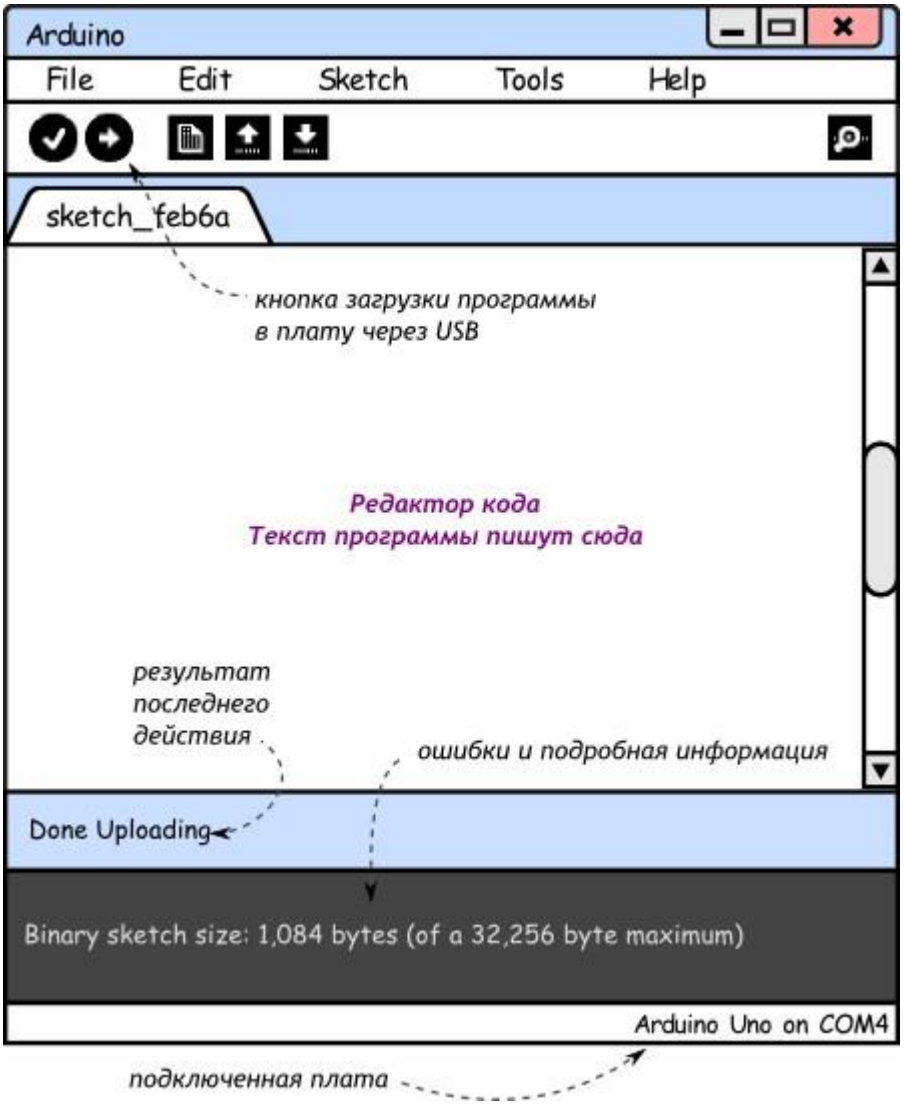
Начало работы с Arduino

1. Скачайте Arduino IDE с официального сайта (http://arduino.cc/en/Main/Software)
2. Если у вас Windows и Arduino IDE из zip-файла, установите драйверы из папки `drivers`
3. Подключите Arduino к компьютеру через USB
4. Запустите Arduino IDE
5. В «Tools → Board» выберите модель вашей платы
6. В «Tools → Serial Port» выберите порт, куда она подключена
7. Пишите программу или загружайте готовый пример из «File → Examples»
8. Жмите «Upload» на панели инструментов для прошивки платы!

   Перепрошивать плату можно сколько угодно раз. Программа сохраняется после обесточивания платы.

Внешний вид Arduino IDE



Если прошивка не удаётся

Проверьте, что:

- Плата получает питание, горит светодиод «ON»
- Драйверы под Windows установились корректно, и в диспетчере устройств вы видите устройство «Arduino Uno»
- Вы выбрали правильную модель платы и правильный порт (пункты 5 и 6)
- USB-кабель исправен

Эксперимент 1. Маячок

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 светодиод
- 1 резистор номиналом 220 Ом
- 2 провода «папа-папа»

Принципиальная схема

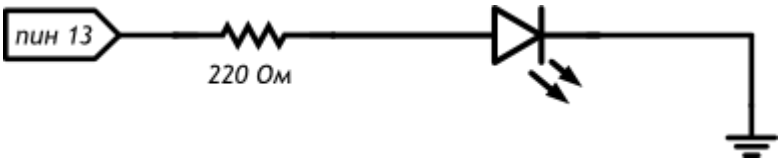
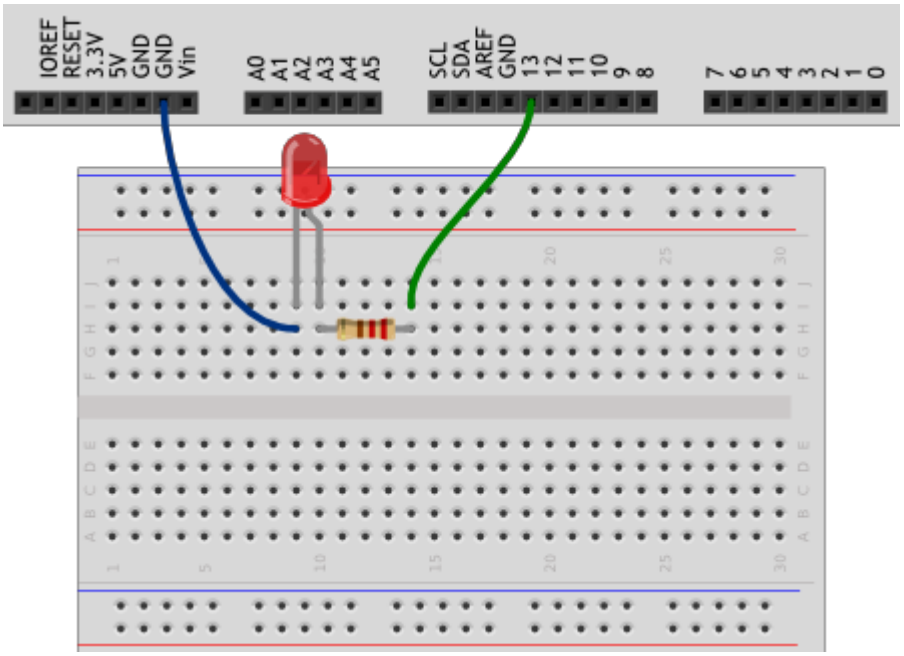


Схема на макетке



Обратите внимание

- Не забудьте, как соединены рельсы в беспаечной макетной плате. Если на вашей макетке красная и синяя линии вдоль длинных рельс прерываются в середине, значит проводник внутри макетки тоже прерывается!
- Катод («минус») светодиода — короткая ножка, именно её нужно соединять с землёй (GND)
- Не пренебрегайте резистором, иначе светодиод выйдет из строя

- Выбрать резистор нужного номинала можно с помощью таблицы маркировки или с помощью мультиметра в режиме измерения сопротивления
- Плата Arduino имеет три пина GND, используйте любой из них

Скетч

```
void setup()
{
  // настраиваем пин №13 в режим выхода,
  // т.е. в режим источника напряжения
  pinMode(13, OUTPUT);
}

void loop()
{
  // подаём на пин 13 «высокий сигнал» (англ. «high»), т.е.
  // выдаём 5 вольт. Через светодиод побежит ток.
  // Это заставит его светиться
  digitalWrite(13, HIGH);

  // задерживаем (англ. «delay») микроконтроллер в этом
  // состоянии на 100 миллисекунд
  delay(100);

  // подаём на пин 13 «низкий сигнал» (англ. «low»), т.е.
  // выдаём 0 вольт или, точнее, приравниваем пин 13 к земле.
  // В результате светодиод погаснет
  digitalWrite(13, LOW);

  // замираем в этом состоянии на 900 миллисекунд
  delay(900);

  // после «размораживания» loop сразу же начнёт исполняться
  // вновь, и со стороны это будет выглядеть так, будто
  // светодиод мигает раз в 100 мс + 900 мс = 1000 мс = 1 сек
}
```

Пояснения к коду

- Процедура `setup` выполняется один раз при запуске микроконтроллера. Обычно она используется для конфигурации портов микроконтроллера и других настроек
- После выполнения `setup` запускается процедура `loop`, которая выполняется в бесконечном цикле. Именно этим мы пользуемся в данном примере, чтобы маячок мигал постоянно
- Процедуры `setup` и `loop` должны присутствовать в любой программе (скетче), даже если вам не нужно ничего выполнять в них — пусть они будут пустые, просто не пишите ничего между фигурными скобками. Например:

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

- Запомните, что каждой открывающей фигурной скобке `{` всегда соответствует закрывающая `}`. Они обозначают границы некого логически завершенного фрагмента кода. Следите за вложенностью фигурных скобок. Для этого удобно после каждой открывающей скобки увеличивать отступ на каждой новой строке на один символ табуляции (клавиша Tab)
- Обращайте внимание на `;` в концах строк. Не стирайте их там, где они есть, и не добавляйте лишних. Вскоре вы будете понимать, где они нужны, а где нет.
- Функция `digitalWrite(pin, value)` не возвращает никакого значения и принимает два параметра:
  - `pin` — номер цифрового порта, на который мы отправляем сигнал
  - `value` — значение, которое мы отправляем на порт. Для цифровых портов значением может быть`HIGH` (высокое, единица) или `LOW` (низкое, ноль)
- Если в качестве второго параметра вы передадите функции `digitalWrite` значение, отличное от `HIGH`,`LOW`, `1` или `0`, компилятор может не выдать ошибку, но считать, что передано `HIGH`. Будьте внимательны
- Обратите внимание, что использованные нами константы: `INPUT`, `OUTPUT`, `LOW`, `HIGH`, пишутся заглавными буквами, иначе компилятор их не распознает и выдаст ошибку. Когда ключевое слово распознано, оно подсвечивается синим цветом в Arduino IDE

Вопросы для проверки себя

1. Что будет, если подключить к земле анод светодиода вместо катода?
2. Что будет, если подключить светодиод с резистором большого номинала (например, 10 кОм)?
3. Что будет, если подключить светодиод без резистора?
4. Зачем нужна встроенная функция `pinMode`? Какие параметры она принимает?
5. Зачем нужна встроенная функция `digitalWrite`? Какие параметры она принимает?
6. С помощью какой встроенной функции можно заставить микроконтроллер ничего не делать?
7. В каких единицах задается длительность паузы для этой функции?

Задания для самостоятельного решения

1. Сделайте так, чтобы маячок светился полсекунды, а пауза между вспышками была равна одной секунде
2. Измените код примера так, чтобы маячок включался на три секунды после запуска устройства, а затем мигал в стандартном режиме

Эксперимент 2. Маячок с нарастающей яркостью

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 светодиод
- 1 резистор номиналом 220 Ом
- 2 провода «папа-папа»

Для дополнительного задания

- еще 1 светодиод
- еще 1 резистор номиналом 220 Ом
- еще 2 провода

Принципиальная схема

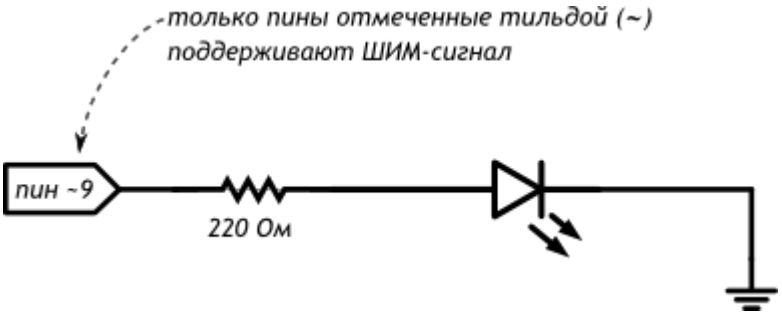
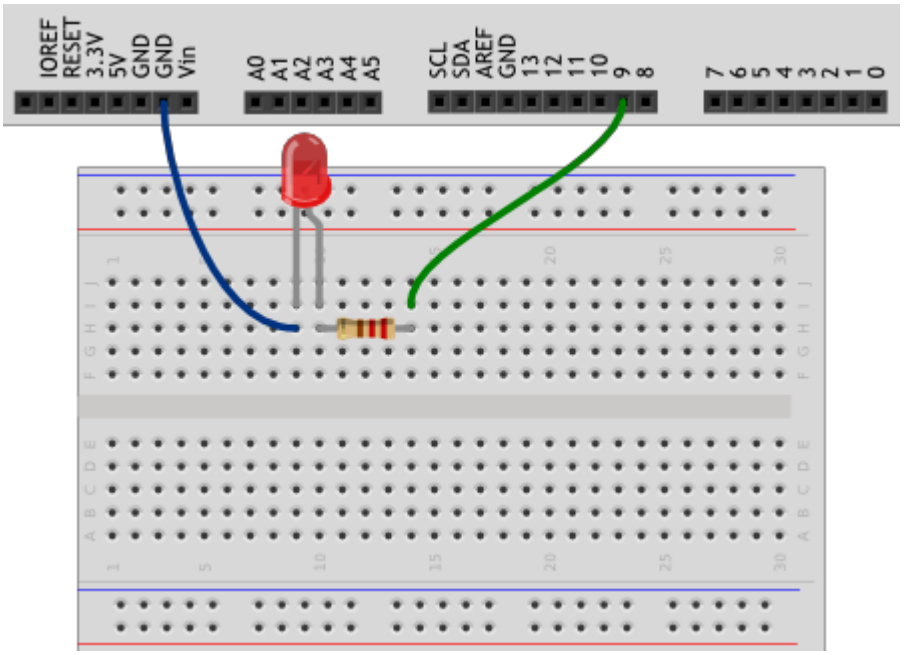


Схема на макетке



Обратите внимание

- Не любой порт Arduino поддерживает широтно-импульсную модуляцию, если вы хотите регулировать напряжение, вам подойдут пины, помеченные символом тильда «~». Для Arduino Uno это пины 3, 5, 6, 9, 10, 11

Скетч

```
// даём разумное имя для пина №9 со светодиодом
// (англ. Light Emitting Diode или просто «LED»)
// Так нам не нужно постоянно вспоминать куда он подключён
#define LED_PIN 9

void setup()
{
  // настраиваем пин со светодиодом в режим выхода,
  // как и раньше
  pinMode(LED_PIN, OUTPUT);
}

void loop()
{
  // выдаём неполное напряжение на светодиод
  // (он же ШИМ-сигнал, он же PWM-сигнал).
  // Микроконтроллер переводит число от 0 до 255 к напряжению
  // от 0 до 5 В. Например, 85 — это 1/3 от 255,
  // т.е. 1/3 от 5 В, т.е. 1,66 В.
  analogWrite(LED_PIN, 85);
  // держим такую яркость 250 миллисекунд
  delay(250);
```

```
  // выдаём 170, т.е. 2/3 от 255, или иными словами — 3,33 В.
  // Больше напряжение — выше яркость!
  analogWrite(LED_PIN, 170);
  delay(250);

  // все 5 В — полный накал!
  analogWrite(LED_PIN, 255);
  // ждём ещё немного перед тем, как начать всё заново
  delay(250);
}
```

Пояснения к коду

- Идентификаторы переменных, констант, функций (в этом примере идентификатор `LED_PIN`) являются одним словом (т.е. нельзя создать идентификатор `LED PIN`).
- Идентификаторы могут состоять из латинских букв, цифр и символов подчеркивания `_`. При этом идентификатор не может начинаться с цифры.

```
PRINT       // верно
PRINT_3D    // верно
MY_PRINT_3D // верно
_PRINT_3D   // верно
3D_PRINT    // ошибка
ПЕЧАТЬ_3Д   // ошибка
PRINT:3D    // ошибка
```

- Регистр букв в идентификаторе имеет значение. Т.е. `LED_PIN`, `LED_pi` и `led pin` с точки зрения компилятора — различные идентификаторы
- Идентификаторы, создаваемые пользователем, не должны совпадать с предопределенными идентификаторами и стандартными конструкциями языка; если среда разработки подсветила введенный идентификтор каким-либо цветом, замените его на другой
- Директива `#define` просто говорит компилятору заменить все вхождения заданного идентификатора на значение, заданное после пробела (здесь `9`), эти директивы помещают в начало кода. В конце данной директивы точка с запятой `;` не допустима
- Названия идентификаторов всегда нужно делать осмысленными, чтобы при возвращении к ранее написанному коду вам было ясно, зачем нужен каждый из них
- Также полезно снабжать код программы комментариями: в примерах мы видим однострочные комментарии, которые начинаются с двух прямых слэшей `//` и многострочные, заключённые между `/* */`

```
// однострочный комментарий следует после двойного слеша до конца строки
```

```
/* многострочный комментарий
   помещается между парой слеш-звездочка и звездочка-слеш */
```

комментарии игнорируются компилятором, зато полезны людям при чтении давно написанного, а особенно чужого, кода

- Функция `analogWrite(pin, value)` не возвращает никакого значения и принимает два параметра:
  - `pin` — номер порта, на который мы отправляем сигнал
  - `value` — значение скважности ШИМ, которое мы отправляем на порт. Он может принимать целочисленное значение от 0 до 255, где 0 — это 0%, а 255 — это 100%

Вопросы для проверки себя

1. Какие из следующих идентификаторов корректны и не вызовут ошибку?
   - `13pin`
   - `MOTOR 1`
   - `контакт_светодиода`
   - `sensor value`
   - `leftServo`
   - `my-var`
   - `distance eval2`
2. Что произойдет, если создать директиву `#define HIGH LOW`?
3. Почему мы не сможем регулировать яркость светодиода, подключенного к порту 7?
4. Какое усреднённое напряжение мы получим на пине 6, если вызовем функцию `analogWrite(6, 153)`?
5. Какое значение параметра `value` нужно передать функции `analogWrite`, чтобы получить усреднённое напряжение 2 В?

Задания для самостоятельного решения

1. Отключите питание, отключите светодиод от 9-го порта и подключите к 11-му. Измените программу так, чтобы схема снова заработала
2. Измените код программы так, чтобы в течение секунды на светодиод последовательно подавалось усреднённое напряжение 0, 1, 2, 3, 4, 5 В
3. Возьмите еще один светодиод, резистор на 220 Ом и соберите аналогичную схему на этой же макетке, подключив светодиод к пину номер 3 и другому входу GND, измените программу так, чтобы светодиоды мигали в противофазу: первый выключен, второй горит максимально ярко и до противоположного состояния

Эксперимент 3. Светильник с управляемой яркостью

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 светодиод
- 1 резистор номиналом 220 Ом
- 6 проводов «папа-папа»
- 1 потенциометр

Для дополнительного задания

- еще 1 светодиод
- еще 1 резистор номиналом 220 Ом
- еще 2 провода

Принципиальная схема

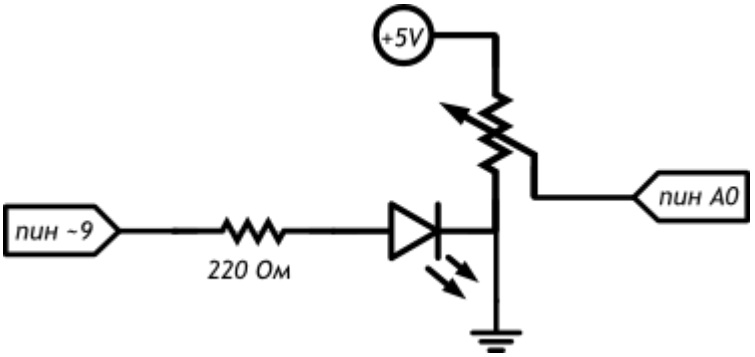
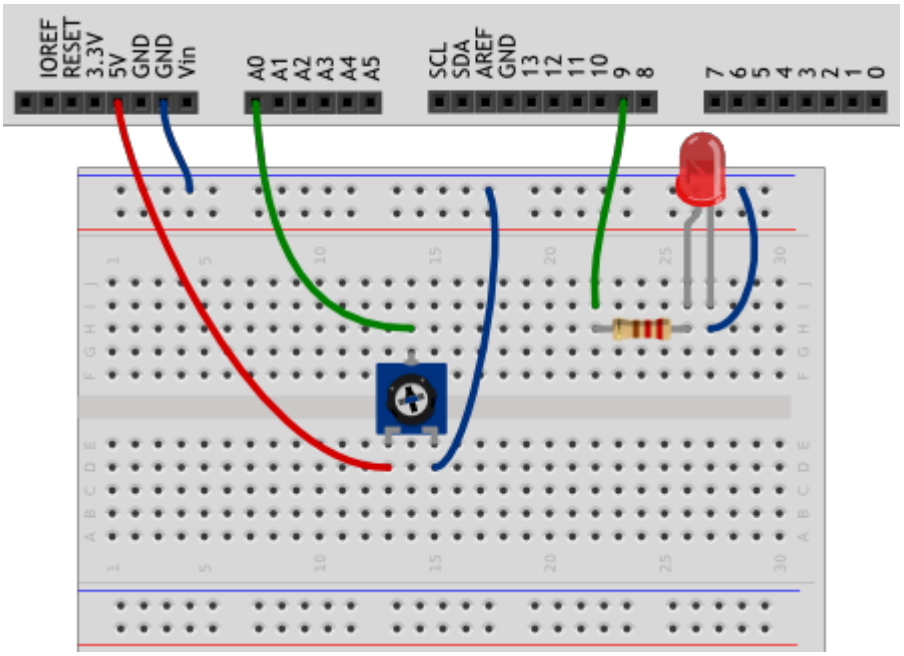


Схема на макетке



Обратите внимание

- Мы подключили «землю» светодиода и переменного резистора (потенциометра) к длинной рельсе «-» макетной платы, и уже ее соединили с входом GND микроконтроллера. Таким образом мы использовали меньше входов и от макетки к контроллеру тянется меньше проводов.
- Подписи «+» и «-» на макетке не обязывают вас использовать их строго для питания, просто чаще всего они используются именно так и маркировка нам помогает
- Не важно, какая из крайних ножек потенциометра будет подключена к 5 В, а какая к GND, поменяется только направление, в котором нужно крутить ручку для увеличения напряжения. Запомните, что сигнал мы считываем со средней ножки
- Для считывания аналогового сигнала, принимающего широкий спектр значений, а не просто 0 или 1, как цифровой, подходят только порты, помеченные на плате как «ANALOG IN» и пронумерованные с префиксом `A`. Для Arduino Uno — это A0-A5.

Скетч

```
// даём разумные имена для пинов со светодиодом
// и потенциометром (англ potentiometer или просто «pot»)
#define LED_PIN     9
#define POT_PIN     A0

void setup()
{
  // пин со светодиодом — выход, как и раньше...
  pinMode(LED_PIN, OUTPUT);
```

```
  // ...а вот пин с потенциометром должен быть входом
  // (англ. «input»): мы хотим считывать напряжение,
  // выдаваемое им
  pinMode(POT_PIN, INPUT);
}

void loop()
{
  // заявляем, что далее мы будем использовать 2 переменные с
  // именами rotation и brightness, и что хранить в них будем
  // целые числа (англ. «integer», сокращённо просто «int»)
  int rotation, brightness;

  // считываем в rotation напряжение с потенциометра:
  // микроконтроллер выдаст число от 0 до 1023
  // пропорциональное углу поворота ручки
  rotation = analogRead(POT_PIN);

  // в brightness записываем полученное ранее значение rotation
  // делённое на 4. Поскольку в переменных мы пожелали хранить
  // целые значения, дробная часть от деления будет отброшена.
  // В итоге мы получим целое число от 0 до 255
  brightness = rotation / 4;

  // выдаём результат на светодиод
  analogWrite(LED_PIN, brightness);
}
```

Пояснения к коду

- С помощью директивы `#define` мы сказали компилятору заменять идентификатор `POT_PIN` на `A0` — номер аналогового входа. Вы можете встретить код, где обращение к аналоговому порту будет по номеру без индекса `A`. Такой код будет работать, но во избежание путаницы с цифровыми портами используйте индекс.
- Переменным принято давать названия, начинающиеся со строчной буквы.
- Чтобы использовать переменную, необходимо ее объявить, что мы и делаем инструкцией:

```
int rotation, brightness;
```

- Для объявления переменной необходимо указать ее тип, здесь — `int` (от англ. integer) — целочисленное значение в диапазоне от -32 768 до 32 767, с другими типами мы познакомимся позднее

- Переменные одного типа можно объявить в одной инструкции, перечислив их через запятую, что мы и сделали
- Функция `analogRead(pinA)` возвращает целочисленное значение в диапазоне от 0 до 1023, пропорциональное напряжению, поданному на аналоговый вход, номер которого мы передаем функции в качестве параметра `pinA`
- Обратите внимание, как мы получили значение, возвращенное функцией `analogRead()`: мы просто поместили его в переменную `rotation` с помощью оператора присваивания `=`, который записывает то, что находится справа от него в ту переменную, которая стоит слева

Вопросы для проверки себя

1. Можем ли мы при сборке схемы подключить светодиод и потенциометр напрямую к разным входам GND микроконтроллера?
2. В какую сторону нужно крутить переменный резистор для увеличения яркости светодиода?
3. Что будет, если стереть из программы строчку `pinMode(LED_PIN, OUTPUT)`? строчку `pinMode(POT_PIN, INPUT)`?
4. Зачем мы делим значение, полученное с аналогового входа перед тем, как задать яркость светодиода? что будет, если этого не сделать?

Задания для самостоятельного решения

1. Отключите питание платы, подключите к порту 5 еще один светодиод. Измените код таким образом, чтобы второй светодиод светился на 1/8 от яркости первого

Эксперимент 4. Терменвокс

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 пьезопищалка
- 6 проводов «папа-папа»
- 1 резистор номиналом 10 кОм
- 1 фоторезистор

Принципиальная схема

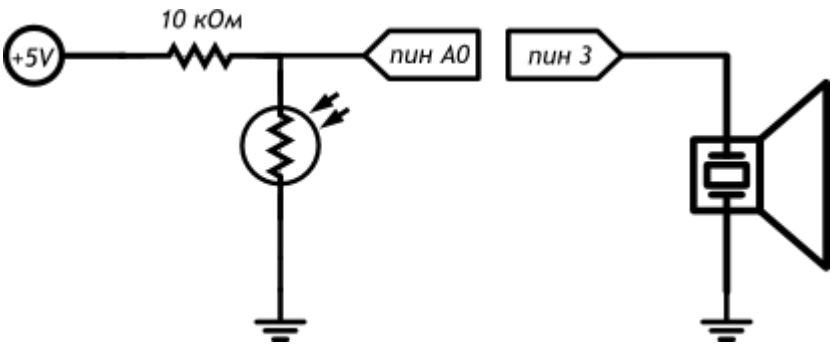
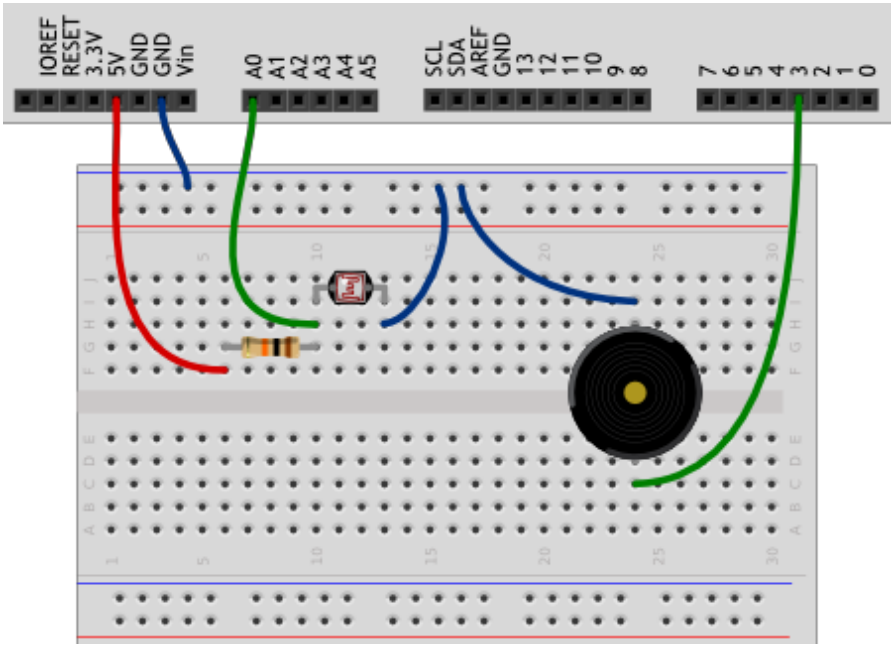


Схема на макетке



Обратите внимание

- В данной схеме мы используем резистор нового номинала, посмотрите таблицу маркировки, чтобы найти резистор на 10 кОм или воспользуйтесь мультиметром
- Полярность фоторезистора, как и обычного резистора, не играет роли. Его можно устанавливать любой стороной
- В данном упражнении мы собираем простой вариант схемы включения пьезодинамика
- Полярность пьезопищалки роли не играет: вы можете подключать любую из ее ножек к земле, любую к порту микроконтроллера
- На Arduino Uno использование функции `tone` мешает использованию ШИМ на 3-м и 11-м портах. Зато можно подключить ее к одному из них
- Вспомните как устроен делитель напряжения: фоторезистор помещается в позицию R2 — между аналоговым входом и землей. Так мы получаем резистивный фотосенсор.

Скетч

```
// даём имена для пинов с пьезопищалкой (англ. buzzer) и фото-
// резистором (англ. Light Dependent Resistor или просто LDR)
#define BUZZER_PIN  3
#define LDR_PIN     A0

void setup()
{
  // пин с пьезопищалкой — выход...
  pinMode(BUZZER_PIN, OUTPUT);

  // ...а все остальные пины являются входами изначально,
  // всякий раз при подаче питания или сбросе микроконтроллера.
  // Поэтому, на самом деле, нам совершенно необязательно
  // настраивать LDR_PIN в режим входа: он и так им является
}

void loop()
{
  int val, frequency;

  // считываем уровень освещённости так же, как для
  // потенциометра: в виде значения от 0 до 1023.
  val = analogRead(LDR_PIN);

  // рассчитываем частоту звучания пищалки в герцах (ноту),
  // используя функцию проекции (англ. map). Она отображает
  // значение из одного диапазона на другой, строя пропорцию.
  // В нашем случае [0; 1023] -> [3500; 4500]. Так мы получим
  // частоту от 3,5 до 4,5 кГц.
  frequency = map(val, 0, 1023, 3500, 4500);
```

```
    // заставляем пин с пищалкой «вибрировать», т.е. звучать
    // (англ. tone) на заданной частоте 20 миллисекунд. При
    // следующих проходах loop, tone будет вызван снова и снова,
    // и на деле мы услышим непрерывный звук тональностью, которая
    // зависит от количества света, попадающего на фоторезистор
    tone(BUZZER_PIN, frequency, 20);
}
```

Пояснения к коду

- Функция `map(value, fromLow, fromHigh, toLow, toHigh)` возвращает целочисленное значение из интервала [`toLow`, `toHigh`], которое является пропорциональным отображением содержимого `value`из интервала [`fromLow`, `fromHigh`]
- Верхние границы `map` не обязательно должны быть больше нижних и могут быть отрицательными. К примеру, значение из интервала [1, 10] можно отобразить в интервал [10,-5]
- Если при вычислении значения `map` образуется дробное значение, оно будет отброшено, а не округлено
- Функция `map`не будет отбрасывать значения за пределами указанных диапазонов, а также масштабирует их по заданному правилу.
- Если вам нужно ограничить множество допустимых значений, используйте функцию `constrain(value, from, to)`, которая вернет:
  - `value`, если это значение попадает в диапазон [`from`, `to`]
  - `from`, если `value` меньше него
  - `to`, если `value` больше него
- Функция `tone(pin, frequency, duration)` заставляет пьезопищалку, подключенную к порту `pin`, издавать звук высотой `frequency` герц на протяжении `duration` миллисекунд
- Параметр `duration` не является обязательным. Если его не передать, звук включится навсегда. Чтобы его выключить, вам понадобится функция `noTone(pin)`. Ей нужно передать номер порта с пищалкой, которую нужно выключить
- Одновременно можно управлять только одной пищалкой. Если во время звучания вызвать `tone` для другого порта, ничего не произойдет.
- Вызов `tone` для уже звучащего порта обновит частоту и длительность звучания

Вопросы для проверки себя

1. Каким сопротивлением должен обладать фоторезистор, чтобы на аналоговый вход было подано напряжение 1 В?
2. Можем ли мы регулировать яркость светодиода, подключенного к 11-му порту, во время звучания пьезопищалки?

3. Что изменится в работе терменвокса, если заменить резистор на 10 кОм резистором на 100 кОм? Попробуйте ответить без эксперимента. Затем отключите питание, замените резистор и проверьте.
4. Каков будет результат вызова `map(30,0,90,90,-90)`?
5. Как будет работать вызов `tone` без указания длительности звучания?
6. Можно ли устроить полифоническое звучание с помощью функции `tone`?

Задания для самостоятельного решения

1. Уберите из программы чтение датчика освещенности и пропищите азбукой Морзе позывной SOS: три точки, три тире, три точки
2. Измените код программы так, чтобы с падением освещенности звук становился ниже (например, падал от 5 кГц до 2,5 кГц)
3. Измените код программы так, чтобы звук терменвокса раздавался не непрерывно, а 10 раз в секунду с различимыми паузами

Эксперимент 5. Ночной светильник

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 светодиод
- 1 фоторезистор
- 1 резистор номиналом 220 Ом
- 1 резистор номиналом 10 кОм
- 1 переменный резистор (потенциометр)
- 10 проводов «папа-папа»

Для дополнительного задания

- еще 1 светодиод
- еще 1 резистор номиналом 220 Ом
- еще 2 провода

Принципиальная схема

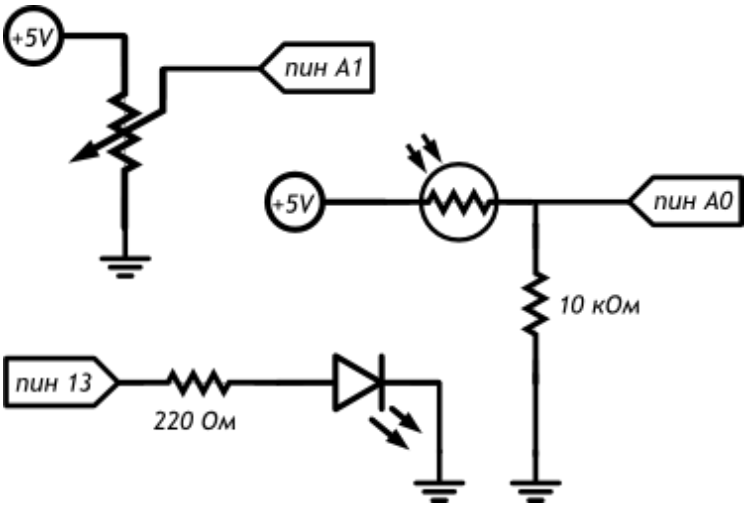
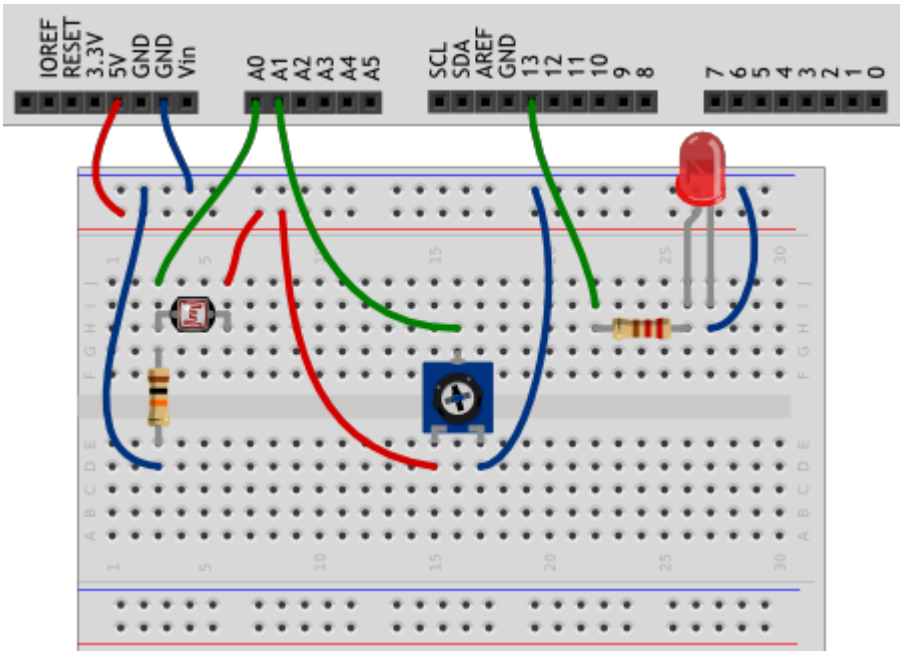


Схема на макетке



Обратите внимание

- В этом эксперименте мы устанавливаем фоторезистор между питанием и аналоговым входом, т.е. в позицию R1 в схеме делителя напряжения. Это нам нужно для того, чтобы при уменьшении освещенности мы получали меньшее напряжение на аналоговом входе.
- Постарайтесь разместить компоненты так, чтобы светодиод не засвечивал фоторезистор.

Скетч

```
#define LED_PIN  13
#define LDR_PIN  A0
#define POT_PIN  A1

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

void loop()
{
  // считываем уровень освещённости. Кстати, объявлять
  // переменную и присваивать ей значение можно разом
  int lightness = analogRead(LDR_PIN);

  // считываем значение с потенциометра, которым мы регулируем
  // пороговое значение между условными темнотой и светом
```

```
  int threshold = analogRead(POT_PIN);

  // объявляем логическую переменную и назначаем ей значение
  // «темно ли сейчас». Логические переменные, в отличие от
  // целочисленных, могут содержать лишь одно из двух значений:
  // истину (англ. true) или ложь (англ. false). Такие значения
  // ещё называют булевыми (англ. boolean).
  boolean tooDark = (lightness < threshold);

  // используем ветвление программы: процессор исполнит один из
  // двух блоков кода в зависимости от исполнения условия.
  // Если (англ. «if») слишком темно...
  if (tooDark) {
    // ...включаем освещение
    digitalWrite(LED_PIN, HIGH);
  } else {
    // ...иначе свет не нужен — выключаем его
    digitalWrite(LED_PIN, LOW);
  }
}
```

Пояснения к коду

- Мы используем новый тип переменных — `boolean`, которые хранят только значения `true` (истина, 1) или `false` (ложь, 0). Эти значения являются результатом вычисления логических выражений. В данном примере логическое выражение — это `lightness < threshold`. На человеческом языке это звучит как: «освещенность ниже порогового уровня». Такое высказывание будет истинным, когда освещенность ниже порогового уровня. Микроконтроллер может сравнить значения переменных `lightness` и`threshold`, которые, в свою очередь, являются результатами измерений, и вычислить истинность логического выражения.
- Мы взяли это логическое выражение в скобки только для наглядности. Всегда лучше писать читабельный код. В других случаях скобки могут влиять на порядок действий, как в обычной арифметике.
- В нашем эксперименте логическое выражение будет истинным, когда значение `lightness` меньше значения `threshold`, потому что мы использовали оператор `<`. Мы можем использовать операторы `>`,`<=`, `>=`, `==`, `!=`, которые значат «больше», «меньше или равно», «больше или равно», «равно», «не равно» соответственно.
- Будьте особенно внимательны с логическим оператором `==` и не путайте его с оператором присваивания`=`. В первом случае мы сравниваем значения выражений и получаем логическое значение (истина или ложь), а во втором случае присваиваем левому операнду значение

правого. Компилятор не знает наших намерений и ошибку не выдаст, а мы можем нечаянно изменить значение какой-нибудь переменной и затем долго разыскивать ошибку.

- Условный оператор `if` («если») — один из ключевых в большинстве языков программирования. С его помощью мы можем выполнять не только жестко заданную последовательность действий, но принимать решения, по какой ветви алгоритма идти, в зависимости от неких условий.
- У логического выражения `lightness < threshold` есть значение: `true` или `false`. Мы вычислили его и поместили в булеву переменную `tooDark` («слишком темно»). Таким образом мы как бы говорим «если слишком темно, то включить светодиод»
- С таким же успехом мы могли бы сказать «если освещенность меньше порогового уровня, то включить светодиод», т.е. передать в `if` всё логическое выражение:

```
if (lightness < threshold) {
  // ...
}
```

- За условным оператором `if` обязательно следует блок кода, который выполняется в случае истинности логического выражения. Не забывайте про обе фигурные скобки `{}`!
- Если в случае истинности выражения нам нужно выполнить только *одну* инструкцию, ее можно написать сразу после `if (…)` без фигурных скобок:

```
if (lightness < threshold)
  digitalWrite(LED_PIN, HIGH);
```

- Оператор `if` может быть расширен конструкцией `else` («иначе»). Блок кода или единственная инструкция, следующий за ней, будет выполнен только если логическое выражение в `if` имеет значение `false`, «ложь». Правила, касающиеся фигурных скобок, такие же. В нашем эксперименте мы написали «если слишком темно, включить светодиод, иначе выключить светодиод».

Вопросы для проверки себя

1. Если мы установим фоторезистор между аналоговым входом и землей, наше устройство будет работать наоборот: светодиод будет включаться при увеличении количества света. Почему?
2. Какой результат работы устройства мы получим, если свет от светодиода будет падать на фоторезистор?

3. Если мы все же установили фоторезистор так, как сказано в предыдущем вопросе, как нам нужно изменить программу, чтобы устройство работало верно?
4. Допустим, у нас есть код `if (условие) {действие;}`. В каких случаях будет выполнено `действие`?
5. При каких значениях `y` выражение `x + y > 0` будет истинным, если `x > 0`?
6. Обязательно ли указывать, какие инструкции выполнять, если условие в операторе `if` ложно?
7. Чем отличается оператор `==` от оператора `=`?
8. Если мы используем конструкцию `if (условие) действие1; else действие2;`, может ли быть ситуация, когда ни одно из действий не выполнится? Почему?

Задания для самостоятельного решения

1. Перепишите программу без использования переменной `tooDark` с сохранением функционала устройства.
2. Добавьте в схему еще один светодиод. Дополните программу так, чтобы при падении освещенности ниже порогового значения включался один светодиод, а при падении освещенности ниже половины от порогового значения включались оба светодиода.
3. Измените схему и программу так, чтобы светодиоды включались по прежнему принципу, но светились тем сильнее, чем меньше света падает на фоторезистор.

Эксперимент 6. Пульсар

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 биполярный транзистор
- 1 светодиодная шкала
- 1 резистор номиналом 1 кОм
- 10 резисторов номиналом 220 Ом
- 13 проводов «папа-папа»

Принципиальная схема

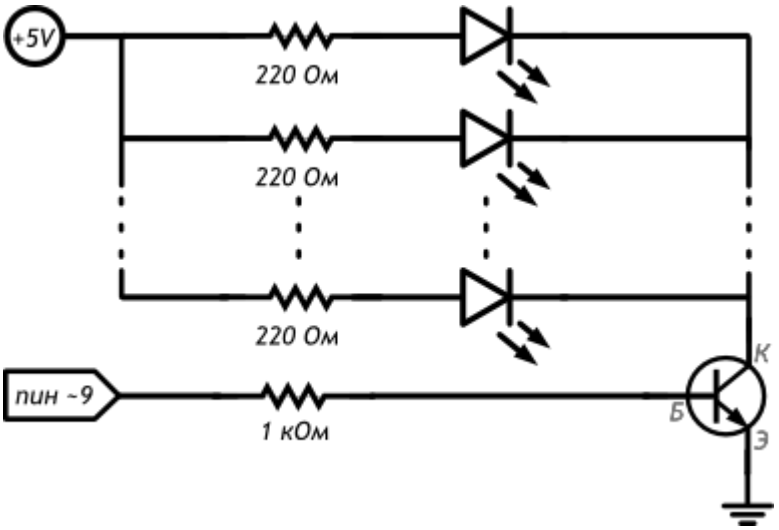
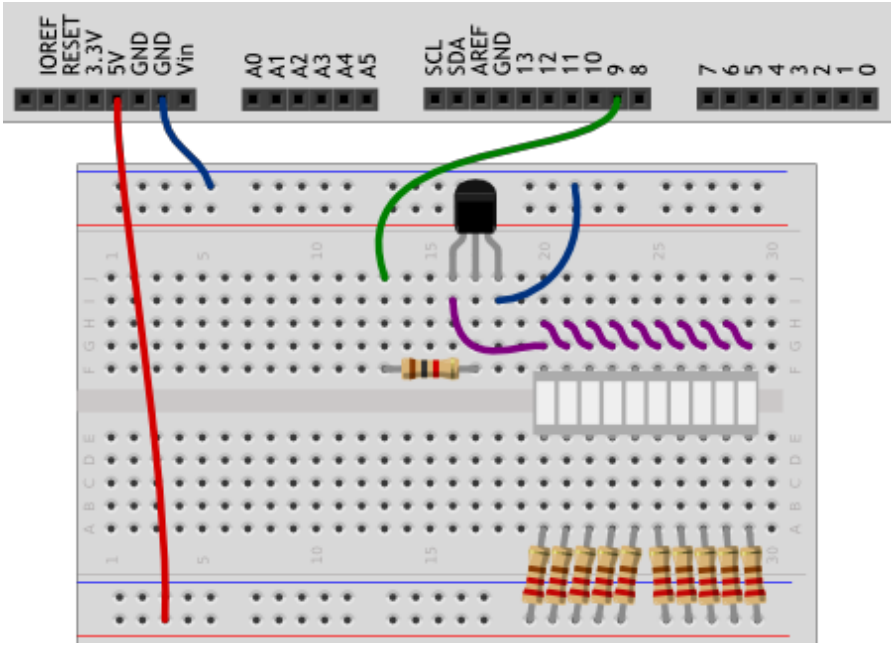


Схема на макетке



Обратите внимание

- Светодиодная шкала — это несколько светодиодов в одном корпусе. Нам нужно чтобы питание шло к их анодам, а катоды направлялись к земле. Скорее всего на вашей шкале аноды находятся со стороны маркировки. Если шкала не светится, когда должна, попробуйте перевернуть ее.
- База биполярного транзистора — это его средняя ножка. Если повернуть транзистор плоской стороной к себе, ножками вниз, то левая ножка это коллектор, а правая — эмиттер.
- Если эту схему собрать без резистора между базой транзистора и портом Arduino, мы практически устроим короткое замыкание порта на землю. Рано или поздно это выведет из строя транзистор или ножку микроконтроллера.
- Зачем здесь вообще транзистор? Без него такое количество светодиодов будет потреблять больше тока, чем 40 мА, которые может себе позволить цифровой пин платы. Поэтому мы берем питание из порта 5V, рассчитанного на ток до 500 мА, а на цифровой порт ставим транзистор, чтобы с помощью малого тока управлять большим.
- В данном случае мы включили 10 светодиодов параллельно, каждый через отдельный резистор. Включать их через один резистор неправильно: даже светодиоды из одной партии имеют минимальный разброс вольт-амперных характеристик, вследствие чего они:
  - Светились бы с различной яркостью
  - Из-за минимальной разницы во времени включения, больший ток, прошедший через первый включившийся светодиод, мог бы вывести его из строя. И так по цепочке.

Скетч

```
#define CONTROL_PIN  9

// переменные верхнего уровня, т.е. объявленные вне функций,
// называют глобальными. Их значения сохраняются всё время,
// пока работает микроконтроллер
int brightness = 0;

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

void loop()
{
  // увеличиваем значение яркости на единицу, чтобы нарастить
  // яркость. Однако яркость не должна быть более 255, поэтому
  // используем операцию остатка от деления, чтобы при
  // достижении значения 255, следующим значением снова стал 0
  // Y % X — это остаток от деления Y на X;
  // плюс, минус, делить, умножить, скобки — как в алгебре.
  brightness = (brightness + 1) % 256;
```

```
    // подаём вычисленный ШИМ-сигнал яркости на пин с базой
    // управляющего транзистора
    analogWrite(CONTROL_PIN, brightness);

    // ждём 10 мс перед следующим наращиванием яркости. Таким
    // образом, полный накал будет происходить в течение
    // 256×10 = 2560 мс
    delay(10);
}
```

Пояснения к коду

- Как мы уже знаем, `analogWrite(pin, value)` в качестве `value` принимает значения от 0 до 255. Если передать значение из-за пределов этого диапазона, функция сработает, но в общем случае вы получите неожиданный результат.
- Оператор `X % Y` дает остаток от деления `X` на `Y`. Если `X` меньше `Y`, т.е. целая часть результата деления равна 0, оператор `%` будет возвращать `X`. Таким образом:
  - Пока `brightness + 1` меньше 256, в `brightness` записывается значение `brightness + 1`
  - Как только `brightness + 1` принимает значение 256, результатом `(brightness + 1) % 256`становится 0 и на следующей итерации `loop()` всё начинается сначала.
- Оператор `%` работает только с целыми операндами.
- В выражении `(brightness + 1) % 256` скобки используются для назначения порядка действий. Операция `%` имеет больший приоритет, чем `+`, а сложение нам нужно выполнять раньше. С операциями умножения и деления оператор взятия остатка имеет одинаковый приоритет.

Вопросы для проверки себя

1. Почему у светодиодной шкалы на 10 сегментов 20 ножек?
2. Зачем в схеме биполярный транзистор?
3. За счет чего увеличивается яркость шкалы?
4. Почему после достижения значения 255 переменная `brightness` обнуляется?

Задания для самостоятельного решения

1. Измените программу так, чтобы яркость шкалы росла только до половины от максимальной.
2. Измените программу так, чтобы шкала становилась максимально яркой в три раза быстрее, без изменения функции `delay`.
3. Измените исходную программу так, чтобы такой же результат был получен без использования операции`%`, но с применением условного оператора `if`.

Эксперимент 7. Бегущий огонёк

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 светодиодная шкала
- 10 резисторов номиналом 220 Ом
- 11 проводов «папа-папа»

Принципиальная схема

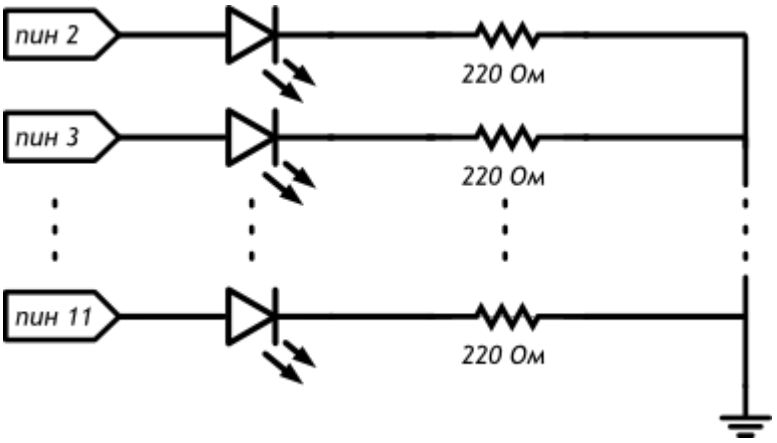
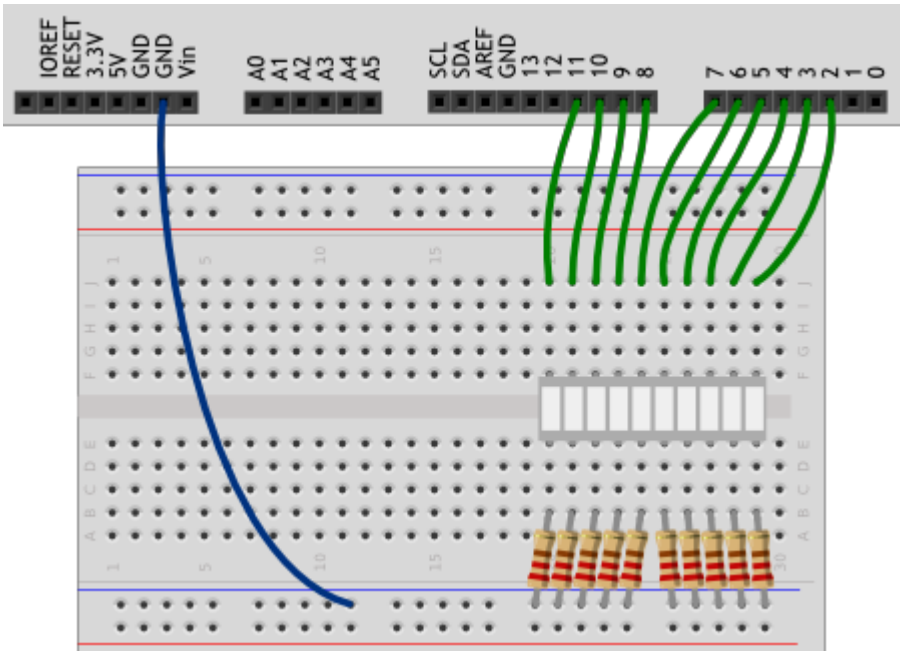


Схема на макетке



Обратите внимание

- Обратите внимание, что в данном эксперименте резисторы установлены между катодами и землей в отличие от эксперимента пульсар.

- Мы подключаем светодиоды к цифровым портам, начиная с порта 2. Мы можем использовать порты 0 и 1, но они являются каналами передачи данных последовательного порта и для каждой перепрошивки платы придется отключать устройства, подключенные к ним.

Скетч

```
// светодиодная шкала подключена к группе пинов расположенных
// подряд. Даём понятные имена первому и последнему пинам
#define FIRST_LED_PIN  2
#define LAST_LED_PIN   11

void setup()
{
  // в шкале 10 светодиодов. Мы бы могли написать pinMode 10
  // раз: для каждого из пинов, но это бы раздуло код и
  // сделало его изменение более проблематичным.
  // Поэтому лучше воспользоваться циклом. Мы выполняем
  // pinMode для (англ. for) каждого пина (переменная pin)
  // от первого (= FIRST_LED_PIN) до последнего включительно
  // (<= LAST_LED_PIN), всякий раз продвигаясь к следующему
  // (++pin увеличивает значение pin на единицу)
  // Так все пины от 2-го по 11-й друг за другом станут выходами
  for (int pin = FIRST_LED_PIN; pin <= LAST_LED_PIN; ++pin)
    pinMode(pin, OUTPUT);
}

void loop()
{
  // получаем время в миллисекундах, прошедшее с момента
  // включения микроконтроллера
  unsigned int ms = millis();
  // нехитрой арифметикой вычисляем, какой светодиод
  // должен гореть именно сейчас. Смена будет происходить
  // каждые 120 миллисекунд. Y % X — это остаток от
  // деления Y на X; плюс, минус, скобки — как в алгебре.
  int pin = FIRST_LED_PIN + (ms / 120) % 10;
  // включаем нужный светодиод на 10 миллисекунд, затем —
  // выключаем. На следующем проходе цикла он снова включится,
  // если гореть его черёд, и мы вообще не заметим отключения
  digitalWrite(pin, HIGH);
  delay(10);
  digitalWrite(pin, LOW);
}
```

Пояснения к коду

- С помощью выражения `for` мы организуем *цикл со счетчиком*. В данном случае для настройки портов на выход. Чтобы сделать такой цикл, нужно:

    - Инициализировать переменную-счетчик, присвоив ей первоначальное значение. В нашем случае: `int pin = FIRST_LED_PIN`
    - Указать условие, до достижения которого будет повторяться цикл. В нашем случае: `pin <= LAST_LED_PIN`
    - Определить правило, по которому будет изменяться счетчик. В нашем случае `++pin` (см. ниже об операторе `++`).
- Например, можно сделать цикл `for (int i = 10; i > 0; i = i - 1)`. В этом случае:
    - Переменной `i` присваивается значение 10
    - Это значение удовлетворяет условию `i > 0`
    - Поэтому блок кода, помещенный в цикл, выполняется первый раз
    - Значение `i` уменьшается на единицу, согласно заданному правилу, и принимает значение 9
    - Блок кода выполняется второй раз.
    - Всё повторяется снова и снова вплоть до значения `i` равного 0
    - Когда `i` станет равна 0, условие `i > 0` не выполнится, и выполнение цикла закончится
    - Контроллер перейдет к коду, следующему за циклом `for`
- Помещайте код, который нужно зациклить, между парой фигурных скобок `{}`, если в нем больше одной инструкции.
- Переменная-счетчик, объявляемая в операторе `for`, может использоваться внутри цикла. Например, в данном эксперименте `pin` последовательно принимает значения от 2 до 11 и, будучи переданной в `pinMode`, позволяет настроить 10 портов одной строкой, помещенной в цикл.
- Переменные-счетчики видны только внутри цикла. Т.е. если обратиться к `pin` до или после цикла, компилятор выдаст ошибку о необъявленной переменной.
- Конструкция `i = i - 1` в пояснении выше не является уравнением! Мы используем оператор присваивания `=` для того, чтобы в переменную `i` поместить значение, равное текущему значению `i`, уменьшенному на 1.
- Выражение `++pin` — это т.н. оператор *инкремента*, примененный к переменной `pin`. Эта инструкция даст тот же результат, что `pin = pin + 1`
- Аналогично инкременту работает оператор *декремента* `--`, уменьшающий значение на единицу. Подробнее об этом в статье про арифметические операции.
- Тип данных `unsigned int` используют для хранения целых чисел без знака, т.е. только *неотрицательных*. За счет лишнего бита, который теперь не используется для хранения знака, мы можем хранить в переменной такого типа значения до 65 535.
- Функция `millis` возвращает количество миллисекунд, прошедших с момента включения или перезагрузки микроконтроллера. Здесь мы используем ее для отсчета времени между переключениями светодиодов.

- С помощью выражения `(ms / 120) % 10` мы определяем, который из 10 светодиодов должен гореть сейчас. Перефразируя, мы определяем какой отрезок длиной в 120 мс идет сейчас и каков его номер внутри текущего десятка. Мы добавляем порядковый номер отрезка к номеру того порта, который в текущем наборе выступает первым.
- То, что мы гасим светодиод с помощью `digitalWrite(pin, LOW)` всего через 10 мс после включения не заметно глазу, т.к. очень скоро будет вновь вычислено, какой из светодиодов включать, и он будет включен — только что погашенный или следующий.

Вопросы для проверки себя

1. Почему в данном эксперименте мы подключаем светодиодную шкалу, не используя транзистор?
2. Если бы мы включали светодиоды только на портах 5, 6, 7, 8, 9, что нужно было бы изменить в программе?
3. С помощью какой другой инструкции можно выполнить действие, эквивалентное `++pin`?
4. В чем разница между переменными типов `int` и `unsigned int`?
5. Что возвращает функция `millis()`?
6. Как в данном эксперименте мы вычисляем номер порта, на котором нужно включить светодиод?

Задания для самостоятельного решения

1. Измените код так, чтобы светодиоды переключались раз в секунду.
2. Не выключая порты, сделайте так, чтобы огонёк бежал только по средним четырем делениям шкалы.
3. Переделайте программу так, чтобы вместо `int pin = FIRST_LED_PIN + (ms / 120) % 10`перемещением огонька управлял цикл `for`
4. Не меняя местами провода, измените программу так, чтобы огонёк бегал в обратном направлении.

Эксперимент 8. Мерзкое пианино

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 пьезопищалка
- 3 тактовых кнопки
- 3 резистора номиналом 10 кОм
- 10 проводов «папа-папа»

Для дополнительного задания

- еще 2 кнопки
- еще 2 резистора номиналом 10 кОм
- еще 2 провода

Принципиальная схема

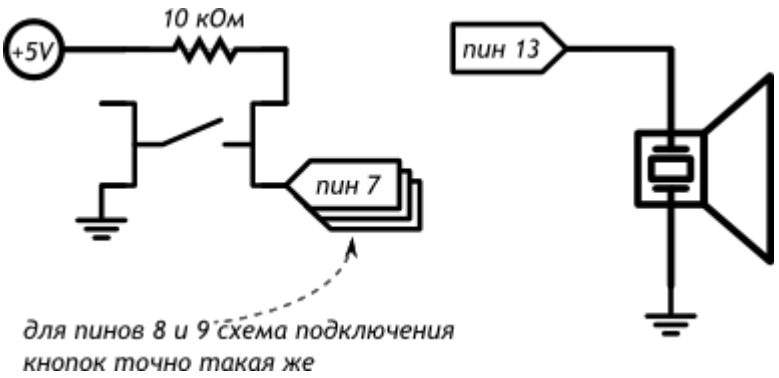
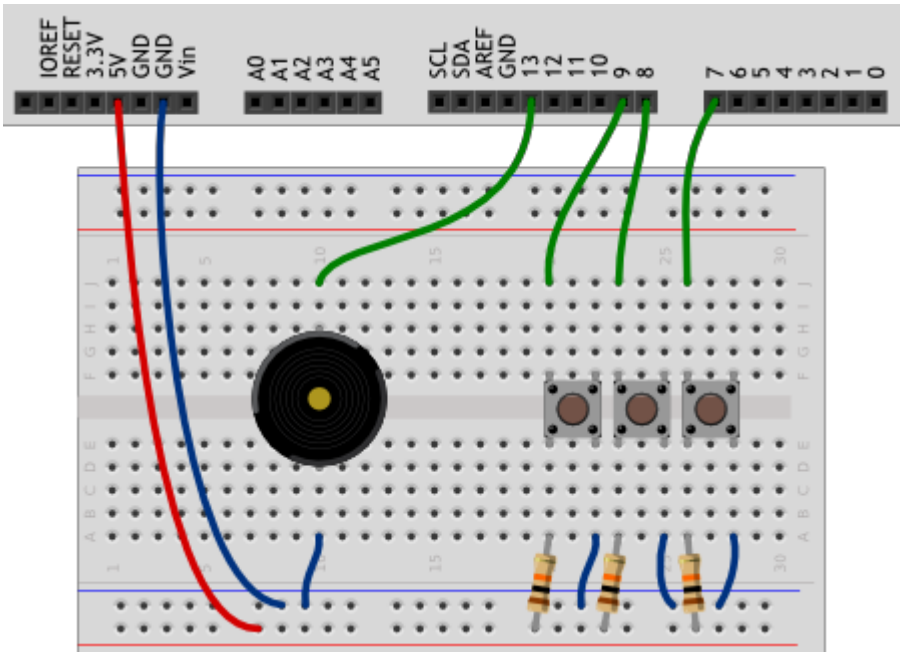


Схема на макетке



Обратите внимание

- Ножки тактовой кнопки, расположенные с одной стороны, разомкнуты, когда кнопка не нажата. Ножки, расположенные друг напротив друга на противоположных сторонах макетки находятся на одной «рельсе». Воспользовавшись этим, мы можем расположить резистор с одной стороны макетки, а провод, подключаемый к порту Arduino, с другой стороны.
- В данном эксперименте мы подключаем кнопки по схеме с подтягивающим резистором.
- Для того, чтобы данный вариант программы работал, важно, чтобы кнопки были подключены к портам, находящимся рядом друг с другом, т.е. имеющим соседние номера.

Скетч

```
#define BUZZER_PIN    13 // пин с пищалкой (англ. «buzzer»)
#define FIRST_KEY_PIN 7  // первый пин с клавишей (англ. «key»)
#define KEY_COUNT     3  // общее количество клавиш

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

void loop()
{
  // в цикле бежим по всем номерам кнопок от 0-го по 2-й
  for (int i = 0; i < KEY_COUNT; ++i) {
    // на основе номера кнопки вычисляем номер её пина
    int keyPin = i + FIRST_KEY_PIN;

    // считываем значение с кнопки. Возможны всего 2 варианта:
    //  * высокий сигнал, 5 вольт, истина — кнопка отпущена
    //  * низкий сигнал, земля, ложь — кнопка зажата
    boolean keyUp = digitalRead(keyPin);

    // проверяем условие «если не кнопка отпущена». Знак «!»
    // перед булевой переменной означает отрицание, т.е. «не».
    if (!keyUp) {
      // рассчитываем высоту ноты в герцах в зависимости от
      // клавиши, которую рассматриваем на данном этапе цикла.
      // Мы получим значение 3500, 4000 или 4500
      int frequency = 3500 + i * 500;

      // Заставляем пищалку пищать с нужной частотой в течение
      // 20 миллисекунд. Если клавиша останется зажатой, пищалка
      // вновь зазвучит при следующем проходе loop, а мы услышим
      // непрерывный звук
      tone(BUZZER_PIN, frequency, 20);
    }
```

```
    }
}
```

Пояснения к коду

- Благодаря тому, что в начале программы мы определили `FIRST_KEY_PIN` и `KEY_COUNT`, мы можем подключать произвольное количество кнопок к любым идущим друг за другом цифровым пинам, и для корректировки программы нам не придется менять параметры цикла `for`. Изменить понадобится лишь эти константы:
  - цикл в любом случае пробегает от 0 до `KEY_COUNT`;
  - перед считыванием порта мы задаем смещение на номер первого используемого порта — `FIRST_KEY_PIN`.
- Функция `digitalRead(pin)` возвращает состояние порта, номер которого передан ей параметром `pin`. Это может быть состояние `HIGH` или `LOW`. Или, выражаясь иначе: высокое напряжение или низкое, 1 или 0, `true` или `false`
- Поскольку мы получаем с порта одно из двух состояний, мы сохраняем его в переменную уже знакомого нам типа `boolean`, и можем работать с ней как с логическим значением.
- Мы используем логический оператор отрицания «не» `!`. Если `keyUp` имеет значение 0, выражение `!keyUp` будет иметь значение 1 и наоборот.
- Поскольку мы собрали схему с подтягивающим резистором, при нажатии кнопки мы будем получать на соответствующем порте 0.
- Действия, описанные в условном выражении `if`, выполняются, когда его условие имеет значение «истина» (единица). Поэтому для выполнения действия по нажатию, мы инвертируем сигнал с кнопки.

Вопросы для проверки себя

1. Почему мы не настраивали порты, к которым подключены кнопки, как `INPUT`, но устройство работает?
2. Каким образом мы избежали написания отдельного когда для чтения каждой кнопки?
3. Почему разные «ноты», издаваемые пищалкой, звучат с разной громкостью?
4. Для чего мы использовали оператор логического отрицания `!`?

Задания для самостоятельного решения

1. Сделайте так, чтобы наше пианино звучало в диапазоне от 2 кГц до 5 кГц.
2. Добавьте еще 2 кнопки и измените программу так, чтобы можно было извлечь 5 различных нот.
3. Подключите кнопки по схеме со стягивающим резистором и измените программу так, чтобы она продолжала работать.

Эксперимент 9. Миксер

Список деталей для эксперимента

- 1 плата Arduino Uno
- беспаечная макетная плата
- 3 тактовых кнопки
- 1 коллекторный двигатель
- 1 выпрямительный диод
- 1 полевой MOSFET-транзистор
- 15 проводов «папа-папа»
- 1 клеммник, если вы используете мотор с проводами, которые плохо втыкаются в макетку

Для дополнительного задания

- еще 1 кнопка
- еще 2 провода

Принципиальная схема

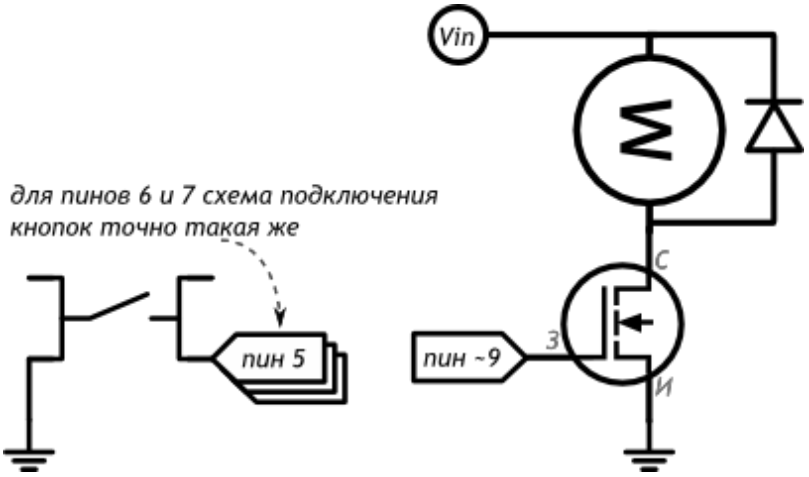
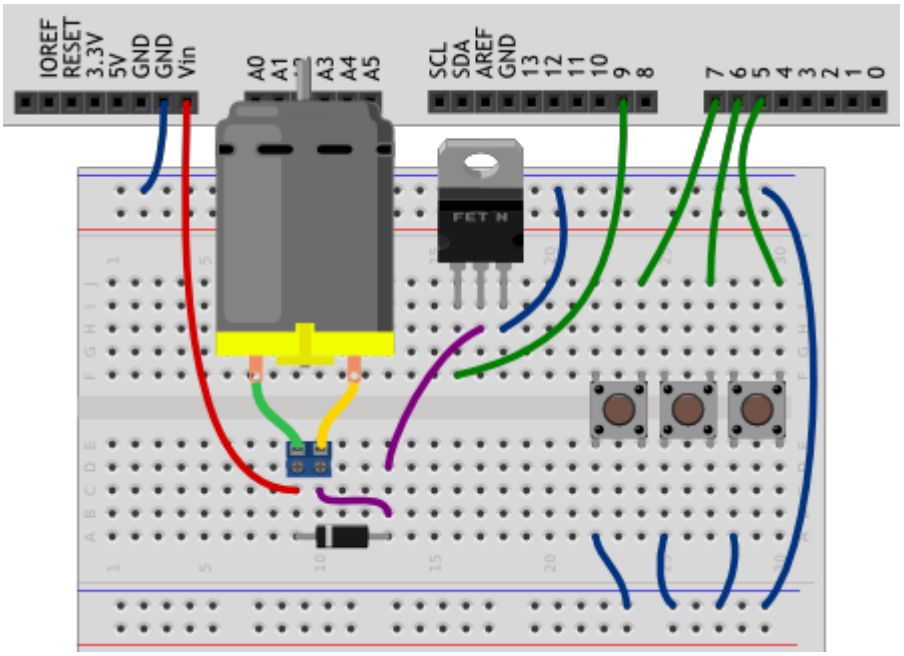


Схема на макетке



Обратите внимание

- Защитный диод нам нужен для того, чтобы ток обратного направления, который начнет создавать двигатель, вращаясь по инерции, не вывел из строя транзистор.
- Не перепутайте полярность диода, иначе, открыв транзистор, вы устроите короткое замыкание!
- Причину отсутствия подтягивающих/стягивающих резисторов в схеме вы поймете, ознакомившись с программой.
- Мы подключили питание схемы к выходу Vin платы микроконтроллера, потому что, в отличие выхода 5V, отсюда можно получить напряжение, подключенное к плате, без изменений и без ограничений по величине тока.

Скетч

```
#define MOTOR_PIN         9
#define FIRST_BUTTON_PIN 5
#define BUTTON_COUNT      3
// имена можно давать не только числам, но и целым выражениям.
// Мы определяем с каким шагом (англ. step) нужно менять
// скорость (англ. speed) мотора при нажатии очередной кнопки
#define SPEED_STEP  (255 / (BUTTON_COUNT - 1))

void setup()
{
  pinMode(MOTOR_PIN, OUTPUT);
  // на самом деле, в каждом пине уже есть подтягивающий
  // резистор. Для его включения необходимо явно настроить пин
  // как вход с подтяжкой (англ. input with pull up)
```

```
  for (int i = 0; i < BUTTON_COUNT; ++i)
    pinMode(i + FIRST_BUTTON_PIN, INPUT_PULLUP);
}

void loop()
{
  for (int i = 0; i < BUTTON_COUNT; ++i) {
    // если кнопка отпущена, нам она не интересна. Пропускаем
    // оставшуюся часть цикла for, продолжая (англ. continue)
    // его дальше, для следующего значения i
    if (digitalRead(i + FIRST_BUTTON_PIN))
      continue;

    // кнопка нажата — выставляем соответствующую ей скорость
    // мотора. Нулевая кнопка остановит вращение, первая
    // заставит крутиться в полсилы, вторая — на полную
    int speed = i * SPEED_STEP;

    // подача ШИМ-сигнала на мотор заставит его крутиться с
    // указанной скоростью: 0 — стоп машина, 127 — полсилы,
    // 255 — полный вперёд!
    analogWrite(MOTOR_PIN, speed);
  }
}
```

Пояснения к коду

- Мы использовали новый режим работы портов: `INPUT_PULLUP`. На цифровых портах Arduino есть встроенные подтягивающие резисторы, которые можно включить указанным образом одновременно с настройкой порта на вход. Именно поэтому мы не использовали резисторы при сборке схемы.
- На каждой итерации цикла мы задаем мотору скорость вращения, пропорциональную текущему значению счетчика. Но выполнение инструкций не дойдет до назначения новой скорости, если при проверке нажатия кнопки она окажется отпущенной.

  Инструкция `continue`, которая выполнится в этом случае, отменит продолжение данной итерации цикла и выполнение программы продолжится со следующей. А мотор будет крутиться со скоростью, заданной при последнем нажатии на какую-то из кнопок.

Вопросы для проверки себя

1. Зачем в схеме использован диод?
2. Почему мы использовали полевой MOSFET-транзистор, а не биполярный?
3. Почему мы не использовали резистор между портом Arduino и затвором транзистора?
4. Как работает инструкция `continue`, использованная в цикле `for`?

Задания для самостоятельного решения

1. Внесите единственное изменение в программу, после которого максимальной скоростью вращения мотора составит половину от возможной.
2. Перепишите программу без использования инструкции `continue`.
3. Добавьте в схему еще одну кнопку, чтобы у миксера стало три режима. Понадобилось ли изменять что-либо в программе?

Эксперимент 10. Кнопочный переключатель

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 тактовая кнопка
- 1 резистор номиналом 220 Ом
- 1 светодиод
- 5 проводов «папа-папа»

Для дополнительного задания

- еще 1 кнопка
- еще 2 провода

Принципиальная схема

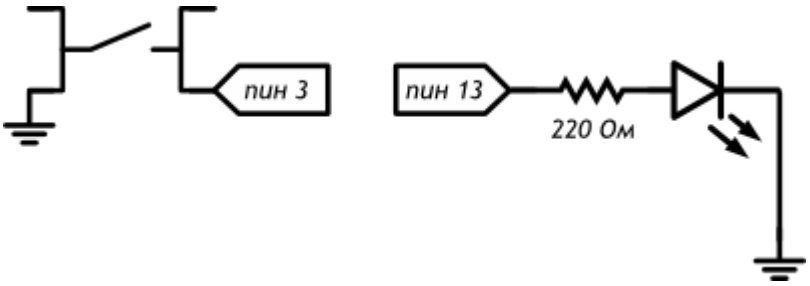
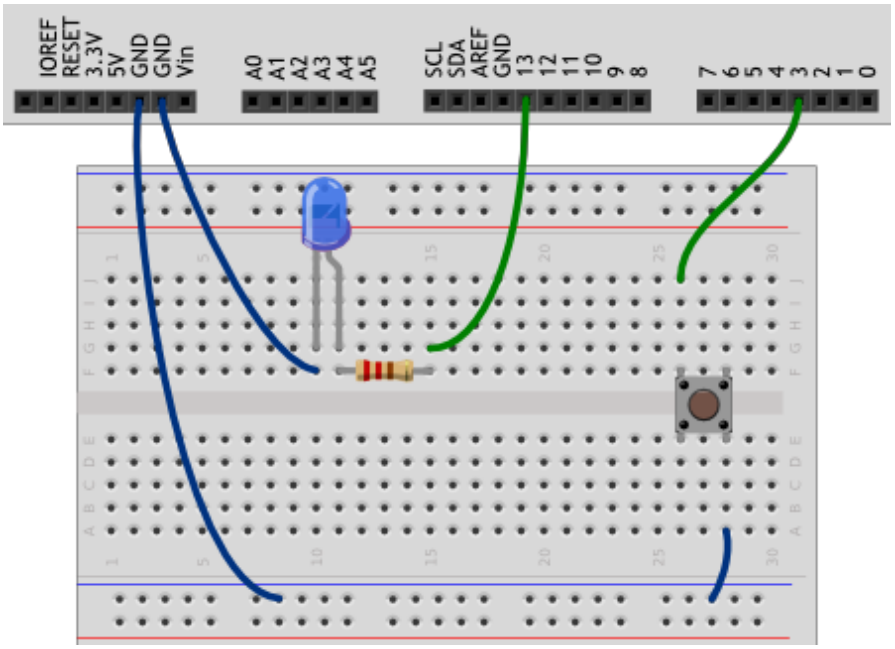


Схема на макетке



Обратите внимание

- Мы могли бы один из контактов кнопки соединить проводом напрямую с одним из входов GND, но мы сначала «раздали» «землю» на длинную рельсу макетки. Если мы работаем с макетной платой, так поступать удобнее, т.к. в схеме могут появляться новые участки, которые тоже нужно будет соединить с «землей»
- Также полезно руководствоваться соображениями аккуратности изделия, поэтому катод светодиода мы соединяем с другим входом GND отдельным проводом, который не мешает нам работать в середине макетки.

Скетч

```
#define BUTTON_PIN  3
#define LED_PIN     13

boolean buttonWasUp = true;   // была ли кнопка отпущена?
boolean ledEnabled = false;   // включен ли свет?

void setup()
{
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
}

void loop()
{
  // определить момент «клика» несколько сложнее, чем факт того,
  // что кнопка сейчас просто нажата. Для определения клика мы
  // сначала понимаем, отпущена ли кнопка прямо сейчас...
  boolean buttonIsUp = digitalRead(BUTTON_PIN);

  // ...если «кнопка была отпущена и (&&) не отпущена сейчас»...
  if (buttonWasUp && !buttonIsUp) {
    // ...может это «клик», а может и ложный сигнал (дребезг),
    // возникающий в момент замыкания/размыкания пластин кнопки,
    // поэтому даём кнопке полностью «успокоиться»...
    delay(10);
    // ...и считываем сигнал снова
    buttonIsUp = digitalRead(BUTTON_PIN);
    if (!buttonIsUp) {  // если она всё ещё нажата...
      // ...это клик! Переворачиваем сигнал светодиода
      ledEnabled = !ledEnabled;
      digitalWrite(LED_PIN, ledEnabled);
    }
  }

  // запоминаем последнее состояние кнопки для новой итерации
```

```
  buttonWasUp = buttonIsUp;
}
```

Пояснения к коду

- Поскольку мы сконфигурировали вход кнопки как `INPUT_PULLUP`, при нажатии на кнопку на данном входе мы будем получать 0. Поэтому мы получим значение `true` («истина») в булевой переменной`buttonIsUp` («кнопка отпущена»), когда кнопка отпущена.
- Логический оператор `&&` («и») возвращает значение «истина» только в случае истинности обоих его операндов. Взглянем на так называемую таблицу истинности для выражения `buttonWasUp && !buttonIsUp` («кнопка была отпущена и кнопка не отпущена»):

`buttonWasUp`	`buttonIsUp`	`!buttonIsUp`	`buttonWasUp && !buttonIsUp`
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

  Здесь рассмотрены все возможные сочетания предыдущего и текущего состояний кнопки и мы видим, что наш условный оператор `if` сработает только в случае, когда кнопка нажата только что: предыдущее состояние 1 («была отпущена»), а текущее 0 («не отпущена»).
- Через 10 миллисекунд мы проверяем еще раз, нажата ли кнопка: этот интервал больше, чем длительность «дребезга», но меньше, чем время, за которое человек успел бы дважды нажать на кнопку. Если кнопка всё еще нажата, значит, это был не дребезг.
- Мы передаем в `digitalWrite` не конкретное значение `HIGH` или `LOW`, а просто булеву переменную`ledEnabled`. В зависимости от того, какое значение было для нее вычислено, светодиод будет зажигаться или гаситься.
- Последняя инструкция в `buttonWasUp = buttonIsUp` сохраняет текущее состояние кнопки в переменную предыдущего состояния, ведь на следующей итерации `loop` текущее состояние уже станет историей.

Вопросы для проверки себя

1. В каком случае оператор `&&` возвращает значение «истина»?
2. Что такое «дребезг»?
3. Как мы с ним боремся в программе?
4. Как можно избежать явного указания значения уровня напряжения при вызове `digitalWrite`?

Задания для самостоятельного решения

1. Измените код так, чтобы светодиод переключался только после отпускания кнопки.
2. Добавьте в схему еще одну кнопку и доработайте код, чтобы светодиод зажигался только при нажатии обеих кнопок.

Эксперимент 11. Светильник с кнопочным управлением

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 2 тактовых кнопки
- 1 резистор номиналом 220 Ом
- 1 светодиод
- 7 проводов «папа-папа»

Принципиальная схема

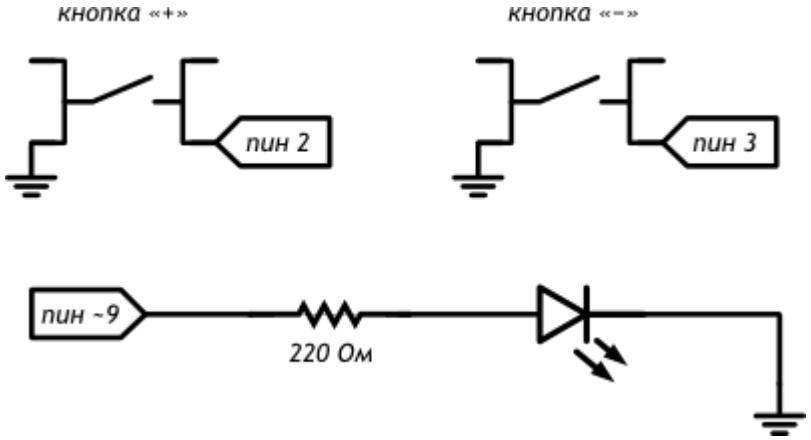
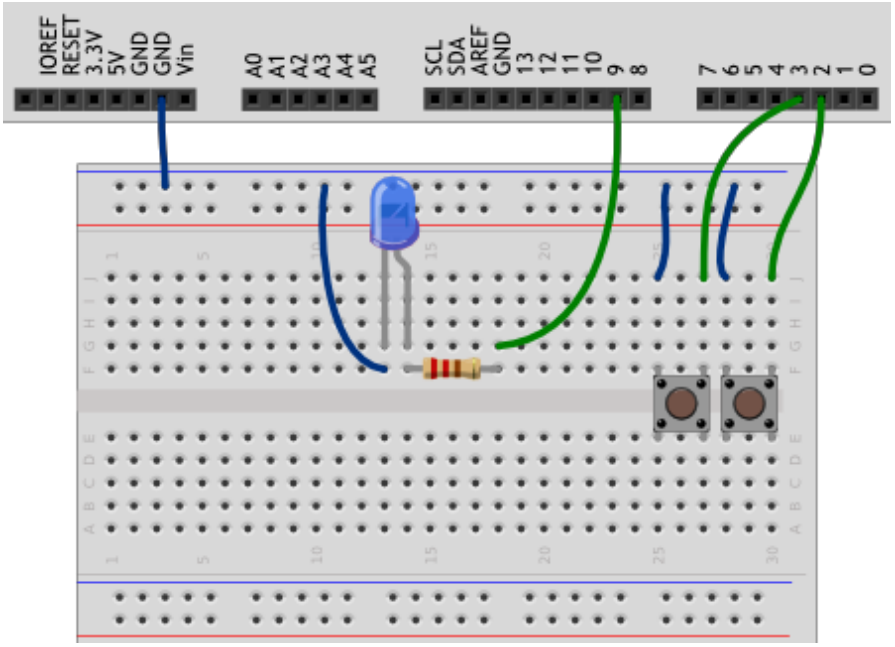


Схема на макетке



Обратите внимание

- Если вы переделываете схему из схемы предыдущего эксперимента, обратите внимание, что на этот раз нам нужно подключить светодиод к порту, поддерживающему ШИМ.

Скетч

```
#define PLUS_BUTTON_PIN     2
#define MINUS_BUTTON_PIN    3
#define LED_PIN             9

int brightness = 100;
boolean plusUp = true;
boolean minusUp = true;

void setup()
{
  pinMode(LED_PIN, OUTPUT);
  pinMode(PLUS_BUTTON_PIN, INPUT_PULLUP);
  pinMode(MINUS_BUTTON_PIN, INPUT_PULLUP);
}

void loop()
{
  analogWrite(LED_PIN, brightness);
  // реагируем на нажатия с помощью функции, написанной нами
  plusUp = handleClick(PLUS_BUTTON_PIN, plusUp, +35);
  minusUp = handleClick(MINUS_BUTTON_PIN, minusUp, -35);
}
// Собственная функция с 3 параметрами: номером пина с кнопкой
// (buttonPin), состоянием до проверки (wasUp) и градацией
// яркости при клике на кнопку (delta). Функция возвращает
// (англ. return) обратно новое, текущее состояние кнопки
boolean handleClick(int buttonPin, boolean wasUp, int delta)
{
  boolean isUp = digitalRead(buttonPin);
  if (wasUp && !isUp) {
    delay(10);
    isUp = digitalRead(buttonPin);
    // если был клик, меняем яркость в пределах от 0 до 255
    if (!isUp)
      brightness = constrain(brightness + delta, 0, 255);
  }
  return isUp; // возвращаем значение обратно, в вызывающий код
}
```

Пояснения к коду

- Мы можем пользоваться не только встроенными функциями, но и создавать собственные. Это обоснованно, когда нам нужно повторять одни и те же действия в разных местах кода или, например, нужно выполнять одни и те же действия над разными данными, как в данном случае: обработать сигнал с цифровых портов 2 и 3.
- Определять собственные функции можно в любом месте кода вне кода других функций. В нашем примере, мы определили функцию после `loop`.
- Чтобы определить собственную функцию, нам нужно:
  - Объявить, какой тип данных она будет возвращать. В нашем случае это `boolean`. Если функция только выполняет какие-то действия и не возвращает никакого значения, используйте ключевое слово `void`
  - Назначить функции имя — идентификатор. Здесь действуют те же правила, что при именовании переменных и констант. Называть функции принято в том же стиле `какПеременные`.
  - В круглых скобках перечислить передаваемые в функцию параметры, указав тип каждого. Это является объявлением переменных, видимых внутри вновь создаваемой функции, и только внутри нее. Например, если в данном эксперименте мы попробуем обратиться к `wasUp` или `isUp` из `loop()` получим от компилятора сообщение об ошибке. Точно так же, переменные, объявленные в `loop`, другим функциям не видны, но их значения можно передать в качестве параметров.
  - Между парой фигурных скобой написать код, выполняемый функцией
  - Если функция должна вернуть какое-то значение, с помощью ключевого слова `return` указать, какое значение возвращать. Это значение должно быть того типа, который мы объявили
- Так называемые глобальные переменные, т.е. переменные, к которым можно обратиться из любой функции, обычно объявляются в начале программы. В нашем случае — это `brightness`.
- Внутри созданной нами функции `handleClick` происходит всё то же самое, что в эксперименте «Кнопочный переключатель».
- Поскольку при шаге прироста яркости 35 не более чем через восемь нажатий подряд на одну из кнопок значение выражения `brightness + delta` выйдет за пределы интервала [0, 255]. С помощью функции `constrain` мы ограничиваем допустимые значения для переменной `brightness` указанными границами интервала.
- В выражении `plusUp = handleClick(PLUS_BUTTON_PIN, plusUp, +35)` мы обращаемся к переменной `plusUp` дважды. Поскольку `=` помещает значение правого операнда в левый, сначала вычисляется, что вернет `handleClick`. Поэтому когда мы передаем ей `plusUp` в качестве параметра, она имеет еще старое значение, вычисленное при прошлом вызове `handleClick`.

- Внутри `handleClick` мы вычисляем новое значение яркости светодиода и записываем его в глобальную переменную `brightness`, которая на каждой итерации `loop` просто передается в `analogWrite`.

Вопросы для проверки себя

1. Что необходимо для определения собственной функции?
2. Что означает ключевое слово `void`?
3. Как ведет себя программа при упоминании одной переменной с разных сторон от оператора присваивания `=`?

Задания для самостоятельного решения

1. Доработайте код таким образом, чтобы шаг изменения яркости настраивался в одном месте.
2. Создайте еще одну функцию и переделайте код так, чтобы одна функция отвечала за отслеживание нажатий, а другая — за вычисление яркости светодиода и возвращала его в `analogWrite`.

Эксперимент 12. Кнопочные ковбои

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 2 тактовых кнопки
- 2 резистора номиналом 220 Ом
- 2 светодиода
- 1 пьезопищалка
- 10 проводов «папа-папа»

Принципиальная схема

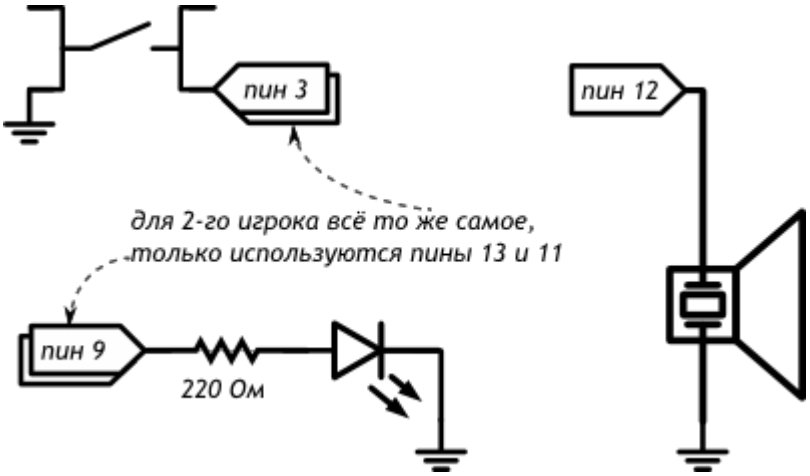
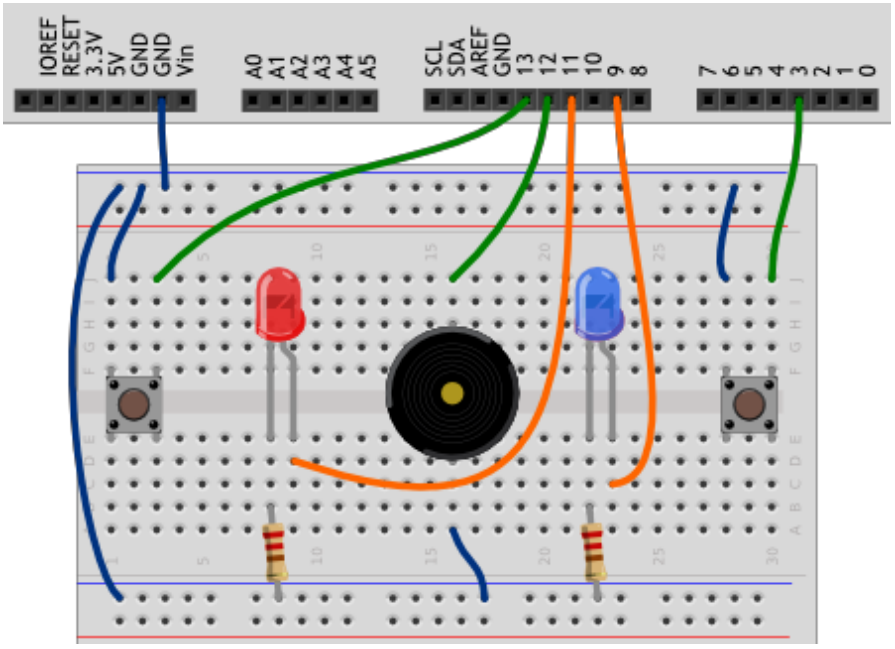


Схема на макетке



Скетч

```
#define BUZZER_PIN   12  // пин с пищалкой
#define PLAYER_COUNT 2   // количество игроков-ковбоев
// вместо перечисления всех пинов по-одному, мы объявляем пару
// списков: один с номерами пинов с кнопками, другой — со
// светодиодами. Списки также называют массивами (англ. array)
int buttonPins[PLAYER_COUNT] = {3, 13};
int ledPins[PLAYER_COUNT] = {9, 11};

void setup()
{
  pinMode(BUZZER_PIN, OUTPUT);
  for (int player = 0; player < PLAYER_COUNT; ++player) {
    // при помощи квадратных скобок получают значение в массиве
    // под указанным в них номером. Нумерация начинается с нуля
    pinMode(ledPins[player], OUTPUT);
    pinMode(buttonPins[player], INPUT_PULLUP);
  }
}

void loop()
{
  // даём сигнал «пли!», выждав случайное время от 2 до 7 сек
  delay(random(2000, 7000));
  tone(BUZZER_PIN, 3000, 250); // 3 килогерца, 250 миллисекунд

  for (int player = 0; ; player = (player+1) % PLAYER_COUNT) {
    // если игрок номер «player» нажал кнопку...
    if (!digitalRead(buttonPins[player])) {
      // ...включаем его светодиод и сигнал победы на 1 сек
      digitalWrite(ledPins[player], HIGH);
      tone(BUZZER_PIN, 4000, 1000);
      delay(1000);
      digitalWrite(ledPins[player], LOW);
      break; // Есть победитель! Выходим (англ. break) из цикла
    }
  }
}
```

Пояснения к коду

- Массив состоит из элементов одного типа, в нашем случае `int`.
- Объявить массив можно следующими способами:

```
int firstArray[6]; // 6 целых чисел с неопределёнными начальными значениями
int pwmPins[] = {3, 5, 6, 9, 10, 11}; // 6 целых чисел, длина вычисляется автоматом
```

```
boolean buttonState[3] = {false, true, false}; // можно использовать элементы любого типа
```

- Когда мы объявляем массив с указанием количества его элементов n, это число всегда на 1 больше, чем номер последнего элемента (n-1), т.к. индекс первого элемента — 0.
- Считать или записать значение элемента массива можно, обратившись к нему по индексу, например `firstArray[2]` или `buttonState[counter]`, где `counter` — переменная, такая как счетчик цикла
- В переменных типа `long` можно хранить значения до 2 147 483 647. `unsigned int` в этом случае нам будет недостаточно, потому что 65 535 миллисекунд пройдут чуть больше чем за минуту!
- Функция `random(min, max)` возвращает целое псевдослучайное число в интервале [min, max]. Для драматичности каждая игра начинается с паузы случайной длины.
- Благодаря массивам в этом эксперименте мы настраиваем порты, считываем кнопки и включаем светодиоды в циклах со счетчиком, который используется как индекс элемента.
- Мы используем цикл `fo` без условия его завершения, поэтому пока мы явно того не потребуем, цикл будет крутиться до бесконечности.
- Мы использовали выражение `player = (player+1) % PLAYER COUNT` для счётчика цикла, чтобы не только увеличивать его на единицу каждый раз, но и обнулять при достижении последнего игрока.
- Инструкция `break` прекращает работу цикла и выполнение программы продолжается с инструкции после его конца.

Вопросы для проверки себя

1. Можно ли поместить в один массив элементы типа `boolean` и `int`?
2. Обязательно ли при объявлении массива заполнять его значениями?
3. Чем удобно использование массива?
4. Как обратиться к элементу массива, чтобы прочитать его значение?
5. Почему для хранения времени прошлого сигнала мы используем переменную типа `long`?
6. Чем отличаются инструкции `continue` и `break`?

Задания для самостоятельного решения

1. Сделайте напряженный вариант игры: пусть интервал между сигналами будет в диапазоне от 10 до 15 секунд.
2. В игре есть лазейка: кнопку можно зажать до сигнала «пли!» и таким образом сразу же выиграть. Дополните программу так, чтобы так выиграть было нельзя.
3. Добавьте в игру еще двух ковбоев!

Эксперимент 13. Секундомер

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 семисегментный индикатор
- 7 резисторов номиналом 220 Ом
- 9 проводов «папа-папа»

Принципиальная схема

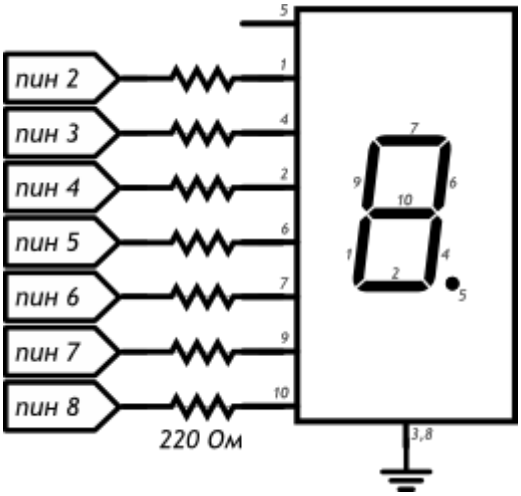
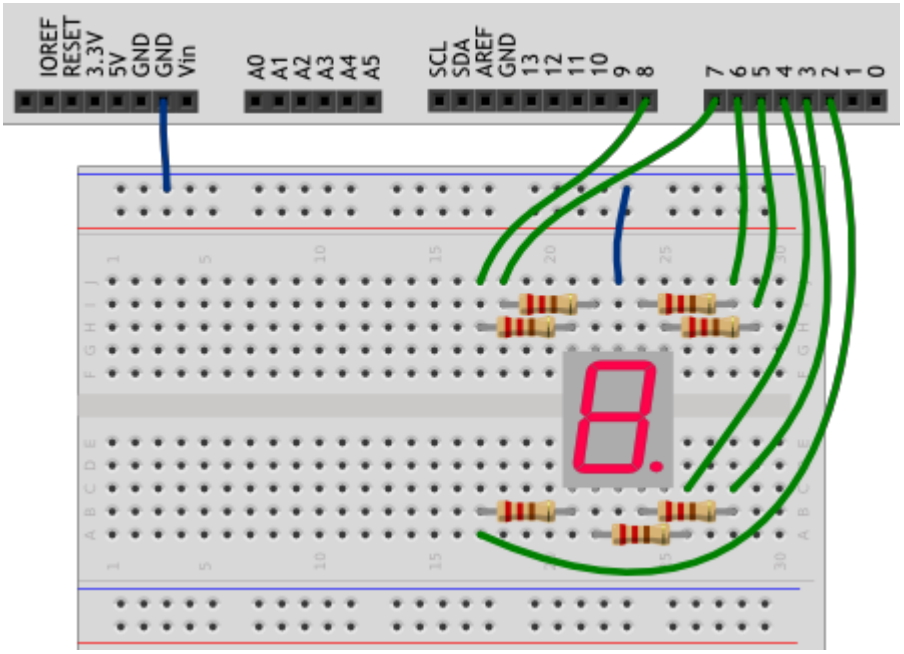


Схема на макетке



Обратите внимание

- Выводы 3 и 8 семисегментного индикатора оба являются катодами, к земле можете подключать любой из них.
- Внимательно рассмотрите схему, сопоставьте сегменты индикатора с номерами его ножек, а те, в свою очередь, с пинами Arduino, к которым мы их подключаем.
- Вывод 5 индикатора — это точка. Мы не используем её в этом эксперименте
- Сегменты индикатора — просто светодиоды, поэтому мы используем резистор с каждым из них.

Скетч

```
#define FIRST_SEGMENT_PIN   2
#define SEGMENT_COUNT       7

// префикс «0b» означает, что целое число за ним записано в
// в двоичном коде. Единицами мы обозначим номера сегментов
// индикатора, которые должны быть включены для отображения
// арабской цифры. Всего цифр 10, поэтому в массиве 10 чисел.
// Нам достаточно всего байта (англ. byte, 8 бит) для хранения
// комбинации сегментов для каждой из цифр.
byte numberSegments[10] = {
  0b00111111, 0b00001010, 0b01011101, 0b01011110, 0b01101010,
  0b01110110, 0b01110111, 0b00011010, 0b01111111, 0b01111110,
};

void setup()
{
  for (int i = 0; i < SEGMENT_COUNT; ++i)
    pinMode(i + FIRST_SEGMENT_PIN, OUTPUT);
}

void loop()
{
  // определяем число, которое собираемся отображать. Пусть им
  // будет номер текущей секунды, зацикленный на десятке
  int number = (millis() / 1000) % 10;
  // получаем код, в котором зашифрована арабская цифра
  int mask = numberSegments[number];
  // для каждого из 7 сегментов индикатора...
  for (int i = 0; i < SEGMENT_COUNT; ++i) {
    // ...определяем: должен ли он быть включён. Для этого
    // считываем бит (англ. read bit), соответствующий текущему
    // сегменту «i». Истина — он установлен (1), ложь — нет (0)
    boolean enableSegment = bitRead(mask, i);
    // включаем/выключаем сегмент на основе полученного значения
    digitalWrite(i + FIRST_SEGMENT_PIN, enableSegment);
```

```
    }
}
```

Пояснения к коду

- Мы создали массив типа `byte`: каждый его элемент это 1 байт, 8 бит, может принимать значения от 0 до 255.
- Символы арабских цифр закодированы состоянием пинов, которые соединены с выводами соответствующих сегментов: 0, если сегмент должен быть выключен, и 1, если включен.
- В переменную `mask` мы помещаем тот элемент массива `numberSegments`, который соответствует текущей секунде, вычисленной в предыдущей инструкции.
- В цикле `for` мы пробегаем по всем сегментам, извлекая с помощью встроенной функции `bitRead`нужное состояние для текущего пина, в которое его и приводим с помощью `digitalWrite` и переменной`enableSegment`
- `bitRead(x, n)` возвращает `boolean` значение: **n**-ный бит *справа* в байте **x**

Вопросы для проверки себя

1. К которой ножке нашего семисегментного индикатора нужно подключать землю?
2. Как мы храним закодированные символы цифр?
3. Каким образом мы выводим символ на индикатор?

Задания для самостоятельного решения

1. Измените код, чтобы индикатор отсчитывал десятые секунды.
2. Поменяйте программу так, чтобы вместо символа «0» отображался символ «A».
3. Дополните схему и программу таким образом, чтобы сегмент-точка включался при прохождении четных чисел и выключался на нечетных

Эксперимент 14. Счётчик нажатий

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 тактовая кнопка
- 1 выходной сдвиговый регистр 74HC595
- 1 семисегментный индикатор
- 7 резисторов номиналом 220 Ом
- 24 провода «папа-папа»

Для дополнительного задания

- 1 фоторезистор
- 1 резистор номиналом 10 кОм
- еще 1 провод

Принципиальная схема

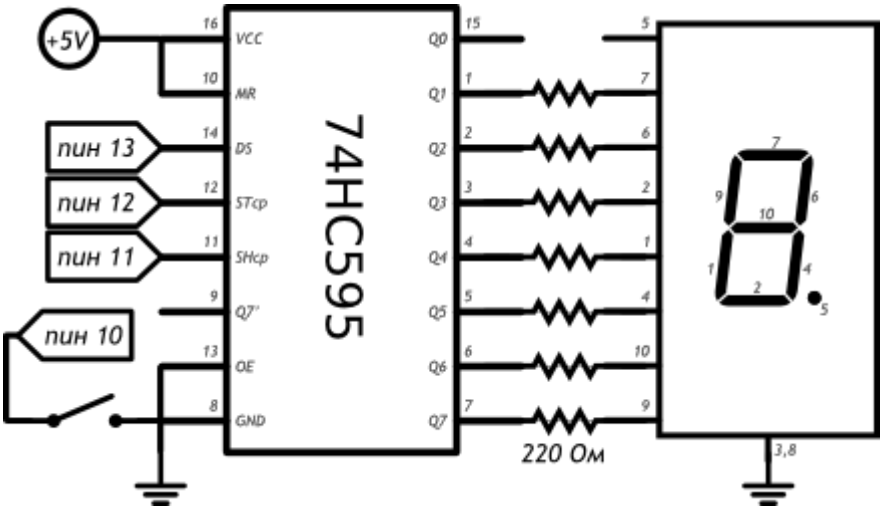
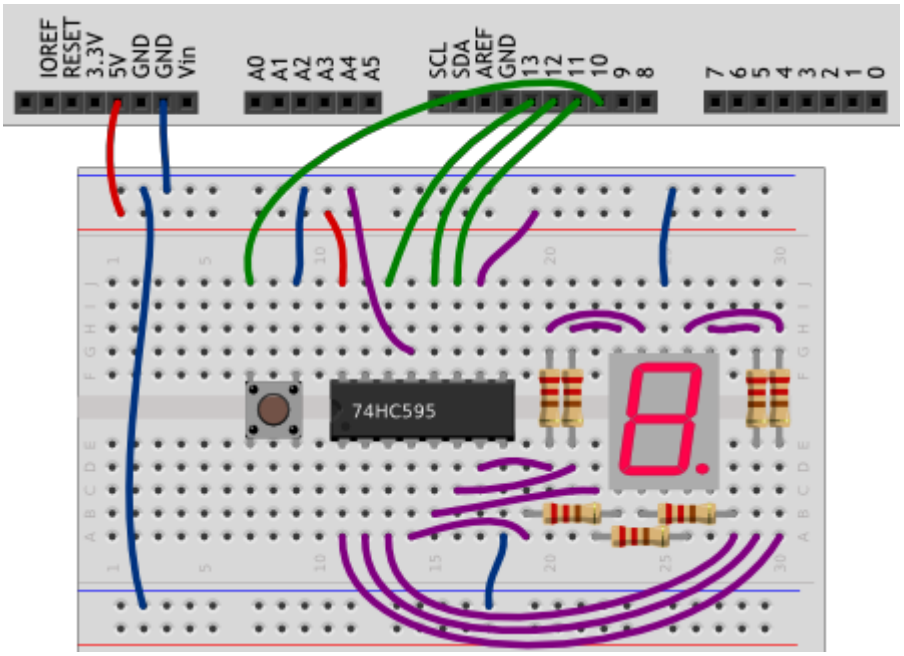


Схема на макетке



Обратите внимание

- В этом эксперименте мы впервые используем микросхему, в данном случае — выходной сдвиговый регистр 74HC595. Микросхемы полезны тем, что позволяют решать определенную задачу, не собирая каждый раз стандартную схему.
- Выходной сдвиговый регистр дает нам возможность «сэкономить» цифровые выходы, использовав всего 3 вместо 8. Каскад регистров позволил бы давать 16 и т.д. сигналов через те же три пина.
- Перед использованием микросхемы нужно внимательно изучить схему ее подключения в datasheet'e. Для того, чтобы понять, откуда считать ножки микросхемы, на них с одной стороны есть полукруглая выемка. Если мы расположим нашу 74HC595 выемкой влево, то в нижнем ряду будут ножки 1—8, а в верхнем 16—9.
- На принципиальной схеме нашего эксперимента ножки расположены в другом порядке, чтобы не вышло путаницы в соединениях. Назначения выводов согласно datasheet'у подписаны внутри изображения микросхемы, номера ножек — снаружи.
- Напомним, что на изображении семисегментного индикатора подписаны номера его ножек и их соответствие сегментам.

Скетч

```
#define DATA_PIN    13 // пин данных (англ. data)
#define LATCH_PIN   12 // пин такта (англ. clock)
#define CLOCK_PIN   11 // пин строба (англ. latch)
#define BUTTON_PIN  10

int clicks = 0;
boolean buttonWasUp = true;
```

```
byte segments[10] = {
  0b01111101, 0b00100100, 0b01111010, 0b01110110, 0b00100111,
  0b01010111, 0b01011111, 0b01100100, 0b01111111, 0b01110111
};

void setup()
{
  pinMode(DATA_PIN, OUTPUT);
  pinMode(CLOCK_PIN, OUTPUT);
  pinMode(LATCH_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
}

void loop()
{
  // считаем клики кнопки, как уже делали это раньше
  if (buttonWasUp && !digitalRead(BUTTON_PIN)) {
    delay(10);
    if (!digitalRead(BUTTON_PIN))
      clicks = (clicks + 1) % 10;
  }
  buttonWasUp = digitalRead(BUTTON_PIN);
  // для записи в 74HC595 нужно притянуть пин строба к земле
  digitalWrite(LATCH_PIN, LOW);
  // задвигаем (англ. shift out) байт-маску бит за битом,
  // начиная с младшего (англ. Least Significant Bit first)
  shiftOut(DATA_PIN, CLOCK_PIN, LSBFIRST, segments[clicks]);
  // чтобы переданный байт отразился на выходах Qx, нужно
  // подать на пин строба высокий сигнал
  digitalWrite(LATCH_PIN, HIGH);
}
```

Пояснения к коду

- Обратите внимание, что в этом эксперименте кодировки символов отличаются от кодировок из эксперимента «Секундомер».
- Для того, чтобы передать порцию данных, которые будут отправлены через сдвиговый регистр далее, нам нужно подать `LOW` на latch pin (вход ST_{cp} микросхемы), затем передать данные, а затем отправить `HIGH` на latch pin, после чего на соответствующих выходах 74HC595 появится переданная комбинация высоких и низких уровней сигнала.
- Для передачи данных мы использовали функцию `shiftOut(dataPin, clockPin, bitOrder, value)`. Функция ничего не возвращает, а в качестве параметров ей нужно сообщить
  - пин Arduino, который подключен ко входу DS микросхемы (data pin),
  - пин Arduino, соединенный со входом SH_{cp} (clock pin),

- порядок записи битов: `LSBFIRST` (least significant bit first) — начиная с младшего, или `MSBFIRST`(most significant bit first) — начиная со старшего,
- байт данных, который нужно передать. Функция работает с порциями данных в один байт, так что если вам нужно передать больше, придется вызывать ее несколько раз.

Вопросы для проверки себя

1. Для чего нужны микросхемы? Для чего нужен выходной сдвиговый регистр?
2. Как найти ножку микросхемы, на которую отправляются данные?
3. Что нужно сделать до и после отправки собственно данных на 74HC595?
4. Сколько данных можно передать с помощью `shiftOut()` и как управлять порядком их передачи?

Задания для самостоятельного решения

1. Заставьте `shiftOut()` отправлять биты, начиная со старшего, и измените код так, чтобы счетчик по-прежнему показывал арабские цифры.
2. Замените кнопку на датчик света (фоторезистор в схеме делителя напряжения) и переделайте программу так, чтобы индикатор цифрой показывал уровень освещенности.

Эксперимент 15. Комнатный термометр

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 светодиодная шкала
- 1 резистор номиналом 10 кОм
- 1 термистор
- 10 резисторов номиналом 220 Ом
- 14 проводов «папа-папа»

Для дополнительного задания

- 1 пьезопищалка
- еще 2 провода

Принципиальная схема

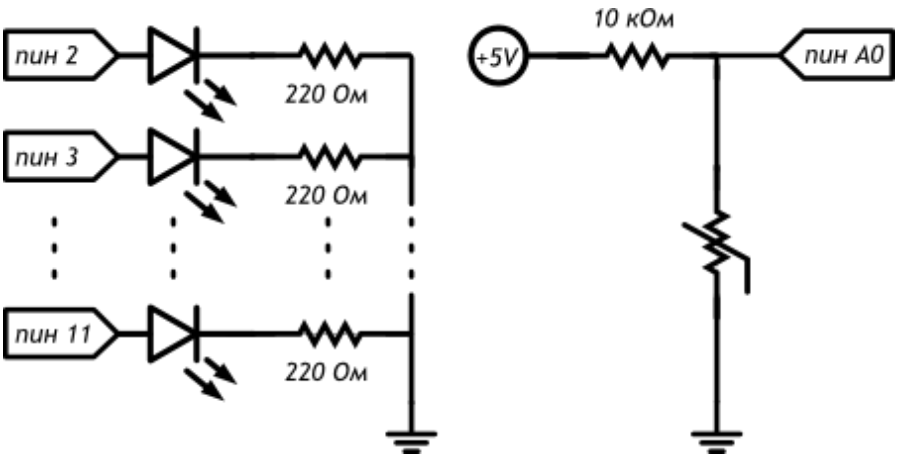
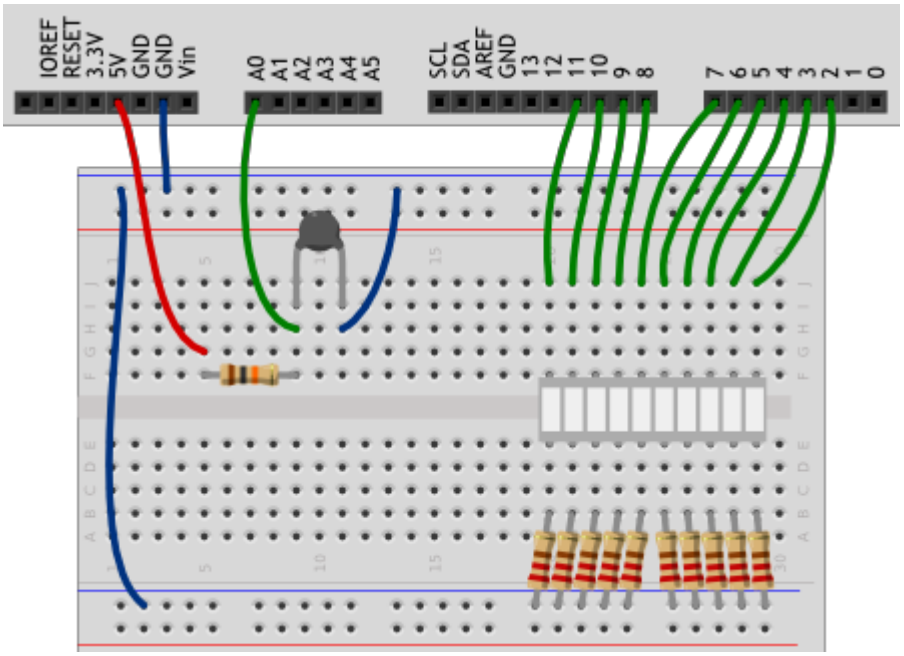


Схема на макетке



Обратите внимание

- Термистор мы включили в известную нам схему делителя напряжения.

Скетч

```
// Огромное количество готового кода уже написано другими людьми
// и хранится в виде отдельных файлов, которые называются
// библиотеками. Для использования кода из библиотеки, её нужно
// подключить (англ. include). Библиотека «math» даёт разные
// математические функции, в том числе функцию логарифма
// (англ. log), которая нам понадобится далее
#include <math.h>

#define FIRST_LED_PIN 2
#define LED_COUNT     10

// Параметр конкретного типа термистора (из datasheet):
#define TERMIST_B 4300

#define VIN 5.0

void setup()
{
  for (int i = 0; i < LED_COUNT; ++i)
    pinMode(i + FIRST_LED_PIN, OUTPUT);
}

void loop()
{
```

```
  // вычисляем температуру в °C с помощью магической формулы.
  // Используем при этом не целые числа, а вещественные. Их ещё
  // называют числами с плавающей (англ. float) точкой. В
  // выражениях с вещественными числами обязательно нужно явно
  // указывать дробную часть у всех констант. Иначе дробная
  // часть результата будет отброшена

   float voltage = analogRead(A0) * VIN / 1023.0;
   float r1 = voltage / (VIN - voltage);


   float temperature = 1./( 1./(TERMIST_B)*log(r1)+1./(25. + 273.) ) - 273;

  for (int i = 0; i < LED_COUNT; ++i) {
    // при 21°C должен гореть один сегмент, при 22°C — два и
    // т.д. Определяем должен ли гореть i-й нехитрым способом
    boolean enableSegment = (temperature >= 21+i);
    digitalWrite(i + FIRST_LED_PIN, enableSegment);
  }
}
```

Пояснения к коду

- Директивы для подключения библиотек `#include` включаются в начало программы.
- В этом эксперименте мы подключаем библиотеку `math.h` для того, чтобы использовать функцию взятия натурального логарифма **x** `log(x)`.
- В переменных типа `floa` можно хранить дробные числа, числа с плавающей точкой.
- При использовании переменных данного типа имейте в виду:
  - при операциях с их использованием, указывайте нулевую дробную часть у целых констант, как в примере
  - они могут принимать значения от -3.4028235×10^{38} до 3.4028235×10^{38},
  - при этом количество значащих цифр может быть 6-7: всех цифр, не только после запятой!
  - точность вычислений с такими данными невелика, у вас могут возникнуть неожиданные ошибки, например, при использовании `float` в условном операторе. Не полагайтесь на точность!
  - вычисления с `float` происходят медленнее, чем с целыми числами
- Показания термистора связаны с температурой нелинейно, поэтому нам приходится использовать такую громоздкую формулу.

Вопросы для проверки себя

1. Как нужно подключить термистор, чтобы получать на Arduino данные о температуре?

2. Каким образом можно воспользоваться ранее разработанными функциями, не переписывая их в программный код?
3. Чем неудобно использование чисел с плавающей точкой на Arduino?
4. Что за выражение стоит справа от `=` при объявлении булевой переменной `enableSegment`?

Задания для самостоятельного решения

1. Измените код программы таким образом, чтобы индикатор включался при 0 градусов и его показания прирастали на одно деление каждые 5 градусов.
2. Добавьте в схему пьезопищалку и доработайте программу так, чтобы срабатывала звуковая сигнализация при достижении температуры, например, 25 градусов.

Эксперимент 16. Метеостанция

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 резистор номиналом 10 кОм
- 1 термистор
- 3 провода «папа-папа»

Принципиальная схема

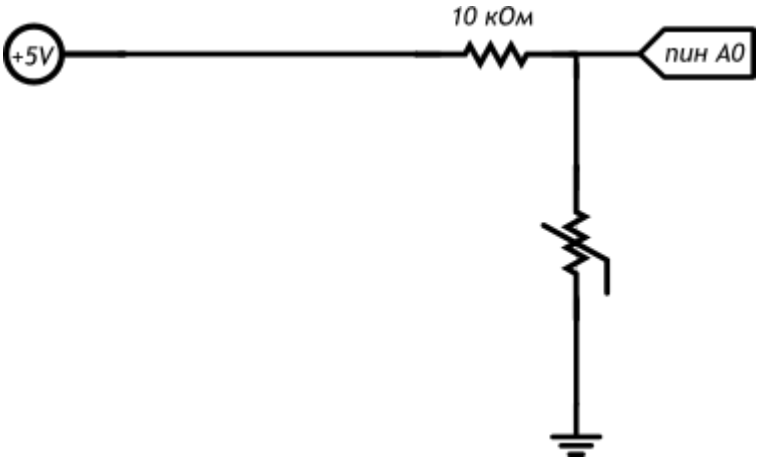
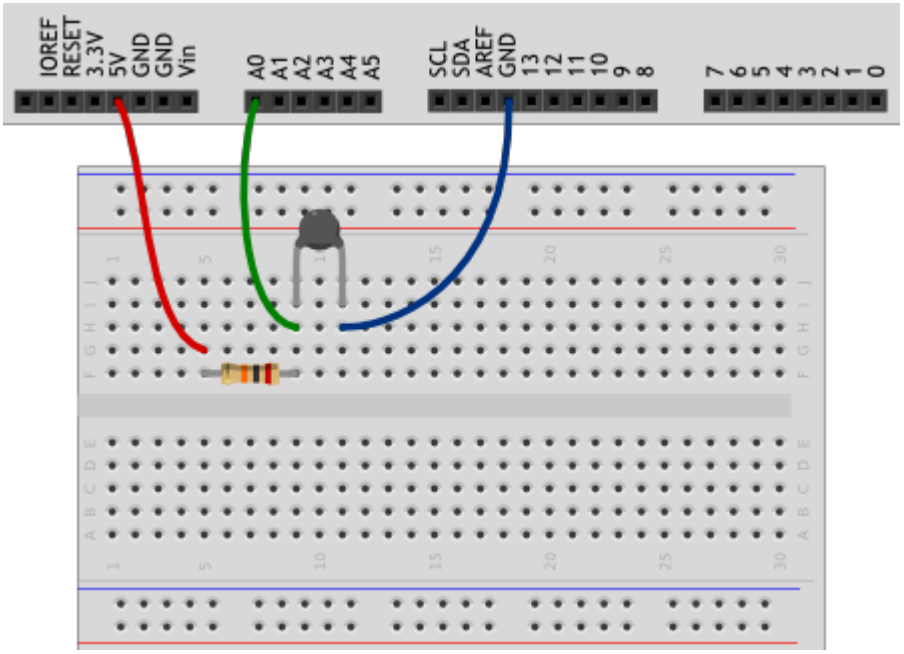


Схема на макетке



Скетч

```
#include <math.h>
```

```
int minute = 1;

// Параметр конкретного типа термистора (из datasheet):
#define TERMIST_B 4300

#define VIN 5.0

void setup()
{
  // мы хотим передавать информацию на компьютер через USB, а
  // точнее через последовательный (англ. serial) порт.
  // Для этого необходимо начать (англ. begin) передачу, указав
  // скорость. 9600 бит в секунду — традиционная скорость.
  // Функция «begin» не является глобальной, она принадлежит
  // объекту с именем «Serial». Объекты — это «продвинутые»
  // переменные, которые обладают собственными функциями,
  // к которым обращаются через символ точки.
  Serial.begin(9600);
  // передаём заголовок нашей таблицы в текстовом виде, иначе
  // говоря печатаем строку (англ. print line). Символы «\t» —
  // это специальная последовательность, которая заменяется на
  // знак табуляции (англ. tab): 8-кратный выровненный пробел
  Serial.println("Minute\tTemperature");
}

void loop()
{
  // вычисляем температуру в °C с помощью магической формулы.
  // Используем при этом не целые числа, а вещественные. Их ещё
  // называют числами с плавающей (англ. float) точкой. В
  // выражениях с вещественными числами обязательно нужно явно
  // указывать дробную часть у всех констант. Иначе дробная
  // часть результата будет отброшена

   float voltage = analogRead(A0) * VIN / 1024.0;
   float r1 = voltage / (VIN - voltage);


   float temperature = 1./( 1./(TERMIST_B)*log(r1)+1./(25. + 273.) ) - 273;
  // печатаем текущую минуту и температуру, разделяя их табом.
  // println переводит курсор на новую строку, а print — нет
  Serial.print(minute);
  Serial.print("\t");
  Serial.println(temperature);

  delay(60000); // засыпаем на минуту
  ++minute;     // увеличиваем значение минуты на 1
```

```
  // откройте окно Serial Monitor в среде Arduino, оставьте на
  // сутки, скопируйте данные в Excel, чтобы построить графики
}
```

Пояснения к коду

- Очень часто бывает полезно обмениваться данными, например, с компьютером. В частности, для отладки работы устройства: можно, например, смотреть, какие значения принимают переменные.
- В данном эксперименте мы знакомимся со стандартным объектом `Serial`, который предназначен для работы с последовательным портом (UART) Arduino, и его методами (функциями, созданными для работы с данным объектом) `begin()`, `print()` и `println()`, которые вызываются после точки, идущей за именем объекта:
  - чтобы обмениваться данными, нужно начать соединение, поэтому `Serial.begin(baudrate)` вызывается в `setup()`
  - `Serial.print(data)` отправляет содержимое `data`. Если мы хотим отправить текст, можно просто заключить его в пару двойных кавычек: `""`. Кириллица, скорее всего, будет отображаться некорректно.
  - `Serial.println(data)` делает то же самое, только добавляет в конце невидимый символ новой строки.
- В `print()` и `println()` можно использовать второй необязательный параметр: выбор системы счисления, в которой выводить число (это может быть `DEC`, `BIN`, `HEX`, `OCT` для десятичной, двоичной, шестнадцатеричной и восьмеричной систем счисления соответственно) или количество знаков после запятой для дробных чисел.
  Например,

```
Serial.println(18,BIN);
Serial.print(3.14159,3);
```

  в мониторе порта даст результат

```
10010
3.142
```

- Монитор порта, входящий в Arduino IDE, открывается через меню Сервис или сочетанием клавиш Ctrl+Shift+M. Следите за тем, чтобы в мониторе и в скетче была указана одинаковая скорость обмена данными, `baudrate`. Скорости 9600 бит в секунду обычно достаточно. Другие стандартные значения можете посмотреть в выпадающем меню справа внизу окна монитора порта.

- Вам не удастся использовать цифровые порты 0 и 1 одновременно с передачей данных по последовательному порту, потому что по ним также идет передача данных, как и через USB-порт платы.
- При запуске монитора порта скетч в микроконтроллере перезагружается и начинает работать с начала. Это удобно, если вам нельзя упустить какие-то данные, которые начинаю передаваться сразу же. Но в других ситуациях это может мешать, помните об этом нюансе!
- Если вы хотите читать какие-то данные в реальном времени, не забывайте делать `delay()` хотя бы на 100 миллисекунд, иначе бегущие числа в мониторе будет невозможно разобрать. Вы можете отправлять данные и без задержки, а затем, к примеру, скопировать их для обработки в стороннем приложении.
- Последовательность `\t` выводится как символ табуляции (8 пробелов с выравниванием). Также вы можете использовать, например, последовательность `\n` для перевода строки. Если вы хотите использовать обратный слеш, его нужно экранировать вторым таким же: `\\`.

Вопросы для проверки себя

1. Какие действия нужно предпринять, чтобы читать на компьютере данные с Arduino?
2. О каких ограничениях не следует забывать при работе с последовательным портом?
3. Как избежать ошибки в передаче данных, содержащих обратный слэш (`\`)?

Задания для самостоятельного решения

1. Перед таблицей данных о температуре добавьте заголовок (например, "Meteostation").
2. Добавьте столбец, содержащий количество секунд, прошедших с момента запуска микроконтроллера. Можно уменьшить интервал передачи данных.

Эксперимент 17. Пантограф

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 сервопривод
- 1 конденсатор емкостью 220 мкФ
- 1 потенциометр
- 11 проводов «папа-папа»

Принципиальная схема

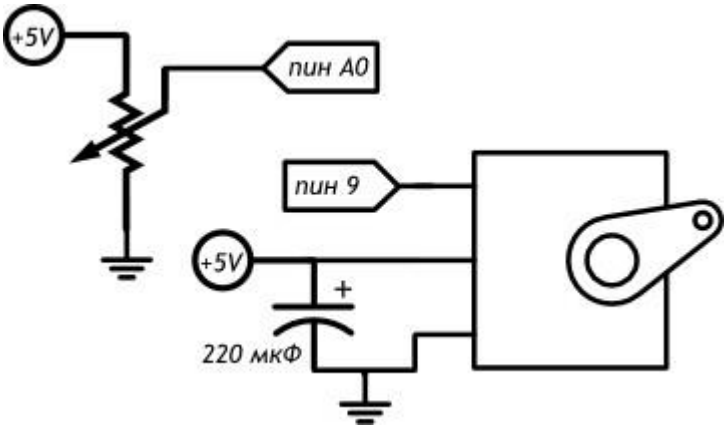
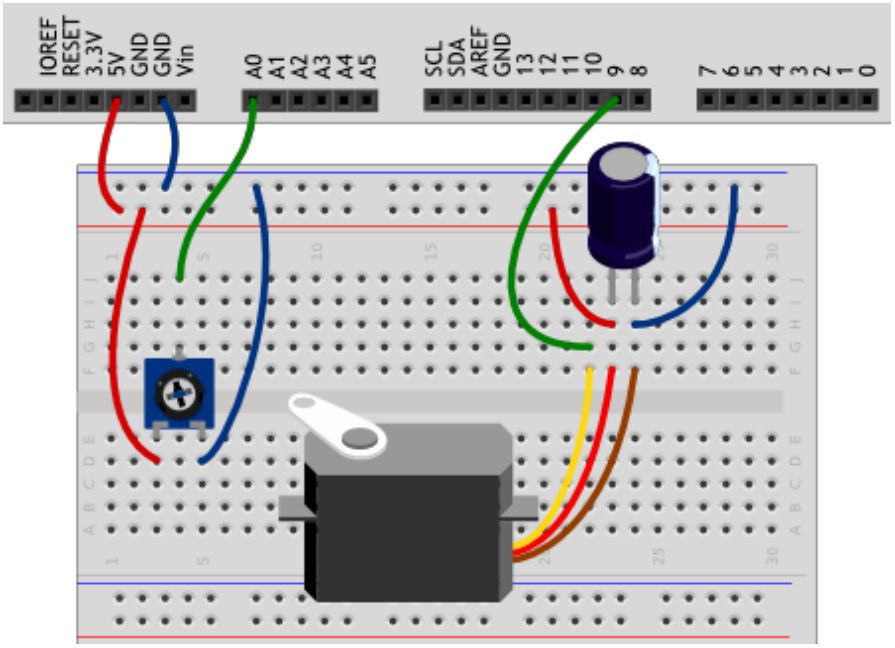


Схема на макетке



Обратите внимание

- Конденсатор в данной схеме нам нужен для того, чтобы при включении сервопривода избежать просадки питания платы.
- Не забывайте про то, что нужно соблюдать полярность элетролитического конденсатора. Короткая ножка (со стороны белой полосы на корпусе) — «минус».
- Вы можете соединить провод сервопривода с макетной платой проводами «папа-папа»: коричневый это земля, красный — питание, оранжевый — сигнал.
- В данном эксперименте мы подключаем питние сервопривода к 5V-выходу Arduino. С одним сервоприводом плата справится, но если в каком-либо проекте вам нужно больше серв, используйте специальные платы-драйвера с отдельным источником питания для серв.

Скетч

```
// управлять сервоприводами (англ. servo motor) самостоятельно
// не так то просто, но в стандартной библиотеке уже всё
// заготовлено, что делает задачу тривиальной
#include <Servo.h>

#define POT_MAX_ANGLE 270.0 // макс. угол поворота потенциометра

// объявляем объект типа Servo с именем myServo. Ранее мы
// использовали int, boolean, float, а теперь точно также
// используем тип Servo, предоставляемый библиотекой. В случае
// Serial мы использовали объект сразу же: он уже был создан
// для нас, но в случае с Servo, мы должны сделать это явно.
// Ведь в нашем проекте могут быть одновременно несколько
// приводов, и нам понадобится различать их по именам
Servo myServo;

void setup()
{
  // прикрепляем (англ. attach) нашу серву к 9-му пину. Явный
  // вызов pinMode не нужен: функция attach сделает всё за нас
  myServo.attach(9);
}

void loop()
{
  int val = analogRead(A0);
  // на основе сигнала понимаем реальный угол поворота движка.
  // Используем вещественные числа в расчётах, но полученный
  // результат округляем обратно до целого числа
  int angle = int(val / 1024.0 * POT_MAX_ANGLE);
  // обычная серва не сможет повторить угол потенциометра на
  // всём диапазоне углов. Она умеет вставать в углы от 0° до
  // 180°. Ограничиваем угол соответствующе
```

```
  angle = constrain(angle, 0, 180);
  // и, наконец, подаём серве команду встать в указанный угол
  myServo.write(angle);
}
```

Пояснения к коду

- В данном эксперименте мы также имеем дело с объектом, на этот раз он нужен для простого управления сервоприводом. Как отмечено в комментариях, в отличие от объекта `Serial`, объекты типа `Servo` нам нужно явно создать: `Servo myServo`, предварительно подключив библиотеку `<Servo.h>`.
- Далее мы используем два метода для работы с ним:
  - `myServo.attach(pin)` — сначала «подключаем» серву к порту, с которым физически соединен его сигнальный провод. `pinMode()` не нужна, метод `attach()` займется этим.
  - `myServo.write(angle)` — задаем угол, т.е. позицию, которую должен принять вал сервопривода. Обычно это 0—180°.
- `myServo` здесь это имя объекта, идентификатор, который мы придумываем так же, как названия переменных. Например, если вы хотите управлять двумя захватами, у вас могут быть объекты `leftGrip`и `rightGrip`.
- Мы использовали функцию `int()` для явного преобразования числа с плавающей точкой в целочисленное значение. Она принимает в качестве параметра значение любого типа, а возвращает целое число. Когда в одном выражении мы имеем дело с различными типами данных, нужно позаботиться о том, чтобы не получить непредсказуемый ошибочный результат.

Вопросы для проверки себя

1. Зачем нужен конденсатор при включении в схему сервопривода?
2. Каким образом библиотека `<Servo.h>` позволяет нам работать с сервоприводом?
3. Зачем мы ограничиваем область допустимых значений для `angle`?
4. Как быть уверенным в том, что в переменную типа `int` после вычислений попадет корректное значение?

Задания для самостоятельного решения

1. Измените программу так, чтобы по мере поворота ручки потенциометра, сервопривод последовательно занимал 8 положений: 45, 135, 87, 0, 65, 90, 180, 150°.
2. Предположим, что сервопривод управляет шторкой, и нам нужно поддерживать постоянное количество света в помещении. Создайте такой механизм.

Эксперимент 18. Тестер батареек

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 2 резистора номиналом 10 кОм
- 1 выпрямительный диод
- 1 текстовый экран
- 16 проводов «папа-папа»
- 1 клеммник

Принципиальная схема

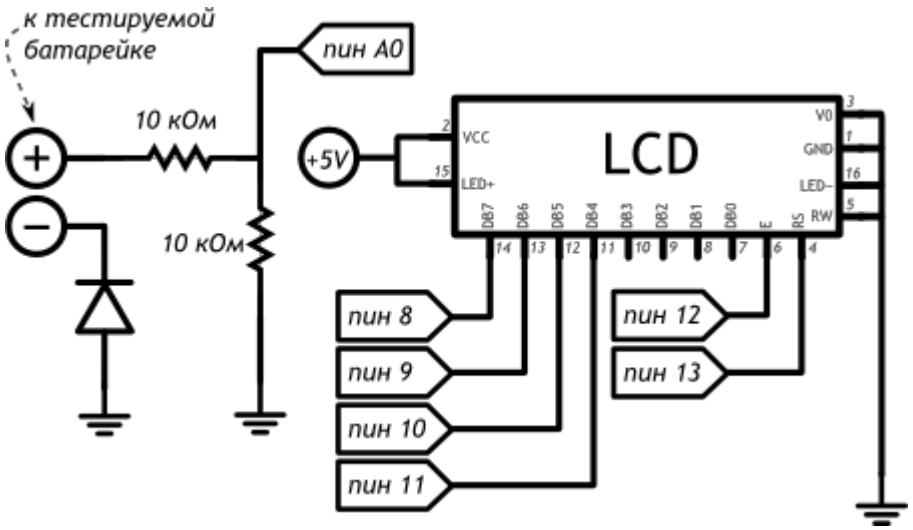
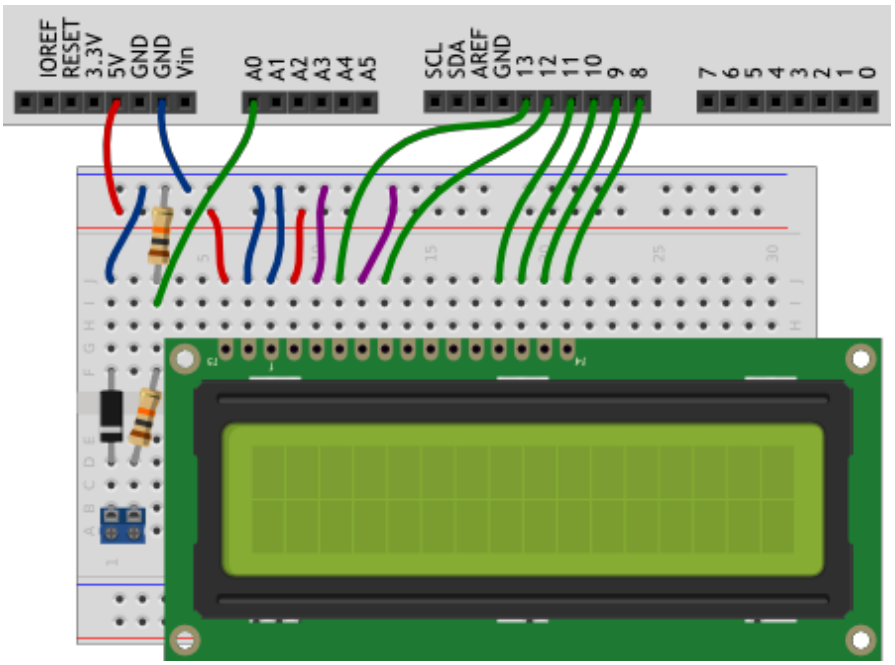


Схема на макетке



Обратите внимание

- Мы подключаем «плюс» батарейки через делитель напряжения с равными плечами (R1 = R2 = 10 кОм), таким образом деля подаваемое напряжение пополам. Поскольку в аналоговый вход Arduino мы можем подавать до 5В, мы можем измерять напряжение до 10В. Не пробуйте измерять большее напряжение, вы можете повредить плату!
- На принципиальной схеме внутри изображения дисплея подписаны названия его выводов согласно datasheet, а снаружи — номера его ножек.
- Ножки нашего ЖК-дисплея нумеруются не подряд: 15 и 16 ножки находятся перед 1.
- Диод пригодится, если пользователь тестера перепутает «+» и «-» батарейки, главное нам самим не забыть про направление, в котором через диод может течь ток, и установить его верно!

Скетч

```
// Подключаем библиотеку для работы с жидкокристаллическим
// экраном (англ. Liquid Crystal Display или просто LCD)
#include <LiquidCrystal.h>
// на диоде, защищающем от неверной полярности, падает доля
// напряжения (англ. voltage drop). Необходимо это учитывать
#define DIODE_DROP  0.7
// Объявляем объект, для управления дисплеем. Для его создания
// необходимо указать номера пинов, к которым он подключен в
// порядке:        RS   E DB5 DB6 DB7 DB8
LiquidCrystal lcd(13, 12, 11, 10,  9,  8);

void setup()
{
  // начинаем работу с экраном. Сообщаем объекту количество
  // строк и столбцов. Опять же, вызывать pinMode не требуется:
  // функция begin сделает всё за нас
  lcd.begin(16, 2);
  // печатаем сообщение на первой строке
  lcd.print("Battery voltage:");
}

void loop()
{
  // высчитываем напряжение подключенной батарейки
  float voltage = analogRead(A0) / 1024.0 * 10.0;
  // если напряжение на делителе напряжения было зафиксировано,
  // нужно прибавить напряжение на диоде, т.к. оно было съедено
  if (voltage > 0.1)
    voltage += DIODE_DROP;
  // устанавливаем курсор, колонку 0, строку 1. На деле — это
  // левый квадрат 2-й строки, т.к. нумерация начинается с нуля
  lcd.setCursor(0, 1);
```

```
  // печатаем напряжение в батарейке с точностью до сотых долей
  lcd.print(voltage, 2);
  // следом печатаем единицы измерения
  lcd.print(" Volts");
}
```

Пояснения к коду

- Если вы используете диод, падение напряжения на котором происходит на другую величину, не забудьте исправить макроопределение `DIODE_DROP`.
- В этом эксперименте мы снова пользуемся готовой библиотекой `<LiquidCrystal.h>` для создания объекта `lcd` и использования его методов
  - `lcd.begin(cols,rows)` с помощью которого мы задаем количество колонок и строк нашего дисплея
  - `lcd.print(data)` для вывода данных. У него есть второй необязательный параметр `BASE`, передав который, можно выбрать систему счисления, так же, как в примере с `Serial.print()`.
  - `lcd.setCursor(col,row)` устанавливает курсор в переданную колонку и строку. Последующий вывод будет осуществляться с этого места.
- При создании `lcd` мы передали параметрами пины, к которым подключены выводы дисплея, через которые мы будем им управлять и передавать данные.
- О том, как выводить текст кириллицей, и о других подробностях работы с дисплеем в нашей вики есть отдельная статья.

Вопросы для проверки себя

1. Из-за чего измерения напряжения в этом эксперименте могут быть неточными (на что мы можем повлиять)?
2. Какая библиотека облегчает работу с нашим текстовым экраном? Какие шаги нужно предпринять до начала вывода текста на него?
3. Каким образом мы задаем позицию, с которой на экран выводится текст?
4. Можем ли мы писать на экране кириллицей? Как?

Задания для самостоятельного решения

Возможно, вы захотите воспользоваться еще одним методом вашего объекта `lcd` — `clear()`: он очищает экран и устанавливает курсор в левую колонку верхней строчки.

1. Создайте секундомер, который будет отсчитывать время, прошедшее с начала работы Arduino и выводить секунды и сотые секунд на экран.
2. Совместите отсчет времени и измерение напряжения. Отобразите все данные на дисплее. Отправляйте их раз в 10 секунд на компьютер.

Теперь вы можете выводить без компьютера и проводов любые данные, с которыми работаете, и использовать это как в режиме эксплуатации вашего устройства, так и во время отладки!

Эксперимент 19. Светильник, управляемый по USB

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 светодиод
- 1 резистор номиналом 220 Ом
- 2 провода «папа-папа»

Принципиальная схема

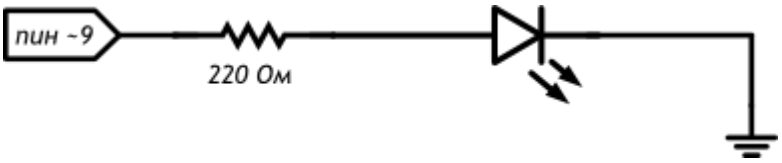
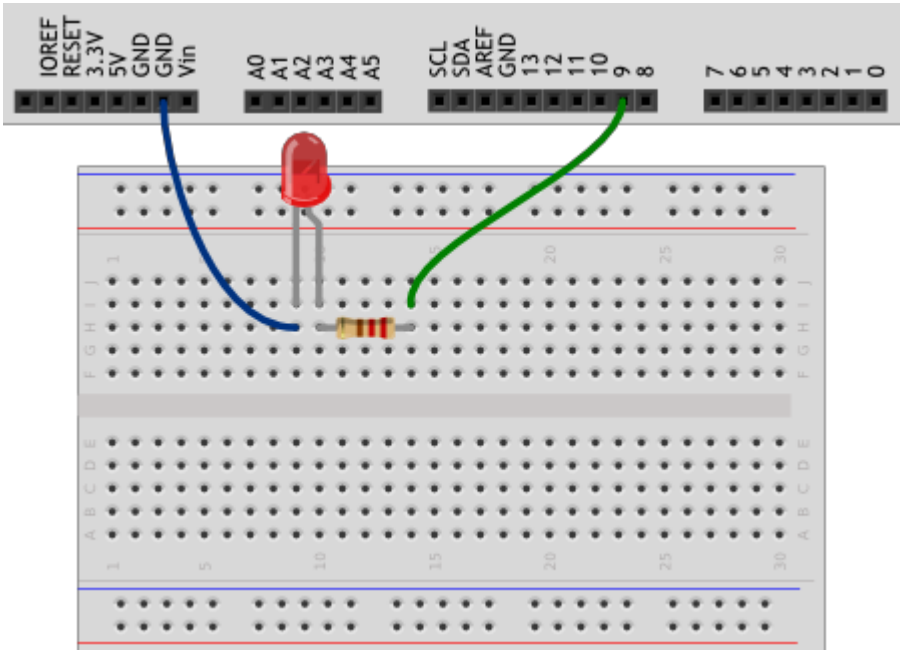


Схема на макетке



Скетч

```
#define LED_PIN 9
// для работы с текстом существуют объекты-строки (англ. string)
String message;

void setup()
{
```

```
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  // передаваемые с компьютера данные поставляются байт за
  // байтом, в виде отдельных символов (англ. character). Нам
  // нужно последовательно их обрабатывать пока (англ. while)
  // в порту доступны (англ. available) новые данные
  while (Serial.available()) {
    // считываем (англ. read) пришедший символ в переменную
    char incomingChar = Serial.read();
    // не стоит путать целые числа и символы. Они соотносятся
    // друг с другом по таблице, называемой кодировкой. Например
    // '0' — это 48, '9' — 57, 'A' — 65, 'B' — 66 и т.п. Символы
    // в программе записываются в одинарных кавычках
    if (incomingChar >= '0' && incomingChar <= '9') {
      // если пришёл символ-цифра, добавляем его к сообщению
      message += incomingChar;
    } else if (incomingChar == '\n') {
      // если пришёл символ новой строки, т.е. enter, переводим
      // накопленное сообщение в целое число (англ. to integer).
      // Так последовательность символов '1', '2', '3' станет
      // числом 123. Результат выводим на светодиод
      analogWrite(LED_PIN, message.toInt());
      // обнуляем накопленное сообщение, чтобы начать всё заново
      message = "";
    }
  }
  // посылайте сообщения-числа с компьютера через Serial Monitor
}
```

Пояснения к коду

- В этой программе мы создаем объект класса `String`. Это встроенный класс, предназначенный для работы со строками, т.е. с текстом.
- Не путайте его с типом данных `string`, который является просто массивом символов. `String` же позволяет использовать ряд методов для удобной работы со строками.
- Мы знакомимся с новым видом циклов: цикл с условием `while`. В отличие от цикла со счетчиком `for`, цикл `while(expression)` выполняется до тех пор, пока логическое выражение `expression` истинно.
- Метод `available()` объекта `Serial` возвращает количество байт, полученных через последовательный порт.

- В данном эксперименте цикл `while` работает до тех пор, пока `available()` возвращает ненулевое значение, любое из которых приводится к `true`.
- Переменные типа `char` могут хранить один символ.
- В этом примере символ мы получаем методом `Serial.read()`, который возвращает первый байт, пришедший на последовательный порт, или -1, если ничего не пришло.
- Обратите внимание, что в `if` мы сравниваем не пришедший символ с 0 и 9, но их коды. Если пришел какой-то символ, который не является цифрой, мы не будем его добавлять к нашей строке `message`.
- Объекты типа `String` позволяют производить конкатенацию, т.е. объединение строк. Это можно сделать так: `message = message + incomingChar`, но можно записать в сокращенной форме: `message += incomingChar`.
- В этой программе мы дополняем `if` конструкцией `else if`. Это еще один условный оператор, который проверяется только в случае ложности выражения, данного первому оператору. Несколько `else if`могут следовать друг за другом, при этом каждое следующее условие будет проверяться только в случае невыполнения всех предыдущих. Если в конце разместить `else`, он выполнится только если ни одно из условий не выполнено.
- Напомним, что последовательностью `\n` кодируется символ переноса строки. Если он был передан устройству, мы передаем полученные ранее символы как параметр для `analogWrite()`, которая включает светодиод.
- Мы используем один из методов `String`, `toInt()`, который заставляет считать строку не набором цифр, но числом. Он возвращает значение типа `long`, при этом, если строка начинается с символа, не являющегося цифрой, будет возвращен 0. Если после цифр, идущих в начале строки, будут символы не-цифры, на них конверсия остновится.
- Обратите внимание на выпадающее меню внизу монитора порта: чтобы наше устройство получало символ перевода строки, там должно быть выбрано «Новая строка (NL)»
- Пустая строка обозначается так: `""`. Опустошив ее, мы готовы собирать новую последовательность символов.

Вопросы для проверки себя

1. Какие объекты позволяют легко манипулировать текстовыми данными?
2. Что возвращают методы `Serial.available()` и `Serial.read()`?
3. Чем отличаются конструкции `for` и `while`?
4. Каким образом можно организовать более сложное ветвление, чем `if … else`?
5. Как можно объединить текстовые строки?
6. Как можно привести текстовую строку, содержащую цифры, к числовому типу?

Задания для самостоятельного решения

1. Проверьте, попадает ли переданное число в диапазон значений, которые нужно передавать в`analogWrite()`. Передайте на компьютер сообщение об ошибке, если нет.

2. Переделайте программу так, чтобы устройство распознавало текстовые команды, например, «on» и «off», и соответственно включало и выключало светодиод.
   Вам может пригодиться один из
   методов `String`: `toLowerCase(yourString)` или`toUpperCase(yourString)`, которые возвращают переданную строку `yourString`, приведенную к нижнему или верхнему регистру соответственно.

Эксперимент 20. Перетягивание каната

Список деталей для эксперимента

- 1 плата Arduino Uno
- 1 беспаечная макетная плата
- 1 светодиодная шкала
- 10 резисторов номиналом 220 Ом
- 4 резисторов номиналом 100 кОм
- 2 тактовых кнопки
- 2 керамических конденсатора номиналом 100 нФ
- 1 пьезопищалка
- 1 инвертирующий триггер Шмитта
- 24 провода «папа-папа»

Для дополнительного задания

- 1 сервопривод
- 1 конденсатор 220 мкФ

Принципиальная схема

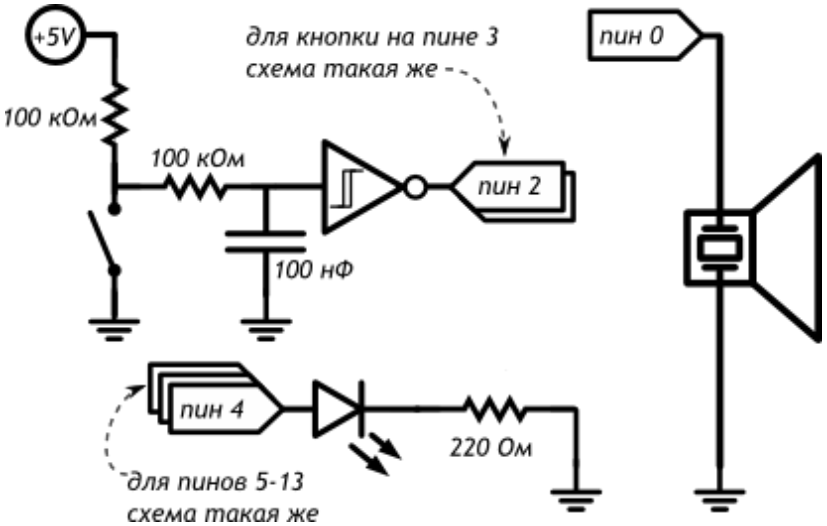
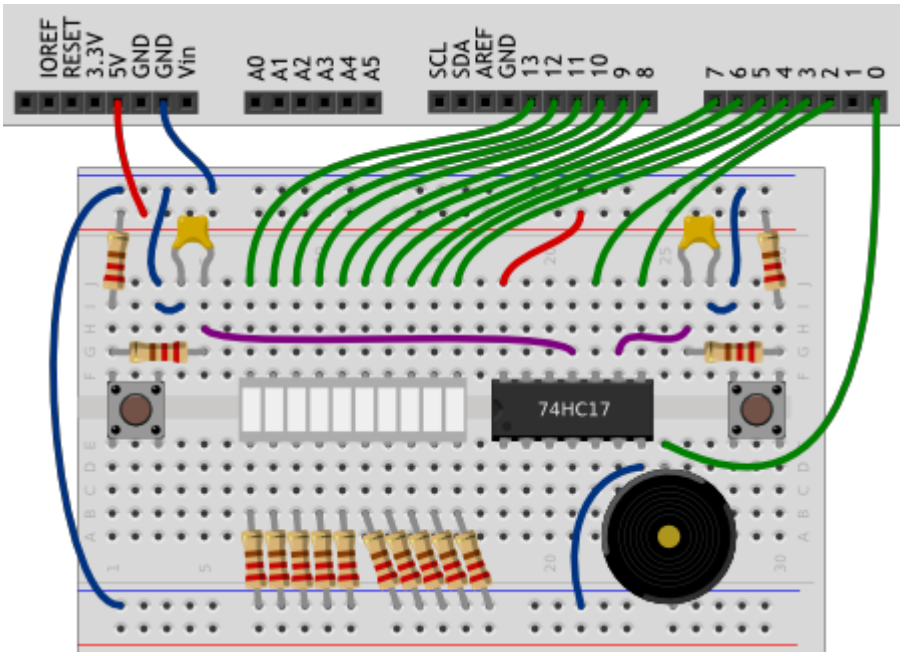


Схема на макетке



Обратите внимание

- Схема подключения кнопок с использованием конденсаторов, резисторов и микросхемы 74HC17, которая называется инвертирующий триггер Шмитта, нужна для аппаратного подавления дребезга. Посмотритевидеоурок на эту тему.
- В этом эксперименте нам нужно очень много цифровых портов, поэтому нам пришлось использовать порт 0. Пользоваться им неудобно из-за того, что он соединен с одним из каналов последовательного порта, поэтому перед прошивкой микроконтроллера нам придется отключать провод, идущий к пьезопищалке, а после прошивки подключать его обратно.

Скетч

```
#define BUZZER_PIN     0
#define FIRST_BAR_PIN  4
#define BAR_COUNT      10
#define MAX_SCORE      20
// глобальные переменные, используемые в прерываниях (см. далее)
// должны быть отмечены как нестабильные (англ. volatile)
volatile int score = 0;

void setup()
{
  for (int i = 0; i < BAR_COUNT; ++i)
    pinMode(i + FIRST_BAR_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);
  // Прерывание (англ. interrupt) приостанавливает основную
  // программу, выполняет заданную функцию, а затем возобновляет
  // основную программу. Нам нужно прерывание на нажатие кнопки,
  // т.е. при смене сигнала с высокого на низкий, т.е. на
```

```
  // нисходящем (англ. falling) фронте
  attachInterrupt(INT1, pushP1, FALLING); // INT1 — это 3-й пин
  attachInterrupt(INT0, pushP2, FALLING); // INT0 — это 2-й пин
}

void pushP1() { ++score; } // функция-прерывание 1-го игрока
void pushP2() { --score; } // функция-прерывание 2-го игрока
void loop()
{
  tone(BUZZER_PIN, 2000, 1000); // даём сигнал к старту.
  // пока никто из игроков не выиграл, обновляем «канат»
  while (abs(score) < MAX_SCORE) {
    int bound = map(score, -MAX_SCORE, MAX_SCORE, 0, BAR_COUNT);
    int left = min(bound, BAR_COUNT / 2 - 1);
    int right = max(bound, BAR_COUNT / 2);
    for (int i = 0; i < BAR_COUNT; ++i)
      digitalWrite(i + FIRST_BAR_PIN, i >= left && i <= right);
  }
  tone(BUZZER_PIN, 4000, 1000); // даём сигнал победы
  while (true) {} // «подвешиваем» плату до перезагрузки
}
```

Пояснения к коду

- Код нашей обычной программы исполняется инструкция за инструкцией и если мы, например, проверяем состояние датчика, мы к нему обратимся только в те моменты, когда очередь дойдет до соответствующей инструкции. Однако мы можем использовать прерывания:
  - по наступлении *определенного события*
  - на *определенном порту* ход программы будет приостанавливаться для выполнения
  - *определенной функции*, а затем программа продолжит исполняться с того места, где была приостановлена.
- Arduino Uno позволяет делать прерывания на портах 2 и 3.
- В `setup()` прописывается инструкция `attachInterrupt(interrupt, action, event)`, где
  - `interrupt` может принимать значения `INT0` или `INT1` для портов 2 и 3 соответственно
  - `action` — имя функции, которая будет вызываться при наступлении события
  - `event` — событие, которое мы отслеживаем. Может принимать значение `RISING` (изменение от низкого уровня сигнала к высокому, от 0 к 1), `FALLING` (от высокого уровня к низкому, от 1 к 0),`CHANGE` (от 0 к 1 или от 1 к 0), `LOW` (при низком уровне сигнала).

- Глобальные переменные, к которым мы обращаемся из функции, обрабатывающей прерывания, должны объявляться с использованием ключевого слова `volatile`, как в данном эксперименте `volatile int score = 0`.
- Внутри функции, вызываемой по прерыванию, нельзя использовать `delay()`.
- Функция `abs(value)` возвращает абсолютное значение `value` (значение по модулю). Обратите внимание, что функция может сработать некорректно, если передавать ей выражение, которое еще не вычислено, например `abs(++a)`, лучше передавать ей просто переменную.
- Функция `min(val1, val2)` вернет меньшее из `val1` и `val2`.
- Функция `max(val1, val2)` вернет большее из `val1` и `val2`.
- В данном эксперименте мы вычисляем значение, которое записывается на светодиоды, прямо в`digitalWrite()`
- Мы уже знакомы с логическим «и» (`&&`). Нередко нужен оператор «логическое «или»: `||`. Он возвращает «истину», если хотя бы один из операндов имеет значение «истина». `false || false`вернет `false`, а `true || true`, `true || false` и `false || true` вернут `true`.
- Мы использовали `while(true){}` для того, чтобы `loop()` остановился после того, как кто-то выиграл: у`while` всегда истинное условие и он бесконечно ничего не выполняет!

Вопросы для проверки себя

1. Каким образом мы подавляем дребезг аппаратно?
2. Для чего используются прерывания?
3. Каким образом можно включить обработку внешних прерываний?
4. О каких нюансах работы с уже известными нам вещами следует помнить при работе с прерываниями?
5. Как выбрать максимальное из двух значений? Минимальное?
6. Как получить абсолютное значение переменной? Чего следует избегать при использовании этой функции?
7. Когда оператор логическое «или» возвращает «ложь»?

Задания для самостоятельного решения

1. Вместо светодиодной шкалы подключите сервопривод и измените код таким образом, чтобы перетягивание демонстрировалось путем отклонения сервопривода от среднего положения.