Практическая работа 3

Проект светильник с управляемой яркостью

В этом проекте меняем яркость светодиода, вращая ручку переменного резистора.

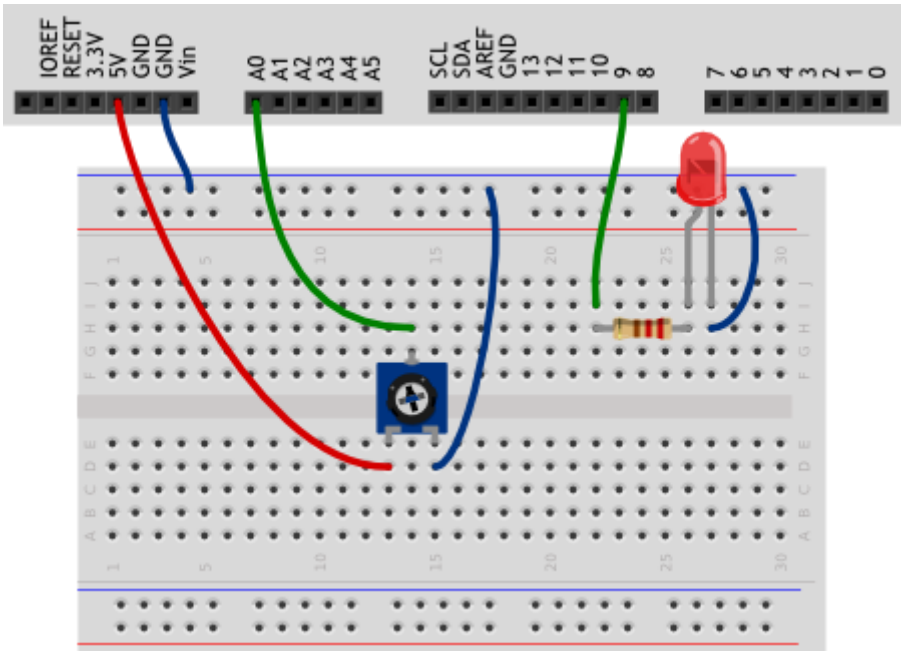
Задание 1. Ответить на вопросы

1. Что такое потенциометр?
2. Чем потенциометр отличается от триммера и от переменного резистора?
3. Как на схеме обозначить потенциометр?
4. Какому напряжению соответствует значение 0, 128, 256, 512, 1023 на порту потенциометра?

Задание 2. Список деталей для эксперимента

1. 1 плата Arduino Uno
2. 1 беспаечная макетная плата
3. 1 светодиод
4. 1 резистор номиналом 220 Ом
5. 6 проводов «папа-папа»
6. Потенциометр

Схема на макетной плате:



1. Зарисуйте принципиальную схему установки.

Обратите внимание

✓ Не забудьте, как соединены рельсы в беспаечной макетной плате. Если на вашей макетке красная и синяя линии вдоль длинных рельс прерываются в середине, значит проводник внутри макетки тоже прерывается!

✓ Катод («минус») светодиода — короткая ножка, именно её нужно соединять с землёй (GND)

✓ Не пренебрегайте резистором, иначе светодиод выйдет из строя

✓ Мы подключили «землю» светодиода и переменного резистора (потенциометра) к длинной рельсе «-» макетной платы, и уже ее соединили с входом GND микроконтроллера. Таким образом мы использовали меньше входов и от макетки к контроллеру тянется меньше проводов.

✓ Подписи «+» и «-» на макетке не обязывают вас использовать их строго для питания, просто чаще всего они используются именно так и маркировка нам помогает

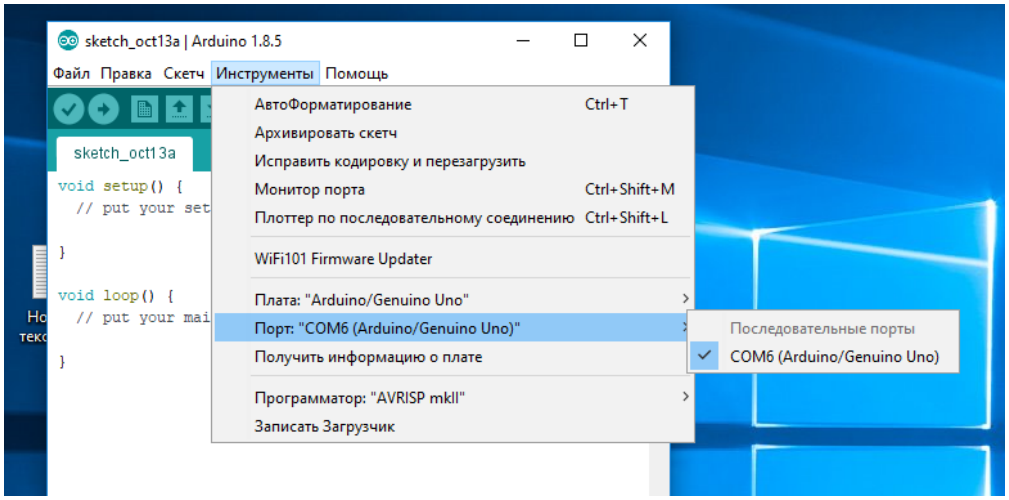
✓ Не важно, какая из крайних ножек потенциометра будет подключена к 5 В, а какая к GND, поменяется только направление, в котором нужно крутить ручку для увеличения напряжения. Запомните, что сигнал мы считываем со средней ножки

✓ Для считывания аналогового сигнала, принимающего широкий спектр значений, а не просто 0 или 1, как цифровой, подходят только порты, помеченные на плате как «ANALOG IN» и пронумерованные с префиксом A. Для Arduino Uno — это A0-A5.

Задание 3. Программирование микроконтроллера

1. Запустите приложение

2. Убедитесь, что выбран нужный порт



Переменные

Макроопределения хороши для именования значений, которые не могут измениться по ходу выполнения программы. Мы вряд ли захотим на лету, без изменения кода и перезагрузки Arduino, перенести светодиод маячка с одного пина Arduino на другой. Но что делать, если какие-то параметры программы всё же должны изменяться с течением времени?

Для этого существуют **переменные** — именованные значения, которые могут изменяться при исполнении программы процессором. Как и когда они должны изменяться зависит от вас, от того какой алгоритм вы задумали и как написали программу для этого.

Например, давайте рассмотрим программу «помирающего маячка». Первый раз он мигает через 1000 мс, затем через 1100 мс, затем через 1200 мс и так далее до бесконечности:

```
#define LED_PIN 13
int blinkDelay = 900;

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

void loop()
{
    digitalWrite(LED_PIN, HIGH);
    delay(100);

    digitalWrite(LED_PIN, LOW);
    delay(blinkDelay);

    blinkDelay += 100;
}
```

Вместо того, чтобы обозначить время до следующего подмигивания через макроопределение #define, мы использовали конструкцию:

```
int blinkDelay = 900;
```

Это называется определением переменной. Мы таким образом заявили, что хотим иметь ячейку памяти, к которой будем обращаться по имени blinkDelay и изначально, при старте Arduino, в ней должно лежать значение 900.

Перед именем переменной в определении указывается тип данных для этой переменной. В нашем случае — это int, что означает «целое число» (int — сокращение от английского «integer»: целочисленный).

Обратите внимание: объявление переменной, в отличие от макроопределения — это обычное выражение, поэтому оно должно завершаться точкой с запятой.

Итак, мы сообщили компилятору, что у нас есть переменная целочисленного типа, с именем blinkDelayи начальным значением 900. Теперь мы можем ей пользоваться.

Пользоваться переменной можно в двух смыслах: получать её значение и изменять её значение. Получение значения выглядит ровно так же, как и использование макроопределений: мы просто используем её имя в коде, а в результате, при исполнении программы в действительности используется её значение:

```
delay(blinkDelay);
```

В нашей программе это выражение означает «уснуть на столько миллисекунд, сколько сейчас записано в переменной с именем blinkDelay». При первом исполнении этого выражения значение будет изначальным, таким образом мы уснём на 900 мс.

Далее в нашем скетче мы можем видеть:

blinkDelay += 100;

Это так называемое арифметическое выражение (англ. expression). Символ += называется оператором и означает в C++ «увеличить на». Таким образом после исполнения этого выражения, значение переменной blinkDelay станет на сотню больше и сохранится в ней до следующего изменения.

Как результат:

- ✓ При первом исполнении функции `loop` мы выжидаем с выключенным светодиодом 900 мс
- ✓ К началу второго исполнения `blinkDelay` уже равна `1000`, поэтому мы будем выжидать 1000 мс
- ✓ При третьем исполнении мы будем выжидать 1100 мс
- ✓ …и так далее до бесконечности

В итоге мы получаем что и хотели: «помирающий» маячок.

Об именах переменных

Как и в случае с макроопределениями существует общепринятая конвенция о том, как нужно называть переменные. Их принято именовать в так называемом «верблюжьем стиле» (camelCase). То есть, начинать строчными буквами, а каждое новое слово писать слитно, с заглавной буквы.

```
// плохо
int BLINK_DELAY;
int BlinkDelay;
int blink_delay;
```

```
// хорошо

int blinkDelay;
```

Также отличительным признаком профессионализма является использование понятных, лаконичных имён из которых чётко понятно зачем нужна конкретная переменная в программе.

Ничто так не выносит мозг, как использование одно- или двухбуквенных имён без смысла или применение транслита.

```
// плохо

int d;

int asd;

int zaderzhkaMigalki;



// хорошо

int blinkDelay;

int offDelay;

int blinkInterval;

int blinkTimeout;
```

Составление арифметических выражений

В нашем примере мы использовали оператор += для увеличения целочисленного значения нашей переменной.

Конечно же, это не единственный оператор в C++. Кроме того += — это так называемый синтаксический сахар (syntax sugar) — удобная и короткая запись полного выражения. На самом деле выражение:

```
blinkDelay += 100;
```

эквивалентно такой, полной записи:

```
blinkDelay = blinkDelay + 100;
```

Не стоит воспринимать символ = дословно, как «равно». В C++ этот символ называется оператором присваивания или просто присваиванием. Нужно читать это выражение так: присвоить переменной blinkDelay (то, что слева от =) вычисленное значение blinkDelay + 100 (то, что справа от =).

Вас не должно смущать, что blinkDelay упоминается и в левой и в правой части выражения. Опять же, это не означает «900 = 1000», это означает «записать в переменную blinkDelay новое значение: результат сложения текущего значения blinkDelay и числа 100.

3. Наберите в редакторе кода следующий код программы:

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

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

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

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

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

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

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

✓ С помощью директивы #define мы сказали компилятору заменять идентификатор POT_PIN на A0 — номер аналогового входа. Вы можете встретить код, где обращение к аналоговому порту будет по номеру без

индекса A. Такой код будет работать, но во избежание путаницы с цифровыми портами используйте индекс.

- ✓ Переменным принято давать названия, начинающиеся со строчной буквы.
- ✓ Чтобы использовать переменную, необходимо ее объявить, что мы и делаем инструкцией:
- ✓ int rotation, brightness;
- ✓ Для объявления переменной необходимо указать ее тип, здесь — int (от англ. integer) — целочисленное значение в диапазоне от -32 768 до 32 767, с другими типами мы познакомимся позднее
- ✓ Переменные одного типа можно объявить в одной инструкции, перечислив их через запятую, что мы и сделали
- ✓ Функция analogRead(pinA) возвращает целочисленное значение в диапазоне от 0 до 1023, пропорциональное напряжению, поданному на аналоговый вход, номер которого мы передаем функции в качестве параметра pinA
- ✓ Обратите внимание, как мы получили значение, возвращенное функцией analogRead(): мы просто поместили его в переменную rotation с помощью оператора присваивания =, который записывает то, что находится справа от него в ту переменную, которая стоит слева

Задание 4. Ответьте на следующие вопросы

1. Можем ли мы при сборке схемы подключить светодиод и потенциометр напрямую к разным входам GND микроконтроллера?
2. В какую сторону нужно крутить переменный резистор для увеличения яркости светодиода?
3. Что будет, если стереть из программы строчку pinMode(LED_PIN, OUTPUT)? строчку pinMode(POT_PIN, INPUT)?

4. Зачем мы делим значение, полученное с аналогового входа перед тем, как задать яркость светодиода? что будет, если этого не сделать?

Задание 5. Самостоятельно измените существующую программу и схему

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