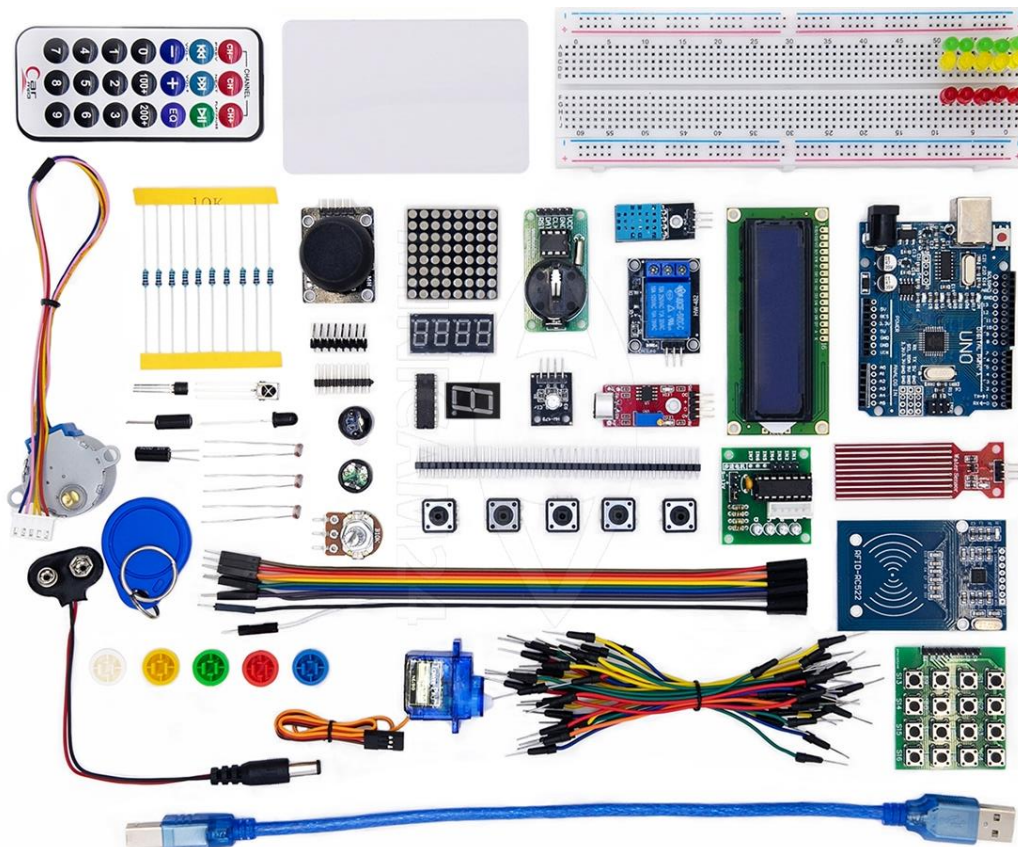




MINICAM24

РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

**Набор для моделирования Ардуино
(Arduino UNO R3) 9V Maximum KIT с
RFID модулем**



СОДЕРЖАНИЕ

1. Введение	3
2. Собираем электрическую цепь с мигающим светодиодом	3
3. Классическая программа «Hello, world!»	7
4. Основные функции Arduino	8
5. Цифровые выходы Arduino	9
6. Последовательные порты Arduino	10
7. Последовательные входы Arduino	13
8. Аналоговые входы Arduino	15
9. Аналоговые выходы Arduino	16

1. Введение

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/батареями. Работа с набором не требует пайки.

2. 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.

Используются 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 питания микроконтроллера 5 В. Опираясь на данные параметры, выбирается светодиод с рабочим напряжением 1,7 В и рабочим током 15 мА.

Сопrotивление токоограничивающего резистора рассчитывается как отношение разности общего напряжения в цепи и напряжения светодиода к току в цепи. Таким образом, сопротивление равно $(5-1,7)/0,015 = 220 \text{ Ом}$.

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

Здесь и далее представлена работа на платформе Windows 7 (32-bit). Если вы используете другую платформу, скачивайте версию, которая подходит именно вам.

В корне папки должно быть несколько подпапок и файлов, как показано на скриншоте ниже.

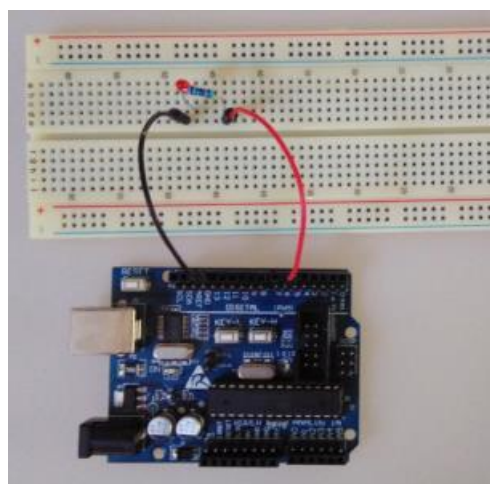
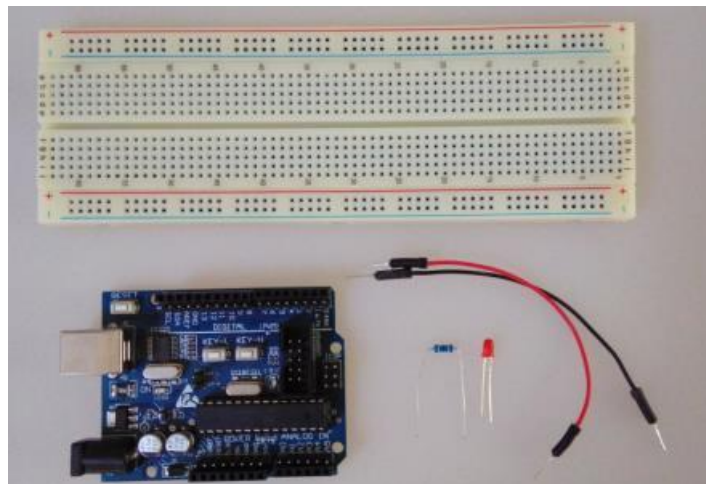
- Arduino.exe – это загрузочный файл;
- папка drivers содержит драйверы для USB-контроллера.

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	文件夹
drivers	2015-08-28 9:32	文件夹
examples	2015-08-28 9:32	文件夹
hardware	2015-08-28 9:32	文件夹
java	2015-08-28 9:32	文件夹
lib	2015-08-28 9:32	文件夹
libraries	2015-10-13 16:23	文件夹
reference	2015-08-28 9:32	文件夹
tools	2015-08-28 9:32	文件夹
arduino.exe	2015-08-28 9:32	应用程序
arduino.l4j.ini	2015-08-28 9:32	配置设置
arduino_debug.exe	2015-08-28 9:32	应用程序
arduino_debug.l4j.ini	2015-08-28 9:32	配置设置
libusb0.dll	2015-08-28 9:32	应用程序扩展

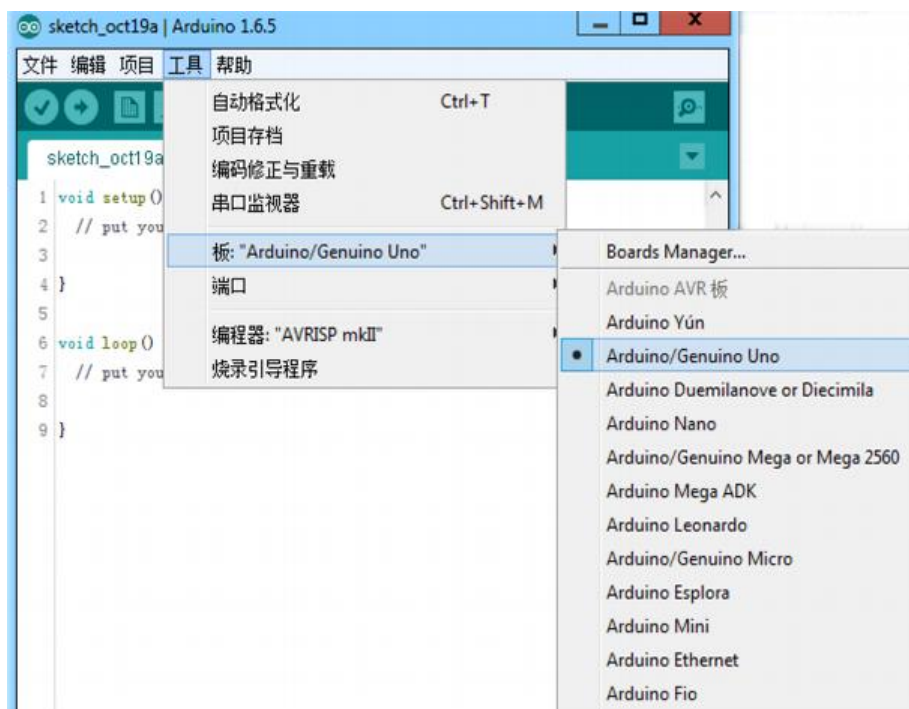
Оборудование, которое мы будем использовать в данном опыте, показано на снимке ниже: это макетная плата, светодиод, резистор 1кОм и 2 провода.

На втором снимке представлен способ подключения. У светодиода две контактных ножки: короткая 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.

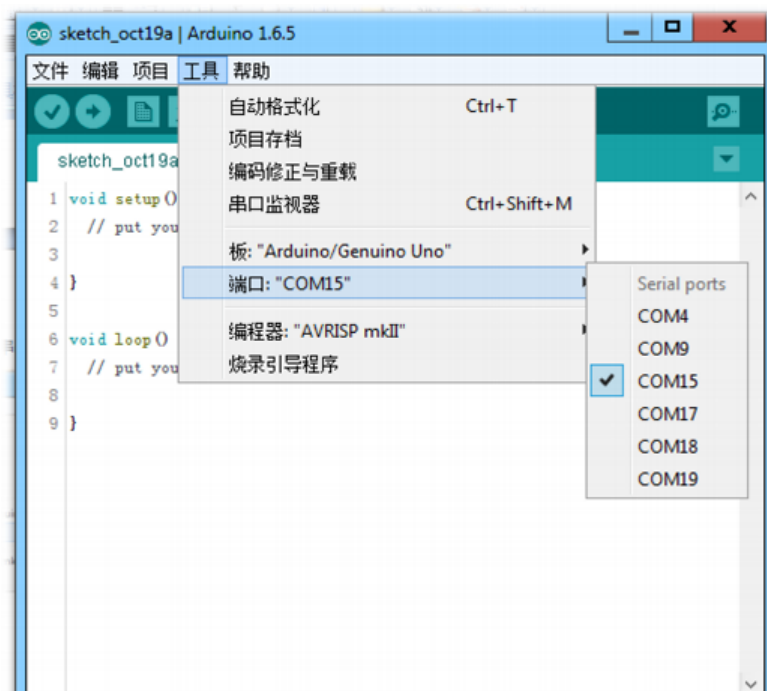


Все элементы подключаются так, как показано на снимке сверху.

После того, как все контактные входы элементов соединены через макетную плату, подключите линию USB и настройте все контакты и входы макетной платы. Перед написанием программы определите входы макетной платы, как указано на скриншотах ниже:



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



Уточните номер используемого последовательного порта в панели управления устройствами, как показано на скриншоте ниже:



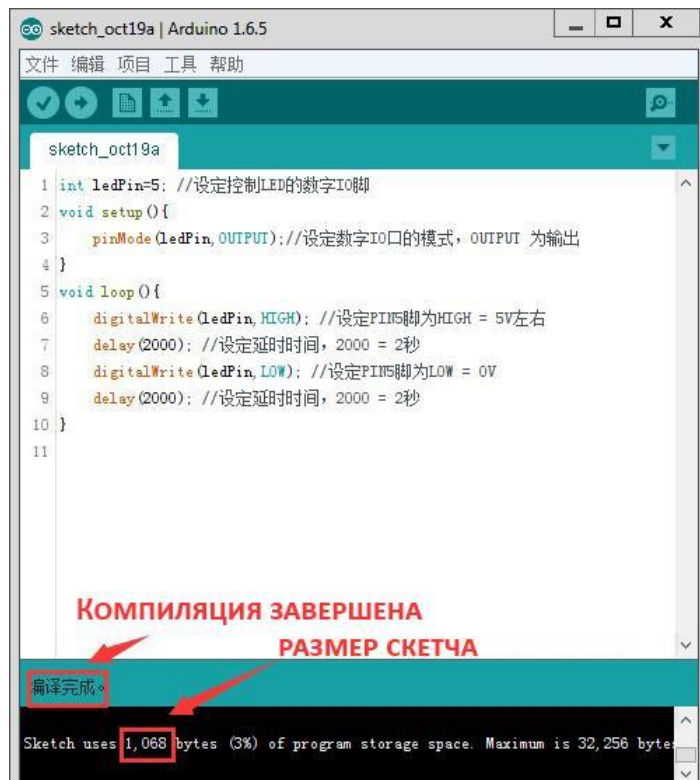
Сначала скопируйте в окно код программы (скетча):


```
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 



Резюме:

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.

3. Классическая программа «Hello, world!»

Проверка работоспособности 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!».

4. 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 ledPin=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

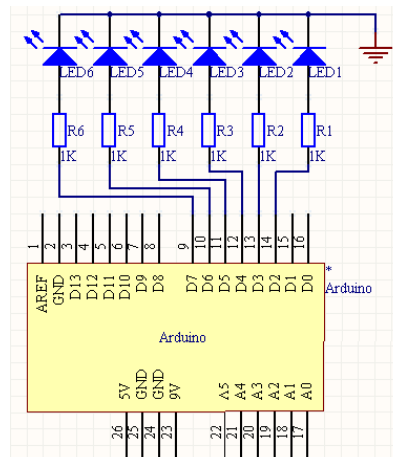
АЦП, что позволяет преобразовывать входное напряжение 0-5 В в целочисленное количество отсчетов от 0 до 1023.

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.

5. 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. Проверьте функциональность всех выводов I/O Arduino, мы используем бегущие огни на шести светодиодах. Схема показана ниже:

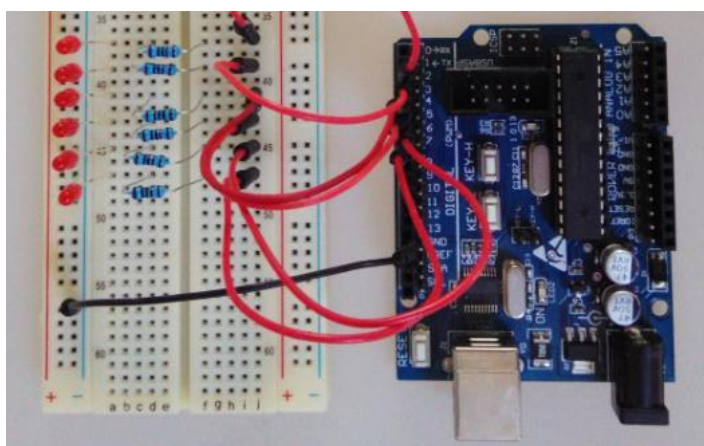


Чтобы увеличить эквивалентное сопротивление светодиодов, снизить протекающий в ветвях ток и исключить вероятность выхода светодиодов из строя, в каждую ветвь мы подключаем резисторы 1 кОм.

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.

6. Последовательные порты Arduino

Прием и передачу данных при подключении к ПК или микроконтроллеру проще всего осуществлять через последовательные порты (СОМ-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 PCMCIA-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. Скetch 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()`.

В языке Arduino есть функция `Serial.available()`, и она 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 интеракцию.

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

7. 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 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 между 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, можно 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);
}

```

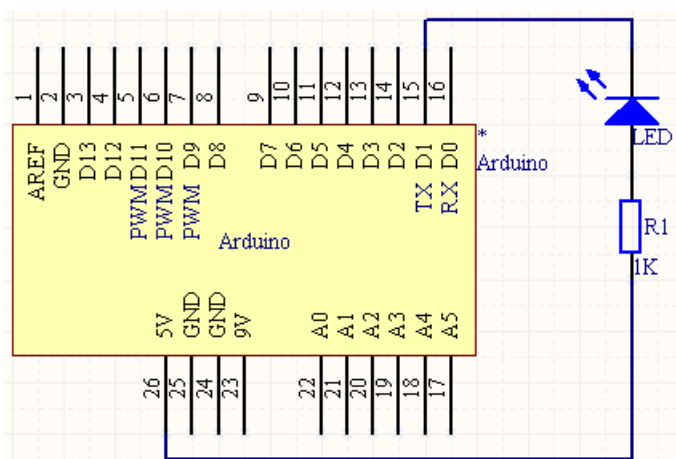
Загрузите скетч в модуль Arduino, после запуска в интегрированной среде разработки нажмите кнопку “Serial Monitor” на панели инструментов, откроется окно последовательного порта:



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.

8. Аналоговые входы 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).

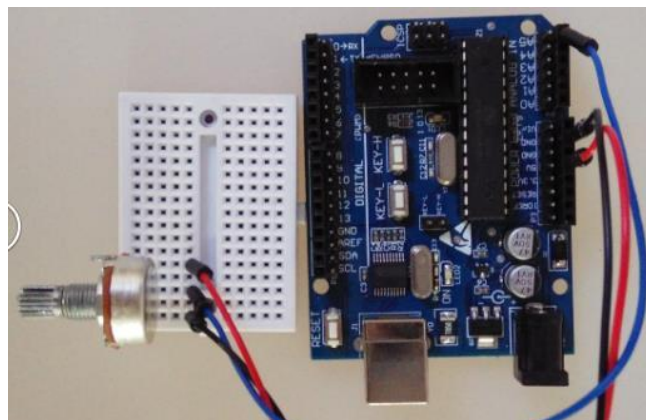
Returns возвращает значение функции.

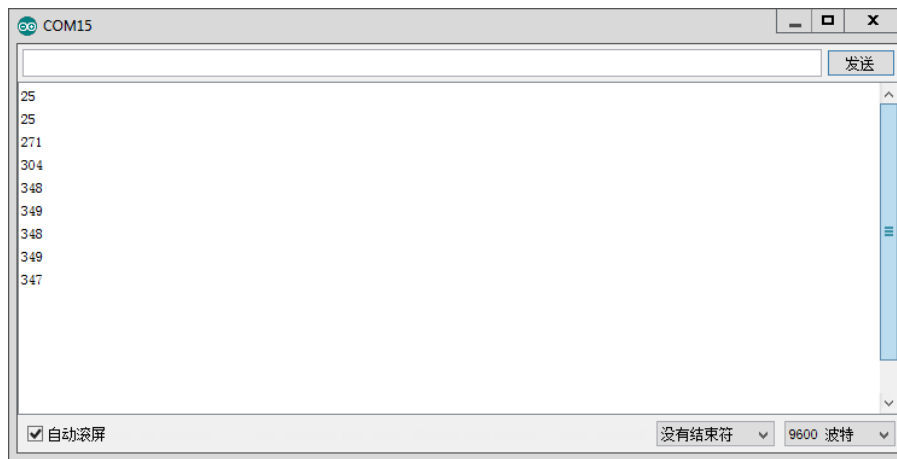
int (от 0 до 1023).

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);
}
```



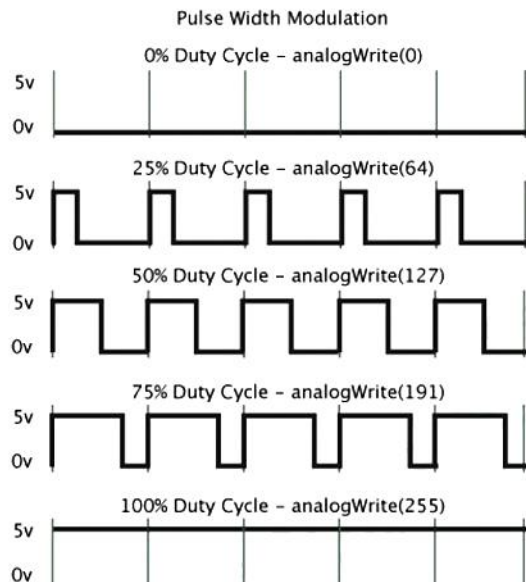


9. Аналоговые выводы Arduino

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 работают 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 `millis()` 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); }
```

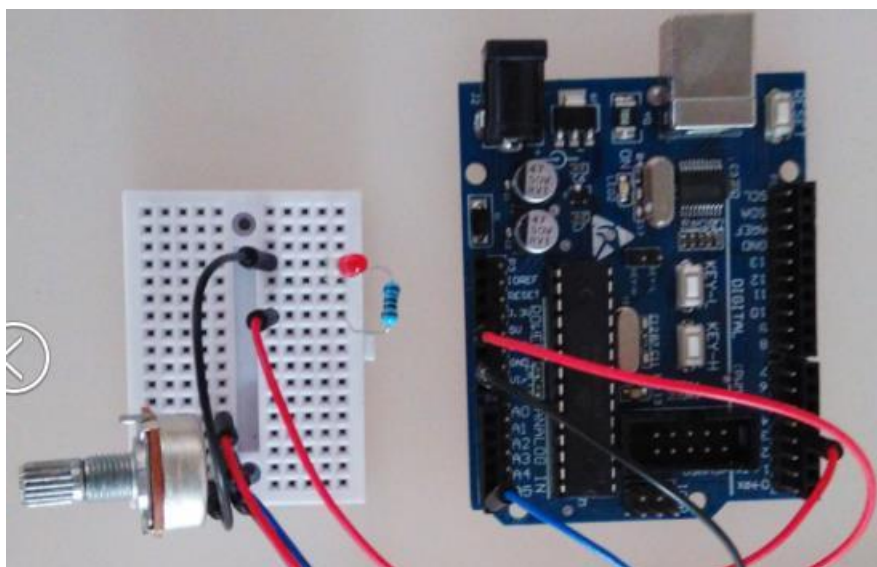
```
void loop()
```

```
{
```

```
val = analogRead(analogPin);
```

```
analogWrite(ledPin, val/4);
```

```
}
```



Приятного использования!

Сайт: minicam24.ru

E-mail: info@minicam24.ru

Товар в наличии в 120 городах России и Казахстана

Телефон бесплатной горячей линии: **8(800)200-85-66**