Практическая работа 1

Проект маячок

В этом проекте научимся собирать простые схемы и управлять яркостью светодиода.

Задание 1. Ответить на вопросы

1. Что такое электрический ток?
2. Какие способы записи электрических схем Вы знаете? Назовите их достоинства и недостатки.
3. Запишите закон Ома для участка цепи.
4. Что такое короткое замыкание?
5. Какие способы управления электричеством Вы знаете?
6. Как узнать, что произошло короткое замыкание на плате Arduino?
7. Как определить сопротивление резистора? Назовите несколько способов.
8. Опишите принцип работы диода и светодиода

Задание 2. Определение номинала резистора

Резистор — искусственное «препятствие» для тока. Сопротивление в чистом виде. Резистор ограничивает силу тока, переводя часть электроэнергии в тепло.

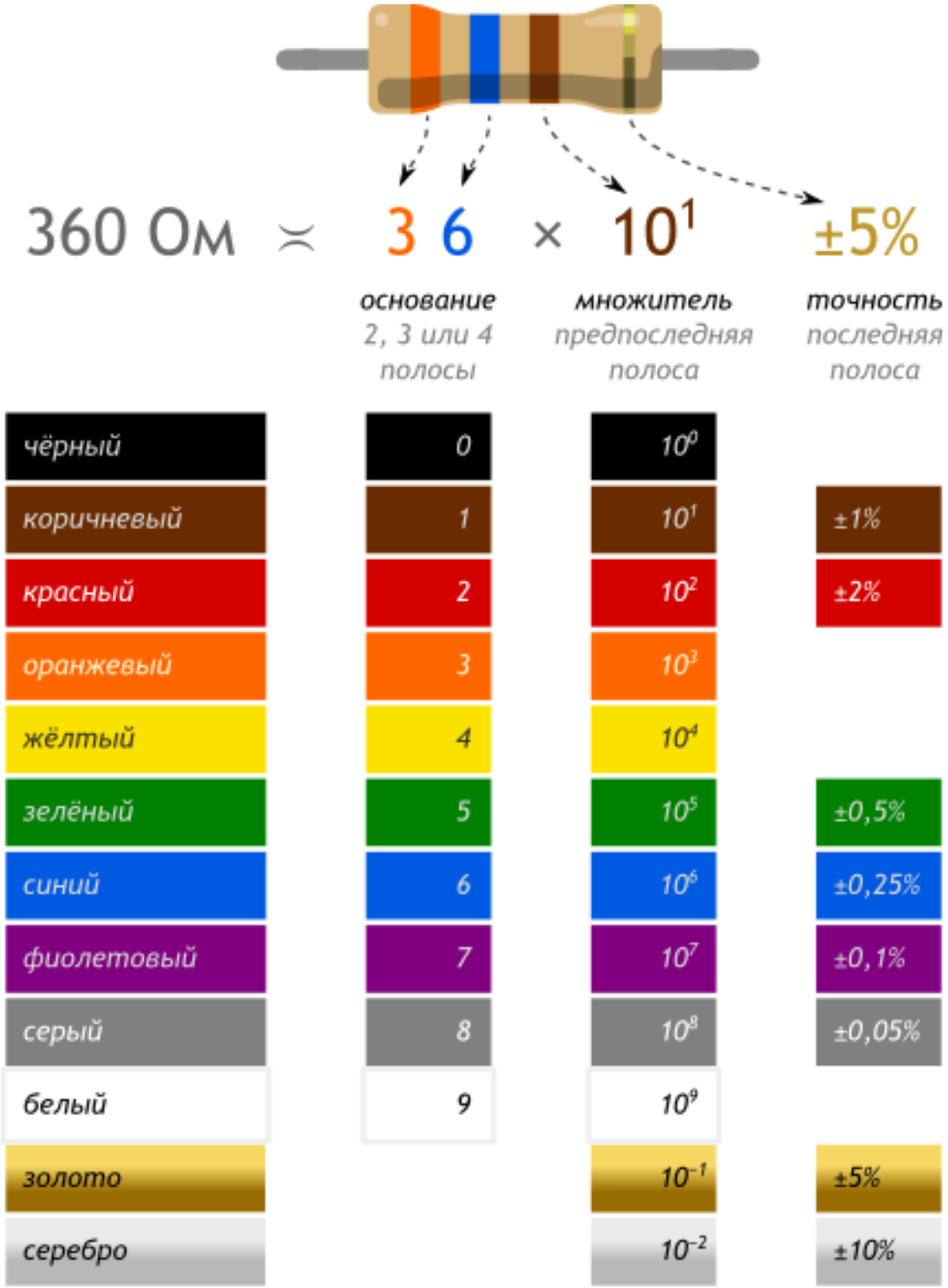
Основные характеристики

Сопротивление (номинал)	R	Ом
Точность (допуск)	±	%

Мощность	P	Ватт

Цветовая кодировка резисторов

Наносить номинал резистора на корпус числами — дорого и непрактично: они получаются очень мелкими. Поэтому номинал и допуск кодируют цветными полосками.



Разные серии резисторов содержат разное количество полос, но принцип расшифровки одинаков.

Цвет корпуса резистора может быть бежевым, голубым, белым. Это не играет роли.

Если не уверены в том, что правильно прочитали полосы, можете проверить себя с помощью мультиметра.

Типовые номиналы для экспериментов

1. Заполните таблицу (в тетради или электронном варианте)

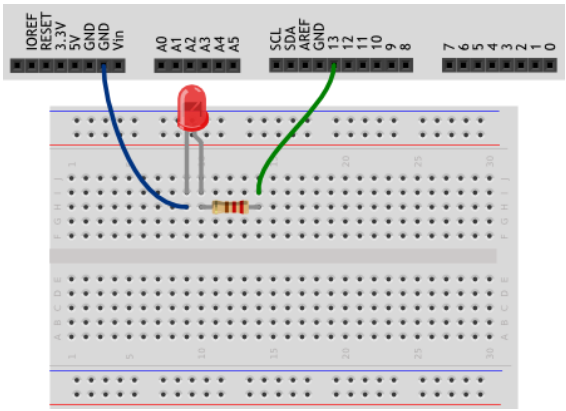
№	Рисунок резистора	Номинал, Ом	Точность, %
1.			
2.			
3.			

4.			
5.			

Задание 3. Список деталей для эксперимента

1. 1 плата Arduino Uno
2. 1 беспаечная макетная плата
3. 1 светодиод
4. 1 резистор номиналом 220 Ом
5. 2 провода «папа-папа»

Схема на макетной плате:



1. Зарисуйте принципиальную схему установки.

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

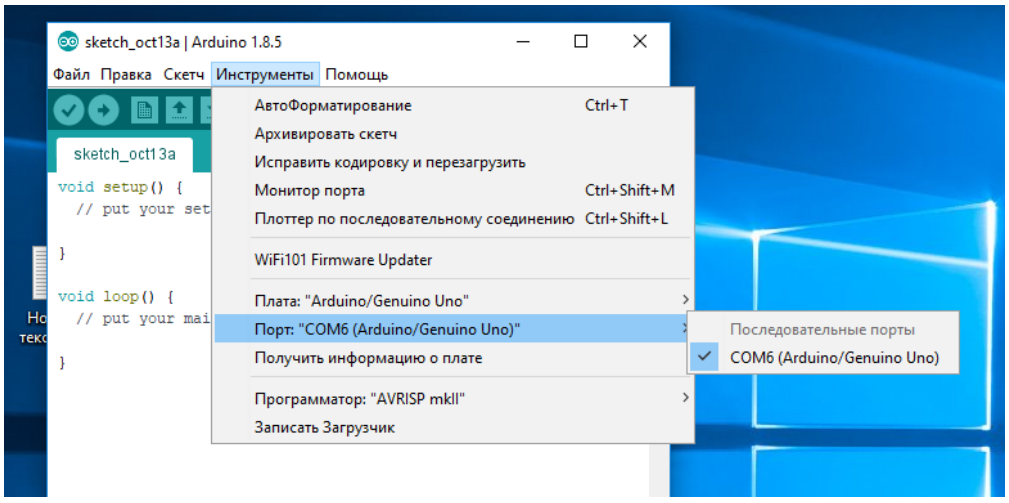
✓ Выбрать резистор нужного номинала можно с помощью таблицы маркировки или с помощью мультиметра в режиме измерения сопротивления

✓ Плата Arduino имеет три пина GND, используйте любой из них

Задание 4. Программирование микроконтроллера

1. Запустите приложение

2. Убедитесь, что выбран нужный порт



Каждое выражение — это приказ процессору сделать нечто. Выражения в рамках одного блока исполняются одно за другим, строго по порядку без

всяких пауз и переключений. То есть, если мы говорим об одном конкретном блоке кода, его можно читать сверху вниз, чтобы понять, что делается.

Теперь давайте поймём в каком порядке исполняются сами блоки, т.е. функции setup и loop. Не задумывайтесь пока что значат конкретные выражения, просто понаблюдайте за порядком.

Как только **Arduino** включается, перепрошивается или нажимается кнопка RESET, «нечто» вызывает функцию **setup**. То есть заставляет исполняться выражения в ней.

Как только работа **setup** завершается, сразу же «нечто» вызывает функцию **loop**.

Как только работа **loop** завершается, сразу же «нечто» вызывает функцию **loop** ещё раз и так до бесконечности.

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() {
  // put your setup code here, to run once:

}
```

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.

3. Напишите следующий код в процедуру Setup

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

Теперь давайте попробуем понять почему написанная программа приводит к конкретным действиям.

Как известно, пины Arduino могут работать и как выходы и как входы. Когда мы хотим чем-то управлять, то есть выдавать сигнал, нам нужно перевести управляющий пин в состояние работы на выход. В нашем примере мы управляем светодиодом на 13-м пине, поэтому 13-й пин перед использованием нужно сделать выходом.

Это делается выражением в функции setup:

pinMode(13, OUTPUT);

Выражения бывают разными: арифметическими, декларациями, определениями, условными и т.д. В данном случае мы в выражении осуществляем вызов функции. Помните? У нас есть свои функции setup и loop, которые вызываются чем-то, что мы назвали «нечто». Так вот теперь мы вызываем функции, которые уже написаны где-то.

Конкретно в нашем **setup** мы вызываем функцию с именем **pinMode**. Она устанавливает заданный по номеру пин в заданный режим: вход или выход.

О каком пине и о каком режиме идёт речь указывается нами в круглых скобках, через запятую, сразу после имени функции. В нашем случае мы хотим, чтобы **13-й пин** работал как выход. **OUTPUT** означает выход, **INPUT** — вход.

Уточняющие значения, такие как 13 и OUTPUT называются аргументами функции. Совершенно не обязательно, что у всех функций должно быть по 2 аргумента. Сколько у функции аргументов зависит от сути функции, от того как её написал автор. Могут быть функции с одним аргументом, тремя, двадцатью; функции могут быть без аргументов вовсе. Тогда для их вызова круглые скобка открывается и тут же закрывается:

noInterrupts();

На самом деле, вы могли заметить, наши функции **setup** и **loop** также не принимают никакие аргументы. И загадочное «нечто» точно так же вызывает их с пустыми скобками в нужный момент.

Вернёмся к нашему коду. Итак, поскольку мы планируем вечно мигать светодиодом, управляющий пин должен один раз быть сделан выходом и затем мы не хотим вспоминать об этом. Для этого идеологически и предназначена функция **setup**: настроить плату как нужно, чтобы затем с ней работать.

4. Пропишите следующий код в функцию Loop:

```
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
}
```

Она, как говорилось, вызывается сразу после **setup**. И вызывается снова и снова как только сама заканчивается. Функция **loop** называется основным циклом программы и идеологически предназначена для выполнения полезной работы. В нашем случае полезная работа — мигание светодиодом.

Пройдёмся по выражениям по порядку. Итак, первое выражение — это вызов встроенной функции **digitalWrite**. Она предназначена для подачи на заданный пин логического нуля (**LOW**, 0 вольт) или логической единицы (**HIGH**, 5 вольт) В функцию **digitalWrite** передаётся 2 аргумента: номер пина и логическое значение. В итоге, первым делом мы зажигаем светодиод на 13-м пине, подавая на него 5 вольт.

Как только это сделано процессор моментально приступает к следующему выражению. У нас это вызов функции **delay**. Функция **delay** — это, опять же, встроенная функция, которая заставляет процессор уснуть на определённое время. Она принимает всего один аргумент: время в миллисекундах, которое следует спать. В нашем случае это 100 мс.

Пока мы спим всё остаётся как есть, т.е. светодиод продолжает гореть. Как только 100 мс истекают, процессор просыпается и тут же переходит к следующему выражению. В нашем примере это снова вызов знакомой нам встроенной функции **digitalWrite**. Правда на этот раз вторым аргументом мы передаём значение **LOW**. То есть устанавливаем на 13-м пине логический ноль, то есть подаём 0 вольт, то есть гасим светодиод.

После того, как светодиод погашен мы приступаем к следующему выражению. И снова это вызов функции **delay**. На этот раз мы засыпаем на 900 мс.

Как только сон окончен, функция **loop** завершается. По факту завершения «нечто» тут же вызывает её ещё раз и всё происходит снова: светодиод поджигается, горит, гаснет, ждёт и т.д.

Если перевести написанное на русский, получится следующий алгоритм:

1) Поджигаем светодиод
2) Спим 100 миллисекунд
3) Гасим светодиод

4) Спим 900 миллисекунд
5) Переходим к пункту 1

Таким образом мы получили Arduino с маячком, мигающим каждые 100 + 900 мс = 1000 мс = 1 сек.

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

5. Запустите программу на компиляцию, и, если компилятор не выдаст ошибок, прошейте плату с помощью кнопки 

Задание 5. Ответьте на следующие вопросы

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?

Задание 6. Самостоятельно измените существующую программу и схему

1. Измените проект так, чтобы светодиод был включен в пин 5
2. Измените проект так, чтобы за 1 с светодиод моргал 2 раза
3. Измените проект так, чтобы маячок включался на 3 с после запуска устройства, а потом моргал в обычном режиме.
4. Добавьте в проект еще один светодиод. Настройте проект таким образом, чтобы:

4.1 светодиоды моргали одновременно.

4.2 Светодиоды моргали попеременно (сначала 1 потом 2)

4.3 Светодиоды моргали на манер железнодорожного семафора

5. Создать проект «Светофор», добавив к проекту еще 1 светодиод.

Принцип работы светофора: горит красный, красный одновременно с желтым, красный гаснет, горит один желтый, зеленый, зеленый моргает, желтый.