Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования Санкт-Петербургский горный университет

Кафедра системного анализа и управления

ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ

Методические указания к лабораторным работам для студентов бакалавриата направления 27.03.04

САНКТ-ПЕТЕРБУРГ
2023

УДК 004.43 (073)

ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ: Методические указания к лабораторным работам / Санкт-Петербургский горный университет. Сост.: *Ю.В. Ильюшин, Т.В. Кухарова*. СПб, 2023. 39 с.

В методических указаниях содержатся краткие теоретические сведения и методические рекомендации к лабораторным работам по дисциплине «Прикладное программирование».

Предназначены для подготовки студентов бакалавриата по направлению подготовки 27.03.04 «Управление в технических системах».

Научный редактор проф. *Д.А. Первухин*

Рецензент проф. *И.М. Першин* (Северо-Кавказский федеральный университет)

ВВЕДЕНИЕ

Курс «Прикладное программирование» является одним из курсов, направленных на формирование общепрофессиональных и профессиональных компетенций при подготовке специалистов по профилю «Информационные технологии в управлении» направления 27.03.04 – «Управление в технических системах». Изучение данного курса является необходимым элементом при подготовке высококвалифицированных кадров.

Основной целью выполнения лабораторных работ по дисциплине «Прикладное программирование» является получение обучающимися навыков разработки схемных решений на платформе Arduino и написания программ для микроконтроллеров. В методических рекомендациях кратко изложены основные теоретические сведения по каждой из тем, приведены примеры и задания для самостоятельного выполнения.

По результатам выполнения лабораторных работ обучающиеся формируют отчеты.

Отчет о выполнении лабораторной работы должен содержать:

– цель работы;

– задания на лабораторную работу;

– схему разработанного на платформе Arduino устройства;

– листинг программы, реализующей разработанный алгоритм;

– результаты тестирования программы;

– выводы.

ЛАБОРАТОРНАЯ РАБОТА №1. УПРАВЛЕНИЕ ЦИФРОВЫМИ ВЫВОДАМИ ARDUINO

Цель работы: получить навыки конфигурирования назначения цифровых выводов, подключения внешних компонентов, использования широтно-импульсной модуляции.

Работа с макетной платой

Макетная плата - удобный инструмент для экспериментов, позволяющий легко собирать простые схемы без изготовления печатных плат и пайки. С двух сторон по всей длине макетной платы расположены красные и синие отверстия. Все красные отверстия соединены между собой и служат, как правило, для подачи питания. Для большинства проектов из этой книги это +5 В. Все синие отверстия тоже электрически соединены друг с другом и играют роль шины заземления.

Каждые пять отверстий, расположенных вертикальными рядами, также соединены друг с другом. Посередине есть свободное место для удобства установки компонентов на макетной плате. Электрические соединения отверстий показаны на рис. 1.1 утолщенными линиями.

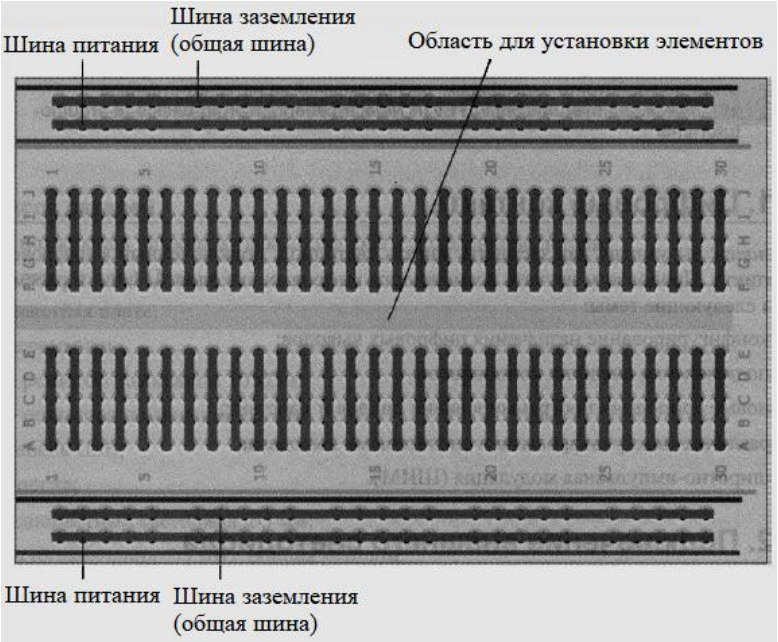

Рис. 1.1. Электрические соединения макетной платы

Подсоединение светодиодов

Подключая светодиоды, необходимо соблюдать правильную полярность. Положительный вывод светодиода называется анодом, отрицательный- катодом. Определить назначение контактов светодиода можно визуально: вывод катода короче, чем анода.

Ток через светодиод течет только в одном направлении: от анода к катоду. Поскольку ток протекает от положительного полюса к отрицательному, анод светодиода следует подключить к источнику тока (цифровой выход +5 В), а катод - к земле. Резистор может быть подключен последовательно с любым из выводов светодиода. Полярность подключения для резисторов не важна.

Подключать светодиод к контакту 9 Arduino нужно последовательно с резистором, который выступает в качестве ограничителя тока. Чем больше сопротивление резистора, тем сильнее он ограничивает ток. В этом примере мы применим резистор номиналом 220 Ом. Монтажная схема изображена на рис. 1.2.

Программирование цифровых выводов

По умолчанию все внешние контакты Arduino сконфигурированы как входы. Если необходимо использовать контакт Arduino как выход, нужно его переконфигурировать, подав соответствующую команду микроконтроллеру.

Каждая программа для Arduino должна включать две обязательные функции: setup() и loop() .

Функция setup() запускается один раз в начале программы, а loop() работает как цикл. Поскольку каждый контакт обычно конфигурируется в программе один раз, логично делать это в теле функции setup() .

Для начала напишем простую программу, которая при запуске сконфигурирует контакт 9 как выход. В программе будут еще две функции: pinMode() - для конфигурации контакта и digitalWrite() - для установки значения HIGH (5 В) на этом контакте:

```
const int LED=9; // Константа - номер контакта светодиода
```

```
void setup () {
pinMode (LED,OUTPUT); // Конфигурируем контакт светодиода как //выход
digitalWrite(LED, HIGH); // Устанавливаем значение HIGH на выходе
}
void loop() {
// В цикле ничего не выполняем
}
```

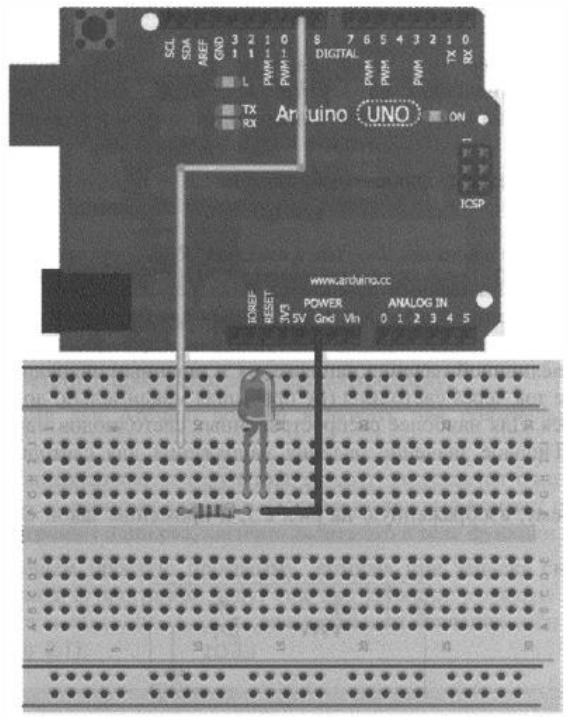

Рис. 1.2. Подключение светодиода к плате Arduino Uno

Соберите схему, как показано на рис. 1.2, и загрузите код листинга в плату Arduino.

Широтно-импульсная модуляция с помощью analogWrite()

Цифровые контакты Arduino очень удобны для переключения светодиодов, управления реле и двигателями постоянного тока. Но что делать, если необходимо вывести напряжение, отличное от 0 и 5 В? С помощью контактов одной только платы Arduino Uno это невозможно. Придется задействовать цифроаналоговый преобразователь или взять плату Arduino Due или добавить внешнюю микросхему ЦАП.

Тем не менее, можно сымитировать генерацию аналоговых значений на цифровых контактах с помощью широтно-импульсной модуляции (ШИМ). Для некоторых контактов Arduino сформировать ШИМ-сигнал можно командой analogWrite ().

Контакты, которые могут выдавать ШИМ-сигнал на определенные периферийные устройства, помечены символом ~ на плате Arduino. На Arduino Uno контакты 3, 5, 6, 9, 10, 11 поддерживают выдачу ШИМ-сигнала. При наличии Arduino Uno проверить команду analogWrite() можно с помощью схемы, изображенной на рис. 1.1.

Если уменьшить напряжение на контакте 9 Arduino, яркость свечения светодиода должна стать меньше, потому что снизится ток, текущий через него. Этого эффекта можно добиться с помощью ШИМ и команды analogWrite () .

Функция analogWrite () имеет два аргумента: номер контакта и 8-разрядное значение в диапазоне от 0 до 255, устанавливаемое на этом контакте.

В следующем листинге приведен код программы генерации ШИМ-сигнала на контакте 9 для плавного управления яркостью светодиода.

```
const int LED=9; // Константа номера контакта светодиода
void setup () {
pinMode (LED, OUTPUT); // Конфигурируем контакт светодиода как //выход
}
void loop() {
```

```
for (int i=0; i<256; i++) {
analogWrite(LED, i);
delay (10);
}
for (int i=255; i>=0; i--) {
analogWrite(LED, i);
delay(10);
}
}
```

Вы будете наблюдать, как свечение светодиода изменяется от тусклого к яркому в одном цикле for, а затем от яркого к тусклому в другом цикле for. Все это будет происходить в основном цикле loop () до бесконечности.

RGB-светодиод

Самый длинный вывод RGB-светодиода - это общий катод, а все остальные - это аноды, каждый отвечает за свой цвет (рис. 1.3): самый нижний - красный, второй сверху - зеленый, самый верхний - синий.

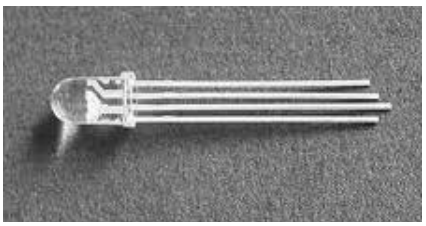

Рис. 1.3. RGB-светодиод

Задание на лабораторную работу

Собрать схему и написать программу для плавного циклического изменения цвета RGB-светодиода: в первом цикле увеличиваем яркость зеленого от 0 до максимума, при этом уменьшаем яркость красного от максимума до 0; во втором цикле яркость зеленого уменьшаем, увеличиваем яркость синего; в третьем уменьшаем яркость синего, увеличиваем яркость красного.

ЛАБОРАТОРНАЯ РАБОТА № 2. СЧИТЫВАНИЕ ДАННЫХ С ЦИФРОВЫХ КОНТАКТОВ

Цель работы: научиться считывать значения на входе, обрабатывать нажатие кнопки.

Рассмотрим еще одну функцию цифровых контактов. До сих пор мы использовали их в качестве выходов, генерируя цифровой сигнал и ШИМ-сигнал. Следующий шаг - функционирование контактов платы Arduino в качестве входов. Это позволит подключить, например, переключатели и кнопки для взаимодействия со своим устройством в режиме реального времени.

Считывание цифровых входов со стягивающим резистором

Подключим к цифровому контакту кнопку и стягивающий резистор, в результате схема примет вид, представленный на рис. 2.1.

Прежде чем написать программу опроса состояния кнопки, важно понять назначение резистора в этой схеме. Почти для всех цифровых входов необходим дополнительный стягивающий **(pull-down)** или подтягивающий **(pull-up)** резисторы для установки"значения по умолчанию" на входном контакте. Представьте себе, что в схеме на рис. 2.1 нет резистора 10 кОм. В этом случае при нажатии на кнопку на выводе будет значение HIGH.

Но что происходит, когда кнопка не нажата? В такой ситуации входной контакт не привязан ни к чему, как говорят, "висит в воздухе".

А поскольку вывод физически не подключен ни к 0 В, ни к 5 В, чтение значения может дать неожиданный результат. Электрические помехи на близлежащих выводах могут привести к тому, что значение напряжения будет колебаться между HIGH и LOW. Чтобы предотвратить это, стягивающий резистор подключают так, как показано на рис. 2.1.

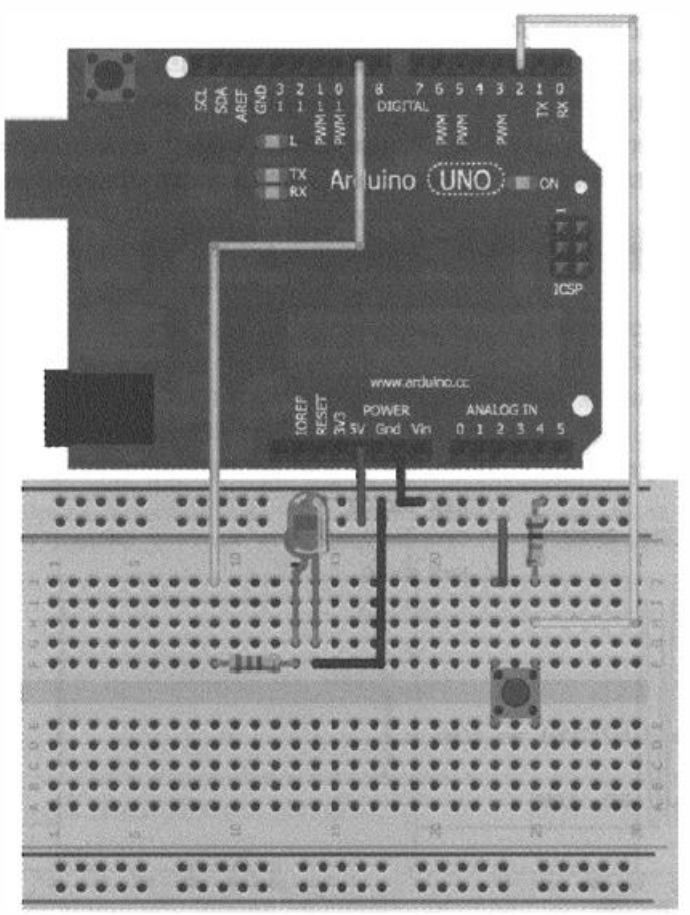

Рис. 2.1. Схема подключения элементов

Посмотрим, что происходит, когда кнопка не нажата, а входной контакт подключен через стягивающий резистор 10 кОм к земле. Через резистор протекает ток утечки и на входном контакте будет установлено значение напряжения LOW. 10 кОм – довольно распространенный номинал для стягивающего резистора. При нажатии на кнопку входной контакт оказывается напрямую связан с шиной 5 В. Теперь ток может течь двумя путями:

- через практически нулевое сопротивление нажатой кнопки к шине 5 В;

- через высокое сопротивление резистора на землю.

В соответствии с законом Ома ток всегда будет идти по пути наименьшего сопротивления.

Большая часть тока будет протекать через замкнутую кнопку и на входе установится уровень HIGH.

В рассмотренном примере используется стягивающий резистор, но возможна установка и подтягивающего резистора, подключенного к шине 5 В, тогда кнопка должна быть соединена с землей. В таком случае на входном контакте будет значение HIGH при отпущенной кнопке и значение LOW, когда кнопка нажата.

Стягивающие и подтягивающие резисторы важны, потому что они гарантируют, что кнопка не создаст короткое замыкание между 5 В и землей при нажатии и что входной контакт не останется в "подвешенном" состоянии.

Теперь напишем программу для рассмотренной схемы. Светодиод должен гореть, пока кнопка нажата, и быть выключенным, когда кнопка отжата.

```
const int LED=9; // Контакт 9 для подключения светодиода
const int BUTTON=2; // Контакт 2 для подключения кнопки
void setup () {
pinMode (LED, OUTPUT); // Сконфигурировать контакт светодиода как выход
pinMode (BUTTON, INPUT); // Сконфигурировать контакт кнопки как вход
}

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

В коде листинга реализованы некоторые новые элементы: функция digitalRead () и оператор if/else. Константа BUTTON типа int добавлена для контакта кнопки. Кроме того, в функции setup () конфигурируем контакт BUTTON как вход.

Это необязательно, т.к. выводы Arduino являются входами по умолчанию. Функция digitalRead () считывает значение сигнала на входе. Если кнопка нажата, digitalRead() возвращает значение HIGH (лог. 1). Если кнопка не нажата, то получаемLOW (лог. 0).

Проверяем содержимое внутри оператора if() . Если условие внутри оператора if() истинно (кнопка не нажата, digitalRead() ==Low), вызываем функцию digitalWrite (LED, LOW) (гасим светодиод). В противном случае (кнопка нажата) выполняем код после оператора else (включаем светодиод функцией digitalWrite(LED, HIGH)).

Устранение "дребезга" кнопок

Удобно ли держать кнопку постоянно нажатой для свечения светодиода? Гораздо лучше иметь возможность нажать кнопку один раз, чтобы включить светодиод, и нажав ее еще раз, выключить. При таком варианте, для горения светодиода кнопку не придется удерживать нажатой. К сожалению, сделать это не так легко, как кажется.

Нельзя просто считывать значение сигнала на входе, необходимо учитывать явление, называемое дребезгом контактов.

Обычные кнопки представляют собой механические устройства с пружинным контактом.

При нажатии на кнопку сигнал не просто меняется от низкого до высокого, он на протяжении нескольких миллисекунд неоднократно меняет свое значение, прежде чем установится уровень LOW.

Предположение, что состояние кнопки можно определить, считав значение с входа контакта неверно. Кнопка фактически возвращается вверх-вниз, пока значение не установится. Теперь, зная, как ведет себя кнопка, можно написать программу для кнопки с дребезгом, которая фиксирует изменение состояния кнопки,

некоторое время ждет и затем снова читает состояние переключателя.

Алгоритм работы такой программы можно записать следующим образом:

1. Сохраняем предыдущее и текущее состояния кнопки (при инициализации LOW).

2. Считываем текущее состояние кнопки.

3. Если текущее состояние кнопки отличается от предыдущего, ждем 5 мс, потому что кнопка, возможно, изменит свое состояние.

4. Подождав 5 мс, считываем состояние кнопки и делаем его текущим состоянием кнопки.

5. Если предыдущее состояние кнопки было LOW, а текущее - HIGH, переключаем состояние светодиода.

6. Устанавливаем предыдущее состояние кнопки в качестве текущего.

7. Возвращаемся к шагу 2.

Данный алгоритм - прекрасный пример для изучения функций. Функция – это оператор, который может принимать входные аргументы, выполнять фрагмент кода с их использованием и, возможно, возвращать результат. Не зная этого, вы уже встречали функции в программах. Например, digitalWrite() - это функция, которая принимает в качестве аргументов номер контакта и значение (HIGH или LOW), и устанавливает это значение на контакте. Чтобы упростить программу, можно определить свои собственные функции для инкапсуляции действий, которые придется повторять неоднократно.

Процесс выполнения программы представляет собой многократное повторение шагов.

Напишем функцию для устранения дребезга контактов, которую можно вызывать неоднократно. Наша функция будет принимать предыдущее состояние кнопки в качестве входных данных, выполнять противодребезговую защиту и выводить установившееся состояние кнопки. Основной цикл программы переключает состояние светодиода при каждом нажатии кнопки.

```
const int LED=9; // Контакт 9 для подключения светодиода
const int BUTTON=2; // Контакт 2 для подключения кнопки
bool lastButton = LOW; // Переменная для сохранения предыдущего состояния кнопки
bool currentButton = LOW; // Переменная для сохранения текущего состояния кнопки
bool ledOn = false; // Текущее состояние светодиода
//(включен/выключен)
void setup () {
pinMode (LED, OUTPUT); // Сконфигурировать контакт светодиода как выход
pinMode (BUTTON, INPUT); // Сконфигурировать контакт кнопки как вход
}
/*
Функция сглаживания дребезга
принимает в качестве аргумента предыдущее состояние кнопки и выдает фактическое.
*/
bool debounce(bool last)
{
bool current = digitalRead(BUTTON); // Считать состояние кнопки
if (last != current) // Если изменилось ...
{
delay(5);// Ждем 5 мс
current = digitalRead(BUTTON}; // Считываем состояние кнопки
return current;// Возвращаем состояние кнопки
}
}
void loop() {
currentButton = debounce(lastButton);
if (lastButton == LOW && currentButton == HIGH) // Если нажатие
{
```

```
ledOn= !ledOn;// Инвертировать значение
// состояния светодиода
}
lastButton = currentButton;
digitalWrite(LED, ledOn); // Изменить статус
// состояния светодиода
}
```

Теперь рассмотрим текст листинга подробнее. Сначала заданы номера контактов для подключения кнопки и светодиода. Затем объявлены три глобальные логические переменные, которые будут изменяться в программе (значение глобальной переменной можно менять в любой части программы). Каждой из трех переменных присвоены начальные значения (LOW, LOW и false). Далее в программе значения этих переменных могут изменяться с помощью оператора присваивания =.

Рассмотрим функцию подавления дребезга кнопки bool debounce() . Эта функция принимает логическую переменную (имеющую только два состояния: true/false, HIGH/LOW, вкл./выкл., 1/0) предыдущего состояния кнопки и возвращает текущее значение состояния кнопки. Внутри функции текущее состояние кнопки сравнивается с предыдущим с помощью оператора != (не равно). Если состояния отличаются, то кнопка, возможно, нажата. Затем ожидаем 5 мс (этого достаточно, чтобы состояние кнопки стабилизировалось после дребезга), прежде чем проверить состояние кнопки снова. Затем вновь проверяем состояние кнопки. Как вы помните, функции могут возвращать результат. Данная функция возвращает текущее значение булевой локальной переменной, которая объявлена и используется только в функции debounce(). Когда функция debounce() вызывается из основного цикла, возвращенное значение записывается в глобальную переменную currentButton, которая была определена в начале программы.

После вызова функции debounce() и установки значения переменной currentButton происходит сравнение текущего и предыдущего значений состояния кнопки с помощью оператора &&

(логический оператор "И", означающий, что выражение в скобках выполнится, только если истинно каждое из равенств, разделенных оператором &&).

Если ранее состояние кнопки было LOW, а теперь HIGH, значит, кнопка была нажата и нужно инвертировать значение переменной ledOn. Это действие выполняет оператор ! перед переменной ledOn. Цикл закончен, обновляем предыдущую переменную состояния кнопки и изменяем состояние светодиода.

Программа изменяет состояние светодиода после каждого нажатия кнопки. При отсутствии проверки дребезга кнопки результаты будут непредсказуемыми.

Задания на лабораторную работу

1. Собрать схему и написать программу обработки нажатия кнопки, чтобы на входном контакте было значение HIGH при отпущенной кнопке и значение LOW, когда кнопка нажата.

2. Собрать схему и написать программу циклического переключения 8 цветовых комбинаций RGB-светодиода с режимом включения/выключения режима мерцания.

ЛАБОРАТОРНАЯ РАБОТА № 3. ОПРОС АНАЛОГОВЫХ ДАТЧИКОВ

Цель работы: научиться считывать значения на аналоговых входах платы Arduino.

Преобразовать аналоговые значения напряжения в числа, которые может обрабатывать контроллер, позволяет аналого-цифровой преобразователь Arduino. На плате Arduino Uno установлен10-разрядный АЦП.

Опорное напряжение определяет максимальное напряжение на входе АЦП, его значение соответствует коду 1023. При нулевом входном напряжении АЦП выдает на выходе 0, при входном напряжении 2,5 В на выходе будет значение 512 (половина от 1023), при входном напряжении 5 В выходной код равен 1023.

Чтение данных с потенциометра

Самый простой аналоговый датчик, с которого можно получить аналоговый сигнал, это потенциометр. Их используют в стереосистемах, звуковых колонках, термостатах и в других изделиях. Потенциометры действуют как регулируемые делители напряжения и снабжены ручкой-регулятором. Они бывают разных размеров и форм, но всегда имеют три вывода.

Подключите один крайний вывод потенциометра к земле, а другой к шине 5 В. Потенциометры симметричны, так что не имеет значения, с какой стороны вы подключите шину питания, а с какой землю. Средний вывод соедините с аналоговым контактом 0 на плате Arduino. Как правильно подключить потенциометр к Arduino, показано на рис. 3.1. При повороте ручки потенциометра аналоговый входной сигнал будет плавно меняться от 0 до 5 В.

Прежде чем использовать потенциометр для управления другим оборудованием, посмотрим, как считать значение сопротивления потенциометра с помощью АЦП и передать через последовательный порт Arduino для просмотра значений на компьютере. Для чтения значения аналогового входа предусмотрена функция analogRead(), для вывода значений в последовательный порт Arduino IDE – функция serial.println().

```
// Программа чтения данных с потенциометра
const int POT = 0; // Аналоговый вход для подключения потенциометра
int val = 0;
void setup() {
// Переменная для хранения значения потенциометра
Serial.begin(9600);
}
void loop() {
val = analogRead(POT);
Serial.println(val);
delay(500);
}
```

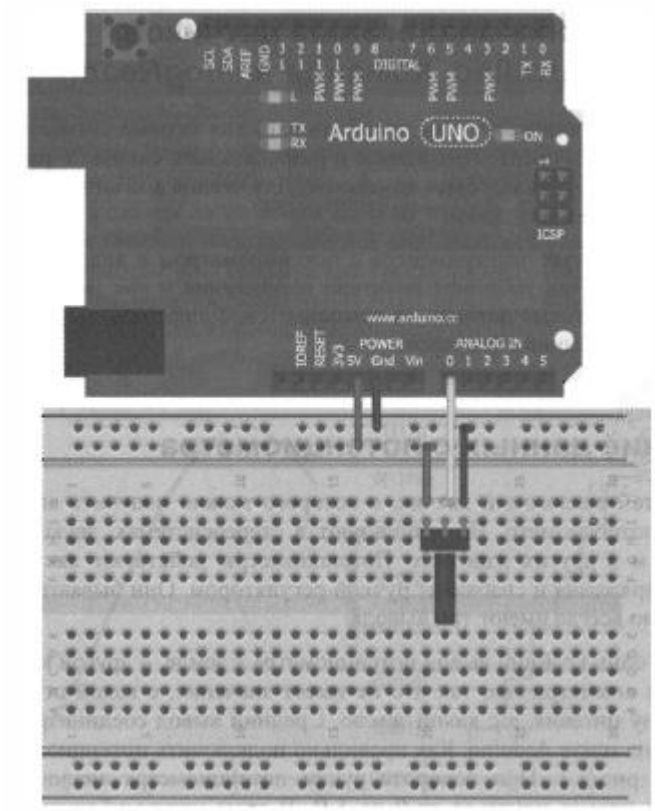

Рис. 3.1. Подключение потенциометра

Сначала необходимо инициировать последовательное соединение, вызвав функцию Serial.begin(), единственный аргумент которой задает скорость передачи данных в бодах. Скорость передачи данных определяет количество битов, передаваемых в секунду. Высокая скорость передачи позволяет передавать больше данных за меньшее время, но может привести к ошибкам в некоторых системах связи. В наших примерах выбрана скорость 9600 бод.

В каждой итерации цикла переменная val получает аналоговое значение, считанное командой analogRead() с входа, соединенного со средним контактом потенциометра (в нашем случае

это вход A0). Далее это значение функция serial.println() выводит в последовательный порт, соединенный с компьютером. Затем следует задержка в полсекунды (чтобы числа выводились не быстрее, чем вы можете их прочитать).

После запуска монитора последовательного порта на экране компьютера появляется окно с отображением потока передаваемых чисел. Поверните ручку потенциометра, и вы увидите, что выводимые значения меняются. Если повернуть ручку в одном направлении, числа начинают приближаться к 0, если в другом - к 1023.

Задание на лабораторную работу

Создать систему, сигнализирующую об изменении температуры с использованием аналогового датчика TMP36. RGB-светодиод должен гореть зеленым, когда температура находится в пределах допустимого диапазона, красным, когда станет жарко, и синим, когда становится холодно. Пределы допустимого диапазона определить самостоятельно.

ЛАБОРАТОРНАЯ РАБОТА № 4. ИСПОЛЬЗОВАНИЕ ПЕРЕМЕННЫХ РЕЗИСТОРОВ ДЛЯ СОЗДАНИЯ АНАЛОГОВЫХ ДАТЧИКОВ. УПРАВЛЕНИЕ АНАЛОГОВЫМИ ВЫХОДАМИ ПО СИГНАЛУ ОТ АНАЛОГОВЫХ ВХОДОВ

Цель работы: научиться управлять аналоговыми выходами по сигналу от аналоговых входов.

Резистивный делитель напряжения

Благодаря достижениям в области физики, мы имеем множество материалов, способных изменять сопротивление в результате физического воздействия. Например, проводящие краски изменяют свое сопротивление при изгибе и скручивании, полупроводники меняют сопротивление под действием света (фоторезисторы), сопротивление некоторых материалов зависит от нагрева и охлаждения (термисторы).

Это всего лишь несколько примеров, которые позволят создать свои собственные аналоговые датчики.

Поскольку упомянутые датчики меняют сопротивление, а не напряжение, в схеме потребуется создать делитель напряжения, чтобы можно было измерить изменение сопротивления.

Резистивный делитель напряжения состоит из двух резисторов, от соотношения сопротивлений которых зависит выходное напряжение. Так, если один из резисторов переменный, то на выходе можно получить изменение напряжения. Другой резистор определяет чувствительность схемы, если это подстроечный резистор, то чувствительность можно корректировать.

Рассмотрим нерегулируемый резистивный делитель (рис. 4.1) и напряжение на его выходе. Обозначение A0 на рис. 4.1 - это аналоговый вход A0 на плате Arduino.

Зависимость выходного напряжения делителя от входного:

$U_{вых} = U_{вх} (R2/(Rl + R2))$.

В нашем случае на вход делителя подано напряжение 5 В, а выход подключен к аналоговому контакту AO платы Arduino. Если Rl и R2 одинаковы (как, например, 10 кОм), то 5 В делится пополам, и на аналоговом входе будет 2,5 В. Проверьте это, подставив значения в формулу:

$U_{вых} = 5$ В (10 кОм/(10 кОм+ 10 кОм)) = 2,5 В.

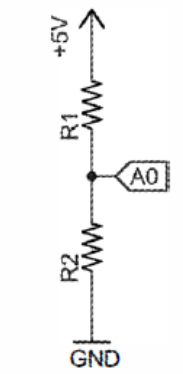

Рис. 4.1. Простой делитель напряжения

Теперь предположим, что один из этих резисторов переменный, например фоторезистор. Сопротивление фоторезистора зависит от интенсивности падающего на него света. Если использовать фоторезистор с номинальным сопротивлением 200 кОм, то в полной темноте его сопротивление около 200 кОм, при ярком свете оно падает почти до нуля. От того, какой резистор (Rl или R2) поменять на фоторезистор, и от номинала постоянного резистора будет зависеть масштаб и точность показаний.

В качестве примера заменим R1 на фоторезистор, а R2 возьмем постоянным с номиналом 10 кОм.

Загрузите программу считывания аналоговых данных и выдачи результата в последовательный порт и поменяйте освещенность фоторезистора. Вы не сможете получить весь диапазон значений от 0 до 1023, потому что у фоторезистора никогда не будет нулевого сопротивления. В результате вы определите минимальное и максимальное значения напряжения на выходе. Эти данные потребуются, чтобы сделать "интеллектуальный" ночник, который будет светить более ярко в темном помещении, и наоборот. Выберите аналоговые значения освещенности для помещения, соответствующие темноте и максимальной освещенности. Например, значения 200 (темнота) и 900 (максимальное освещение). Они зависят от условий освещения, значения резистора R2 и характеристик фоторезистора.

Управление аналоговыми выходами по сигналу от аналоговых входов

Функция analogWrite() позволяет изменять яркость светодиода. Но аргумент этой функции 8-разрядный, т. е. находится в диапазоне от 0 до 255, в то время как АЦП выдает значения от 0 до 1023. В языке программирования Arduino есть удобные функции для пропорционального преобразования значений от одного диапазона к другому: map() и constrain(). Синтаксис функции map() выглядит следующим образом:

output = map(value, fromLow, fromHigh, toLow, toHigh).

Здесь value - преобразуемое значение (напряжение на аналоговом входе). fromLow и fromHigh - это нижняя и верхняя границы текущего диапазона. В нашем примере это минимальная и максимальная освещенность в помещении (200 и 900). toLow и toHigh - нижняя и верхняя границы нового диапазона. Аргумент функции analogWrite() должен быть в диапазоне от 0 до 255. Но мы хотим меньшей освещенности сопоставить большую яркость светодиода, т. е. минимальным значениям на аналоговом входе должны соответствовать максимальные значения на выводах светодиода. Удобно, что функция map() делает это автоматически. Функция map() осуществляет линейное отображение. Например, если fromLow и fromHigh равны 200 и 900, соответственно, а toLow и toHigh равны 255 и 0, то 550 превратится в 127, потому что 550 находится посередине между 200 и 900, а 127 посередине между 255 и 0. Следует учесть, что функция map() не ограничивает значения, если они выходят за границы диапазона. Если value окажется меньше 200 (для нашего примера), то output будет больше 255. Это неудобно, т. к. передать функции analogWrite() значение, превышающее 255, нельзя. Для ограничения значений есть функция constrain(), синтаксис которой выглядит следующим образом:

output = constrain(value, min, max).

При передаче значения из функции map() в функцию constrain() можно установить аргумент min равным 0 и max - 255, тогда величины, выходящие за рамки этого диапазона, будут ограничены.

Задание на лабораторную работу

Создать систему, регулирующую яркость светодиода в зависимости от освещенности помещения (меньшей освещенности сопоставить большую яркость светодиода).

ЛАБОРАТОРНАЯ РАБОТА № 5. ИСПОЛЬЗОВАНИЕ ТРАНЗИСТОРОВ ДЛЯ УПРАВЛЕНИЯ ДВИГАТЕЛЯМИ ПОСТОЯННОГО ТОКА

Цель работы: научиться управлять аналоговыми выходами по сигналу от аналоговых входов.

Двигатели постоянного тока

Вал двигателя постоянного тока вращается при подаче постоянного напряжения на его контакты. Подобные двигатели можно встретить во многих бытовых приборах, например, в радиоуправляемых автомобилях, в приводе DVD-плеера. Регулируя напряжение, подаваемое на двигатель, можно менять скорость его вращения. Переключая полярность приложенного напряжения, можно изменять направление вращения.

Щеточные двигатели постоянного тока состоят из неподвижных магнитов (статора)и вращающейся обмотки (ротора). Электроэнергию подводят через контакты "щетки",поэтому двигатели называются щеточными. В отличие от электродвигателей постоянного тока других типов (таких, например, как шаговые двигатели), щеточные электродвигатели дешевле и скорость вращения легко регулировать. Однако их срок службы невелик, потому что щетки со временем изнашиваются.

Борьба с выбросами напряжения

Двигатели постоянного тока обычно требуют ток больше, чем может выдать встроенный в Arduino блок питания, к тому же они могут создавать опасные выбросы напряжения. Для решения этой проблемы необходимо научиться эффективно изолировать двигатель постоянного тока от платы Arduino и подключать его к отдельному источнику питания. Транзистор позволит безопасно

включать двигатель, а также управлять его скоростью с помощью методов ШИМ. Схема подключения двигателя постоянного тока приведена на рис. 5.1. Рассмотрим основные компоненты схемы:

Q1 - n-p-n биполярный плоскостной транзистор действует как ключ, включая и выключая внешний источник питания 9 В. Говоря упрощенно, n-p-n транзистор представляет собой переключатель, управляемый напряжением, что позволяет подавать или отключать ток;

Rl - резистор номиналом 1 кОм, соединяющий контакт платы Arduino с базой транзистора;

U1 - двигатель постоянного тока;

C1 - конденсатор для фильтрации помех, вызванных работой двигателя;

Dl - диод для защиты блока питания от обратного напряжения.

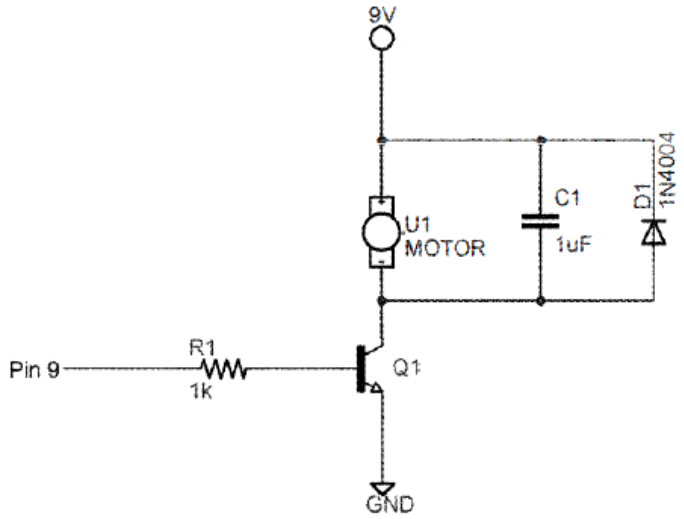

Рис. 5.1. Схема включения двигателя постоянного тока

Использование транзистора в качестве переключателя

Транзисторы применяются во многих устройствах: от усилителей до компонентов центрального процессора в компьютерах и смартфонах. У нас транзистор будет работать в качестве простого электрически управляемого переключателя. Каждый биполярный транзистор имеет три контакта: эмиттер (E), коллектор (C) и базу (B).

Между коллектором и эмиттером течет большой ток, величина которого зависит от малого тока базы. Изменяя ток базы, мы можем регулировать ток через транзистор и менять скорость вращения двигателя. Напряжения 5 В, подаваемого на выход Arduino, достаточно для включения транзистора. Используя ШИМ, можно управлять скоростью вращения двигателя. Поскольку механические детали двигателя обладают инерцией, быстрое переключение транзистора под действием ШИМ сигнала с разной скважностью приведет к плавной регулировке скорости вращения.

Назначение защитных диодов

Одна из проблем электродвигателей постоянного тока - наличие противо-ЭДС. В двигателе есть обмотки, в которых создается магнитный поток. При работе двигателя энергия магнитного поля запасается в обмотках. При выключении электродвигателя на концах обмотки возникает выброс напряжения обратной полярности, опасный для источника питания. Предотвратить воздействие электрических выбросов на внешние цепи можно с помощью защитных диодов. Подключив защитный диод, можно быть уверенным, что он устранит выброс напряжения при выключении двигателя.

Назначение отдельного источника питания

В схеме, изображенной на рис. 5.1, двигатель подключен к отдельному источнику напряжением 9 В, а не к контакту 5 В разъема USB. Для данного примера подойдет также внешний источник с напряжением 5 В. Внешний источник питания необходим по двум причинам:

- уменьшается вероятность повреждения платы Arduino при неправильном подключении электродвигателя;

- ток и напряжение могут быть больше, чем обеспечивает встроенный в Arduino источник питания.

Некоторые двигатели постоянного тока потребляют ток, больший, чем может выдать плата Arduino. Кроме того, рабочее напряжение многих двигателей превышает 5 В. Хотя они и будут вращаться при напряжении 5-вольтовом питании, но достичь заданной скорости вращения могут только при питании 9 или 12 В (в зависимости от технических характеристик двигателя).

Обратите внимание, что необходимо соединить землю отдельного источника питания с землей Arduino. Это обеспечит общую точку между уровнями напряжения в двух частях схемы.

Подключение двигателя

Теперь, когда мы рассмотрели все тонкости управления щеточным двигателем постоянного тока, установим его макетную плату и подключим. Соберите схему, изображенную на рис. 5.1, а затем проверьте правильность монтажа по рис. 5.2. Важно научиться хорошо читать электрические схемы без использования графического макета.

Перед включением питания проверьте следующее:

- убедитесь, что земля от батареи 9 В соединена с землей платы Arduino, для этого можно общую шину земли на макетной плате, как показано на рис. 5.2;

- убедитесь, что провод питания +9 В не подключен к проводу питания +5 В;

- убедитесь, что транзистор подключен правильно;

- убедитесь, что диод включен правильно.

Перед написанием программы необходимо проверить работу схемы. Подсоединим батарею, подадим питание на плату Arduino через USB-кабель, подключим базу транзистора к выводу+5 В, это имитирует высокий логический уровень на выходном контакте Arduino.

Вал двигателя должен начать вращаться. При подключении базы транзистора к земле двигатель останавливается. Если это не

так, проверьте правильность монтажа. Если все работает как описано, переходим к следующему шагу - программированию.

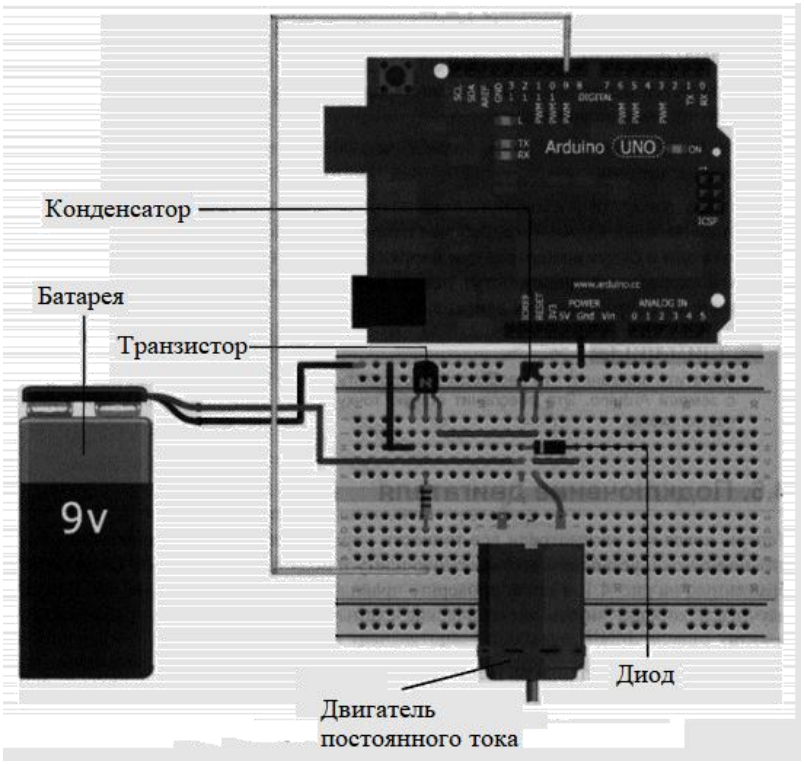

Рис. 5.2. Подключение двигателя постоянного тока

Управление скоростью вращения двигателя с помощью ШИМ

Появление на выходе платы Arduino ШИМ-сигнала вызывает быстрый запуск и остановку двигателя с разным периодом, что эквивалентно изменению скорости вращения.

// Пример управления скоростью вращения двигателя

```
const int MOTOR=9; // Вывод 9 Arduino для подключения двигателя
void setup () {
pinMode (MOTOR, OUTPUT);
}
void loop() {
for (inti=0; i<256; i++) {
analogWrite(MOTOR, i);
delay(10);
}
delay(2000);
for (inti=255; i>=0; i--) {
analogWrite(MOTOR, i);
delay (10);
}
delay(2000);
}
```

Если все собрано правильно, то после запуска программы скорость вращения двигателя сначала плавно увеличится, а затем уменьшится. На основе описанного проекта можно сконструировать, например, макет движущегося робота.

Задание на лабораторную работу

Создать систему, регулирующую скорость вращения вала двигателя с помощью потенциометра.

ЛАБОРАТОРНАЯ РАБОТА № 6. УПРАВЛЕНИЕ НАПРАВЛЕНИЕМ ВРАЩЕНИЯ ДВИГАТЕЛЯ ПОСТОЯННОГО ТОКА С ПОМОЩЬЮ Н-МОСТА

Цель работы: научиться управлять направлением вращения двигателя постоянного тока с помощью Н-моста.

Для изменения направления вращения двигателя применим устройство, называемое Н-мостом. Принцип работы Н-моста поясняет схема, изображенная на рис. 6.1.

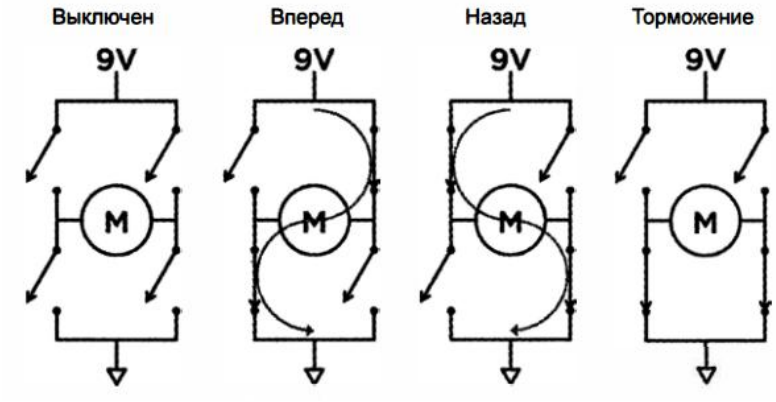

Рис. 6.1. Схема работы Н-моста

В реальной схеме Н-моста в качестве изображенных на схеме выключателей используются транзисторы, также есть некоторые дополнительные цепи, в том числе защитные диоды.

Н-мост может находиться в четырех основных состояниях: "выключен", "торможение", "вперед" и "назад". В выключенном состоянии все выключатели разомкнуты и двигатель не вращается. В состоянии "вперед" два выключателя замкнуты, в результате через обмотку двигателя течет ток и вал вращается. В состоянии "назад"ток течет в противоположном направлении, и направление вращения вала обратное. Если Н-мост находится в состоянии торможения, то обмотка замкнута, вращение замедляется и двигатель останавливается.

Необходимо помнить об опасности короткого замыкания цепей Н-моста. Что произойдет, если все четыре выключателя будут замкнуты? Это вызовет короткое замыкание между шиной +9 В и землей. В результате батарея очень быстро нагреется, что может привести к ее разрыву. Кроме того, короткое замыкание может повредить Н-мост или другие элементы схемы.

Для нашего эксперимента выберем микросхему SN754410 – четырехканальный драйвер Н-полумоста, которая имеет встроенную защиту от короткого замыкания (или один из ее аналогов: L293, L293D, L293DNE). Два драйвера Н-полумоста образуют полный драйвер Н-моста. Для управления одним электродвигателем постоянного тока используются два из четырех драйверов Н-полумоста. Если вы хотите сделать, например, радиоуправляемый автомобиль, можно управлять двумя колесами с помощью одной микросхемы SN754410. На рис. 6.2 приведена цоколевка микросхемы SN754410.

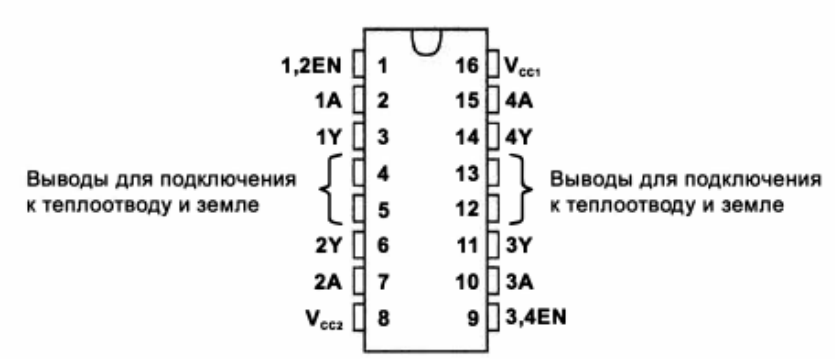

Рис. 6.2. Цоколевка микросхемы SN754410

Нумерация контактов на микросхемах начинается с левого верхнего угла и идет против часовой стрелки. На корпусе всегда имеется метка у контакта 1 (полукруг, точка или что-то другое).

Соответствие состояний входов и выходов драйвера SN754410 иллюстрирует табл. 6.1 (условные обозначения в таблице: H - высокий уровень; L - низкий уровень; X - безразличное состояние; Z -высокоимпедансное состояние).

Таблица 6.1.

Состояния входов и выходов драйвера SN754410

Вход		Выход
A	EN	Y
H	H	H
L	H	L
X	L	Z

Рассмотрим назначение контактов микросхемы SN754410:

GND (контакты 4, 5, 12, 13) - выводы для подключения к земляной шине монтажной платы;

Vcc2 (контакт 8) - напряжение питания двигателя (подсоедините к 9 В);

Vcc1 (контакт 16) - напряжение питания микросхемы (подсоедините к 5 В);

1Y и 2Y (контакты 3 и 6) - выходы для подключения первого двигателя;

1A и 2A (контакты 2 и 7) - коммутация первого двигателя, эти выводы соединены с управляющими контактами Arduino;

1,2 EN (контакт 1) - включение и отключение левого драйвера. Данный вывод соединен с ШИМ-контактами на плате Arduino, что позволяет динамически регулировать скорость двигателей;

3Y и 4Y (контакты 11 и 14) - выходы для подключения второго двигателя;

3A и 4A (контакты 10 и 15) - коммутация второго двигателя, эти выводы соединены с управляющими контактами Arduino;

3,4 EN (контакт 9) - вывод включения или отключения правого драйвера. Он соединен с ШИМ-контактами на Arduino, что позволяет динамически регулировать скорость двигателей.

Проверьте монтаж по рис. 6.3. Прежде чем приступать к программированию, проверим работоспособность схемы. Подключите один из входных контактов (2 или 7) микросхемы H-моста к шине 5 В, другой к земле. Двигатель начнет вращаться. Поменяйте подключение контактов 2 и 7, двигатель будет вращаться в другую сторону.

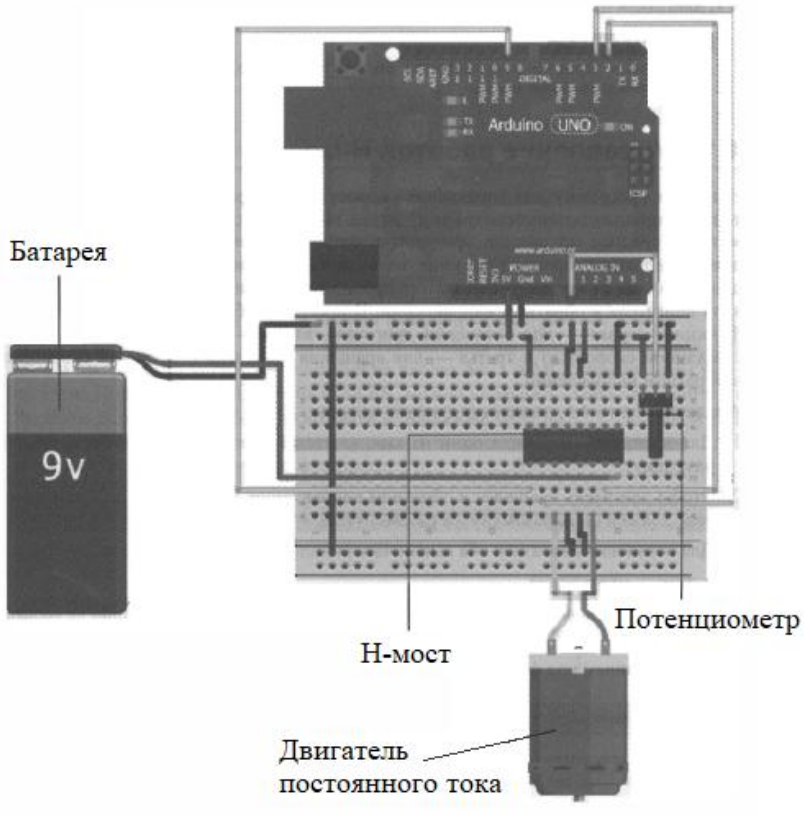

Рис. 6.3. Схема подключения Н-моста

Во время переключения контактов необходимо отключать батарею, чтобы случайно не вызвать короткое замыкание моста.

Управление работой Н-моста

Напишем программу для управления скоростью и направлением вращения двигателя с помощью потенциометра и драйвера Н-моста. Установка движка потенциометра в среднее положение приводит к остановке двигателя, при перемещении движка вправо скорость вращения вала двигателя увеличивается, перемещение движка влево от среднего положения приводит к

увеличению скорости вращения вала двигателя в обратном направлении. В программе будут три вспомогательные функции: первая - для остановки двигателя, вторая - для вращения двигателя с заданной скоростью и третья - для вращения двигателя с заданной скоростью в обратном направлении.

Анализируя рис. 6.1, делаем следующие выводы:

1. Для вращения двигателя один из выключателей должен быть замкнут, другой разомкнут.

2. Чтобы двигатель вращался в обратном направлении, замкнутый в п. 1 выключатель должен быть разомкнут, а разомкнутый - замкнут.

3. Для остановки двигателя оба выключателя должны быть разомкнуты.

ПРИМЕЧАНИЕ

Перед изменением состояния выключателей всегда отключайте ток, чтобы не вызвать короткого замыкания Н-моста.

Сначала напишем код функций для выполнения описанных действий.

```
// Движение двигателя вперед с заданной скоростью
// (диапазон 0-255)
void forward (int rate) {
digitalWrite(EN, LOW);
digitalWrite(MC1, HIGH);
digitalWrite(MC2, LOW);
analogWrite(EN, rate);
}
// Движение двигателя в обратном направлении с заданной скоростью
// (диапазон 0-255)
void reverse (int rate) {
digitalWrite(EN, LOW);
digitalWrite(MC1, LOW);
digitalWrite(MC2, HIGH);
analogWrite(EN, rate);
}
```

```
// Остановка двигателя
void brake() {
digitalWrite(EN, LOW);
digitalWrite(MC1, LOW);
digitalWrite(MC2, LOW);
digitalWrite(EN, HIGH);
}
```

Обратите внимание, что в начале каждой функции на контакте EN всегда устанавливается низкий уровень, и затем задаются значения на входах блока управления MC1 и MC2. После установки значений на входах MCl и MC2 можно снова включить ток. Подавая сигнал ШИМ на вход EN, можно управлять скоростью двигателя.

Значение переменной rate должно быть в диапазоне от 0 до 255. Основной цикл программы считывает данные с потенциометра и в зависимости от результата вызывает требуемую функцию.

```
void loop() {
val = analogRead(POT);
// Движение вперед
if (val> 562) {
velocity = map(val, 563, 1023, 0, 255);
forward(velocity);
}
// Движение назад
else if (val< 462) {
velocity = map(val, 461, 0, 0, 255);
reverse(velocity);
}
// Остановка
else {
brake();
}
}
```

Сигнал с аналогового входа преобразуется в цифровое значение в диапазоне от 0 до 1023. При значениях сигнала от потенциометра в диапазоне от 462 до 562 (100 отсчетов в районе средней точки) вызывается функция brake() для остановки двигателя, при значениях от 562 до 1023 - функция запуска двигателя в прямом направлении forward(), при значениях от 0 до 462 - функция запуска двигателя в обратном направлении reverse(). При определении обратной скорости значению потенциометра 461 соответствует значение скорости 0, а значению потенциометра 0 соответствует значение скорости 255. Функция map() инвертирует значения так, что на вход они подаются в обратном порядке. Объединив цикл loop() со вспомогательными функциями и начальной установкой setup (), получим полный код программы управления скоростью и направлением движения двигателя с помощью потенциометра.

```
// Управление двигателем с помощью Н-моста
const int  EN=9; // Вход включения двигателя EN
const int MC1=3;
const int MC2=2;
const int POT=0;

intval=0; // Переменная для хранения значения потенциометра
intvelocity=0; //Переменная для хранения скорости двигателя (0-255)
void setup () {
pinMode(EN, OUTPUT);
pinMode(MC1, OUTPUT);
pinMode(MC2, OUTPUT);
brake(); // Остановка двигателя при инициализации
}
void loop() {
val = analogRead(POT);
// Движение вперед
if (val> 562) {
```

```
velocity = map(val, 563, 1023, 0, 255);
forward(velocity);
}
// Движение назад
else if (val< 462) {
velocity = map(val, 461, 0, 0, 255);
reverse(velocity);
}
// Остановка
else {
brake();
}
}
// Движение двигателя вперед с заданной скоростью
// (диапазон 0-255)
void forward (int rate) {
digitalWrite(EN, LOW);
digitalWrite(MC1, HIGH);
digitalWrite(MC2, LOW);
analogWrite(EN, rate);
}
// Движение двигателя в обратном направлении с
заданной скоростью
// (диапазон 0-255)
void reverse (int rate) {
digitalWrite(EN, LOW);
digitalWrite(MC1, LOW);
digitalWrite(MC2, HIGH);
analogWrite(EN, rate);
}
// Остановка двигателя
void brake() {
digitalWrite(EN, LOW);
digitalWrite(MC1, LOW);
digitalWrite(MC2, LOW);
digitalWrite(EN, HIGH);
}
```

Задание на лабораторную работу

Подключите к драйверу Н-моста второй двигатель постоянного тока и напишите программу управления двумя двигателями.

РЕКОМЕНДАТЕЛЬНЫЙ БИБЛИОГРАФИЧЕСКИЙ СПИСОК

а) основная литература

1. *Соммер У.* Программирование микроконтроллерных плат Arduino/Freeduino. – СПб.: БХВ-Петербург, 2012. – 256 с.
2. *Воронцова Е.А.* Программирование на С++ с погружением: практические задания и примеры кода - М.:НИЦ ИНФРА-М, 2016. - 80 с.
3. Теория алгоритмов: учебное пособие / В.И. Игошин. - М.: ИНФРА-М, 2012. - 318 с.

б) дополнительная литература

4. Полезное программирование: Практическое пособие / Комлев Н.Ю. - М.: СОЛОН-Пр., 2016. - 256 с.
5. Программирование графики на С++. Теория и примеры : учеб. пособие / В.И. Корнеев, Л.Г. Гагарина, М.В. Корнеева. — М. : ИД «ФОРУМ» : ИНФРА-М, 2017. - 517 с.
6. *Ступина, А.А.* Технология надежностного программирования задач автоматизации управления в технических системах: монография / А.А. Ступина, С.Н. Ежеманская. - Красноярск : Сиб. федер. ун-т, 2011. - 164 с.

Содержание

ПРИКЛАДНОЕ ПРОГРАММИРОВАНИЕ

Методические указания к лабораторным работам для студентов бакалавриата направления 27.03.04

Сост.: *Ю.В. Ильюшин, Т.В. Кухарова*

Печатается с оригинал-макета, подготовленного кафедрой системного анализа и управления

Ответственный за выпуск *Ю.В. Ильюшин*

Лицензия ИД № 06517 от 09.01.2002

Подписано к печати 02.02.2023. Формат 60×84/16.
Усл. печ. л. 2,3. Усл.кр.-отт. 2,3. Уч.-изд.л. 2,0. Тираж 50 экз. Заказ 48.

Санкт-Петербургский горный университет
РИЦ Санкт-Петербургского горного университета
Адрес университета и РИЦ: 199106 Санкт-Петербург, 21-я линия, 2