Муниципальное общеобразовательное учреждения
«Средняя общеобразовательная школа №2»
п. Бабынино Бабынинского района Калужской области

Принята педагогическим
советом школы
№ 1 от 30.08.2021г.

Утверждена приказом
№108 от 30.08.2021г.
Директор ОУ: М.С. Волошедова

Программа дополнительного образования в рамках Федерального проекта «Успех каждого» «Робототехника и схемотехника»

Срок реализации: 1 год
Возраст обучающихся: 13-14 лет

Бабынино
2021

Раздел 1. Комплекс основных характеристик программы

1.1. Пояснительная записка

Данная программа является дополнительной общеобразовательной общеразвивающей **технической направленности**, очной формы обучения, для **обучающихся 13-14 лет**, сроком реализации 1 год.

Программа разработана для обучения школьников робототехнике и схемотехники на платформе Arduino, которая позволяет собирать всевозможные электронные устройства и действующие модели роботов.

Актуальность и педагогическая целесообразность программы «Робототехника и схемотехника» состоит в том, что в ходе освоения создаётся уникальная образовательная среда, которая способствует развитию инженерного, конструкторского мышления. В процессе работы обучающиеся приобретают опыт решения как типовых, так и нешаблонных задач по конструированию, программированию, сбору данных. Кроме того, работа в команде способствует формированию умения взаимодействовать, формулировать, анализировать, критически оценивать, отстаивать свои идеи.

Программа составлен в соответствии с государственными требованиями к образовательным программам системы дополнительного образования детей на основе следующих нормативных документов:

1.Федеральный закон от 29 декабря 2012 года № 273-ФЗ «Об образовании в Российской Федерации»;

2.Приказ Министерства просвещения Российской Федерации от 09 ноября 2018 года № 196 «Об утверждении порядка организации и осуществления образовательной деятельности по дополнительным общеобразовательным программам»;

3.Письмо Минобрнауки РФ от 18.11.2015 № 09-3242 «О направлении рекомендаций» (вместе «Методические рекомендации по проектированию дополнительных общеразвивающих программ (включая разноуровневые программы)»;

4. Распоряжение Правительства Российской Федерации от 4 сентября 2014 года № 1726-р «Концепция развития дополнительного образования детей»;

5. Распоряжение Правительства Российской Федерации от 29 мая 2015 года № 996-р «Стратегия развития воспитания в Российской Федерации на период до 2025 года»;

6. Постановление Правительства Российской Федерации от 30 декабря 2015 года № 1493 «О государственной программе «Патриотическое воспитание граждан Российской Федерации на 2016-2020 годы»;

7. Постановление Главного государственного санитарного врача РФ от 04.07.2014 N 41"Об утверждении СанПиН 2.4.4.3172-14 "Санитарно-эпидемиологические требования к устройству, содержанию и организации режима работы образовательных организаций дополнительного образования детей;

8.Федеральная целевая программа развития образования на 2016-2020 годы, утвержденная Постановлением Правительства Российской Федерацииот23 мая 2015 года № 1499;

9.Устав учреждения. Локальные нормативные акты учреждения.

Отличительной особенностью программы является использование платформы Arduino, которая обеспечивает простоту при сборке начальных моделей, что позволяет получить результат в пределах одного или пары уроков. Возможности в изменении моделей и программ очень широкие и такой подход позволяет обучающимся усложнять модель и программу, проявлять самостоятельность в изучении темы. Программное обеспечение Arduino обладает очень широкими возможностями.

Программа модифицированная - составлена на основе программ дополнительного образования по робототехнике, разработанных другими педагогами и изученных в сети Интернет.

Адресат программы

Обучение по данной программе рассчитано на обучающихся в возрасте 13-14 лет.

Срок обучения 1 год

Объём программы – 35 часов.

Уровень освоения содержания –стартовый

Форма обучения – очная

При планировании образовательного процесса предусматриваются следующие **формы организации познавательной деятельности:**

- коллективные (фронтальные со всем составом);
- групповые (работа в группах, бригадах, парах);
- индивидуальные.

Формы организации учебных занятий

- консультации;
- практикумы;
- проекты;
- проверки и коррекции знаний и умений;
- выставки;
- соревнования.

Виды занятий – контрольные и открытые занятия, соревнования.

Срок освоения программы – 1 год

Режим занятий –1 раз в неделю по 1 часу.

Условия реализации программы

Группы формируются в соответствии с возрастом обучающихся, без предварительного отбора, по заявлению родителей. Допускается комплектование разновозрастных групп.

1.2. Цель и задачи программы

Цель программы:

Формирование у обучающихся теоретических знаний и практических навыков в области начального технического конструирования, схемотехники и основ программирования, применяемых при последующей разработке робототехнических и электронных устройств в малых группах.

Задачи программы:

Образовательные

- Ознакомить обучающихся с комплексом базовых технологий, применяемых при создании электронных устройств;

- Подготовить к изучению школьных курсов физики, информатики и реализовать межпредметные связи с математикой;
- Научить решать ряд кибернетических задач, результатом каждой из которых будет работающий механизм или электронное устройство с автономным управлением;
- Организовать участие в играх, конкурсах и состязаниях;
- Познакомить с миром инженерных профессий;
- Способствовать ранней профессиональной ориентации обучающихся;

Развивающие

- Развивать у обучающихся инженерное мышление, навыки конструирования, программирования и эффективного использования кибернетических систем;
- Развивать мелкую моторику, внимательность, аккуратность и изобретательность;
- Развивать креативное и проектное мышление;
- Развивать пространственное воображение;
- Развивать навыки инженерного мышления;

Воспитательные

- Повышать мотивацию обучающихся к изобретательству и созданию собственных роботизированных систем;
- Формировать у обучающихся стремление к получению качественного законченного результата;
- Формировать навыки работы в команде.

1.3. Учебно-тематический план

№	Наименование разделов и тем	Всего часов	Теория / Практика	Формы аттестации/ контроля
1	Повторение. Основы работы с конструктором «Матрешка» Техника безопасности при работе.	1	1	
2	Электричество. Управление электричеством.	1	1	Проверочная работа
3	Основные элементы схемы. Принципиальная схема. Сборка схем.	1	1	
4	Резистор, диод, светодиод. Широтно-импульсная модуляция.	1	1	
5	Делитель напряжения. Пьезодинамик.	1	1	
6	Биполярный транзистор, Светодиодные сборки.	1	1	
7	Полевой транзистор, мотор. Конденсатор	1	1	
8	Эксперимент 9. Миксер	2	1 / 1	
9	Эксперимент 10. Кнопочный переключатель	2	1 / 1	

10	Эксперимент 11. Светильник с кнопочным управлением	2	1 / 1	
11	Эксперимент 12. Кнопочные ковбои	2	1 / 1	
12	Эксперимент 13. Секудномер	2	1 / 1	
13	Эксперимент 14. Счётчик нажатий	2	1 / 1	
14	Эксперимент 15. Комнатный термометр	2	1 / 1	
15	Эксперимент 16. Метеостанция	2	1 / 1	
16	Эксперимент 17. Пантограф	2	1 / 1	
17	Эксперимент 18. Тестер батареек	2	1 / 1	
18	Эксперимент 19. Светильник, управляемый по USB	2	1 / 1	
19	Эксперимент 20. Перетягивание каната	2	1 / 1	
20	Проектная деятельность	3	/ 3	
21	Выставка. Защита проектов.	1	/ 1	

ВСЕГО	35	19 16	

Экспериментальные работы

Эксперимент 9. Миксер

В этом эксприменте мы создаем модель миксера с двумя скоростями работы.

Прочтите перед выполнением

- Полевой транзистор
- Мотор

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

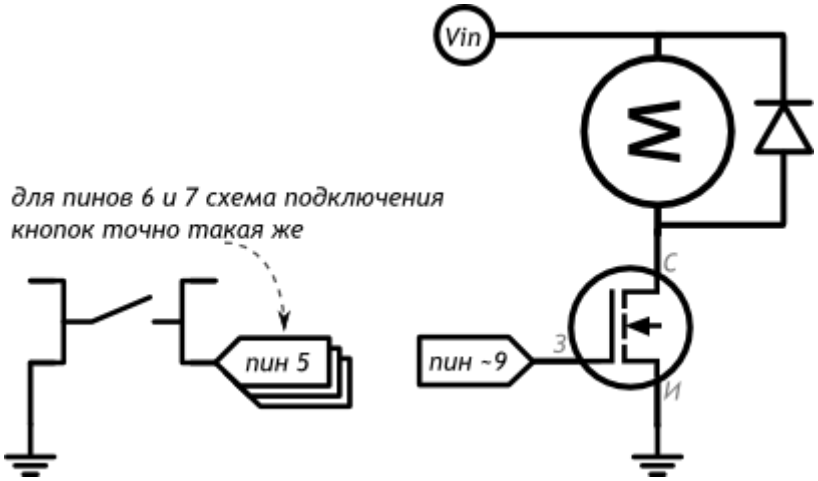
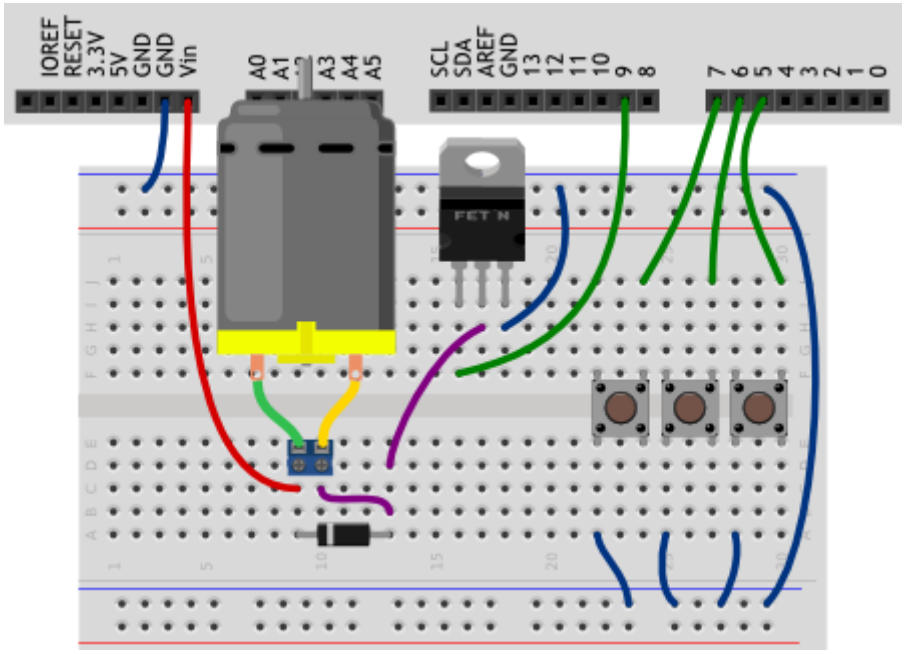


Схема на макетке



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

p090_mixer.ino

```
#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

В этом эксперименте мы делаем из тактовой кнопки триггер, борясь с «дребезгом».

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

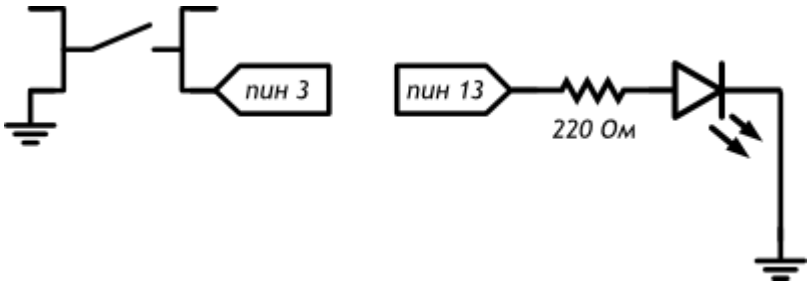
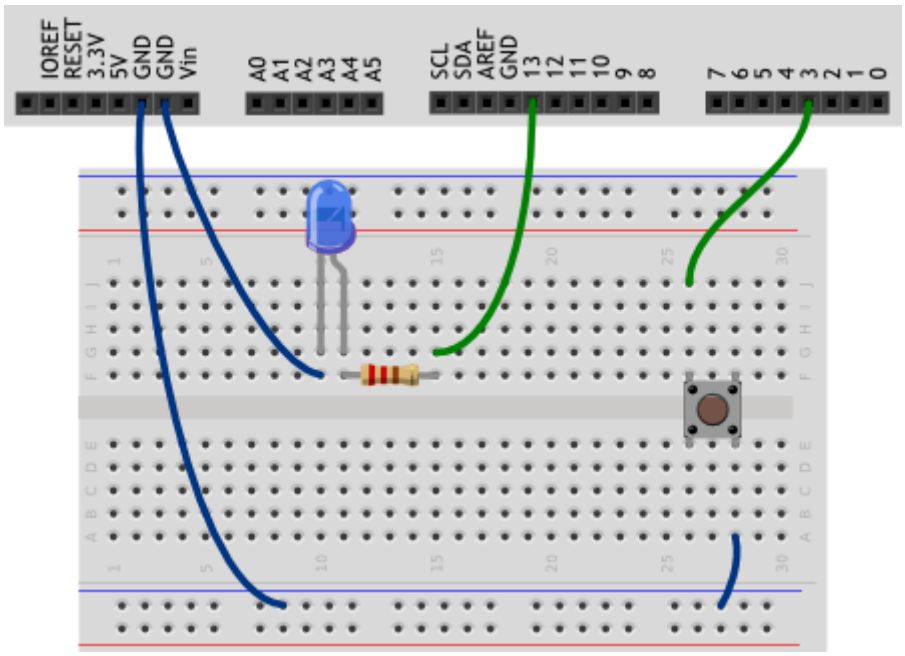


Схема на макетке



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

p100_led_toggle.ino

```
#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

В этом эксперименте мы добавляем порцию яркости светодиоду одной кнопкой и убавляем другой.

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

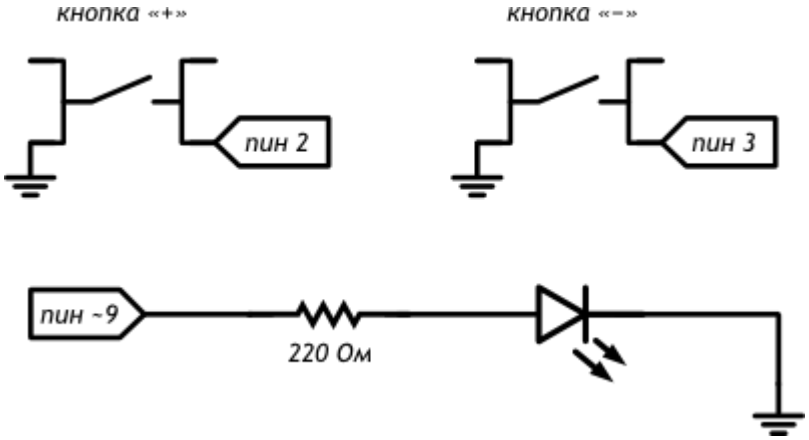
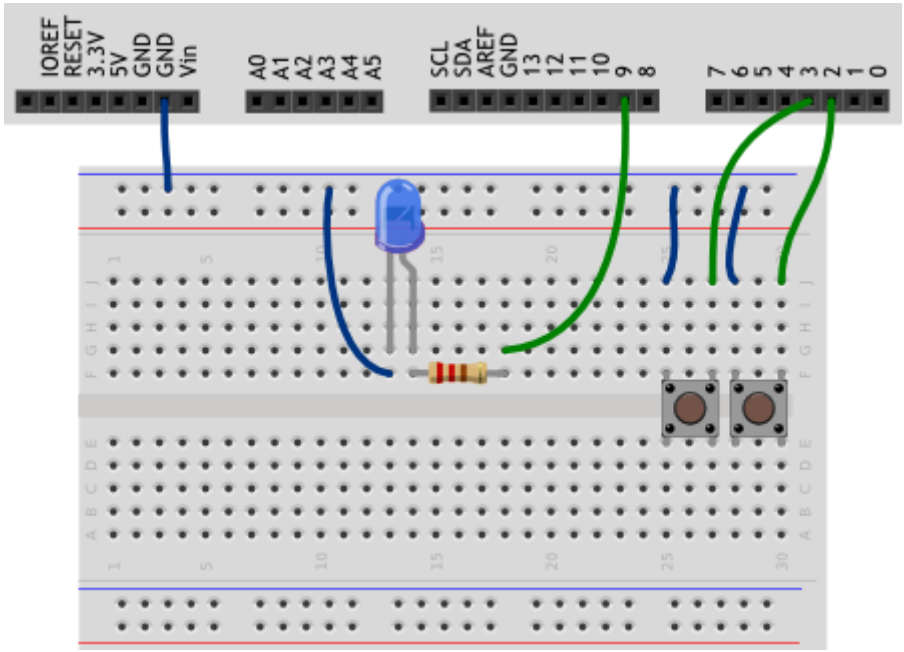


Схема на макетке



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

p110_plus_minus_light.ino

```
#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

В этом эксперименте мы создаем игрушку на реакцию: кто быстрее нажмет кнопку по сигналу.

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

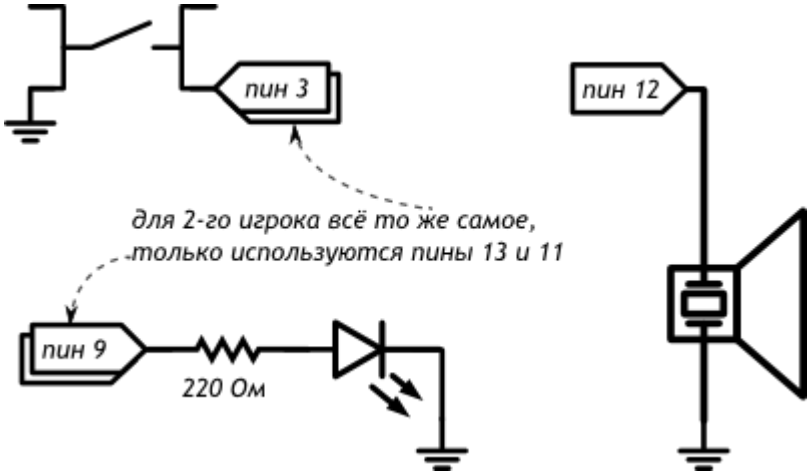
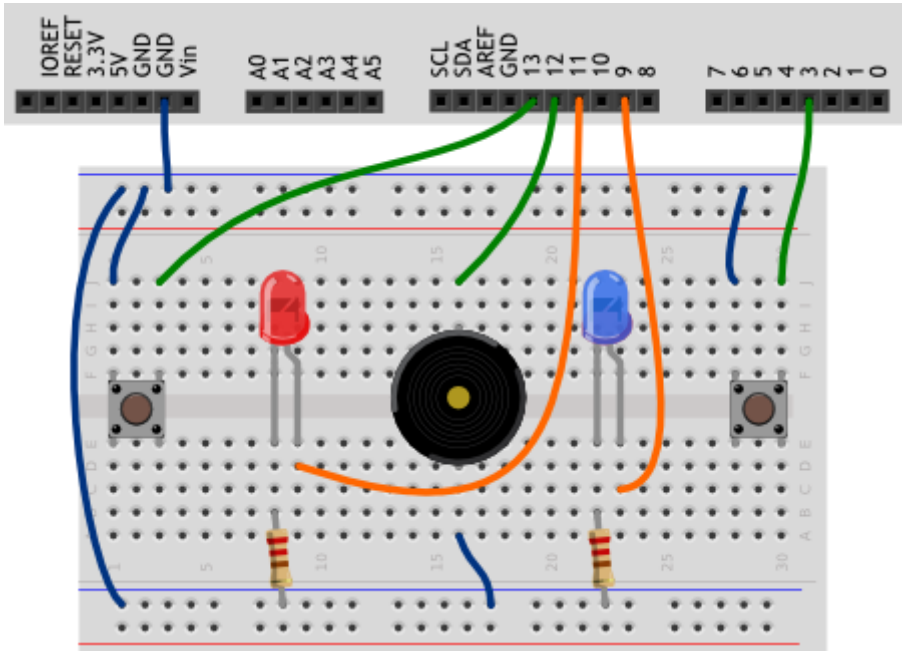


Схема на макетке



Скетч

p120_button_cowboys.ino

```
#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 `for` 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. 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

В этом эксперименте мы создаем секундомер, который считает до 10.

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

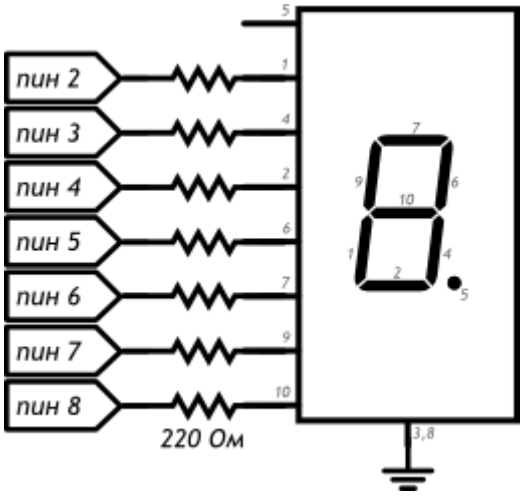
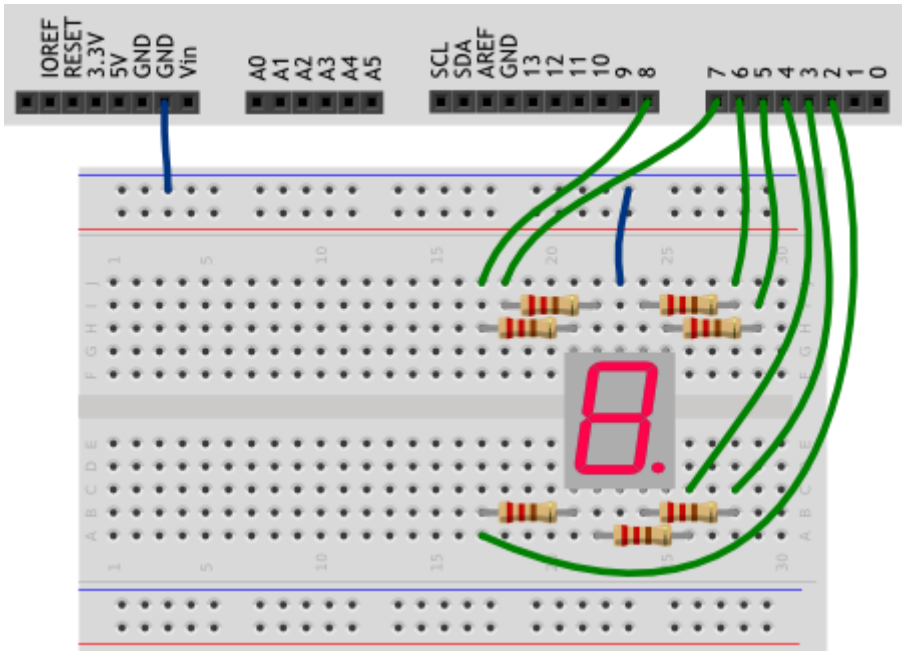


Схема на макетке



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

p130_seven_segment_counter.ino

```
#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

В этом эксперименте мы выводим на семисегментный индикатор количество нажатий на кнопку (единицы).

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

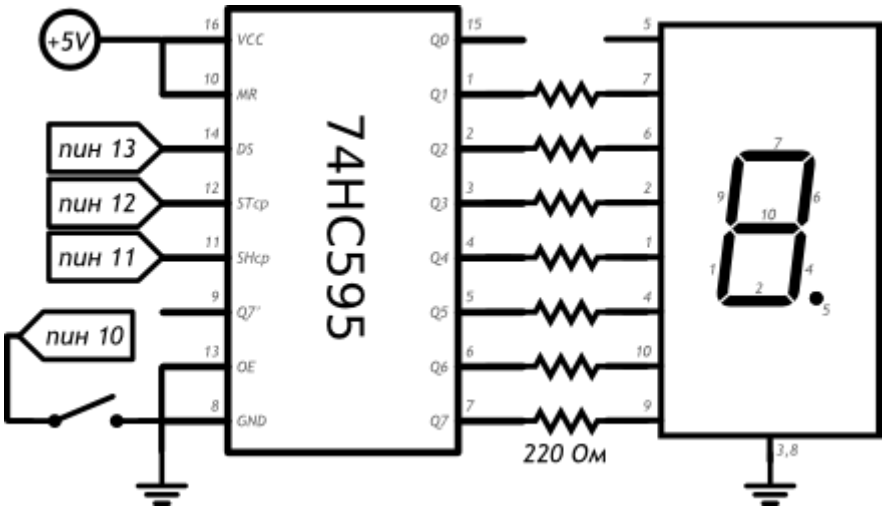
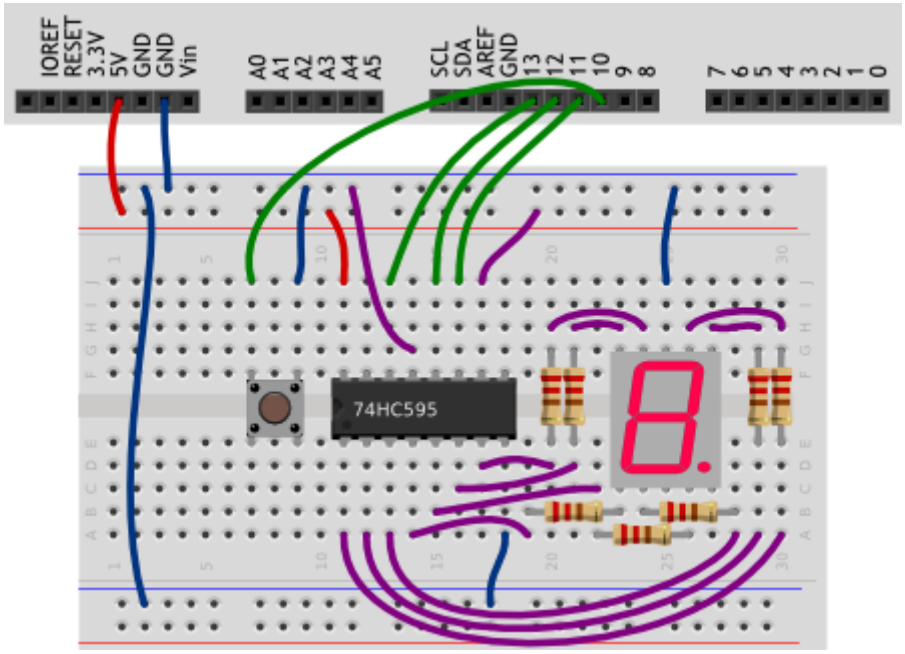


Схема на макетке



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'е. 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

p140_seven_segment_clicker.ino

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

В этом эксперименте мы измеряем температуру окружающей устройство среды и с помощью шкалы показываем, на сколько она превышает заданный порог.

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

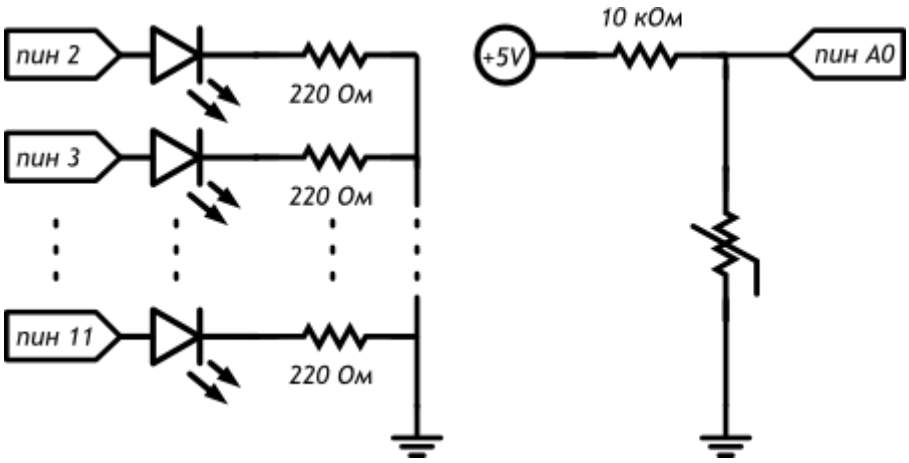
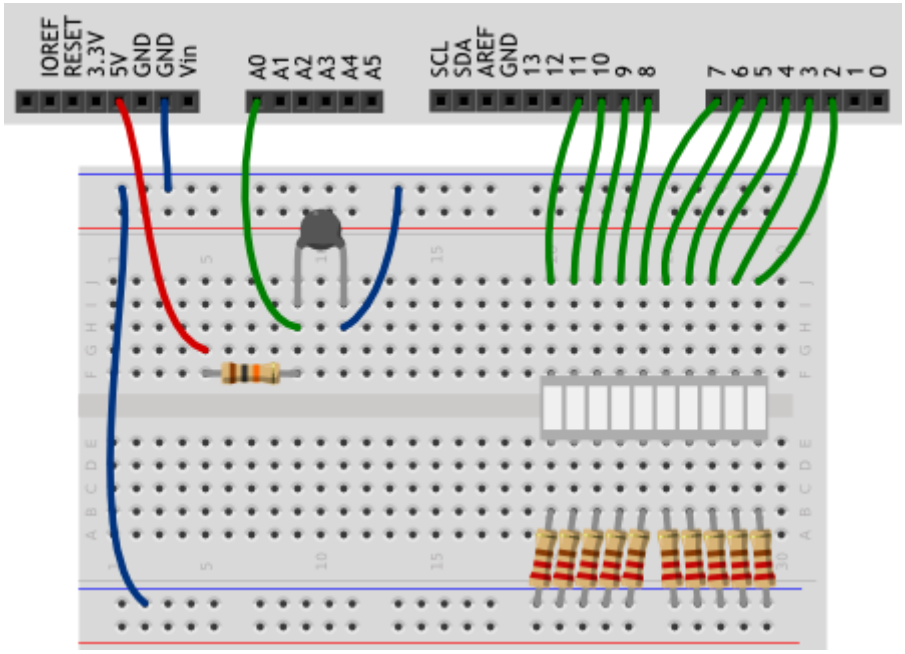


Схема на макетке



Обратите внимание

- Термистор мы включили в известную нам схему делителя напряжения.

Скетч

p150_bar_thermometer.ino

```
// 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 `float` 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 $-3.4028235 \times 10^{38}$ 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

В этом эксперименте мы передаем данные об измерениях температуры на компьютер (например, для последующей обработки).

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

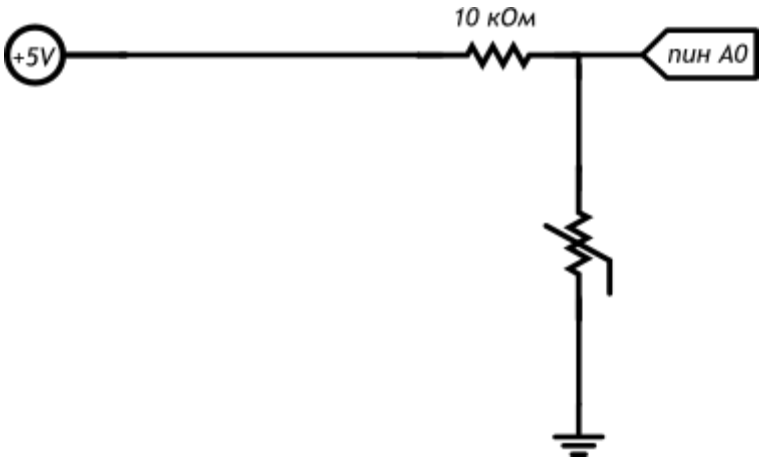
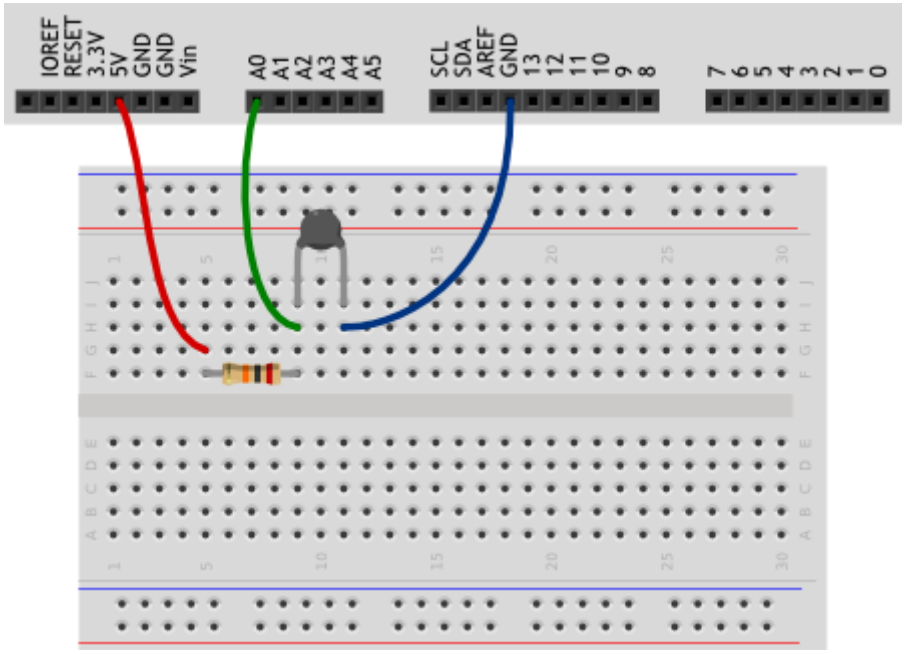


Схема на макетке



Скетч

p160_meteostation.ino

```
#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

В этом эксперименте мы вращаем сервопривод на угол, задаваемый потенциометром.

Прочтите перед выполнением

- Конденсатор

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

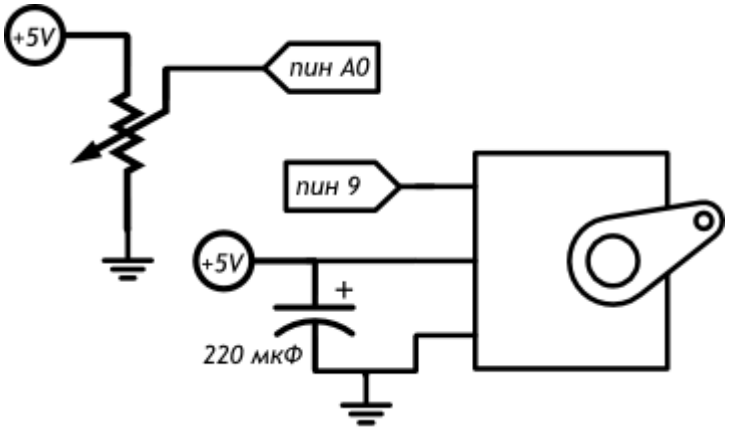
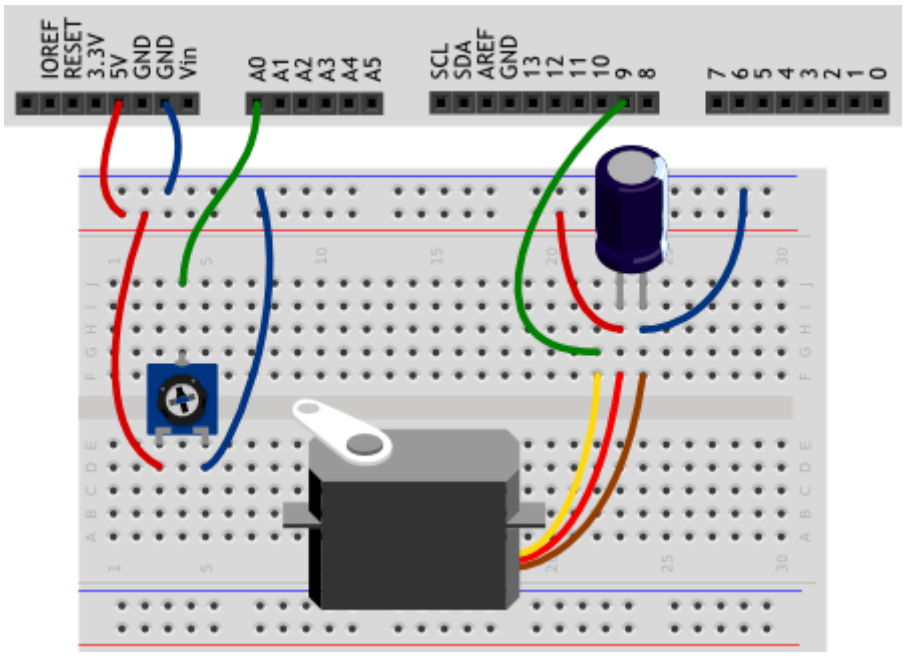


Схема на макетке



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

p170_servo.ino

```
// 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

В этом эксперименте мы выводим на жидкокристаллический дисплей данные о напряжении, измеренном на батарейке.

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

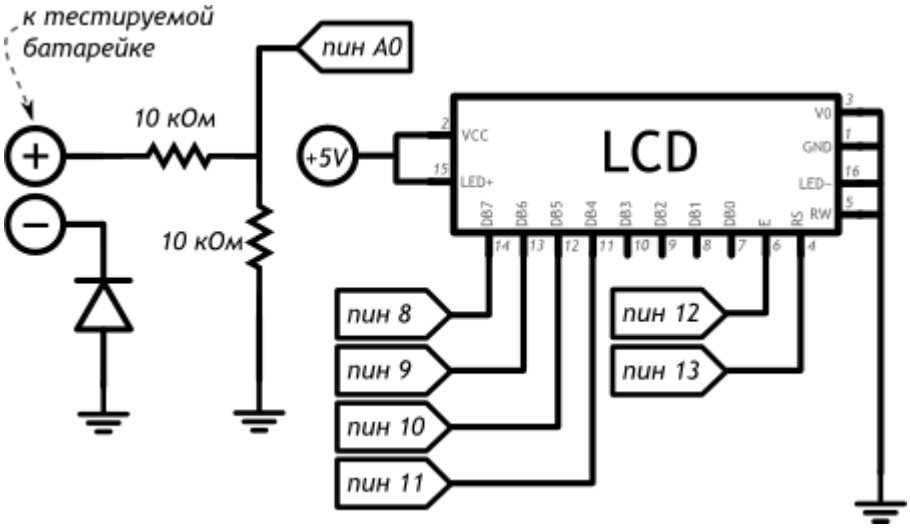
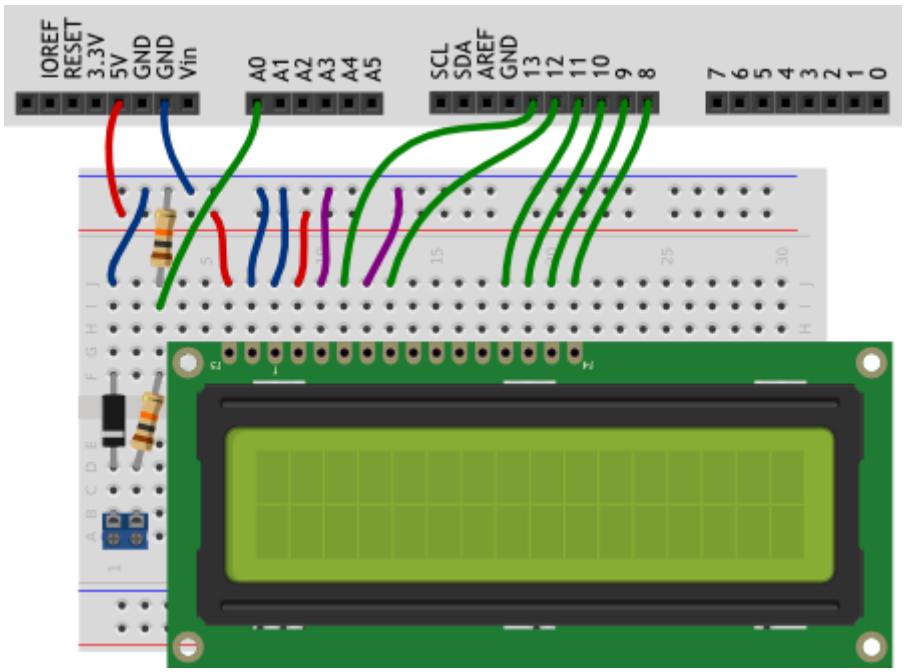


Схема на макетке



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

p180_lcd.ino

```
// 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 DB4 DB5 DB6 DB7
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

В этом эксперименте мы отправляем устройству команды, как ему светить.

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

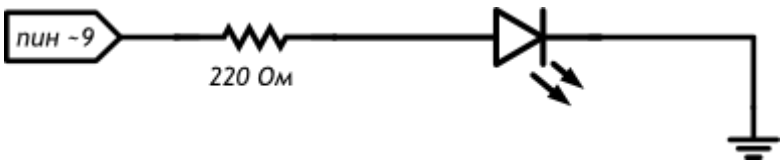
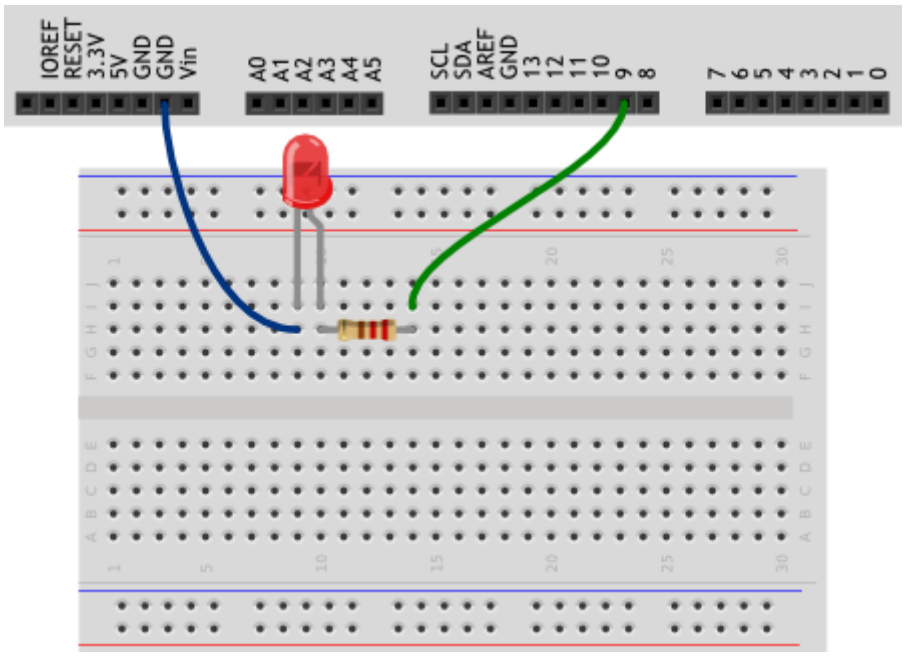


Схема на макетке



Скетч

p190_serial_brightness.ino

```
#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

В этом эксперименте мы создаем еще одну игру, на этот раз нужно быстрее соперника нажать кнопку 20 раз.

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

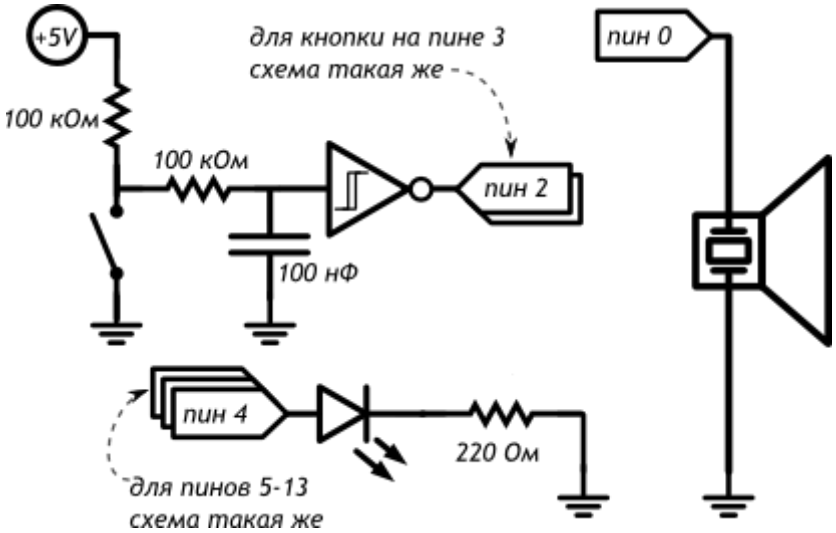
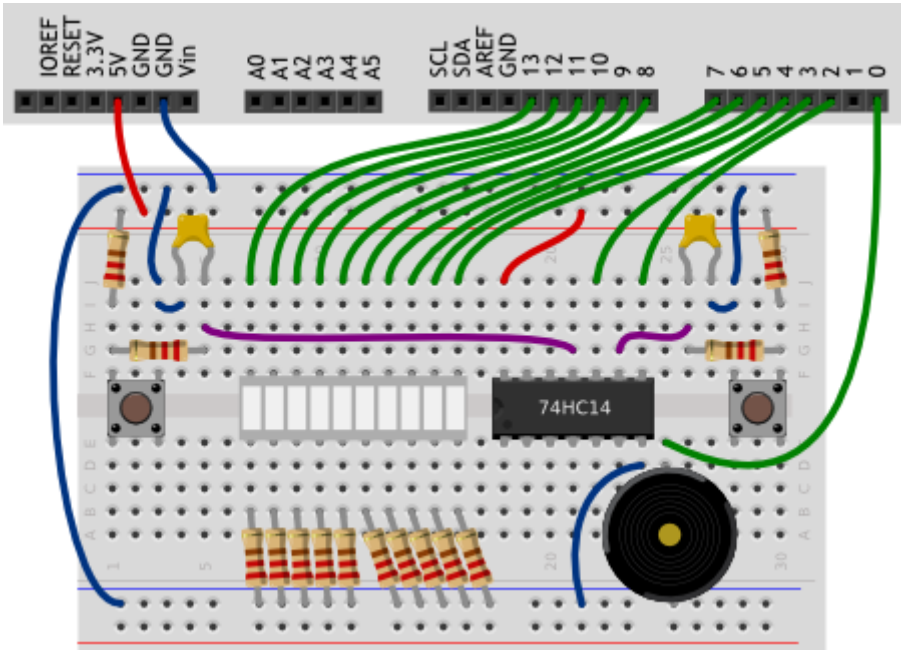


Схема на макетке



Обратите внимание

- Схема подключения кнопок с использованием конденсаторов, резисторов и микросхемы 74HC14, которая называется инвертирующий триггер Шмитта, нужна для аппаратного подавления дребезга. Посмотрите видеоурок 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

p200_button_wrestling.ino

```
#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. Можно задать эти значения и с помощью функции `digitalPinToInterrupt(pin)`, где вместо `pin` указать номер пина.
  - `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.

1.5. Планируемые результаты

1. Предметные результаты.

В результате освоения программы обучающиеся должны:

Знать:

1. Основные понятия электричества и принципы работы электронных устройств;
2. роль и место робототехники, программирования и схемотехники в жизни современного общества;
3. основные сведение из истории развития робототехники в России и мире;
4. основные понятия схемотехники, основные технические термины, связанные с процессами конструирования и программирования электронных устройств;
5. общее устройство и принципы действия электронных устройств;
6. методику проверки работоспособности отдельных узлов и деталей;
7. основы популярных языков программирования;
8. основные законы электрических цепей, правила безопасности при работе с электрическими цепями, основные радиоэлектронные компоненты;
9. определения робототехнического устройства, наиболее распространенные ситуации, в которых применяются роботы;
10. о перспективах развития робототехники и схемотехники, основные компоненты программных сред;

Уметь:

1. собирать простейшие модели с использованием конструктора «Матрешка»;
2. самостоятельно проектировать и собирать простейшие электронные устройства;
3. работать в визуальной среде программирования, программировать собранные конструкции под задачи начального уровня сложности;
4. пользоваться компьютером, программными продуктами, необходимыми для обучения программе;
5. подбирать необходимые датчики и исполнительные устройства, собирать простейшие устройства с одним или несколькими датчиками, собирать и отлаживать простейшие электронные устройства;

6. правильно выбирать вид передачи механического воздействия для различных технических ситуаций, собирать действующие модели роботов, а также их основные узлы и системы.

2. Личностные результаты:

1. уметь ориентироваться в информационном пространстве;
2. искать информацию в свободных источниках и структурировать её;
3.самостоятельно создавать способы решения проблем творческого и поискового характера;
4. обладать навыками критического мышления;
5. уметь генерировать, комбинировать, видоизменять и улучшать идеи;
6. уметь с уважением относиться к собственному и чужому труду.

3. Метапредметные результаты:

1. уметь слушать и слышать собеседника;
2. уметь аргументировано отстаивать точку зрения;
3. уметь работать индивидуально и в группе;
4. уметь формулировать проблему, выдвигать гипотезу, ставить вопросы;
5. уметь правильно организовывать рабочее место и время для достижения поставленных целей;
6. уметь вести собственный проект.

1.6. Условия реализации программы

Материально-техническое обеспечение

1. Hi-Tech конструктор на основе платформы Arduino «Матрешка»
2. При организации практических занятий и творческих проектов формируются малые группы, состоящие из 2-3 обучающихся. Для каждой группы выделяется отдельное рабочее место, состоящее из компьютера и конструктора.