Стартовый набор Starter Kit UNO R3 Arduino

Базовое руководство пользователя

Arduino UNO R3 – это плата-микроконтроллер, имеющая 14 цифровых входов/выходов (из которых 6 могут быть использованы как ШИМ-выходы), 6 аналоговых входов, 16 МГц керамический резонатор, USB интерфейс, силовой вход, ICSP программатор и кнопку сброса. Комплект включает все необходимое для начала работы с микроконтроллером – достаточно подключить плату к USB разъему компьютера или запитать устройство адаптером/батарейками. Работа с набором не требует пайки.

Оглавление

1. Собираем электрическую цепь с мигающим светодиодом

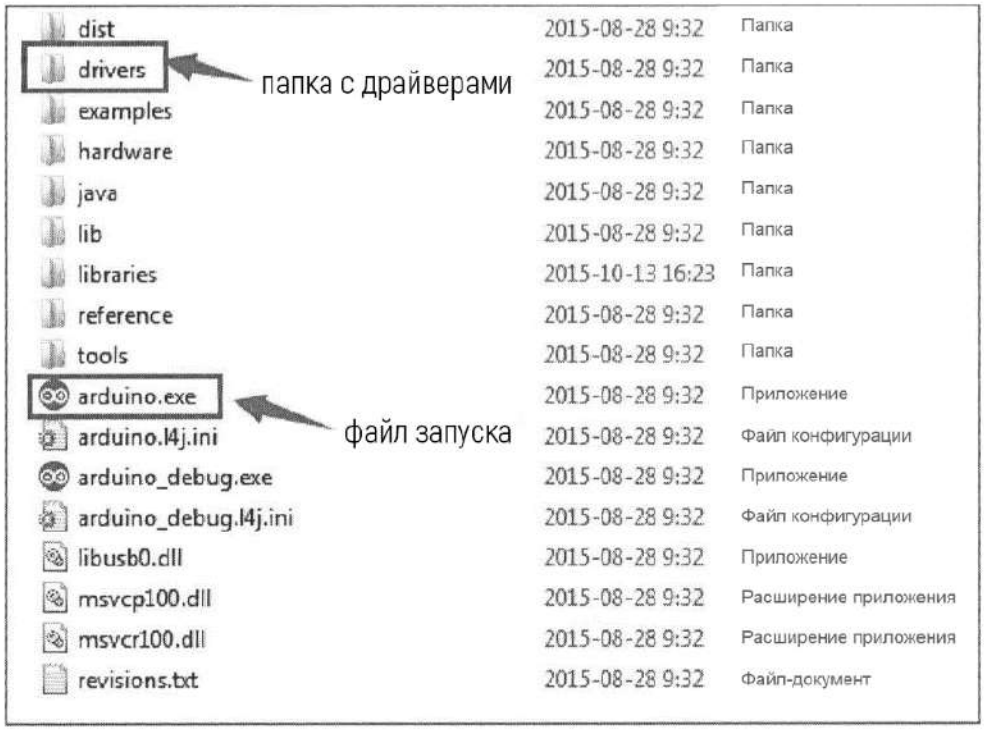
Прежде, чем приступить к сборке электрической цепи, необходимо определить основные параметры её элементов, такие как номинальное напряжение, номинальный ток и т.д.

Для первого опыта мы используем параметры светодиода, найденного в интернете, с диапазоном рабочего напряжения 1,5-2,0 В, рабочего тока 10-20 мА и обратным напряжением 5 В. Напряжение питание микроконтроллера 5 В. Опираясь на данные параметры, выбираем светодиод с рабочим напряжением 1,7 В и рабочим током 15 мА. Сопротивление токоограничивающего резистора рассчитывается как отношение разности общего напряжения в цепи и напряжение светодиода к току в цепи. Таким образом, сопротивление равно (5-1,7)/0,015 = 220 Ом.

В начале работы необходимо скачать программное обеспечение с официального сайта Arduino по ссылке: http://arduino.cc/en/Main/Software

Автор данного руководства работает на платформе Windows 7 (32-bit). Если вы используете другую платформу, скачивайте версию, которая подходит именно вам.

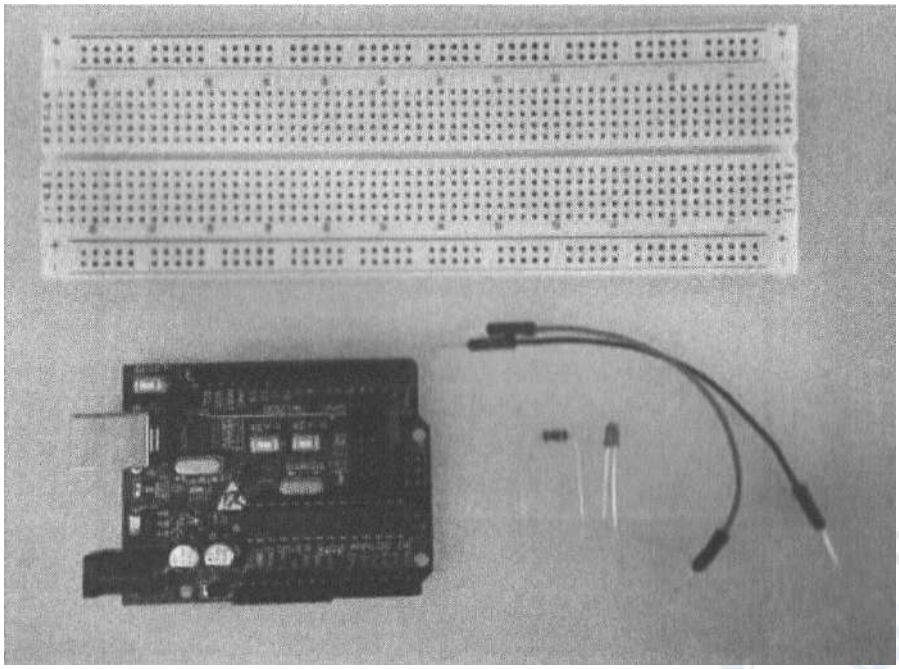
После скачивания и распаковки в корне папки должно быть несколько подпапок и файлов, как показано на скриншоте ниже. Arduino.exe – это загрузочный файл; папка drivers содержит драйверы для USB-контроллера. После того, как вы подключите USB, и вам потребуется указать путь к драйверу, можно указать путь в эту папку.

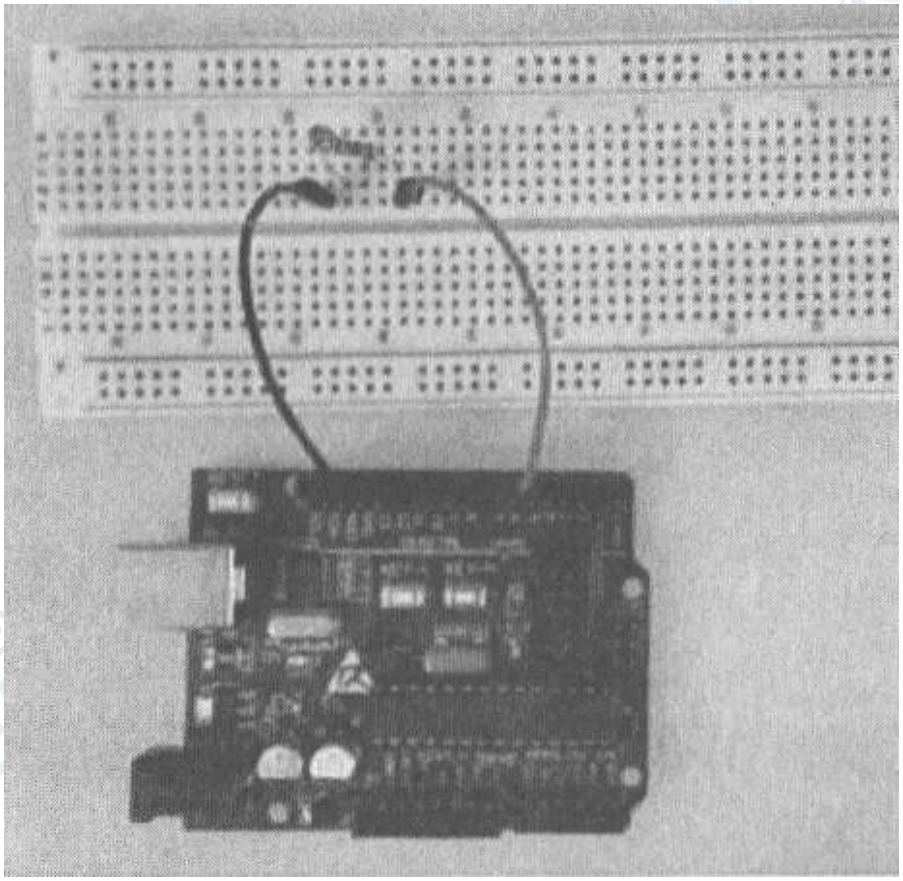


dist	2015-08-28 9:32	Папка
drivers	2015-08-28 9:32	Папка
examples	2015-08-28 9:32	Папка
hardware	2015-08-28 9:32	Папка
java	2015-08-28 9:32	Папка
lib	2015-08-28 9:32	Папка
libraries	2015-10-13 16:23	Папка
reference	2015-08-28 9:32	Папка
tools	2015-08-28 9:32	Папка
arduino.exe	2015-08-28 9:32	Приложение
arduino.l4j.ini	2015-08-28 9:32	Файл конфигурации
arduino_debug.exe	2015-08-28 9:32	Приложение
arduino_debug.l4j.ini	2015-08-28 9:32	Файл конфигурации
libusb0.dll	2015-08-28 9:32	Приложение
msvcp100.dll	2015-08-28 9:32	Расширение приложения
msvcr100.dll	2015-08-28 9:32	Расширение приложения
revisions.txt	2015-08-28 9:32	Файл-документ

Оборудование, которое мы будем использовать в данном опыте, показано на снимке ниже: это макетная плата, светодиод, резистор 1кОм и 2 провода.

На втором снимке представлен способ подключения. У светодиода две контактных ножки: коротая и длинная. Короткая ножка подключается в GND, а длинная подключается к «плюсу». Перед подключением длинной ножки необходимо присоединить резистор на 220 Ом к цифровому контакту №5.



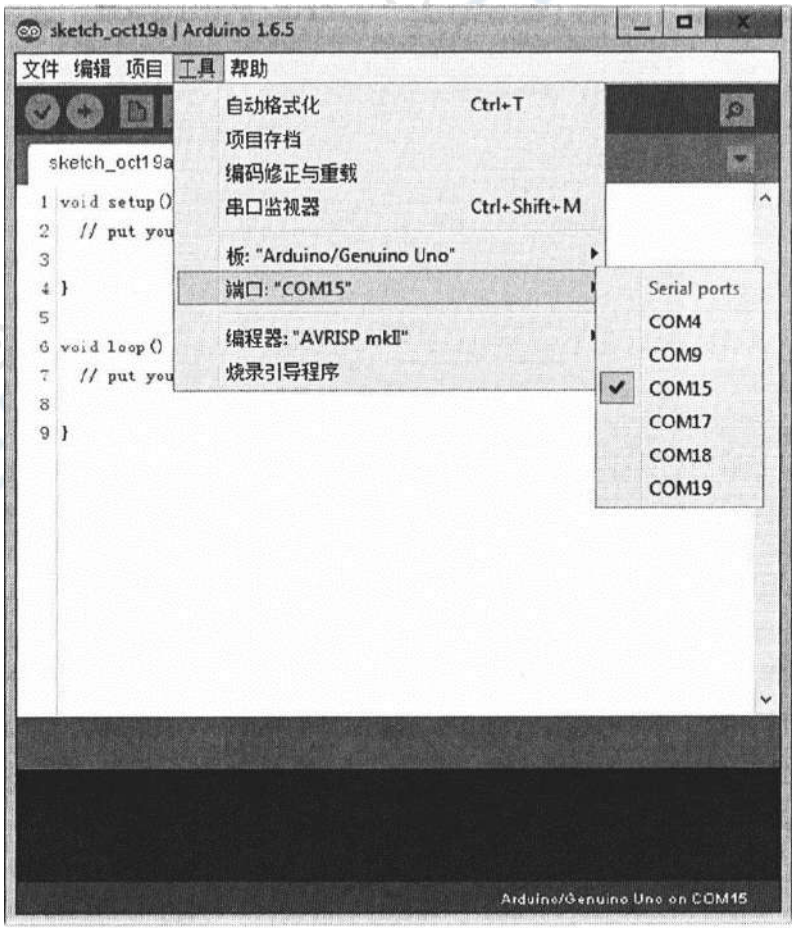


Все элементы подключаются так, как показано на снимке сверху.

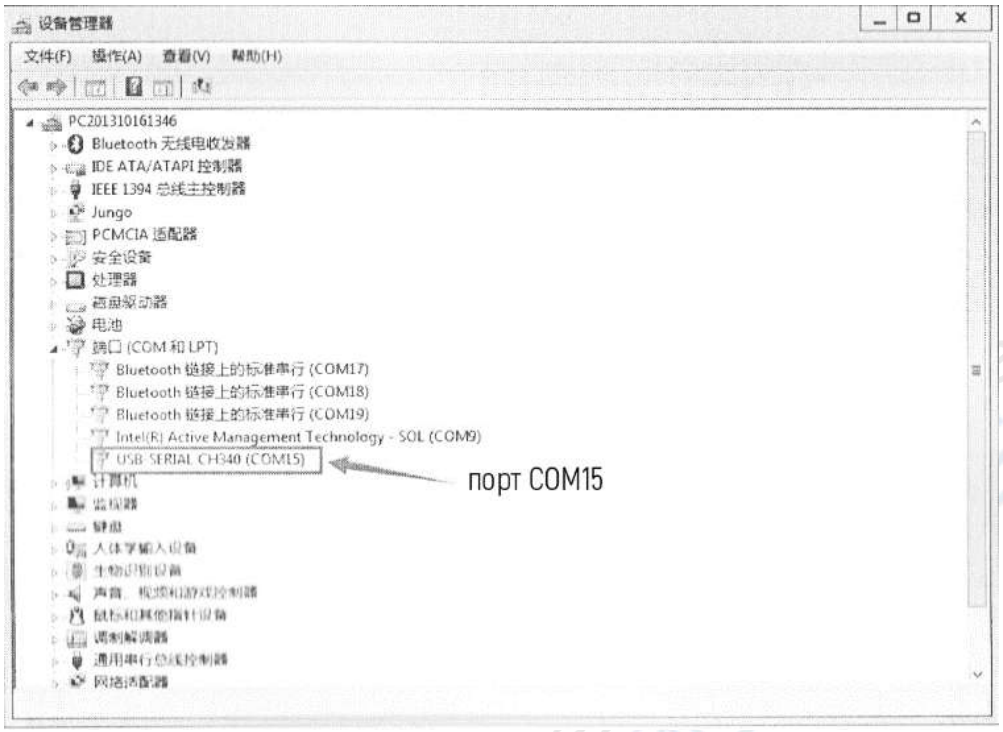
После того, как все контактные вводы элементов соединены через макетную плату, подключите линию USB и настройте все контакты и вводы макетной платы.

Возможно, мои объяснения сперва покажутся не очень понятными, поэтому просто соберите схему так, как показано на снимке. По мере выполнения задач вы будете все больше и больше вникать в суть происходящего. Перед написанием программы нам нужно определить вводы макетной платы, как указано на скриншотах ниже:

После выбора ввода необходимо определить номер последовательного порта. Автор работает с последовательным портом COM15:



Вы можете уточнить номер используемого последовательного порта в панели управления устройствами, как показано на скриншоте ниже:

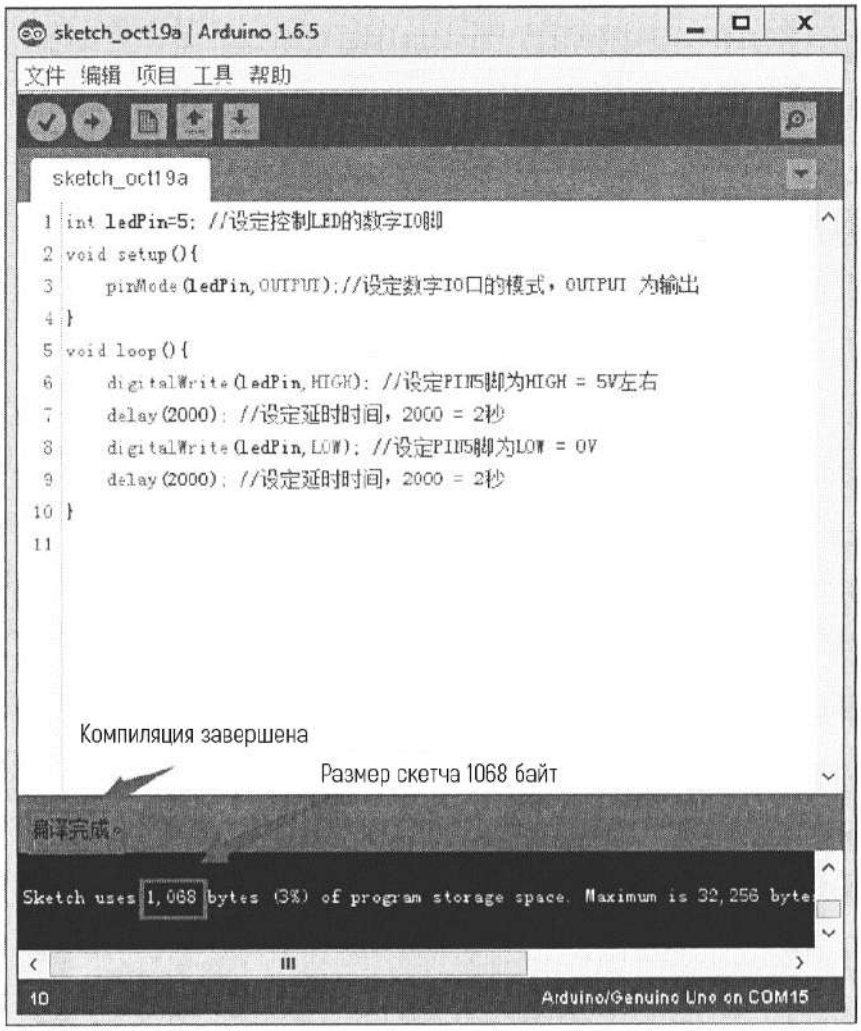


Сначала скопируйте в окно код программы (скетча):

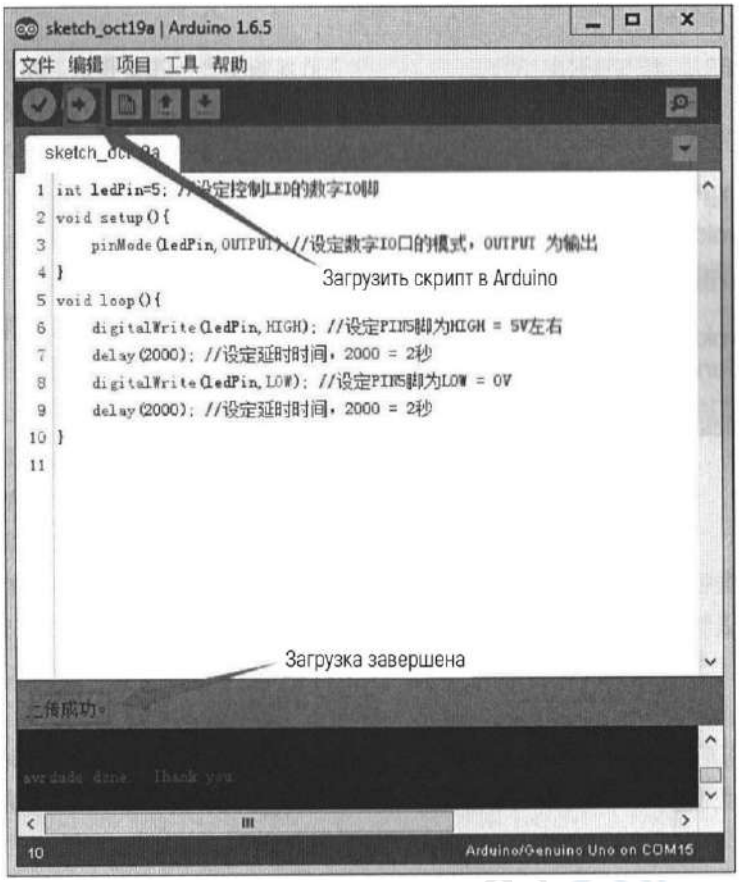
```
int ledPin=5; // номер цифрового контакта IO светодиода
void setup(){
        pinMode(ledPin,OUTPUT); // обозначаем контакт как «выход»
}
void loop(){
        digitalWrite(ledPin,HIGH); // подаем на контакт PIN5 напряжение 5 В
        delay(2000); // указываем время задержки, 2000 = 2 сек
        digitalWrite(ledPin,LOW); // подаем на контакт PIN5 напряжение 0 В
        delay(2000); // указываем время задержки, 2000 = 2 сек
}
```

Строки коричневого цвета вроде int; void **setup** – это системные команды; надписи вроде OUTPUT определяют функционал команды; черный цвет используется для переменных и параметров.

После компиляции программы в нижнем окне появится сообщение о размере файла. Файл со скетчем для данного опыта занимает 1068 байт.



Чтобы загрузить скетч со скомпилированной программой в контроллер Arduino, нажмите кнопку загрузки.



По окончании загрузки появится сообщение, что загрузка успешно завершена.

Если вы сделали все правильно, светодиод начнет мигать.

Резюмируем:

int; void **setup** и т.д. – это системные команды коричневого цвета; OUTPUT и другие синие надписи обозначают функционал команды. Черный цвет используется для переменных и параметров.

Строка «int ledPin=5;» определяет управляемые контакты светодиода, в данном случае номер управляемого контакт «5», а ledpin – это название переменной, которое можно заменить на «xxx» и любое другое. Чтобы управлять контактами IO в программе, каждому контакту присваивается переменная со своим именем. Так их гораздо проще идентифицировать в сложном коде программы.

2. Классическая программа «Hello, world!»

Как проверить работоспособность последовательного порта с помощью Arduino?

Простейший пример:

```
void setup() {
        Serial.begin(9600); // включает COM-порт, частота 9600 бит/с
}
void loop() {
Serial.println("Hello world!");

delay(1000);
}
```

Если схема собрана корректно, номер порта выбран верно и программа успешно загружена, то после нажатия кнопки Serial Monitor на компиляторе в окне Serial Monitor появится надпись «Hello world!».

3. Основные функции Arduino

В процессе изучения языка необходимо посвятить один урок разбору его структуры и синтаксиса. Существует несколько основных функций:

1. Функция объявления переменных (int val; int lepPin=13;).
2. Функция setup() вызывается тогда, когда скетч начинает исполняться. Предназначена для инициализации параметров и режимов работы портов, может обращаться к базам данных и т.д. (пример: pinMode(ledPin, OUTPUT);).
3. Функция loop() – после вызова setup() и инициализации параметров, функция loop() запускает цикл подпрограммы. Используется для непрерывной работы Arduino.

Ниже описаны самые распространенные функции, которые необходимо изучить.

1. pinMode (имя порта, OUTPUT/ INPUT) – определяет, является контакт входным или выходным, прописывается внутри функции setup().

2. digitalWrite (имя порта, HIGH/LOW) – определяет, включен или отключен цифровой контакт.

3. digitalRead (имя порта) – считывает значение переменной цифрового контакта.

4. analogWrite (имя порта, значение) – определяет величину аналогового сигнала (ШИМ-волна). Для Arduino на базе микроконтроллеров ATmega168 (в том числе Mini или BT) данная функция поддерживается для цифровых контактов 3 5, 6, 9, 10 и 11. Для более старых версий USB на базе ATmega8 и serial Arduino поддерживается работа с контактами 9, 10 и 11.

5. analogRead (имя порта) – считывает сигнал с указанного аналогового порта. Для аналоговых сигналов в Arduino используется 10-битный АЦП, что позволяет преобразовывать входное напряжение 0-5 В в целочисленное количество отсчетов от 0 до 1023.

6. delay() – устанавливает значение задержки, delay(1000) – задержка в 1 секунду.

7. Serial.begin (скорость передачи данных) – задает скорость передачи данных (бит/с) через последовательный порт. Для передачи данных на компьютер поддерживаются скорости: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 и 115200. Данные

значения устанавливаются по желанию пользователя в любом порядке. Пример: для контакта 0 или 1 необходимо задать отдельную скорость передачи данных. Прописывается в теле функции setup().

8. Serial.read () – считывает выходные данные.

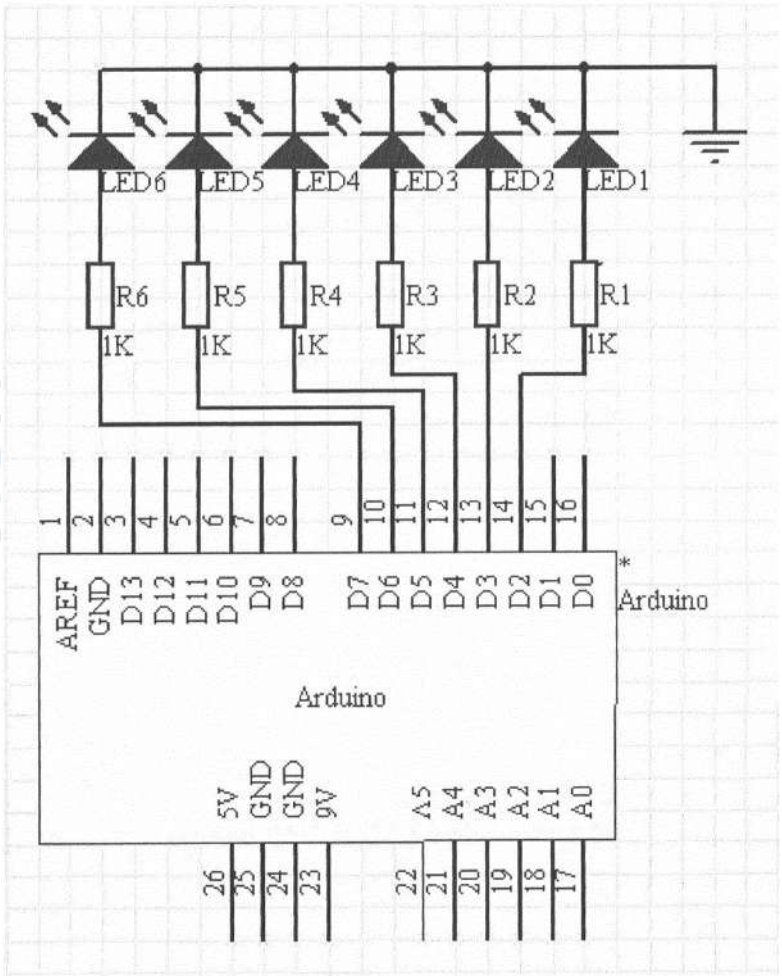
9. Serial.print (данные, система счисления) – выводит данные через последовательный порт. Serial.print (данные) – используется по умолчанию для данных в десятичной системе Serial.print (данные, DEC).

10. Serial.println (данные, система счисления) - выводит данные через последовательный порт с последующими за ними символом переноса строки и символом новой строки. Обладает теми же свойствами, что и функция Serial.print().

Указанные выше функции являются наиболее важными при работе с Arduino. Все остальные функции мы будем разбирать по мере изучения.

4. Цифровые выводы Arduino

Цифровые выводы I/O Arduino располагаются с одной стороны микроконтроллера, сверху и снизу находится по 6 контактов (пинов), это пины с 2-ого по 7-ой и пины с 8-ого по 13-ый. В 13-ый пин мы подключаем резистор 1 кОм, а все остальные напрямую подключаем к контроллеру ATmega. Чтобы проверить функциональность всех выводов I/O Arduino, мы используем бегущие огни на шести светодиодах. Принципиальная схема показана ниже:

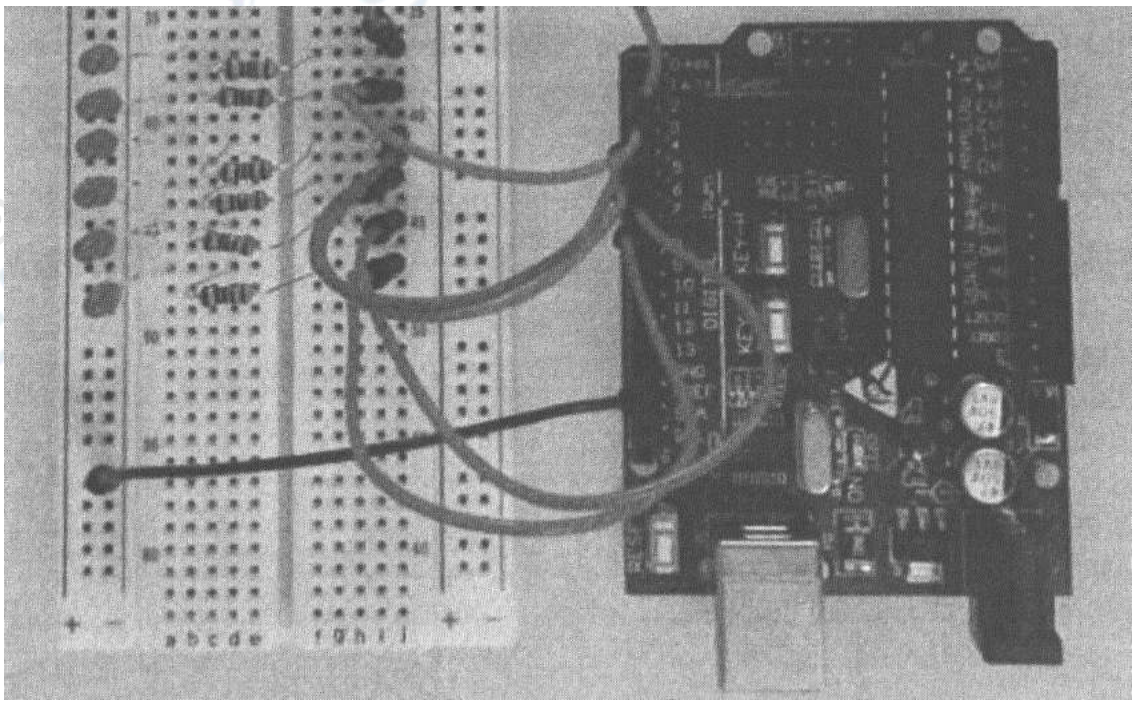


Чтобы увеличить эквивалентное сопротивление светодиодов, снизить протекающий в ветвях ток и исключить вероятность выхода светодиодов из строя, в каждую ветвь мы подключаем резисторы 1 кОм.

Скетч программы для данного опыта:

```
int startPin = 2;
int endPin = 7;
int index = 0;

void setup(){
for (int i = startPin; i < = endPin; i + +){
        pinMode(i, OUTPUT);
}
}

void loop(){
for (int i = startPin; i <= endPin; i + +){
        digitalWrite(i,LOW);
}
digitalWrite(startPin + index, HIGH);
index = (index + 1) %(endPin-startPin+1);
delay(100);
}
```

Загрузите и запустите скетч. Бегущие огни с шестью светодиодами, подключенными между пинами 2 и 7, должны загораться на 0,1 секунды и гаснуть.



Эту схему применяют для проверки работоспособности выводов I/O. Микроконтроллер Arduino оснащен двенадцатью цифровыми пинами, и подобным образом

можно проверить остальные шесть контактов. Для этого необходимо собрать схему, аналогичную той, что указана выше, но подключиться к верхним шести контактам.

5. Последовательные порты Arduino

Прием и передачу данных при подключении к ПК или микроконтроллеру проще всего осуществлять через последовательные порты (COM-порты). В старых ПК обычно предусматривалось последовательное подключение по стандартным интерфейсам RS-232 или RS-422, сейчас эта концепция несколько изменилась и для быстрой передачи данных стало гораздо удобнее использовать USB, однако сам процесс последовательного подключения немного усложнился. Несмотря на то, что в некоторых современных ПК интерфейсы RS-232 или RS-422 отсутствуют, в устройствах такого типа мы можем провести последовательное подключение с помощью переходников USB-COM или PMCIA-COM.

Через последовательные порты ПК и Arduino взаимно обмениваются данными. Например, раньше при работе со звуком или видео на ПК во многих случаях требовалось, чтобы Arduino мог осуществлять идущие с ПК команды через последовательные порты и обладал соответствующим функционалом. Теперь все эти функции можно прописать с помощью языка Arduino внутри Serial.read().

В этом опыте нам не требуется собирать электрическую схему, мы будем изучать передачу данных между ПК и Arduino через последовательные порты. Скетч программы:

```
int ledPin = 13;
int val;

void setup(){
pinMode(ledPin, OUTPUT);
Serial.begin(9600);
}

void loop(){
val = Serial.read();
if (-1 !=val) {
        if ('H' == val) {
                digitalWrite(ledPin, HIGH);
                delay(500);
                digitalWrite(ledPin, LOW);
        }
}
}
```

После загрузки скетча в Arduino в интегрированной среде разработки откроется окно последовательного порта, скорость передачи данных устанавливается 9600, на модуль Arduino передается символ H.

После запуска можно редактировать данные функции Serial.read() без остановки программы. Язык Arduino не блокирует данную функцию, таким образом, если данные с последовательного порта не достигают своего назначения, они тут же возвращаются обратно. Функция Serial.read() каждый раз считывает один байт из последовательного соединения, и когда данные достигают ПК или контроллера, то функция возвращает им

значение в соответствии с кодировкой ASCII. Когда с последовательного порта не поступает никаких данных, функция возвращает значение -1.

В справочнике по языку Arduino нет достаточного пояснения к функции Serial.read(). И вот, что мне не совсем понятно: если количество данных, переданных с ПК за раз, слишком велик, обеспечивает ли Arduino сохранение тех данных, которые не помещаются в указанный размер, в буфере последовательного соединения и не теряются ли они?

В языке Arduino есть функция Serial.available(), и она больше подходит для следующего опыта:

```
int ledPin = 13;
int val;

void setup(){
pinMode(ledPin, OUTPUT);
Serial.begin(9600);
}

void loop(){
val = Serial.read();
if (-1 ! = val) {
        if ('H' == val) {
                digitalWrite(ledPin, HIGH);
                delay(500);
                digitalWrite(ledPin, LOW);
Serial.print("Available: ");
                Serial.println(Serial.available(), DEC);
        }
}
}
```

Назначение функции Serial.available() - возвращать оставшиеся текущие данные из буфера последовательного порта. Из описания функции, предоставленного Arduino, мы знаем, что максимальное количество байтов, которое может храниться в буферной памяти последовательного порта – 128 байтов. Чтобы проверить эту функцию, мы передадим множество символов на модуль Arduino за одну интерацию.

В этом опыте всякий раз, когда Arduino принимает один символ H, подсоединенный к цифровому контакту 13 светодиод загорается.

6. Последовательные выводы Arduino

Во многих случаях нам необходимо, чтобы между Arduino и другим устройством осуществлялся взаимный обмен данными, а самый простой и распространённый способ передачи – это передача данных через последовательное соединение. При такой форме обмена цифровые импульсы передаются между двумя устройствами в строго определенном протоколом порядке, что обеспечивает полноту и достоверность всех данных.

Самый распространенный протокол передачи в ПК – это последовательное соединение RS-232. В различных микроконтроллерах применяются соединения с протоколом TTL. При соединении ПК и микроконтроллеров необходимо помнить о существенных различиях в уровнях мощности этих протоколов. Чтобы настроить передачу

данных меду уровнями RS-232 и TTL обычно применяются микросхемы типа MAX232 и т.д. В Arduino передача данных на соответствующих уровнях мощности предусмотрена по умолчанию.

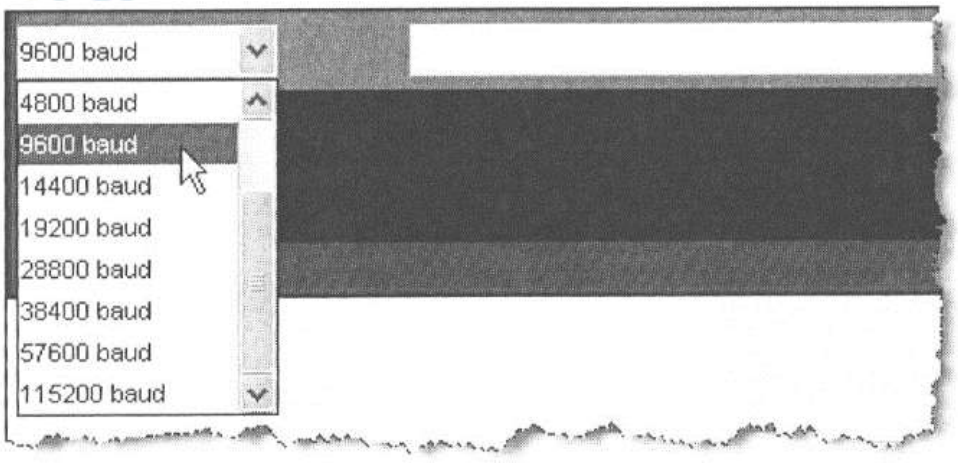
На принципиальной схеме Arduino не трудно заметить, что контактные ножки ATmega RX и TX одной стороной напрямую подключаются к цифровым пинам I/O 0 и 1, а другие ножки посредством схемы переключения мощности подключены в последовательный порт. Таким образом, если нужно организовать передачу данных между ПК и Arduino, модно соединить их через последовательные порты; когда нам нужно организовать связь между Arduino и микроконтроллером (например, другим модулем Arduino), можно соединить их через цифровые пины 0 и 1.

Трудность соединения через последовательные порты заключается в настройках параметров, таких как скорость передачи данных, количество битов данных, стоповых битов и т.д. Для упрощения задачи в языке Arduino есть функция Serial.begin(). Чтобы осуществить передачу данных, Arduino предлагает функции Serial.print() и Serial.println(). Разница между ними в том, что принимающее устройство после запрошенных данных добавляет знак переноса строки для удобочитаемости конечного результата.

В этом опыте мы не задействовали никаких дополнительных схем, нам потребуются только последовательные провода для подключения ПК и Arduino. Соответствующий скетч:

```
void setup(){
        Serial.begin(9600);
}

void loop(){
        Serial.println("Hello World!");
        delay(1000);
}
```

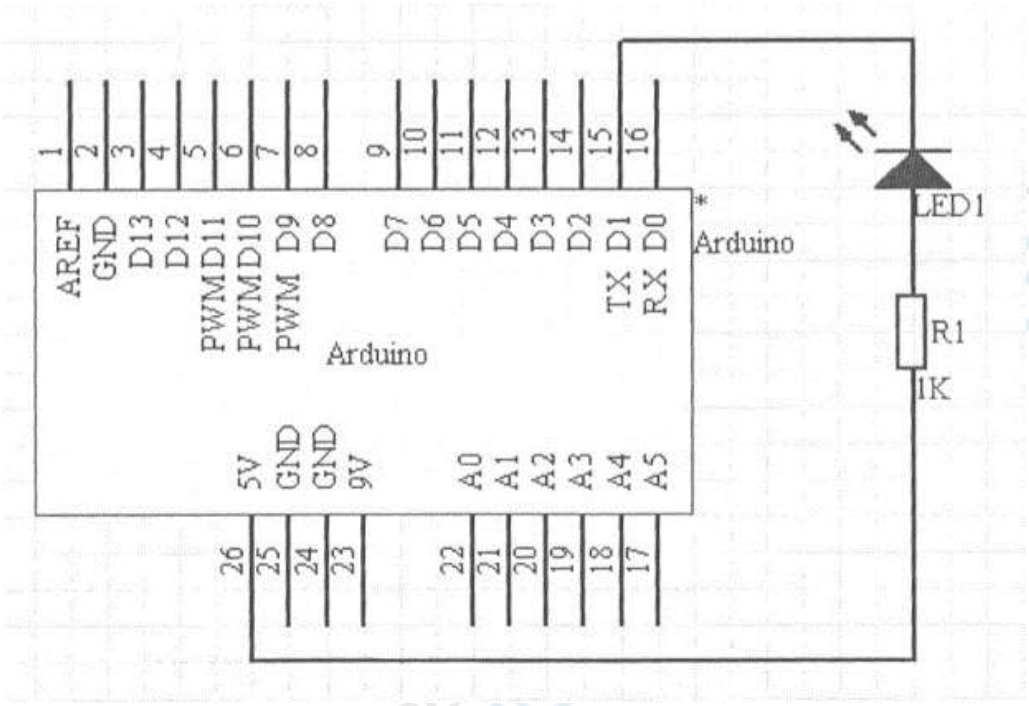
Загрузите скетч в модуль Arduino, после запуска в интегрированной среде разработки нажмите кнопку “Serial Monitor” на панели инструментов, откроется окно последовательного порта:



Затем выберите скорость передачи данных 9600, чтобы обеспечить согласование настроек в скетче.

Если все работает нормально, в окне Console интегрированной среды разработки Arduino мы увидим выходные данные последовательного порта.

Чтобы проверить, корректно ли передаются данные через последовательный порт, самый простой способ - подключить светодиод между цифровым пином 1 (TX) и источником питания 5 В, как показано на схеме ниже:



В такой схеме, когда Arduino будет передавать данные на ПК через последовательный порт, светодиод будет гореть. Этот способ очень удобен способ для отладки схемы.

7. Аналоговые вводы Arduino

Используются для считывания аналогового сигнала с определенного вывода. На корпусе Arduino есть 6 каналов (Mini и Nano предусматривают 8 каналов, Mega – 16 каналов), 10 AD преобразователей (модулей). Это означает, что выходное напряжение 0-5 В соответствует 0-1023 байтам. Иными словами, точность чтения составляет 5 В/1024 бита, что в среднем равняется 0.049В на 1 бит (4.9 мВ/1 бит). Диапазон выходных значений и шаг можно задавать с помощью функции analogReference().

Цикл считывания аналоговых выводов 100 микросекунд (0.0001 сек), поэтому максимальная скорость чтения составляет 10.000 бит в секунду.

pin: номер контактной ноги аналогового вывода, с которого происходит считывание (большинство модулей Arduino оснащены пинами A0-A5, Mini и Nano – A0-A7, Mega – A0-A15).

Returns возвращает значение функции.

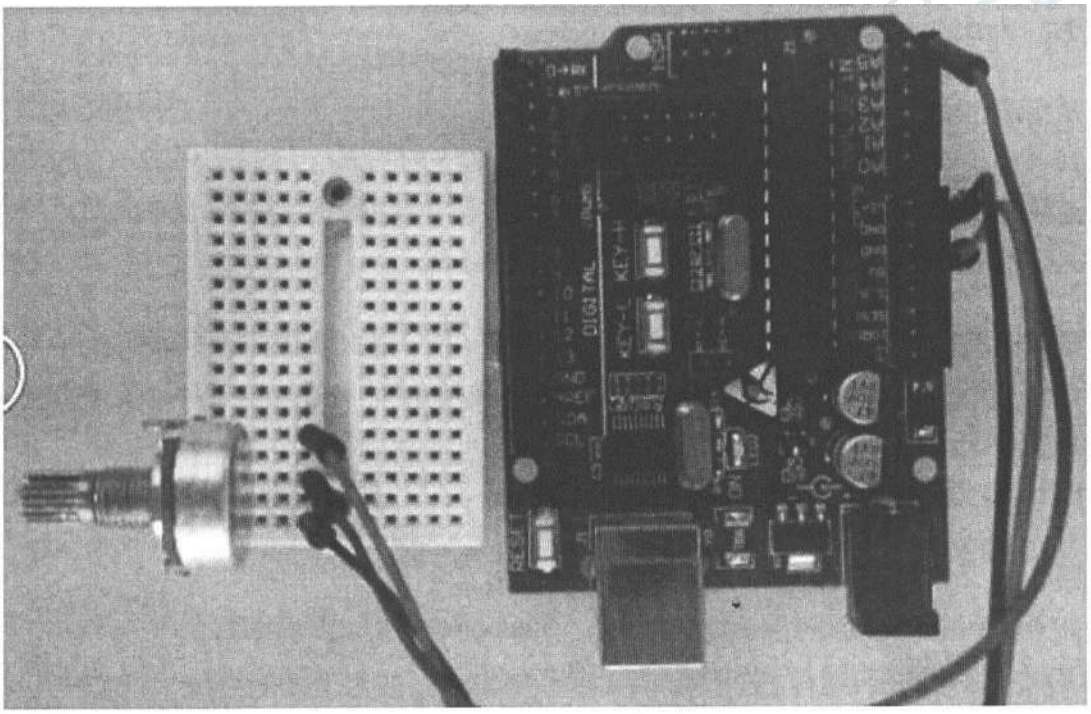
int (от 0 до 1023).

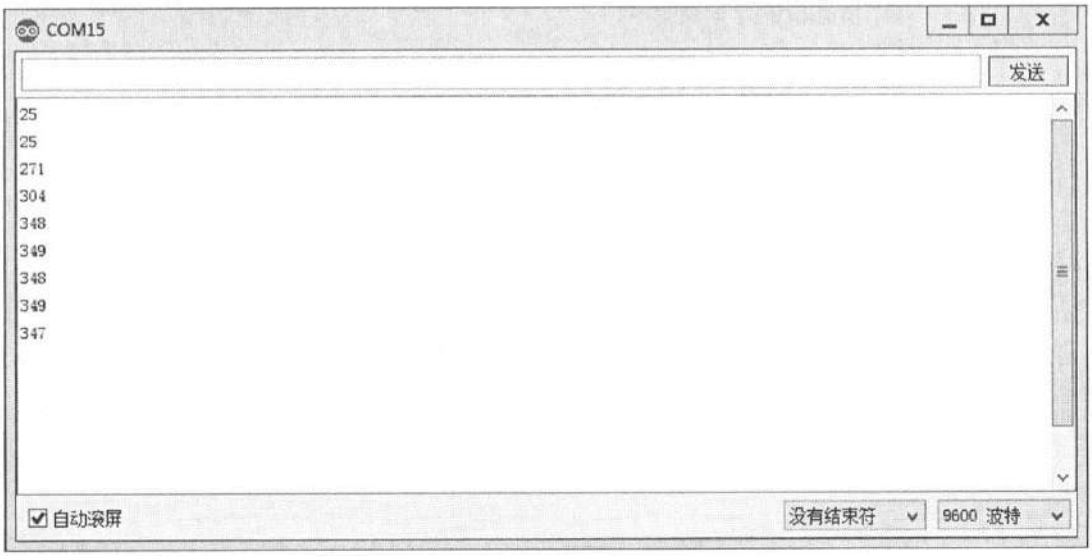
Целочисленный int (от 0 до 1023).

Если контактная ножка аналогового вывода ни к чему не подключена, возвращаемое значение функции analogRead() будет колебаться в зависимости от внешних возмущений

(например, работы других аналоговых выводов, прикосновения руки или помех от устройств вблизи модуля)

```
int analogPin = A5;
int val = 0;
void setup(){
        Serial.begin(9600);
}
void loop(){
        val = analogRead(analogPin);
        Serial.println(val);
Delay(1000);
}
```

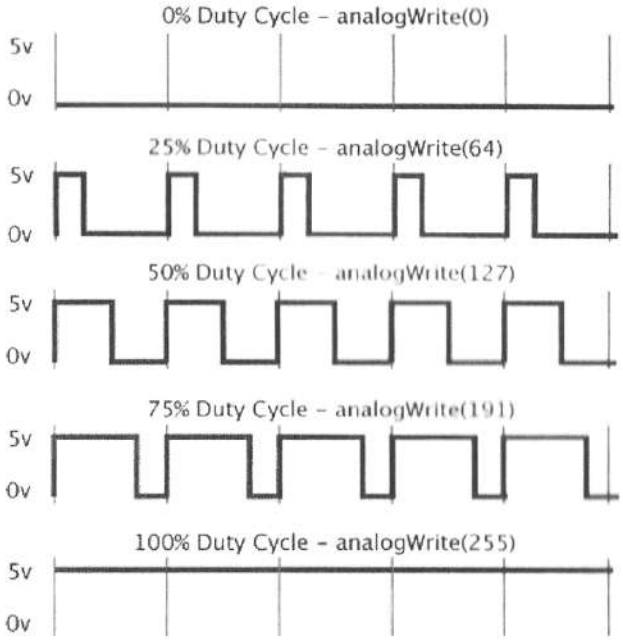




8. Аналоговые выводы Arduino

Широтно-импульсная модуляция или ШИМ – это технология получения аналогового сигнала из среднего значения цифрового сигнала. Цифровой сигнал используется для создания прямоугольной волны и переключается между состояниями «ключ открыт» и «ключ закрыт». При такой модели переключателя отношение интервалов времени в положениях «закрыт» и «открыт» моделирует напряжение между 5 В («ключ открыт») и 0 В («ключ закрыт»). Время, которое ключ находится в открытом состоянии, называется «шириной импульса». Чтобы получить различные аналоговые сигналы, можно регулировать ширину импульсов. Если вы увеличите скорость переключения, яркость светодиода будет управляться устойчивым промежуточным напряжением между 0 В и 5 В.

На рисунке ниже зеленые линии обозначают установленный интервал времени. Длительность или период – это обратная величина частоты ШИМ. Другими словами, средняя частота ШИМ Arduino 500 Гц, каждая зеленая линия соответствует 2 миллисекундам. Интервал вызова функции analogWrite() изменяются от 0 до 255 и называется коэффициентом заполнения. Например, analogWrite(255) задает полное заполнение (ключ всегда открыт), analogWrite(127) задает 50%-ное заполнение (ключ открыт половину всего времени).



Широтно-импульсная модуляция

Когда вы запустите данный пример, следите за обратными колебаниями Arduino. Для наших глаз каждое паразитное колебание превращается в мерцание светодиода. Из-за потерь в светодиоде длины некоторых импульсов могут увеличиться и сокращаться, и таким образом мы можем увидеть разницу в ширине импульсов.

Аналоговый сигнал (ШИМ-волна) выводится на контактную ножку. ШИМ применяется для регулирования яркости светодиодов, для плавного запуска двигателей, регулирования мощности и т.д.. После вызова analogWrite() с контактного ножки будет производиться прямоугольная волна установленной ширины до следующего вызова analogWrite() (или вызова функций digitalRead() и digitalWrite() для того же контакта). Частота ШИМ составляет 490 Гц.

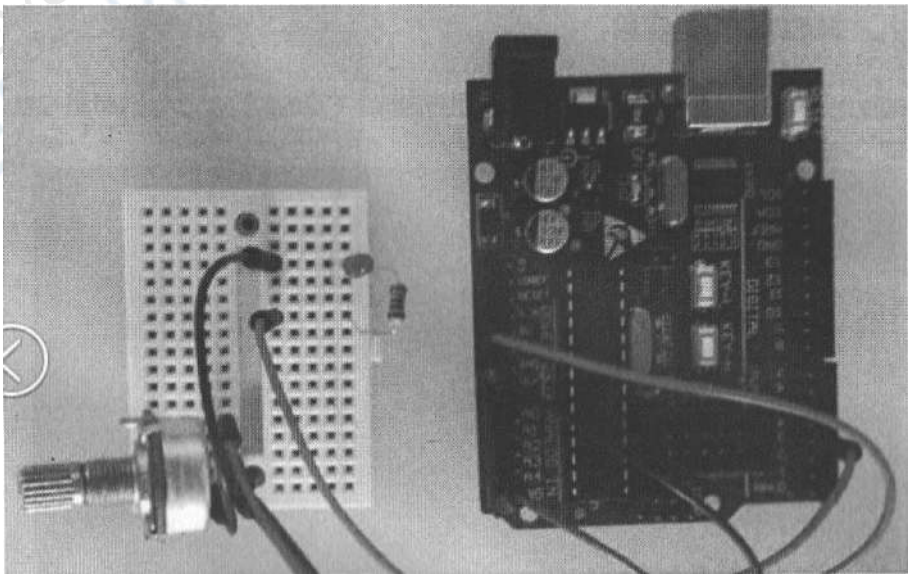
В большинстве модулей Arduino (ATmega168 или ATmega328) эти функции поддерживаются для пинов 3, 5, 6, 9, 10 и 11. В ArduinoMega они работает для пинов с 2 по 13. В более старых модулях Arduino на базе ATmega8 для данной функции поддерживаются пины 9,10 11. Таким образом, для аналогового вывода не нужно прописывать вызов функции pinMode() перед вызовом функции analogWrite().

Этот метод работы с аналоговыми выходами имеет мало общего с функциями analogWrite и analogRead.

analogWrite(pin, value)
parameters – параметры.
pin: номер выходного пина.
value: коэффициент заполнения, от 0 (всегда закрыт) до 255 (всегда открыт).

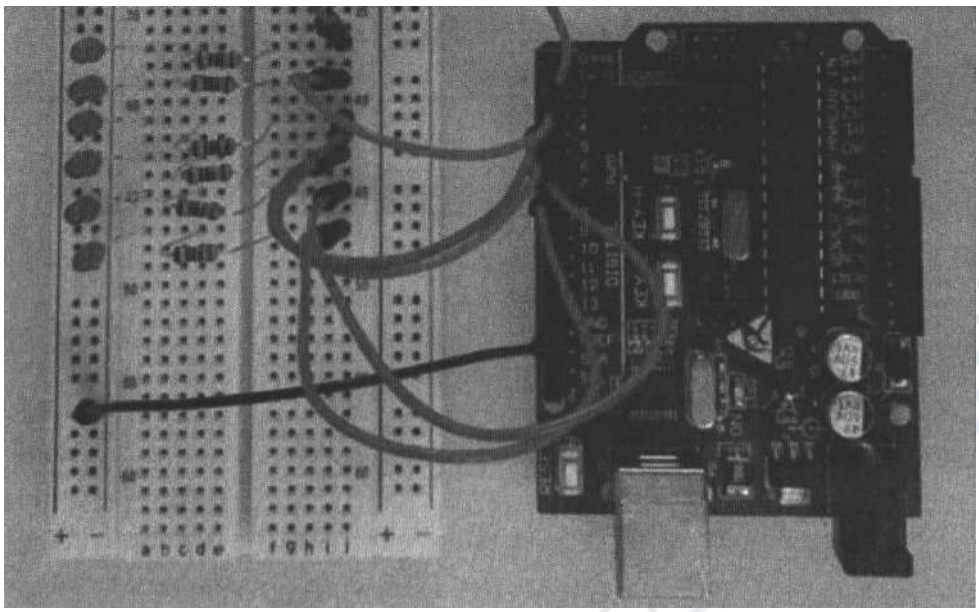
На выходе пинов 5 и 6 создается ШИМ с коэффициентом заполнения выше установленного. Для этого предусмотрены функции milllis() и delay(), которые обеспечивают взаимодействие выходным ШИМ для устройства в одно и то же время. Стоит помнить, что в большинстве случаев для пинов 5 и 6 при низких значениях коэффициента заполнения (например, 0-10) и даже при нуле сигнал не всегда будет отсутствовать.

```
int ledPin = 3;
int analogPin = A5;
int val = 0;
void setup()
{
        pinMode (ledPin, OUTPUT);
}
void loop()
{
        val = analogRead(analogPin);
        analogWrite(ledPin, val/4);
}
```



9. Опыт с бегущими огнями на 6-ти светодиодах

В предыдущем пункте мы изучили, как заставить светодиод мерцать. Следующий опыт посвящен управлению шестью светодиодам. Посмотрите на схему соединений:



После сборки цепи загрузите скетч программы, состоящий из двух частей, и посмотрите, чем они отличаются. С помощью скетчей мы подробнее познакомимся со структурой языка Arduino.

```
//Определяем управляемые контакты светодиода
int Led1 = 1;
int Led2 = 2;
int Led3 = 3
int Led4 = 4;
int Led5 = 5;
int Led6 = 6;
//1-ый стиль индикации светодиода
void style_1(void)
{
  unsigned char j;
  for(j=1; j<=6; j++)
                    {
                    digitalWrite(j,HIGH);
                    delay(200);
          }
           for(j=6; j>=1; j--)
          {
                    digitalWrite(j,LOW);
                    delay(200);
          }
}
void setup()
{
```

```
        unsigned char i;
        for(i=1;i<=6;i++)// пины 1-6 работают как выходные
                pinMode(i, OUTPUT);// i-й пин работает как выходной
}
void loop()
{
        style_1();//стиль 1


}
// Определяем управляемые контакты светодиода
int Led1 = 1;
int Led2 = 2;
int Led3 = 3
int Led4 = 4;
int Led5 = 5;
int Led6 = 6;
//1-ый стиль индикации светодиода
void style_1(void)
{
 unsigned char j;
 for(j=1; j<=6; j++)
        digitalWrite(j,HIGH);
        delay(200);

         for(j=6; j>=1; j--)
        {
                digitalWrite(j,LOW);
                delay(200);
        }
}
void setup()
{
        unsigned char i;
        for(i=1;i<=6;i++)// пины 1-6 работают как выходные
        pinMode(i, OUTPUT);// i-й работает как выходной
}
void loop()
{
        style_1();//стиль 1
}
```

После загрузки скетча первая часть программы организует цикл, в котором светодиоды сперва последовательно загораются с 1-ого по 6-й, а затем гаснут с 6-ого по 1-й.

Во второй части программы организован цикл, в котором все 6 светодиодов загораются одновременно, а затем гаснут в порядке с 6-ого по 1-й. Проанализируем работу функций данной программы.

Снизу приведена подпрограмма цикла for, где j – это номер горящего светодиода. Там же мы задаем задержку 200 миллисекунд, и цикл повторяется снова. В результаты подпрограммы все 6 светодиодов загораются по порядку с задержкой в 100 миллисекунд:

```
for(j=1; j<=6; j++)
        {
        digitalWrite(j,HIGH);
        delay(200);
        }
```

Следующая часть программы составлена не совсем по правилам синтаксиса. Подпрограмма цикла for обычно записывается в фигурных скобках {}. Если скобки отсутствуют, компилятор может автоматически заключить тело цикла в {}. Если в цикле много итераций, перебор может быть затруднен.

```
for(j=1; j<=6; j++)
        digitalWrite(j,HIGH);
        delay(200);
```

Правильный вариант написания цикла представлен ниже:

```
for(j=1; j<=6; j++)      {
        digitalWrite(j,HIGH);
        }
        delay(200);
```

Шесть светодиодов загораются один за другим, затем спустя 200 миллисекунд цикл повторяется. Поскольку скорость переключения светодиодов велика, нам кажется, что они загораются одновременно.

Функция void(без типа) в языке Arduino служит для указания типа данных и часто используется для конкретного события. Если процесс управления односложен, то void не прописывается, а шаблон программы выглядит так:

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

Задает начало или цикл для события.

Если процесс управления включает несколько событий, для каждого события тип void прописывается отдельно.

Загрузите в микроконтроллер следующий код и посмотрите на работу светодиода.

```
// Определяем управляемые контакты светодиода
int Led1 = 1;
int Led2 = 2;
int Led3 = 3
int Led4 = 4;
int Led5 = 5;
int Led6 = 6;
//1-ый стиль индикации светодиода
void style_1(void)
{
 unsigned char j;
 for(j=1; j<=6; j++)// светодиоды 1-6 загораются по порядку с задержкой 200 ms
        {
            digitalWrite(j,HIGH);//загорается j-тый светодиод
            delay(200);//задержка 200ms
        }
        for(j=1; j<=6; j--)// светодиоды 6-1 загораются с задержкой 200 ms
        {
            digitalWrite(j,LOW);//гаснет j-тый светодиод
            delay(200);//задержка 200ms
        }
}
//программа для мигания светодиодов
void flash(void)
{
        unsigned char j,k;
        for(k=0; k<=1; k++)// светодиод мигает дважды
        {
         for(j=1; j<=6; j++)// светодиоды 1-6 загораются по порядку
           digitalWrite(j,HIGH);// digitalWrite(j,HIGH);//загорается j-тый светодиод
           delay(200);//задержка 200ms
         for(j=1; j<=6; j++)// светодиоды 1-6 гаснут по порядку
           digitalWrite(j,LOW);// гаснет j-тый светодиод
           delay(200);//задержка 200ms
        }
}
//2-ый стиль индикации светодиода
void style_2(void)
{
 unsigned char j,k;
 k=1;//начальное значение k равно 1
 for(j=3;j>=1;j--)
 {
        digitalWrite(j,HIGH);//светодиод горит
        digitalWrite(j+k,HIGH);//светодиод горит
        delay(400);//задержка 400 ms
        k +=2;//прибавляем «2» к «k»
}
```

```
k=5;// значение k равно 5
for(j=1;j<=3;j++)
        {
        digitalWrite(j,LOW);//светодиод гаснет
        digitalWrite(j+k,LOW);//светодиод гаснет
        delay(400);//задержка 400 ms
        k -=2;//отнимаем «2» от «k»
        }
}
//3-ый стиль индикации светодиода
void style_3(void)
{
 unsigned char j,k;// для 3-его стиля индикации светодиода
 k=5;// значение k равно 5
 for(j=1;j>=3;j++)
        {
        digitalWrite(j,HIGH);//светодиод горит
        digitalWrite(j+k,HIGH);//светодиод горит
        delay(400);//задержка 400 ms
        digitalWrite(j,LOW);//светодиод гаснет
        digitalWrite(j+k,LOW);//светодиод гаснет
        k -=2;//отнимаем «2» от «k»
        }
k=3;// значение k равно 53
for(j=2; j>=1; j--)
        {
        digitalWrite(j,HIGH);//светодиод горит
        digitalWrite(j+k,HIGH);//светодиод горит
        delay(400);//задержка 400 ms
        digitalWrite(j,LOW);//светодиод гаснет
        digitalWrite(j+k,LOW);//светодиод гаснет
        k +=2;//прибавляем «2» к «k»
        }
}
void setup()
{
        unsigned char i;
        for(i=1;i<=6;i++)// пины 1-6 работают как выходные
        pinMode(i, OUTPUT);// i-й работает как выходной
}
void loop()
{
style_1();//стиль 1
flash();//мигание
style_2();//стиль 2
flash();//мигание
style_3();//стиль 3
flash();//мигание
        }
```

10. Опыт с зуммером

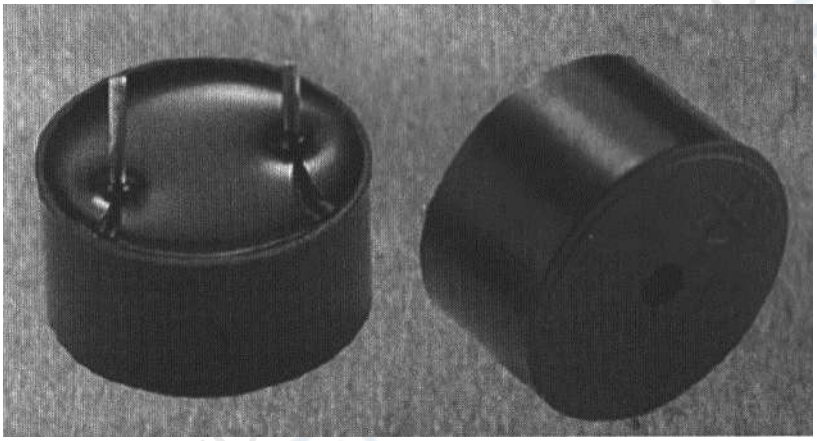
Следующий опыт посвящен работе с миниатюрным зуммером без источника питания.

Исходя из параметров, взятых с интернет-ресурса, рабочее напряжение зуммера 5 В, что совпадает с выходным напряжением цифрового разъема на панели Arduino, поэтому зуммер можно подключать напрямую без дополнительного резистора.

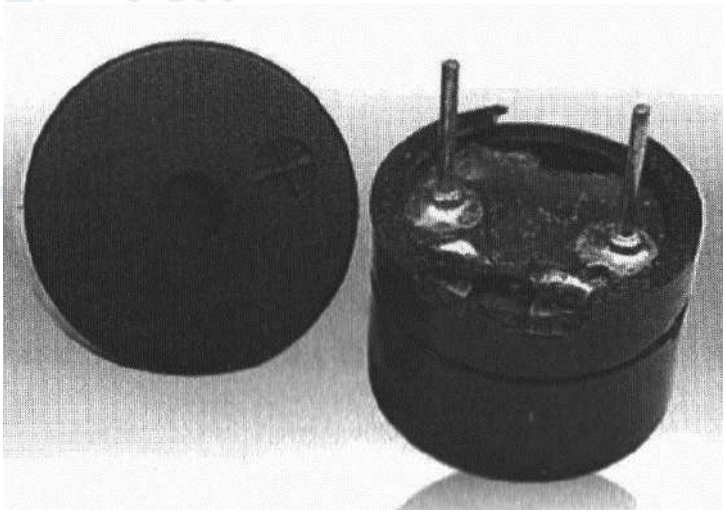
Коротко ознакомимся с миниатюрным зуммером.

Из-за небольших размеров (диаметр всего 6 мм), маленького веса, низкой цены и надежной конструкции миниатюрные зуммеры широко используются в электрооборудовании, электронике, микросхемах и любых других устройствах для звукового оповещения. Зуммеры бывают со встроенными генераторами и без них.

Зуммер со встроенным генератором



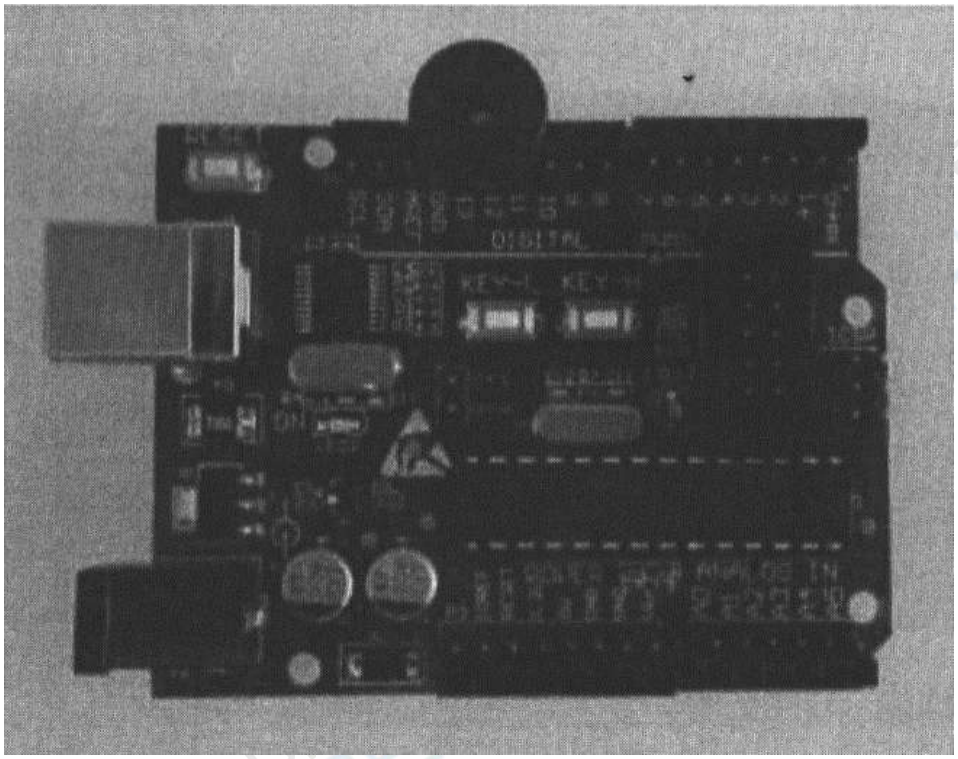
Зуммер без генератора



Внешне эти два типа почти идентичны, но есть разница в подключении зуммеров к пинам микроконтроллера. Зуммер с зеленой микросхемой – без генератора. Зуммер с черной крышкой и без микросхемы – со встроенным генератором. Визуально сложно отличить устройства, самый надежный способ проверки, кроме, разумеется, прочтения инструкции, измерить сопротивление зуммера с помощью универсального мультиметра. У зуммера без генератора сопротивление 8 Ом или 16 Ом, у зуммеров со встроенным генератором сопротивление в сотни раз выше.

Зуммер со встроенным генератором при прямом подключении к номинальному напряжению (все новые устройства имеют наклейки с пояснениями) будут непрерывно издавать звук. Зуммеры без генератора, как и электромагнитные динамики, нужно подключать в отдельную аудио-цепь.

После ознакомления с устройством, подключите зуммер, как показано ниже:



Загрузите код скетча в микроконтроллер Arduino и проанализируйте результат.

```
int buzzer=11;// номер управляющего пина зуммера
void setup()
{
 pinMode(buzzer,OUTPUT);//пин работает как выходной
}
void loop()
{
 unsigned char i,j;//задаем переменные
 while(1)
{
        for(i=1;i<80;i++)//выходная частота звука
{
          digitalWrite(buzzer,HIGH);//зуммер издает звук
          delay(1);//задержка 1 ms
          digitalWrite(buzzer,LOW);// зуммер не издает звук
          delay(1);// задержка 1 ms
  }
  for(i=0;i<100;i++)//другая выходная частота звука
  {
         digitalWrite(buzzer,HIGH);// зуммер издает звук
```

```
            delay(2);// задержка 2 ms
            digitalWrite(buzzer,LOW);// зуммер не издает звук
             delay(2);// задержка 2 ms
             }
   }
}
```

При первой частоте зуммер издает звук в течение 1 мс, а затем замолкает на 1 мс. Одна секунда эквивалентна 1000 миллисекунд, весь цикл занимает 2 миллисекунды. В итоге частота звука равна 500 Гц.

При второй частоте зуммер пищит 2 мс и молчит 2 мс, весь цикл занимает 4 мс. Частота звука равна 250 Гц.

Цикл события с частотой звука 500 Гц длится 80 мс, затем еще 100 мс происходит событие с частотой 250 Гц.

В данном опыте цикл записывается с помощью оператора while().

В подпрограмме loop оператор while прописывается следующим образом:

while(выражение)

подпрограмма оператора

Оператор определяет условия завершения цикла, смысл такой: когда переменная вычисляется до значения выражения в скобках (не до нуля), цикл останавливается.

Назначение: выполняет цикл по условию «до тех пор, пока». Подпрограмма цикла выполняется до тех пор, пока текущее значение не станет равным выражению в скобках.

11. Опыт с цифровым индикатором

Описание цифрового индикатора

Цифровой индикатор – это полупроводниковое показывающее устройство на светодиодах. Цифровые индикаторы делятся на 7-сегметные и 8-сегментные и отличаются количеством светодиодов (в 7-сегментном на один светодиод и на один символ меньше, чем в 8-сегментном).

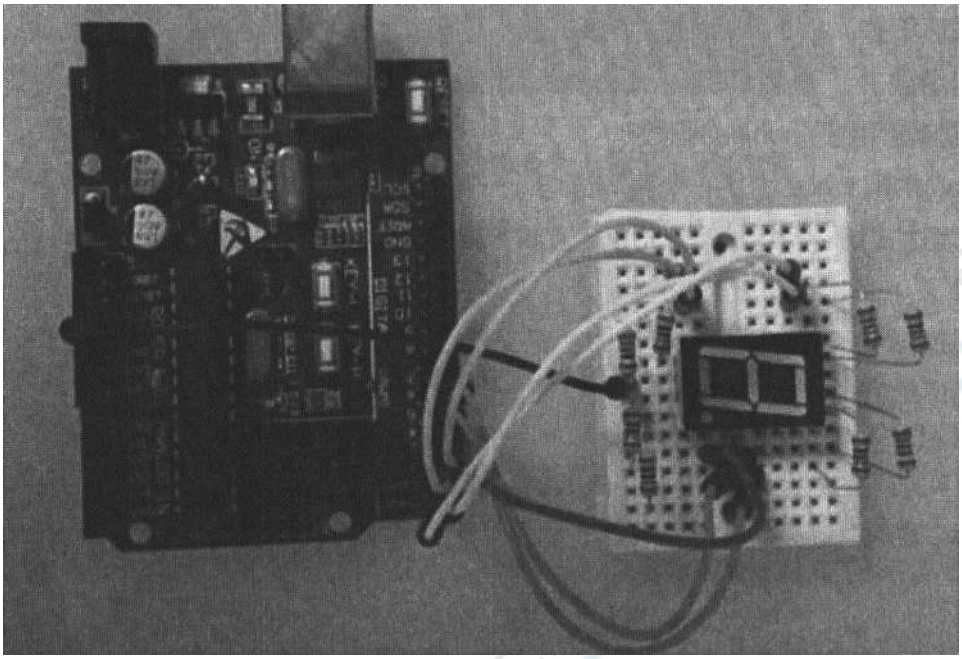
Цифровые индикаторы имеют схемы включения с общим катодом и общим анодом. У индикаторов с общим анодом общие выводы (аноды) подключаются к COM-порту. При использовании индикатора с общим катодом общий контакт «PWR» подключается к разъему «PWR» источника питания. Когда на общий катод какого-либо сегмента цифрового индикатора подается мощность низкого уровня, сегмент начинает гореть, а когда подается мощность высокого уровня, соответствующий сегмент не горит. У индикаторов с общим катодом все устроено наоборот, общие катоды образуют один катод, общие аноды подключаются отдельно.

Для начала познакомимся с цифровым индикатором для данного опыта.

Ниже указана схема подключения цифрового индикатора модели SM41056 с высотой символа 0,5″.

Как указано на обратной стороне индикатора, по четырем сторонам устройства имеются 2 питающих входа и два промаркированных пина 5, 10. Отдельно обозначены контактные штырьки 1, 6, 5, 10. Цифровой индикатор, как и светодиоды, нуждается в дополнительном сопротивлении при подключении. Поскольку в интернете нет подробного

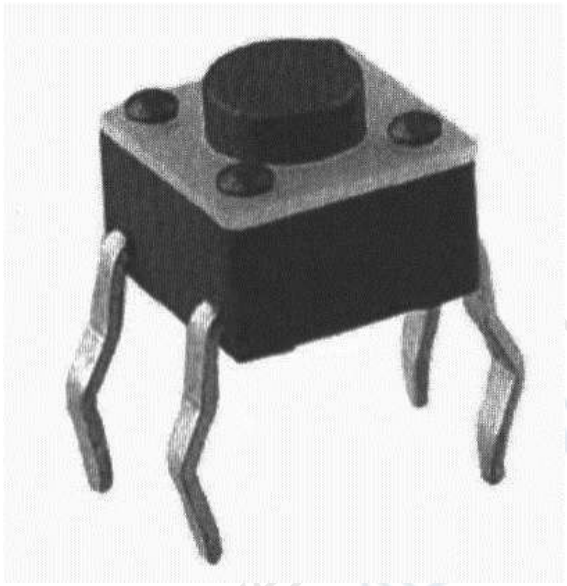
паспорта с параметрами цифрового индикатора, нам неизвестно его напряжения пробоя. Для того, чтобы напряжение источника питания не повредило индикатор, мы выбираем резистор номиналом 1 кОм и источник питания 3,3 В.



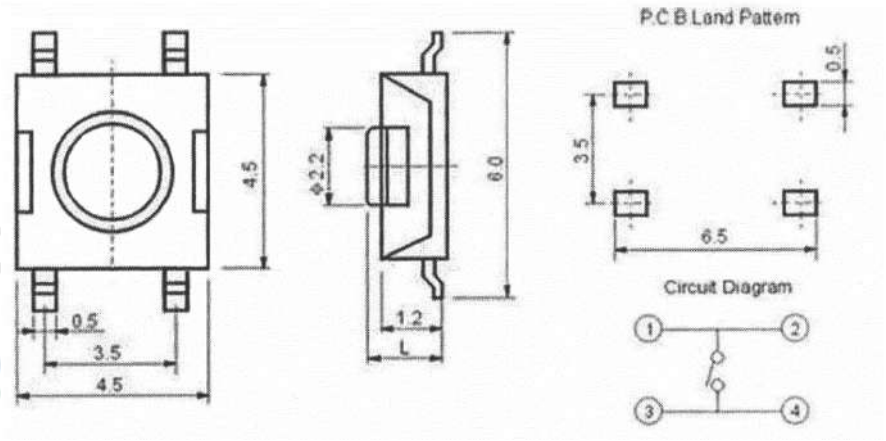
```
byte index=0;
const byte LEDs[10]={ //для светодиода с общим анодом
        B0000001,
        B1001111,
        B0010010,
        B0000110,
        B1001100,
        B0100100,
        B0100000,
        B0001111,
        B0000000,
        B0000100
};
void setup() {
        DDRD=B11111111;

}

void loop() {
        PORTD=LEDs[index];
        index++;
        if(index==10){
                index=0;
            }
        delay(1000);
        }
```

12. Опыт с кнопкой

Кнопка – один из самых распространенных элементов, используется для замыкания и размыкания электрических цепей, для управления электродвигателями или для включения/выключения устройств. Внешний вид кнопок может быть совершенно разным. В данном опыте используется микрокнопка 6 мм, показанная ниже.



Кнопка имеет 4 контактных штырька, вид сверху показан ниже:



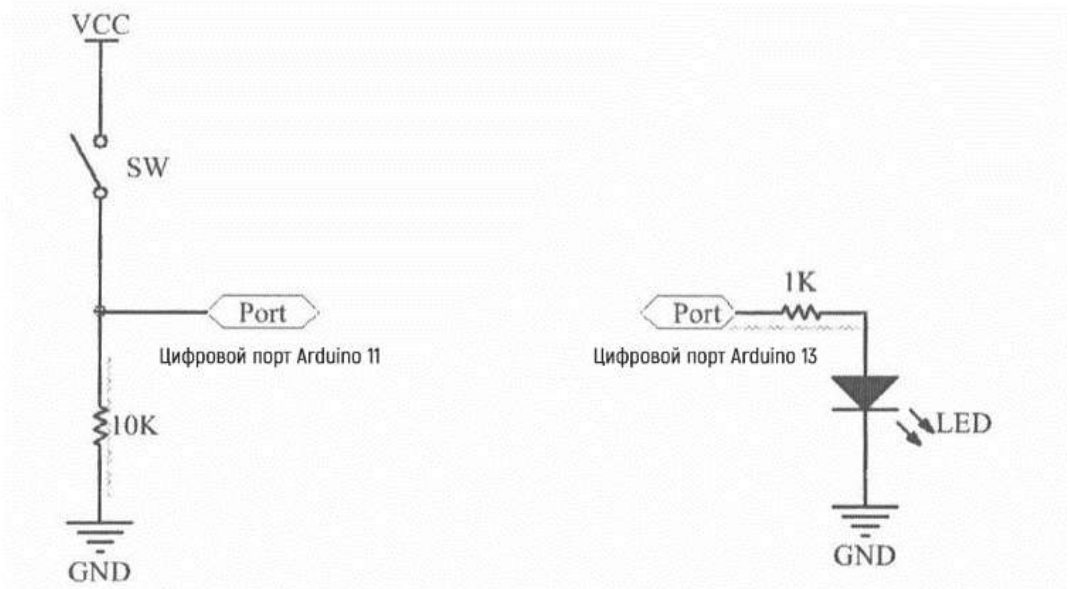
Когда кнопка не вжата, ножки 1-2 и 3-4 соединены попарно. Когда кнопки вжата, все ножки 1,2,3,4 соединяются между собой.

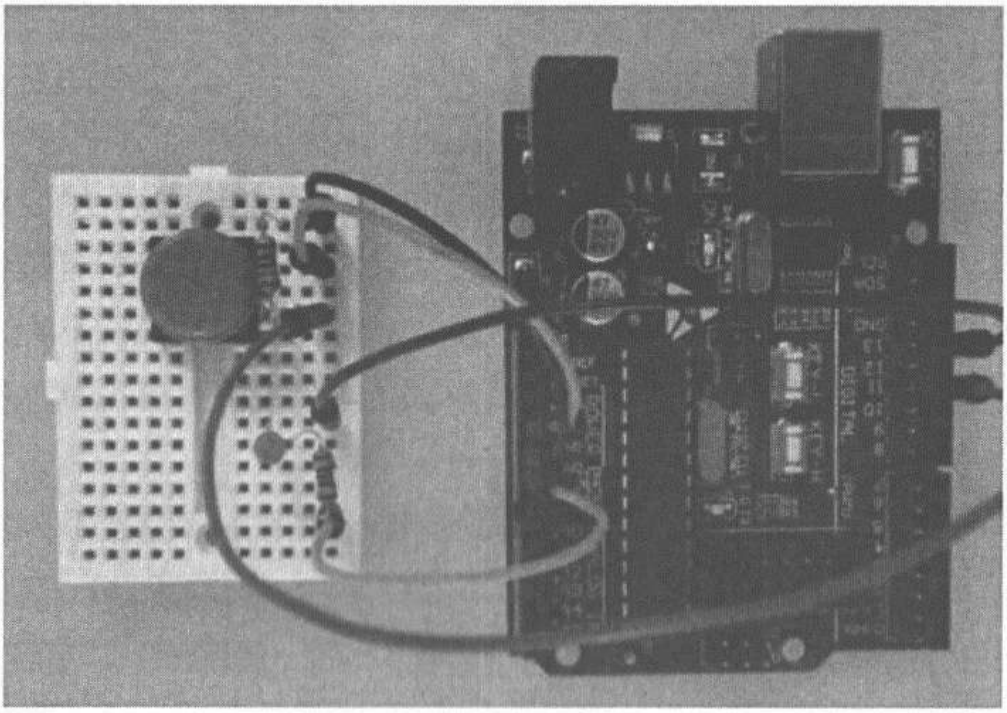
В данном опыте кнопка используется для управления светодиодом.

Обычно кнопка подключается напрямую в цепь последовательно со светодиодом, однако наш случай нестандартный, мы будем использовать косвенный метод подключения, а срабатывание кнопки будет зависеть от уровня напряжения в цепи. Если напряжение больше 4,88 В, то на светодиод будет подаваться напряжение высокого уровня. В противном случае будет подаваться напряжение низкого уровня. Мы будем управлять включением и отключением светодиода с помощью логики, это более распространенный метод при практическом использовании.

Электрическая схема показана ниже. У кнопки есть две группы контактов – первая группа подключается к 5 В, вторая группа подключается в порт 11 и к «земле» (GND) через

резистор. Длинный контактный штырь светодиода подключается последовательно к резистору 1кОм и в порт 13, короткий штырь подключается к «земле» (GND).





Загрузите код скетча в микроконтроллер Arduino и проанализируйте результат.

```
const byte LED=13;// управляемый контакт светодиода
const byte SW=11;// управляемый контакт переключателя
void setup()
{
  pinMode(LED, OUTPUT);//контакт светодиода работает как выходной
  pinMode(SW, INPUT);
  // digitalWrite(SW, HIGH);
}
```

```
void loop()
{
   boolean val=digitalRead(SW);
         if(val){
            digitalWrite(LED, HIGH);
            }
         else{
            digitalWrite(LED, LOW);
           }
         }
```

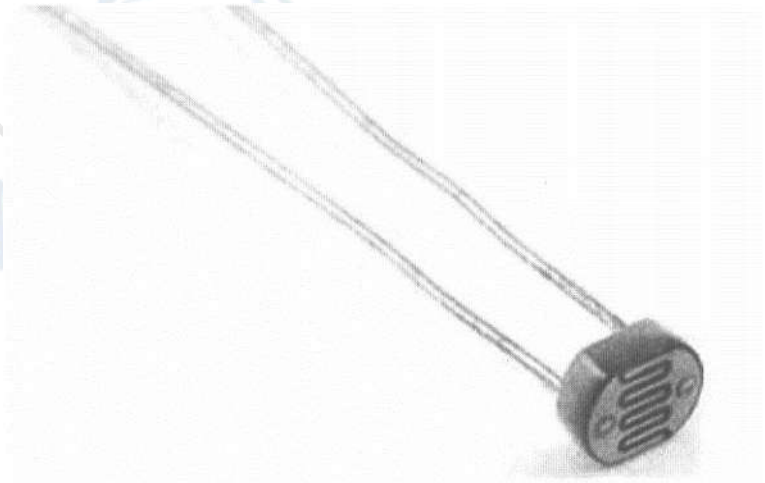
13. Опыт с оптическим датчиком

Фоточувствительные сопротивления так же называются фоторезисторами. Распространенные материалы для фоторезисторов – сульфид кадмия, сульфид алюминия, сернистый свинец, сернистый висмут и т.д. Эти материалы имеют определенную длину волны света, с падающей характеристикой сопротивления. Поскольку под действием света генерируются носители заряда, ток в проводнике возрастает, во внешнем электрическом поле фоторезистора возникает дрейф, а его сопротивление стремительно падает.

Принцип работы фоторезистора основывается на внутреннем фотоэффекте. В двух нагруженных электродах из полупроводниковых светочувствительных материалов, монтированных в прозрачном кожухе, возникает фоторезистивный эффект. Чтобы увеличить светочувствительность, металлические проводники фоторезистора выполняются в виде гребенок.

Когда свет попадает на фоторезистор, то электрическое сопротивление падает. Когда падающий свет ослабевает, электрическое сопротивление растет.

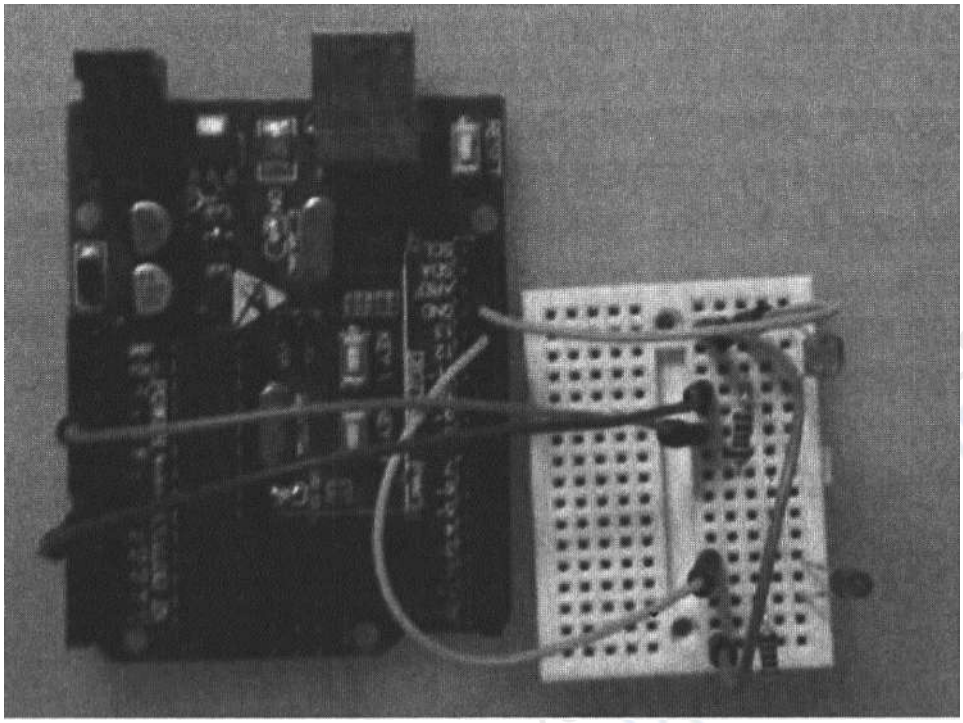
На фото ниже показан фоторезистор



Суть данного опыта – заставить светодиод гаснуть при наличии света и загораться, когда вокруг становится темно. Поскольку фотосопротивление зависит от количества и природы света, при одинаковой интенсивности солнечного света и лампы дневного света реакция фоторезистора может быть разной и связана преимущественно с длиной световой волны. В данном опыте мы задействуем параметры энергосберегающих ламп 60 W трех основных цветов (в помещении нет естественного света). При разных окружающих условиях параметры будут отличаться, корректируйте их по мере надобности.

Перед опытом сначала измерьте текущее значение сопротивление фоторезистора при свете и в темноте.

Сопротивление светодиода при свете – 9,1 кОм.
Сопротивление светодиода в темноте – 32,4 кОм.
Схема соединения показана ниже



Поскольку сопротивление при свете 9,1 кОм, а сопротивление в темноте 32,4 Ом, мы выбираем нагрузочное сопротивление 10 кОм. При наличии преград для света сопротивление может значительно изменятся. Предполагается, что сопротивление при свете – 10 кОм (для фоторезистора разница с измеренным сопротивлением 9,1 кОм, поэтому удобно проводить расчеты), нагрузочное сопротивление 10 кОм. Отпирающее напряжение аналогового порта 2 тоже составляет 10 кОм, при питающем напряжении источника 5 В, и соответствующих значений сопротивления при свете и темноте, отпирающее напряжение равно 5x10/(10+10)=2,5 В.

```
const byte LED=13;// переменная LED  определяет уровень мощности порта
const byte cds=A0;//  в переменную cds считывается напряжение порта
void setup() {
  pinMode(LED, OUTPUT);//ledPin в режиме выхода
}
void loop() {
int val;
    val=analogRead(cds);//считывает показания датчика
        if(val>=512){   //512=2.5 В, значение тем выше, чем выше чувствительность
                        датчика, и тем ниже, чем ниже чувствительность датчика
    digitalWrite(LED, HIGH); // когда val меньше 512(2,5 В), светодиод горит
    }
    else{
        digitalWrite(LED, LOW);
  }
}
```

14. Опыт с ЖК-дисплеем 1602

В данном опыте микроконтроллер Arduino управляет отображением текста на ЖК-дисплее 1602.

ЖК-дисплеи 1602 очень распространены, самые ранние дисплеи использовались в контроллерах HD44780, сейчас все модули 1602 совместимы с IC, поэтому характеристики модулей по большей части аналогичны.

Основные параметры 1602 LCD
Емкость дисплея 16x2 символов;
Рабочее напряжение микросхемы 4,5-5,5 В;
Рабочий ток 2,0 мА (5,0 В);
Наиболее подходящее напряжение модуля 5,0 В;
Размеры символа 2,95x4,35 мм (Ш x В).

Характеристики портов и контактов ЖК-дисплея 1602

Символ	Код	Пояснения к контакту	Номер	Код	Примечания
1	VSS	Заземление	9	D2	Date I/O
2	VDD	«Плюс» источника	10	D3	Date I/O
3	VL	Регулировка контрастности	11	D4	Date I/O
4	RS	Выбор регистра (V/L)	12	D5	Date I/O
5	R/W	Линия чтения/записи (H/L)	13	D6	Date I/O
6	E	Клемма включения	14	D7	Date I/O
7	D0	Date I/O	15	BLA	«Плюс» подсветки
8	D1	Date I/O	16	BLK	«Минус» подсветки

Пояснения к портам:

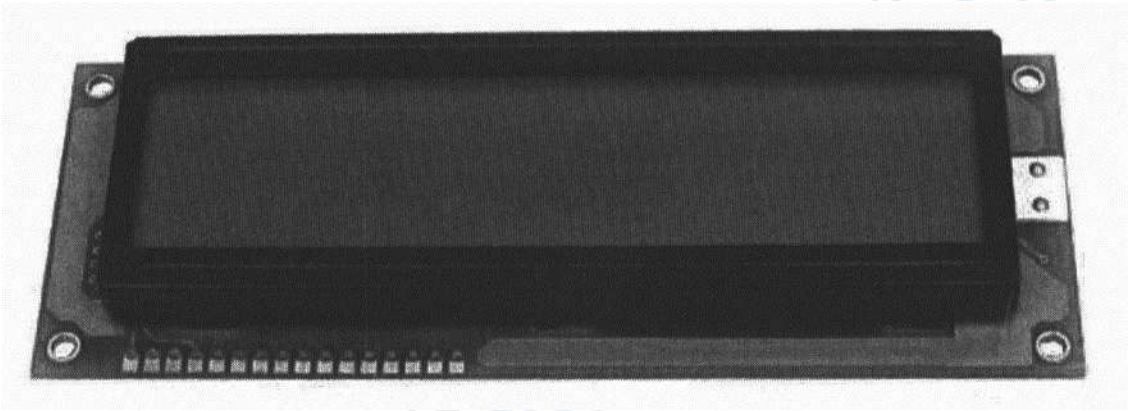
1. Двухгрупповой блок питания. Первый блок – модульный источник напряжения; второй блок – для питания подсветки, в среднем потребляет 5 В. В данном опыте подсветка может работать на напряжении 3,3 В.
2. VL – клеммы для регулировки контрастности дисплея, корректировку производят при последовательно подключенном потенциометре с сопротивлением не больше 5 кОм. В данном опыте для настройки контрастности применяется сопротивление 1 кОм. Клеммы подключаются к высокому и низкому потенциалам. В данном опыте используется метод низкого потенциала, клемма подсоединяется к земле (GND) через резистор 1 кОм.
3. У большинства ЖК-дисплеем имеется RS-контакт, это клемма для выбора регистра данных и команд. Для данных выбирается высокий уровень регистра; для команд выбирается низкий уровень регистра.
4. Во многих ЖК-дисплеях используется контакт RW, клемма для чтения/записи. Для чтения выбирается высокий уровень мощности, для записи выбирается низкий уровень мощности.
5. Линия E присутствует во многих дисплеях. Часто после стабилизации сигнала общей линии положительный сигнал используется для считывания. Когда на данный контакт поступает высокая мощность, сигнал общей линии не меняется (данные остаются на одном фронте).
6. 8-клеммная D0-D7 двухнаправленная линия, используется для передачи команд и данных.

7. BLA – положительный контакт для подсветки, BLK – отрицательный контакт для подсветки.

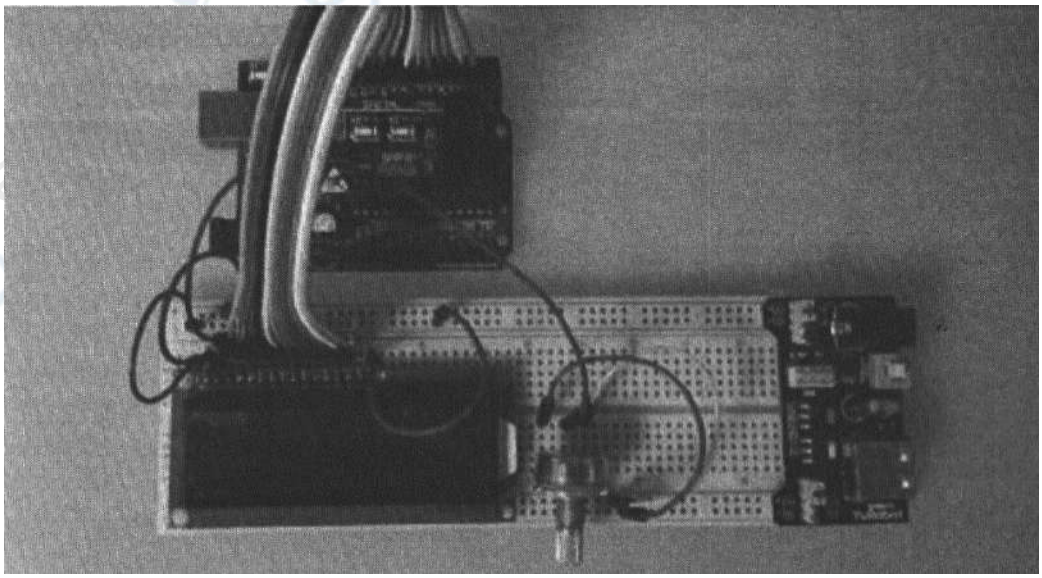
Основные схемы управления ЖК-дисплея 1602:

Режим чтения	Вход	RS=L, R/W=H, E=H	Выход	D0-D7= Слово состояния
Запись команд	Вход	RS=L, R/W=L, D0-D7=код команды, E=высокий уровень импульса	Выход	нет
Чтение команд	Вход	RS=H, R/W=H, E=H	Выход	D0-D7= данные
Запись данных	Вход	RS=H, R/W=L, D0-D7=данные, E=высокий уровень импульса	Выход	нет

Внешний вид ЖК-дисплея 1602



Дисплей 1602 напрямую подключается к Arduino по правилам, указанным в инструкции. Есть способы подключения на 4 клеммы и 8 клемм. Сперва мы проведем опыт с подключением к 8-ми клеммам. Схема подключения показана ниже:



```
#include <LiquidCrystal.h> //расширенная память LCD
LiquidCrystal lcd(12,11,10,9,8,7,6,5,4,3,2);
// определяем контакты LCD, rs=12, rw=11, en=10, D0-D7=2-9
void setup()        {
lcd.begin(16,2);
```

```
lcd.print(''hello, world!'');

}

void loop() {
    lcd.setCursor(0,1);
    lcd.print(mills()/1000);
}
```

15. Опыт с датчиком температуры LM35

Проводя следующий опыт, мы будем использовать знания из работы с дисплеем 1602. Мы возьмем простой датчик температуры и будем выводить измеренную датчиком LM35 температуру в комнате на экран дисплея 1602.

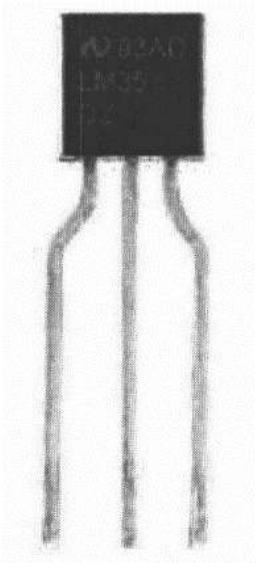
В прошлый раз мы работали с дисплеем с помощью прописанного вручную кода, в этот раз мы будем использовать встроенный в Arduino драйвер LiquidCrystal. Эта библиотека позволяет микроконтроллеру Arduino управлять дисплеями на основе Hitachi HD44780 или с микросхемами с аналогичной емкостью и позволяет работать в режим 4bit или 8bit.

Настройки библиотеки Liquid Crystal можно корректировать в среде Arduno.

Что же такое датчик температуры?

Датчики температуры используются для измерения характеристик температуры среды, в них температура преобразуется в напряжение датчика. Метода измерения бывают двух типов: контактное и бесконтактное измерение. Исходя из материала датчика и характеристик элементов есть датчики термопары и датчики сопротивления. В паяльниках, например, используется датчик термопары, в данном опыте мы используем датчик LM35.

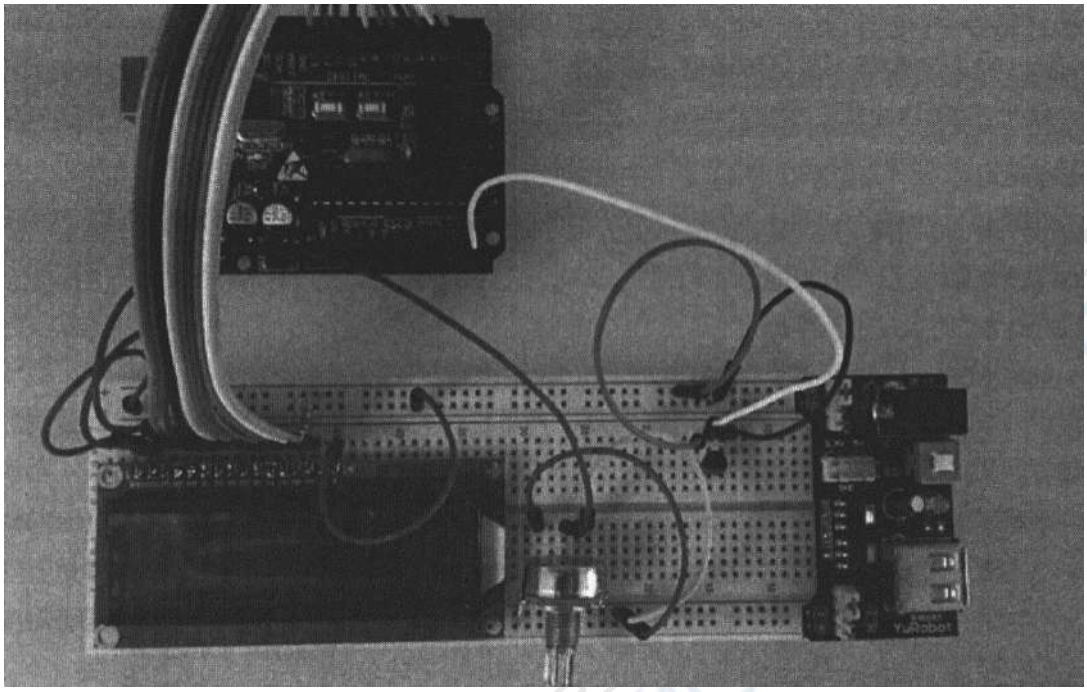
Датчик температуры LM35



Датчики LM35 очень распространены, применяются во внутреннем механизме компенсации, выходной сигнал датчика возникает при температуре 0 ° С и выше. Для оболочки используется T0992, рабочее напряжение 4-30 В. Для указанного диапазона напряжений рабочий ток микросхемы не превышает 60 мА. Исходя из инструкции к устройству, выходное напряжение датчика LM35 зависит от характеристики температуры

по Цельсию. При 0 ° С напряжение равно 0 В, при возрастании температуры на 1 градус, напряжение увеличивается на 10 мВ.

Схема для данного опыта показана ниже:



Код скетча

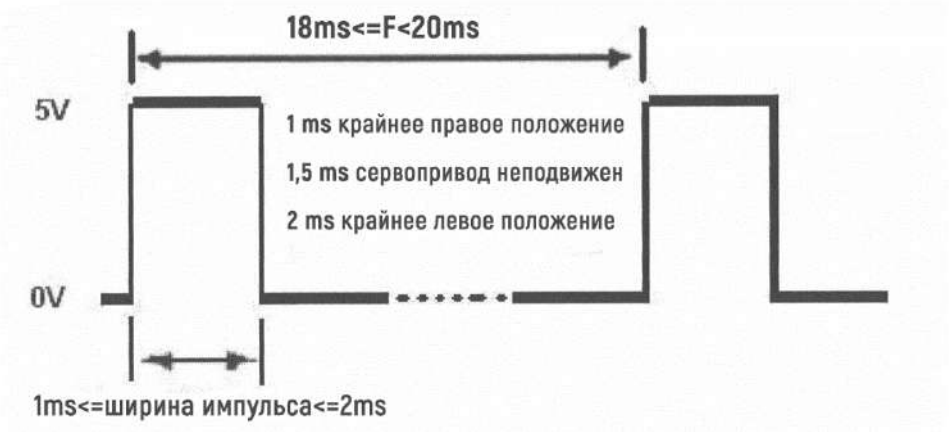
```
#include <LiquidCrystal.h>  //загрузка встроенной библиотеки LiquidCrystal
LiquidCrystal lcd(12,11,10,9,8,7,6,5,4,3,2); //настройки портов
int potPin=A5; //Аналоговый порт 4 задается как вход датчика LM35
float temperature = 0; //переменная temperature с плавающей точкой
long val=0; //переменная val - длинное целое число
void setup()
{
        lcd.begin(16,2); //инициализация LCD
        lcd.print(''LM35 Thermometer''); //вывод на дисплей "LM35 Thermometer"
        delay(1000);// задержка 1000 ms
}
void loop()
{
val=analogRead(potPin); //в переменную val считывается значение с датчика LM35
temperature=((val+1)*0.0048828125*1000);//увеличивает в 10 раз считанное значение температуры
lcd.clear();//очистка
lcd.print("LM35 Thermometer");// вывод на дисплей "LM35 Thermometer"
lcd.setCursor(0,1);//настройка положения курсора в первую строку, второй ряд
lcd.print((long)temperature/10);//температура отображается как длинное целое число
lcd.print(".");
lcd.print((long)temperature % 10);//одно число после запятой
lcd.print((char)223);//отображается буква о
lcd.print("C");//отображается буква «С»
delay(2000);//задержка 2000 ms, скорость обновления дисплея
}
```

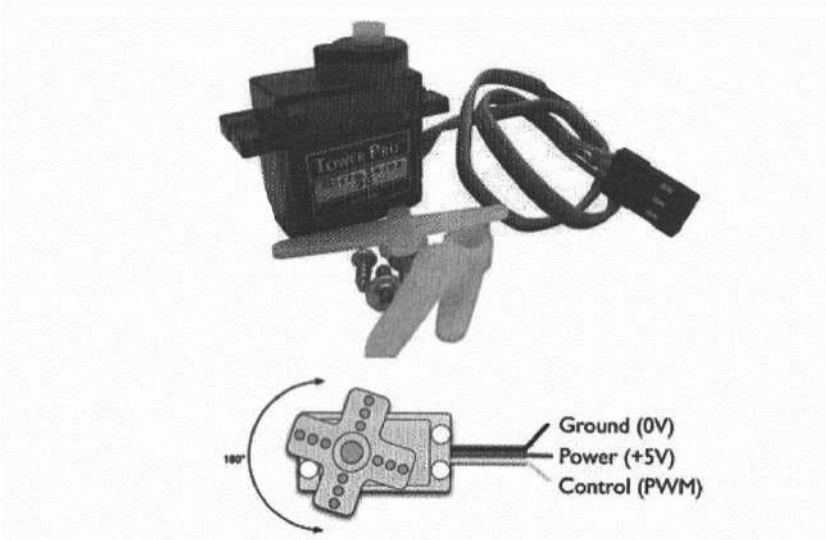
16. Опыт с управлением сервоприводом

Сервопривод (английской название Servo): двигатели постоянного тока (ДПТ), редуктор с валом, датчики и цепь управления образуют автоматическую систему управления. С помощью сигнала управления задается угол вращения объекта. У сервоприводов максимальный угол вращения (например, 180 градусов) зависит от ДПТ. Если ротор ДПТ делает несколько оборотов, то вал сервопривода проворачивается на несколько градусов, он не может сделать полный оборот (цифровые сервоприводы с режимами сервопривода и двигателя такой проблемы не имеют). В стандартныых ДПТ не предусмотрено обратной связи по углу вращения ротора, а в сервоприводах она есть. Назначение сервопривода так же отличается: стандартный ДПТ используется для создания механического вращающего момента, а сервоприводы используются для управления углом поворота объекта (например, в робототехнике).

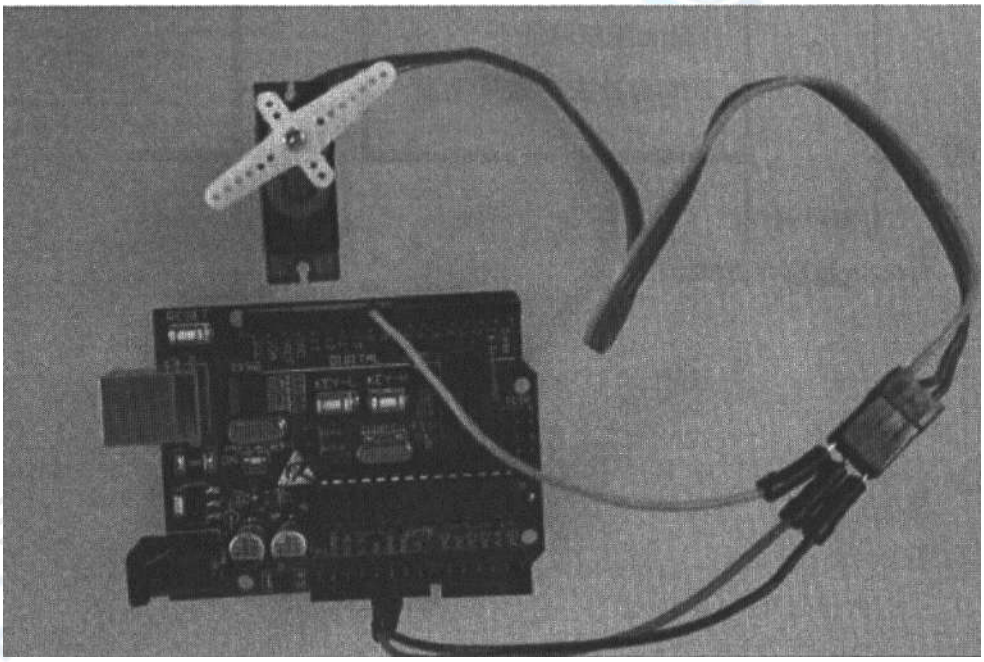
Принцип работы:

Управляющий сигнал, с помощью устройств приема и передачи, поступает на микросхему с модулятором сигнала, и создается напряжение смещения постоянного тока. Внутри сервопривода есть стандартная электрическая цепь с периодом 20 мс и шириной стандартного сигнала 1,5 мс, на которую поступает напряжение смещения постоянного тока и сравнивается с напряжением потенциометра, в результате чего на выходе микросхемы появляется разность напряжений. В конце разность напряжений с положительного и отрицательного выводов поступает на управляющую двигателем микросхему и в зависимости от закона управления задаются прямое и обратное вращения двигателя.





Как показано на рисунке выше, сервопривод имеет три линии подключения. Черная (или коричневая) линия используется для заземления, красная линия подключается к источнику 5 В, желтая линия (белая или оранжевая) – это управляющий кабель.



Управляющий сигнал (на изображении «H») это ШИМ-волна, все производственные микроконтроллеры используют этот тип сигнала. В этом документе мы используем микроконтроллер со средой разработки Arduino.

Длина высокого уровня импульса составляет от 1 до 2 миллисекунд или от 1000 до 2000 микросекунд. При 1000 микросекунд происходит поворот вправо. Регулируя ширину импульса, вы можете увеличивать или уменьшать диапазон движения сервопривода.

Длина низкого уровня импульса составляет 20 миллисекунд. Каждые 20 миллисекунд (в течение 50 секунд) сигнал снова переключается на высокий уровень, в ином случае сервопривод может застрять, и будет сложно гарантировать его стабильную работу. Однако, если вы хотите заставить сервопривод «пританцовывать», можно использовать такой способ.

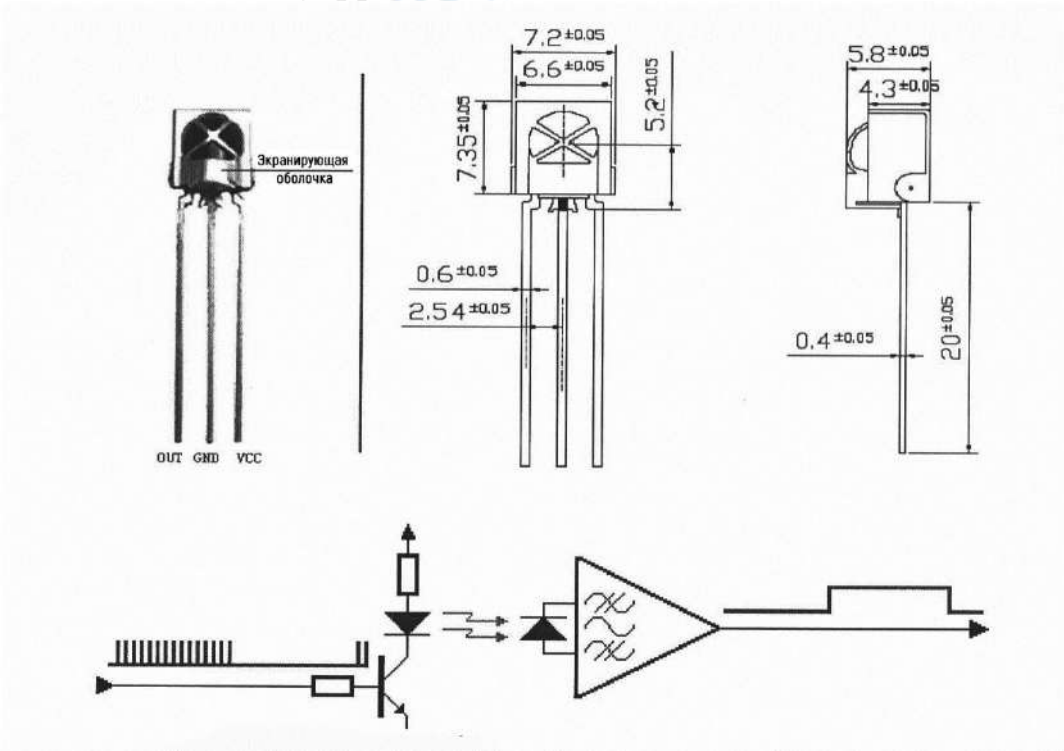
```
int servoPin=9;
```

```
int servoPosition=1500;//позиция в микросекундах
void setup() {
        pinMode(servoPin, OUTPUT);

}
void loop() {
        digitalWrite(servoPin, HIGH);
        delayMicroseconds(servoPosition);
        digitalWrite(servoPin, LOW);
        delay(20);
}
```

17. Инфракрасный приемник сигналов

Принцип работы инфракрасного приемника:
Состоит из систем излучения и приема ИК-сигнала.

1. ИК-приемник: цепь приемника образуется из приемника ИК-сигнала и усилителя, интегрированных в общий модуль. Способен выполнять все процессы от приема ИК-сигнала до передачи сигнала на уровень TTL. Подходит для дистанционного приема данных. Модуль ИК-приемника имеет три контакта: линию ИК-сигнала, VCC- и GND-линии. Приемник удобно подключать к Arduino и другим микроконтроллерам.
2. ИК-излучатель: дистанционный излучатель с сигналом несущей частоты 38 кГц, производит кодировку сигнала посредством кодирующей микросхемы. Принцип кодирования сигнала представлен в инструкции по эксплуатации ИК-приемника.




```
#include <IRemote.h>
int RECV_PIN=11;//управляемый контакт ИК-приемника
IRecv irrecv(RECV_PIN);
```

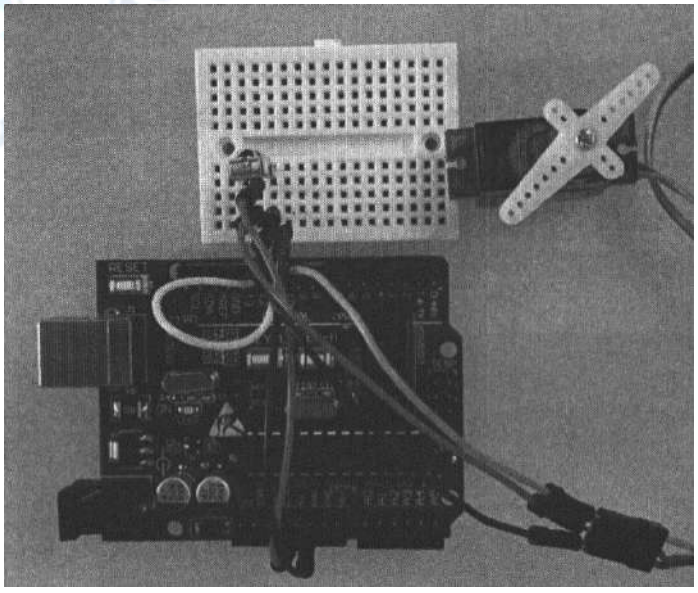
```
decode_results results;
void setup()
{
        Serial.begin(9600);
        irrecv.enableIRIn();//Инициализация переменной ИК-приемника
}
void loop() {
        if (irrecv.decode(&results)){
          Serial.println(results.value, HEX);// Вывод в порт кода в 16-ричном виде
          Serial.println();//добавляем пустую строку для удобства отображения
          irrecv.resume();//принимаем следующее значение
        }
}
```

В окне Serial monitor можно увидеть ключевые значения

18. Управление сервоприводом с помощью ИК-приемника

```
#include <IRemote.h>
#include <Servo.h>
Servo servo;
const byte RECV_PIN=11;
const byte LED_PIN=13;
const byte SERVO_PIN=8;

boolean sw=false;
byte servoPos=90;
IRrecv irrecv(RECV_PIN);
decode_results results;

void setup(){
irrecv.enableIRIn();
pinMode(LED_PIN,OUTPUT);
```

```
servo.attach(SERVO_PIN);
servo.write(servoPos);
}

void loop() {
        if(irrecv.decode(&results)){
        switch(results.value){
        case 0xFFA25D:
        sw=!sw;
        digitalWrite(LED_PIN,sw);
        break;
        case 0xFFA857;
        if (servoPos>10){
        servoPos-=10;
        servo.write(servoPos);
        }
        break;
        case 0xFF906F;
        if(servoPos<170){
        servoPos+=10;
        servo.write(servoPos);
        }
        break;

          }
          irrecv.resume();
          }
        }
```

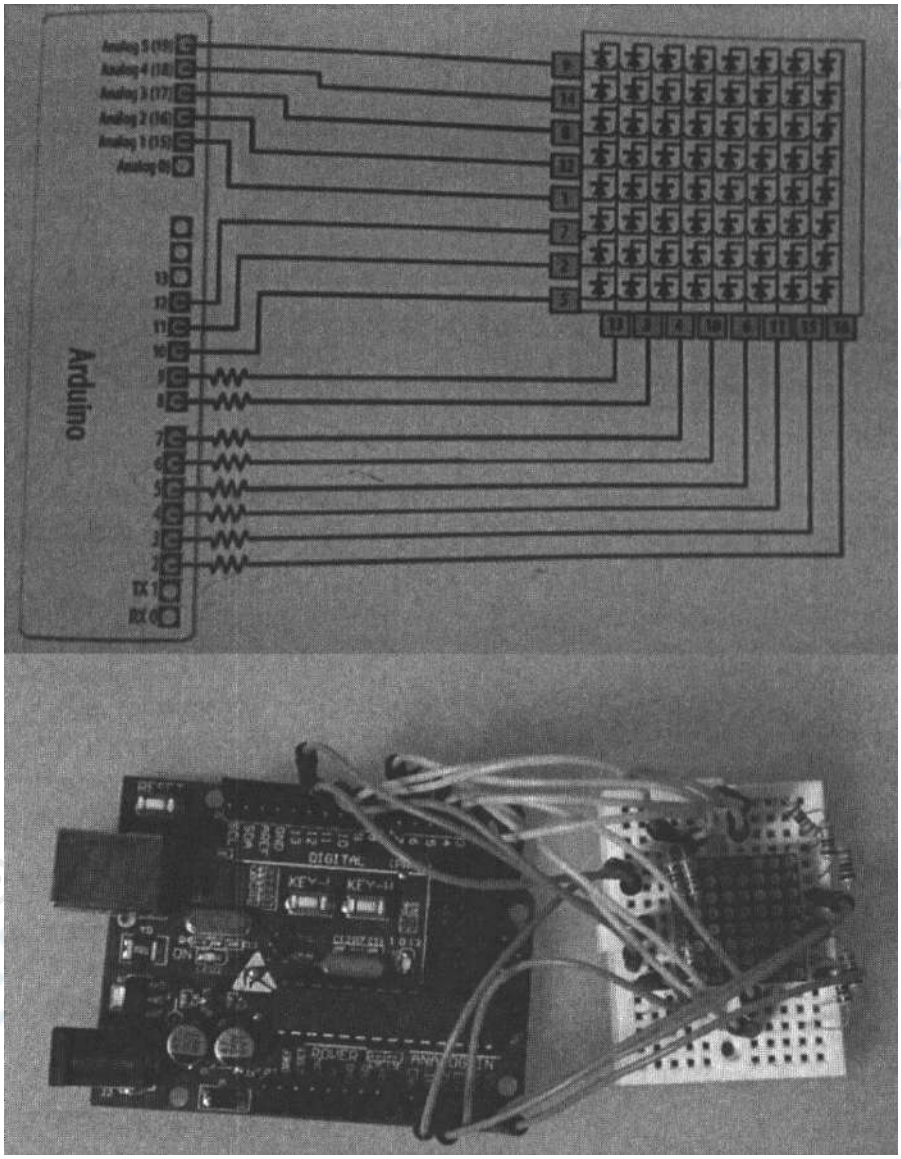


19. Опыт со светодиодной сеткой

Светодиодные экраны-сетки – одно из современных и широко распространенных электронных устройств с гибкой поверхностью отображения (может монтироваться посегментно в произвольном порядке), отличается высокой яркостью, долговечностью, цифровой формой, практичностью и т.д.

На передней части экрана располагаются маленькие светодиоды и цифровые индикаторы. По аналогии с цифровыми индикаторами, собранными из 8-ми групп светодиодов, светодиодная сетка собрана из 8x8 групп светодиодов и всего содержит 64 группы.

Схема подключения показана ниже.




```
const int columnPins[]={2,3,4,5,6,7,8,9};
const int rowPins[]={10,11,12,A1,A2,A3,A4,A5};
int pixel=0; //светодиоды в сетке от 0 до 63
int columnLevel=0; //значения пикселей преобразовывается в номер ряда сетки
int rowLevel=0; // значения пикселей преобразовывается в номер строки сетки

void setup() {
```

```
            for (int i=0; i<8;i++){
            pinMode(columnPins[i], OUTPUT);
            pinMode(rowPins[i], OUTPUT);
            }
        }

        void loop() {
        pixel=pixel+1;
        if (pixel>63) pixel=0;
        columnLevel=pixel/8; //преобразовываем в номер ряда
        rowLevel=pixel%8;// преобразовываем в номер строки
        for (int column=0; column<8; column++){
          digitalWrite(columnPins[column], LOW);
          for (int row=0;row<8;row++){
                if(columnLevel>column){
                  digitalWrite(rowPins[row], HIGH);
                }
                if(columnLevel==column&&rowLevel>=row){
                 digitalWrite(rowLins[row], HIGH);
                }
                else{
                  digitalWrite(columnPins[column], LOW);
                }
                delayMicroseconds(300);
                digitalWrite(rowPins[row], LOW);
          }
          digitalWrite(columnPins[column], HIGH);
        }
}
```

20. Опыт с микросхемой 74HC595

Микросхема 74HC595 использует ввод в 10-чных цифрах (0-255), преобразовывает их в восьмиразрядные числа в двоичной системе и с помощью разных уровней мощности выводит их на соответствующие контакты.

Номер контакта	Имя контакта	Указания к функциям
1-7, 15	Q0-Q7	Параллельные выходы
8	GND	Заземление
9	Q7’	Последовательный выход
10	MR	Master reset (сброс), подключение к 5В
11	SH_CP	Вход для тактовых импульсов
12	ST_CP	Вход импульсов регистра хранения
13	OE	Вход состояния выходов (активен при LOW)
14	DS	Вход последовательных данных
16	VCC	Питающий порт (5 В)

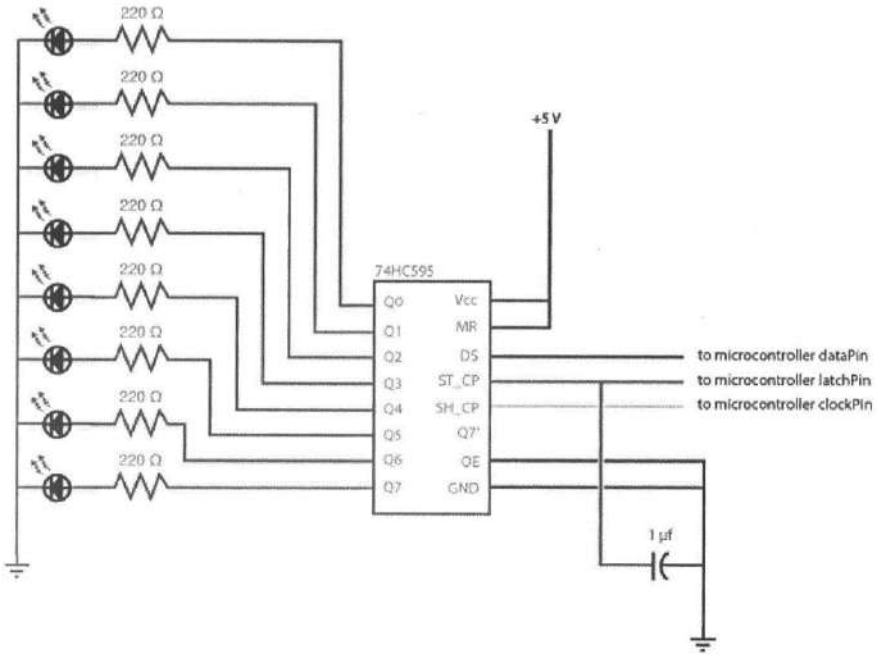
Контакты Q0-Q7 являются выходными. VCC подключается к 5 В. MR – контакт для сброса настроек, активен на низком уровне мощности, поэтому мы подаем на него высокое напряжение, как и на питающий порт 5 В.

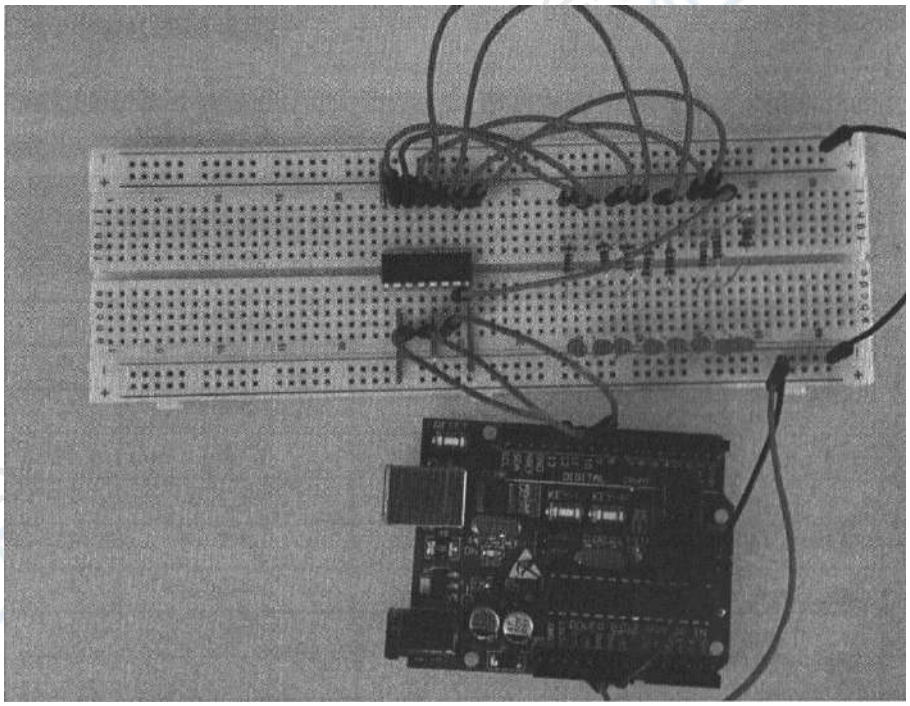
OE передает данные на микроконтроллер, активен только на низком уровне мощности, поэтому мы подключаем его к «земле».

Контакт DS (pin 14) мы подключаем в цифровой порт 11 Arduino, контакт SH_CP (pin 11) подключаем в порт 12, контакт ST_CP (pin 12) подключаем в цифровой порт 8.

```
int latchPin = 8;
int clockPin = 12;
int dataPin = 11; // задаем переменные контактов
void setup()
{
        pinMode(latchPin, OUTPUT);
        pinMode(clockPin, OUTPUT);
        pinMode(dataPin, OUTPUT);//контакты работают как выходы
}
void loop() {
   for(int a=0; a<256; a++)
   /* в цикле к переменной «a» прибавляется 1, подпрограмма выполняется до тех пор, пока «a» не станет равным 256*/
   {
      digitalWrite(latchPin, LOW);//на ST_CP подается низкая мощность для активации входа
      shiftOut(dataPin, clockPin, MSBFIRST, a);
       /*С помощью параметра MSBFIRST на контакты 0-7 передается высокий уровень мощности
      (Низкий уровень мощности LSBFIRST) – параметр dataPin; параметр clockPin - переменная a. Ранее мы говорили, что эта переменная возрстает до 256, а это десятеричное число, которое затем преобразовывается в восьмеричное */
      digitalWrite(latchPin, HIGH);//на ST_CP восстанавливается высокий уровень мощности
      delay(1000);//задержка в 1 сек для визуализации результата
```

```
  }
}
```





В данном опыте мерцание светодиода означает, что переменные 0 или 1-8 светодиодов представляются, как двоичные восьмиразрядные числа, которые затем пересчитываются в десятеричные от 0 до 255 в соответствии с кодировкой.

Проще говоря, числа в двоичной системе пересчитываются в числа десятичной системы на основе таблицы ASCII. Например, двоичное число 101 соответствует 00000101. Таким образом, число 4 и 1, это 4+1=5, десятеричное число равно 5, переменной так же присваивается значение 5. В результате светодиод будет работать по алгоритму «выкл, выкл, выкл, выкл, выкл, вкл, выкл, вкл» (где 0 – «выкл», а 1 – «вкл»). Данный принцип распространяется на все 8 контактов.

Таблица кодировки ASCII

Верхние 4 цифры / Нижние 4 цифры		Непечатные символы ASCII									
		0000					0001				
		0					1				
		10-ные числа	Символ	CTRL	Значение	Код	10-ные числа	Символ	CTRL	Значение	Код
0000	0	0	BLANK NULL	^@	Нуль	NUL	16	▶	^P	Ключ связи данных	DLE
0001	1	1	☺	^A	Начало заголовка	SOH	17	◀	^Q	Управление устройством 1	DC1
0010	2	2	☻	^B	Начало текста	STX	18	↕	^R	Управление устройством 2	DC2
0011	3	3	♥	^C	Конец текста	ETX	19	‼	^S	Управление устройством 3	DC3
0100	4	4	♦	^D	Конец передачи	EOT	20	¶	^T	Управление устройством 4	DC4
0101	5	5	♣	^E	Запрос	ENQ	21	§	^U	Отрицательное подтверждение	NAK
0110	6	6	♠	^F	Подтверждение	ACK	22	▬	^V	Синхронизация	SYN
0111	7	7	•	^G	Сигнал (звонок)	BEL	23	↨	^W	Конец передаваемого блока	ETB
1000	8	8	◘	^H	Забой (шаг назад)	BS	24	↑	^X	Отказ	CAN
1001	9	9	○	^I	Горизонтальная табуляция	TAB	25	↓	^Y	Конец среды	EM
1010	A	10	◙	^J	Перевод строки	LF	26	→	^Z	Замена	SUB
1011	B	11	♂	^K	Вертикальная табуляция	VI	27	←	^ [	Ключ	ESC
1100	C	12	♀	^L	Новая страница	FF	28	∟	^ \	Разделитель файлов	FS
1101	D	13	♪	^M	Возврат каретки	CR	29	↔	^]	Разделитель группы	GS
1110	E	14	♫	^N	Выключить сдвиг	SO	30	▲	^ 6	Разделитель записей	RS
1111	F	15	☼	^O	Включить сдвиг	SI	31	▼	^ -	Разделитель модулей	US

Верхние 4 цифры / Нижние 4 цифры		Печатные символы ASCII												
		0010		0011		0100		0101		0110		0111		
		2		3		4		5		6		7		
		10-ные числа	Символ	10-ные числа	Символ	10-ные числа	Символ	10-ные числа	Символ	10-ные числа	Символ	10-ные числа	Символ	Ctrl
0000	0	32		48	0	64	@	80	P	96	`	112	P	
0001	1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	34	“	50	2	66	B	82	R	98	b	114	r	
0011	3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	36	$	52	4	68	D	84	T	100	d	116	t	
0101	5	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	39	'	55	7	71	G	87	W	103	g	119	w	
1000	8	40	(	56	8	72	H	88	X	104	h	120	x	
1001	9	41	)	57	9	73	I	89	Y	105	i	121	y	
1010	A	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	43	+	59	;	75	K	91	[	107	k	123	{	
1100	C	44	‘	60	<	76	L	92	\	108	l	124	|	
1101	D	45	-	61	=	77	M	93	]	109	m	125	}	
1110	E	46	.	62	>	78	N	94	^	110	n	126	~	
1111	F	47	/	63	?	79	O	95	_	111	o	127	del	‘Back space

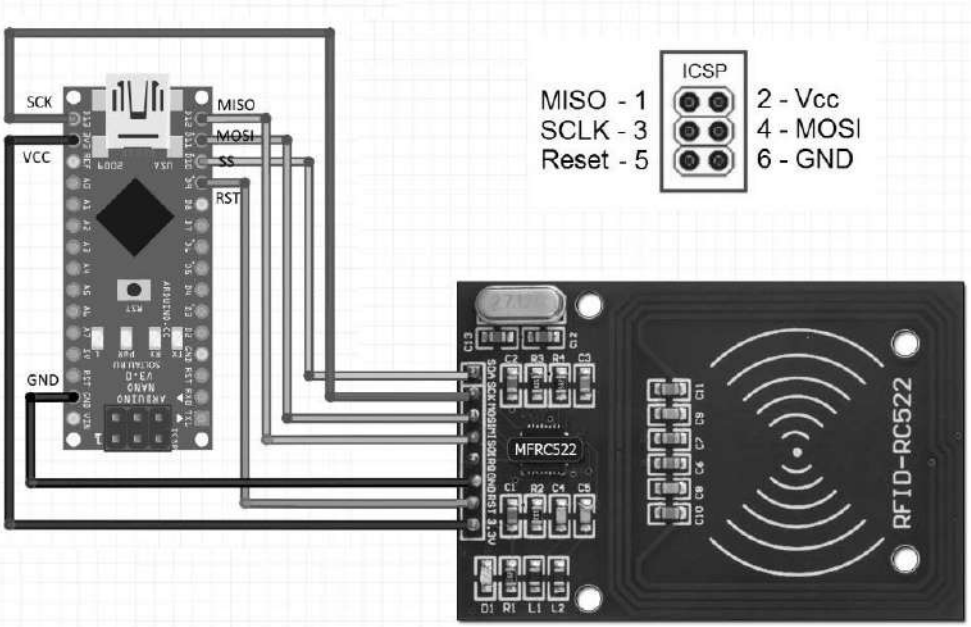
ATMEGA328
ATMEGA328
ICSP
USB JACK
POWER BARREL JACK
LEGEND
GND
CONTROL
PHYSICAL PIN
PORT PIN
ATMEGA328 PIN FUNC
DIGITAL PIN
ANALOG-RELATED PIN
PWM PIN
SERIAL PIN
ARDUINO PIN
Source Total 150mA
Source Total 150mA
General Information
Pay Attention
No Reply
PAY ATTENTION
LED
Absolute max per pin 40mA
recommended 20mA
Absolute max 200mA
for entire package
Reserved
R3 Only
Sink Total 100mA
7-12v depending
on current drawn
GND
RESET
3V3
5V
GND
GND
Vin
SCL
SDA
AREF
GND
MOSI
SCK
MISO
RESET
RX
TX
Connected to the ATMega
and used for USB program
and communicating with it

20. Опыт с RFID модулем

Модуль RFID-RC522 выполнен на микросхеме MFRC522 фирмы NXP. Эта микросхема обеспечивает двухстороннюю беспроводную (до 6 см) коммуникацию на частоте 13,56 МГц.

С помощью данного модуля можно записывать и считывать данные с различных RFID-меток: брелоков от домофонов, пластиковых карточек-пропусков и билетов на метро и наземный транспорт, а также набирающих популярность NFC-меток.

Подключим модуль RFID-RC522 к Arduino по интерфейсу SPI по приведённой схеме.

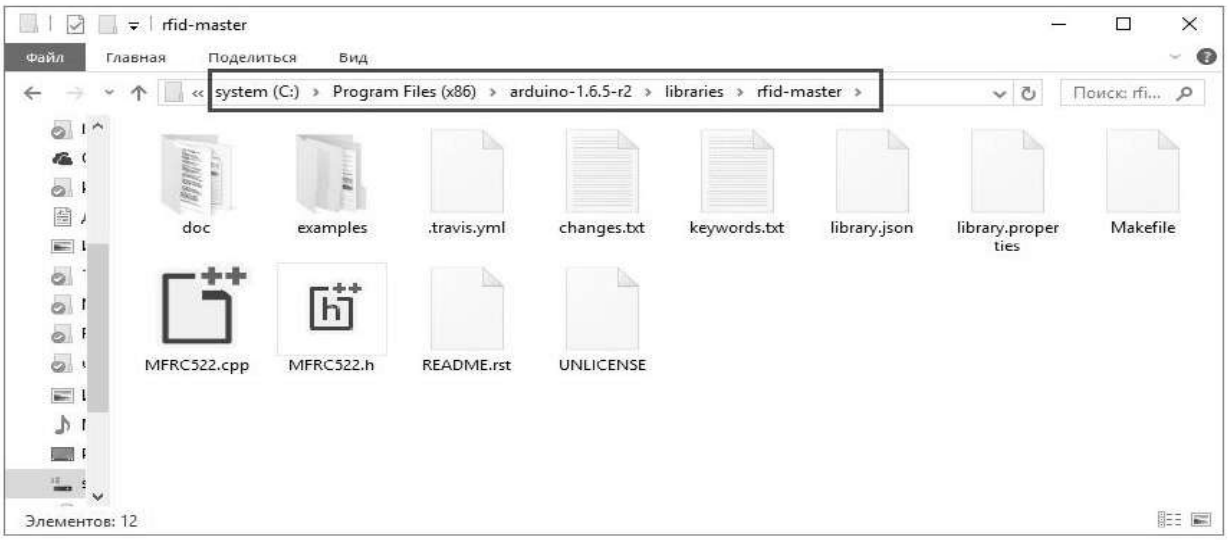


Питание модуля обеспечивается напряжением от 2,5 до 3,3 В. Остальные выводы подключаем к Arduino так:

Пин RC522	Пин Arduino
RST	D9
SDA (SS)	D10
MOSI	D11
MISO	D12
SCK	D13

Напоминаем, что Arduino имеет разъем ICSP для работы по интерфейсу SPI. Его распиновка также приведена на иллюстрации выше. Можно подключить RST, SCK, MOSI, MISO и GND модуля RC522 к разъему ICSP на Arduino.

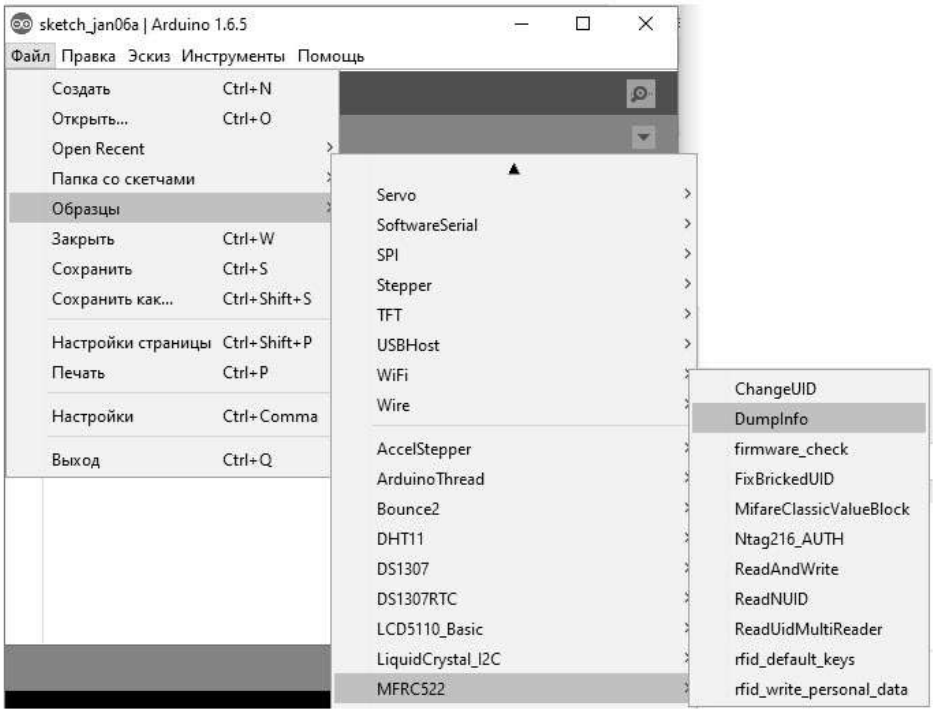
Для знакомства с возможностями микросхемы MFRC522 воспользуемся одной из готовых библиотек, написанных для работы Arduino с RC522. Скачайте ее на странице нашего сайта с данным товаром и распакуйте в директорию.



После этого запустите среду разработки Arduino IDE.

Скетч для считывания информации, записанной на RFID-метке

Откройте скетч из примеров: **Файл → Образцы → MFRC522 → DumpInfo** и загрузите его в память Arduino.

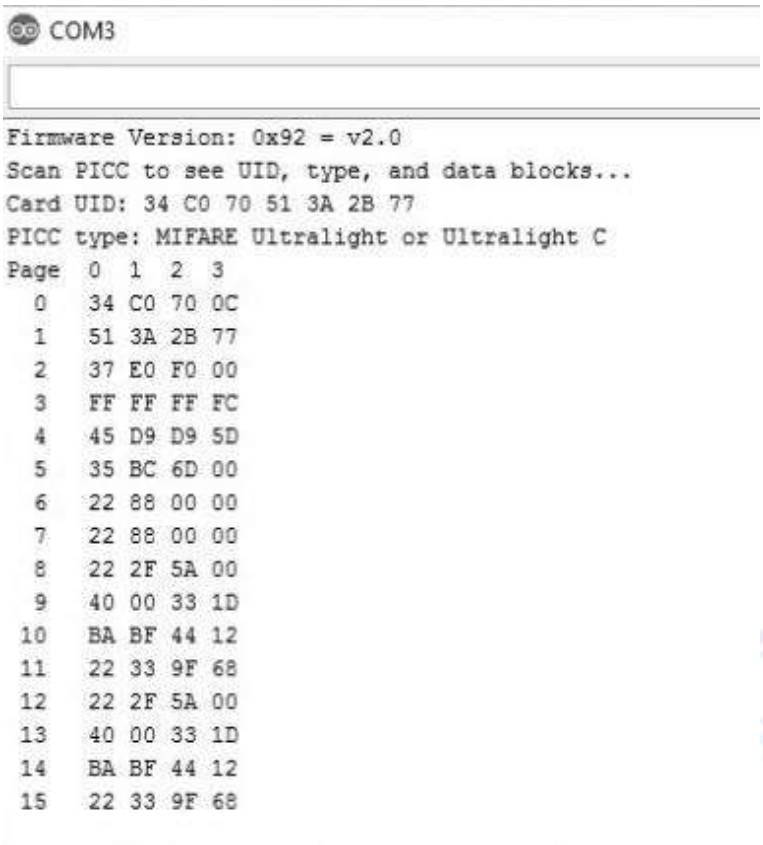


Данный скетч определяет тип приложенного к считывателю устройства и считывает данные, записанные на RFID-метке или карте, а затем выводит их в последовательный порт.

```
#include <SPI.h>
#include <MFRC522.h>

const int RST_PIN = 9; // пин RST
const int SS_PIN = 10; // пин SDA (SS)

MFRC522 mfrc522(SS_PIN, RST_PIN); // создаём объект MFRC522

void setup() {
	Serial.begin(9600); // инициализация послед. порта
	SPI.begin(); // инициализация шины SPI
	mfrc522.PCD_Init(); // инициализация считывателя RC522
}

void loop() {
	// Ожидание прикладывания новой RFID-метки:
	if ( ! mfrc522.PICC_IsNewCardPresent()) {
		return; // выход, если не приложена новая карта
	}

	// Считываем серийный номер:
	if ( ! mfrc522.PICC_ReadCardSerial()) {
		return; // выход, если невозможно считать сер. номер
	}

	// Вывод дампа в послед. порт:
	mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
}
```

Дамп данных с RFID-метки

Запустите монитор последовательного порта сочетанием клавиш Ctrl+Shift+M, через меню **Инструменты** или кнопкой с изображением лупы. Теперь приложите к считывателю билет метро или любую другую RFID-метку. Монитор последовательного порта покажет данные, записанные на RFID-метку или билет.



Например, в нашем случае здесь зашифрованы уникальный номер билета, дата покупки, срок действия, количество оставшихся поездок, а также служебная информация.