**Применение резисторов при проектировании простых схем Arduino**

Резисторы являются важными элементами электронных схем. В контексте разработки проектов на Arduino их используют для:

* Ограничения тока, чтобы защитить компоненты.
* Формирования делителей напряжения.
* Подтяжки или стягивания входных сигналов к определённому уровню.

**Ограничение тока (защита элементов)**

При подключении **светодиодов и других** к Arduino через цифровой пин почти всегда необходим **токоограничивающий резистор** (обычно 220–1kΩ). Без него подключаемое устройство может выйти из строя.

Данный пример уже рассматривался раннее, но он хорошо иллюстрирует подключение светодиода через резистор – минус светодиода подключается к земле (GND), а плюс подключаем через ограничивающий резистор к питанию (OUTPUT пину).

Изображение выглядит как текст, снимок экрана, дизайн

Контент, сгенерированный ИИ, может содержать ошибки.

**Подтягивающие (Pull-up) и стягивающие (Pull-down) резисторы**

Подтягивающие и стягивающие резисторы используются для установки **определённого логического уровня** (HIGH или LOW) на входе микроконтроллера, когда нет активного сигнала. Без этих резисторов вход может находиться в неопределённом состоянии, что приводит к ложным срабатываниям.

* **Подтягивающий резистор (Pull-up)** – соединяет вход с **питанием (VCC)** через резистор. Когда источник сигнала отключён, вход остаётся в **состоянии HIGH**.
* **Стягивающий резистор (Pull-down)** – соединяет вход с **землёй (GND)** через резистор. Когда источник сигнала отключён, вход остаётся в **состоянии LOW**.

На приведенной схеме резистор, выделенный желтым (имеет сопротивление 10 кОм), является стягивающим, он тянет значение на пине к земле (LOW), чтобы лампочка не горела пока кнопка не нажата. Резистор, выделенный голубым цветом имеет 240 Ом и является ограничивающим

Изображение выглядит как текст, снимок экрана, Электронная техника, число

Контент, сгенерированный ИИ, может содержать ошибки.



// описываем логическую переменную state

// будем хранить состояние кнопки - HIGH или LOW

bool state;

void setup()

{

    // пин кнопки настраиваем на вход - INPUT

    pinMode(8, INPUT);

    // пин светодиода настраиваем на выход - OUTPUT

    pinMode(9, OUTPUT);

}

void loop()

{

    // считываем логическое значение с пина 8

    // сохраняем значение в переменной state

    state = digitalRead(8);

    // значение переменной state (состояние кнопки)

    // определит состояние светодиода (вкл/выкл)

    digitalWrite(9, state);

}

В конфигурации на следующем изображении тот же резистор будет подтягивающим, он устанавливает значение на 8 пине к HIGH, соответственно здесь лампочка горит когда кнопка не нажата, и гаснет когда нажата



Изображение выглядит как текст, снимок экрана, диаграмма, линия

Контент, сгенерированный ИИ, может содержать ошибки.



**Самостоятельное задание:**

Модифицируйте код так, что бы нажатие на кнопку переключало состояние лампочки (включена/выключена)

Подтягивающие пины для Arduino предусмотрены на аппаратном уровне, и могут быть включены программно при помощи pinMode(n, INPUT\_PULLUP);

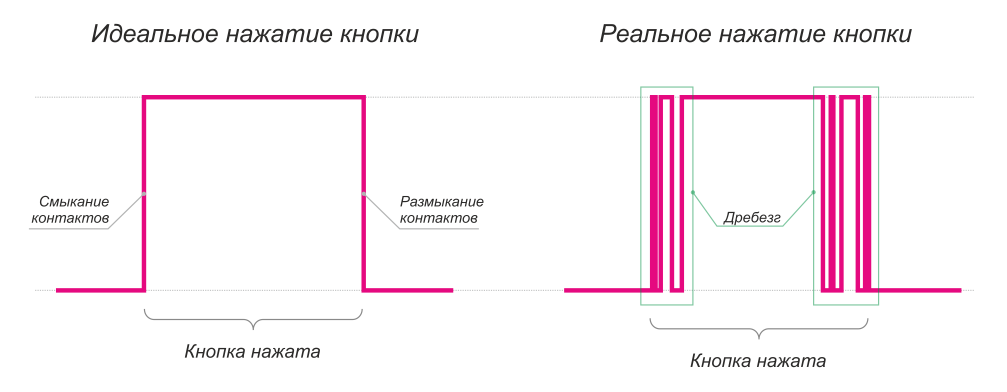
В таком случае нет необходимости использовать внешний резистор.

**Дребезг контактов**

Дребезг контактов кнопки Arduino – одно из самых неприятных и непонятных явлений, с которыми сталкивается начинающий ардуинщик. Устранение дребезга необходимо для корректной работы проекта, в противном случае на короткий отрезок времени схема становится практически неуправляемы.

Кнопка ардуино – один из самых популярных и простых видов датчиков. В основе работы любой кнопки лежит механический способ смыкания-размыкания контактов. Нажимая на любую, даже самую простую тактовую кнопку, мы формируем определенное давление на внутренние механизмы (пластины или пружины), в результате чего происходит сближение или расхождение металлических пластин.

Мы понимаем, что идеального в мире ничего не существует, в том числе идеально гладких поверхностей, контактов без неровностей, сопротивления и паразитной емкости. В нашем неидеальном мире в момент нажатия на кнопку в месте соединения контакты не соприкасаются мгновенно, микро-неровности на поверхности не позволяют пластинам мгновенно соединиться. Из-за этого в короткий промежуток времени на границе пластинок меняется и сопротивление, и взаимная емкость, из-за чего возникают масса разнообразных изменений уровня тока и напряжения. Другими словами, возникают очень интересные, хотя и не очень приятные процессы, которые в электротехнике называют переходными.



Переходные процессы протекают очень быстро и исчезают за доли миллисекунд. Поэтому мы редко их замечаем, например, когда включаем свет в комнате. Лампа накаливания не может менять свою яркость с такой скоростью, и тем более не может реагировать на изменения наш мозг. Но, обрабатывая сигал от кнопки на таком быстром устройстве, как Arduino, мы вполне можем столкнуться с такими переходными эффектами и должны их учитывать при программировании.

Как отразится дребезг на нашем проекте? Да самым прямым образом – мы будем получать на входе совершенно случайный набор значений. Ведь если мы считываем значение с кнопки непрерывно, в каждом новом рабочем цикле функции loop, то будем замечать все “всплески” и “падения” сигнала. Потому что пауза между двумя вызовами loop составляет микросекунды и мы измерим все мелкие изменения.

Вот пример скетча, в котором непременно обнаружится ошибка дребезга. Мы сможем увидеть в мониторе порта в первые мгновения после нажатия целый набор нулей и единиц в случайной последовательности (не важно, что означает 1 – нажатие или отпускание кнопки, важен сам факт появления хаоса).

void loop() {

if (digitalRead(PIN\_BUTTON)) {

Serial.println("1");

} else {

Serial.println("0");

}

}

Самым простым способом справиться с проблемой дребезга кнопки является выдерживание паузы. Мы просто останавливаемся и ждем, пока переходный процесс не завершится. Для этого можно использовать функцию delay()или millis()  (за подробной информации можете обратиться к статье про использование функций delay() и millis() в ардуино). 10-50 миллисекунд – вполне нормальное значение паузы для большинства случаев.

int currentValue, prevValue;

void loop() {

currentValue = digitalRead(PIN\_BUTTON);

if (currentValue != prevValue) {

// Что-то изменилось, здесь возможна зона неопределенности

// Делаем задержку

delay(10);

// А вот теперь спокойно считываем значение, считая, что нестабильность исчезла

currentValue = digitalRead(PIN\_BUTTON);

}

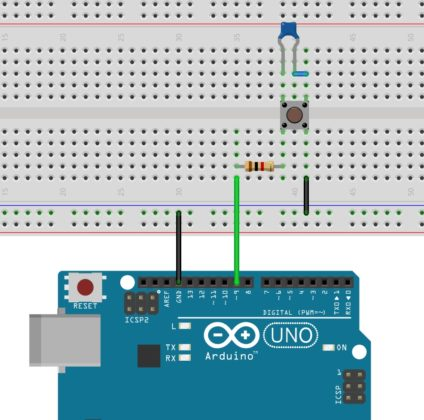
prevValue = currentValue;

Serial.println(currentValue);

}

Более правильный (и более сложный) способ борьбы с дребезгом – использование аппаратного решения, сглаживающего импульсы, посылаемые с кнопки. Для этого, правда, придется внести изменения в схему.

Аппаратный способ устранения дребезга основан на использовании сглаживающих фильтров. Сглаживающий фильтр, как следует из названия, занимается сглаживанием всплесков сигналов за счет добавления в схему элементов, имеющих своеобразную “инерцию” по отношению к таким электрическим параметрам как ток или напряжение. Самым распространенным примером таких “инерционных” электронных компонентов является конденсатор. Он может “поглощать” все резкие пики, медленно накапливая и отдавая энергию точно так же, как это делает пружина в амортизаторах.



**К сожалению, в симуляторе Wokwi не поддерживаются внешние конденсаторы**

**Использование пьезоэлемента**

Пьезоэлементы широко применяются в электронике для создания звуковых эффектов, сигнализации и даже воспроизведения мелодий. В сочетании с платформой Arduino они позволяют легко программировать звучание с различной частотой и длительностью.

Пьезоэлектрический элемент работает на основе обратного пьезоэлектрического эффекта: при подаче электрического сигнала он изменяет свою форму, создавая вибрации, которые превращаются в звук. Частота этих вибраций определяет высоту звука.

Подключение пьезоэлемента к Arduino. Для работы потребуется:

* Arduino (Uno, Nano, Mega и др.)
* Пьезоизлучатель (например, модель *BZV85* или аналог)
* Резистор 1 кОм (по желанию, для снижения громкости)
* Соединительные провода

Схема подключения:

* Один контакт пьезоэлемента подключается к GND (земля)
* Второй контакт подключается к цифровому выходу Arduino, например, 11
* Дополнительно можно поставить резистор между пьезоэлементом и землёй

Изображение выглядит как текст, снимок экрана, электроника, круг

Контент, сгенерированный ИИ, может содержать ошибки.

Arduino предоставляет встроенную функцию tone(), которая позволяет легко генерировать звук.

Простейший код для воспроизведения звука:

const int piezoPin = 11; // Пин, к которому подключен пьезоэлемент

void setup() {

pinMode(piezoPin, OUTPUT);

}

void loop() {

tone(piezoPin, 1000); // Воспроизведение звука частотой 1000 Гц

delay(500); // Держим 0.5 секунды

noTone(piezoPin); // Выключаем звук

delay(500);

}

Функция tone(pin, frequency, duration) принимает:

* pin — номер цифрового вывода
* frequency — частоту в Гц (например, 440 Гц — нота Ля)
* duration *(необязательно)* — длительность звучания в миллисекундах

Функция noTone(pin) отключает звук.

Создадим простую мелодию, используя массив частот нот.

const int piezoPin = 11;

int melody[] = {262, 294, 330, 349, 392, 440, 494, 523}; // Ноты до, ре, ми, фа, соль, ля, си, до

int noteDurations[] = {300, 300, 300, 300, 300, 300, 300, 300};

void setup() {

pinMode(piezoPin, OUTPUT);

}

void loop() {

for (int i = 0; i < 8; i++) {

tone(piezoPin, melody[i], noteDurations[i]);

delay(noteDurations[i] + 50); // Короткая пауза между нотами

}

delay(2000); // Пауза перед повтором мелодии

}

Этот код последовательно проигрывает ноты **C-D-E-F-G-A-B-C** (до, ре, ми, фа, соль, ля, си, до).

Другие интересные примеры мелодий для **Arduino** можно посмотреть по ссылке

<https://github.com/robsoncouto/arduino-songs>

Пьезоизлучатели — простой и доступный способ создания звуковых сигналов и музыки на **Arduino**. Используя функции **tone()** и **noTone()**, можно программировать воспроизведение мелодий, сигналы тревоги и даже простые музыкальные инструменты.

**Самостоятельное задание:**

Спроектируйте схему с пьезоизлучателем и проиграйте мелодии из папки по ссылке

<https://github.com/ReyRom-Edu/SYSPR/tree/main/Resources/arduino_songs>