количество бит передаваемых в секунду. Есть несколько стандартных скоростей: 4800, 9600, 19200, 38400, 57600, 115200. Так, например, скорость 9600 бод — это скорость, при которой по проводу за одну секунду передаётся 9600 бит (т.е. больше 1 КБ данных).

Iskra Neo для общения с компьютером использует асинхронный последовательный порт. Компьютер имеет специальное устройство для общения по асинхронному последовательному порту. Такое устройство называется **UART**²³ или универсальный асинхронный приемник и передатчик.

Мы можем легко общаться с Iskra Neo, просто подключив её кабелем к любому USB-порту компьютера.

6.2 Как передавать данные с компьютера на Iskra Neo:

Serial, baud, read, write, print

Итак, давайте попробуем передать что-нибудь с компьютера.

Для этого загрузите на Iskra Neo следующий код:²⁴

```
void setup()
{
    // Инициализируем последовательный порт
    // и говорим ему, что будем работать
    // на скорости 9600 бод
    Serial.begin(9600);
    // Посыпаем по последовательному порту
    // сообщение "Hello"
    Serial.println("Hello");
}

void loop()
{
    // Смотрим, пришла ли какая-нибудь
    // информация с компьютера
    if (Serial.available() > 0) {
        // Пришла! Считываем её в переменную b
        byte b = Serial.read();
        // Отправляем считанный байт
    }
}
```

²³ Аббревиатура от Universal Asynchronous Receiver and Transmitter

²⁴ File → Examples → Amperka → p06_serial_hello

```
// обратно на компьютер
Serial.write(b);
}
}
```

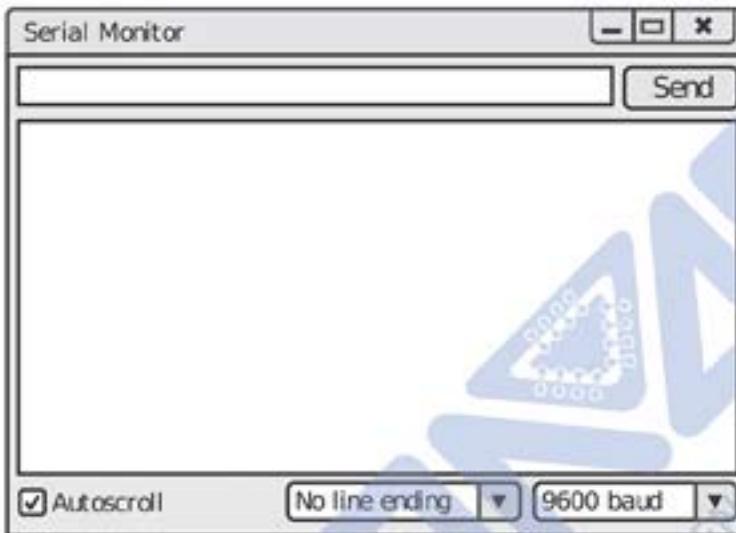


Рис. 6.1: Выбор скорости коммуникации в окне Serial Monitor

Теперь откройте в среде разработки терминал последовательного порта, доступный через меню Tools → Serial Monitor или через кнопку на панели инструментов. Установленная скорость передачи данных на обоих устройствах должна совпадать, чтобы они могли понимать друг друга. Поэтому выберите скорость 9600 бод (рис. 6.1), которую мы установили в загруженной на Iskra Neo программе.

В открывшемся окне должно появиться слово «Hello». Теперь можете вводить любой текст в поле ввода, и он после отправки (нажатия Enter) будет отображаться у вас на экране.

Обратите внимание, что на плате Iskra Neo есть два светодиода: RX и TX. RX загорается, когда мы отправляем на Iskra Neo какие-нибудь данные, TX загорается, когда мы данные отправляем с Iskra Neo.

Как это работает

`Serial` — это особый встроенный объект (экземпляр класса) для работы с последовательным портом, который имеет методы `begin`, `read`, `write`, `print` и другие.

Метод `begin` открывает порт для передачи данных. В этот метод передаётся значение скорости в бодах, на которой будет происходить общение. Если мы собираемся использовать последовательный порт Iskra Neo в программе, первое, что мы должны сделать — вызвать этот метод.

Метод `available` возвращает целое число, говорящее, сколько присланных байт Iskra Neo ещё не прочитала. Если оно равно нулю, это значит, что никаких данных по последовательному порту не приходило. Данные, если их не прочитать, откладываются в буфер (склад данных) и ждут, пока их не прочитают.

Метод `read` считывает пришедший байт.

Метод `print` передаёт по последовательному порту с Iskra Neo текст или число, переданные в качестве аргумента. Процедура `println25` делает то же, что и `print`, но при этом ставит символ перехода на новую строку в конце сообщения. Если мы пишем `println`, то последующая информация будет передаваться уже с новой строки. Если мы хотим передать всего один байт «как есть», а не строку или число в строковом представлении, следует использовать метод `write`.

В нашей процедуре `loop` мы смотрим, не появился ли для нас непрочитанный байт на последовательном порту. Если такой есть, мы его считываем и тут же отправляем обратно. Таким образом, наша Iskra Neo с помощью этой программы работает, как устройство попугай.

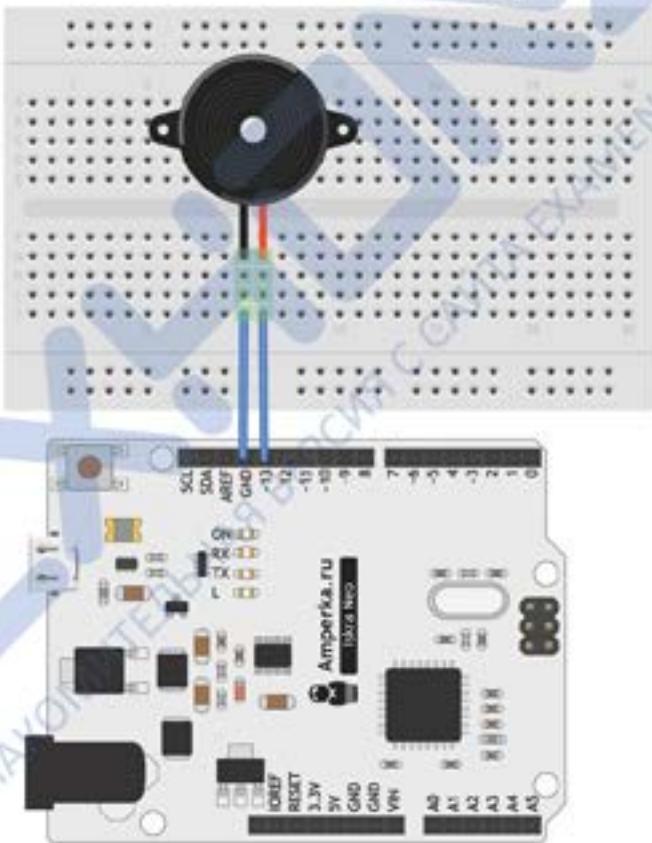


Рис. 6.2: Схема подключения пьезо-излучателя на макетной доске

²⁵ `println` — это сокращение от `print line`: напечатать строку

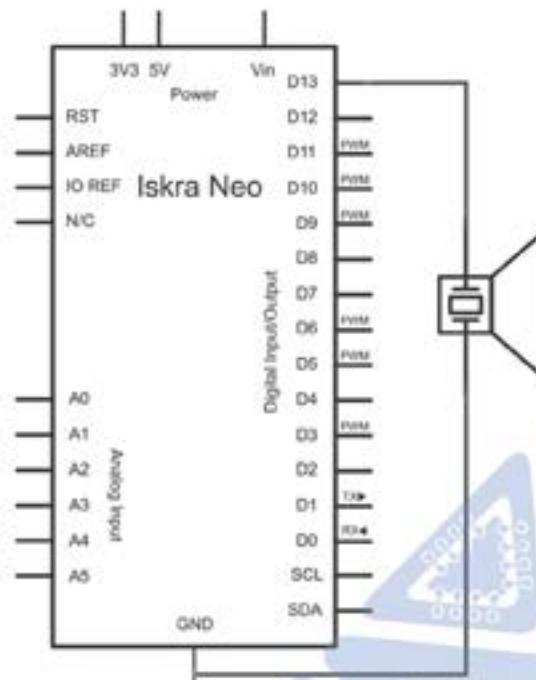


Рис. 6.3: Принципиальная схема подключения пьезо-излучателя

6.3 Как научить компьютер говорить на азбуке Морзе при помощи Iskra Neo

А теперь давайте объединим знание о последовательном порте и пищалке из параграфа 5. Мы сделаем такое устройство, которое будет воспроизводить на азбуке Морзе любую фразу, которую мы введём на компьютере.

Для этого соберите ту же схему, что и ранее. Она изображена на рисунках 6.2 и 6.3. И загрузите следующий код:²⁶

```
int soundPin = 13; // Номер пина с пищалкой
int dotDelay = 50; // Время длительности «точки»
long time; // Вспомогательная переменная
int rate = 100; // Период колебания в микросекундах.
                // Чем больше период, тем ниже частота
void sound(int duration)
{
    time = millis();
    while (millis() - time < duration) {
        digitalWrite(soundPin, HIGH);
```

²⁶ File → Examples → Amperka → p06_serial_morse

```

        delayMicroseconds(rate);
        digitalWrite(soundPin, LOW);
        delayMicroseconds(rate);
    }
}

// Точка
void dot()
{
    sound(dotDelay);
    delay(dotDelay);
}

// Тире
void dash()
{
    sound(3 * dotDelay);
    delay(dotDelay);
}

// Окончание буквы
void letterEnd()
{
    delay(2 * dotDelay);
}

// Окончание слова
void wordEnd()
{
    delay(6 * dotDelay);
}

// Буква void morseLetter(char c)
{
    switch (c) {
        case 'a': dot();dash(); break;
        case 'b': dash();dot();dot();dot(); break;
    }
}

```

```
        case 'c': dash();dot();dash();dot(); break;
        case 'd': dash();dot();dot(); break;
        case 'e': dot(); break;
        case 'f': dot();dot();dash();dot(); break;
        case 'g': dash();dash();dot(); break;
        case 'h': dot();dot();dot(); break;
        case 'i': dot();dot(); break;
        case 'j': dot();dash();dash();dash(); break;
        case 'k': dash();dot();dash(); break;
        case 'l': dot();dash();dot();dot(); break;
        case 'm': dash();dash(); break;
        case 'n': dash();dot(); break;
        case 'o': dash();dash();dash(); break;
        case 'p': dot();dash();dash();dot(); break;
        case 'q': dash();dash();dot();dash(); break;
        case 'r': dot();dash();dot(); break;
        case 's': dot();dot();dot(); break;
        case 't': dash(); break;
        case 'u': dot ();dot();dash(); break;
        case 'v': dot();dot();dot();dash(); break ;
        case 'w': dot();dash();dash(); break;
        case 'x': dash();dot();dot();dash(); break;
        case 'y': dash();dot();dash();dash(); break;
        case 'z': dash();dash();dot();dot(); break;
        case '_': wordEnd(); break;
    }
    letterEnd();
}
void setup()
{
    pinMode(soundPin, OUTPUT);
    Serial.begin(9600);
```

```

}

void loop()
{
    // Если есть не считанный байт
    if (Serial.available() > 0) {
        // Считываем его...
        char c = Serial.read();
        // ...и воспроизводим на азбуке Морзе
        morseLetter(c);
    }
}

```

Теперь откройте терминал Arduino IDE и введите любую фразу маленькими английскими буквами. Iskra Neo её просигналит.

По большому счёту эта программа отличается от уже сделанной нами ранее лишь одним. Вместо того, чтобы всё время писать строку, жёстко прописанную в программе, теперь мы принимаем её с компьютера. Мы не тронули никакие функции, кроме `setup` и `loop`, но получили гораздо более полезное устройство. Почувствуйте, каким мощным инструментом является грамотное разделение кода на процедуры функции.

Итак, вы познакомились с различными способами передачи данных. На практике использовали последовательную асинхронную коммуникацию с компьютером. Это был лишь пример: не всегда общение ведётся с компьютером. Бывают модули, датчики и даже моторы, с которыми обмен данными происходит такими высокоразвитыми способами.

После небольшой практики вам не должно составить труда использовать любые из них.



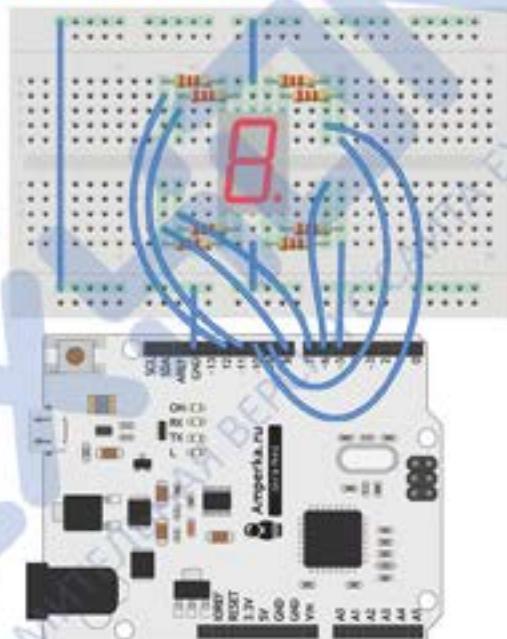
§7. СЕМИСЕГМЕНТНЫЙ ИНДИКАТОР

- 7.1 Как работает индикатор
- 7.2 Как включить индикатор
- 7.3 Как научить Iskra Neo считать до десяти



СЕМИСЕГМЕНТНЫЙ ИНДИКАТОР

§7



§7. СЕМИСЕГМЕНТНЫЙ ИНДИКАТОР

7.1 Как работает индикатор

Светодиоды — это хорошо. Пищалка — это тоже хорошо. Но иногда требуется простая и понятная человеку индикация. К примеру, циферная.

Для этого были придуманы семисегментные индикаторы. Посмотрите на рисунок 7.1, чтобы понять, как они выглядят. Каждая чёрточка в восьмёрке, изображённой на рисунке, называется сегментом. Посчитайте: их ровно семь. Поэтому индикатор и называется семисегментным.

Индикаторы бывают механическими: те, в которых сегменты поворачиваются то белой, то чёрной стороной. Их можно собрать из люминисцентных ламп. Лет сорок назад активно применялись ламповые семисегментные индикаторы. Электронным антиквариатом ныне стали индикаторы на флюоресцентных стёклах. В настоящее время чаще всего используются светодиодные индикаторы. Технически это восемь уже прекрасно вам знакомых светодиодов: по одному на каждый сегмент и ещё один для отображения десятичной точки.

Индикаторы, как и многие многоцветные светодиоды, светодиодные и диодные сборки, делятся на те, что с общим катодом, и те, что с общим анодом. Как следует из названия, у них объединены либо отрицательные выводы — такие сборки управляются подачей положительного напряжения на аноды; либо положительные — тогда на общий анод подаётся питание, а управление индикатором замыканием отдельных катодов на землю.



Рис. 7.1: Семисегментный индикатор

7.2 Как включить индикатор

Так как это просто набор светодиодов, давайте подключим их как обычно: к цифровым выходам Iskra Neo. Возьмём индикатор с общим катодом и соберём схему, как показано на рисунках 7.2 и 7.3. Мы соединяем каждый анод с отдельным пином Iskra Neo через токоограничивающие резисторы. Как и раньше, резисторы номиналом в 200–240 Ом вполне подойдут. Общий катод подключаем к земле: он один на всех.

Загрузим следующий код:²⁷

```
// Пины, на которых расположен индикатор
int pins[8] = {5, 6, 7, 8, 9, 10, 11, 12};
void setup()
{
    // Для светодиода в индикаторе инициализируем
    // пин, соответствующий ему, и подаём 5 В.
    for(int i = 0; i < 8; i++){
        pinMode(pins[i], OUTPUT);
    }
}
```

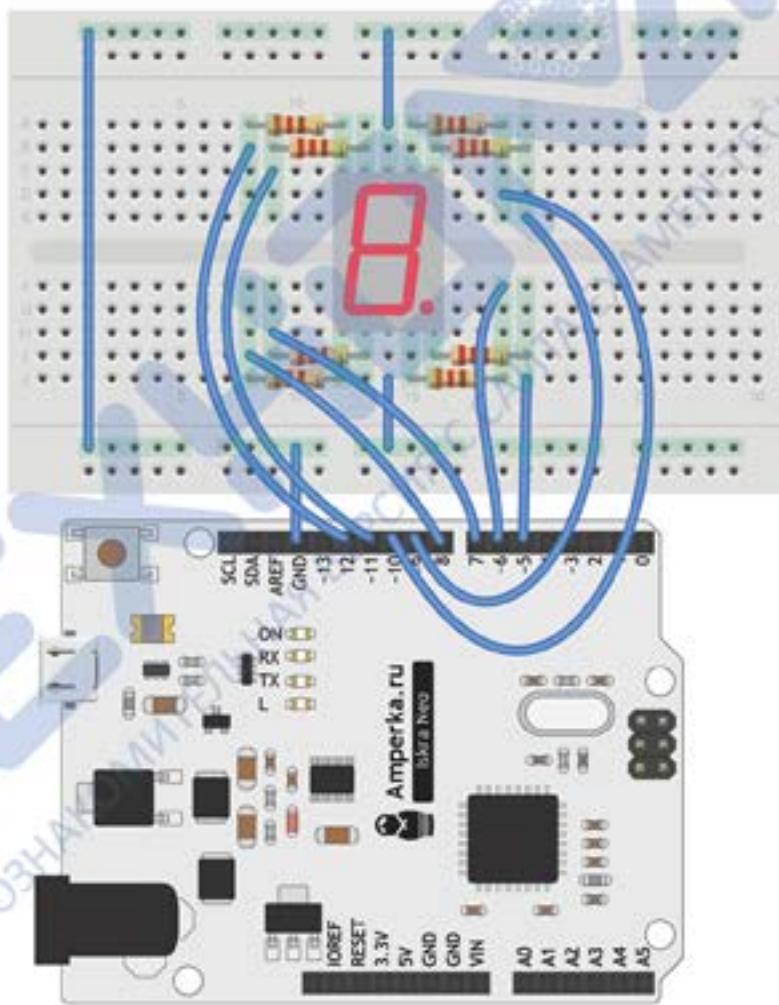


Рис. 7.2: Схема включения семисегментного индикатора на макетной доске

²⁷ File → Examples → Amperka → p07_7_segment

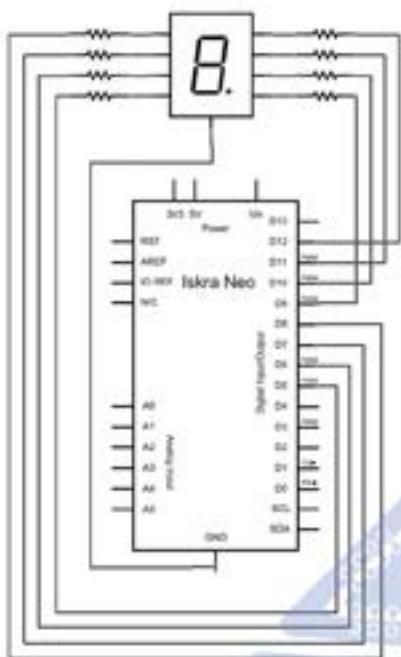


Рис. 7.3: Принципиальная схема включения семисегментного индикатора

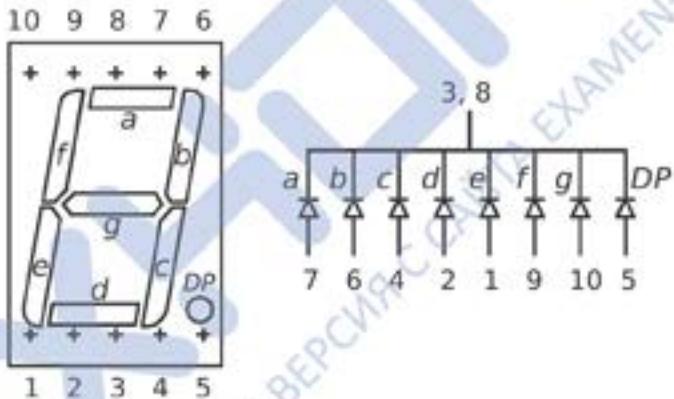


Рис. 7.4: Кодировка чисел в семисегментном индикаторе

```

        digitalWrite(pins[i], HIGH);
    }
}

void loop()
{
    // мы всё уже сделали в setup
    // цикл loop остаётся пустым
}

```

Скетч подаст на все задействованные выходы высокий уровень, и мы увидим восьмерку.

7.3 Как научить Iskra Neo считать до десяти

Счёт, естественно, не ограничивается восьмёркой. Посмотрите на иллюстрацию 7.4. На ней слева видно, что каждый сегмент обозначен буквой латинского алфавита, а сверху и снизу изображена нумерация выводов семисегментного индикатора. Соответствие каждого сегмента и номера вывода изображено справа.

Так, видно, что выводы 3 и 8 - это общий катод - их нужно подключить к GND, а чтобы загорелся сегмент обозначенный буквой a нужно будет через токоограничивающий резистор подать напряжение на вывод с номером 7. Так можно закодировать цифры от нуля до девя蒂. Весьма напоминает цифры на калькуляторе.

Не мудрено: в калькуляторах используются такие же индикаторы, просто они не светодиодные, а жидкокристаллические.

Итак, например, для вывода пятёрки надо подать напряжение на сегменты a, f, g, c, d. А двойки – a, b, g, e, d. Давайте напишем функцию, которая за нас переводит привычные цифры в разряды индикатора:²⁸

```
/*
Пины, на которых расположен индикатор
5-й пин - точка
6-й пин - нижний правый светодиод (с)
7-й пин - нижний светодиод (d)
8-й пин - нижний левый светодиод (e)
9-й пин - верхний правый светодиод (b)
10-й пин - верхний светодиод (a)
11-й пин - верхний левый светодиод (f)
12-й пин - средний светодиод (g)
*/
int pins[8] = {5, 6, 7, 8, 9, 10, 11, 12};
```

Массив массивов единиц и нулей. Каждый элемент массива является другим массивом, обозначающим, какие пины (по порядку, начиная с шестого, то есть с pins[1]) должны быть включены, чтобы загорелась соответствующая цифра

1 означает, что пин должен быть включён

²⁸ File → Examples → Amperka → p07_7_segment_count

```

0 означает, что пин должен быть выключен

*/
boolean digits[10][7] = {
    {1,1,1,1,1,1,0},
    {1,0,0,1,0,0,0},
    {0,1,1,1,1,0,1},
    {1,1,0,1,1,0,1},
    {1,0,0,1,0,1,1},
    {1,1,0,0,1,1,1},
    {1,1,1,0,1,1,1},
    {1,0,0,1,1,0,0},
    {1,1,1,1,1,1,1},
    {1,1,0,1,1,1,1}
};

/*
Процедура, отображающая на индикаторе
определенную цифру
*/
void showDigit(int num)
{
    // Пробегаемся по элементам массива digits
    // и смотрим, если на i-м месте этого элемента
    // стоит 1, включаем соответствующий пин,
    // если 0 - выключаем
    for (int i = 0; i < 7; i++)
        digitalWrite(pins[i+1], digits[num][i]);
}

void setup()
{
    // Для каждого светодиода в индикаторе
    // инициализируем пин, соответствующий ему,
    // как работающий на вывод
    for (int i = 0; i < 8; i++)

```

```

pinMode(pins[i], OUTPUT); }

void loop() {
    // Пробегаем все цифры от 0 до 9 и выводим их
    // После каждой ждём одну секунду
    for (int i = 0; i <= 9; i++) {
        showDigit(i);
        delay(1000);
    }
}

```

Загрузите скетч на Iskra Neo и наблюдайте, как устройство ведёт отсчёт от 0 до 9 и затем начинает всё заново.

В приведённой программе мы впервые встретились с понятием многомерного массива: `boolean digits[10][7]`. Это не что иное, как массив из 10 массивов из 7 элементов каждый. Просто рассматривайте массив из 7 элементов как новый, отдельный тип данных, вещь в себе. В нашем случае он определяет, какие сегменты нужно зажигать, а какие нет. Элементов такого типа данных нам нужно 10: для всех цифр от 0 до 9. Удобным способом определить такую структуру данных и является многомерный массив размером 10×7 .

Вы можете создавать трёх-, четырёхмерные массивы. Количество измерений не ограничено. Главное, затем в них не запутаться.

Возвращаясь к индикаторам, как вывести двузначное число? Правильно, можно поставить ещё один индикатор рядом с имеющимся, модифицировать программу и показывать числа от 0 до 99. Однако, в этом случае будет занято как минимум 14 выводов Iskra Neo. Довольно расточительно для вывода всего двух цифр.

О том, как сэкономить выводы, мы поговорим в следующем параграфе.

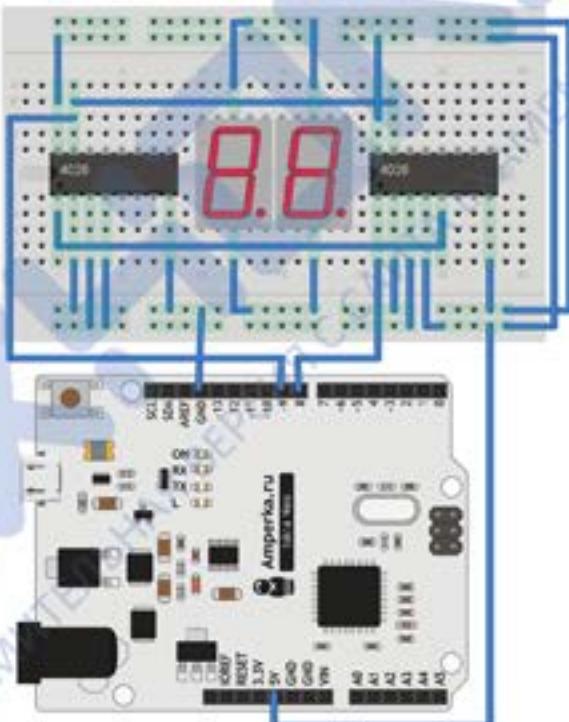




§8. МИКРОСХЕМЫ

- 8.1 Зачем нужны микросхемы
- 8.2 Как упростить работу с индикатором: драйвер CD4026
- 8.3 Как сосчитать до 99 при помощи драйвера
- 8.4 Как вывести произвольное число

МИКРОСХЕМЫ



§8. МИКРОСХЕМЫ

8.1 Зачем нужны микросхемы

В общем-то, все люди одинаковые. Каждому из нас требуется еда, одежда и кров. Стулья, диваны, табуреты, автомобили и обои на кухню, книги, окорочки, мысли и клавиатуры создаются индустриальным способом, и, совершенно одинаковые, подходят разным людям. Электронщики ничем не отличаются от остальных людей: многим для абсолютно разных схем требуются одинаковые усилители сигналов, преобразователи интерфейсов, ячейки памяти, микроконтроллеры, акселерометры, драйверы транзисторов, генераторы импульсов.

Раньше, лет двести назад, одежда и обувь шилась на каждого, от крестьянина до царя, индивидуально. Женой, матерью или личными портными — но каждая вещь делалась вручную уникальной. Сукно же в это время уже зачастую производилось на мануфактурах. Сегодня сотни и тысячи одинаковых штанов и кроссовок выпускаются на фабриках, потому что так значительно дешевле. То же самое произошло с электронными компонентами. Если лет пятьдесят назад усилитель или логический элемент каждый радиолюбитель или инженер собирал из отдельных простейших деталей: резисторов, транзисторов, ламп, индуктивностей, то сейчас они создаются на поверхности микроскопического кристалла кремния. К кристаллу подключаются металлические ножки, он заливается пластиком — и вот, перед нами микросхема, стандартный кубик электроники. Из этих кубиков схемы собираются намного быстрее.

Микросхемы различаются по размеру, форме и количеству ножек, но суть остается неизменной. Каждая микросхема решает свою маленькую задачу и делает это хорошо. Например, одна микросхема может в 1000 раз усиливать слабый входной сигнал; другая — выдавать пульсы каждую секунду и считать минуты, часы, дни, года; третья — управлять USB-подключением. Разновидностей микросхем — десятки тысяч.

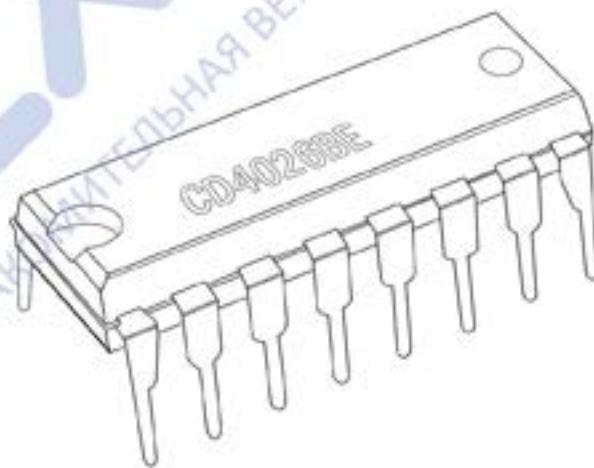


Рис. 8.1: Микросхема CD4026

8.2 Как упростить работу с индикатором: драйвер CD4026

К примеру, на рисунке 8.1 изображена микросхема-драйвер CD4026 для 7-сегментного светодиодного индикатора. Она умеет считать от нуля до девяти и выдавать значение счётчика на подключённый к ней 7-сегментный индикатор. Драйвером²⁹ она называется, так как управляет другим, более примитивным устройством. В данном случае — индикатором.

Эта микросхема имеет 16 ножек как показано на схеме 8.2. Давайте поймём, за что отвечает каждая ножка.

Каждый импульс, т.е. подачу и сброс напряжения, приходящий ей на ножку *clock*, увеличивает счётчик на единицу. Значение счётчика хранится внутри микросхемы бесконечно, пока на неё подаётся питание. После девятки счёт снова начинается с нуля.

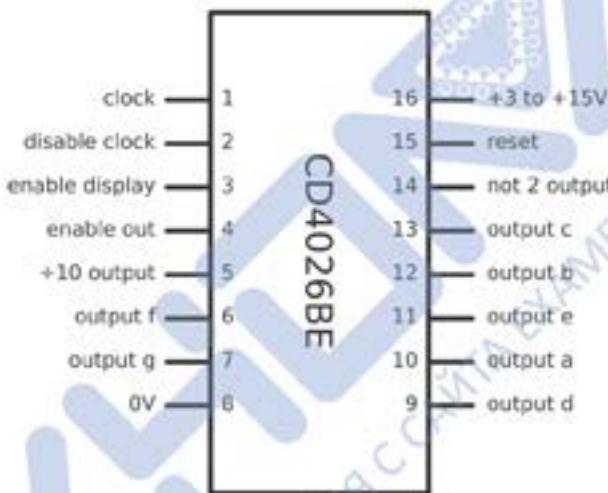


Рис. 8.2: Распиновка микросхемы CD4026

Ножка *reset* отвечает за сброс значения. Каждый раз, когда на эту ножку подаётся импульс, значение счётчика обнуляется. Ножка $\div 10$ посылает импульс каждый раз, когда счётчик дошёл до девятки и к нему прибавляется единица. В таком случае счётчик становится равным нулю, а на ножку подаётся напряжение. Именно поэтому, если соединить эту ножку с ножкой *clock* другой микросхемы, её счётчик будет увеличиваться каждый раз, когда счётчик этой дошёл до десяти. Таким образом, мы можем организовать подсчёт двузначных чисел. Конечно, если мы проделаем то же самое со второй микросхемой, т.е. соединим ножку $\div 10$ второй микросхемы с ножкой *clock* третьей, мы получим уже возможность выводить трёхзначные числа. И так далее.

Через ножку $+3$ to $+15V$ подаётся питание на микросхему, ножка $0V$ должна быть соединена с землёй. Ножка *enable display* управляет тем, нужно ли показывать цифру на индикаторе или нет; стоит подать на неё 5 В, чтобы мы увидели показания.

Все ножки, обозначенные как *output X*, управляют соответствующими сегментами индикатора. Их нумерацию и расположение мы обсуждали в предыдущем параграфе.

²⁹ От англ. driver: водитель, рулевой

8.3 Как сосчитать до 99 при помощи драйвера

Всё это позволяет сделать с помощью данной микросхемы индикатор с любым количеством цифр, управляемый всего двумя ножками микроконтроллера! Давайте соберём его для двух цифр. Нам потребуется два индикатора и два драйвера.

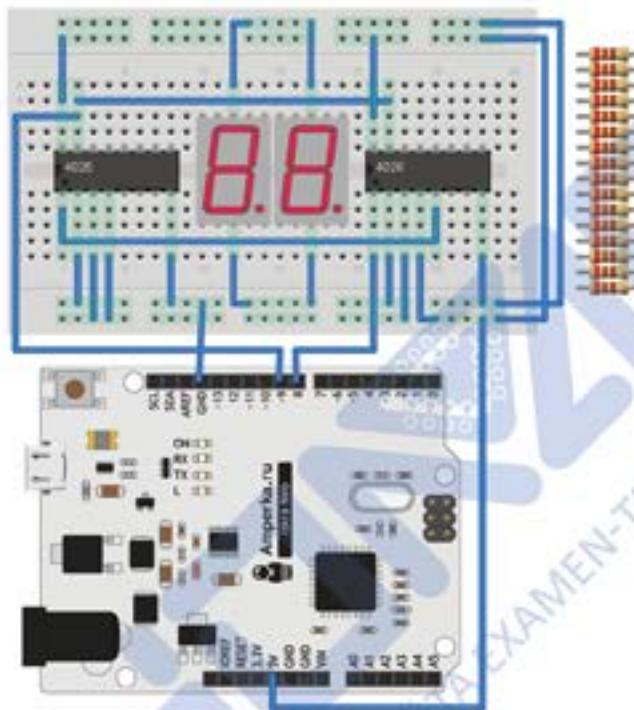


Рис. 8.3: Начало подключения пары микросхем CD4026 на макетной доске

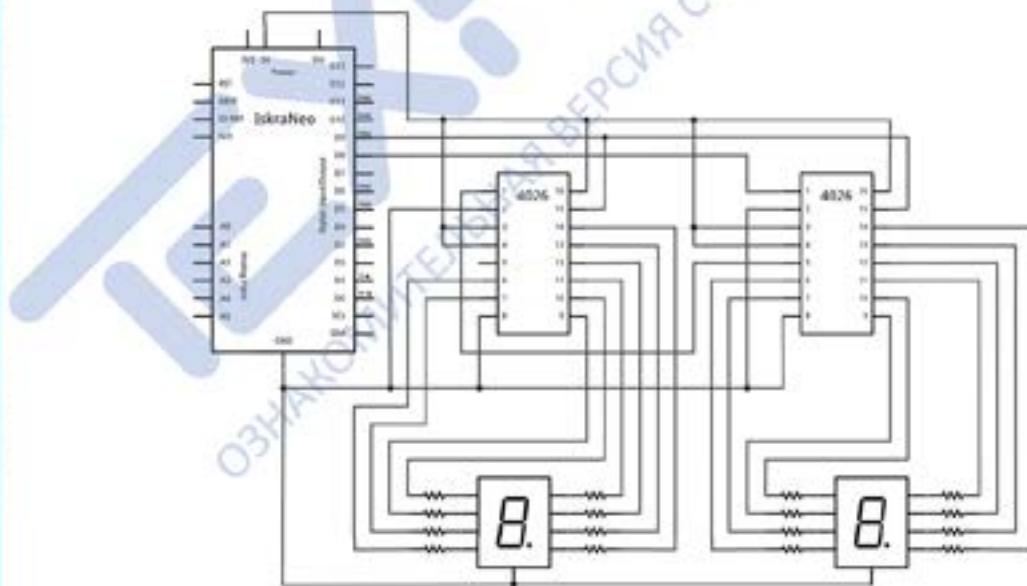


Рис. 8.4: Принципиальная схема включения пары драйверов CD4026 и ведомых индикаторов

Соберите схему, показанную на рисунках 8.3 и 8.4. Вас может привести в ужас количество необходимых соединений. Но на самом деле всё не сложно, просто нужно понять принцип и собирать схему внимательно. Разбейте сборку на отдельные простые этапы:

1. Установите на макетной доске пару индикаторов и микросхем. Обратите внимание, чтобы канавка макетной доски проходила под компонентами, между линейками их ножек. В противном случае, ножки на противоположных сторонах окажутся попарно соединены электрически, и в лучшем случае вы получите не то, что ожидали. В худшем — выведете из строя микросхемы.
2. Подключите к общему питанию и земле ножки компонентов, которые для этого предназначены: θV , $+xV$, катоды индикаторов.
3. Подключите к общему питанию и земле ножки микросхем, которые мы не будем использовать, чтобы они всегда находились в нужном нам состоянии: *disable clock* — к земле (т.е. к нулю), т.к. мы не собираемся отключать линию такта; *enable display* и *enable out* — к питанию (т.е. к единице), т.к. мы хотим, чтобы выводы работали всегда.
4. Подключите линии логики: контакты *reset* обоих драйверов подключите к пину Iskra Neo (в нашем случае мы использовали пин 9); контакт *clock* драйвера левого индикатора — к другому пину Iskra Neo (мы использовали 8-й), а *clock* правого — к контакту « $\div 10$ » левого. Схема 8.3 собрана именно до этого этапа.
5. Далее, ориентируясь на схемы 8.2 и 7.4, соедините соответствующие контакты *a, b, c, d, e, f, g* драйверов и индикаторов, обязательно через токоограничивающие резисторы.

После того, как схема готова, загрузите такой скетч:³⁰

```
int clockPin = 8; // Пин, соединённый с ножкой Clock
int resetPin = 9; // Пин, соединённый с ножкой Reset
void setup()
{
    pinMode(clockPin, OUTPUT);
    pinMode(resetPin, OUTPUT);
    // Посыпаем импульс на ножку Reset
    // для того, чтобы счётчик обнулился
    digitalWrite(resetPin, HIGH);
    digitalWrite(resetPin, LOW);
}
void loop(){
    // Посыпаем импульс на ножку Clock
}
```

³⁰ File → Examples → Amperka → p08_counter

```

    // для того, чтобы добавить к счётчику единицу
    digitalWrite(clockPin, HIGH);
    digitalWrite(clockPin, LOW);
    // Ждём полсекунды
    delay(500);
}

```

Таким образом, микроконтроллер подаёт на вход CD4026 импульсы. Счётчик увеличивается на единицу с каждым импульсом, и соответствующее число показывается на индикаторе. С переходом через девятку импульс подаётся на следующую микросхему, и уже её значение увеличивается на единицу.

8.4 Как вывести произвольное число

Итак, мы научились выводить последовательно числа от 0 до 99, однако нам, чаще всего, нужно вывести просто конкретное число, например, 46. Но как это сделать, если наша микросхема умеет только прибавлять по единице за раз и тут же выводить результат на индикатор?

Ответ очень прост: нужно просто очень быстро пробежать все числа от нуля до заданного числа, скажем, от нуля до сорока шести. Если мы сделаем это очень быстро, то человек даже не заметит, что индикатор ему показал не только сорок шесть, а все числа от нуля до сорока шести. Давайте попробуем сделать это.

Для этого залейте на Iskra Neo следующий код:³¹

```

int clockPin = 8; // Пин, соединённый с ножкой Clock
int resetPin = 9; // Пин, соединённый с ножкой Reset

void setup()
{
    pinMode(clockPin, OUTPUT);
    pinMode(resetPin, OUTPUT);
}

/*
Функция, выводящая на индикатор заданное число
*/
void showNumber(int num)

```

³¹ File → Examples → Amperka → p08_numbers

```

{
    // Посыпаем импульс на ножку Reset
    // для того чтобы счётчик обнулился
    digitalWrite(resetPin, HIGH);
    digitalWrite(resetPin, LOW);

    // Подаём импульс на ножку драйвера Clock
    // пока счётчик, прибавляя по единице,
    // не накопит нужное нам число
    while (num--) {
        digitalWrite(clockPin, HIGH);
        digitalWrite(clockPin, LOW);
    }
}

void loop()
{
    showNumber(4); delay(1000);
    showNumber(8); delay(1000);

    showNumber(15); delay(1000);
    showNumber(16); delay(1000);

    showNumber(23); delay(1000);
    showNumber(42); delay(1000);
}

```

Эта программа заставляет устройство по кругу выводить магический позывной из сериала «Lost».

Вам могла показаться странной конструкция `while (num--)`. Это поначалу немного сложный для понимания способ выполнить цикл столько раз, сколько записано в переменной `num`. Но он быстрый и компактный. Дело в том, что оператор постинкремента `--` возвращает текущее значение переменной, а только потом уменьшает её на единицу.

Таким образом, если `num` изначально равен 1, `num--` вернёт 1, условие `while` проверит его, примет за истину и только затем значение будет уменьшено до 0.

Таким образом, программа попадёт в тело цикла `while` один раз, а при проверке на следующей итерации цикл окончится, т.к. к этому моменту значение переменной `num` уже равно нулю. Проверьте мысленно: если значение переменной изначально равнялось 3, мы попадём внутрь цикла ровно 3 раза. А это то, что нам и нужно.

Итак, вы узнали, что такое микросхемы и на практике познакомились с одной из них. Она дала несомненное преимущество: вместо 14 выводов `Iskra Neo` для вывода двухзначного числа мы использовали всего 2.

Подобным образом другие микросхемы также могут упрощать проект. Главное, подобрать нужную.

Ознакомительная версия с сайта EXAMEN-TECHNOLAB.RU



§9. ШИМ И СМЕШЕНИЕ ЦВЕТОВ

9.1 Понятие ШИМ и инертности восприятия

9.2 Управление яркостью светодиода

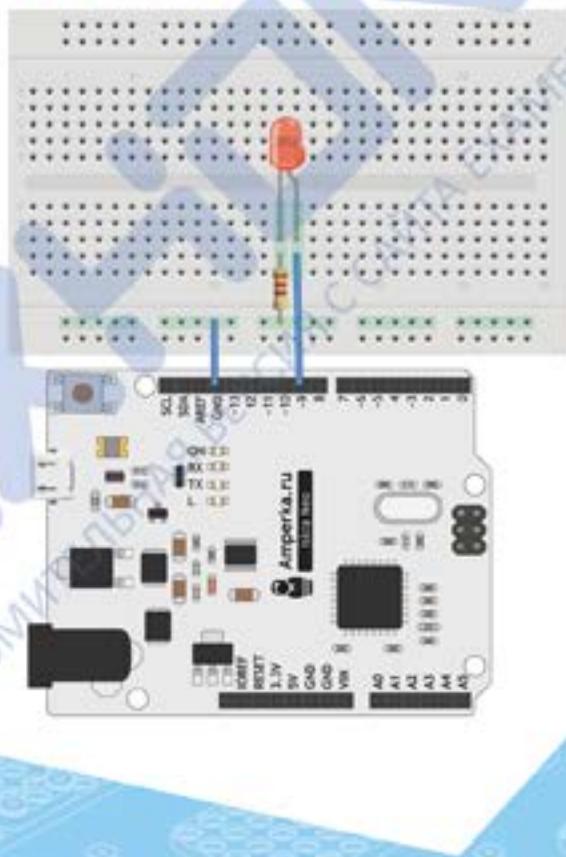
9.3 Смешение и восприятие цветов

9.4 Радуга из трёхцветного светодиода



ШИМ И СМЕШЕНИЕ ЦВЕТОВ

§9



§9. ШИМ И СМЕШЕНИЕ ЦВЕТОВ

9.1 Как обмануть наши чувства: цифровой и аналоговый сигналы, ШИМ, частота, инертность восприятия

Ну что же, мы с вами познакомились с цифровыми сигналами. То есть сигналами, которые несут лишь два состояния: есть напряжение, нет напряжения. Так, светодиод горел, когда напряжение было и не горел, когда его не было. Всё, что имеет два состояния, может быть закодировано при помощи одного бита (1 или 0). Но далеко не все в этом мире может быть определено через ноль и единицу. Яркость свечения лампочки накаливания, скорость двигателя, положение сервомотора — все это имеет свойство плавно изменяться от нуля до какого-то предела. В то же время, микроконтроллеры по-настоящему хорошо умеют работать только с цифровыми сигналами. Но мы хотим управлять и лампами, и двигателями с помощью контроллера. Как нам поступить в таком случае?

Пусть ножка нашего микроконтроллера переключается раз в секунду. Одну секунду светодиод горит, на следующую — выключается. Наш глаз легко определяет мигание. Теперь изменим нашу мигалку, чтобы переключение происходило два раза в секунду; четыре раза в секунду; шестнадцать раз и так далее. Будет ли наш глаз так же успешно разделять отдельные переключения?

Нет. Наш глаз способен воспринимать информацию со скоростью около двадцати четырёх кадров в секунду. Всё, что происходит быстрее, для невооружённого глаза сольётся в мутное пятно — примерно как вращающийся винт вертолёта. Поэтому, когда светодиод станет включаться и выключаться, к примеру, тридцать раз в секунду, для нашего глаза он будет просто гореть в два раза слабее, чем когда он питался постоянным током. Если светодиод в начале горел одну секунду, а гас на две — в три раза слабее. Если светодиод горел две секунды, а не горел одну, при увеличении частоты переключения он станет гореть в две трети яркости. Поняли идею?

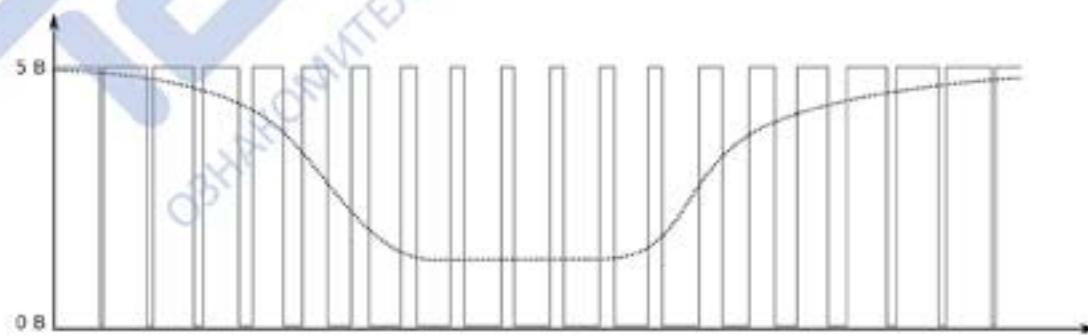


Рис. 9.1: Различие ШИМ и аналогового сигналов

Собственно, в этом и есть суть широтно-импульсной модуляции сигнала, или ШИМ. Мы управляем тем, какой промежуток времени был включен выход микроконтроллера и какой — выключен. При этом переключаем его очень часто. В результате наш инертный глаз воспринимает свечение равномерным. А вот, к примеру, скоростная кинокамера могла бы разделить отдельные кадры.

На рисунке 9.1 можно видеть разницу между аналоговым и ШИМ-сигналом. Аналоговый сигнал изображен пунктиром, а примерно соответствующий ему ШИМ-сигнал — сплошной линией.

Но зачем так сложно? Не проще ли сразу подать на светодиод половину напряжения? Да, проще. Но микроконтроллеры и логические схемы не умеют работать с половинами, третями и другими кусочками. Либо всё, либо ничего. Кроме этого, есть еще одна причина, но о ней мы расскажем, когда будем изучать работу транзисторов.

Если мы хотим повысить равномерность сигнала на выходе, мы всегда можем поставить конденсатор подходящего номинала. Помните, мы сравнивали его с ванной? Конденсатор будет одну треть времени заряжаться, а две трети — разряжаться. В итоге, он сгладит пульсации напряжения. У двигателей, скорость вращения которых также регулируют с помощью ШИМа, все еще лучше. Массивный ротор, вращающийся тысячи раз в минуту, невозможно мгновенно остановить или разогнать вдвое. Его масса сглаживает переключения, переход происходит плавно.

9.2 Как управлять яркостью светодиода: ШИМ, analogWrite

Сделаем устройство, которое будет плавно управлять яркостью подключенного светодиода: то заставлять его набирать яркость, то плавно угасать. Соберите схему, как показано на рисунках 9.2 и 9.3.

Затем загрузите в *Iskra Neo* следующий код:³²

```
int brightness = 0; // изначальная яркость светодиода
int fadeAmount = 5; // скорость затухания/нарастания
// яркости

void setup()
{
    // инициализируем пин 9, как работающий на выход
    pinMode(9, OUTPUT);
}

void loop()
{
```

³² File → Examples → Amperka → p09_led_fade

```
// изменяем яркость светодиода
analogWrite(9, brightness);

// на каждом шаге увеличиваем яркость
// на скорость затухания
brightness = brightness + fadeAmount;

// в конце затухания меняем его на
// нарастание яркости и наоборот
if (brightness == 0 || brightness == 255)
    fadeAmount = -fadeAmount;

// ждём 30 миллисекунд
delay(30);
}
```

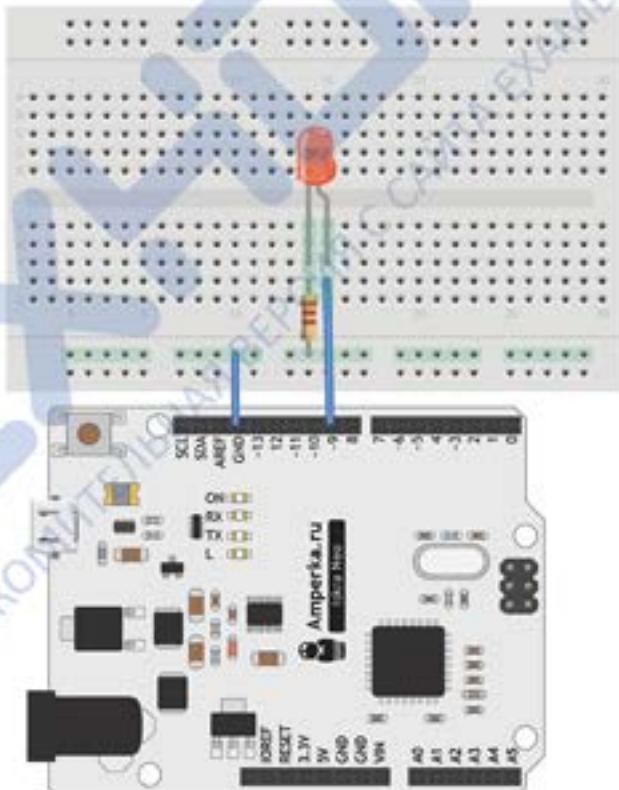


Рис. 9.2: Схема затухающего светодиода на макетной доске

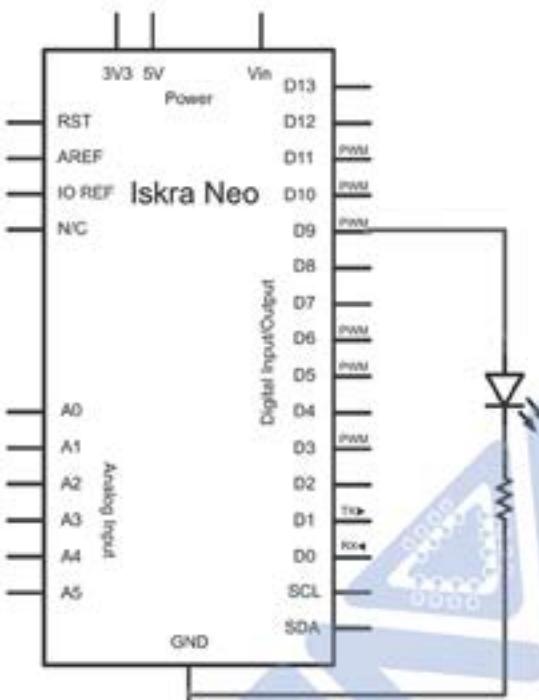


Рис. 9.3: Принципиальная схема затухающего светодиода

Как это работает

В предыдущих параграфах мы часто пользовались функцией `digitalWrite(- номерПина, Значение)`. Ей в качестве значения мы могли подавать либо HIGH, что означало, что на pin нужно подать 5 В; либо LOW, что означало, что на pin нужно подать 0 В.

`analogWrite` устроена точно так же, как и `digitalWrite`, и выполняет ту же функцию, но сюда мы уже можем подавать значения от 0 до 255.³³ В зависимости от этого значения, на выбранный pin подается соответствующий ШИМ-сигнал, смысл которого описан выше.

Переключением между 0 и 5 вольтами занимается Iskra Neo. Он производит эту операцию примерно 32 тысячи раз в секунду, а передаваемым значением мы указываем отношение длительности 0 вольт к длительности 5 вольт при каждом переключении. 0 — это постоянные 0 В; 255 — это постоянные 5 В; 128 — это пополам: то 0 В, то 5 В; 64 — это четверть 0 В и три четверти 5 В; и т.д.

Это отношение продолжительности 0 В к 5 В называют скважностью ШИМ-сигнала.

Обратите внимание, что далеко не все пины поддерживают работу с ШИМ. Те, которые могут это делать, помечены на Iskra Neo символом тильды (~).

³³ 255 — это максимальное значение байта, восемь единиц в двоичной системе счисления, т.к. $256=2^8$

Таким образом, наша программа на каждом шаге прибавляет к подающемуся на pin ШИМ сигналу величину, равную скорости затухания. После того, как ШИМ становится максимальным (255), что значит, что светодиод светит с максимальной яркостью, мы начинаем убавлять величину ШИМ-сигнала, пока он не станет равным нулю. Затем начинаем снова прибавлять и так далее. Для реализации этой задумки мы используем следующий трюк:

```
if (brightness == 0 || brightness == 255)
    fadeAmount = -fadeAmount;
```

Символ «дабл бар» || на языке Arduino означает «или». Всё условие считается выполненным, если хотя бы одно из подусловий brightness == 0 или brightness == 255 справедливо. При достижении граничного значения мы меняем знак переменной fadeAmount на противоположный. Таким образом, затухание превращается в нарастание яркости и наоборот.

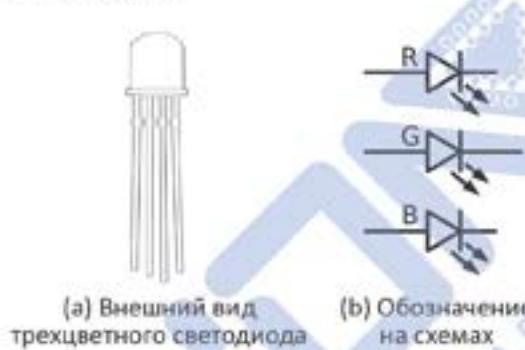


Рис. 9.4: Светодиод

9.3 Можно ли из красного, зелёного и синего получить белый: цвета, пиксели, человеческое восприятие

Теперь, когда мы научились управлять яркостью отдельного светодиода, сделаем радугу с помощью нашего Iskra Neo. Для этого нам потребуется всего одна деталь — трёхцветный светодиод. Это три светодиода: красный, зелёный и синий с одним на всех катодом, собранные в одну деталь. Катод — это отрицательный электрод, который должен подключаться к земле, помните? Трёхцветный светодиод ещё называют RGB³⁴-светодиодом. Он изображён на рис. 9.4.

Как можно из красного, зелёного и синего цвета получить, к примеру, фиолетовый? Немного углубимся в особенности человеческого зрения. У нас в глазу три вида клеток, различающие цвета. Одни включаются от лучей красных и оранжевых, вторые — от зелёных до голубоватых, третьи — от синих до фиолетовых оттенков. Наш мозг судит о принятом глазом цвете по количеству активировавшихся клеток. К примеру, жёлтый свет активирует «красных» и «зелёных» клеток поровну. А оранжевый — больше «красных», чем «зелёных».

³⁴ RGB расшифровывается как «Red Green Blue»: красно-зелено-синий



Рис. 9.5: Принцип смешения цветов

Это значит, что мы можем подменить жёлтый свет на смесь красного и зелёного в равных пропорциях, и мозг не заметит обмана. На этом и основаны все экраны: каждый пиксель монитора перед вами — это три ячейки с красным, зелёным и синим светофильтром управляемой прозрачности. Рассмотрите экран через лупу или поместите на него каплю воды, и вы отчётливо увидите разноцветные элементы пикселей.

Если надо получить жёлтый цвет пикселя, синий светофильтр становится непрозрачным, а светятся только красный и зелёный. Теперь понятно, как три светодиода дадут нам $256^3 = 16777216$ цветов и оттенков. Фраза «шестнадцать миллионов цветов» звучит эффектно, а потому часто фигурирует в рекламах всевозможных гаджетов.

9.4 Как при помощи светодиода сделать радугу: трёхцветный светодиод

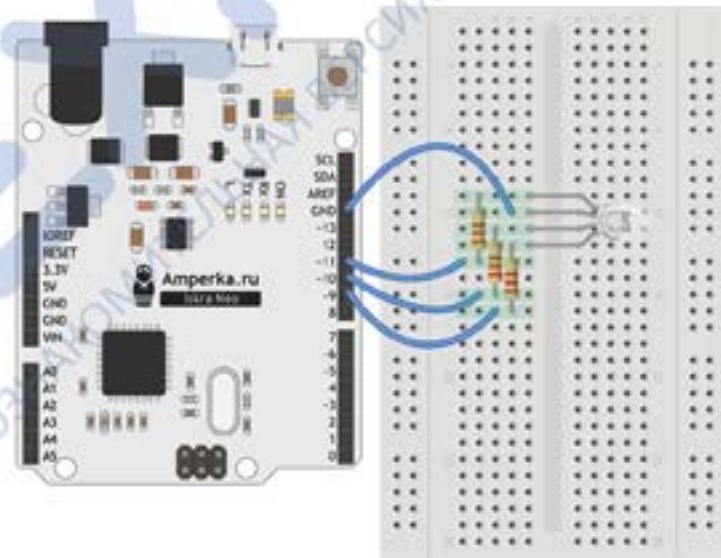


Рис. 9.6: Подключение трёхцветного светодиода на макетной доске

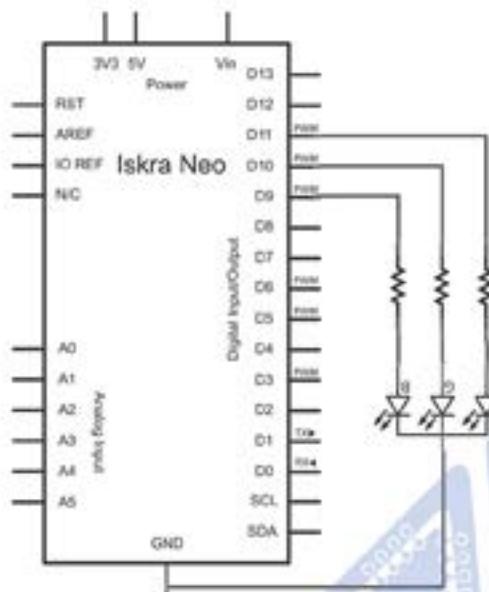


Рис. 9.7: Принципиальная схема подключения трёхцветного светодиода

Итак, давайте сделаем светодиод, переливающийся всеми цветами радуги.

Для этого подключите трёхцветный светодиод, как показано на схемах 9.6 и 9.7.

Загрузите на Iskra Neo следующий код:³⁵

```
int brightness = 255; // текущее значение яркости
int fadeAmount = 5; // скорость затухания
// пины, на которых расположен светодиод
int leds[3] = {9, 10, 11};

// индексы пинов, с которыми мы работаем
// в текущий момент
int curPin[2] = {0, 1};

void setup()
{
    for(int i = 0; i < 3; i++)
        pinMode(leds[i], OUTPUT);
}

void loop()
```

³⁵ File → Examples → Amperka → p09_rainbow

```

{
    // подаём на два разных пина значение равное
    // brightness и 255-brightness
    analogWrite(leds[curPin[0]], brightness);
    analogWrite(leds[curPin[1]], 255 - brightness);

    // уменьшаем значение brightness
    // на скорость затухания
    brightness = brightness - fadeAmount;

    // когда значение brightness становится равным
    // нулю, меняем номера пинов, с которыми мы
    // работаем
    if (brightness == 0) {
        curPin[0] = (curPin[0] + 1) % 3;
        curPin[1] = (curPin[1] + 1) % 3;
        brightness = 255;
    }
    delay(30);
}

```

Как это работает

Итак, в программе мы работаем сразу с тремя пинами. По сути, данный код ничем не отличается от кода, где мы программируем затухание. Только здесь мы управляем тремя светодиодами, которые находятся в одном корпусе и имеют общий катод.

```
int leds[3] = {9, 10, 11};
```

В этом массиве мы задаём номера используемых pinов, к которым подключены аноды светодиода. Следите за тем, чтобы данные пины поддерживали ШИМ-сигналы: они обозначены символом «~».

В каждый момент времени мы управляем двумя светодиодами, которые определяются значениями элементов массива curPin:

```

analogWrite(leds[curPin[0]], brightness);
analogWrite(leds[curPin[1]], 255 - brightness);

```

Здесь мы подаём на один светодиод сигнал, содержащийся в переменной `brightness`, а на другой — обратный сигнал, т.е. `255 - brightness`.

Каждую итерацию мы убавляем переменную `brightness` на скорость затухания.

`brightness = brightness - fadeAmount;`

И, наконец, когда `brightness` становится равным 0, мы снова присваиваем ей значение 255 и меняем номера пинов, с которыми будем работать.

```
if (brightness == 0) {  
    curPin[0] = (curPin[0] + 1) % 3;  
    curPin[1] = (curPin[1] + 1) % 3;  
    brightness = 255;  
}
```

В примере мы встречаемся с новым оператором «%». Это такой же оператор как и `+, -, *, /`, но означает он остаток от деления. Например, `x % 3` в результате даст остаток от деления значения переменной `x` на 3.

Возвращаясь к нашей программе, если у нас затухал красный светодиод и увеличивалось свечение синего, мы переходим к следующей паре пинов, и с этого момента начинает уменьшаться свечение синего светодиода и возрастать свечение зелёного. Чтобы при достижении последнего индекса мы снова переключались на нулевой, а не на следующий, несуществующий индекс, как раз и используется оператор остатка от деления.

Итак, вы познакомились с понятием ШИМ-сигнала, очень важной разновидности сигнала в цифровой электронике. На практике мы использовали его для плавного управления яркостью светодиода.

Плавно управлять можно не только светодиодами, но, как мы уже говорили, и моторами, и другими компонентами. Теперь это не составит для вас труда.



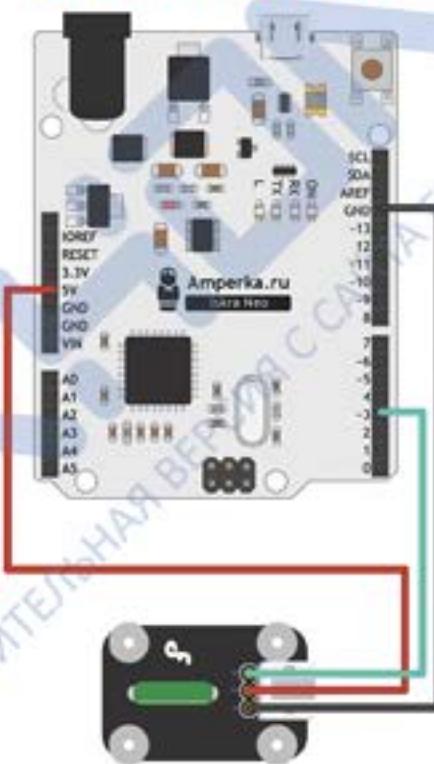
§10. СЕНСОРЫ

10.1 Что такое сенсоры

10.2 Аналоговый и цифровой сигналы

10.3 Как распознать наклон: датчик наклона, `digitalRead`

СЕНСОРЫ



§10



§10. СЕНСОРЫ

10.1 Что такое сенсоры: камера, микрофон, датчик давления, гироскоп, акселерометр

Итак, микроконтроллер умеет изменять состояние своих выводов по заданной программе. Но он бы не был и в половину так интересен, если бы у него не было входов. Без входов его программа знать бы не знала, что происходит во внешнем мире. Поэтому, мы с вами дадим Iskra Neo органы чувств и связь с внешним миром.

То, что у человека называется органами чувств, в электронике называется сенсорами или датчиками. Это синонимы. Сейчас вы сидите или лежите, читаете эту книгу. Вы точно знаете, где и в каком положении находится ваше тело, ваши руки и ноги — это мускульное чувство. Вы знаете, где расположен верх, а где низ — это вестибулярный аппарат. Ваши глаза видят этот текст: книгу, планшет или монитор и все, что рядом — это зрение.

Слух сообщает, что происходит вокруг: с улицы доносится шум города, играет музыка, жужжит вентилятор. Осязание дает вам почувствовать фактуру вашей одежды, текстуру книги, дуновение ветерка, температуру воздуха. Нюх подсказывает, что на кухне есть что-то вкусное.

Современная электроника может повторить все эти сенсоры, хотя и не всегда в таком качестве. Глаза заменяют камеры, а уши — микрофоны (хотя пока компьютерное распознавание образов и звуков не столь хорошо, как у человека, но все очень быстро изменяется). Осязание — датчики давления (тензодатчики) и термометры. Обоняние — чувствительные газоанализаторы. Вестибулярный аппарат — гироскопы и акселерометры.

Некоторые электронные датчики значительно чувствительнее человеческих сенсоров и работают там, где невозможно находиться: к примеру, в доменной печи или в открытом космосе. Они умеют распознать то, на что человек неспособен: радиацию или магнитное поле.

10.2 Как обмениваться информацией: аналоговый и цифровой способы передачи

Чтобы сообщить о показаниях, датчики дают нам какой-то сигнал. Зачастую это простой цифровой сигнал: ответ «Да-Нет». К примеру, датчик затопления отвечает на вопрос: «Есть ли вода между моими контактами?». Напомним, что в электронном мире «Да» и «Нет» — это высокий и низкий уровень напряжения. Для Iskra Neo «Да» — это пять вольт, а «Нет» — это ноль вольт.

Естественно, что не на все вопросы можно ответить «Да» или «Нет». Сколько грамм весит эта книга? Какова температура расплавленной платины? Сколько сахара в чашке чая? На эти вопросы можно дать ответ в единицах по какой-либо шкале: 400 грамм; 3000 градусов по шкале Цельсия; 2 ложки.

Если мы даем ответ в числах, этим числам можно поставить в соответствие другие значения, которые уже могут быть обработаны электрически. К примеру, уровень напряжения или силу тока (рис. 10.1). Если соответствие однозначно: при измерении температуры, например, кружка кипятка и человеческое тело дают разные уровни напряжения сигнала; и воспроизводимо: две кружки кипятка дают один и тот же сигнал; значит, у нас есть прибор-преобразователь, выдающий так называемый аналоговый сигнал. Аналоговый сигнал — непрерывный, не имеющий выделенных уровней. Его тоже может обрабатывать Iskra Neo.

Для этого в Iskra Neo существует специальный механизм, который называется аналого-цифровой преобразователь, или, сокращённо, АЦП. Как уже неоднократно говорилось, ни один процессор не умеет работать с непрерывно меняющимся сигналом. Только ноль и единица, только цифры! Поэтому, «аналоговый» вход — на деле вход АЦП. АЦП — это 1024 так называемых компаратора: элемента, умеющего сравнивать сигнал с другим, известным напряжением и сигнализировать, если входное напряжение больше этого известного, или, как ещё говорят, опорного напряжения.



Рис. 10.1: Пример соответствия температуры в градусах Цельсия и аналогового сигнала в вольтах

Опорное же напряжение создаётся следующим образом: промежуток от нуля до пяти вольт разбивается на 1024 ступени опорного напряжения ($\frac{5}{1024}$ В, $\frac{10}{1024}$ В, ...), и принимаемый сигнал поочерёдно сравнивается с каждой. После этого микроконтроллер знает, что, например, на входе у него не меньше, чем 2 вольта и не больше, чем 2,02 вольта.

Кроме уже описанных выше сложностей с оцифровкой аналогового сигнала, есть ещё несколько соображений, исходя из которых большая часть техники ныне — цифровая. Аналоговый сигнал очень чувствителен к каналу передачи и помехам. Цифровой — не так сильно, его уровни имеют люфт: единицей, к примеру, считается сигнал от 3 до 5,2 Вольт. Аналоговый сигнал трудно сохранить и скопировать. Попробуйте скопировать бабушкину грампластинку, на которой глубиной дорожки зашифрован звук! Без Апрелевского завода не обойдёшься.

Также и с аналоговым сигналом: по мере прохождения его по проводам, к нему примешиваются помехи и шумы окружающей среды. Чем длиннее провод, тем больше помех.

Вышедшие с датчика 3 В, при достижении приёмника, могут превратиться в 2,9 В или 3,1 В, и показания будут уже искажены.

Теперь сравните это с тем, как вы копируете файлы на цифровых носителях. Файл теоретически вечен, а RAID-массивы³⁶ и иные средства резервного копирования приближают эту теорию к практике.

Кстати, о файлах. Любая оцифровка происходит почти так же, как мы описывали работу АЦП. Изображение, звук или напряжение разбиваются на таблицу, и каждому элементу таблицы ставится в соответствие некое число. Изображение — это таблица пикселей и информация о цвете каждого. Звук — это таблица отрывков в миллисекунды и меньше, и информация о том, что надо выдать на динамик. Осциллограмма — это таблица, в первом столбце которой — время измерения, а во втором — измеренное напряжение. Но с этим вы сможете разобраться сами чуть позднее.

Несмотря на недостатки аналогового сигнала, он остаётся крайне важным звеном в электротехнике. Аналоговый сигнал крайне часто является первоисточником, так как в природе нет ничего однозначного: абсолютно белого, абсолютно громкого или абсолютно тёплого. Поэтому, научиться работать с аналоговыми сигналами так же важно, как и с цифровыми. С ними мы познакомимся в одном из следующих параграфов.

Пока же дадим нашему Iskra Neo простой орган чувств — подобие вестибулярного аппарата.

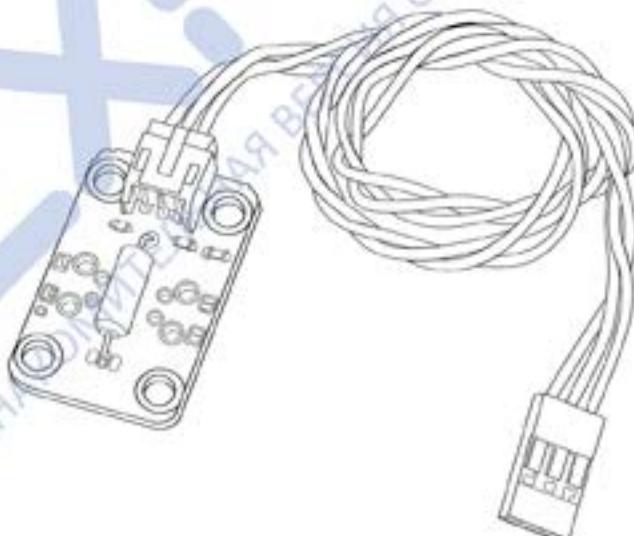


Рис. 10.2: Внешний вид датчика наклона

³⁶ Зеркальный RAID-массив — устройство, в котором несколько жёстких дисков хранят одну и ту же информацию на случай отказа одного из них

10.3 Как распознать наклон: датчик наклона, `digitalRead`

На рисунке 10.2 изображён датчик наклона. Это простой цифровой датчик, умеющий отвечать «Да» или «Нет» на вопрос: «Наклонён ли ты больше, чем на пятнадцать градусов?». Внутри капсулы на датчике катается шарик, замыкающий контакты. Именно благодаря этому явлению мы можем судить об уровне наклона.

У датчика — три вывода: питание, земля и сигнал. Почему нельзя было сделать просто размыкатель, мы поймём в следующем параграфе, когда станем подключать к Iskra Neo кнопку.

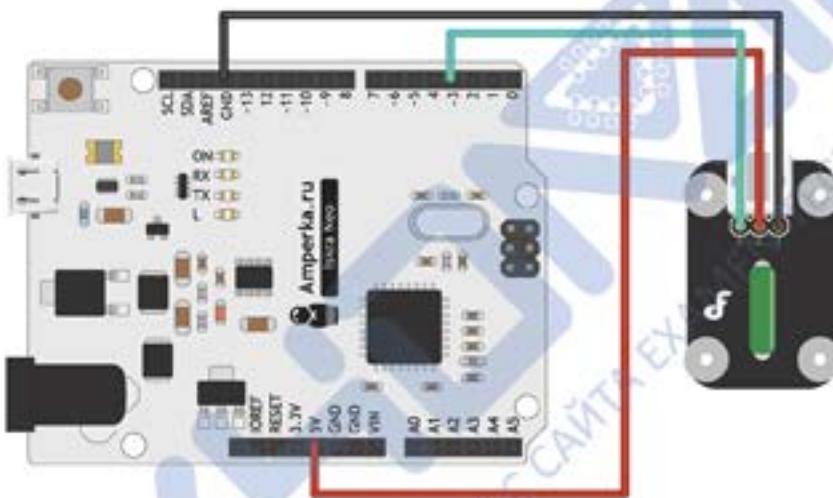


Рис. 10.3: Визуальная схема подключения датчика наклона.

Красный — питание, чёрный — земля, зелёный — сигнал

Сейчас подключите датчик к Iskra Neo так, как показано на рисунке 10.3.

И загрузите такой скетч:³⁷

```
int ledPin = 13; // Пин со светодиодом
int switcher = 3; // Пин с датчиком наклона
boolean tilt; // Наклонён ли датчик?

void setup()
{
    // Инициализируем пин со светодиодом
    // как работающий на вывод
```

³⁷ File → Examples → Amperka → p10_tilt

```
pinMode(ledPin, OUTPUT);

    // Инициализируем пин с датчиком
    // как работающий на ввод
    pinMode(switcher, INPUT);

}

void loop()
{
    // Читаем значение сенсора
    tilt = digitalRead(switcher);

    if (tilt == HIGH) {
        // Включаем светодиод, если датчик наклонён
        digitalWrite(ledPin, HIGH);
    } else {
        // Выключаем, если не наклонён
        digitalWrite(ledPin, LOW);
    }
}
```

Понаклоняйте датчик: светодиод должен загораться, когда датчик наклонён, и выключаться в противном случае.

Как это работает

В приведённом примере мы столкнулись с несколькими новыми командами. Давайте разберёмся, что они означают.

Мы подключили сигнальный провод датчика к 3-му pinu Iskra Neo. Если мы хотим использовать pin для считывания показаний, а не для вывода, мы должны настроить его должным образом.

Это делается командой `pinMode(switcher, INPUT)`.

Вы уже знакомы с встроенной функцией `pinMode`, просто в случае, когда мы хотим считывать данные, в качестве второго аргумента должна передаваться константа `INPUT`, что означает «вход» или «ввод». Первый аргумент — это по-прежнему номер настраиваемого пина.

На самом деле все пины Iskra Neo, если не указано иное, и так по умолчанию настроены, как входы. Поэтому данная строка в нашей процедуре `setup` необязательна. Однако, делая явное определение, вы делаете код более понятным: чётко заявляете о своих намерениях. Это правило хорошего тона.

Далее, в теле процедуры `loop`, мы, первым делом, считываем показания с датчика. Это делается с помощью встроенной функции `digitalRead`. Она принимает лишь один параметр: номер пина, с которого нужно произвести чтение. Функция `digitalRead` возвращает значение `HIGH`, если на входе — цифровой сигнал, соответствующий логической единице (т.е. напряжение более 3 В); и `LOW`, если на входе — логический ноль (т.е. напряжение менее 2 В).

Мы сохраняем результат для дальнейшей работы с ним в переменную `tilt`, которая имеет тип `boolean`. Тип `boolean` — это тип переменных, которые могут принимать лишь два значения: истина или ложь, 1 или 0, «Да» или «Нет». Как раз то, что нам нужно для цифрового сигнала.

Далее, в зависимости от значения `tilt`, с помощью уже знакомой конструкции `if/else` мы принимаем решение о том, нужно ли нам включить или выключить светодиод.

В примере программы код процедуры `loop` излишне подробен: каждую операцию мы проделываем одну за одной. Можно вспомнить о том, что вызовы функций можно делать прямо при передаче аргументов в другие функции. Тогда возвращаемые ими значения будут сразу же переданы в качестве значения аргумента, и нам не придётся использовать дополнительные переменные. Если переписать наш `loop` с учётом этого, получится гораздо более простой код, делающий всё то же самое:

```
void loop()
{
    digitalWrite(ledPin, digitalRead(switcher));
}
```

В данном случае мы пишем на выход со светодиодом то, что только что считали с сенсора: логический ноль или единицу. Тем самым, мы включаем или выключаем светодиод.

Итак, мы познакомились с понятием сенсоров и видами входных сигналов. В следующих параграфах мы углубимся в эту тему.

Так, вы сможете собирать устройства, общающиеся с внешним миром и наделите их звуками искусственного интеллекта.

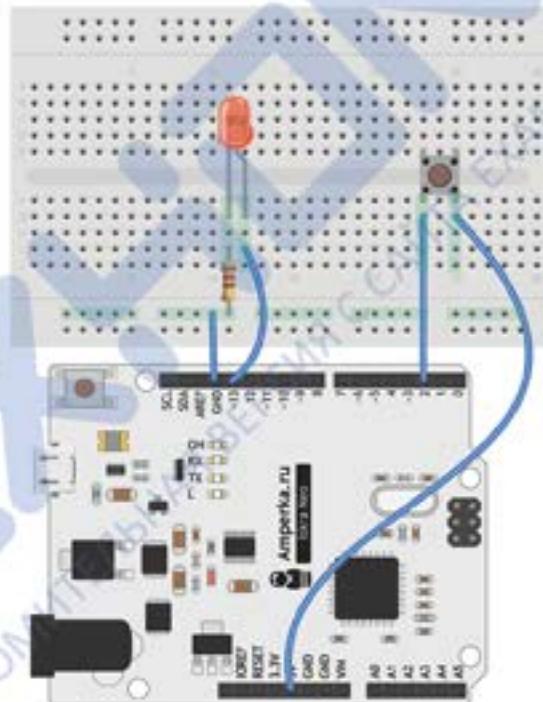
§11. КНОПКА – ДАТЧИК НАЖАТИЯ

- 11.1 Как работает тактовая кнопка
- 11.2 Как при помощи кнопки зажечь светодиод
- 11.3 Как сделать кнопочный выключатель
- 11.4 Шумы, дребезг, стабилизация сигнала кнопки



КНОПКА – ДАТЧИК НАЖАТИЯ

§11



§11. КНОПКА – ДАТЧИК НАЖАТИЯ

11.1 Как работает тактовая кнопка

Теперь мы подключим новый сенсор — кнопку. Это самый простой датчик нажатия. Кнопка имеет два положения: включена и выключена. Типичная тактовая кнопка изображена на рисунке 11.1.

Кнопки настолько распространены и привычны, что может вызвать удивление, что её относят к разряду датчиков. Однако это ни что иное, как компонент, позволяющий микроконтроллеру взаимодействовать с внешним миром, поэтому такая классификация оправдана.

Кнопка — это не только кусок пластмассы на вашей клавиатуре. Поставьте кнопку на направляющую токарного станка с ЧПУ, и она станет его концевым выключателем. Установите кнопку на окно, она станет датчиком открытия. Фантазия подскажет другие применения.

Нам потребуется тактовая кнопка — наподобие той, что стоит на вашей клавиатуре. Она замыкает цепь, когда вы зажимаете её, и возвращается в исходное положение, размыкая цепь, когда вы её отпускаете. У тактовой кнопки есть четыре контакта, соединённые попарно — две противоположные грани при нажатии замыкаются.

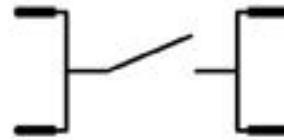
11.2 Как при помощи кнопки зажечь светодиод



(а) Внешний вид кнопки



(б) Обозначение на схемах с одной парой контактов



(с) Обозначение на схемах с двумя парами контактов

Рис. 11.1: Типичная кнопка

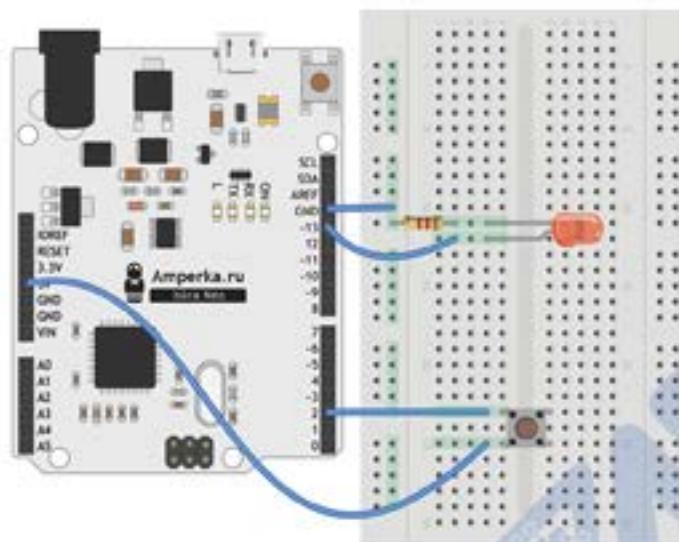


Рис. 11.2: Наивная схема подключения кнопки на макетной плате

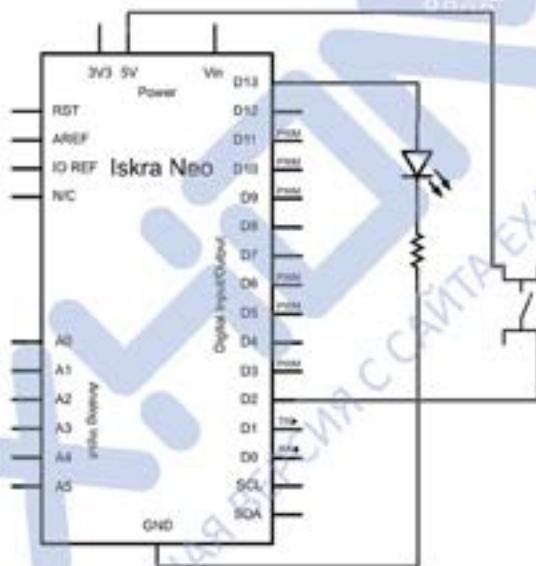


Рис. 11.3: Наивная принципиальная схема подключения кнопки

Соберите схему, как показано на рисунках 11.2 и 11.3.

И загрузите на Iskra Neo следующий код:³⁸

```
int buttonPin = 2; // пин с кнопкой
int ledPin = 13; // пин со светодиодом

// состояние кнопки: зажата/отжата
boolean buttonState = LOW;
```

³⁸ File → Examples → Amperka → p11_push_light

```

void setup()
{
    pinMode(ledPin, OUTPUT);
    // настраиваем пин с кнопкой как вход
    pinMode(buttonPin, INPUT);
}

void loop()
{
    // считываем показания кнопки
    buttonState = digitalRead(buttonPin);
    if (buttonState == HIGH) {
        // если кнопка нажата включаем светодиод
        digitalWrite(ledPin, HIGH);
    } else {
        // если кнопка отжата включаем светодиод
        digitalWrite(ledPin, LOW);
    }
}

```

Ничего не напоминает? Всё верно, это код программы из предыдущего параграфа, где мы экспериментировали с датчиком наклона. Просто на этот раз мы подменили датчик на уровне схемы, микроконтроллер об этом и не догадывается.

Стало быть, вы уже знакомы с функцией `digitalRead`, которая используется для считывания показаний и знаете, как настроить пин в роли входа: `pinMode(buttonPin, INPUT)`.

Теперь, по нажатию кнопки, светодиод начинает гореть ярко. Как только вы её отпустите, он... Ага, он почему-то не гаснет. Где наша ошибка?

Ответьте, сколько вольт на входе `Iskra Neo` после того, как кнопка выключена? Нет, не ноль. Совсем не ноль.

Дело в том, что ножка микроконтроллера, переключенная в режим входа, представляет собой очень, очень большое сопротивление. В нашей любимой схемотехнической аналогии: большая и толстая труба токопроводящих дорожек упирается в трубочку от шприца ножки микроконтроллера. Естественно, когда перекрывается кран, т.е. размыкается кнопка, вода никуда не уходит. Она очень медленно стекает через ногу микроконтроллера.

Токопроводящая дорожка, провод и дорожка макетной платы превращаются в ёмкость, и ёмкость заряженную. Теперь, чтобы освободить этот участок трубы от воды, надо пробить в ней дырку — не слишком большую, чтобы через неё не стекло все и сразу, но достаточную, чтобы отводить излишки воды, когда кран перекрыт. Для этого мы применим стягивающий резистор.

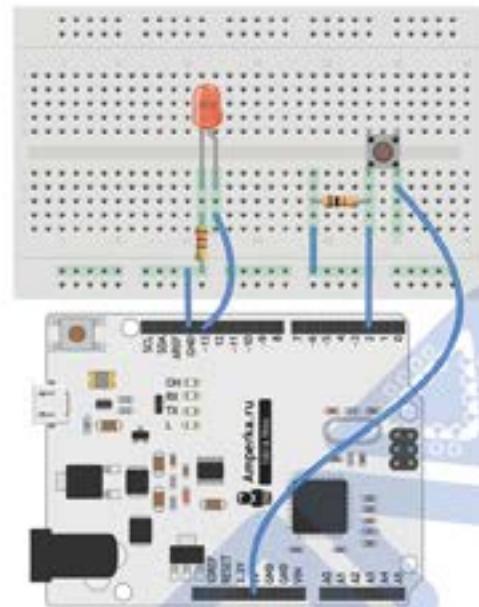


Рис. 11.4: Схема подключения кнопки со стягивающим резистором на макетной плате

Возьмите резистор номиналом 10 кОм и подключите его между землёй и вторым пином, так, как показано на рисунках 11.4 и 11.5.



Рис. 11.5: Принципиальная схема подключения кнопки со стягивающим резистором

Теперь попробуйте снова. Как видите, все работает. Наш резистор исполнил роль того самого сливного отверстия. 10 кОм — достаточно большое сопротивление, чтобы не мешать работе, пока кнопка зажата, но при этом его достаточно для отвода тока при отпущеной кнопке: сопротивление ножки микроконтроллера измеряется мегаомами, поэтому ток предпочтёт более простую дорогу.

Подобные резисторы очень распространены в схемотехнике и называются стягивающими резисторами. Они стягивают излишки тока в землю.

Конечно, микроконтроллер для такой схемы — это явное излишество. Можно было подключить светодиод просто через кнопку. Помните, что многое можно реализовать и без микроконтроллера: на кнопках, резисторах, транзисторах и микросхемах стандартной логики. Читайте книги по радиоэлектронике, это интересно. Но для нашей цели, освоения микроконтроллера, пример очень даже уместен: поняв базовые принципы, вы сможете делать гораздо более сложные и полезные схемы.

11.3 Как сделать кнопочный выключатель

Теперь несколько усложним программу. Пусть первое нажатие кнопки включает светодиод, второе — выключает. Такое использование кнопок очень распространено. Наверняка, так же включается и выключается ваш телевизор, телефон и фотоаппарат. Загрузите в *Iskra Neo* такую программу:³⁹

```
int buttonPin = 2; // пин с кнопкой
int ledPin = 13; // пин со светодиодом

// состояние светодиода
boolean ledState = HIGH;

// предыдущее состояние кнопки
boolean lastButtonState = LOW;

void setup()
{
    pinMode(ledPin, OUTPUT);
    pinMode(buttonPin, INPUT);
}

void loop()
{
```

³⁹ File → Examples → Amperka → p11_toggle_light

```
// считываем показание кнопки
boolean reading = digitalRead(buttonPin);

// Если кнопка зажата, и при этом в предыдущий
// раз она была отжата, значит «клик» произошёл
// именно сейчас. Меняем состояние светодиода.
if (reading && !lastButtonState) {
    ledState = !ledState;
    digitalWrite(ledPin, ledState);
}

// запоминаем состояние кнопки для последующего
// прогона цикла loop
lastButtonState = reading;
}
```

В этом примере вы можете заметить несколько новых вещей. Во-первых, это двойной амперсанд `&&` в условном выражении `if`. Подобно тому, как `||` означает «или», `&&` означает «и». Таким образом, условие будет считаться выполненным если только выполняется и то, что записано перед оператором `&&` и то, что после.

Во-вторых, дважды фигурирует выражение вида `!someState`.

Восклицательный знак перед переменной — это логическое отрицание. Если `someState` имеет значение истины, `HIGH` или `1`; выражение `someState` вернёт ложь, `LOW`, `0`. И наоборот. Обратите внимание, что само значение переменной просто добавлением к ней восклицательного знака изменено не будет: будет просто возвращено временное инвертированное значение. Чтобы сохранить отрицание, новое значение нужно присвоить, что и делается выражением: `ledState = !ledState`.

В-третьих, если раньше в условии выражения `if` мы явно сравнивали наши переменные с `HIGH` или `LOW` операторами `==` или `!=`, то на этот раз какие-либо операторы сравнения отсутствуют. При такой записи за истинность или ложь условия принимается значение самой переменной. Таким образом, условия `if (reading)` и `if`

`(reading == HIGH)` — эквивалентны. Просто первая запись более короткая, элегантная и предпочтительная среди программистов.

Вернёмся к сути схемы. Попереключайте несколько раз и внимательно следите за светодиодом. Заметили? Иногда при размыкании ваша кнопка срабатывает дважды, и в итоге светодиод не переключает своё состояние. И это вовсе не брак кнопки.

11.4 Почему не работает кнопка: шумы, дребезг, стабилизация

Всё правильно. Кнопка замыкается и размыкается мгновенно только по человеческим меркам. Микроконтроллер же способен отследить маленькие искорки, пробегающие между лепестками контактов за те микросекунды, пока один отходит от другого. И, вообще говоря, кнопка срабатывает не дважды, а несколько десятков раз. Просто в зависимости от того, сработала ли она чётное или нечётное количество раз, вы наблюдаете различный конечный эффект. Это называется шумом или дребезгом. На диаграмме 11.6 схематично изображён этот эффект.

Чтобы бороться с дребезгом, можно слегка замедлить микроконтроллер во время подобного переключения. Пусть он повторно проверяет замыкание через сто миллисекунд. И если и через сто миллисекунд цепь разомкнута или замкнута — Iskra Neo сделает соответствующие выводы.

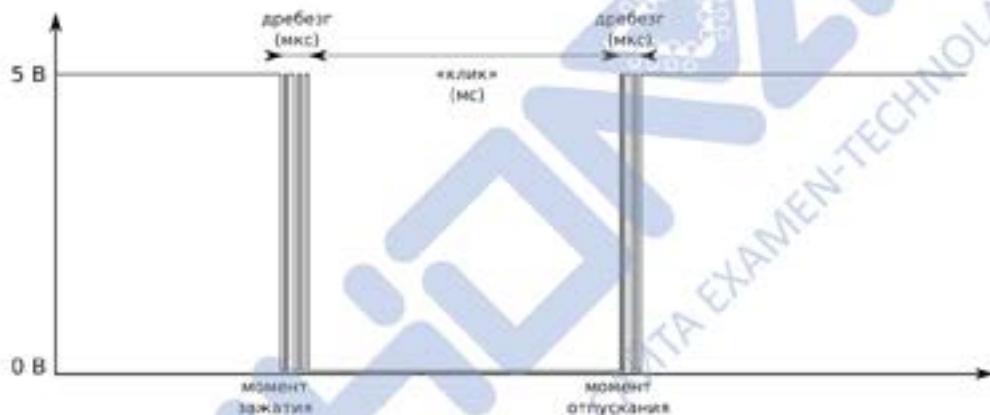


Рис. 11.6: Эффект дребезга механической кнопки

Так и сделаем, добавим в нашу программу повторную проверку после задержки в 100 мс.⁴⁰

```
int buttonPin = 2; // пин с кнопкой
int ledPin = 13; // пин со светодиодом

// состояние светодиода
boolean ledState = HIGH;

// предыдущее состояние кнопки
boolean lastButtonState = LOW;

void setup()
{
```

⁴⁰ File → Examples → Amperka → p11_toggle_light_debounce

```
pinMode(buttonPin, INPUT);
pinMode(ledPin, OUTPUT);
}

void loop()
{
    // считываем показание кнопки
    boolean reading = digitalRead(buttonPin);

    // если состояние кнопки изменилось,
    // ждём 100 мс и проверяем ещё раз
    if (reading != lastButtonState) {
        // ждём
        delay(100);

        // снова считываем показание кнопки
        reading = digitalRead(buttonPin);

        // если текущее состояние снова не
        // равно предыдущему, значит это не шум
        if(reading != lastButtonState){
            // запоминаем новое состояние кнопки
            // для последующего прогона
            loop lastButtonState = reading;

            // если кнопка была-таки нажата, а не
            // отжата, изменяем состояние светодиода
            if (reading) { ledState = !ledState;
                digitalWrite(ledPin, ledState);
            }
        }
    }
}
```

Таким образом, мы просто закрываем глаза на те переходные процессы, которые творятся при нажатии кнопки. Этот трюк называется программной стабилизацией сигнала.

Итак, вы научились бороться с паразитными явлениями электричества с помощью стягивающих резисторов и программной стабилизации сигнала.

Эти знания помогут вам ещё не раз в ваших дальнейших экспериментах.



TECHNOLAB
Ознакомительная версия с сайта EXAMEN-TECHNOLAB.RU

§12. ПЕРЕМЕННЫЕ РЕЗИСТОРЫ

12.1 Как преобразовать сигнал: делитель напряжения

12.2 Как делить напряжение «на ходу»: потенциометр

12.3 Как Iskra Neo видит свет: фоторезистор

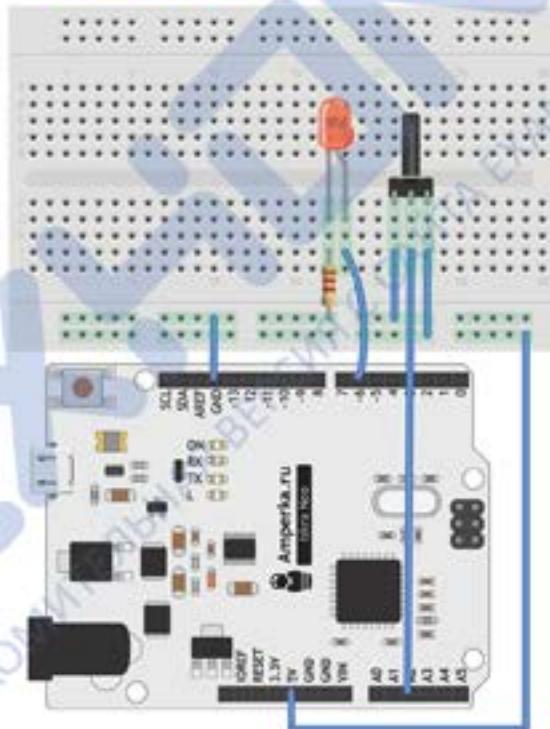
12.4 Как измерить температуру: термистор



ЭКЗАМЕН
ТЕХНОЛАБ

ПЕРЕМЕННЫЕ РЕЗИСТОРЫ

§12



§12. ПЕРЕМЕННЫЕ РЕЗИСТОРЫ

12.1 Как преобразовать сигнал: делитель напряжения

Теперь, когда мы научились работать с цифровыми входами, стоит научиться работать с аналоговыми.

Для того, чтобы воспользоваться аналоговыми входами, нам нужен непрерывно изменяющийся в пределах от нуля до пяти вольт сигнал. Естественно, что именно таких сигналов в природе не очень много. К счастью, слишком большие сигналы можно привести к малым, а слишком слабые — усилить.

Для того, чтобы привести слишком большой сигнал (от десятков вольт до десятков киловольт) к удобоваримому для Iskra Neo виду, воспользуемся простейшим резистивным делителем напряжения. Делителем напряжения называется участок схемы, который собирается из двух резисторов так, как показано на рисунке 12.1.

По закону Киргхофа, ток через резисторы одинаковый, а напряжение, по закону Ома, пропорционально сопротивлению.

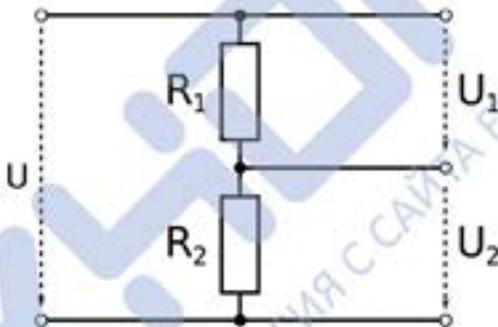


Рис. 12.1: Схема сборки делителя напряжения

$$U_1 = I \times R_1, \quad U_2 = I \times R_2, \quad U_1 + U_2 = U$$

Собрав три уравнения в одно, получим:

$$U_2 = U \frac{R_2}{R_1 + R_2}$$

Соберём для примера резистивный делитель, подключённый к питанию Iskra Neo. Возьмите резисторы номиналами 220 Ом и 1 кОм. Соберите из них схему делителя напряжения так, как показано на рисунке 12.2.

Iskra Neo питается от 5 В, номиналы резисторов нам известны. Подставив эти значения в формулу, получим:

$$U_2 = 5 \text{ В} \cdot \frac{1000 \text{ Ом}}{1000 \text{ Ом} + 220 \text{ Ом}} \approx 4,1 \text{ В}$$

Таким образом, напряжение вокруг резистора R2 составит 4,1 В. Легко показать, что напряжение вокруг резистора R1 в этот момент составит 0,9 В. Проверьте это на практике, воспользовавшись мультиметром. Таким образом, исходное напряжение в 5 В было поделено двумя резисторами в пропорции, зависящей от их сопротивлений. Измерьте напряжения мультиметром, чтобы убедиться в этом на практике.

Особый интерес при работе с Iskra Neo представляет напряжение на нижнем резисторе делителя, на том, что одним концом подключён к земле, т.е. на R2. Другой его конец мы можем подключить к одному из аналоговых входов и, таким образом, получать значение напряжения вокруг него прямо из программы. Помните, мы обсуждали АЦП (или аналогово-цифровой преобразователь) в параграфе 10? Он сравнивает сигнал на аналоговом входе относительно земли с сотнями уже известных ему опорных напряжений, чтобы определить величину этого сигнала.

Если же вы думаете об использовании резистивного делителя для того, чтобы, к примеру, уменьшить яркость лампочки — это крайне неэффективно. Дело в том, что сопротивление нагрузки, т.е. этой лампочки, подключённой параллельно второму резистору, должно быть во много раз больше R2. Потому, что если оно будет сопоставимо R2, через резисторы R1 и R2 потечёт неодинаковый ток: заметная его часть побежит через нагрузку. В итоге, все наши расчёты, сделанные с предположением о равенстве тока I, развалятся.

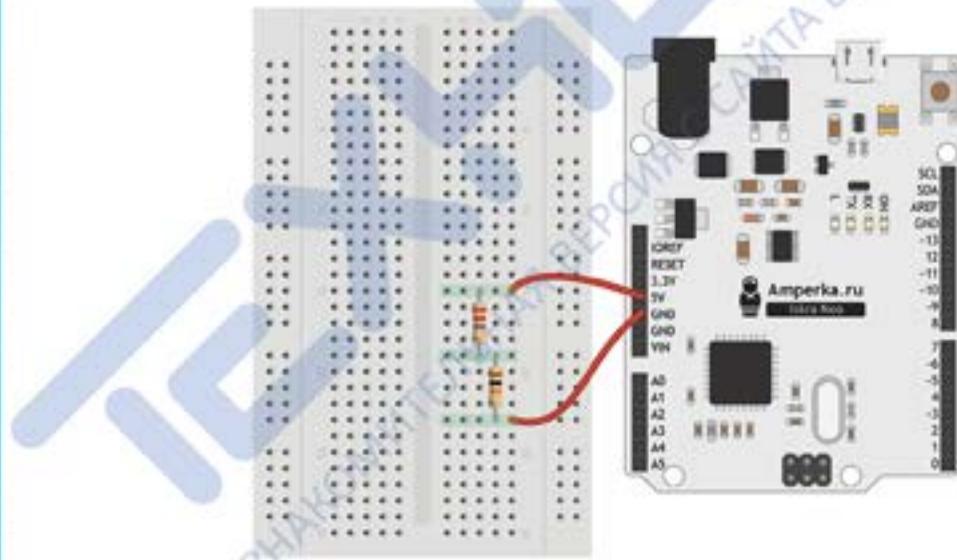


Рис. 12.2: Делитель напряжения с питанием от Iskra Neo на макетной доске

Можно попробовать решить проблему, уменьшив значения R1 и R2 так, чтобы пропорция сохранилась. Например, взять оба резистора с сопротивлениями, меньшими в 100 раз. Тогда, казалось бы, сопротивление нагрузки станет в сравнении с ними относительно большим. Но не всё так просто. В результате уменьшения сопротивле-

ния, через эти резисторы течёт в десять-сто раз больший ток, чем через нагрузку. То есть ток потечёт мимо лампочки попусту. Электроэнергия будет тратиться на бесполезный для нас нагрев резисторов. Не говоря уже о том, что потребуются очень мощные резисторы, чтобы выдержать такой ток.

Поэтому резистивные делители обычно используются для измерений. У всех вольтметров и на входах АЦП платы Iskra Neo очень высокое сопротивление (десятки и сотни мегаом) и, следовательно, очень малый потребляемый ток. Поэтому, даже если через резистивный делитель течёт в 1000 раз больший ток, чем через вольтметр или вход АЦП, он все равно остаётся очень маленьким: миллиамперы и микроамперы.

12.2 Как делить напряжение «на ходу»: потенциометр

Семейство резисторов не ограничивается постоянными резисторами. Существуют элементы, изменяющие своё сопротивление при определённых условиях. При давлении и изгибе — тензорезисторы, при нагреве и охлаждении — термисторы, при освещении — фоторезисторы.

Потенциометр (или переменный резистор) меняет своё сопротивление при изменении положения рукоятки. Типичный потенциометр изображён на рисунке 12.3. Обычно у потенциометра три вывода: два — выводы постоянного резистора, а один — выход с ползунка, скользящего вдоль этого резистора (рис. 12.4).



Рис. 12.3: Типичный потенциометр



Рис. 12.4: Устройство потенциометра

Поворачивая бегунок, мы изменяем длину металлической дорожки между выводами A-W и W-B. Больше длина — больше сопротивление. То есть на самом деле, потенциометр — это уже знакомый нам делитель напряжения, в удобном корпусе и с возможностью поменять отношение R1 к R2.

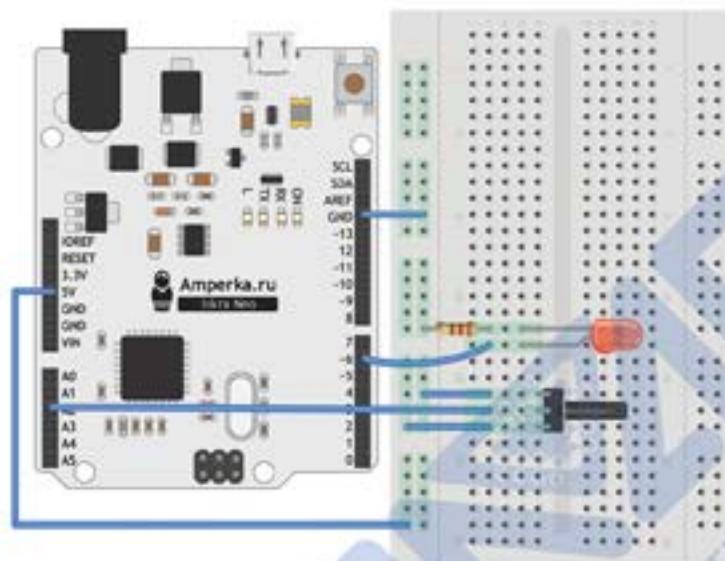


Рис. 12.5: Схема подключения потенциометра на макетной доске

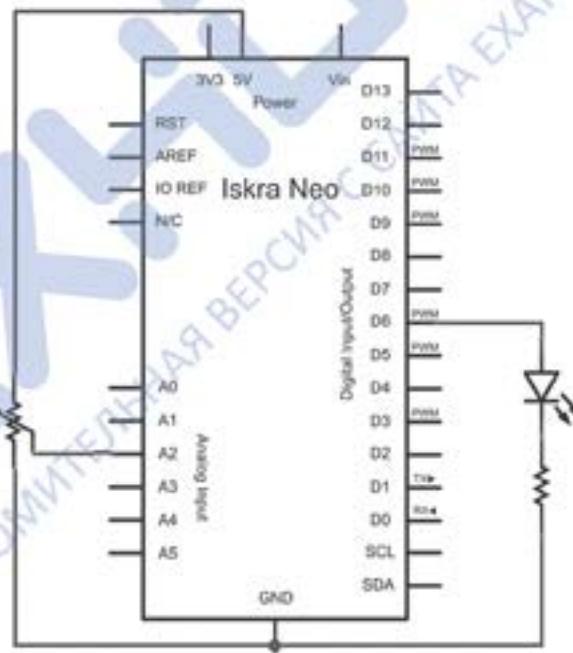


Рис. 12.6: Принципиальная схема подключения потенциометра

Попробуем при помощи потенциометра управлять яркостью светодиода.
Соберите схему на макетной плате как показано на рисунках 12.5 и 12.6.

Загрузите на Iskra Neo следующий код:⁴¹

```

int potPin = A2; // Аналоговый пин с потенциометром
int ledPin = 6; // Цифровой пин со светодиодом
int val = 0; // Значение потенциометра

void setup()
{
    pinMode(ledPin, OUTPUT);
}

void loop()
{
    // Считываем значение потенциометра
    val = analogRead(potPin);
    // Т.к. значение аналогового сигнала варьируется
    // от 0 до 1023, а ШИМ-сигнала - от 0 до 255,
    // поделим полученное значение на 4
    val /= 4;

    // Подаём на светодиод ШИМ-сигнал,
    // пропорциональный сигналу с потенциометра
    analogWrite(ledPin, val);
}

```

В представленной программе вам всё должно быть знакомо, за исключением новой встроенной функции `analogRead`. Она подобна `digitalRead`, но считывает значение с аналогового пина и возвращает не просто единицу или ноль, а целое число в диапазоне от 0 до 1023, соответствующее напряжению на считываемом пине. Так, нулю вольт соответствует значение 0, пяти вольтам соответствует 1023, одному вольту — 204, двум вольтам — 408 и т.д.

Поворачивая бегунок переменного резистора, мы изменяем напряжение на аналоговом входе. Как следствие, меняется яркость светодиода.

⁴¹ File → Examples → Amperka → p12_pot_light

12.3 Как Iskra Neo видит свет: фотодиод

Менять сопротивление резистора можно многими способами. Мы с вами испробуем в деле *фоторезисторы и термисторы*.

У фоторезистора сопротивление изменяется в зависимости от освещённости. Его внешний вид и представление на схеме приведены на рисунке 12.7.

Давайте сделаем «ночную подсветку»: включение светодиода в темноте. Соберите делитель напряжения, заменив одно сопротивление на фоторезистор. Теперь, если фоторезистор освещён, изменяется коэффициент деления, то есть измеряемое напряжение.



Рис. 12.7: Типичный фоторезистор

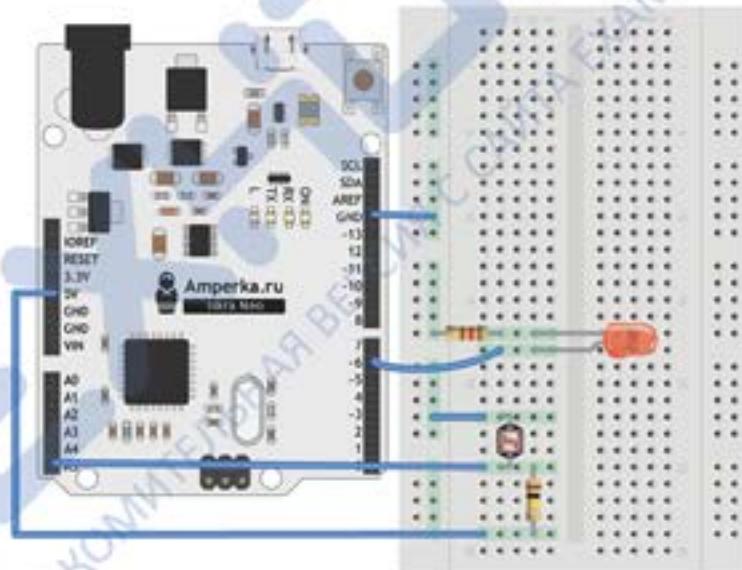


Рис. 12.8: Схема подключения фоторезистора на макетной доске

Соберите схему, как показано на рисунках 12.8 и 12.9. При этом в качестве постоянного сопротивления, которое работает в паре с фоторезистором, возьмите резистор на 100 кОм. Фоторезистор в темноте имеет сопротивление в сотни килоом, а когда освещён — десятки килоом. Нам нужно взять постоянный резистор сопоставимого номинала, чтобы деление было заметным.

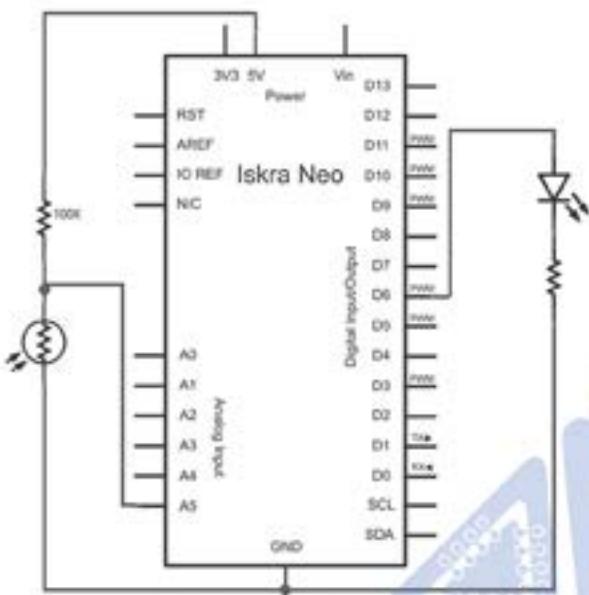


Рис. 12.9: Принципиальная схема подключения фотодиода

Именно поэтому мы взяли 100 кОм.

Загрузите следующий код:⁴²

```
int lightPin = A5; // Аналоговый пин с фотодиодом
int ledPin = 6;    // Цифровой пин со светоиздом
void setup()
{
    pinMode(ledPin, OUTPUT);
}
void loop()
{
    // Считываем показания фотодиода и сравниваем
    // их с пороговым значением, принятым за «темноту»
    // В данном случае, для примера, мы выбрали
    // значение 25
    if (analogRead(lightPin) > 25) {
        // Если освещенность слабая,
        // включаем светоиздом
    }
}
```

⁴² File → Examples → Amperka → p12_ldr_light

```
    digitalWrite(ledPin, HIGH);
} else {
    // В противном случае - выключаем.
    digitalWrite(ledPin, LOW);
}
delay(100);
}
```

Теперь прикройте фотодиод от света ладонью. Светодиод загорелся?
Отлично!

Если в темноте ваша схема начинает мигать — проверьте, возможно свет от дисплея попадает на фотодиод.

Кстати, можете собрать лазерную сигнализацию, известную по фильмам. Напечатайте лазерную указку на фотодиод, и пусть Iskra Neo включает пищалку при прерывании луча. Оставляем это вам в качестве задания для самостоятельного исполнения.

12.4 Как измерить температуру: термистор



Рис. 12.10: Типичный NTC-термистор

Ещё один резистор, с которым мы познакомимся — это *термистор* (рис. 12.10). Поскольку его сопротивление меняется от температуры, его можно использовать в качестве термометра.

Соберите ту же схему, только используйте термистор вместо фотодиода. Для того, чтобы его откалибровать — сопоставить значения сопротивления или напряжения со значениями температуры — можете использовать в качестве опорных точек температуру кастрюли с тающим льдом или снегом (0°C), температуру вашего тела ($36,6^{\circ}\text{C}$) и известную по другому термометру температуру в комнате.

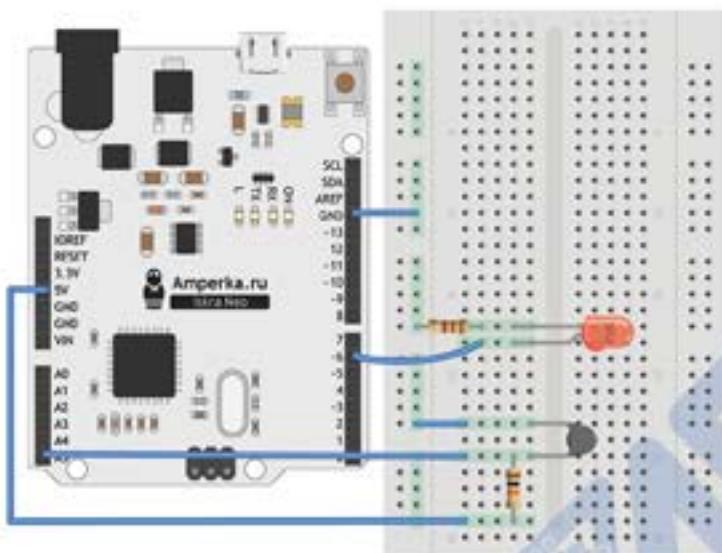


Рис. 12.11: Схема подключения термистора на макетной доске

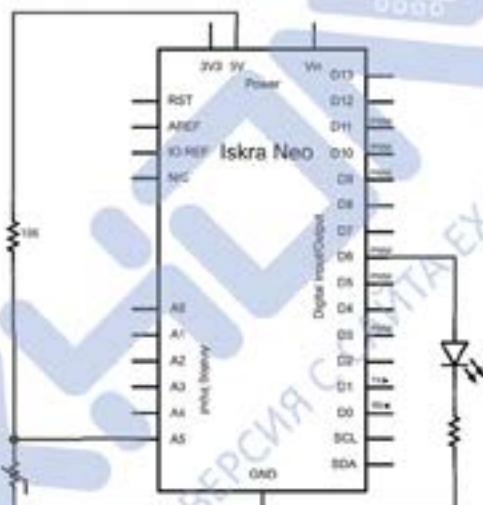


Рис. 12.12: Принципиальная схема подключения термистора

Соберите схему, как показано на рисунках 12.11 и 12.12.

А на Iskra Neo загрузите следующий скетч:⁴³

```
int termPin = A5; // Аналоговый пин с термистором
int ledPin = 6; // Цифровой пин со светодиодом
void setup()
{
pinMode(ledPin, OUTPUT);
}
```

⁴³ File → Examples → Amperka → p12_thermistor_light

```
void loop()
{
    // Считываем показания термистора и сравниваем
    // их с пороговым значением, принятым за «холод»
    if (analogRead(termPin) < 400) {
        // Если температура низкая включаем светодиод
        digitalWrite(ledPin, HIGH);
    } else {
        // В противном случае - выключаем.
        digitalWrite(ledPin, LOW);
    }
}
```

Попробуйте нагреть термистор пальцами рук и увидите, что светодиод выключается.

Итак, вы узнали, что такое делитель напряжения, какие специальные резисторы существуют, и как их подружить с Iskra Neo.

Теперь вы можете получать показания с большого количества существующих в мире датчиков и делать что-то полезное в зависимости от их показаний.



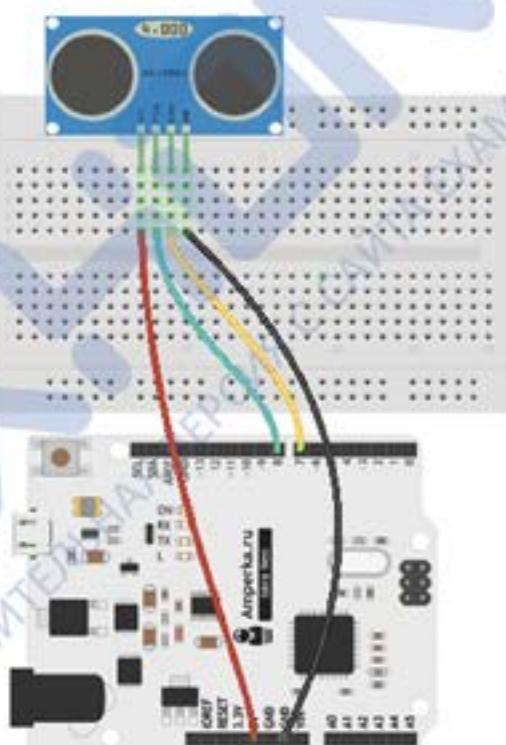
ЭКЗАМЕН
ТЕХНОЛАБ

§13. ДАЛЬНОМЕРЫ

- 13.1 Ультразвуковой дальномер
- 13.2 Характеристики ультразвукового дальномера HC-SR04
- 13.3 Выводы ультразвукового дальномера HC-SR04
- 13.4 Как это работает

ДАЛЬНОМЕРЫ

§13



§13. ДАЛЬНОМЕРЫ

Чтобы робот мог ориентироваться в пространстве с различными объектами, объезжать препятствия, проходить лабиринты, необходимо применить датчики, измеряющие расстояние до предметов, так называемые *дальномеры*.

По принципу действия дальномеры подразделяются на *ультразвуковые* (*ultrasonic range finder*), *оптические инфракрасные* (*IR range finder*) и *лазерные* (*laser range finder*).

Ультразвуковые датчики при работе используют эхолокацию, подобно летучим мышам. Датчик издает импульс на ультразвуковой частоте и замеряет время, за которое отраженный от препятствия сигнал (эхо) вернется на микрофон. Поскольку скорость распространения звука в воздухе известна, то зная время отражения звукового сигнала, можно рассчитать расстояние до препятствия.

Для определения расстояния *оптическими инфракрасными дальномерами* используется метод триангуляции. Луч света излучаемый инфракрасным (IR - infra-red) светодиодом проходит через фокусирующую линзу, отражается от препятствия и попадает на приемник, представляющий собой позиционно-чувствительный фотодиод (PSD - position-sensitive detector). Таким образом луч образует прямоугольный треугольник, причем от расстояния до препятствия зависит угол падения луча на приемник, который фиксируется PSD. Зная угол наклона принятого луча, с помощью тригонометрических выражений для прямоугольного треугольника несложно вычислить расстояние до препятствия.

Благодаря тому, что у оптических дальномеров луч узконаправленный, они могут использоваться для сканирования поверхностей и целых предметов, с более высоким разрешением, чем ультразвуковые.

Лазерные дальномеры используются для измерений больших расстояний. Принцип их работы аналогичен действию ультразвуковых эхолотов, только вместо звука используется отраженный от препятствия лазерный луч. К сожалению, стоимость лазерных дальномеров довольно высока, что ограничивает их применение.

13.1. Ультразвуковой дальномер



Рис. 13.1: Ультразвуковой датчик HC-SR04

Ультразвуковые сенсоры (рис. 13.1) хорошо могут определять расстояние до различных твердотельных предметов, в том числе прозрачных и бликующих. Забавно, но мягкие и пушистые объекты, например, кот или меховая шапка, могут плохо обнаруживаться такими сонарами потому, что поглощают звуковые волны. Также нужно помнить, что эхолокация имеет достаточно большой угол диаграммы направленности (рис. 13.2), что может привести к неоднозначности и, следовательно, к погрешностям измерений.

13.2 Характеристики ультразвукового дальномера HC-SR04

Мы рассмотрим простой в работе ультразвуковой дальномер HCSR04. Из-за простоты работы с ним а также невысокой цены его часто применяют в различных технических проектах.

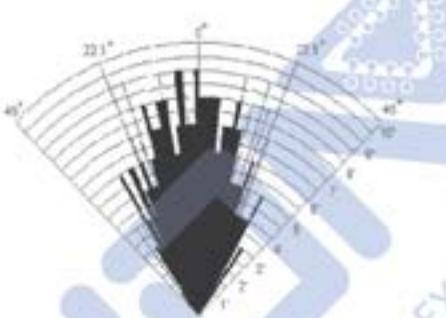


Рис. 13.2: Диаграмма направленности ультразвукового датчика HCSR04

Характеристики дальномера таковы:

- Напряжение питания 5 Вольт
- Потребление в режиме тишины: 2 мА
- Потребление при работе: 15 мА
- Диапазон расстояний: 2–400 см
- Эффективный угол наблюдения: 15°
- Рабочий угол наблюдения: 30°

13.3 Выводы ультразвукового дальномера HC-SR04

- Vcc — положительный контакт питания.
- Trig — цифровой вход. Для запуска измерения необходимо подать на этот вход логическую единицу на 10 мкс. Следующее измерение рекомендуется выполнять не ранее чем через 50 мс.

- Echo — цифровой выход. После завершения измерения, на этот выход дальномером будет подана логическая единица на время, пропорциональное расстоянию до объекта.
- GND — отрицательный контакт питания.

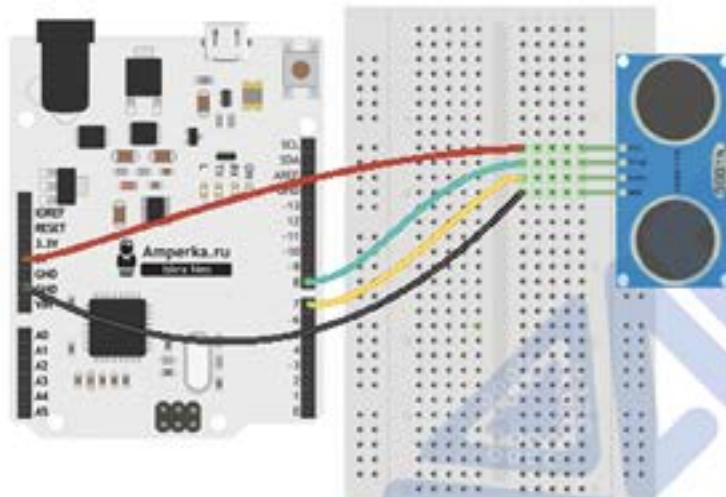


Рис. 13.3: Схема подключения ультразвукового дальномера на макетной доске

После подачи на вход Trig импульса длительностью 10 мкс, сенсор излучает короткий ультразвуковой сигнал с частотой 40кГц и обнаруживает его эхо, которое отражается от объекта (препятствия) и принимается сенсором. Расстояние рассчитывается исходя из времени от подачи сигнала до получения эха и скорости звука в воздухе. Сенсор передает Iskra Neo измеренное расстояние, которое кодируется длительностью электрического сигнала на выходе датчика Echo.

Подключите ультразвуковой дальномер HC-SR04 к Iskra Neo как показано на рисунке 13.3. Расположите провода таким образом, чтобы они не находились перед излучателями дальномера, иначе его показания будут искажены.

Загрузите в Iskra Neo следующий скетч:⁴⁴

```
// Работа с ультразвуковым дальномером HC-SR04
int trigPin = 8; //пин для входа Trig дальномера
int echoPin = 7; //пин для выхода Echo дальномера

unsigned int timeEcho;
unsigned int distance;

void setup()
{
```

⁴⁴ File → Examples → Amperka → p13_ultrasonic_hc_sr04

```

//Инициализируем на выход пин, соединенный
//с портом Trig дальномера
pinMode(trigPin, OUTPUT);

//Инициализируем на вход пин, соединенный
//с портом Echo дальномера
pinMode(echoPin, INPUT);

//Инициализируем последовательный порт
//для работы на скорости 9600 бод
Serial.begin(9600);

}

void loop()
{
    //Подаем сигнал высокого уровня на вход
    //дальномера
    digitalWrite(trigPin, HIGH);

    delayMicroseconds(10); // Ожидание 10 микросекунд

    // Подаем сигнал низкого уровня на вход дальномера
    digitalWrite(trigPin, LOW);

    // Замеряем длину импульса с ожиданием
    // сигнала (таймаут) 40 миллисекунд
    timeEcho = pulseIn(echoPin, HIGH, 40000);

    // Высчитываем дистанцию в сантиметрах
    distance = timeEcho/58;

    // Посыпаем по последовательному порту дистанцию
    Serial.println(distance);
    delay(100); // Ожидание до следующего измерения
}

```

Запустите Serial Monitor, в окне которого вы будете наблюдать значения измерений расстояния от дальномера до объектов. Проведите серию экспериментов, положив у дальномера начало сантиметровой линейки, на которой установите некоторый объект в качестве препятствия. Перемещая объект по линейке, сравните показания дальномера на экране компьютера в окне Serial Monitor с измеренным линейкой расстоянием от дальномера до объекта.

13.4 Как это работает

Работает скетч просто. В функции `setup()` назначаем пины для работы с портами дальномера и инициализируем последовательный интерфейс для работы на скорости 9600 бод. Затем в циклически повторяющейся функции `loop()` для измерения дистанции подаем значение HIGH напорт Trig, ожидаем 10 микросекунд, затем подаем уровень LOW напорт Trig дальномера и начинаем отсчитывать длину импульса в микросекундах командой `pulseIn()`, пока на пине, соединенном с выводом дальномера Echo, уровень LOW не изменится на уровень HIGH, но не более 40 миллисекунд:

```
// Замеряем длину импульса с ожиданием
// сигнала (таймаут) 40 миллисекунд
timeEcho = pulseIn(echoPin, HIGH, 40000);
```

Полученные микросекунды, разделив на 58, переводим в сантиметры:

```
// Высчитываем дистанцию в сантиметрах
distance = timeEcho/58;
```

Полученные в результате данные о расстоянии до препятствия посыпаем по последовательному порту:

```
// Посыпаем по последовательному порту дистанцию
Serial.println(distance);
```

Обязательно добавим ожидание до следующего измерения, чтобы утихло эхо от только что посланного ультразвукового сигнала.

Применяя дальномер при конструировании робота, вы сможете потом запрограммировать робота так, чтобы он избегал столкновений с препятствиями, или, наоборот, находил объекты в окружающем пространстве, для выполнения с ними необходимых манипуляций.

Классическими задачами для начинающих робототехников являются «Кегельбринг», в которой робот должен обнаружить дальномером несколько кеглей и вытолкнуть их за круг, и «Лабиринт», в которой робот по показаниям дальномераходит выход из лабиринта, например, по правилу «правой руки».

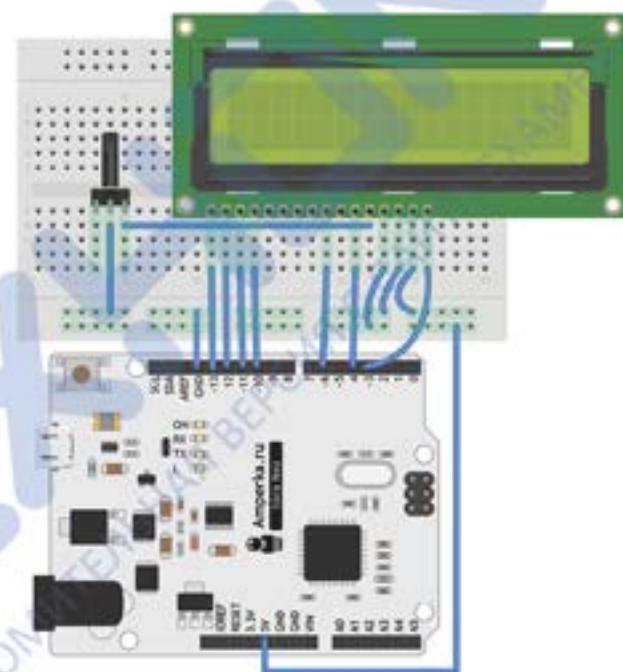
§14. ЖИДКОКРИСТАЛЛИЧЕСКИЕ ЭКРАНЫ

- 14.1 Как работает текстовый дисплей
- 14.2 Как вывести приветствие: библиотека, класс, объект
- 14.3 Как вывести русскую надпись



ЖИДКОКРИСТАЛЛИЧЕСКИЕ ЭКРАНЫ

§14



§14. ЖИДКОКРИСТАЛЛИЧЕСКИЕ ЭКРАНЫ

14.1 Как работает текстовый дисплей

В параграфе 12 мы познакомились с микросхемами и научились выводить на индикаторах различные цифры. Так, мы при помощи всего двух пинов *Iskra Neo*, могли вывести любое число от 0 до 99. Однако мир информации не ограничивается одними только числами. Иногда нужно отображать текст или рисунки. Конечно, в таком случае 7-сегментным индикатором не обойтись. Для таких целей прекрасно подходит **жидкокристаллический дисплей**.

Вместо слова «дисплей» часто говорят «экран»; а вместо «жидкокристаллический» часто используют сокращённое «ЖК» или англоязычное «LCD»⁴⁵.

Так, LCD-экран, ЖК-экран, ЖК-дисплей — всё это синонимы.

Экран текстового дисплея представляет из себя набор прямоугольников, разделённых между собой. Внешний вид типичного такого экрана приведён на рисунке 14.1. Прямоугольники формируют места под отдельные буквы и строки. Каждый такой прямоугольник состоит из маленьких квадратов-пикселей. Это отражено на рисунке 14.2. При помощи зажигания этих пикселей мы и получаем нужный нам символ. А из символов составляется текст. Точно также в предыдущем параграфе мы из сегментов составляли нужную цифру.

Таким образом, на дисплей можно вывести информацию, состоящую из нескольких строк определённой длины.

Существуют экраны различных конфигураций и размеров. Мы будем использовать дисплей, у которого 2 строки по 16 символов в каждой. При этом каждый символ строится в прямоугольнике размером 5x8 пикселей.

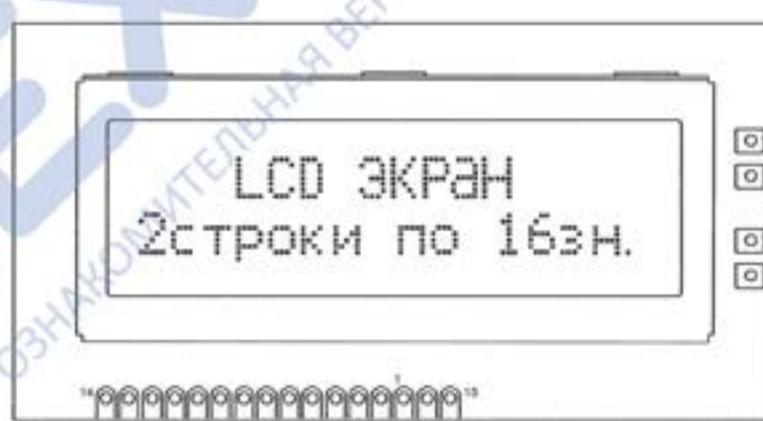


Рис. 14.1: Внешний вид LCD-экрана

⁴⁵ LCD — английская аббревиатура от Liquid Crystal Display: дисплей на жидких кристаллах



Рис. 14.2: Знакосинтезирующие элементы экрана

Пин	Обозначение	Назначение
1	GND (Vs)	Общая земля (0 В)
2	UCC (Vcc)	Напряжение питания (5 В)
3	U ₀ (V ₀)	Управление контрастностью
4	A ₀ (RS)	Адресный сигнал — выбор между передачей данных и команд управления
5	R/W	Выбор режима записи или чтения
6	E	Разрешение обращений к модулю, строб данных
7	DB0	Шина данных в 8-битном режиме, младший бит в 8-битном режиме
8	DB1	Шина данных в 8-битном режиме
9	DB2	Шина данных в 8-битном режиме
10	DB3	Шина данных в 8-битном режиме
11	DB4	Шина данных в 8- и 4-битном режиме, младший бит в 4-битном режиме
12	DB5	Шина данных в 8- и 4-битном режиме
13	DB6	Шина данных в 8- и 4-битном режиме
14	DB7	Шина данных в 8- и 4-битном режиме, старший бит
15	+LED	+ питание подсветки
16	-LED	- питание подсветки

Таблица 14.1: Выводы LCD-экрана

Итак, давайте теперь разберёмся, как же передать информацию на экран.

Если вы посмотрите на дисплей, то увидите 16 выводов. Каждый из этих выводов несёт определённую функцию. Выводы пронумерованы. С одной стороны от них обозначено число 1, а с другой — 16. При этом вовсе не обязательно, что физически они окажутся крайними. Тот вывод, что ближе к единице, считается первым, за ним идёт второй, и так далее, последовательно, до шестнадцатого.

В таблице 14.1 показано за что отвечает каждый из выводов.

GND и UCC — земля и питание (в нашем случае 5 В), без которых не обходится ни одно электронное устройство.

U₀ — управление контрастностью. Его нужно будет подключить к потенциометру.

R/W — выбор режима: чтение или запись. Для наших целей будет достаточно только записи: мы будем только выводить текст на экран, но не считывать его. Поэтому подцепим этот вывод к земле.

DB0–DB7 — выводы, служащие для передачи данных. Именно по этим выводам происходит обмен данными между Iskra Neo и дисплеем. При этом наш дисплей позволяет работать в двух режимах: 8-ми битном — он очень быстрый, но занимает 8 пинов Iskra Neo; и 4-х битном — он медленнее 8-битного, но зато использует только 4 пина. Для наших задач скорость не так важна. Поэтому мы будем использовать 4-х битный режим. И поэтому будем использовать только выводы DB4–DB7.

A0 (или RS) и E — выводы, управляющие передачей данных. Когда на них подаются сигналы, это означает, что мы подготовили данные на выводах DB*, их можно забирать и отображать.

И, наконец, последние два пина +LED и -LED — это питание и земля для фоновой подсветки. Землю мы соединим с землёй Iskra Neo, а питание подключим к 13-му pinу, чтобы можно было управлять подсветкой.

Отображением цифр в параграфе 10 мы управляли самостоятельно. Отдельных сегментов было всего 7, поэтому сделать это было относительно несложно. Но у экрана сотни пикселей и всего несколько выводов для управления ими. Для того, чтобы управление стало возможным, экран должен получать команды по определённому протоколу: в виде определённых импульсов, в определённые моменты, определённой длины, на определённые выводы. Само по себе такое управление довольно сложно. Но мы пойдём на хитрость, чтобы многократно упростить задачу.

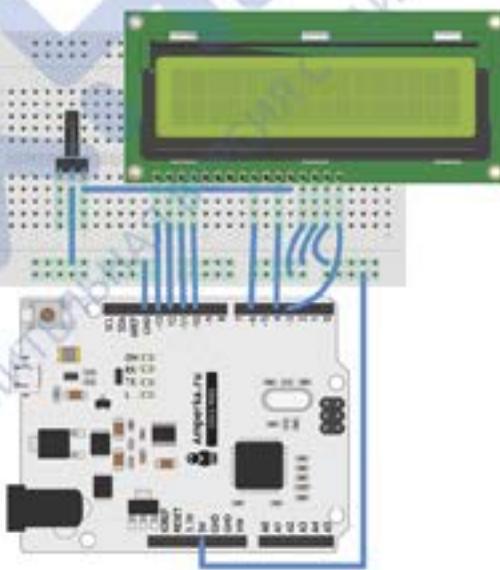


Рис. 14.3: Схема подключения ЖК-экрана на макетной доске

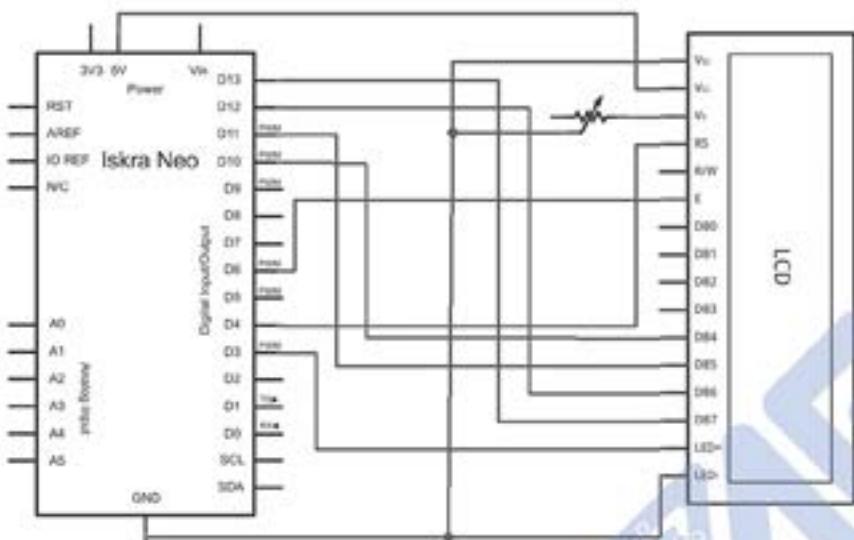


Рис. 14.4: Принципиальная схема подключения ЖК-экрана

14.2 Как вывести приветствие: библиотека, класс, объект

Давайте теперь соберём всё, как показано на схемах 14.3 и 14.4.

Загрузим на Iskra Neo следующий код:⁴⁶

```
// Подключаем библиотеку
#include <LiquidCrystal.h>
// При помощи пина 3 мы будем управлять подсветкой
int backlight = 3;

// Инициализируем пины, по которым будет
// производиться обмен данными с дисплеем
LiquidCrystal lcd(4, 6, 10, 11, 12, 13);

void setup()
{
    // Включаем подсветку
    pinMode(backlight, OUTPUT);
    digitalWrite(backlight, HIGH);
    // Выставляем количество колонок и строк
    lcd.begin(16, 2);
```

⁴⁶ File → Examples → Amperka → p14_lcd_hello

```
// Выводим приветствие
lcd.print("Hello world!");

}

void loop()
{
```

На экране высветится надпись «Hello world!». Поуправляйте потенциометром. Вы можете увидеть, что символы становятся тусклее и ярче в зависимости от положения рукоятки.

Как это работает

В самом начале программы мы видим такую строку:

```
#include <LiquidCrystal.h>
```

Так подключается библиотека.⁴⁷ Давайте выясним, что такое библиотека, и зачем мы её подключаем.

В параграфе 4 мы узнали, что такое процедуры и функции. Это некоторый кусок программного кода, который можно запустить, вызвав его по имени. Мы поняли, что их удобно использовать для того, чтобы избежать повторения логики в одном и том же коде, т.е. чтобы не писать одно и то же много раз. Но что делать, если мы знаем, что будем использовать какие-то части нашего кода в разных программах? Чтобы не копировать все полезные процедуры из одной программы в другую, существуют так называемые библиотеки.

Библиотека – это кусок программного кода, состоящий из процедур, функций и классов (о них мы расскажем ниже), которые можно подключить в любой программе при помощи директивы `#include`.

Итак, мы подключили библиотеку `LiquidCrystal.h`. Эта библиотека нам нужна для удобного управления дисплеем. Подключив её, мы уже можем не беспокоиться о том, какой сигнал, в какой момент и по какому pinу нужно посыпать для отображения нужной нам информации. В этой библиотеке всё уже описано. Нам остается лишь научиться пользоваться этой библиотекой.

Далее в программе идёт строка:

```
LiquidCrystal lcd(4, 6, 10, 11, 12, 13);
```

Таким образом объявляется экземпляр класса⁴⁸ `LiquidCrystal`.

Класс – это ни что иное, как новый тип данных, наподобие `int` или `boolean`, просто чуть более сложный и богатый возможностями. Мы получили возможность ис-

⁴⁷ Подключить библиотеку можно также, перейдя в среде разработки к меню Sketch → Import Library.

⁴⁸ Экземпляр класса также называют объектом.

пользовать класс `LiquidCrystal` в нашей программе благодаря тому, что подключили библиотеку, в которой он описан.

Класс – это составной тип данных, состоящий из полей и методов. Поля – это его переменные, а методы – это его процедуры и функции. Скажем, школьника можно описать в виде класса так:

```
class Школьник {
    char имя[20];
    char фамилия[20];
    void сделатьДомашнееЗадание();
    int ответитьНаУроке();
}
```

Так, мы видим, что в классе `Школьник` есть два поля: имя и фамилия; и два метода: `сделатьДомашнееЗадание` и `ответитьНаУроке`. Чтобы создать отдельный экземпляр класса `Школьник`, нужно его объявить.

```
Школьник иванов;
иванов.имя = "Иван";
иванов.фамилия = "Иванов";
```

Теперь можно заставить Ивана Иванова сделать домашнее задание и ответить на уроке при помощи вызова его методов.

```
иванов.сделатьДомашнееЗадание();
int оценка = иванов.ответитьНаУроке();
```

Естественно, это просто грубый пример. Здесь мы не стали расписывать, что именно должны делать методы, какой код выполнять. В реальности их, конечно же, нужно будет расписать. Написание собственных классов – довольно обширная тема, которая останется за рамками этого учебника. Однако это совершенно не мешает нам использовать уже имеющиеся библиотеки, написанные другими. Поэтому сосредоточимся именно на этом.

В общем виде новый экземпляр (или объект) класса объявляется так:

```
ИмяКласса имяОбъекта(Аргумент1, Аргумент2, ...);
```

Аргументов может и не быть. Это зависит от конкретного класса: необходимо заглянуть в документацию, составленную его автором или непосредственно в код, где он описан.

Например, так мы инициализируем экземпляр класса `LiquidCrystal`:

```
LiquidCrystal lcd(4, 6, 10, 11, 12, 13);
```

В качестве аргументов в этом случае мы должны передать номера пинов, к которым мы подключили используемые выводы дисплея. Порядок аргументов важен

так же, как и при вызове процедур. Что на каком месте должно стоять, опять же, описано в документации.

Теперь мы можем вызывать методы нашего объекта. Что мы и делаем в процедуре `setup`:

```
lcd.begin(16, 2);
lcd.print("Hello world!");
```

Метод `begin` выставляет количество колонок и строк. В нашем случае колонок будет 16, а строк – 2. Это просто подсказка для библиотеки: сам экран не может сообщить информацию о собственном размере. Если вы укажете размер неправильно, программа будет работать не так, как ожидается или не станет работать вовсе.

Далее, метод `print` печатает на экране строку, которая в него передаётся. В данном случае мы печатаем классическую фразу «Hello world!».

Давайте ещё раз взглянем, чем отличается метод класса от обычной функции. В первую очередь тем, что он выполняется только для одного экземпляра. Скажем, у нас есть два экземпляра школьника: Иванов и Петров. Если мы у Петрова вызываем метод `сделатьДомашнееЗадание`, то Иванова это никак не касается: он не будет делать домашнее задание.

Зачем это нужно? Чтобы в одной программе легко управлять сразу несколькими похожими друг на друга вещами. Так, например, мы можем подключить два экрана к различным пинам `Iskra Neo` и управлять ими отдельно.

```
#include <LiquidCrystal.h>
LiquidCrystal lcd1(11, 12, 2, 3, 4, 5);
LiquidCrystal lcd2(13, 10, 9, 8, 7, 6);
void setup()
{
    lcd1.begin(16, 2);
    lcd2.begin(16, 2);
    lcd1.print("I'm LCD 1");
    lcd2.print("I'm LCD 2");
}
void loop()
```

В таком случае на первом экране будет надпись «I'm LCD 1», а на втором — «I'm LCD 2».

14.3 Как вывести русскую надпись на дисплее: кодировка, UTF, cp1251, кириллица

Любой символ цифровое устройство (в том числе и компьютер) воспринимает в виде нулей и единиц, т.е. в виде некоторого кода. С кодами английских букв мы уже сталкивались в параграфе 5. Их код почти везде одинаков. Но языков в мире очень много: русские буквы, арабская вязь, китайские иероглифы. Как закодировать их все? Для этого было придумано множество разных вариантов. Эти варианты называют **кодировками**.

Набор русских букв называют **кириллицей**. Буквы кириллицы кодируются по-разному, в зависимости от кодировки. Одни из самых распространённых кодировок, содержащих кириллические символы – это **cp1251** (она же **windows-1251**), **UTF-8** и **UTF-16**.

Так, например, если перевести нули и единицы в шестнадцатеричную систему счисления, заглавная буква «А» в cp1251 кодируется одним байтом C0, а в UTF-16 кодируется при помощи двух последовательных байт 04 10.

Библиотека **LiquidCrystal** не поддерживает русские буквы. Да и далеко не все дисплеи их поддерживают. Поэтому, если вы напишете, **lcd.print("Здравствуй, мир")**, то получите не то, что ожидали.

Дисплеи от МЭЛТ, с которыми вы работаете, умеют отображать кириллицу. Но, чтобы её отобразить, нужно написать код соответствующего символа.

Давайте русифицируем нашу программу и добавим немного динамики.

Загрузите на **Iskra Neo** следующий код:⁴⁹

```
#include <LiquidCrystal.h>
int backlight = 3;

LiquidCrystal lcd(4, 6, 10, 11, 12, 13);
void setup()
{
    pinMode(backlight, OUTPUT);
    digitalWrite(backlight, HIGH);

    lcd.begin(16, 2);
    // Печатаем сообщение "Здравствуй, мир!"
    lcd.print("\xa4\xe3" "pa\xb3" "c\xbf\xb3\x79\xb9, " " \xb3\xb8" "p!");
}

}
```

⁴⁹ File → Examples → Amperka → p14_lcd_ru

```
void loop()
{
    // Переходим на вторую строку
    lcd.setCursor(0, 1);

    // Пишем количество секунд
    lcd.print(millis() / 1000);

    delay(1000);
}
```

Как это работает

Нас интересует безумная на первый взгляд строка:

```
lcd.print("\xA4\xE3""pa\xB3""c\xBF\xB3\x79\xB9, " +
"\xBC\xB8""p!");
```

Если вы посмотрите на таблицу символов 14.5, то увидите, что, скажем, большая буква «З» кодируется, как «A4», именно она у нас и стоит первым символом в методе print. Специальная последовательность «\x» означает, что следующие два символа будут кодом одного байта.

Такие буквы, как «р», «а» и «с» выглядят одинаково как на русском, так и на английском языке. Поэтому мы, чтобы не искать их код, просто пишем их английские аналоги. Но важно помнить, что в одних кавычках нельзя после последовательности кодированных байтов писать английские символы. Поэтому, мы пользуемся тем фактом, что строки написанные друг за другом, «склеиваются» в одну на стадии компиляции.

Так, мы можем написать:

```
lcd.print("M\xB8""p!")
```

но не можем использовать:

```
lcd.print("M\xB8p!")
```

При попытке скомпилировать такую программу будет выдана ошибка.

Старшая цифра кода символа (в шестнадцатеричном виде)															
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x	...	0	@	P	`	R	...	+	Б	Ю	Ч	.	Д	Ч	
1x	!!	!	1	А	Q	а	ҟ	!	І	Е	Г	Я	ш	І	Ц
2x	÷	"	2	В	R	в	и	и	•	Ё	б	ъ	и	Щ	И
3x	→	#	3	С	S	c	s	III	◊	Ж	ы	и	д	Ч	
4x	←	\$	4	D	T	d	t	►	✓	З	г	ъ	Ф	І	
5x	\	%	5	E	U	e	и	І	і	Н	ё	э	х	ц	Ч
6x	р	&	6	F	U	f	v	▼	1	И	ж	ю	щ	и	
7x	Н	'	7	G	W	э	w	▲	2	Л	з	я	I	'	І
8x	ъ	0	(8	H	X	и	х	R	З	П	и	«	І	І
9x	μ	0)	9	I	Y	і	у	т	°	Ч	й	»	↑	~
Ax	ÿ	≤	*	:	J	Z	j	z	-	€	Ф	к	«	↓	é
Bx	10	≥	+	;	K	С	K	и	{	и	Ч	л	”	и	‡
Cx	ë	Г	,	<	L	Ф	1	и	2	и	Ш	М	и	ї	К
Dx	ї	¥	-	=	M	Э	m	и	5	и	Ђ	ъ	и	и	з
Ex	€	=	.	>	N	^	n	и	и	и	и	и	и	и	и
Fx	€	₩	/	?0	_o	и	и	и	и	и	и	и	и	и	и

Младшая цифра кода символа (в шестнадцатеричном виде)

Рис. 14.5: Кодировка символов в дисплеях МЭЛТ

Далее, давайте взглянем на нашу процедуру `loop`. Мы использовали новый метод `setCursor`. Он переводит невидимый курсор в новое положение, чтобы затем вывод начался именно с этой позиции. Метод принимает 2 параметра: номер квадрата знакосинтезатора по горизонтали и номер по вертикали. Нумерация начинается с нуля. Поэтому `lcd.setCursor(0, 1)` переводит курсор в начало второй строки.

Таким образом, мы сохраняем нашу надпись «Здравствуй, мир» на первой строке, но затем каждую секунду обновляем текст на второй строке, выводя в ней количество секунд, прошедших с момента старта `Iskra Neo`.

Итак, мы познакомились с ЖК-экранами, классом для управления ими и способами представления строковой информации. Теперь возможности общения `Iskra Neo` с внешним миром стали значительно богаче. Используйте их!

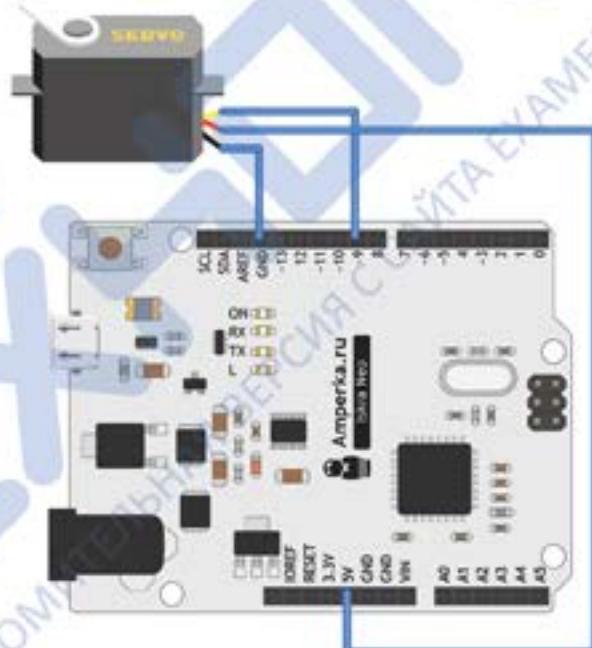
§15. ДИГАТЕЛИ

- 15.1 Разновидности двигателей:
постоянные, шаговые, серво
15.2 Как управлять серводвигателем с
Iskra Neo



ДИГАТЕЛИ

§15



§15. ДВИГАТЕЛИ

15.1 Как заставить предметы двигаться: постоянные двигатели, шаговые, серводвигатели

Итак, мы уже научились зажигать светодиод, воспроизводить звук, пользоваться индикатором, выводить текст на дисплей, общаться с компьютером. Однако мы так и не дошли до главного: как научить *Iskra Neo* двигать предметы.

Для этого нам потребуется нечто, умеющее преобразовывать электрический ток — от сети или от батарей — в движение. Это так называемые *актуаторы*: *двигатели* и *сolenоиды*. Поговорим о *двигателях*.

В магнитном поле виток провода, по которому течёт ток, стремится повернуться, как стрелка компаса. Чтобы он, повернувшись, продолжил вращаться, можно изменять направление тока в контуре.

Так устроены *коллекторные двигатели постоянного тока* (рис. 15.1). Коллектор — это как раз узел, в котором каждые пол оборота меняется направление тока. Коллекторники очень удобны в управлении: подай на них постоянный ток, и они начнут крутиться. Напряжением можно менять скорость вращения. Существуют как крохотные модельные двигатели, размером с фалангу мизинца, так и большие, толкающие электровозы. Их недостаток — щётки: контакты коллектора, передающие ток во вращающийся виток. Они за несколько лет могут истереться и сломаться. Второй недостаток, важный в промышленности: им требуется выпрямитель, делающий из переменного тока в розетке нужный им постоянный. Однако коллекторные двигатели — самые дешёвые, поэтому они чрезвычайно распространены в хобби-электронике, прототипах и несложных устройствах.

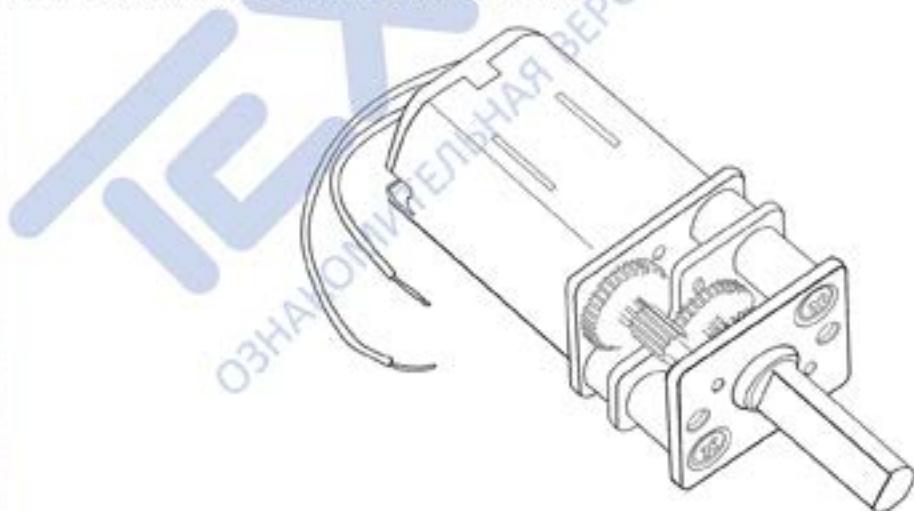


Рис. 15.1: Внешний вид коллекторного двигателя постоянного тока

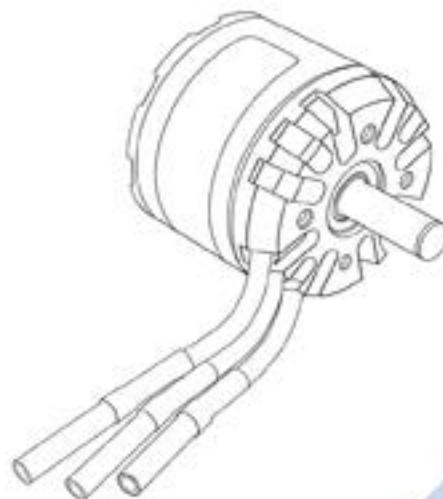


Рис. 15.2: Внешний вид бесколлекторного двигателя

От перечисленных недостатков избавлен двигатель переменного тока. Ток, крутящий его, уже изменяется сам по себе: он приходит из распределительной сети. В таком случае коллектор не нужен. Поэтому такие двигатели называются бесколлекторными (рис. 15.2). Они просты, технологичны, экономичны и долговечны. Но, выиграв в стоимости, мы потеряли в управляемости. Пока двигатель работает с постоянной скоростью, всё прекрасно. Но, если мы захотим регулировать скорость его вращения, возникают проблемы. Если мы станем понижать напряжение на нем, мы потеряли не только в скорости, но и в мощности. Да и понизить напряжение для сильных токов, в десятки ампер, не так-то просто. Можно изменить частоту вращения двигателя, изменив частоту питающего его тока. Естественно, никто для нас не станет тормозить или ускорять вращение генераторов на АЭС, ГЭС и ТЭС. Поэтому нам придётся выпрямить ток, и затем из постоянного сделать переменный ток нужной частоты. Частотные преобразователи, способные на это — довольно сложная и дорогая электроника.

Однако бесколлекторные двигатели с редкоземельными магнитами — самые лёгкие и мощные. Поэтому приходится мириться с усложнением электронной схемы. Такие двигатели можно использовать для вращения винтов вертолёта, колёс радиоуправляемой машинки или в качестве основы для вентилятора.

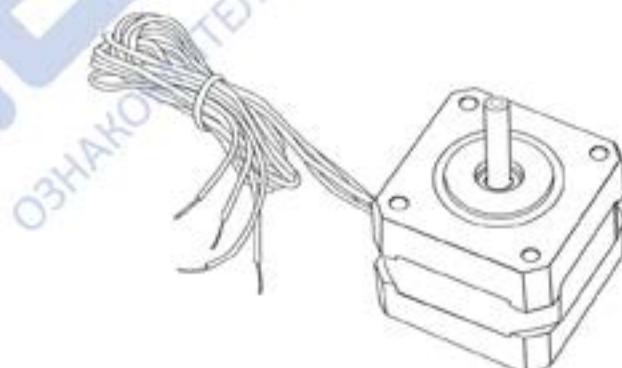


Рис. 15.3: Внешний вид шагового двигателя

Еще одна разновидность двигателей, которые могут нам пригодиться — **шаговые** (рис. 15.3). Шаговые двигатели умеют по команде сделать шаг, т.е. повернуться на заранее заданный угол. Величина шага (угла поворота) зависит от двигателя. Скажем, если величина шага равна 24 градусам, то двигатель делает полный оборот за 15 шагов. Такие двигатели необходимо использовать в тех случаях, если вы хотите контролировать величину передвижения. Например, если вы поставите шаговые двигатели на машинку, то вам не нужно будет замерять скорость и время их работы, чтобы узнать, сколько она проехала. Вам достаточно будет посчитать, сколько раз вы подали команду шаговому двигателю.

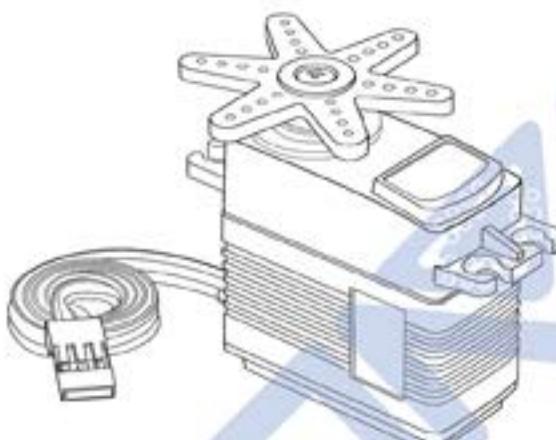


Рис. 15.4: Внешний вид сервопривода

Последний вид двигателей, с которым мы с вами познакомимся — это **серводвигатели**⁵⁰ (рис. 15.4). Серводвигатели могут поворачиваться на заданный угол и удерживать его. Если мы захотим воспроизвести механическую руку, то нам не обойтись без них. Каждый сустав представляет из себя серводвигатель. Так, например, если мы захотим повернуть свою руку на 90 градусов и удерживать в таком положении, мы без труда сможем это сделать. Серводвигатель позволит сделать то же самое и механической руке. Механическая рука — сложный манипулятор, но для решения более простых задач сервоприводы часто подходят также как нельзя кстати. Примера ради: в радиоуправляемых авиамоделях именно сервы наклоняют винты вертолёта и отклоняют элероны самолёта; а в радиоуправляемых машинах именно они отвечают за поворот передних колёс.

15.2 Как управлять серводвигателем с Iskra Neo: Servo, attach, write

Поскольку выходной ток пинов Iskra Neo не должен быть больше 40 миллиампер, а токи двигателя — сотни и тысячи миллиампер, нужна промежуточная плата, которая будет управлять двигателем по сигналам с контроллера. Эта плата называется **драйвером** двигателя.

⁵⁰ Серводвигатель также называют сервоприводом, сервомотором или сервомашинкой

Обычно для коллекторных, бесколлекторных и шаговых двигателей драйвер нужно покупать или делать самостоятельно. А в сервомашинке уже интегрирована вся необходимая электроника. Поэтому с ней, для примера, и поработаем.

Взгляните на сервомотор. У него — три вывода. Один из них коричневый, это земля. Его надо соединить с землёй Iskra Neo. Второй — красный, это питание. На него надо подать напряжение, указанное в инструкции к сервомотору.

Обычные сервомоторы для хобби рассчитаны на 4,8 — 6 вольт. Поэтому питание от Iskra Neo в 5 вольт как раз подойдёт. Третий провод — жёлтый, это сигнал, и его надо подключить к управляющему выводу.

Посмотрите на схему 15.5 и сделайте аналогично.

Загрузите на Iskra Neo следующий код:⁵¹

```
// Подключаем библиотеку Servo.h
// для работы с серводвигателями
#include <Servo.h>

// Создаём объект класса Servo
// из библиотеки Servo.h
Servo myservo;

// Вспомогательная переменная с
// текущим углом
int pos = 0;

void setup()
{
    // Обозначаем, что серводвигатель
    // подсоединен к 9-му pinu
    myservo.attach(9);
}

void loop()
{
    // Пробегаем все значения от 0 до 180 градусов
    // по одному градусу на каждом шаге
    for(pos = 0; pos < 180; pos += 1) {
```

⁵¹ File → Examples → Amperka → p15_servo_sweep

```

// Даём команду серво повернуться на
// заданный угол
myservo.write(pos);

// Ждём 15 мс, пока серво повернётся на °1.
// Время поворота зависит от модели сервы,
// 15 мс -- просто подобранная величина
delay(15);

}

// Делаем то же, что и в предыдущем цикле,
// но в обратном порядке
for(pos = 180; pos>=1; pos-=1) {
myservo.write(pos);
delay(15);
}
}

```

Серводвигатель должен начать крутиться от 0° до 180°. Это два крайних положения типичного сервопривода.

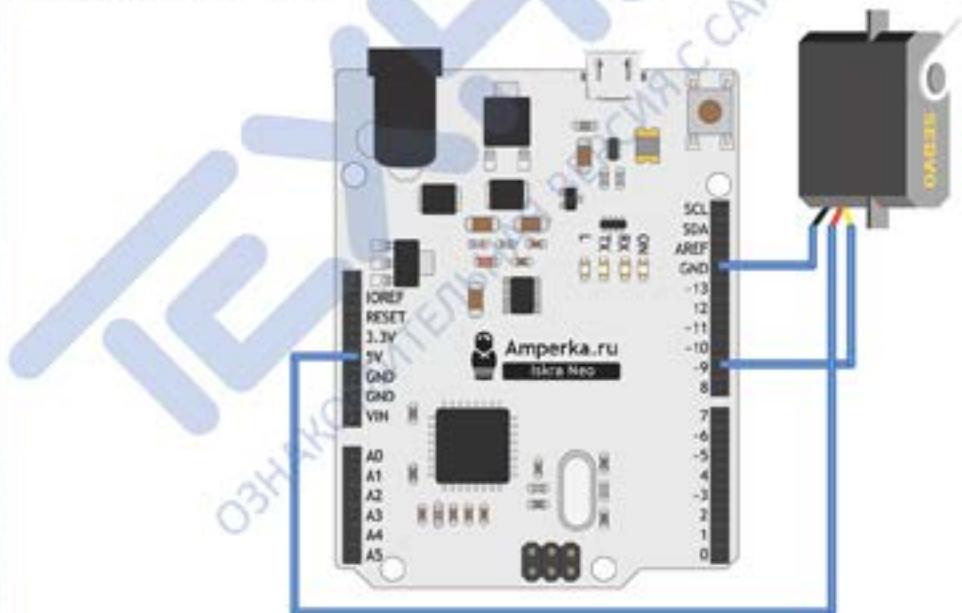


Рис. 15.5: Подключение сервопривода к Iskra Neo

Как это работает

Мы подключаем библиотеку для работы с сервоприводом.

```
#include <Servo.h>
```

Сервопривод управляет довольно сложными сигналами, но, к счастью, подобно тому, как это было с ЖК-экраном, для нас уже написали готовую библиотеку, решающую эту проблему. Нам остаётся лишь правильно ей воспользоваться.

Далее мы инициализируем объект myservo класса Servo.

```
Servo myservo;
```

Весьма похоже на то, что мы делали с дисплеем. Не так ли?

Далее в нашей процедуре `setup`, мы вызываем метод `attach`. Он назначает пин Iskra Neo, по которому будут даваться команды для серводвигателя. Именно к нему должен быть подключен сигнальный провод сервы.

```
myservo.attach(9);
```

Далее, метод `write` поворачивает серводвигатель на заданный угол. Он принимает значения от 0 до 180 градусов. Например, если написать `myservo.write(90)`, то серводвигатель повернётся на угол 90° с той скоростью, с которой ему позволяет питание.

Итак, наша программа, начиная от ноля градусов, каждые 15 миллисекунд прибавляет по одному градусу к текущему значению угла поворота сервопривода. Дойдя до 180 градусов, серводвигатель начинает крутиться в обратную сторону, от 180° до 0.

Стоит прикрепить к качельке сервопривода щётку, и у нас есть готовый «дворник»!

Мы познакомились с разными видами приводов и научились работать с сервоприводом: самым умным мотором семейства. Одного этого достаточно для постройки многих любопытных устройств. Но дальше — больше: в следующих параграфах мы научимся использовать двигатели для перемещения мобильных платформ на колёсах и вплотную приблизимся к созданию мобильного робота.



§16. ТРАНЗИСТОРЫ

16.1 Как управлять электричеством:
транзистор

16.2 Разновидности транзисторов

16.3 Как вращать двигатель

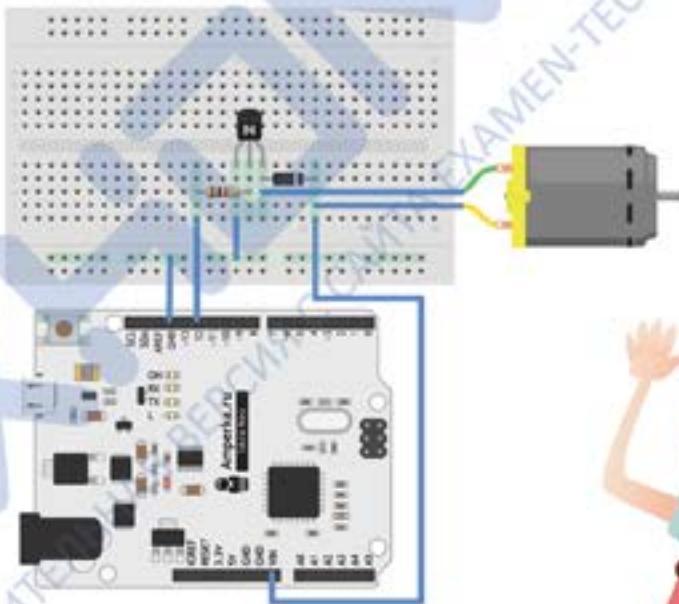
16.4 Как управлять скоростью двигателя



ЭКЗАМЕН
ТЕХНОЛАБ

§16

ТРАНЗИСТОРЫ



§16. ТРАНЗИСТОРЫ

16.1 Как управлять электричеством: транзистор

В этом параграфе мы с вами разберёмся, как с помощью Iskra Neo управлять чем-то более серьёзным, чем один светодиод: лампами, двигателями, клапанами.

Для этого давайте познакомимся с новым элементом управления — *транзистором*. Транзисторы — главный элемент в микрэлектронике. На них построены и микроконтроллеры с процессорами, и драйверы электродвигателей трамваев, и музыкальные усилители, и радиопередатчики.

Что нам даёт транзистор? Он позволяет управлять протекающим через него током. Это кран, который можно открыть и пропустить больше тока; или закрыть совсем и не пускать ни капли. Прелесть его в том, что небольшим электрическим воздействием мы можем управлять большим током. Точно так же, как вы лёгким поворотом крана можете управлять напором воды с давлением в несколько атмосфер.

В цифровой технике чаще всего используется ключевой режим работы транзистора: кран либо открывается полностью, либо полностью закрывается, промежуточные положения игнорируются. Мы тоже ограничимся ключевым режимом и получим управляемый электрическими сигналами выключатель.

У транзисторов есть 3 ножки. Одна из них — кран: подавая или убирая сигнал на ней, мы разрешаем или запрещаем течение тока между двумя другими.



Рис. 16.1: Биполярный транзистор

16.2 Какие бывают транзисторы: полевые, биполярные, MOSFET

Транзисторы делятся на два вида — полевые и биполярные. Полную теорию и принципы действия транзистора мы здесь приводить не станем: с ней вы познакомитесь в институте или по книгам. Ограничимся несколькими практическими советами, достаточными для использования транзистора по назначению.

Биполярный транзистор (рис. 16.1) — это токовый прибор, как и светодиод. Он управляется протекающим через него током. Поэтому обязателен резистор, ограничивающий этот ток до безопасных пределов. Три вывода биполярного транзистора называются базой, коллектором и эмиттером. База — это «кран», в коллектор ток втекает, из эмиттера — вытекает.

Как выбрать подходящий для ваших целей транзистор? В описании транзистора найдите предельный ток через транзистор (I_{max}) и коэффициент усиления (h_{fe}). Разделите первое число на второе. Такой минимальный ток нужно подать на базу, чтобы наш транзистор открылся полностью. Если ток оказался больше 40 мА — ищите другой транзистор, подбирайте составной, делайте каскад транзисторов.



Рис. 16.2: Полевой транзистор

Iskra Neo, не способен выдавать ток больше 40 мА с управляющего пина. Допустим, управляющий ток — 10 мА. В таком случае, сопротивление резистора можно рассчитать по закону Ома:

$$R = \frac{U}{I} = \frac{5\text{ В}}{10 \times 10^{-3}\text{ А}} = 500\text{ Ом}$$

То есть нам необходимо в идеале взять резистор на 500 Ом. Если сопротивление будет больше, транзистор не откроется до конца и не будет пропускать ток между коллектором и эмиттером полностью. Если сопротивление будет меньше, ничего страшного не произойдёт: транзистор всё равно откроется полностью. Главное в этом случае, следите, чтобы ток с Iskra Neo не превысил 40 мА. Для расчёта, опять же, используйте закон Ома:

$$I = \frac{U}{R}$$

Кроме биполярных существуют полевые транзисторы (рис. 16.2). Их выводы называются исток, сток и затвор и занимаются выводы именно тем, что следует из их названия. Ток от стока к истоку определяется напряжением на затворе. Обратите внимание — напряжением, а не током, как было в случае с биполярными транзисторами.

Вообще, полевые транзисторы идеологически похожи на конденсатор. Если ёмкость на затворе полностью заряжена, ток от стока к истоку не течёт. И от затвора к истоку — тоже. Поэтому полевые транзисторы энергоэффективнее биполярных, и для управления ими не нужен дополнительный резистор.

Существует разновидность полевых транзисторов, называемая MOSFET⁵² или МОП. Они очень популярны в применениях с мощными силовыми нагрузками, потому что легко управляются (особенно серии, специально предназначенные для работы с микроконтроллерами), меньше нагреваются, чем биполярные транзисторы, способны работать на больших частотах, хороши в ключевом режиме.

⁵² Аббревиатура от Metal-Oxide Semiconductor Field Effect Transistor: полевой транзистор из метал-оксидного полупроводника

Впрочем, для небольших — до одного ампера — нагрузок МОП принципиальных преимуществ перед биполярными не имеют. Особенно, если учесть, что МОП, в среднем, дороже.

16.3 Как вращать двигатель

Давайте применим биполярный транзистор как электронную кнопку для управления коллекторным двигателем.

Соберите схему, как на рисунках 16.3 и 16.4.

И, затем, загрузите на Iskra Neo следующий код:⁵³

```
int motorPin = 12; // Пин с транзистором

void setup()
{
    // Инициализируем пин, к которому подключен
    // транзистор, как работающий на вывод
    pinMode(motorPin, OUTPUT); }

void loop()
{
    // Закрываем «кран» на 2 секунды
    digitalWrite(motorPin, LOW);
    delay(2000);

    // Открываем «кран» на 2 секунды
    digitalWrite(motorPin, HIGH);
    delay(2000);
}
```

Двигатель должен начать периодически включаться и выключаться.

⁵³ File → Examples → Amperka → p16_motor_flipflop

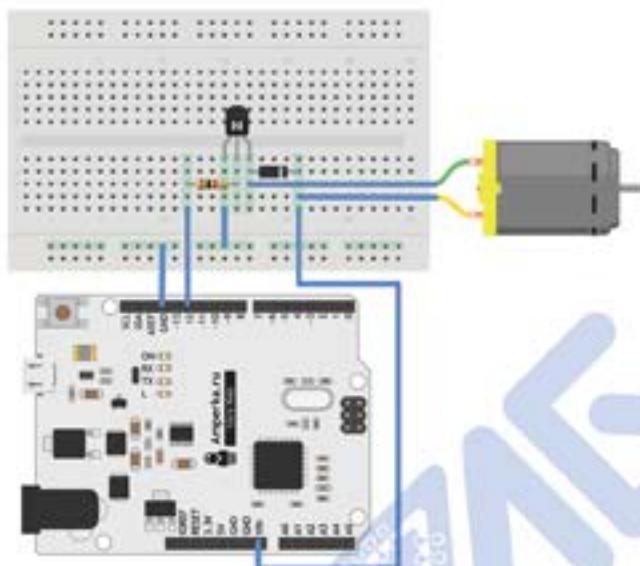


Рис. 16.3: Схема подключения мотора через биполярный транзистор на макетной доске

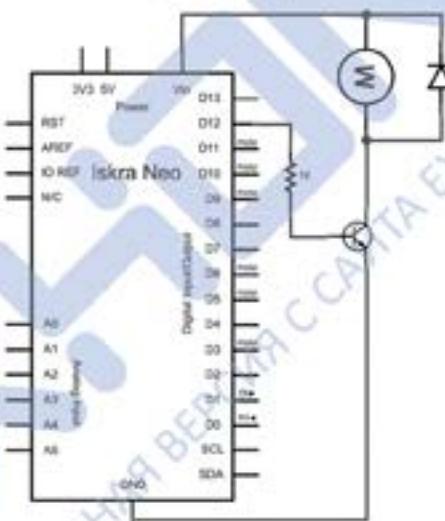


Рис. 16.4: Принципиальная схема подключения мотора через биполярный транзистор

Как это работает

Скетч в нашем устройстве довольно простой. Мы уже встречались с подобным, когда просто заставляли мигать подключённый светодиод: мы попаременно то подаём напряжение на управляющий пин, то убираем. Вся разница в том, что на этот раз мы подключили не светодиод, а схему из мотора, биполярного транзистора и диода. Поэтому давайте сосредоточимся на схеме.

Чтобы мотор вращался, необходимо подключить один из его выводов к земле, а другой — к напряжению питания. В нашей схеме в качестве источника напряжения питания мы использовали пин Vin вместо привычного 5V. Дело в том, что мотор — мощный потребитель тока, и даже небольшой привод может требовать для своей работы сотни, а иногда и тысячи миллиампер. Пин 5V может предоставить максимум 500 mA, а Vin — весь доступный ток от подключённого источника питания.

Ранее источником питания всегда выступал компьютер, к которому была подключена плата. Питание поставлялось через USB-кабель. Этого с головой хватало для работы слаботочных схем, но на этот раз лучше подключить к Iskra Neo внешний источник: батарейку, аккумулятор или сетевой адаптер. USB выдаёт максимум 100–500 mA в зависимости от компьютера, чего может не хватить для полноценного питания мотора.

Итак, если в качестве источника питания вы возьмёте батарейку «Крона» на 9 В, пин Vin будет выдавать напряжение 9 В и такой ток, какой позволяет отдавать батарейка: обычно он исчисляется десятками ампер, что в десятки раз больше того, что может дать USB подключение.

Ещё раз взгляните на нашу схему (рис. 16.4). Между выводом мотора и землёй мы поставили биполярный транзистор. Собственно, базой этого транзистора мы и управляем с Iskra Neo: то открывая «заслонку» между коллектором и эмиттером, то закрывая её. Открытие и закрытие «заслонки» приводит к тому, что ток через мотор то идёт, то нет. Поэтому в итоге двигатель то крутится, то останавливается.

Возникает вопрос: «Зачем нужен диод?». Здесь он выполняет защитную функцию от особого эффекта, которым обладают моторы. Вы уже знаете, что если подать на мотор напряжение, он начинает вращаться. Но наверняка не так очевидно то, что если вращать мотор внешними силами, он сам начинает выдавать напряжение.

На этом принципе работает большинство разновидностей электростанций: пар, вода или ветер вращают турбину, соединённую с мотором (в этом случае мотор называют генератором), а с выводов мотора снимают электричество, которое передают потребителям по проводам ЛЭП.⁵⁴

Вернёмся к нашему устройству. Представьте, что мы 2 секунды подряд подавали на мотор напряжение: его тяжёлый сердечник раскрутился до максимума; а затем мы в один момент перестали поставлять электричество. Сердечник не может остановиться так же мгновенно: он инертный, поэтому будет останавливаться постепенно. На это время мотор превращается в маленькую электростанцию: он начинает создавать напряжение, обратное по направлению тому, что было использовано для разгона.

Если бы мы не поставили диод, при торможении двигателя, в результате создаваемого им самим напряжения, ток бы устремился из земли к мотору, что могло бы повредить транзистор, стоящий на пути. Такие диоды, которые ставятся вокруг моторов, называют *возвратными*. Они позволяют пускать выработанный мотором ток обратно, через себя.

⁵⁴ ЛЭП — аббревиатура от Линий ЭлектроПередач

И последний момент. Обратите внимание, что мы разместили мотор между питанием и транзистором, а не между транзистором и землей. Это важно. Чтобы транзистор открылся, как уже говорилось, должен потечь ток через базу и эмиттер. Если бы мотор стоял между транзистором и землёй, он сам помешал бы течению тока с управляющего пина Iskra Neo через базу и эмиттер в землю. А при нашей конфигурации управляющему току ничего не мешает. Его силу лишь ограничивает резистор на 1 кОм, стоящий перед базой.

Напряжение, выдаваемое пином Iskra Neo — 5 В, поэтому управляющий ток будет:

$$I = \frac{U}{R} = \frac{5 \text{ В}}{10^3 \text{ Ом}} = 0,005 \text{ А} = 5 \text{ мА}$$

Это меньше предельных для пинов Iskra Neo 40 мА и в то же время достаточно для управления током до:

$$I_{\text{out}} = I \times h_{\text{fe}} = 5 \times 10^{-3} \text{ А} \times 200 = 1 \text{ А}$$

16.4 Как управлять скоростью двигателя

А теперь давайте попробуем поуправлять скоростью двигателя. Здесь нам помогут знания о ШИМ. Подобно тому, как ШИМ-сигнал позволял управлять яркостью светодиода, он же, если использовать его в отношении мотора, позволяет управлять скоростью вращения.

На этот раз, для разнообразия, вместо биполярного транзистора используем транзистор MOSFET.

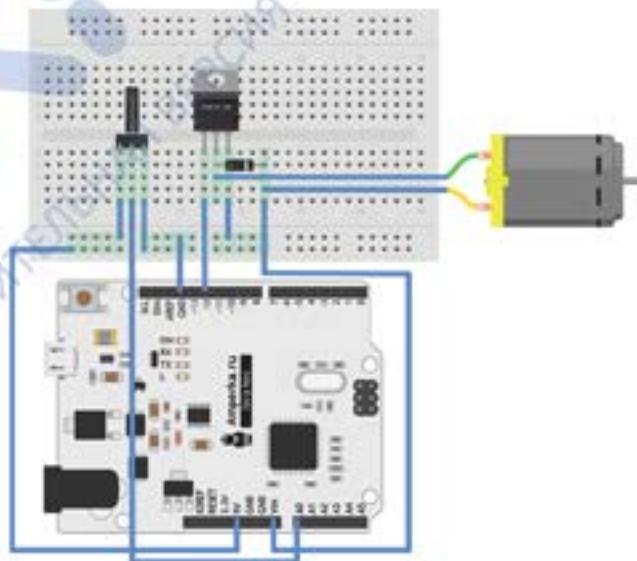


Рис. 16.5: Схема подключения мотора через полевой транзистор на макетной доске

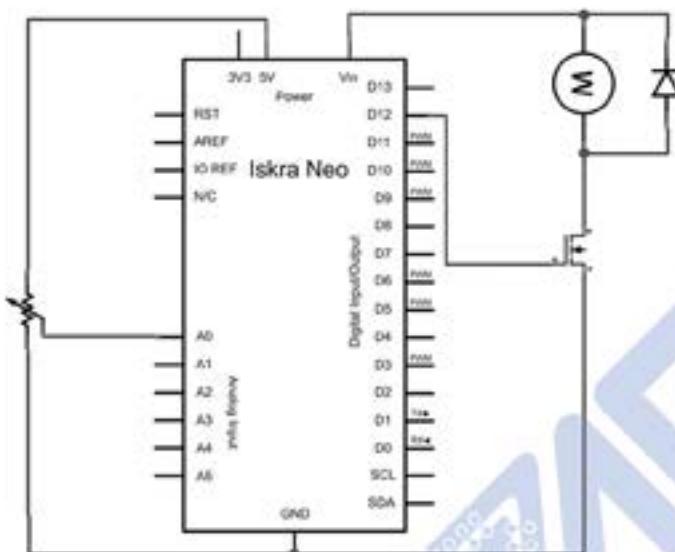


Рис. 16.6: Принципиальная схема подключения мотора через полевой транзистор

Добавьте к схеме потенциометр как на рисунках 16.5 и 16.6.

Загрузите следующий код:⁵⁵

```
int motorPin = 12; // Пин с транзистором
int potPin = A0; // Пин с потенциометром

void setup()
{
    // Инициализируем пин, к которому подключен
    // транзистор, как работающий на вывод
    pinMode(motorPin, OUTPUT);
}

void loop()
{
    // Считываем значение потенциометра
    // и делим его на 4, т.к. значение analogRead
    // варьируется от 0 до 1023, а значение
    // ШИМ-сигнала от 0 до 255
    int val = analogRead(potPin) / 4;
```

⁵⁵ File → Examples → Amperka → p16_motor_pwm

```
// Подаём на двигатель значение потенциометра
analogWrite(motorPin, val);
}
```

Теперь можно управлять скоростью двигателя при помощи потенциометра.

На этот раз мы не использовали в схеме токоограничивающий резистор: MOSFET-транзисторы управляются напряжением, а не током, поэтому ток через затвор не идёт, а следовательно, ограничивать ничего не нужно.

Итак, вы поняли: двигатели — мощная нагрузка, которая может легко спалить микроконтроллер при подключении напрямую. Поэтому мощные нагрузки управляются с помощью транзисторов или более сложных устройств.

Чем мощнее двигатель (или иная нагрузка), тем больше ему требуется ток, тем больше он создаёт побочных эффектов: индуктивных выбросов, способных пробить транзистор, в первую очередь. Для противодействия им появляются защитные диоды, RC-снаббераы, резистивные обвязки.

Для очень мощных двигателей ток заряда затвора MOSFET'а может оказаться слишком велик для микроконтроллера, поэтому одним лишь транзистором обойтись будет уже нельзя. Для этого существуют драйверы силовых транзисторов: микросхемы, усиливающие сигнал *Iskra Neo* и открывающие силовой транзистор, то есть транзисторы, открывающие транзисторы.

Когда же токи через полевой транзистор достигают сотен ампер, а сам транзистор достигает размера ладони, драйверы разрастаются до модулей со всеми возможными защитами, появляются системы жидкостного охлаждения и другие хитрости.

Но это всё играет роль, если вы делаете свой тепловоз. Для небольших моторов, используемых в игрушках, небольших станках, «умном доме», достаточно одного транзистора.

Но, даже с помощью этих моторов, можно сделать невероятное количество различных интересных устройств.



§17. ISKRA NEO И ИНТЕРНЕТ ВЕЩЕЙ

- 17.1 Модуль ESP8266 с поддержкой WiFi
- 17.2 Как проверить модуль ESP8266
- 17.3 Управление модулем ESP8266
- 17.4 Как сделать умный датчик температуры
- 17.5 Как это работает
- 17.6 Перспективы использования модуля ESP8266



ISKRA NEO И ИНТЕРНЕТ ВЕЩЕЙ

§17



§17. ISKRA NEO И ИНТЕРНЕТ ВЕЩЕЙ

Давайте пофантазируем с вами о том, как было бы здорово, если окружающие нас вещи могли бы самостоятельно без нашего участия выполнять функции, чтобы сделать нашу жизнь еще более удобной и комфортной. Чтобы освещенность в помещении изменялась автоматически в зависимости от времени суток или погоды. Кофеварка сама заранее включалась бы к моменту, когда нам захочется ароматного кофе. Чтобы система полива сада сама следила за оптимальной влажностью почвы. Чтобы в домах поддерживалась комфортная температура и при этом сберегалась электроэнергия. Чтобы вещи дома не терялись и всегда находились при необходимости.

По отдельности автоматизация таких систем возможна уже давно. Но если информацию, получаемую от разных «умных вещей» объединить, то можно еще повысить комфортность. Например, в жаркий летний вечер мы вернулись в свою квартиру, открыв замок двери электронным ключом. На основе данных от электронного замка система безопасности отключила режим охраны помещений квартиры. Датчик движения системы безопасности квартиры обнаружил, что в комнату вошел человек, при этом датчик освещенности определил, что наступили сумерки, а «умная лампа» на основе этих данных («в комнате человек» и «сейчас сумерки») включила освещение. «Умный термометр» на стене определил, что в комнате температура для человека намного выше комфортной, а «умный кондиционер» на основе этих данных («в комнате человек» и «в комнате жарко») включил режим охлаждения воздуха. «Умное окно» на основе полученных данных о том, что «человек в комнате», «включена лампа» и «сейчас сумерки», закрыло жалюзи и на основе данных, что «в комнате жарко» и «кондиционер включил режим охлаждения», для эффективности закрыло форточку.

Это фантазия может стать возможной, если каждую вещь снабдить небольшим чипом-меткой с возможностью подключения к нему сенсоров, актуаторов⁵⁶ и передачи данных для объединения устройств, так называемых «умных вещей», в единую вычислительную сеть, обслуживаемую интернет-протоколами. «Интернет вещей» – это не просто множество различных устройств и датчиков, объединенных между собой проводными и беспроводными каналами связи и подключенных к сети Интернет, это скорее более тесная интеграция реального и виртуального миров, благодаря которой осуществляется общение между людьми и устройствами.

Концепция «Интернета вещей», IoT⁵⁷ появилась не так давно, поэтому, создавая проекты по этой технологии, вы безусловно поучаствуете в создании устройств будущего, станете новаторами.

Можем ли мы уже сейчас приступить к созданию «Интернета вещей» вокруг себя? Да, для этого нам сначала нужно научиться создавать датчики (сенсоры) и исполнительные устройства, которые могли бы связываться между собой, желательно посредством радиоканала. Для беспроводной связи в настоящий момент широко применяется WiFi: точки доступа и роутеры WiFi имеются во многих домах,

⁵⁶ Актуатор – исполнительное устройство (привод, электропривод) управляемое с помощью сигналов управления, действующее на реальный мир. ⁵⁷ англ. «Internet of Things» - «Интернет вещей»

большинство ноутбуков, планшетов и смартфонов имеют встроенную поддержку WiFi. Использование этой беспроводной технологии связи позволит нам свободно контролировать и считывать информацию, передаваемую «умными вещами». Но для этого нам нужен модуль, позволяющий связать наши устройства на базе Iskra Neo с другими устройствами по сети WiFi. И такой модуль существует!

17.1 Модуль ESP8266 с поддержкой WiFi

Этот параграф посвящен потрясающему чипу esp8266, который очень скоро изменит многие вещи в окружающем нас мире. Он очень просто позволяет подключить контроллер Iskra Neo к доступной WiFi сети или другому WiFi устройству. Подобные решения были известны и раньше, но дело в том, что цена этого чипа очень доступная. Кроме низкой цены этот чип имеет малые размеры, сравнительно низкое энергопотребление.

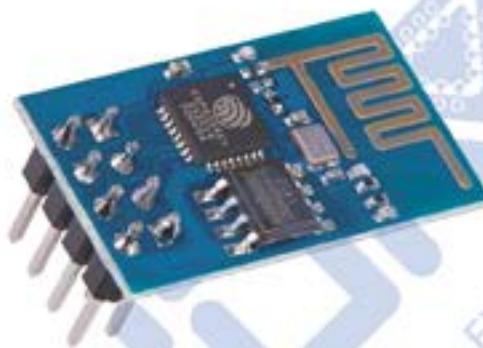


Рис. 17.1: Модуль с чипом ESP8266 ESP-01 V090 с поддержкой WiFi

Модули на базе ESP8266 имеют минимальное количество дополнительных компонентов обвязки. Это позволяет с помощью данного чипа подключить создаваемые нами электронные устройства к домашней сети WiFi и поучаствовать в продвижении технологии интернета вещей. Когда устройства будут подключены к интернету, они смогут стучаться к вам в социальных сетях, будут посыпать письма по электронной почте и взаимодействовать с вами и друг с другом посредством различных интернет-протоколов и интернет-сервисов.

Рассмотрим характеристики модуля ESP8266 ESP-01 V090 (рис. 17.1):

- Модификация: ESP-01 V090
- Беспроводной интерфейс: Wi-Fi 802.11 b/g/n 2,4 ГГц
- Режимы: P2P (клиент), soft-AP (точка доступа)
- Максимальная выходная мощность: 19,5 дБ·мВт (89 мВт)
- Номинальное напряжение: 3,3 В
- Максимальный потребляемый ток: 220 мА

- Антенна: PCB
- Дальность на открытом пространстве: до 400 м
- Портов ввода-вывода свободного назначения: 2
- Частота процессора: 80 МГц
- Объём памяти для кода: 64 КБ
- Объём оперативной памяти: 96 КБ
- Габариты: 21 × 13мм

В семействе модулей ESP8266 есть много разновидностей. Представленный модуль — ESP-01 V090. У него антenna встроена на плату, а на ножки дополнительно выведены 2 GPIO-порта свободного назначения.

Питать модуль ESP8266 нужно стабилизированным напряжением 3,3 вольта. Такое питание можно получить с отдельного регулятора напряжения или с пина 3.3V на некоторых платах Arduino. В перечне характеристик мы видим, что модуль потребляет в пике 220 mA. Регулятора напряжения, используемого на пятивольтовых платах Arduino для пина 3.3V, может оказаться недостаточно! Обратите внимание на характеристики своей платы. Например, Arduino Uno и Arduino Leonardo могут выдать не более 50 mA с пина 3.3V, поэтому они для питания ESP8266 не подходят, а вот Iskra Neo от фирмы Амперка может выдать до 800 mA, поэтому можно питать ESP8266 прямо от этой платы.

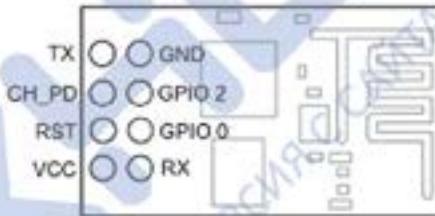


Рис. 17.2: Распиновка модуля ESP8266 ESP-01 V090

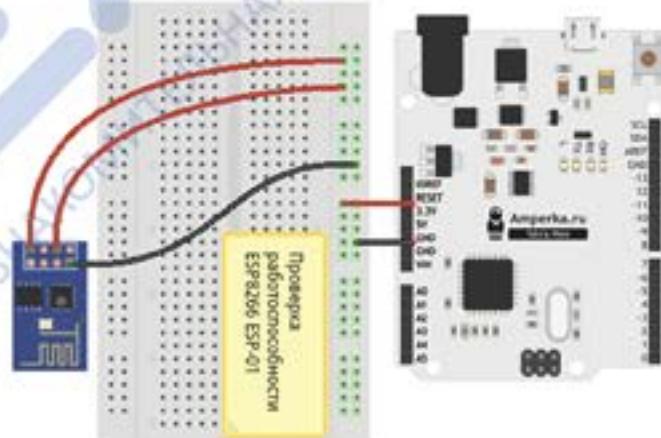


Рис. 17.3: Схема для проверки работоспособности ESP8266 ESP-01 V090

17.2 Как проверить модуль ESP8266

Обозначения пинов модуля изображены на рис. 17.2. Из-за расположения ножек вплотную в 2 ряда, модуль нельзя установить на макетную плату. Поэтому мы воспользуемся проводами с разъёмами «мама»-«папа» для подключения к пинам модуля.

Чтобы проверить работоспособность ESP8266 ESP-01 достаточно подключить три пина: VCC и CH_PD (chip enable) к питанию 3,3 вольта, а GND к земле. Питание 3,3 вольта возьмем от пина 3.3V платы Iskra Neo (рис. 17.3), т.к. мы уже знаем, что регулятор напряжения 3,3 вольта этой платы может выдать до 800 mA.

При успешном старте заводской прошивки на модуле ESP8266 загорится красный светодиод и пару раз мигнет синий (это индикатор передачи данных от модуля к терминалу по линии TX-RX) и в вашей беспроводной сети должна появится новая точка доступа с именем «AI-THINKER_AXxxxx» или «ESP_XXXX», которую вы сможете увидеть с любого WiFi устройства, например, со смартфона. Если такая точка доступа WiFi появилась, то значит модуль исправен, и можно продолжить работу с модулем далее.

17.3 Управление модулем ESP8266

Управлять модулем ESP8266 можно посредством AT-команд⁵⁸. Подробное описание AT-команд для управления ESP8266 вы сможете найти в сети Интернет⁵⁹, но для простоты мы рассмотрим только некоторые из них. Для этого нам необходимо подключить ESP8266 к Iskra Neo так, чтобы они связывались друг с другом посредством UART (Serial1, привязанный к пинам RX/TX) (рис. 17.4 и 17.5) и запрограммировать Iskra Neo так, чтобы данные, вводимые нами в Мониторе порта платы Iskra Neo (Serial, привязанный к USB) передавались на ESP8266, а данные от ESP8266 транслировались в Монитор порта. Питание для ESP8266 будет брать от контакта 3.3V платы Iskra Neo.

Теперь подробнее разберемся с безопасным подключением модуля ESP8266 к плате Iskra Neo. Родное напряжение модуля ESP8266 — 3,3 вольта. Его пины не толерантны к напряжению 5 вольт, с которым работает контроллер Iskra Neo. Если подать напряжение выше, чем 3,3 вольта на пины питания, коммуникации или ввода-вывода, то модуль ESP8266 выйдет из строя.

Поэтому для передачи данных на модуль ESP8266 с 5-вольтовых пинов платы Iskra Neo потребуется использовать делитель напряжения, чтобы перевести напряжение из 5 вольт в допустимый диапазон 3,3 вольта. О делителе напряжения вы уже читали в 12 параграфе. Мы применим делитель напряжения из трех резисторов одинакового номинала в 1 кОм (два резистора номиналом 1 кОм, соединенные последовательно, имеют общее сопротивление 2 кОм) между pinом TX (передача данных) контроллера Iskra Neo и pinом RX (прием данных) модуля ESP8266, таким образом сделаем соединение безопасным для последнего.

⁵⁸ Набор команд, разработанных в 1977 году компанией Hayes для модема. ⁵⁹ Описание AT-команд - <http://microsin.net/adminstuff/hardware/esp8266-at-commands-reference.html>

А вот для передачи данных от ESP8266 в контроллер Iskra Neo никаких «посредников» не нужно. Дело в том, что от пина TX (передача данных) модуля ESP8266 сигнал напряжением 3,3 вольта как есть будет воспринят платой Iskra Neo на пине RX (прием данных), как логическая единица.

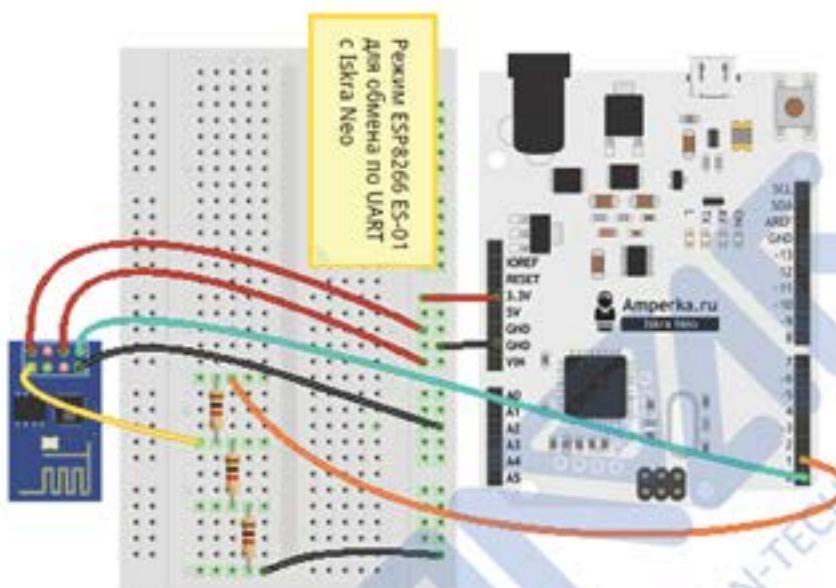


Рис. 17.4: Схема подключения ESP8266 к Iskra Neo для обмена по UART

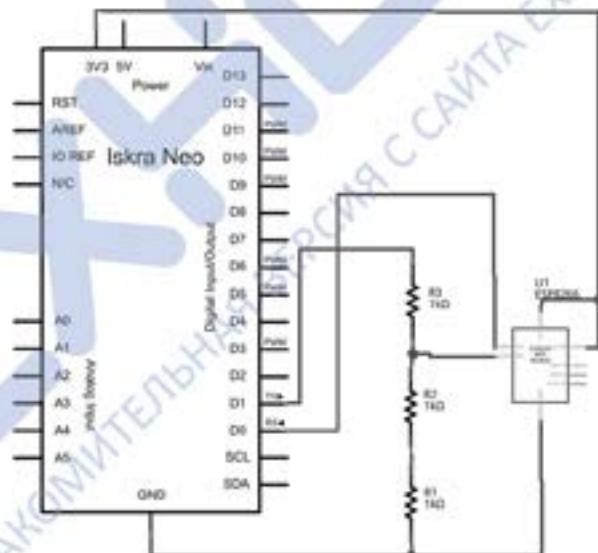


Рис. 17.5: Принципиальная схема подключения ESP8266
к Iskra Neo для обмена по UART

Когда соберете схему, проверите ее и подключите питание к Iskra Neo, то грядущий на макетной плате светодиод будет показывать готовность модуля ESP8266 к работе.

Загрузите следующий код:⁶⁰

```
void setup()
{
    Serial1.begin(115200);
    Serial.begin(115200);
}

void loop()
{
    if (Serial1.available()) {
        Serial.write(Serial1.read());
    }
    if (Serial.available()) {
        Serial1.write(Serial.read());
    }
}
```

Теперь, не отключая USB-кабель от компьютера к Iskra Neo, рассмотрим некоторые AT-команды для управлением модуля ESP8266, которые можно вводить в Мониторе порта и видеть ответные сообщения.

Команда AT+GMR выводит сообщение о версии AT-команд и версии прошивки модуля ESP8266, например (у вас могут быть другие данные):

```
AT+GMR
AT version:0.50.0.0(Sep 18 2015 20:55:38)
SDK version:1.4.0
compile time:Sep 18 2015 21:46:52
OK
```

Команда AT+RST производит рестарт (сброс) модуля ESP8266. Введите эту команду в Мониторе порта, нажмите Enter и увидите ответ от платы в несколько строк: системную информацию о модуле ESP8266.

Команда AT+CWMODE=3 нужна для настройка режима Wi-Fi и перевода ESP8266 в режим станции. Введите эту команду, в ответ увидите «OK», что означает - «команда выполнена».

Следующая команда AT+CWJAP="SSID", "PASSWORD" дает инструкцию модулю ESP8266 подключиться к WiFi-роутеру с названием SSID и паролем PASSWORD.

⁶⁰ File → Examples → Amperka → p17_ESP8266_2_IskraNeo

Вы, конечно, догадались, что вместо SSID и PASSWORD нужно подставить те параметры, которые у вашего WiFi-роутера используются для подключения WiFi-устройств, таких как ноутбук или смартфон. Ответ ESP8266 на эту команду выглядит так:

```
WIFI CONNECTED WIFI GOT IP
```

```
OK
```

Это значит, что модуль ESP8266 успешно подключился к WiFi-роутеру и получил IP-адрес.

Чтобы узнать, какой IP-адрес получил ESP8266, нужно ввести команду AT+CIFSR. В ответ вы увидите несколько строчек, например:

```
AT+CIFSR
```

```
+CIFSR:APIP,"0.254.239.239"
```

```
+CIFSR:APMAC,"00:01:00:00:03:05"
```

```
+CIFSR:STAIP,"192.168.0.201"
```

```
+CIFSR:STAMAC,"18:fe:34:cf:46:50"
```

```
OK
```

Обратите внимание на строку +CIFSR:STAIP,"192.168.0.201". В ней написан IP-адрес 192.168.0.201 модуля ESP8266. Зная IP-адрес, вы можете протестировать интернет-связь, например, в командной строке компьютера, также подключенного к WiFi-роутеру, ввести команду ping:

```
ping 192.168.0.201
```

Отлично! Наш модуль ESP8266 уже в Сети!

Для реализации концепции «Интернета вещей» в качестве точки для обмена информацией часто используется сервер⁶¹. Мы будем использовать возможности модуля ESP8266 подключаться к имеющейся WiFi сети и создавать telnet-сервер⁶². С помощью telnet-сервера можно передавать подключившемуся к нему через WiFi telnet-клиенту данные, полученные через Serial соединение (UART) от платы Iskra Neo.

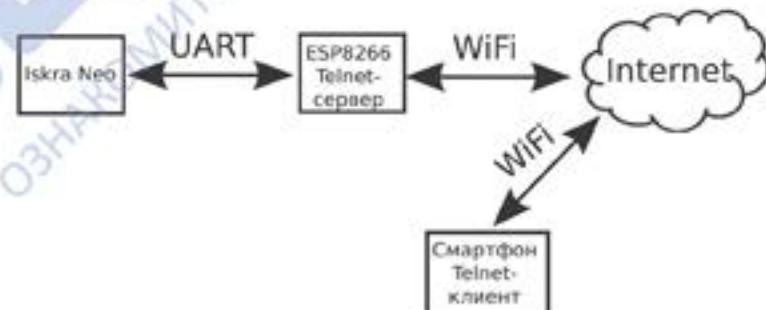


Рис. 17.6: Схема обмена сообщениями между Iskra Neo и telnet-клиентом

⁶¹ Сервер (от англ. server, обслуживающий). ⁶² TELNET (сокр. от англ. terminal network) — сетевой протокол. Стандарт протокола описан в RFC 854.

Мы запрограммируем передачу данных между Iskra Neo и telnetклиентом по схеме 17.6. Из данной схемы видно, что модуль ESP8266 выполняет роль информационного шлюза между Iskra Neo и telnetклиентом. В качестве telnet-клиента можно использовать ноутбук или персональный компьютер, смартфон или планшет, подключенные к WiFi сети, с установленной программой telnet. Пользователь telnet-клиента, сможет получать данные от Iskra Neo и передавать ей свои команды, находясь в любой точке зоны покрытия WiFi сети, а при соответствующей настройке WiFi-роутера - из любой точки земного шара, где имеется доступ к Интернет.

Чтобы модуль ESP8266 заработал как telnet-сервер, введем следующие две команды:

AT+CIPMUX=1 - разрешить множественные подключения к модулю (иначе сервер не запустится),

AT+CIPSERVER=1, 23 - старт на модуле ESP8266 telnet-сервера на TCP-порту 23.

Ответное сообщение "OK" свидетельствует о нормальном запуске сервера.

Теперь можно протестировать работу TCP-сервера. Для этого нам потребуется telnet-клиент. В качестве программы telnet-клиента можно использовать на персональных компьютерах и ноутбуках утилиту putty, которая доступна для скачивания по ссылке <http://putty.org.ru/download.html>, а на планшетах и смартфонах под управлением Android можно использовать, например, приложение Telnet <https://goo.gl/aGvpBZ>. Нам нужно запустить программу telnet-клиента на компьютере, ноутбуке или смартфоне, указав в командной строке IP-адрес и порт сервера, например:

```
telnet 192.168.0.201 23
```

При этом в окне Монитора порта Iskra Neo вы увидите сообщение о новом успешном подключении:

```
0,CONNECT
```

0 - это socket, т.е. номер подключения telnet-клиента к серверу, через который будет совершаться обмен данными между сервером и клиентом. Причем, по-умолчанию к серверу на модуле ESP-8266 могут подключиться до 5 различных telnet-клиентов одновременно, и у каждого будет свой уникальный socket.

Если долгое время между клиентом и сервером не будут передаваться сообщения, то сервер автоматически отключит клиента, и тогда в Мониторе порта мы увидим номер сокета и сообщение о закрытии этого соединения:

```
0,CLOSED
```

Попробуйте, пока соединение не закрыто, в подключенной к серверу на ESP8266 программе telnet ввести "Hello!" и нажать Enter. В Мониторе порта вы увидите такое сообщение:

```
+IPD,0,8>Hello!
```

, где "+IPD" - признак входящего от telnet-клиента сообщения, "0" - socket подключения, "8" - длина сообщения в символах, "Hello!" само сообщение. Но если вы внимательно подсчитали символы, то, наверняка, возразите, что "Hello!" - это 6 символов, откуда же 8?! Дело в том, что когда вы ввели "Hello!" и нажали Enter, то к сообщению добавилось в конце еще два символа - "перевод каретки" (CR) и "новая строка" с кодами 13 и 10 соответственно. В среде Arduino IDE эти символы обозначаются как '\r' и '\n'. Запомним - это нам пригодится.

Чтобы telnet-сервер отправил сообщение клиенту, в Мониторе порта введем команду

```
AT+CIPSEND=0, 3
```

, где "0" - socket клиента, которому отправляется сообщение, "3" - число символов в сообщении. Если модуль ESP8266 ответит символом ">", - это значит, что он готов принять от нас сообщение для передачи. Введем Hi! и нажмем Enter, при этом в окне Монитор порта будут видны такие надписи:

```
AT+CIPSEND=0, 3
```

```
OK
```

```
>
```

```
busy s...
```

```
Recv 3 bytes
```

```
SEND OK
```

В окне программы telnet мы увидим сообщение "Hi!", - сообщение telnet-клиенту доставлено. Ура!

Команда AT+CIPSTATUS нужна для просмотра подробного состояния (статуса) telnet-сервера на ESP8266. Например:

```
AT+CIPSTATUS
```

```
STATUS:3
```

```
+CIPSTATUS:0,"TCP","192.168.0.202",46974,333,1
```

```
+CIPSTATUS:1,"TCP","192.168.0.202",47070,333,1
```

```
OK
```

, где: STATUS:3 - номер текущего состояния модуля ESP8266, бывают:

- 2 - получение IP-адреса от точки доступа,
- 3 - соединение установлено,
- 4 - отключено (Disconnect, Link (Socket) Closed),
- 5 - отключено (Disconnect from Last Used AP).

Далее каждая строка, начинающаяся "+CIPSTATUS:" - соответствует подключенному telnet-клиенту. В ней через запятую указаны:

- socket соединения (0..4) для режима множественного доступа,
- строка, где в двойных кавычках указан тип IP-соединения ("TCP" или "UDP"),
- строка в двойных кавычках, показывающая IP-адрес,
- десятичный номер порта,
- тип подключения: "0" - ESP8266 работает как клиент, или "1" ESP8266 работает как сервер.

Команда AT+CIPCLOSE=0 нужна для отключения клиента с socket'ом "0" от сервера.

Теперь после знакомства с основными AT-командами модуля ESP8266 мы можем приступить к сборке схемы «умного датчика температуры».

17.4 Как сделать умный датчик температуры

В качестве первого умного устройства мы соберем «умный датчик температуры». С помощью термистора, с которым мы познакомились в параграфе 12, контроллер Iskra Neo будет получать данные и вычислять текущую температуру в градусах по шкале Цельсия. Данные о температуре будут по запросу передаваться telnet-клиентам, подключившимся через WiFi к ESP8266. Давайте также введем возможность получения команд контроллером Iskra Neo от telnet-клиентов и их выполнение. Пусть для простоты это будут трехбуквенные команды:

- hlp - показать краткую подсказку telnet-клиенту,
- max - показать максимальную температуру за все время измерений,
- min - показать минимальную температуру,
- rst - сбросить сохраненные значения минимальной и максимальной температур,
- trm - показать текущую температуру всем telnet-клиентам,
- cls - завершить текущее подключение telnet-клиента.

Схему подключения термистора вы уже знаете по параграфу 12.

Нам потребуется сам термистор и резистор номиналом 10 кОм.

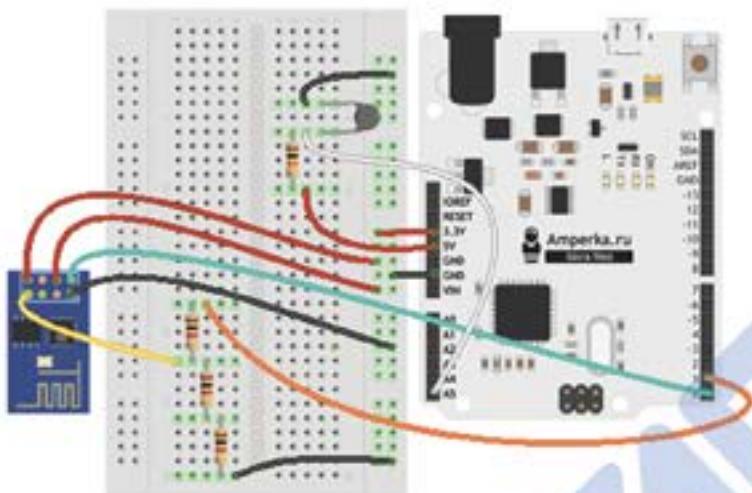


Рис. 17.7: Схема умного датчика температуры

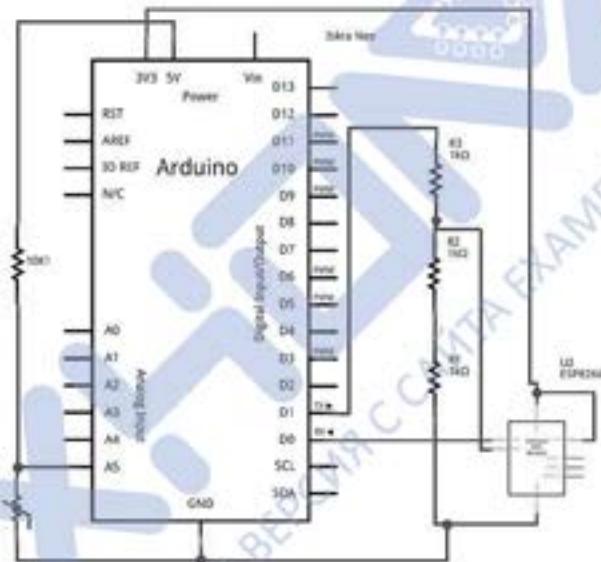


Рис. 17.8: Принципиальная схема умного датчика температуры

Соберите схему умного датчика температуры на макетной плате как это изображено на рисунке и принципиальной схеме (рис. 17.7 и рис. 17.8).

Загрузите в плату Iskra Neo следующий скетч:⁵¹

```
/*
Данный скетч загружается в Iskra Neo.
Работает совместно с модулем
ESP8266 ESP-01 для реализации проекта
Smart termometr - умный датчик температуры
по технологии "Интернет вещей"
```

⁵¹ File → Examples → Amperka → p17_thermistor_smart

```
1. Измеряет температуру термистором
2. Передает данные температуры по UART на ESP8266
3. Получает команды с UART и выполняет их
4. Передает все сообщения с UART на USB
*/
#define DEBUG true //показывать сообщения

//данные для подключения к WiFi-роутеру,
//где SSID - имя WiFi-роутера
#define SSID "****" //замените звездочки на свой SSID
#define PASS "****" //замените звездочки на пароль

//номера соединений клиентов сервера
int connection[5]={0,0,0,0,0};

int termPin = A5; //Аналоговый пин с термистором
int termMax = -100; //для максимума температуры
int termMin = 200; //для минимума температуры
int term = 0; //температура

String msgHelp="Command:\r\nhelp - help\r\n\
rst - Reset of min/max\r\n\
max - Show max temperature\r\n\
min - Show min temperature\r\n\
trm - Show current temperature\r\n\
cls - Close connection\r\n";

//Отправить сообщение telnet-клиенту ESP8266
String sendMsg(String msgClient, int socket,
               boolean debug)
{
    char buf[10];
    String response = "";
    ...
}
```

```

//Создать команду для ESP8266 на
//отправку сообщения
String msg = "AT+CIPSEND=";
sprintf(buf, "%d", socket);
msg+=buf;
msg+=",";
msg+=String(msgClient.length());
msg+="\r\n";

//отправить команду на ESP8266
response = sendData(msg,1000,debug);

//Если ESP8266 ответил ">" отправить
//текст сообщения для телнет-клиента
if (response.indexOf(">") != -1){
    Serial1.print(msgClient);
    long int time = millis();
    response = "";
    //секунду ждать ответ от ESP8266
    while ( (time + 1000) > millis())
        while (Serial1.available()){
            char c = Serial1.read();
            response += c;
        }
    //Если режим DEBUG, то вывести ответ
    //в Монитор порта Iskra Neo
    if (debug){
        Serial.println(response);
        Serial.println("OK \u222a Send \u222a msg");
    }
} else
    if (debug){

```

```
    Serial.println(response);
    Serial.println("ERROR \u2225Not \u2225send \u2225msg");
}
}

//Отправка команды по UART в ESP8266 с учетом таймаута
String sendData(String command, const int timeout,
    boolean debug)
{
    String response = "";
    //Отправить команду на ESP8266
    Serial1.print(command);

    //ждать ответа в течение таймаута
    long int time = millis();
    while ( (time + timeout) > millis())
        while (Serial1.available()) {
            char c = Serial1.read();
            response += c;
        }

    //Если режим DEBUG, то вывести ответ
    //в Монитор порта Iskra Neo
    if (debug) Serial.print(response);
    return response;
}
void setup() {
    String result;
    //составить команду для подключения ESP8266
    //к WiFi роутеру с названием в SSID
    //и паролем в PASS
    String cmd="AT+CWJAP=\"";
    cmd+=SSID;
```

```

cmd+="\", \";
cmd+=PASS;
cmd+="\"\\r\\n";

//Инициализация UART для USB
Serial.begin(115200);

//Инициализация UART для связи с
//ESP8266, подключенного к pinам TX/RX
Serial1.begin(115200);

//Сброс (reset) ESP8266
sendData("AT+RST\\r\\n", 2000, DEBUG);
//Установить режим станции (set station mode)
sendData("AT+CWMODE=3\\r\\n", 1000, DEBUG);
//Подключить к WiFi-роутеру (connect to the wifi)
sendData(cmd, 2000, DEBUG);
while(!Serial1.find("OK")) {
    //Ждать подключения к WiFi
    Serial.print("Connect to the WiFi ");
    Serial.println(SSID);
    delay(500);
}

//включить режим множественных соединений (макс 5)
sendData("AT+CIPMUX=1\\r\\n", 1000, DEBUG);

//Запустить telnet-сервер на порту 23
sendData("AT+CIPSERVER=1,23\\r\\n", 1000, DEBUG);

//показать IP address telnet-сервера ESP8266
// в Мониторе порта Iskra Neo
sendData("AT+CIFSR\\r\\n", 1000, DEBUG);
}

```

```
void loop() {
    String s = "";
    String valueStr = "";
    int value = 0;
    String msg = "";
    char buf[10];

    //Считать показания термистора
    int termData = analogRead(termPin);

    //Перевод единиц с термистора в градусы
    //по шкале Цельсия (температура)
    term = (700 - termData) / 8;

    //Сохранить минимум и максимум температур
    if (term < termMin) termMin = term;
    if (term > termMax) termMax = term;

    //Проверить поступление данных от ESP8266
    while (Serial1.available() > 0){
        //Если пришли данные, то читать символ
        char c = Serial1.read();

        //если символ = CR "конец строки", то
        //разобрать строку s
        if (c == '\n'){
            if (DEBUG) Serial.println(s);
            //====разбор принятых сообщений====
            //Если новое подключение клиента
            if (s.indexOf(",CONNECT") != -1){
                valueStr = s.substring(0,1);
                value = valueStr.toInt();
            }
        }
    }
}
```

```

        //запомнить статус подключения
        connection[value] = 1;
        if (DEBUG){
            Serial.print(value);
            Serial.println("CONNECT-OK");
        }
        //Отправить подсказку новому клиенту
        sendMsg(msgHelp,value,DEBUG);
    }

    //Если отключение клиента
    if (s.indexOf(",CLOSED") != -1){
        valueStr = s.substring(0,1);
        value = valueStr.toInt();
        //удалить статус подключения
        connection[value] = 0;
        if (DEBUG){
            Serial.print(value);
            Serial.println("CLOSED-OK");
        }
    }

    //Поступило сообщение от телнет-клиента
    if (s.indexOf("+IPD,") != -1){
        //определить сокет телнет-клиента
        valueStr = s.substring(value+5,value+6);
        int socket = valueStr.toInt();
        //найти в сообщении ":""
        value = s.indexOf(":");
        //вырезать три символа после ":""
        //в которых содержится команда
        valueStr = s.substring(value+1,value + 4);
        if (DEBUG){
            Serial.println(socket);
        }
    }
}

```

```
        Serial.println("MSG - OK");
    }
    //Если поступила команда hlp (help)
    if (valueStr == "hlp"){
        sendMsg(msgHelp,socket,DEBUG);
    }
    //Если поступила команда rst (restart)
    if (valueStr == "rst"){
        termMax = term;
        termMin = term;
        msg = "Reset min/max - Ok\r\n";
        sendMsg(msg,socket,DEBUG);
    }
    //Если поступила команда min (minimum)
    if (valueStr == "min"){
        sprintf(buf, "%d\r\n", termMin);
        msg = "Minimum temperature: ";
        msg+=buf;
        sendMsg(msg,socket,DEBUG);
    }
    //Если поступила команда max (maximum)
    if (valueStr == "max"){
        sprintf(buf, "%d\r\n", termMax);
        msg = "Maximum temperature: ";
        msg+=buf;
        sendMsg(msg,socket,DEBUG);
    }
    //Если поступила команда trm (Температура)
    //сообщить температуру по UART для ESP8266
    //всем подключенными клиентам
    if (valueStr == "trm"){
        for(int i = 0; i < 5; i++){

```

```

        if(connection[i] == 1) {
            sprintf(buf, "%d\r\n", term);
            msg = "Temperature: ";
            msg+=buf;
            sendMsg(msg,i,DEBUG);
        }
    }

//Если поступила команда закрыть
//соединение с телнет-клиентом
if (valueStr == "cls"){
    //Создать строку с командой
    sprintf(buf,"%d\r\n",socket);
    msg = "AT+CIPCLOSE=";
    msg+=buf;
    //команда для ESP8266 для закрытия
    //соединения с номером socket
    sendData(msg, 0, DEBUG);
}
}

//после разбора очистить строковую переменную
S="";
//====конец разбора принятых сообщений====
}

//записать символ в переменную s else{
    S += c;
    //обязательная задержка
    delay(2);
}

//Передать данные из UART (USB) на UART (ESP8266)

```

```
while (Serial.available() > 0)
    Serial1.write(Serial.read());
}
```

После подключения питания к схеме умного датчика температуры в окне Монитор порта, подключенного к Iskra Neo вы увидите сообщение о статусе подключения модуля ESP8266 к сети WiFi, его ip-адрес и 23 порт. По этому адресу и порту подключитесь к умному датчику температуры по протоколу telnet с помощью ПК, ноутбука, планшета или смартфона. Вы увидите приглашение и сообщение о командах, которые может выполнять умный датчик температуры. Попробуйте нагреть пальцами термистор, и вводите команду `tmp`, чтобы увидеть изменения показаний температуры в telnet программе.

Поскольку мы создали интерактивный датчик, который различает и выполняет некоторые команды, то вы можете в telnet-программе ввести одну из нескольких, запрограммированных нами команд и получить ответ. Например, команда `max` покажет максимальную температуру с момента включения датчика, а команда `min` — минимальную. Командой `rst` можно сбросить сохраненные ранее значения `max` и `min` температур, а команда `help` выведет краткую подсказку по командам умного датчика температуры. Для отключения telnet-клиента можно воспользоваться командой `cls`.

Поздравляем! Вами создана первая умная интернет-вещь!

17.5 Как это работает

В процедуре `setup()` инициализируется два интерфейса UART. Один (`Serial`) для передачи сообщений на USB компьютера, к которому подключена плата Iskra Neo. Благодаря этому, в окне «Монитор порта» мы увидим сообщения, пересылаемые модулем ESP8266 на Iskra Neo, в том числе сообщения о статусе подключения к сети WiFi и ip-адресе. Второй (`Serial1`) для передачи данных на модуль ESP8266 через пины TX/RX. В обоих случаях мы устанавливаем скорость 115200 бод.

```
//Инициализация UART для USB
Serial.begin(115200);

//Инициализация UART для связи с
//ESP8266, подключенного к pinам TX/RX
Serial1.begin(115200);
```

В этой же процедуре с помощью AT-команд, которые передаются в ESP8266 функцией `sendData()`, делаем рестарт модуля, затем настраиваем подключение модуля ESP8266 к WiFi-роутеру и запускаем telnet-сервер на порту 23.

Чтобы перевести показания термистора в градусы по шкале Цельсия и откалибровать их, нам потребуется рассчитать сколько единиц показаний термистора соответствуют одному градусу Цельсия. Для упрощения будем считать, что зависимость показаний термистора от температуры линейная, тогда для расчетов нам достаточно узнать два разных показания термистора и соответствующие им температуры в градусах по Цельсию. Мы провели свои измерения, но для своего термистора вам потребуется провести свои замеры, чтобы точно его откалибровать.

Итак. Температура в нашем помещении согласно комнатного термометра на момент измерения составляла 22 градуса Цельсия, при этом с термистора мы получили показания 411. Нормальная температура тела человека 36°C, поэтому мы стали нагревать пальцами термистор, подождали пока показания перестали меняться и записали показания термистора - 524.

Используя полученные сведения, посчитаем, сколько единиц в показаниях термистора приходится на 1 градус Цельсия:

$$\frac{(524 - 411)}{(36 - 22)} \approx 8$$

Следовательно, теперь мы сможем посчитать, какие показания термистора будут соответствовать температуре 0°C. Заметив, что показания термистора увеличиваются при уменьшении температуры, прибавим к показаниям термистора при температуре 22°C (524) произведение 22 и 8. Именно на столько единиц увеличиваются показания термистора, если температура опустится от 22°C до 0°C:

$$524 + 8 \times 22 = 700$$

Теперь мы знаем, что для перевода единиц показаний термистора в градусы по Цельсию нужно от 700 отнять текущие показания термистора и результат поделить на 8:

$$\frac{(700 - \text{Показания_термистора})}{8} = \text{Температура}(^{\circ}\text{C})$$

Именно так в процедуре `loop()` мы переводим показания термистора в градусы по шкале Цельсия:

```
//Считать показания термистора
int termData = analogRead(termPin);
//Перевод единиц с термистора в градусы
//по шкале Цельсия
int term = (700 - termData) / 8;
```

В случае появления входящих сообщений от ESP8266 формируем строку String s. Если поступает символ "перевод каретки" (CR), то сравниваем содержимое сформированной строки s с "CONNECT", "CLOSE" и "+IPD,". В последнем случае добавляем еще и распознавание трехсимвольных команд от telnet-клиентов и в случае совпадения, программируем их выполнение.

Определяем, вводил ли telnet-клиент команду, например, «hlp», и если команда была введена, то отрабатываем ее:

```
//Поступило сообщение от телнет-клиента
if (s.indexOf("+IPD,") != -1){
    //определить сокет телнет-клиента
    valueStr = s.substring(value+5,value+6);
    int socket = valueStr.toInt();
    //найти в сообщении ":""
    value = s.indexOf(":");
    //вырезать три символа после ":"
    //в которых содержится команда
    valueStr = s.substring(value+1,value + 4);
    if (DEBUG){
        Serial.println(socket);
        Serial.println("MSG - OK");
    }
    //Если поступила команда hlp (help)
    if (valueStr == "hlp"){
        sendMsg(msgHelp,socket,DEBUG);
    }
}
```

В конце скетча добавлены команды, передающие введенные символы в окне Монитор порта на ESP8266, что позволяет вводить AT команды для управления ESP8266 в "ручном" режиме.

```
//Передать данные из UART (USB) на UART (ESP8266)
while (Serial.available() > 0)
    Serial1.write(Serial.read());
```

17.6 Перспективы использования модуля ESP8266

Помимо управления AT-командами Arduino IDE, после некоторой настройки, позволяет создавать прошивки и прошивать их непосредственно прямо в ESP8266, используя USB-Serial адаптер (USB-TTL конвертер) точно так же, как вы это делаете с другими платами Arduino. Кроме того, вы сможете использовать практически все библиотеки для Arduino с ESP8266. Поскольку описание такой возможности выходит за отведенные пределы данной книги, заметим лишь, что для прошивки в ESP8266 скетчей из Arduino IDE нужно дополнительно приобрести и воспользоваться USB-TTL конвертером на 3,3 Вольт. Существуют модели с возможностью переключения на 5 вольт и 3,3 вольт. Обратите внимание, что USB-TTL конвертеры на 5 вольт не подходят, т.к. могут повредить модуль ESP8266!

С модулем ESP8266 вы можете использовать не только telnetсервер. В примерах скетчей для этого модуля приводится множество интересных вариантов программирования, которые вы можете добавить в свои разработки. Модуль ESP8266 может использоваться и самостоятельно, т.к. у него есть два цифровых порта, к которым можно подключить цифровые датчики или исполнительные устройства. Однако в связке с Arduino потенциал этого модуля гораздо шире.

*Желаем вам интересных открытий и проектов
в мире "Интернет вещей"!*



Учебно-методическое издание

Артём Бачинин
Василий Панкратов
Виктор Накоряков
под редакцией
Сергея Косаченко

ОСНОВЫ ПРОГРАММИРОВАНИЯ МИКРОКОНТРОЛЛЕРОВ

Учебно-методическое пособие
к образовательному набору
по микроэлектронике
«Амперка»

ОБРАЗОВАТЕЛЬНЫЙ
РОБОТОТЕХНИЧЕСКИЙ МОДУЛЬ

(БАЗОВЫЙ УРОВЕНЬ)
12 – 15 ЛЕТ

Издательство «ЭКЗАМЕН»
«ЭКЗАМЕН-ТЕХНОЛАБ»

Гигиенический сертификат
№ РОСС RU. AE51. Н 16678 от 20.05.2015 г.

Главный редактор Л. Д. Лаппо

Корректор Баринская И. Д.

Дизайн обложки

и компьютерная верстка А. А. Винокуров

107045, Москва, Луков пер., д. 8.

E-mail: по общим вопросам: robo@examen-technolab.ru;
www.examen-technolab.ru

по вопросам реализации: sale@examen-technolab.ru
тел./факс +7 (495) 641-00-19 (многоканальный)



Амперка



EXAMEN-TECHNOLAB



EXAMEN®

www.examen-technolab.ru

Артикул ТВ-0441-М-1

ISBN 978-5-377-10297-7

9 785377 102977

ОЗНАКОМИТЕЛЬНЫЙ САЙТ EXAMEN-TECHNOLAB.RU

12-15
ЛЕТ

