Эксперимент - Маячок с нарастающей яркостью

В этом эксперименте мы задаем различные уровни яркости светодиода.

Список деталей для эксперимента

- Arduino Uno
- Макетная плата
- Светодиод
- Резистор номиналом 220 Ом
- 2 провода «папа-папа»

Принципиальная схема

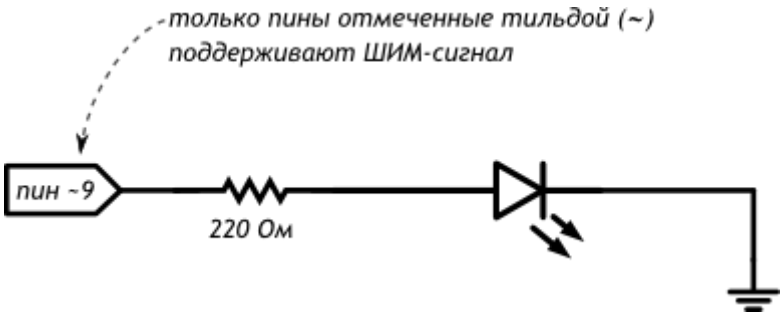
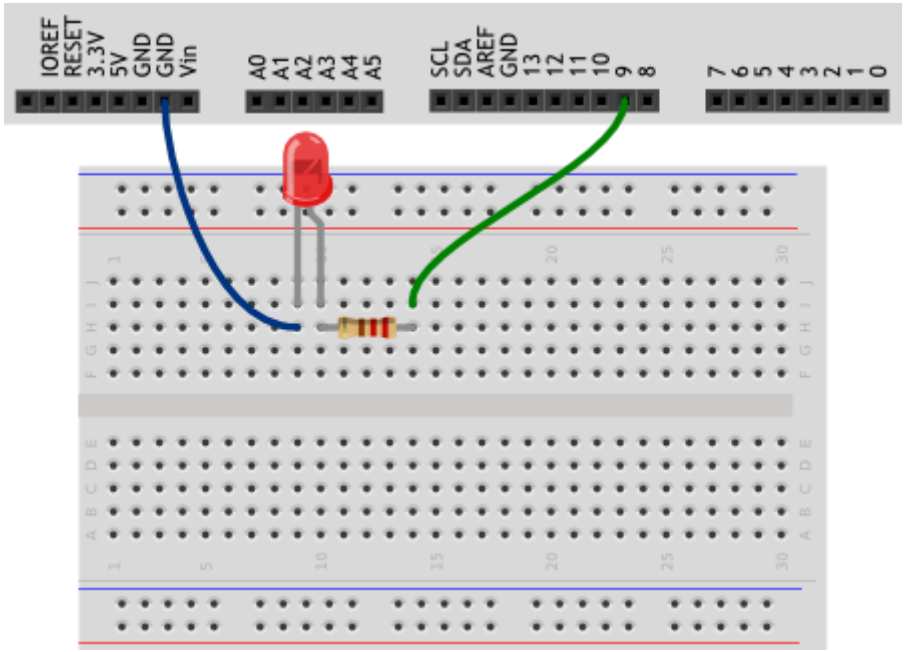


Схема на макетке



Обратите внимание

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

Скетч

p020_pulse_light.ino

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

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

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

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

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

Пояснения к коду

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

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

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

```
// однострочный комментарий следует после двойного слеша до конца строки
/* многострочный комментарий
   помещается между парой слеш-звездочка и звездочка-слеш */
```

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

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