ОДИН ДОМА

МАССИМО БАНЦИ

ARDUINO

ДЛЯ НАЧИНАЮЩИХ ВОЛШЕБНИКОВ

КАК СОЗДАТЬ ПРОГРАММИРУЕМОЕ
ЭЛЕКТРОННОЕ УСТРОЙСТВО,
НЕ ЗНАЯ ЭЛЕКТРОНИКИ
И ПРОГРАММИРОВАНИЯ?

ЧИТАЙ!

MASSIMO BANZI

GETTING STARTED WITH ARDUINO

ARDUINO

First Edition
O'Reilly

Beijing, Cambridge, Farnham, Köln, Sebastopol, Tokyo

МАССИМО БАНЦИ

ARDUINO

ДЛЯ НАЧИНАЮЩИХ ВОЛШЕБНИКОВ

МОСКВА
ЧИТАЙ!
Рид Групп
2012

УДК 004.3
ББК 32.96
Б 23

Перевод с английского — *М. Райтман*

Банци М.

Б 23 Arduino для начинающих волшебников / М. Банци. — М.: Рид Групп, 2012. — 128 с. — (Один дома).

ISBN 978-5-4252-0631-2

Эта книга о платформе Arduino, которая день ото дня становится все популярнее, и целая армия экспериментаторов-надомников, конструкторов-любителей и хакеров начинает использовать ее для воплощения в жизнь как прекрасных, так и совершенно сумасшедших проектов.

С помощью Arduino любой гуманитарий может познакомиться с основами электроники и программирования и быстро начать разработку собственных моделей, не тратя на это значительных материальных и интеллектуальных ресурсов.

Arduino объединяет игру и обучение, позволяет создать что-то стоящее и интересное под влиянием внезапного порыва, воображения и любопытства. Эта платформа расширяет возможности креативного человека в сфере электроники, даже если он в ней ничего не смыслит!

Экспериментируйте и получайте удовольствие!

УДК 004.3
ББК 32.96

Научно-популярное издание

ОДИН ДОМА

Массимо Банци

ARDUINO ДЛЯ НАЧИНАЮЩИХ ВОЛШЕБНИКОВ

Главный редактор *И. Федосова*
Заведующий редакцией *А. Баранов*. Ведущий редактор *И. Липкин*
Художественный редактор *А. Богомолов*. Дизайн обложки *А. Богомолов*
Технический редактор *В. Фотиева*. Верстка *А. Попов*

ООО «Рид Групп»
Москва, ул. Россолимо, 17, стр. 1, тел. (495) 788-0075 (76)

Издание осуществлено при техническом содействии
ООО «Издательство АСТ»

Подписано в печать 14.01.2012. Формат 60x90/ 16.
Усл. печ. л. 8,0. Печать офсетная. Бумага офсетная. Тираж 2 000 экз. Заказ № 5679М.

Отпечатано с готовых диапозитивов в ООО «Полиграфиздат»
144003, г. Электросталь, Московская область, ул. Тевосяна, д. 25

ISBN 978-5-4252-0631-2

Оглавление

Arduino для начинающих волшебников

Массимо Банци

Важное сообщение для наших читателей

За свою безопасность отвечаете вы сами. Это относится к надлежащему использованию оборудования и устройств защиты, а также к определению наличия требуемых навыков и соответствующего опыта. Электричество и другие ресурсы, используемые для описываемых проектов, являются источниками повышенной опасности, если применяются ненадлежащим образом и без необходимых мер предосторожности, в частности, без использования средств защиты. Для более наглядной демонстрации этапов проекта на некоторых иллюстрациях не продемонстрированы меры безопасности и средства защиты. Описываемые проекты не должны реализовывать дети.

Читатели сами принимают решение о выполнении инструкций и рекомендаций, изложенных в настоящей книге. Корпорация O'Reilly Media и автор снимают с себя всякую ответственность за любые поломки, травмы и возможный ущерб. В обязанности пользователя входит проверка соответствия всех работ, выполняемых по инструкциям и рекомендациям, изложенным в книге «Начинаем работать с Arduino», действующим законам, в частности, закону об авторских правах.

O'REILLY®

Make

Предисловие

Несколько лет назад мне сделали интересное предложение. Я должен был преподавать дизайнерам основы электроники, чтобы они могли создавать интерактивные прототипы разрабатываемых объектов.

Я доверился интуиции, которая подсказала мне, что их надо обучать электронике так же, как это делается в школе. Потом я понял, что такой подход не работает. Я вспомнил унылые, скучные до невозможности уроки, всю эту теорию, которую нам вдалбливали в огромных объемах, причем без применения ее на практике.

На самом деле, учась в школе, я уже разбирался в электронике, постигая ее эмпирическим путем (минимум теории — максимум практических занятий). Выглядело это примерно так:

- Я разбирал любое электронное устройство, которое попадало мне в руки.
- Медленно, но верно постигал, для чего нужны были составляющие его элементы.
- Экспериментировал с электронными устройствами: менял некоторые детали и смотрел, что получится. Как правило, выходило нечто среднее между небольшим взрывом и дымовой завесой.
- Я начал собирать устройства из комплектов, которые приходили с журналами по электронике.
- Я комбинировал разобранные устройства. Кроме того, я использовал наборы не так, как это предписывалось, а в соответствии со схемами, которые в корне отличались от журнальных, чтобы получать устройства с новыми функциями.

Еще ребенком я всегда приходил в восторг, когда до меня доходило, как работают те или иные устройства или механизмы. Видимо, поэтому я обычно разбирал их на составные части. Подобный «зуд» нападал на меня всякий раз, когда в моих руках оказывался предмет, не нашедший применения в домашнем хозяйстве. Я немедленно разбирал его на де-

тали. Со временем соседи и даже малознакомые люди стали приносить мне различные устройства для изучения. Самыми крупными моими проектами того времени были посудомоечная машина и компьютер, который выбросила за ненадобностью страховая компания. В комплекте с компьютером мне достался огромный принтер. Компьютер оказался очень интересным: там были разные платы, считыватели магнитных карт и много других сложных узлов, которые было довольно трудно разобрать.

После препарирования колоссального числа самых разных устройств я начал соображать, что представляют собой электронные детали, и в общих чертах стал понимать, как они функционируют. К счастью, дома было полно старых журналов по электронике, которые мой отец покупал еще в начале 70-х годов прошлого века. Я часами читал статьи и рассматривал схемы, мало понимая заложенные в них принципы.

Многократное прочтение статей вкупе с теми сведениями, которые я получал, препарируя электронные устройства, само собой превратилось в некий цикл занятий, позволивший поступательно, не торопясь, но достаточно эффективно получить необходимые знания в данной области.

Прорыв произошел в один из рождественских праздников, когда отец подарил мне электронный конструктор. Каждый компонент здесь был заключен в прозрачный кубик, который мог примагничиваться к другим кубикам, образуя соединение. На верхней стороне каждого кубика была нарисована пиктограмма — обозначение электронного компонента. В то время я ничего не знал о том, что эта игрушка стала поворотным событием в немецком дизайне, поскольку Дитер Рэмз спроектировал ее еще в 60-е годы.

Конструктор давал мне возможность экспериментировать, быстро комбинируя компоненты, и наблюдать за результатами. Таким образом, цикл разработки прототипа все сокращался и сокращался.

Уже потом я разрабатывал и собирал радиоприемники, усилители, схемы, способные издавать как ужасные шумы, так и приятные звуки, датчики дождя и крошечных роботов.

Я потратил много времени на поиск английского слова, которое обозначало бы такой род деятельности, то есть работу без конкретного плана, которая начинается с идеи и завершается непредсказуемым результатом. В конце концов на вооружение было взято слово tinkering (починка). Я осознаю, как это слово используется во многих других сферах для описания вполне определенной деятельности и для характери-

стики людей, вставших на путь исследования. К примеру, французских режиссеров — родоначальников «новой волны» (Nouvelle Vague), называли «лудильщиками». Лучшее, на мой взгляд, определение термина «тинкеринг» я встретил на выставке, которая проводилась в «Эксплораториуме» в Сан-Франциско.

Тинкеринг — это когда пытаются сделать что-то, не понимая как, но под влиянием внезапного порыва, воображения и любопытства. Когда произвольно комбинируют, припаивая друг к другу разные детали. Когда нет инструкций, но при этом нет и ошибок. Когда нет ни правильного, ни неправильного способа сделать какую-то вещь. Этот процесс подразумевает разгадывание принципов работы различных устройств и их переделку.

Хитроумные изобретения, машины или абсолютно не соединимые элементы, работающие гармонично, — это стихия тинкеринга.

Тинкеринг — процесс, объединяющий игру и изучение.

(www.exploratorium.edu/tinkering)

В результате ранних экспериментов я узнал, как много раз надо попробовать и ошибиться, чтобы создать из базовых деталей схему, которая бы работала так, как задумано.

Важное событие в моей жизни произошло летом 1982 г., когда я поехал с родителями в Англию. В то время в Лондонском музее истории науки как раз открыли новую экспозицию, посвященную компьютерам. Там попутно с многочасовыми экскурсиями я выполнил серию экспериментов под руководством инструктора и освоил основы двоичного счисления и программирования.

Именно тогда я понял, что в большинстве приложений инженеры уже не разрабатывают схемы на основе элементарных электронных компонентов, многие логические функции они реализуют с помощью микропроцессоров, а использование программного обеспечения компенсирует львиную долю времени, затрачиваемого на проектирование электронных устройств, а значит, позволяет сократить цикл разработки.

Вернувшись домой, я начал собирать деньги на компьютер, чтобы научиться программировать.

После приобретения персональной вычислительной техники моим главным достижением стало использование новейшей модели компьютера ZX81 для управления сварочным аппаратом. Я понимаю, что для

читателя это звучит не очень впечатляюще, но я нуждался в таком проекте, для меня это был вызов, ведь я только-только учился программировать. Тогда я на практике убедился в том, что составление программы занимает куда меньше времени, чем доработка и переработка сложной схемы.

Через двадцать с лишним лет этот опыт позволяет мне обучать людей, которые не помнят, проходили ли они математику в школе. Я надеюсь заразить их энтузиазмом и стремлением к стихийному изобретательству, которые владели мной в молодости и живут во мне и сегодня.

Массимо

Благодарности

Эта книга посвящается Луизе и Александре.

Прежде всего я хочу поблагодарить моих партнеров по группе, занимающейся Arduino: Девида Квартиле, Девида Мелиса, Жан-Люка Мартино и Тома Игое. Работа с этими парнями оставила неизгладимое впечатление.

Спасибо Барбаре Гелле, которая, возможно, и не подозревает, что без ее ценных советов проект Arduino и эта книга могли не появиться на свет.

Благодарю Билла Верпланка, научившего меня очень многому.

Спасибо Эрнандо Баррагану за работу по коммутации отдельных компонентов и Брайану Джепсону за неподъемную редактуру и постоянную поддержку на протяжении всей работы над книгой.

Благодарю Нэнси Котари, Брайана Скотта, Терри Бронсона и Патти Шинделькмана, превративших мою рукопись в книгу.

Я хотел бы поблагодарить еще многих, но Брайан говорит, что для этого недостаточно места, поэтому я лишь назову тех, кому я глубоко признателен:

Адам Сомлаи-Фишер, Аилади Кортеллетти, Алберто Пецотти, Алессандро Герминаси, Алессандро Массердотти, Андреа Пикколо, Анна Капеллини, Касей Риас, Крис Андерсон, Клаудио Модерини, Клементина Коппини, Кончетта Капеччи, Цаба Вальдхаусер, Дарио Буцини, Дарио Молинари, Дарио Парравичини, Доната Пикколо, Едуардо Брамбилла, Елиса Кандуччи, Фабио Виоланте, Фабио Занола, Фабрицио Пигнолони, Флавио Маури, Франческа Мокеллин, Франческо Монико, Жоржио Оливеро, Джиованна Гарди, Джованни Баттистини, Хитер Мартин, Дженнифер Бове, Лаура Делламотта, Лоренцо Парравичини, Лука Рокко, Марко Байони, Марко Эйнард, Мариа Тереса Лонгони, Массимилиано Болонди, Маттео Риволта, Матиас Рихтер, Маурицио Пирола, Микаел Торпе, Наталиа Джордан, Омбретта Банци, Оресте Банци, Оскар Зоггиа, Пиетро Доре, Проф Салвиони, Раффаелла Феррара, Ренцо Гиусти, Санди Атанас, Сара Карпентиери, Зигрид Ведерхекер, Стефано Мирти, Уби Де Фео, Вероника Букко.

Как связаться с нами

Содержащаяся в книге информация тщательно нами проверена, однако, возможно, вы обнаружите какие-то недочеты или даже ошибки. Если вы пришлете свой отзыв, это поможет нам улучшить последующие издания. Сообщайте нам обо всех ошибках, неточностях и опечатках, обнаруженных в книге.

Расскажите, что, по вашему мнению, можно сделать, чтобы книга стала лучше. Все комментарии будут внимательно рассмотрены, и мы попытаемся учесть ваши предложения в последующих изданиях.

Наш адрес:

Maker Media

1005 Gravenstein Highway North

Sebastopol, CA 95472

(800) 998-9938 (в США или Канаде)

(707) 829-0515 (международный/местный)

(707) 829-0104 (факс)

Maker Media является подразделением компании O'Reilly Media, деятельность которой ориентирована на постоянно растущее сообщество изобретательных людей, которые считают, что в сфере техники можно реализовать практически любую мечту.

Выпуская журналы Make, Craft, Maker Faire, а также серии книг Hacks, Make: Projects и DIY Science, издательство Maker Media способствует развитию рационализаторского мышления, поощряет творческое вдохновение и предоставляет рекомендации по самостоятельной работе.

Дополнительные сведения о Maker Media см. в Интернете по адресам:

MAKE	www.makezine.com
CRAFT:	www.craftzine.com
Maker Faire:	www.makerfaire.com
Hacks	www.hackszine.com

Комментарии по этой книге шлите по электронной почте bookquestion@oreilly.com.

На веб-сайте O'Reilly в разделе, посвященном книге Getting Started with Arduino, приводятся примеры, обнаруженные опечатки и планы по будущим изданиям. Данная страница находится в Интернете по адресу www.makezine.com/getstartedarduino.

Дополнительные сведения об этой и других книгах см. на веб-сайте компании O'Reilly по адресу www.oreilly.com.

Дополнительные сведения о проекте Arduino, форумы и дополнительную документацию можно найти по адресу www.arduino.cc.

Глава 1. Введение

Arduino — это аппаратная платформа, которая содержит простой интерфейс ввода-вывода и поддерживает среду разработки, реализующую открытый язык программирования Processing, основанный на Java (www.processing.org).

Arduino может служить основой для разработки автономных интерактивных устройств или может работать под управлением ПО, установленного на соединенном с ним компьютером. В последнем случае в качестве ПО помимо языка Processing могут использоваться Flash, VVVV и Max/MSP. Arduino можно собрать самостоятельно вручную, а можно приобрести уже в готовом виде. Интегрированную среду разработки с открытым кодом (IDE, Integrated Development Environment) можно загрузить бесплатно с веб-сайта www.arduino.cc.

От других подобных устройств, предлагающихся на рынке, Arduino отличается следующим свойствами:

- Это мультиплатформная среда; она может работать под управлением операционных систем Windows, Macintosh и Linux.
- Она базируется на языке Processing; эта простая и удобная в применении среда разработки используется художниками и дизайнерами.
- Программирование устройства осуществляется посредством USB-интерфейса, а не через последовательный порт. Это создает дополнительное удобство, поскольку последовательный порт отсутствует на большинстве современных компьютеров.

- Это устройство «открыто» в полном смысле слова. Если хотите, вы можете скачать на сайте его принципиальную схему, купить все необходимые элементы и спаять его самостоятельно, ни цента не выплачивая создателям Arduino.
- «Железо», из которого состоит Arduino, недорогое. USB-плата стоит около 20 EBPO, а замена сгоревшего чипа обойдется где-то в 5 EBPO. Так что можно позволить себе сколько угодно ошибаться.
- Существует сообщество активных пользователей, поэтому нет недостатка в людях, которые могут оказать помощь.
- Проект Arduino разрабатывался в университетской среде, поэтому он великолепно подходит для новичков, желающих быстро заставить функционировать задуманные устройства.

Цель данной книги — помочь новичкам разобраться, какие преимущества они могут извлечь из изучения платформы Arduino и используемых в ней подходов.

Целевая аудитория

Эта книга была написана для «настоящих» пользователей Arduino, то есть для тех, кто почти не разбирается ни в электронике, ни в программировании. Поэтому в ней я попытался использовать такую манеру изложения, которая сведет с ума дипломированного инженера. Кстати, один из них назвал вводные главы черновика моей рукописи «полной ерундой». Ему виднее, однако давайте считаться с тем фактом, что большинство инженеров не могут объяснить то, что они делают, не то что дилетантам, но даже своим коллегам. Поэтому давайте копнем поглубже мою «ерунду».

Примечание

В основе Arduino лежат результаты диссертации Хернандо Баррагана, которую он написал под моим руководством и руководством Кейси Риса, используя платформу Wiring.

По мере того как платформа Arduino становилась все более популярной, я начинал понимать, что целая армия всевозможных экспериментаторов, конструкторов-любителей и хакеров начинает использовать ее для воплощения в жизнь как прекрасных, так и совершенно сумасшедших проектов. Я осознал, что все вы, правда каждый по-своему, яв-

ляетесь дилетантами и авантюристами, значит, эта книга написана для вас.

Платформу Arduino придумали для обучения интерактивному проектированию, дисциплине, в центре методологии которой подразумевается разработка прототипов. Существует множество определений интерактивного проектирования, однако я отдаю предпочтение следующей формулировке:

Интерактивное проектирование — это проектирование на основе интерактивного моделирования

В современном мире интерактивное проектирование базируется на постановке экспериментов, связывающих людей и объекты. Оказалось, что это хороший способ исследования самых различных, бывает, что и противоречивых событий, происходящих между нами и технологиями. Интерактивное проектирование базируется на последовательном процессе, который представляет собой создание ряда прототипов с постоянно возрастающей степенью приближения к задуманной модели. Подобный подход применяется и в «традиционном» проектировании, однако он может быть модернизирован за счет создания прототипов на основе различных технологий, в частности, на основе современной электроники.

Специфической областью проектирования, связанной с Arduino, является интерактивное аппаратное моделирование.

Что такое аппаратное моделирование?

Аппаратное моделирование применяется для создания прототипов новых электронных устройств.

Аппаратное моделирование позволяет разрабатывать интерактивные устройства, которые могут взаимодействовать с людьми посредством датчиков и исполнительных механизмов, управляемых микропроцессором, который работает по определенной программе.

Раньше процесс создания электронного прибора сводился к постоянному общению с инженерами, которые разрабатывали на заказ устройства с определенными свойствами. Такой подход не давал возможности, к примеру, дизайнерам самостоятельно заниматься собственными разработками. Практически весь инструментарий разработки предна-

значался для квалифицированных специалистов, и его применение требовало глубоких и обширных знаний. В последние годы микропроцессоры и микроконтроллеры подешевели, существенно упростилось их применение, что позволило создавать более совершенный и гибкий инструментарий для моделирования.

Изобретение Arduino позволило сделать новый шаг в данной сфере. Теперь инструментарий для аппаратного моделирования стал настолько простым и понятным, что даже человек, который никогда этим не занимался, может легко создавать вполне работоспособные прототипы после пары дней изучения платформы и практических занятий с ней.

С помощью Arduino любой гуманитарий может познакомиться с основами электроники и программирования и быстро начать разработку собственных моделей, не тратя на это значительных материальных и интеллектуальных ресурсов.

Глава 2. Концепция Arduino

Философия Arduino основывается на реализации проектов, а не на разговорах о них. Она выражается в постоянном поиске все более быстрых и мощных способов создания прототипов с постоянно улучшающимися параметрами. Именно для этого мы исследовали многочисленные варианты моделирования и разработали методики воплощения мысли непосредственно в «железе».

Классическая инженерия основывается на строго определенном продвижении из точки «А» в точку «Б». Двигаясь по пути, предлагаемому Arduino, получаешь удовольствие от того, что, даже потеряв ориентиры, все равно попадаешь в какую-то точку, если не в «Б», так в «В».

Это тот самый «тинкеринг», который все мы так любим. Это своеобразная игра с физической средой, когда ты идешь по неизведанной дороге и по пути обнаруживаешь удивительные вещи. Пытаясь найти наилучший способ создания прототипа, мы непрерывно экспериментируем не только с «железом», но и с различными вариантами программного обеспечения.

В следующих разделах рассказывается о людях, принципах и событиях, которые вдохнули жизнь в концепцию Arduino.

Моделирование

В основе концепции Arduino лежит моделирование. Мы конструируем и создаем устройства, которые взаимодействуют с другими устройствами, людьми и компьютерами. При этом мы стремимся найти наиболее простой, быстрый и дешевый способ моделирования.

Многие люди, начинающие знакомиться с электроникой, думают, что им необходимо стартовать с нуля. Это пустая трата энергии. Что им действительно надо, так это увидеть, как шустро функционирует какое-то устройство, и захотеть сделать следующий шаг в его усовершенствовании. А может быть, убедить кого-то дать немного денег на создание какой-то штуковины, которую вы сами придумали.

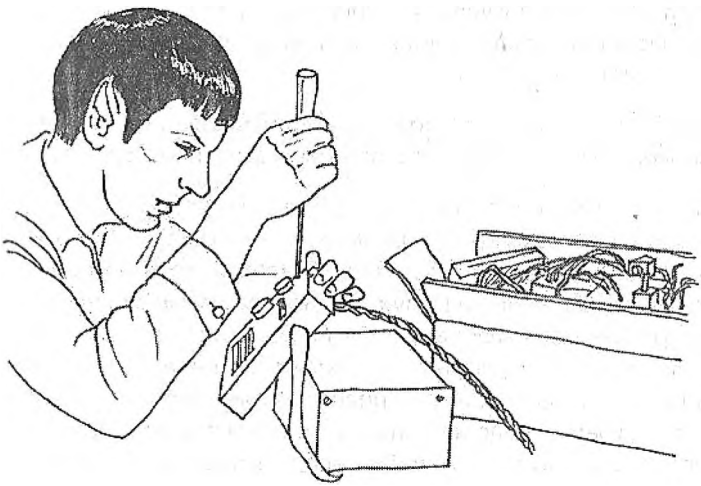
Для этого мы разработали так называемый «меркантильный способ создания прототипов». Действительно, зачем тратить силы, создавая с нуля машину или прибор, разработка которого, по-хорошему, требует времени и глубоких технических знаний, когда можно взять готовые устройства и разобрать их, чтобы воспользоваться плодами солидной работы, выполненной крупными компаниями и квалифицированными инженерами?

Известно, что Джеймс Дайсон создал 5127 прототипов пылесоса, прежде чем его удовлетворил результат (www.international.dyson.com/jd/1947.asp).

Тинкеринг

Главная идея нашей методики — игра в технологии, которая позволяет проводить исследования непосредственно на аппаратуре и программном обеспечении, причем зачастую без какой-то определенной цели.

Повторное использование элементов и узлов готовых промышленных устройств — один из способов любительского тинкеринга. Зачастую для того, чтобы получить замечательный результат, достаточно взять несколько дешевых игрушек или старых ненужных устройств, разобрать их и сделать что-то новое.

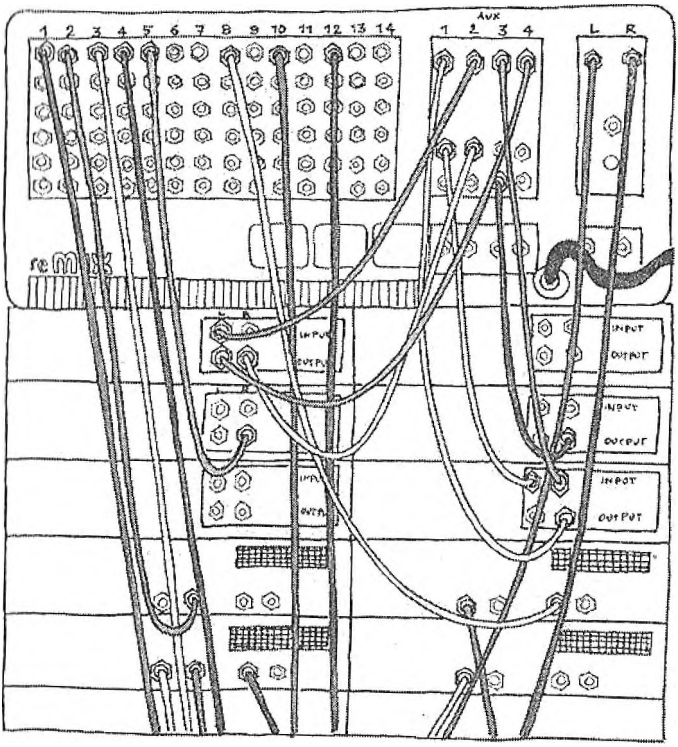


Метод последовательных преобразований

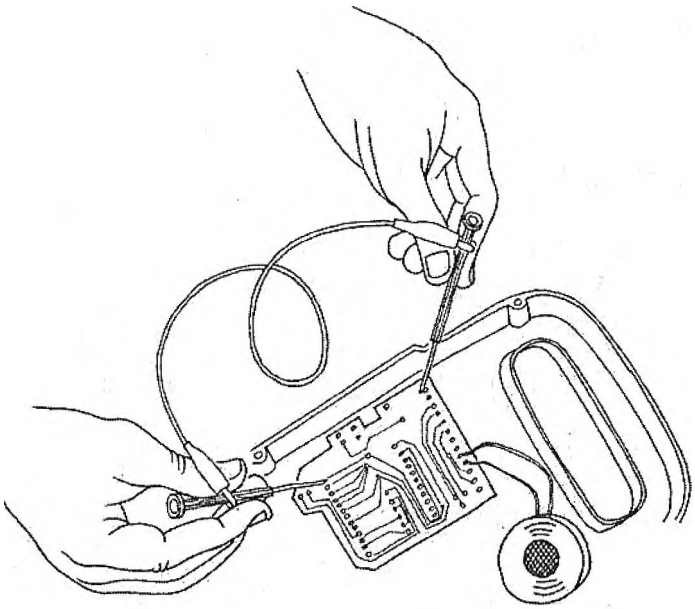
Я всегда восхищался модульными решениями и возможностью создавать сложные системы путем соединения простых устройств. Этот метод очень интересно использовал Роберт Муг в своих аналоговых синтезаторах. На этих устройствах музыканты, бесконечно комбинируя, создавали удивительные, небывалые звуки простым соединением модулей при помощи кабелей. Синтезатор Муга напоминал допотопный телефонный коммутатор, дополненный многочисленными кнопками и переключателями. В свое время он оказался идеальной платформой для экспериментов со звуком и созданием новаторской музыки. Муг описывал этот процесс как нечто среднее между восприятием и открытием. Я уверен, что большинство музыкантов поначалу даже не представляли себе, какую функцию выполняет та или иная кнопка, но они пробовали и пробовали, оттачивая свой неповторимый стиль в этом бесконечном потоке звуков.

В творческом процессе главное — не останавливаться, ведь чем меньше проблем возникает во время работы, тем интереснее результат.

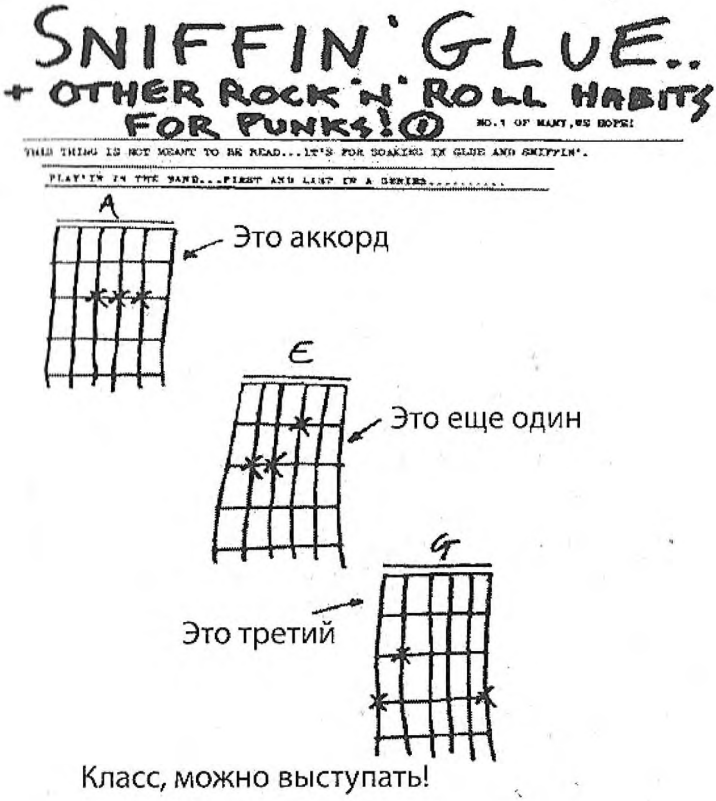
Сегодня этот метод используется в мире компьютеров в таких средах визуального программирования, как Max, Pure Data и VVVV. Эти инструменты визуализируют функции в виде боксов, из которых пользователь по своему усмотрению конструирует программные модули, соединяя боксы в различных комбинациях. Эти среды дают возможность непрерывно экспериментировать с программированием, что совсем не характерно для классического варианта написания кода, при котором сначала программу надо набрать в текстовом редакторе, затем откомпилировать, напороться на ошибки, исправить, снова откомпилировать и т.д.


1 2 3 4 5 6 7 8 9 10 11 12 13 14
AUX
1 2 3 4
L R
INPUT
OUTPUT

Метод коротких замыканий



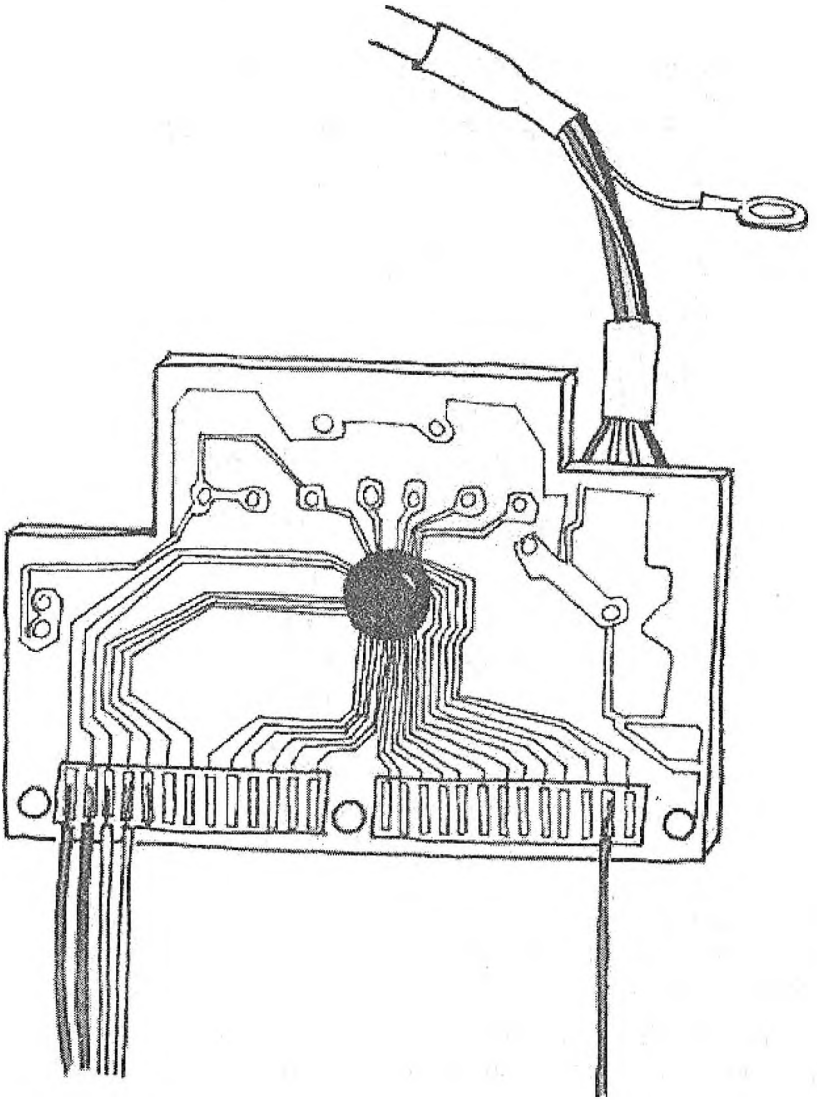
Не пугайтесь, это не более чем установка временной перемычки на плате низковольтного электронного устройства, причем наугад. Кстати, этот метод представляется мне одним из наиболее интересных вариантов тинкеринга. Представьте себе, каких неожиданных эффектов можно добиться, если таким вот образом поэкспериментировать с гитарными педалями или детскими синтезаторами! Суть данного метода заключается в «искусстве случая», а сам метод так же случайно возник в 1966 году, когда Рид Газала что-то замкнул в игрушечном усилителе, выдвигая ящики стола, а в результате возник поток необычайных звуков. Мне нравятся любители коротких замыканий, они способны создавать немыслимые устройства, комбинируя электронные компоненты, при этом совершенно не понимая, что они делают.



Все это немного созвучно напечатанной здесь листовке. В эпоху расцвета панков знания трех аккордов на гитаре было достаточно для создания рок-группы.

Поэтому не позволяйте узким специалистам утверждать, что вы никогда не подниметесь до их уровня. Проигнорируйте и удивите их.

Взлом клавиатуры



Вот уже более 60 лет клавиатура остается основным устройством, обеспечивающим взаимодействие человека с компьютером. Алекс Пентланд, глава лаборатории мультимедиа Массачусетского технологического института, однажды заметил: «Простите за грубость, но мужские писсуары сообразительнее компьютеров. Компьютеры изолированы от того, что вокруг них происходит».

Исповедуя тинкеринг, мы можем реализовать новые способы взаимодействия с программным обеспечением, заменив клавиши на устройства, способные реагировать на изменения окружающей среды.

Для начала давайте разберем компьютерную клавиатуру. Сняв корпус, мы увидим внутри очень простое и дешевое устройство — небольшую, немного вонючую плату обычно зеленого или коричневого цвета с двумя пластиковыми разъемами, куда втыкаются провода, идущие от клавиш. Если удалить схему и при помощи провода соединить два любых контакта, можно увидеть, что на экране компьютера появилась буква.

Если таким же образом подсоединить к клавиатуре датчик перемещения, вы увидите, что буква будет появляться всякий раз, когда кто-нибудь проходит перед компьютером. Запрограммировав получившееся устройство, вы повысите интеллект компьютера до уровня писсуара.

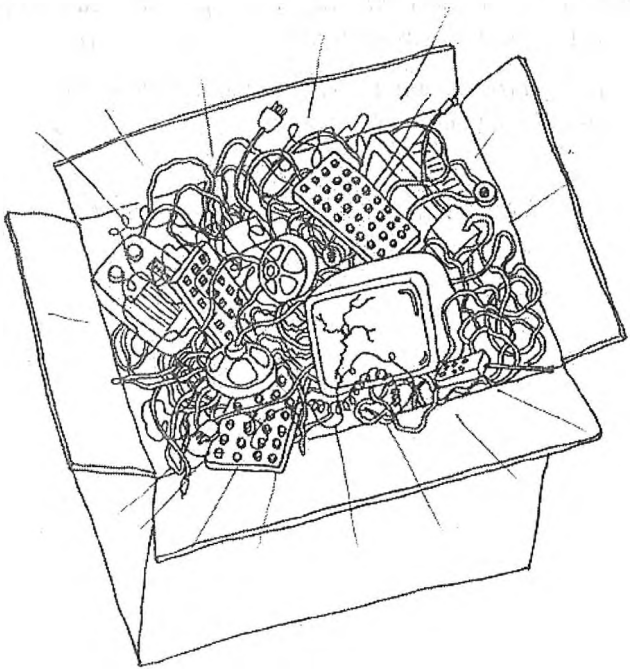
Эксперименты с компьютерной клавиатурой играют ключевую роль в моделировании и создании прототипов.

Мне нравятся свалки!

В наши дни люди выбрасывают на помойку горы устаревшей техники: это принтеры, компьютеры, разнообразные офисные машины, технологическое оборудование и всякое военное барахло. Рынок технического старья процветает, особенно он популярен в среде молодых или небогатых хакеров, а также тех, кто только собирается пополнить их ряды. Такой рынок открыто функционировал в Иври, где находится штаб-квартира Olivetti. Именно в этом городе мы разрабатывали Arduino.

Как известно, компания Olivetti производила компьютеры начиная с 60-х годов прошлого века, а в середине 90-х они вдруг повыбрасывали на местные свалки все, что устарело. В результате там оказалось полным-полно компьютерных деталей, электронных компонентов и всевозможных причудливых устройств. Мы проводили на этом стихийном базаре массу времени, буквально за гроши покупая все, что попадалось под руку, а затем используя весь этот «мусор» в наших прототипах.

Когда вы можете купить тысячу громкоговорителей за смешные деньги, у вас обязательно возникнет какая-нибудь идея. Накапливайте старую технику и разбирайте ее, прежде чем начать конструировать что-то с нуля.

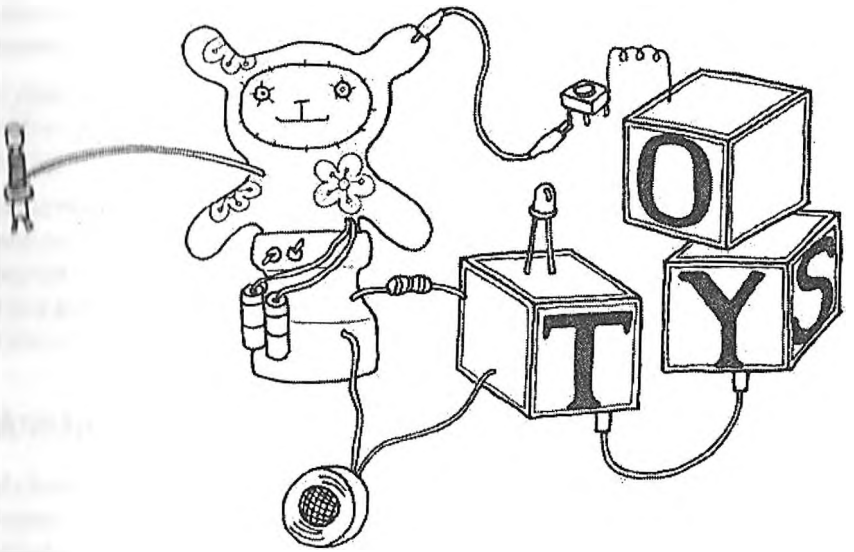


Куклу — на вскрытие!

Как показывает практика тинкеринга, детские игрушки представляют собой фантастический источник дешевых устройств, которые можно многократно переделывать и использовать. Колоссальное разнообразие очень дешевых высокотехнологичных китайских игрушек позволяет реализовывать в «железе» любую шальную мысль при помощи пары тявкающих щенят, танцующей крысы и «лазерного» меча из «Звездных войн».

Я занимаюсь этим уже несколько лет, причем не столько для собственного удовольствия, сколько для того, чтобы наглядно продемонстрировать студентам, что эта техника не такая уж непонятная и необъяснимая.

Одним из моих любимых ресурсов стала брошюрка «Простейшие датчики и исполнительные механизмы» (Low Tech Sensors and Actuators) Усмана Хака и Адама Сомлаи-Фишера. Я полагаю, что в ней авторы прекрасно описали этот метод, и я постоянно пользуюсь им с тех пор, как брошюра попала мне в руки.



Сотрудничество

Сотрудничество пользователей — один из ключевых принципов в мире Arduino. Благодаря форуму на сайте www.arduino.cc люди из разных уголков мира помогают друг другу в изучении этой платформы. Команда Arduino поощряет подобное сотрудничество и помогает участникам сообщества организовываться в группы в каждом городе, куда приезжают наши специалисты.

Кстати, для этих же целей мы создали общедоступную доску объявлений под названием Playground (Игровая площадка) [www.arduino.cc/playground], где пользователи документально оформляют свои изобретения.

Поразительно, столько интереснейшей информации эти люди выплескивают в Интернет для всеобщего пользования.

Культура коллективного творчества и взаимопомощи в сфере Arduino стала большим достижением, которым я очень горжусь.

Глава 3. Платформа Arduino

Arduino состоит из двух основных компонентов: платы, с которой работает пользователь при создании собственных прототипов, и интегрированной среды разработки (IDE) — программного обеспечения, установленного на компьютере. Интегрированная среда разработки используется для создания небольших программных модулей, которые загружаются в Arduino. Именно программный модуль говорит плате, что надо делать.

Еще недавно разработка электронных устройств подразумевала создание с нуля принципиальной схемы, состоящей из сотен различных элементов со странными наименованиями, такими как резистор, емкость, индуктивность, транзистор и т. д. Каждая схема предназначалась для выполнения конкретной задачи, а для внесения изменений требовалось резать существующие соединения, паять новые и выполнять множество другой работы.

С появлением цифровых технологий и микропроцессоров функции, которые ранее реализовывались посредством внесения изменений в соединения, заменили программированием.

Действительно, скорректировать код гораздо легче, чем перепаивать контакты в готовом и работоспособном приборе. Всего нескольких нажатий на клавиши достаточно для того, чтобы полностью поменять логику работы устройства, да опробовать еще два-три варианта за то же самое время, которое потребовалось бы для пайки двух резисторов.

Аппаратные средства Arduino

Arduino представляет собой небольшую плату, на которой установлен микроконтроллер, вернее, небольшой маломощный компьютер, по крайней мере, в тысячу раз менее мощный, чем MacBook, на котором я набирал текст этой книги. Тем не менее компьютер Arduino обладает двумя существенными преимуществами: он исключительно дешев и очень удобен для создания множества интересных устройств.

Посмотрев на плату Arduino, вы увидите черный кристалл с 28 «ножками» — это микросхема ATmega168. Кроме этой микросхемы на плате имеются все компоненты, которые необходимы для надлежащей работы и взаимодействия с внешним пользовательским компьютером.

Существует много версий платы Arduino. В этой книге описывается работа с версией Duemilanove — простейшей в использовании и оптимальной для обучения. Тем не менее приведенные здесь инструкции применимы для любых версий платы, включая свежую Arduino Uno, популярную Arduino Diecimila и старую Arduino NG. На рис. 3-1 показана плата Arduino Duemilanove, а на рис. 3-2 — Arduino NG.

Итак, вы видите плату Arduino, и с непривычки все эти элементы и разъемы могут сбить с вас толку. Не волнуйтесь, сейчас мы объясним, что делает каждый элемент на плате.

14 цифровых разъемов ввода-вывода (контакты 0-13)

Это могут быть как входы, так и выходы, что задается программным модулем, который вы создадите в интегрированной среде разработки на своем компьютере.

6 аналоговых входов (контакты 0-5)

На эти входы подаются от разных датчиков аналоговые сигналы, которые затем преобразуются в цифровые значения в диапазоне от 0 до 1023.

6 аналоговых выходов (контакты 3, 5, 6, 9, 10 и 11)

В действительности это шесть цифровых выводов из тех четырнадцати, о которых мы уже говорили двумя абзацами выше. Только эти шесть выводов можно перепрограммировать в аналоговые при помощи кода, который вы создадите в интегрированной среде разработки на своем компьютере.

Питание на плату может подаваться от USB-порта компьютера, от большинства зарядных USB-устройств или от любого адаптера, который на выходе выдает 9 В постоянного тока. Штекер должен быть диаметром 2,1 мм, а центральный вывод положительный. Если к гнезду электропитания ничего не подключено, напряжение будет подаваться по USB-кабелю. Как только вы вставите в гнездо электропитания штекер адаптера, плата автоматически переключится на этот источник.

Примечание

Если вы используете Arduino NG или Arduino Diecimila, то для выбора источника питания, EXT (внешнее) или USB вам потребуется установить в соответствующее положение переключатель PWR_SEL, который расположен на плате между разъемом для адаптера и USB-портом.

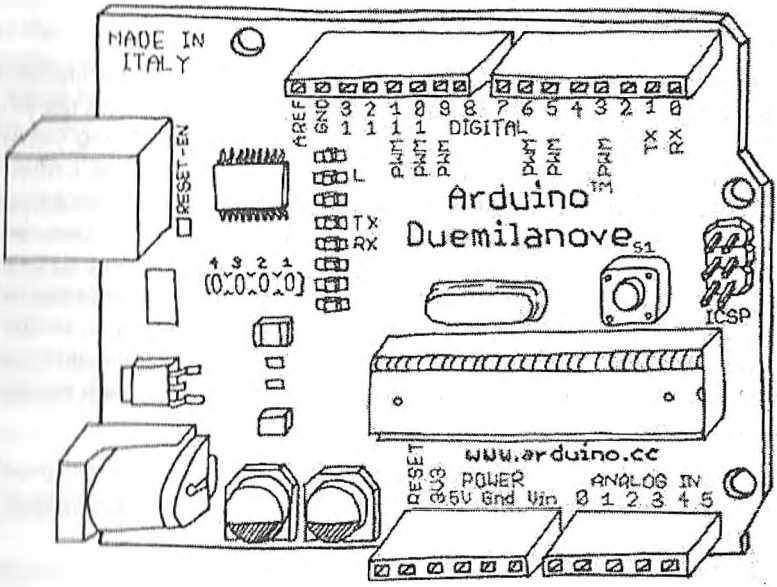


Рис. 3-1. Arduino Duemilanove

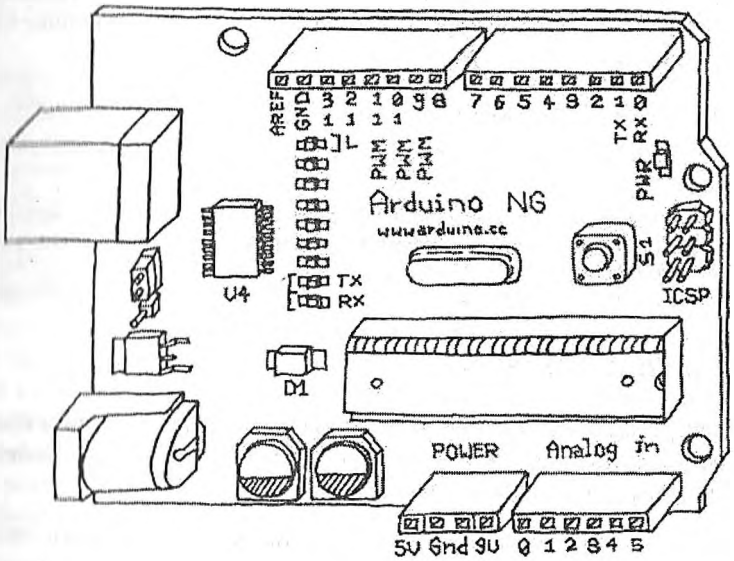


Рис. 3-2. Arduino NG

Программное обеспечение (IDE)

Интегрированная среда разработки (IDE, Integrated Development Environment), установленная на компьютере, позволяет составлять программные модули для платы Arduino на языке Processing (www.processing.org). Волшебство начинается при нажатии клавиши, с помощью которой в плату загружается программный модуль. Написанный вами код преобразуется в язык Си (достаточно трудный для начинающих пользователей) и передается компилятору avr-gcc. Эта важная часть программного обеспечения с открытым кодом выполняет заключительную трансляцию вашего модуля на язык, понятный для микроконтроллера Arduino. Именно этот этап облегчает жизнь пользователю, скрывая от него практически все нюансы программирования микроконтроллеров.

Цикл программирования на платформе Arduino, по существу, включает следующие действия:

- Соединение платы с USB-портом компьютера.
- Составление программного модуля, который должен «оживить» плату.
- Загрузка данного модуля в плату через USB-кабель и ожидание перезапуска платы, который происходит в течение нескольких секунд.
- Выполнение платой действий, заданных в программном модуле.

Примечание

На момент написания данной книги инсталляция Arduino в среде ОС Linux была связана с некоторыми сложностями. Сегодня дела обстоят очень неплохо, полные инструкции по установке см. по адресу www.arduino.cc/playground/Learning/Linux.

Инсталляция Arduino

Для программирования Arduino необходимо сначала загрузить среду разработки (IDE) с веб-сайта www.arduino.cc/en/Main/Software. Выберите правильную версию операционной системы.

Загрузите файл и дважды щелкните по нему, чтобы распаковать содержимое. В результате этих действий будет создана папка с именем arduino-[версия], например arduino-0012. Перетащите эту папку в удоб-

нее для вас место: на рабочий стол, в папку /Applications (на компьютерах Mac) или в папку C:/Program Files (в операционной системе Windows). Теперь, когда возникнет необходимость в запуске этого ПО, надо будет открыть папку arduino и дважды щелкнуть по значку Arduino. Пока делать этого не стоит, ведь нам остался еще один шаг, который должен быть выполнен.

Примечание

Если у вас возникли какие-то трудности с запуском интегрированной среды разработки Arduino, см. главу 7, «Поиск и устранение неполадок».

Теперь необходимо установить драйверы, которые позволят компьютеру общаться с платой через USB-порт.

Установка драйверов: Macintosh

Найдите папку Drivers (Драйверы) внутри папки arduino-0012 и дважды щелкните по файлу с именем *FTDIUSBSerialDriver_x_x_x.dmg*. (x_x_x будет заменяться номером версии драйвера, например *FTDIUSBSerialDriver_v2_2_9_Intel.dmg*). Дважды щелкните по файлу .dmg, чтобы загрузить его.

Примечание

Если вы используете Mac с процессором Intel, такой как MacBook, MacBook Pro, MacBook Air, Mac Pro или построенные на процессоре Intel компьютеры Mac Mini или iMac, не забудьте установить драйвер, содержащий в своем имени слово Intel, как в *FTDIUSBSerialDriver_v2_2_9_Intel.dmg*. В остальных случаях установите драйвер без слова Intel в названии.

Следующим шагом установите программное обеспечение из пакета FTDIUSBSerialDriver, дважды щелкнув по соответствующему значку. Следуйте инструкциям установки и, если программа запросит пароль, введите пароль администратора. В конце процедуры установки перезапустите компьютер, чтобы убедиться, что драйверы загружены надлежащим образом. Теперь подсоедините плату к компьютеру. На плате должен загореться светодиод PWR (Питание), а желтый светодиод L должен начать мигать. Если этого не происходит, см. главу 7, «Поиск и устранение неполадок».

Установка драйверов: Windows

Подсоедините Arduino к компьютеру. Когда на экране появится окно Found New Hardware Wizard (Мастер установки оборудования), операционная система Windows попытается найти нужный драйвер на веб-сайте Windows Update (Обновление Windows).

Windows XP выведет запрос о необходимости обращения к сайту Windows Update. Выберите вариант No, not this time (Нет, не в этот раз) и щелкните кнопку Next (Далее).

В появившемся окне выберите Install from a list or specific location (Установить из списка или конкретной папки) и щелкните по кнопке Next (Далее).

Установите флажок Include this location in the search (Включить в поиск эту папку), щелкните по кнопке Browse (Обзор), выберите папку, где уже находится Arduino, и папку *Drivers.FTDI USB Drivers* для драйверов. Нажмите кнопку OK, а затем Next (Далее).

Windows Vista сначала предпримет попытку найти драйвер на веб-сайте Windows Update. Когда эта попытка безуспешно завершается, можно указать для поиска драйвера папку *Drivers\FTDI USB Drivers*.

Данная процедура выполняется дважды, поскольку компьютер сначала устанавливает драйвер низкого уровня, затем устанавливает фрагмент кода, благодаря которому компьютер начинает идентифицировать плату как последовательный порт.

После установки драйверов можно запустить интегрированную среду разработки и приступить к экспериментам с платформой Arduino.

Далее нам необходимо разобраться, какой последовательный порт назначен для Arduino. Позже эта информация потребуется для программирования. Соответствующие инструкции вы найдете в следующих разделах.

Идентификация портов: Macintosh

Находясь в интегрированной среде разработки Arduino IDE, выберите в меню Tools пункт Serial Port, где выберите тот порт, наименование которого начинается с /dev/cu.usbserial. Это имя использует компьютер в качестве адреса Arduino. На рис. 3-3 показан список портов.

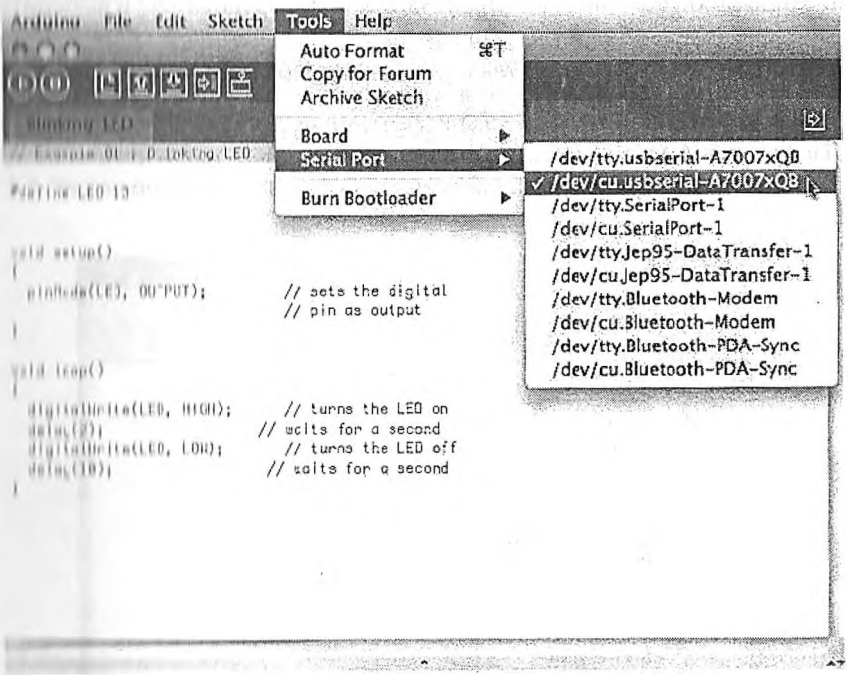


Рис. 3-3. Список последовательных портов в интегрированной среде разработки Arduino

Идентификация портов: Windows

В операционной системе Windows процесс идентификации немного сложнее, по крайней мере вначале. Откройте диспетчер устройств, нажав кнопку Start (Пуск), щелкните правой кнопкой мыши по пункту Computer (Компьютер) в Vista или My computer (Мой компьютер) в XP и выберите в контекстном меню пункт Properties (Свойства).

В Windows XP щелкните по пункту Hardware (Оборудование) и выберите Device Manager (Диспетчер устройств). В операционной системе Vista щелкните по пункту Device Manager (Диспетчер устройств).

Найдите устройство Arduino в списке, озаглавленном Ports (COM & LPT) [Порты (COM & LPT)]. Плата Arduino появится как последовательный USB-порт, и у нее будет имя, аналогичное COM3, как показано на рис. 3-4.

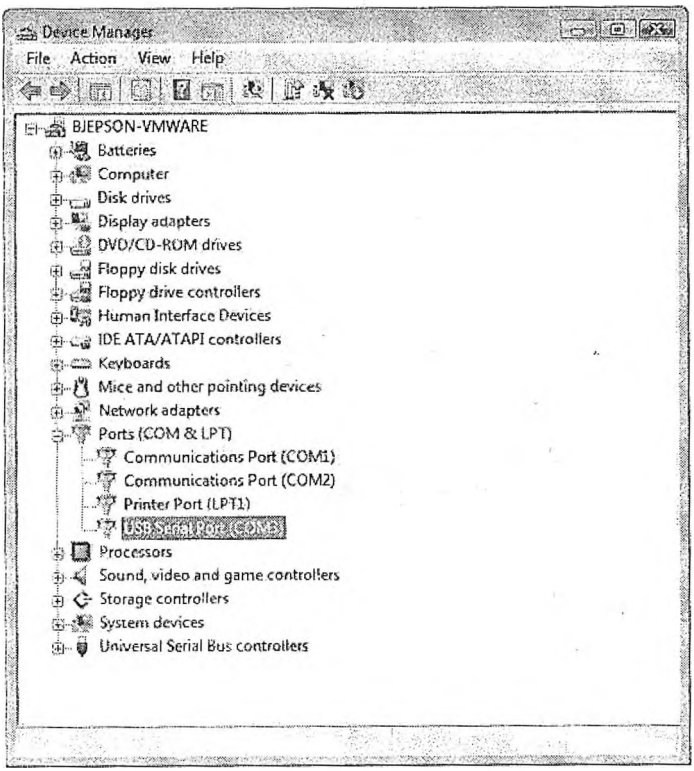


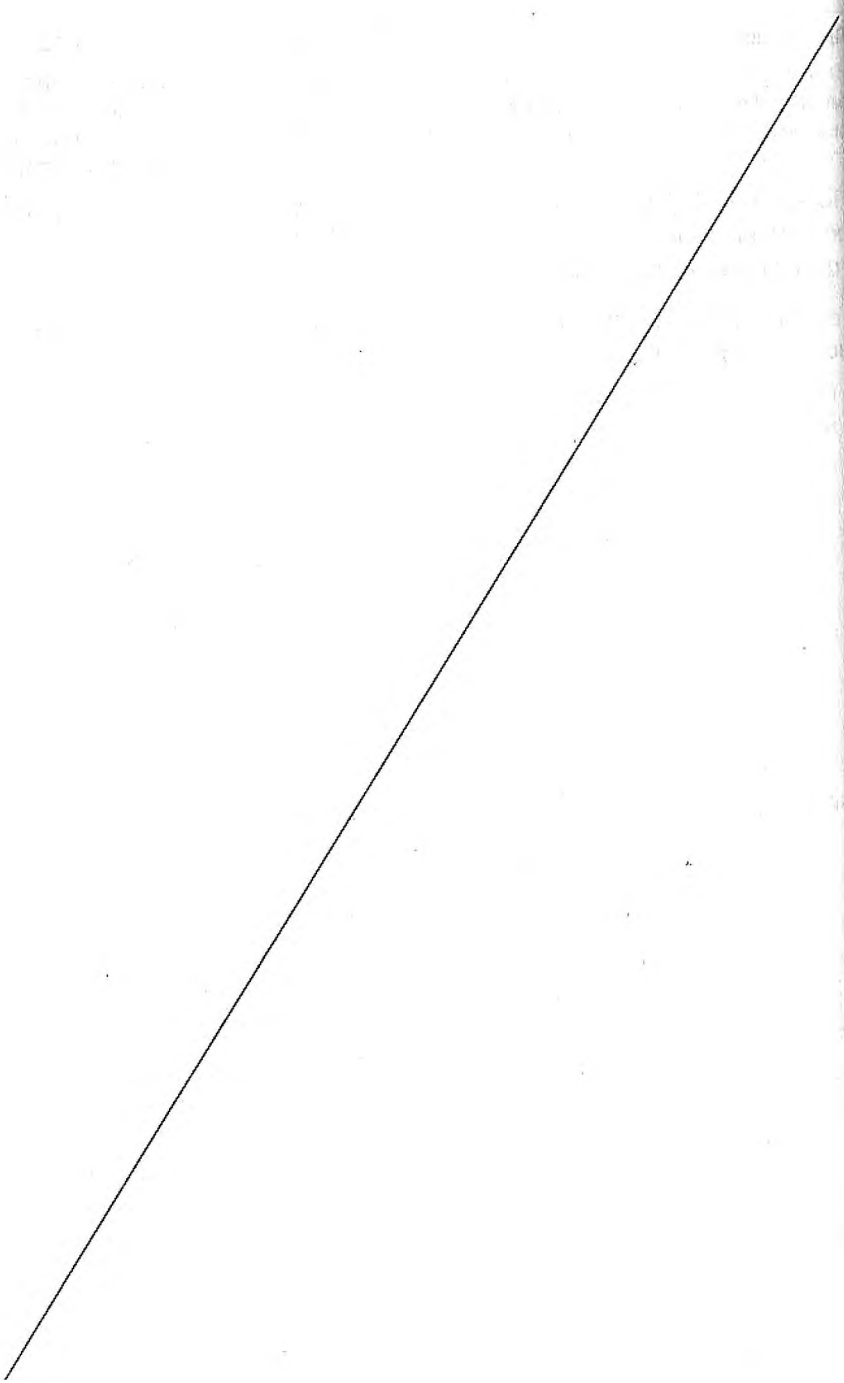
Рис. 3-4. Диспетчер устройств Windows, в котором отображаются все доступные последовательные порты

Примечание

На некоторых компьютерах под управлением Windows порт COM имеет номер больше, чем 9. Такая нумерация создает некоторые проблемы, когда плата Arduino пытается взаимодействовать с портом. Как разрешить эту проблему, можно узнать в главе 7, «Поиск и устранение неполадок».

После того как вы разобрались с назначением порта, можно выбрать его в меню Tools > Serial port (Сервис > Последовательный порт) в интегрированной среде разработки Arduino.

Теперь можно заняться программированием Arduino в интегрированной среде разработки.



Глава 4. Пора приступать к работе с Arduino

В этом разделе вы узнаете, как построить и запрограммировать интерактивное устройство.

Анатомия интерактивного устройства

Все объекты, которые мы будем создавать на основе Arduino, можно объединить одним общим термином — «интерактивные устройства».

Интерактивное устройство — это электронная схема, способная определять состояние внешней среды при помощи датчиков — электронных компонентов, которые преобразуют результаты измерения параметров реального мира в электрические сигналы. Интерактивное устройство обрабатывает информацию, полученную от датчиков, по алгоритму, который определяется программным обеспечением. После этого интерактивное устройство начинает взаимодействовать с окружающей средой с помощью исполнительных механизмов — компонентов, преобразующих электрические сигналы в физическое действие.

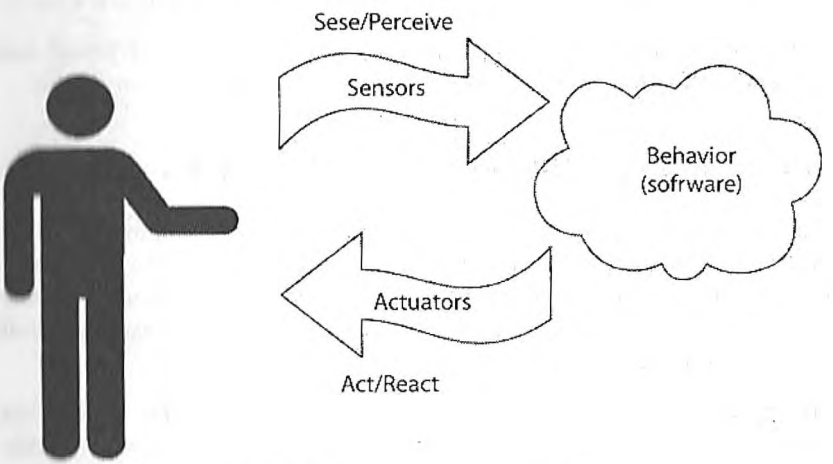


Рис. 4-1. Интерактивное устройство

Датчики и исполнительные механизмы

Датчики и исполнительные механизмы — это электронные компоненты, позволяющие интерактивному устройству взаимодействовать с окружающей средой.

Микроконтроллер представляет собой очень простой компьютер и может обрабатывать только электрические сигналы. Все это чем-то напоминает обработку человеческим мозгом импульсов, передаваемых по нервным волокнам. Чтобы микроконтроллер был в состоянии «ощутить» свет, температуру или другие явления окружающего мира, ему требуются компоненты, которые могли бы преобразовывать все это в электричество. Например, человеческий глаз преобразует свет в сигналы, которые передаются в мозг по нервным волокнам. В электронике для этого можно использовать простое устройство, называемое фоторезистором, которое в зависимости от интенсивности падающего на него света меняет собственное сопротивление, что влияет на проходящий через него ток. Такое изменение электрического тока является сигналом, понятным для микроконтроллера.

Итак, электрические сигналы попадают в микроконтроллер, в котором имеется алгоритм, указывающий, как именно реагировать на полученные сигналы, а собственно реагирование, то есть физическое действие, осуществляется исполнительными механизмами. Примерно по такой же схеме сокращаются наши с вами мускулы, только после получения импульсов из головного мозга. В мире электроники роль мускулов выполняют лампочки, светодиоды, электродвигатели, электромагниты и т.д.

В следующих разделах я расскажу, как считывать сигналы с различных датчиков и как управлять разными исполнительными механизмами.

Включение светодиода в режиме мигания

Для того чтобы определить, работает ли наша плата Arduino и правильно ли она сконфигурирована, давайте попробуем заставить мигать светодиод. Обычно подобное задание играет роль самого первого упражнения, которое выполняют студенты при изучении программирования микроконтроллеров.

Светодиод представляет собой небольшой электронный компонент, внешне напоминающий маленькую электрическую лампочку, от которой он отличается высокой эффективностью, ведь для его работы требуются меньшее напряжение и меньший ток.

Плата Arduino продается с установленным на ней светодиодом, который обозначается буквой «L». Вы можете использовать свой собственный светодиод, подсоединив его так, как показано на рис. 4-2.

«К» обозначает катод (отрицательный вывод), он покороче. «А» обозначает анод (положительный вывод), он подлиннее.

После того как вы подсоединили светодиод, необходимо сообщить плате Arduino, что с ним надо делать. Это делается при помощи кода, то есть списка команд, которые передаются на микроконтроллер для выполнения поставленной задачи.

Откройте на компьютере папку, в которую была скопирована интегрированная среда разработки Arduino. Дважды щелкните по значку Arduino, чтобы запустить интегрированную среду разработки. Выберите

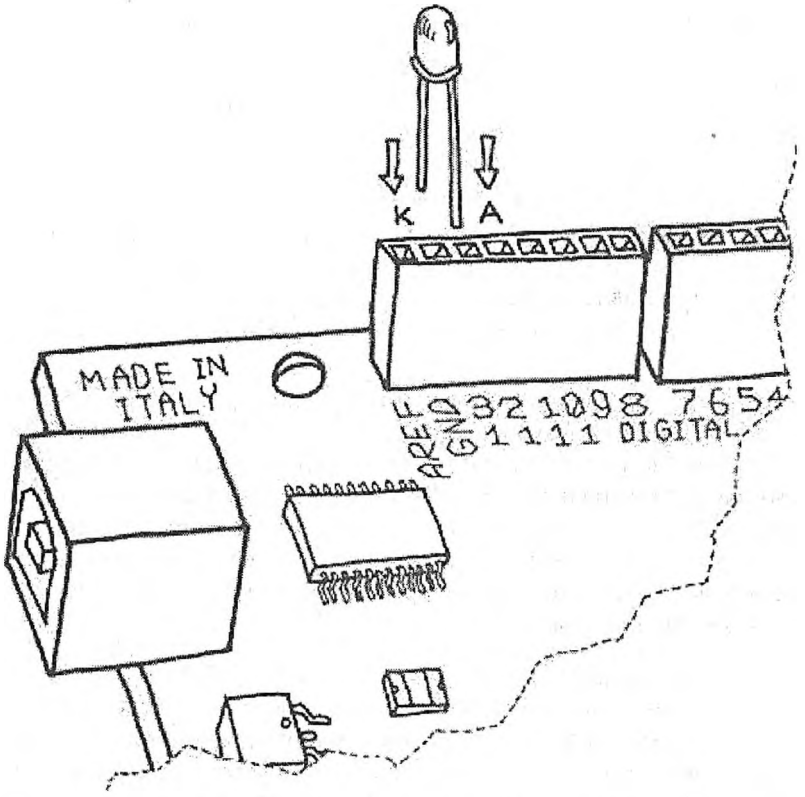


Рис. 4-2. Установка светодиода на плату Arduino

пункты **File > New** (Файл > Создать), после чего на экране появится запрос о выборе имени папки, где будет храниться программный модуль для Arduino. Присвойте ей имя *Blinking_LED* и нажмите кнопку **OK**. Затем введите текст, приведенный в примере 4-1, в редактор Arduino (основное окно интегрированной среды разработки). Этот же текст можно загрузить со страницы www.makezine.com/getstartedarduino. Фрагмент кода должен иметь вид, показанный на рис. 4-3.

Пример 4-1. Мигающий светодиод

```
#define LED 13 // Светодиод подключен
               // к цифровому контакту 13
void setup()
{
  pinMode(LED, OUTPUT); // устанавливает цифровой
                        // контакт в качестве вывода
}
void loop()
{
   digitalWrite(LED, HIGH); // включает светодиод
   delay(1000); // ожидает в течение секунды
   digitalWrite(LED, LOW); // выключает светодиод
   delay(1000); // ожидает в течение секунды
}
```

Теперь, когда код находится в интегрированной среде разработки, необходимо убедиться, что он написан правильно. Нажмите кнопку **Verify** (Проверить) (на рис. 4-3 показано ее расположение). Если все верно, в нижней части окна интегрированной среды разработки появится сообщение **Done compiling** (Компилирование выполнено). Это сообщение означает, что программный модуль оттранслирован в исполняемый код, который может быть загружен в плату (похоже на exe-файл в Windows или app-файл на Mac).

Выгружаем код в плату. Для этого нажимаем кнопку **Upload to I/O Board** (Выгрузить в плату ввода-вывода) (см. рис. 4-3). Это приводит к сбросу платы в исходное состояние, принудительному останову платы и приему исполняемого кода по USB-кабелю. То есть, иными словами, интегрированная среда разработки отправляет написанный нами программный модуль на плату, где он сохраняется в памяти и затем выполняется.

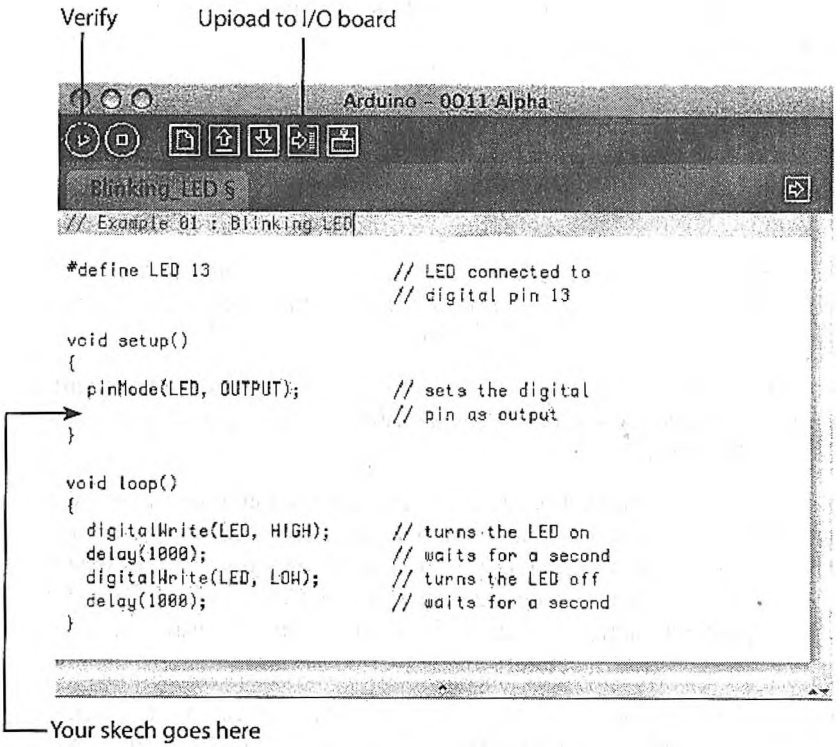


Рис. 4-3. Интегрированная среда разработки Arduino с загруженным программным модулем

Пока все это происходит, внизу окна на черной плашке выводится несколько сообщений, затем непосредственно над этой плашкой появляется сообщение **Done uploading** (Выгрузка выполнена), то есть процедура закончилась без ошибок.

Два находящихся на плате светодиода «RX» и «TX» зажигаются каждый раз, когда плата передает или принимает очередной байт, поэтому во время выгрузки кода они постоянно мигают.

Если светодиоды не мигают или вместо сообщения **Done uploading** выводится сообщение об ошибке, значит, компьютер и плата взаимодействуют неправильно. Убедитесь, что последовательный порт выбран верно (см. главу 3), для чего воспользуйтесь пунктами меню **Tools > Serial Port** (Сервис > Последовательный порт). Кроме того, проверьте меню **Tools > Board** (Сервис > Плата), чтобы убедиться в том, что вы правильно указали модель Arduino.

Если проблему разрешить не удалось, обратитесь к главе 7, «Поиск и устранение неполадок».

После того как мы загрузили исполнимый код на плату Arduino, он остается там до тех пор, пока мы туда не загрузим другой программный модуль. Кстати, код не исчезает, если плата сбрасывается в исходное состояние или выключается. Примерно так же ведут себя данные на жестком диске компьютера.

Если код загружен правильно, светодиод «L» должен начать мигать — секунду гореть, секунду не гореть. Если вы поставили свой светодиод как на рис. 4-2, он тоже будет мигать.

То, что вы только что сделали, называется «составлением и выполнением программного модуля». В случае Arduino программные модули называются «скетчами».

Как я уже говорил, плата Arduino — это небольшой компьютер, значит, его можно запрограммировать на решение какой-то задачи. Набор последовательности инструкций выполняется на языке программирования в интегрированной среде разработки (IDE), которая, в свою очередь, превращает эту последовательность в исполнимый код.

Разумеется, я объясню вам, как разобраться в том, что написано в тексте скетча. Прежде всего, вам надо знать, что Arduino выполняет код сверху вниз, то есть первая строка скетча будет прочитана первой. Затем выполняется вторая строка, следом — третья и т.д., примерно так, как последовательно исполняются музыкальные композиции или видеоролики в программах QuickTime Player или Windows Media Player.

Передайте мне пармезан, пожалуйста

Обратите внимание на фигурные скобки, при помощи которых строки кода группируются в отдельные блоки. Эти скобки особенно полезны, когда нам требуется присвоить имя какой-то группе команд. Объясняю на примере. Предположим, вы обедаете в ресторане и просите соседа: «Передайте мне, пожалуйста, сыр пармезан». Ваша просьба запускает последовательность действий, обобщенных небольшой фразой, которую вы только что произнесли. Но мы-то с вами люди, и для нас как подобная просьба, так и ответное действие вполне естественны. Для того чтобы заставить что-то делать Arduino, возможности которого неизмеримо скромнее, чем у человеческого мозга, ему необходимо каждое действие разложить на элементы, мельчайшие шаги. Каждая команда

скетча — это такой шажок, а группа команд, которая начинается открывающей фигурной скобкой «{» и заканчивается закрывающей фигурной скобкой «}», представляет собой описание определенного действия.

Если вы заметили, в фигурные скобки были заключены два блока кода, перед каждым из которых стояла команда `void setup()`, при помощи которой блоку присваивается имя.

Если бы мы написали программу, которая заставила бы Arduino передать вам пармезан, то в начале блока следовало бы вставить строку `void passTheParmesan()`, и данный блок можно было бы вызвать из любого места в скетче Arduino. Такие поименованные блоки называются функциями.

Что значит вызвать из любого места? Это просто. Если написать в любом месте программы `passTheParmesan()`, Arduino будет выполнять основной код, потом, дойдя до этого места, выполнит команды данной функции, а затем продолжит выполнение основного кода.

Arduino не для лодырей

Для работы Arduino необходимы две функции: одна называется `setup()`, а другая — `loop()`.

Функция `setup()` — это контейнер, куда помещается весь код, который должен однократно выполняться в начале программы, а функция `loop()` содержит ядро программы, которое выполняется многократно. Это делается потому, что плата Arduino не похожа на стандартный компьютер. Во-первых, она не способна выполнять несколько программ одновременно, а во-вторых, программа, запущенная на Arduino, не может завершить работу самостоятельно. Загруженная программа начинает выполняться при подаче питания. Когда необходимо остановить работу, следует просто выключить питание.

Фанаты тинкеринга всегда пишут комментарии

Любой программный текст, начинающийся с двух наклонных черточек `//`, Arduino игнорирует. Этими двумя наклонными оформляются так называемые комментарии, то есть сообщения, которые пользователь оставляет в тексте программы для самого себя, чтобы потом можно было вспомнить, что именно он делал, когда писал этот фрагмент кода. Комментарии могут быть полезны и для кого-то еще, кто хотел бы разобраться в вашем коде.

Как правило, пользователи пишут программный модуль, выгружают его на плату, облегченно вздыхают и говорят примерно следующее: «Все! Что бы я еще раз притронулся к этой фигне!» (я-то знаю, потому что сам постоянно так делаю). А полгода спустя выясняется, что с модулем что-то не так: его хорошо бы обновить или исправить в нем мелкую погрешность. Тогда вы открываете текст программы и обнаруживаете, что комментариев нет. Ваша реакция в лучшем случае будет такой: «Е... лки зеленые, вот это бардак! С чего теперь начинать-то?»

Поэтому, по мере того, как мы будем продвигаться дальше, я научу вас некоторым трюкам, которые помогут сделать программы более удобочитаемыми и пригодными для сопровождения.

Программный код, шаг за шагом

Возможно, вы посчитаете этот раздел ненужным, как ученик итальянской средней школы, которого заставляют учить «Божественную комедию» Данте или еще одну (действительно кошмарную) книгу под названием «Обрученная». Для каждой строки этих сочинений литературоведы написали десятки страниц комментариев.

Однако, когда вы приступите к составлению собственных программных модулей, вы увидите, насколько полезны пояснения.

```
// Пример 01 : Мигающий светодиод
```

Комментарий — это ведь не что иное, как удобный способ организации небольших заметок. Приведенный выше комментарий, содержащий заголовок, напоминает нам, что данная программа (Пример 4-1) реализует мигание светодиода.

```
#define LED 13 // Светодиод подключен к
               // цифровому контакту 13
```

Выражение `#define` в программном коде работает как автоматический поиск и замена. В данном случае оно заставляет Arduino записывать число 13 всякий раз, когда в тексте встречается слово LED.

(Кстати, функция замены выполняется, если щелкнуть по кнопке **Verify** или **Upload to I/O Board**. Результаты замены не видны, так как все делается скрытно.)

Итак, команда `#define` указывает, что светодиод (LED), который мы хотим заставить мигать, подключен к контакту 13 на плате Arduino.

```
voidsetup()
```

Эта строка сообщает Arduino, что следующий блок кода будет вызываться функцией `setup()`.

Блок начинается с открывающей фигурной скобки.

```
{
pinMode(LED, OUTPUT); // установка цифрового контакта
                      // в качестве выхода
```

Вот здесь уже интересно! Команда `pinMode` говорит Arduino, как надо сконфигурировать определенный контакт. Вы уже знаете, что цифровые контакты могут использоваться или как `INPUT` (ВХОД), или как `OUTPUT` (ВЫХОД). Для управления светодиодом нам требуется выходной контакт, так что мы помещаем номер контакта и его режим работы внутри круглых скобок.

Итак, `pinMode` — это функция, а слова или числа, которые задаются внутри круглых скобок, называются аргументами. `INPUT` и `OUTPUT` в языке Arduino являются константами. Константам, как и переменным, присваиваются значения, однако значения констант предопределены и никогда не меняются.

```
}
```

Такая закрывающая фигурная скобка означает конец функции `setup()`.

```
void loop()
{
```

`loop()` — место в программе, где вы задаете поведение интерактивного устройства. Эта функция будет повторяться снова и снова до тех пор, пока вы не выключите напряжение.

```
digitalWrite(LED, HIGH);  // включение светодиода (LED)
```

Как видите, в комментарии говорится, что функция `digitalWrite()` позволяет включить (или выключить) любой контакт, который был сконфигурирован как `OUTPUT` (ВЫХОД). Первый аргумент (в данном случае `LED`) задает, какой контакт должен быть включен или выключен (помните, что `LED` — это константа, ссылающаяся на 13-й как на переключающийся контакт). Второй аргумент позволяет включать (`HIGH`) или выключать (`LOW`) напряжение на этом контакте.

Представьте, что каждый выходной контакт — это крошечная штепсельная розетка, подобная тем, что стоят на стенках в вашей квартире. В Европе на них подается 230 В, в Америке — 110 В, плата Arduino работает на более «скромном» напряжении 5 В.

Волшебство начинается, когда программное обеспечение «одушевляет» аппаратуру. Функция `digitalWrite()` заставляет плату подать на выходной контакт 5 В, и подсоединенный к нему светодиод загорится. Выходит, что в данном месте программы «нематериальная» команда что-то меняет в «материальном» мире, перенаправляя электрический ток к выбранному нами контакту.

Появление или исчезновение напряжения на контакте позволит нам получить результат, который человек сможет воспринимать при помощи органов чувств, в данном конкретном случае — зрения. В роли исполнительного механизма выступает светодиод.

```
delay(1000);  // ожидание в течение секунды
```

По сути, Arduino представляет собой довольно-таки простое устройство. Поэтому, если требуется, чтобы плата «замерла» в определенном состоянии, надо ей приказать «сидеть тихо и не шевелиться» до тех пор, пока не настанет время перейти к следующему шагу. Функция `delay()`, по существу, отдает процессору команду «замереть» и ничего не делать в течение того времени в миллисекундах, которое задается в качестве аргумента. Миллисекунды — это тысячные доли секунды, поэтому 1000 миллисекунд равны 1 секунде. Итак, светодиод сохраняет свое состояние в течение одной секунды.

```
digitalWrite(LED, LOW);  // выключение светодиода (LED)
```

С помощью этой команды ранее включенный светодиод (`LED`) выключается. Почему мы используем аргументы `HIGH` (ВЫСОКИЙ) и `LOW` (НИЗКИЙ)? Просто потому, что такое соглашение давно принято в цифровой электронике. `HIGH` означает, что контакт включен и Arduino подает на контакт 5 В. `LOW` означает 0 В. Эти аргументы соответствуют стандартным надписям на любом бытовом приборе: `ON` (Вкл.) и `OFF` (Выкл.).

```
delay(1000);  // ожидание в течение секунды
```

Здесь выполняется задержка еще на одну секунду, значит, светодиод будет в течение этого промежутка времени находиться в выключенном состоянии.

```
}
```

Данная закрывающая фигурная скобка отмечает конец функции `loop`.

Если разобраться, описанный фрагмент кода выполняет следующие действия:

- превращение контакта 13 в выход (только один раз вначале);
- запуск цикла loop;

- включение светодиода (LED), подключенного к контакту 13;
- ожидание в течение секунды;
- выключение светодиода (LED), подключенного к контакту 13;
- ожидание в течение секунды;
- переход в начало цикла loop.

Я надеюсь, что все это не стало для вас мучительным занятием. Если нет, то вы узнаете о том, как надо программировать, ознакомившись с остальными примерами.

Прежде чем перейти к следующему разделу, я хочу, чтобы вы поэкспериментировали с кодом. Попробуйте уменьшить время задержки, используя другие числа в качестве аргументов, чтобы увидеть разные варианты поведения светодиода. В частности, если задать промежутки времени очень малыми, но разными для включенного и выключенного состояний, настанет момент, когда произойдет нечто странное. Это «нечто» окажется очень полезным при изучении широтно-импульсной модуляции.

Что мы создадим?

Мне всегда нравилось заниматься построением систем управления источниками света. Можно сказать, что мне в некотором смысле повезло, ведь я работал над несколькими интереснейшими проектами в данной сфере.

Платформа Arduino очень хороша для управления источниками света, поэтому на протяжении всего повествования мы будем заниматься конструированием светильников, используя нашу плату для изучения основ построения интерактивных устройств.

В следующем разделе я постараюсь объяснить читателям основы электричества. Возможно, моя методика неприемлема для инженера, однако она не напугает начинающего программиста Arduino.

Что такое электричество?

Если вам приходилось паять что-нибудь у себя дома, значит, вы понимаете, что такое электроника. Для всех остальных объясняю. Лучше всего нам поможет разобраться в том, что такое электричество и электрические цепи, так называемая «водяная аналогия». Давайте возьмем портативный вентилятор. Если разобрать его на части, можно увидеть, что там есть батарейка, пара проводов, электродвигатель и кнопка выключателя (рис. 4-4).

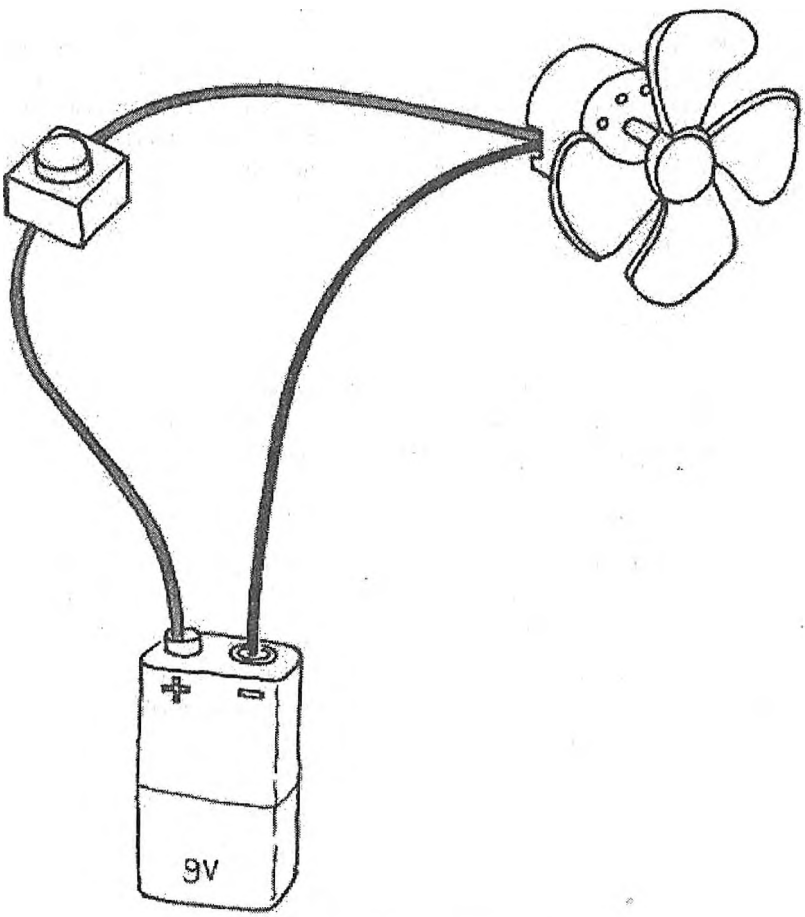


Рис. 4-4. Портативный вентилятор

Если батарейка свежая и кнопка включена, электродвигатель начинает крутить крыльчатку, которая обеспечивает нам желанную прохладу.

Как все это работает? Представьте себе, что батарейка — это резервуар с водой и насос, выключатель — это кран, а электродвигатель — колесо водяной мельницы. Открываете кран, качаете насос, и вода приводит колесо в движение.

В простой гидравлической системе, показанной на рис. 4-5, важны два фактора: давление воды (оно определяется мощностью насоса) и количество воды, которое протекает в трубах (оно зависит от диаметра и длины труб и сопротивления, которое колесо оказывает падающему на него потоку воды).

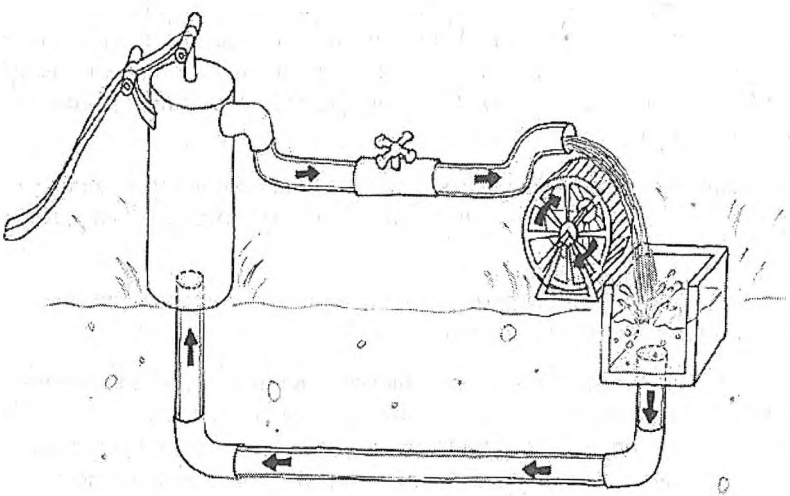


Рис. 4-5. Гидравлическая система

Если требуется ускорить вращение колеса, необходимо увеличить диаметр трубы и увеличить давление, которое может обеспечить насос. При увеличении диаметра трубы растет объем протекающей через нее воды. Устанавливая трубу большего сечения, мы практически уменьшаем ее сопротивление потоку воды. Все это работает до некоторого предела, после которого колесо перестает вращаться быстрее, поскольку давления воды уже недостаточно. Это значит, что для увеличения скорости вращения колеса надо установить более мощный насос. В конце концов, можно дойти до критической отметки, когда колесо разлетится на куски, потому что поток воды его просто разрушит.

Надо учитывать еще одно обстоятельство. При вращении ось колеса будет немного нагреваться, так как независимо от того, насколько хорошо установлено колесо, его ось испытывает трение в отверстиях, куда она вставлена, а если есть трение, значит, выделяется тепло. Как ни крути, в любой системе не вся подводимая энергия преобразуется в движение, часть ее переходит в тепло.

Итак, какие факторы играли в нашей системе главные роли? Разумеется, давление, создаваемое насосом, сопротивление, обусловленное трубами и колесом, и собственно поток воды, то есть количество литров воды, протекающей по системе за одну секунду.

Электричество в проводах ведет себя примерно так же, как вода в трубах. Есть нечто вроде насоса (источник электричества, такой как батарейка или сетевая розетка), который проталкивает электрические заряды («капли» электричества) по трубам (проводам). Электрические устройства способны превращать ток в тепло (электроплита на даче), свет (торшер в гостиной), звук (стереосистема в автомобиле), движение (вентилятор на кухне) и многое другое.

Таким образом, когда вы видите надпись на батарейке «Напряжение 9 В», представьте себе, что это давление воды, которое потенциально создает этот маленький «насос».

Кстати, напряжение измеряется в вольтах в честь Алессандро Вольта, изобретателя первой батарейки.

Если давление воды имеет электрический эквивалент, то для скорости потока воды также существует аналог, который называется током и измеряется в амперах в честь Андре-Мари Ампера, первого исследователя электромагнетизма. Взаимосвязь между напряжением и током можно проиллюстрировать, снова обратившись к водяному колесу: более высокое напряжение (давление) позволяет вращать колесо быстрее; более высокая скорость потока (ток) тоже увеличивает скорость вращения колеса.

Наконец, существует противодействие электрическому току, которое называется сопротивлением и измеряется в омах в честь немецкого физика Георга Ома. Между прочим, господин Ом сформулировал самый важный закон в электричестве и единственную формулу, которую действительно стоит запомнить. Он сумел показать, что в электрической цепи напряжение, ток и сопротивление связаны друг с другом, что сопротивление цепи определяет величину тока, протекающего в ней, при заданном напряжении.

Если вдуматься, то этот закон представляется интуитивно понятным. Возьмем батарею на 9 В и подключим ее к цепи. Измеряя ток, вы обнаружите, что чем больше сопротивлений добавить в цепь, тем меньше будет ток. Вернемся к водяной аналогии: если следом за насосом установить клапан (который можно уподобить переменному сопротивлению в теории электричества), то чем больше закрывать клапан, увеличивая сопротивление потоку воды, тем меньше воды будет протекать по трубам.

Ом обобщил свой закон в следующей формуле:

I (ток) = *V* (напряжение) / *R* (сопротивление)

Повторяю, это единственное правило, которое следует запомнить. Вам надо научиться им пользоваться, так как в большинстве случаев только этот закон будет на самом деле необходим.

Использование кнопки для управления светодиодом

Заставить светодиод мигать было просто, однако я сомневаюсь, что вы останетесь в здравом уме, если настольная лампочка будет постоянно мигать во время чтения этой книги. Поэтому необходимо знать, как управлять этим хозяйством. В предыдущем примере светодиод был исполнительным механизмом, а плата Arduino управляла им. Для полноты картины нам не хватало датчика.

Сейчас мы попытаемся использовать простейший вид датчика: кнопку.

Если вы разберете кнопку, то увидите, как просто она устроена: два металлических контакта, разделенные пружиной, и пластмассовый колпачок, который при нажатии обеспечивает соприкосновение контактов. Когда контакты разъединены, ток через кнопку не идет (клапан закрыт). Когда кнопку нажимают, контакт есть.

Для контроля состояния переключателя в Arduino существует функция `digitalRead()`, которая проверяет, подается ли какое-то напряжение на контакт, который указывается в скобках. При этом в зависимости от результата проверки функция возвращает значение `HIGH` (ВЫСОКОЕ) или `LOW` (НИЗКОЕ).

Если помните, другие команды, которые мы использовали до сих пор, не возвращали никакой информации, они просто выполняли то, что от них требовалось. Функции такого типа ограниченны, так как они вы-

нуждают нас следовать строго определенной последовательности команд без использования информации из внешнего мира. С помощью функции `digitalRead()` вы «задаете вопрос» плате Arduino и получаете ответ, который можно сохранить где-нибудь в памяти и потом воспользоваться им для принятия решений.

Соберите схему, показанную на рис. 4-6. Чтобы сделать это, вам придется приобрести некоторые детали, которые пригодятся при работе и над другими проектами.

- Беспаечная макетная плата. Рекомендуется использовать плату www.amperka.ru/breadboard. (См. приложение «А» - введение в беспаечные макетные платы.)
- Набор готовых перемычек. Рекомендуется использовать набор www.amperka.ru/jumper-wires

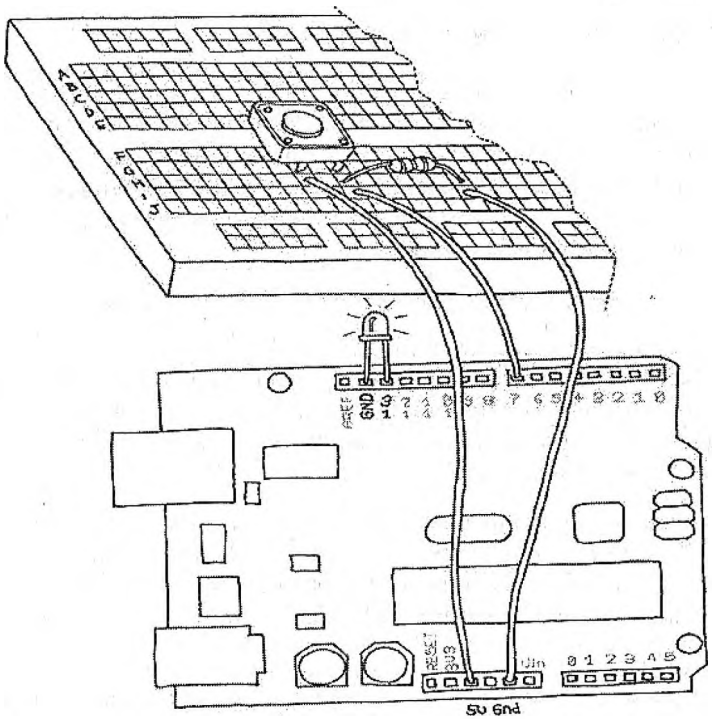


Рис. 4-6. Подключение кнопки

- Один резистор на 10 кОм. Можно использовать резистор www.amperka.ru/resistors
- Сенсорный кнопочный переключатель мгновенного действия. www.amperka.ru/button

---

Примечание

Вместо готовых проволочных перемычек можно взять монтажный провод со сплошной жилой, порезать его на кусочки нужного размера, а концы зачистить с помощью кусачек или инструмента для снятия изоляции.

Давайте посмотрим на программный код, который будет управлять светодиодом при помощи кнопки:

Пример 4-2. Включение светодиода нажатием кнопки

```
#define LED 13 // контакт для светодиода
#define BUTTON 7 // входной контакт, к которому
                 // подключается кнопка
int val = 0; // val используется для хранения состояния
             // входного контакта
void setup() {
   pinMode(LED, OUTPUT); // сообщает Arduino,
      // что светодиод (LED) является выходом,
   pinMode(BUTTON, INPUT); // и BUTTON является входом
}
void loop(){
   val = digitalRead(BUTTON); // считывает и сохраняет
   // входное значение проверяет, равен ли
   // вход значению HIGH (кнопка нажата)
   if (val == HIGH) {
      digitalWrite(LED, HIGH); // включает светодиод
} else {
      digitalWrite(LED, LOW);
   }
}
```

На экране компьютера в интегрированной среде разработки выберите пункты меню **File > New** (Файл > Создать) (если открыта другая программа, лучше сначала сохраните ее). Когда платформа Arduino выводит запрос о присвоении имени новому скетчу, введите с клавиатуры **PushButtonControl**. Наберите с клавиатуры (или загрузите с вебстраницы www.makezine.com/getstartedarduino и вставьте в интегрированную среду разработки) код примера 4-2. Если все правильно, светодиод загорится при нажатии кнопки.

Как это работает?

В данном примере я ввел два новых понятия: функцию, возвращающую результат, и оператор if.

Оператор `if`, возможно, самая важная команда, так как она позволяет компьютеру (всегда помните, что Arduino — это маленький компьютер) принимать решения. После ключевого слова `if` следует внутри круглых скобок написать «вопрос», и если «ответ» будет истиной, начнет выполняться первый блок программного кода. В противном случае будет выполняться блок кода, расположенный после ключевого слова `else`. Обратите внимание на то, что я использовал символ «==» вместо символа «=». Первый символ применяется для сравнения двух сущностей, при этом возвращается значение `TRUE` или `FALSE`. С помощью второго символа присваивают значение переменной. Убедитесь, что вы используете правильный символ, ведь ошибку сделать очень легко, а программа работать не будет. Я это знаю, потому что после 25 лет программирования по-прежнему делаю эту ошибку.

Однако вернемся к нашей схеме. Держать палец на кнопке все время, пока вам нужен свет, непрактично. Может быть, ее стоило бы приклеить? Давайте сделаем это программным путем.

Тысяча вариантов поведения одной электрической цепи

Сейчас я продемонстрирую вам огромное преимущество цифровой программируемой электроники над электроникой классической. Мы попробуем реализовать множество различных вариантов поведения одной и той же электронной цепи, просто изменяя программный код.

Итак, нам стало ясно, что не очень практично держать палец на кнопке, когда вам надо, чтобы светодиод горел. Поэтому мы должны придумать программный механизм, который запомнит, что кнопка нажата, и оставит свет включенным после того, как мы отпустим кнопку.

Чтобы сделать это, мы будем использовать так называемую переменную. (Мы уже использовали переменную, но я не объяснил ее назначение.) Грубо говоря, переменная — это место в памяти Arduino, где можно хранить данные. Представьте себе, что это стикер, который вы приклеиваете на монитор или холодильник для того, чтобы не забыть позвонить кому-то. В языке Arduino это так же просто: вы всего лишь решаете, какой тип данных требуется хранить (например, число или какой-нибудь текст), присваиваете этим данным имя и, когда требуется, сохраняете или извлекаете их.

Пример:

```
int val = 0;
```

`int` означает, что в переменной будет храниться целое число, `val` — имя переменной и выражение `= 0` присваивает переменной начальное значение, равное нулю.

Переменная может стоять в любом месте программного кода, так что где-то дальше в программе можно было бы написать выражение:

```
val = 112;
```

которое присваивает переменной значение 112.

Примечание

Если вы заметили, в Arduino каждый оператор, за исключением `#define`, завершается точкой с запятой. Это делается для того, чтобы компилятору Arduino, превращающему скетч в программу, исполняемую микроконтроллером, было понятно, что здесь закончился один оператор и начался другой. Поэтому не забывайте постоянно ставить точку с запятой, исключая все строки, которые начинаются с выражения `#define`. Выражения `#define` заменяются компилятором еще до того, как скетч транслируется в исполнимый код.

В следующей программе переменная val используется для хранения результата выполнения команды `digitalRead()`. Теперь все, что Arduino получает на входных контактах, становится значением этой переменной и сохраняется до тех пор, пока в другой строке программы переменная не будет изменена. Обратите внимание, что переменные используют память, которая называется ОЗУ (оперативное запоминающее устройство, RAM). Это достаточно быстрая память, однако при выключении платы все данные, хранящиеся в ОЗУ, будут потеряны. Это значит, что когда на плату снова будет подано напряжение, все переменные примут исходные значения. Сами программы хранятся во флеш-памяти, примерно такой же, что используется в мобильном теле-

фоне для хранения телефонных номеров. Здесь содержимое остается даже при выключенном питании.

Давайте воспользуемся еще одной переменной, чтобы запомнить, должен ли светодиод (LED) остаться включенным или выключенным после отпускания кнопки.

Итак, первая попытка, пример 4-3:

Пример 4-3. Включение светодиода нажатием кнопки и обеспечение его работы после отпускания кнопки

```
#define LED 13 // контакт для светодиода (LED)
#define BUTTON 7 // входной контакт, к которому
                 // подсоединена кнопка
int val = 0; // val используется для хранения состояния
             // входного контакта
int state = 0; // 0 = LED светодиод выключен,
               // тогда как при 1 = LED светодиод включен
void setup() {
   pinMode(LED, OUTPUT); // сообщаем Arduino,
                         // что светодиод (LED) является выходом
   pinMode(BUTTON, INPUT); // и BUTTON (кнопка)
                           // является входом
}
void loop() {
   val = digitalRead(BUTTON); // считывает и сохраняет
   // входное значение
   // проверяет, равно ли входное значение
   // HIGH (кнопка нажата) и изменяет состояние
   if (val == HIGH) {
      state = 1 - state;
   }
   if (state == 1) {
     digitalWrite(LED, HIGH); // включается светодиод (LED)
   } else {
     digitalWrite(LED, LOW);
   }
}
```

Если вы протестируете этот программный модуль, то увидите, что он... вроде бы работает. Правда, обнаруживается, что свет мигает так быстро, что физически невозможно его включить или выключить нажатием кнопки.

Давайте рассмотрим наиболее интересные фрагменты кода: `state` — переменная, которая принимает значение «`0`» или «`1`» в зависимости от того, включен или выключен светодиод. После отпускания кнопки переменная принимает значение «`0`» (светодиод выключен).

Затем мы определяем текущее состояние кнопки, и если она нажата (`val = HIGH`), изменяем состояние с «`0`» на «`1`» или наоборот. Поскольку состояние кнопки может принимать только такие значения, мы сделаем небольшой трюк, основанный на том, что 1 – 0 = 1, а 1 – 1 = 0:

```
state = 1 – state;
```

Эта строка, возможно, не имеет большого смысла в школьном курсе алгебры, но в программировании дело обстоит иначе. Символ «=» означает «присвоить переменной, имя которой находится до данного знака, результат того, что стоит после этого знака». В нашем случае переменной присваивается значение «`1`» минус предыдущее значение переменной.

Далее в программе можно увидеть, что мы используем оператор `state`, чтобы выяснить, должен ли светодиод включаться или выключаться. Как я уже говорил, это приводит к несколько странным результатам. А все из-за способа считывания состояния кнопки.

Дело в том, что платформа Arduino очень быстрая. Она работает со скоростью 16 миллионов операций в секунду, иными словами, за одну секунду она способна обрабатывать несколько миллионов строк программного кода. Выходит, что пока вы нажимаете пальцем кнопку, Arduino в состоянии несколько тысяч раз определить ее положение и соответствующим образом изменить значение переменной `state`. Получается, что результат непредсказуем, ведь состояние кнопки может быть «выключено», когда требуется, чтобы было «включено», и наоборот. Но поскольку даже сломанные часы дважды в сутки показывают правильное время, возможно, что и наша программа иногда будет демонстрировать ожидаемые результаты, что звучит обнадеживающе, но и только.

Как же все это исправить? На самом деле нам надо точно определить момент нажатия кнопки, потому что именно тогда должно изменяться состояние. Я предпочитаю делать это следующим образом: сохранять значение `val` до считывания нового значения. Это позволяет сравнивать текущее положение кнопки с ее предыдущим состоянием и изменять его только тогда, когда состояние кнопки становится равным значению `HIGH` после пребывания в значении `LOW`.

В примере 4-4 я привел соответствующий программный код:

Пример 4-4. Включение светодиода нажатием кнопки и обеспечение его работы после отпускания кнопки. Новая усовершенствованная формула!

```
#define LED 13    // контакт для светодиода (LED)
#define BUTTON 7 // входной контакт, к которому
                 // подсоединена кнопка
int val = 0;     // val используется для хранения состояния
                 // входного контакта
int old_val = 0; // в этой переменной хранится предыдущее
                 // значение "val"
int state = 0;   // 0 = LED светодиод выключен
                 // и 1 = LED светодиод включен
void setup() {
   pinMode(LED, OUTPUT); // сообщаем Arduino,
                         // что светодиод (LED) является выходом
   pinMode(BUTTON, INPUT); // и BUTTON (Кнопка)
                           // является входом
}
void loop(){
   val = digitalRead(BUTTON); // считывается и сохраняется
                              // входное значение
                              // класс, свежее решение

   // проверяется, был ли переход
   if ((val == HIGH) && (old_val == LOW)){
     state = 1 - state;
   }
   old_val = val; // val сейчас прежнее, давайте сохраним его
   if (state == 1) {
     digitalWrite(LED, HIGH); // включается светодиод
   } else {
     digitalWrite(LED, LOW);
   }
}
```

Если вы протестируйте этот код, то увидите, что почти достигли цели.

Почему почти? Дело в том, что нам пока не удалось справиться с одной проблемой, присущей механическим переключателям. Кнопка — устройство примитивное: две металлические пластинки, разделенные пружинкой. Когда кнопку нажимают, два контакта соединяются, ток пошел. Все это звучит замечательно, но в реальности соединение не получается идеальным. Если кнопка нажата не полностью, она успевает выдать несколько ложных сигналов, которые в электронике называются «дребезгом».

Когда кнопка «дребезжит», Arduino регистрирует очень быструю последовательность сигналов включения и выключения. Существует множество методов борьбы с дребезгом, но для нашего простого фрагмента я предлагаю добавить задержку в диапазоне 10–50 миллисекунд, когда программный код обнаруживает момент перехода.

В примере 4-5 приводится итоговый вариант:

Пример 4-5. Включение светодиода нажатием кнопки и обеспечение его работы после отпускания кнопки. Простой способ борьбы с «дребезгом»! Еще одна новейшая усовершенствованная формула!

```
#define LED 13    // контакт для светодиода (LED)
#define BUTTON 7 // входной контакт, к которому
                 // подсоединена кнопка
int val = 0;     // val используется для хранения состояния
                  // входного контакта
int old_val = 0; // в этой переменной хранится предыдущее
                  // значение "val"
int state = 0;    // 0 = LED светодиод выключен
                  // и 1 = LED светодиод включен
void setup() {
   pinMode(LED, OUTPUT); // сообщаем Arduino,
                         // что светодиод (LED)
                         // является выходом
   pinMode(BUTTON, INPUT); // и BUTTON (Кнопка)
                           // является входом
}
```

```
void loop(){
    val = digitalRead(BUTTON); // считывается
                               // и сохраняется
                               // входное значение
                               // класс, свежее решение

    // проверяется, был ли переход
    if ((val == HIGH) && (old_val == LOW)){
      state = 1 - state;
      delay(10);
}
    old_val = val; // val сейчас прежнее,
                   // давайте сохраним его
    if (state == 1) {
      digitalWrite(LED, HIGH); // включается светодиод
    } else {
      digitalWrite(LED, LOW);
    }
}
```

Глава 5. Усовершенствованные вход и выход

В главе 4 мы познакомились с простейшими действиями, которые способна выполнять плата Arduino: управлять цифровым выходом и считывать значения цифрового входа. Если бы она понимала какой-нибудь человеческий язык, все наши скетчи можно было бы заменить двумя буквами алфавита. Если предположить, что в алфавите не две буквы, а пять, можно сделать вывод о том, сколько работы еще предстоит, прежде чем из-под нашего пера выйдет целая поэма для Arduino!

Пробуем датчики включения/выключения

Итак, мы научились использовать кнопку. Однако на свете есть множество устройств, соединяющих контакты, и самых разных датчиков, которые могут нам пригодиться.

Переключатели

Действуют примерно так же, как и кнопки, но не разрывают контакт при отпускании.

Термостаты

Переключатель, который размыкает контакт, когда температура достигает установленного значения.

Магнитные переключатели (герконы)

Два контакта, которые соединяются вблизи магнита. Используются в охранной сигнализации (контакт размыкается при открывании окна или двери).

Ковровые переключатели

Маленькие плоские датчики, которые можно класть под ковер или половик. Замыкают или размыкают контакты, если на них наступит человек или, к примеру, упитанный кот.

Качающиеся переключатели

Простое электронное устройство, внутри которого находятся два контакта и небольшой металлический шарик (или шарик ртути, но пользоваться такими переключателями я не рекомендую, здоровья это не прибавит). Пример качающегося переключателя — датчик наклона. На рис. 5-1 показано, что у него внутри. Когда датчик находится в вертикальном положении, шарик замыкает контакты (кнопка нажата). Если датчик наклонить, шарик перекатывается, и контакт размыкается (кнопку отпустили). Используя такой датчик, можно реализовать, например, интерфейсы, которые реагируют на перемещение или тряску.

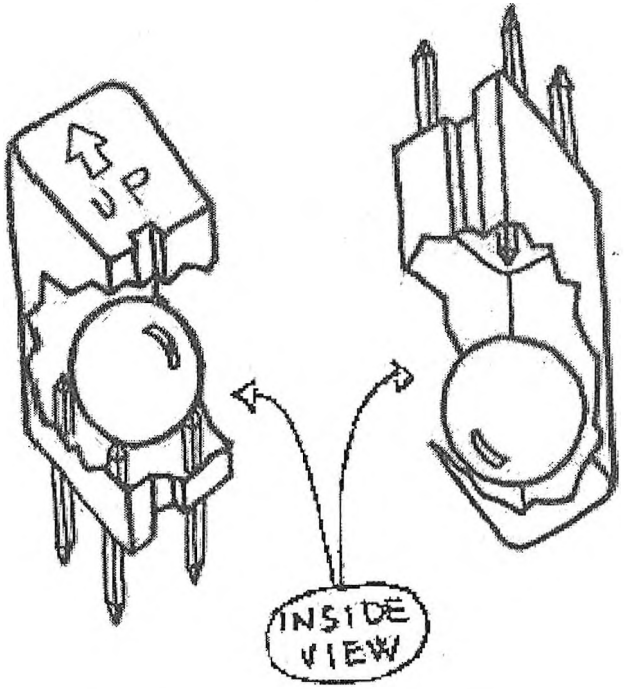


Рис. 5-1. Качающийся датчик, вид изнутри

В охранной сигнализации часто применяется датчик инфракрасного излучения (пассивный ИК-датчик или ПИК-датчик, см. рис. 5-2), который вы, возможно, тоже захотите опробовать. Он срабатывает, когда поблизости оказывается человек или какое-нибудь животное. При помощи такого датчика можно построить систему, реагирующую на движение.

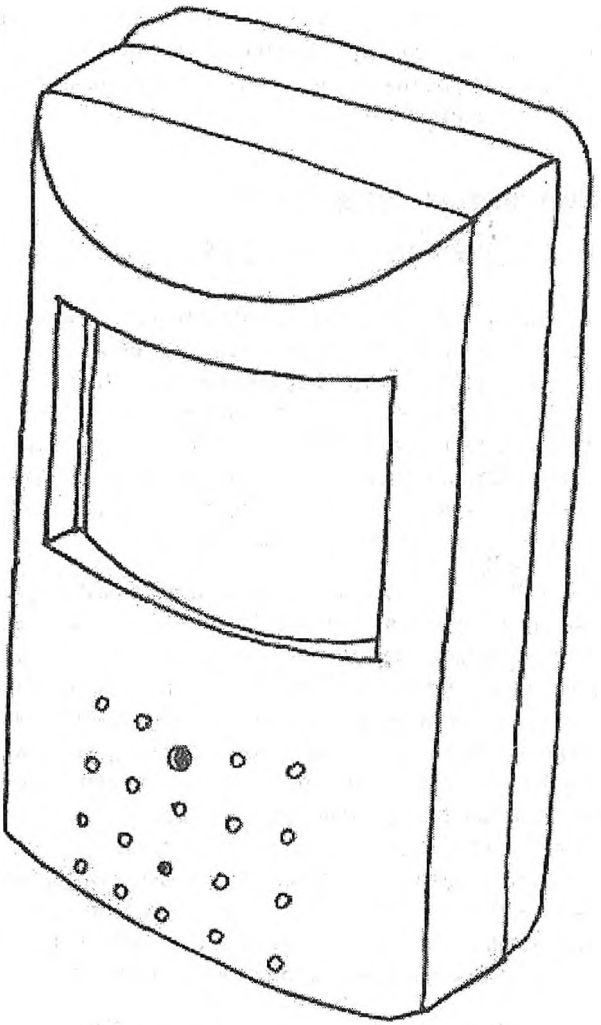


Рис. 5-2. Типичный ПИК-датчик
(пассивный датчик инфракрасного излучения)

Теперь можно поэкспериментировать со всевозможными устройствами, имеющими два контакта, которые замыкаются при определенных условиях. Например, с термостатом, срабатывающим при определенной температуре в помещении. Только используйте старый термостат, который ни к чему не подсоединен. А можно просто закрепить два зачищенных проводка на расстоянии нескольких миллиметров друг от друга и полить их водой.

Используя заключительный пример из главы 4 и ПИК-датчик, можно было бы сделать лампу, реагирующую на присутствие людей, или, подключив качающийся переключатель, можно сделать лампу, которая загорается, когда ее наклоняют набок.

Управление светом с помощью широтно-импульсной модуляции

Опираясь на те знания, которые вы уже получили, можно было бы соорудить интерактивный светильник, который управлялся бы не с помощью примитивного переключателя, а куда изящнее. До сих пор мы создавали системы, в которых свет можно было или включить, или выключить. Хорошо бы теперь сделать такой светильник, у которого можно было бы регулировать интенсивность. Чтобы решить эту проблему, достаточно применить известный фокус, благодаря которому стали возможными такие вещи, как кино и телевидение.

Когда в четвертой главе мы рассматривали первый пример, я намекнул на то, что если мы будем менять числа в функции задержки, наступит момент, когда мигание станет незаметным. Тогда-то вы с удивлением отметите, что, по ощущениям, яркость светодиода снизилась где-то наполовину. Теперь измените числа таким образом, чтобы время подачи тока на светодиод стало в четыре раза меньше того промежутка времени, когда тока нет. Загрузив программу в Arduino и включив напряжение, вы увидите, что яркость светодиода уменьшилась в 4 раза! Метод, который мы применили, называется широтно-импульсной модуляцией (ШИМ). Не пугайтесь терминологии, а просто запомните: если заставить светодиод мигать настолько часто, что мигание становится незаметным, то за счет изменения соотношения времен включения и отключения изменится его яркость. На рис. 5-3 показано, как работает данный метод.

Этот способ работает не только со светодиодами, но и с другими устройствами. К примеру, подобным образом можно управлять скоростью вращения электродвигателя.

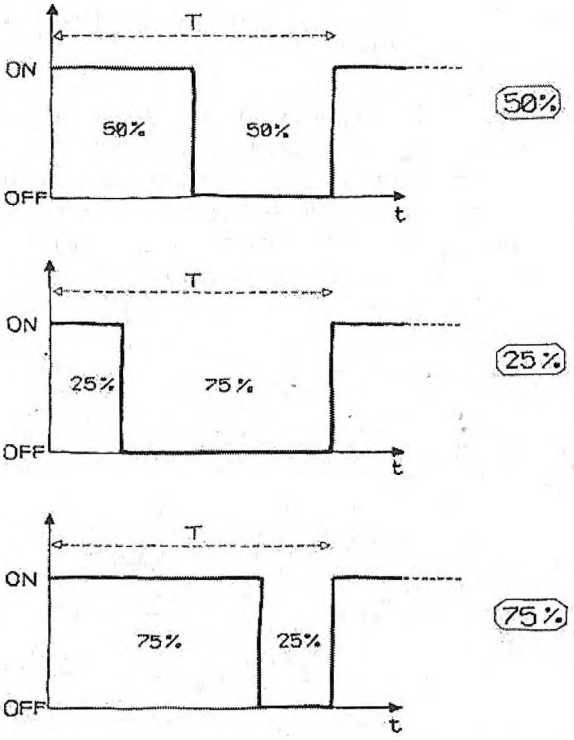


Рис. 5-3. Широтно-импульсная модуляция в действии

Экспериментируя, вы увидите, что заставлять светодиод работать в режиме мигания, устанавливая задержки в программном коде, несколько неудобно. Как только потребуется считать состояние датчика или передать данные в последовательный порт, светодиод будет мерцать, ожидая, пока вы закончите это делать. К счастью, в самом процессоре Arduino предусмотрен механизм, способный весьма эффективно обеспечивать мигание трех светодиодов, пока ваша программа выполняет другие команды. Для реализации данного механизма предусмотрены выводы 9, 10 и 11, которые могут управляться командой `analogWrite()`.

Например, команда analogWrite(9,128) устанавливает 50%-ю яркость светодиода, подключенного к выводу 9. При чем здесь 128? Дело в том, что команда `analogWrite()` использует в качестве аргумента число между 0 и 255, где 255 соответствует максимальной яркости, а 0 — выключенному состоянию.

Примечание

Наличие трех каналов очень важно, ведь если купить красный, зеленый и синий светодиоды, можно смикшировать их излучение и получить любой цвет!

Давайте попробуем. Соберите схему, показанную на рис. 5-44. Обратите внимание на то, что светодиоды имеют строго определенную полярность: длинный вывод (положительный) должен располагаться справа, а короткий вывод (отрицательный) — слева. Кроме того, у большинства светодиодов с той стороны, куда входит отрицательный контакт, корпус имеет плоскую грань, как показано на рисунке.

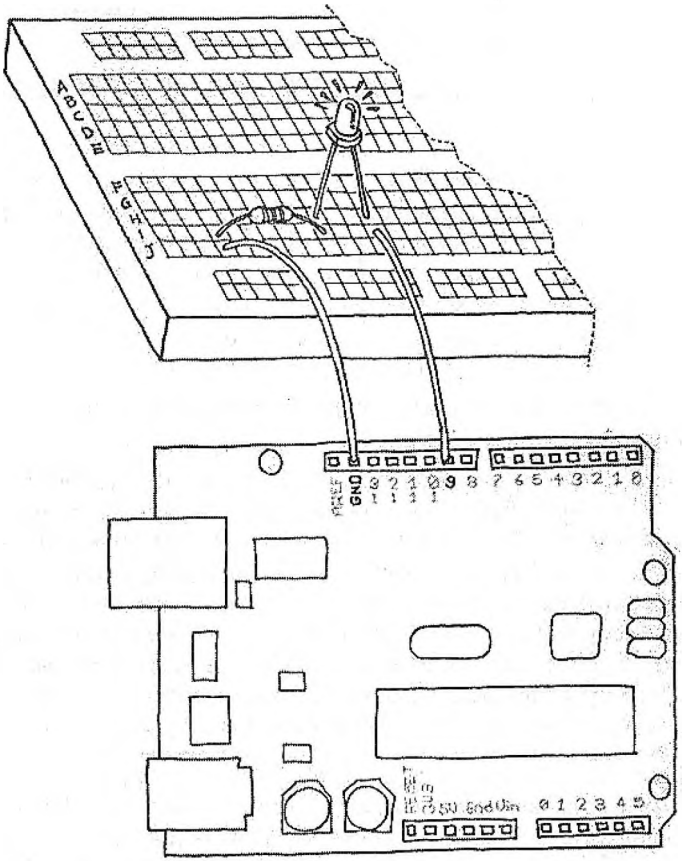


Рис. 5-4. Светодиод, подключенный к выводу ШИМ

Составьте новый скетч в среде Arduino, используя пример 5-1 (коды примеров можно также загрузить с веб-сайта www.makezine.com/getstartedarduino).

Пример 5-1. Увеличение и уменьшение яркости светодиода как на компьютере Apple

```
#define LED 9 // контакт для светодиода (LED)
int i = 0; // Это выражение используется для счета
           // в прямом и обратном направлении
void setup() {
   pinMode(LED, OUTPUT); // сообщаем Arduino, что светодиод
                         // (LED) является выходом
}
void loop(){
   for (i = 0; i < 255; i++) { // цикл от 0 до 254
                               // (увеличение яркости
      analogWrite(LED, i); // задается яркость светодиода
      delay(10); // Ожидание в течение 10 мсек,
                 // так как функция analogWrite
// действует мгновенно и не будет
// видно никаких изменений
   }
   for (i = 255; i > 0; i--) { // цикл от 255 до 1
                               // (уменьшение яркости
      analogWrite(LED, i); // задается яркость светодиода
      delay(10); // Ожидание в течение 10 мсек
   }
}
```

Сейчас вы сымитировали причудливое поведение портативного ПК. Наверное, расточительно использовать Arduino для таких элементарных поделок. Давайте воспользуемся полученными знаниями для усовершенствования нашей лампы.

Постройте на макетной плате схему, которую мы использовали для считывания состояния кнопки (см. главу 4). Посмотрим, сможете ли вы сделать это, не подглядывая на следующую страницу. Вы должны понять, что каждая описываемая мной элементарная схема является «конструк-

тивным блоком» для реализации куда более крупных проектов. Если вам все-таки потребуется подсмотреть, не переживайте. Важно то, что вы потратили некоторое время на обдумывание.

Чтобы собрать нужную цепь, нам потребуется объединить схему, показанную на рис. 5-4, со схемой кнопки (рис. 4-6). Если хотите, можете просто собрать обе схемы на разных концах макетной платы, на ней достаточно свободного места. Одно из достоинств макетной платы (см. приложение «А») состоит в том, что на ней имеются две шины, проходящие горизонтально снизу и сверху. Красная шина предназначена для подключения положительного напряжения, а синяя или черная — для заземления.

В данном конкретном случае нам будут нужны два резистора, которые необходимо подсоединить к выводу GND (заземление) на плате Arduino. Так как на плате Arduino есть два вывода GND, можно было бы просто подсоединить схемы, приведенные на рис. 4-6 и на рис. 5-4, как показано на каждом из рисунков. А можно соединить проводником земляную шину макетной платы и один из выводов GND на плате Arduino, а затем взять провода, подключенные к GND на наших рисунках, и подсоединить их к земляной шине макетной платы.

Если вы еще не готовы опробовать этот вариант, не беспокойтесь: просто сделайте так, как показано на рис. 4-6 и рис. 4-5. С примером использования шин макетной платы я познакомлю вас в главе 6.

Возвращаясь к нашему примеру, зададимся вопросом: как при помощи всего одной кнопки управлять яркостью лампы?

Для этого нам надо будет изучить еще один метод: определение продолжительности нажатия кнопки. Чтобы сделать это, мы модернизируем пример 4-5 главы 4 и добавим туда фрагмент, ответственный за интенсивность света. Идея заключается в создании «интерфейса», где нажатие и отпускание кнопки включает и выключает свет, а нажатие и удержание кнопки позволяет изменить яркость.

Давайте посмотрим на соответствующий скетч:

Пример 5-2. Включение светодиода нажатием кнопки, обеспечение включенного состояния после того, как кнопка отпущена, обеспечение борьбы с дребезгом. Если кнопка удерживается в нажатом состоянии, изменяется яркость.

```
#define LED 9 // контакт для светодиода (LED)
#define BUTTON 7 // входной контакт для кнопки
int val = 0; // сохранение состояния входного вывода
int old_val = 0; // сохранение предыдущего значения
                 // переменной "val"
int state = 0; // 0 = LED светодиод выключен,
               // 1 = LED светодиод включен
int brightness = 128; // Сохранение значения яркости
unsigned long startTime = 0; // когда нажата кнопка?
void setup() {
   pinMode(LED, OUTPUT); // сообщаем Arduino, что светодиод
                         // (LED) является выходом
   pinMode(BUTTON, INPUT); // и BUTTON (КНОПКА)
                           // является входом
}
void loop() {
   val = digitalRead(BUTTON); // считывается входное
                              // значение и сохраняется,
                              // свежее решение проверка
                              // выполнения перехода
   if ((val == HIGH) && (old_val == LOW)) {
      state = 1 - state; // изменение состояния
                         // из выключенного
                         // или наоборот на включенное
      startTime = millis(); // millis() — часы в Arduino
       // они возвращают количество миллисекунд,
       // прошедших с момента сброса платы
       // в исходное состояние.
       // (в этой строке запоминается, когда кнопка
       // была нажата последний раз)
       delay(10);
   }
```

```
    // проверка, удерживается ли кнопка нажатой
      if ((val == HIGH) && (old_val == HIGH)) {
    // Если кнопка удерживается нажатой более чем 500 мсек.
      if (state == 1 && (millis() - startTime) > 500) {
        brightness++; // увеличение яркости на 1
        delay(10); // задержка во избежание слишком быстрого
                   // увеличения яркости
      if (brightness > 255) { // 255 — максимальная яркость
          brightness = 0; // если значение превышает 255,
          // давайте вернем его обратно в 0
        }
    }
}
old_val = val; // значение val теперь является старым,
               // сохраним его
    if (state == 1) {
      analogWrite(LED, brightness); // включить светодиод
                                    // (LED) при текущем
                                    // уровне яркости
    } else {
      analogWrite(LED, 0); // выключить светодиод (LED)
    }
}
```

Теперь опробуйте предложенный скетч. Как видите, наша модель взаимодействия приобретает определенные свойства. Если нажать кнопку и сразу отпустить ее, лампа включается или выключается. Если удерживать кнопку нажатой, изменяется яркость. При достижении нужной яркости просто отпустите кнопку.

Теперь давайте научимся использовать некоторые еще более интересные датчики.

Светочувствительный элемент вместо кнопки

Давайте проведем интересный эксперимент. Для этого нам понадобится светочувствительный элемент – фоторезистор, подобный тому, что показан на рис. 5-5. (Рекомендуется www.amperka.ru/LDR)

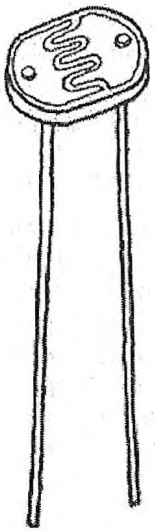


Рис. 5-5. Фоторезистор

В темноте сопротивление фоторезистора достаточно велико. Если на него направить свет, сопротивление быстро уменьшится, и он станет неплохим проводником. Таким образом, фоторезистор представляет собой переключатель, приводимый в действие светом.

Соберите схему, которая соответствовала примеру 4-2 (см. раздел «Использование кнопки для управления светодиодом» главы 4), затем загрузите код из примера 4-2 в интегрированную среду разработки.

Вставьте в макетную плату вместо кнопки фоторезистор. Теперь, если закрыть фоторезистор рукой, светодиод выключится. Уберите руку, и свет включится. Только что вы создали устройство, в котором светодиод управляется при помощи датчика. Это важно, так как впервые в этой книге мы использовали электронный компонент, не являющийся простым механическим устройством. Это настоящий датчик, обладающий широкими возможностями.

Аналоговый вход

Как было показано в предыдущем разделе, Arduino позволяет определять, подается ли напряжение на один из его входов, и сообщать об этом с помощью функции `digitalRead()`.

Этот вариант ответа «или/или» прекрасно работает во многих случаях, однако фоторезистор, который мы только что использовали, способен сообщить нам не только о наличии света, но и о его интенсивности. Именно в этом заключается разница между датчиком типа «Вкл./Выкл.», который уведомляет нас о наличии или отсутствии чего-либо, и аналоговым датчиком, состояние которого плавно изменяется. Чтобы считать показание такого датчика, необходим вход другого типа.

В нижнем правом углу платы Arduino вы видите шесть контактов с маркировкой Analog In (Аналоговый вход). В отличие от цифровых, эти входы могут сообщить не только о том, приложено ли к ним напряжение, но и определить величину поданного напряжения. Используя функцию `analogRead()`, можно считать напряжение, приложенное к любому из этих контактов. Данная функция возвращает число от 0 до 1023, которое соответствует напряжению в диапазоне от 0 до 5 В. Например, если на контакт номер 0 подано напряжение 2,5 В, функция `analogRead(0)` возвращает число 512.

Соберите схему, показанную на рис. 5-6, используя резистор на 10 кОм, и выполните программу, приведенную в примере 5-3. Расположенный на плате светодиод (можно было бы вставить и свой собственный светодиод в контакты 13 и GND, как показано в разделе «Включение светодиода в режиме мигания» главы 4) начинает мигать с частотой, зависящей от интенсивности падающего на датчик света.

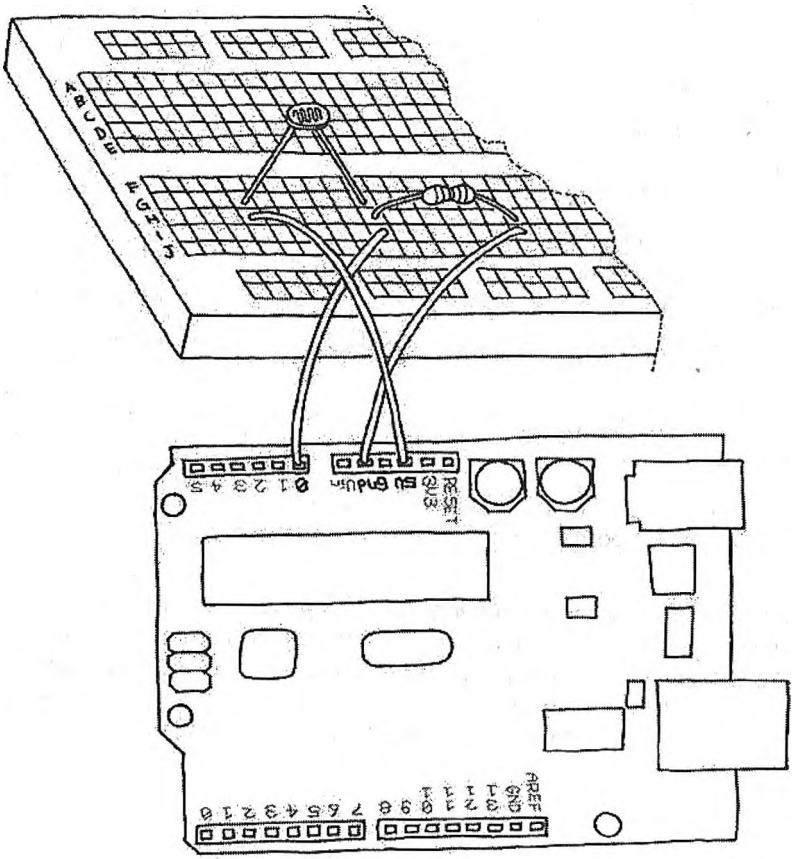


Рис. 5-6. Схема аналогового датчика

Пример 5-3. Частота мигания светодиода зависит от величины сопротивления на аналоговом входе

```
#define LED 13 // Контакт для светодиода (LED)
int val = 0; // переменная, используемая для хранения
             // значения, поступающего от датчика
void setup() {
   pinMode(LED, OUTPUT); // Светодиод действует
                         // как OUTPUT (ВЫХОД)
   // Примечание. Аналоговые контакты
   // автоматически устанавливаются в качестве входов
}
void loop() {
   val = analogRead(0); // Считывание значения с
                        // датчика
   digitalWrite(13, HIGH); // Включение светодиода
   delay(val); // Остановка выполнения программы
               // на некоторое время
   digitalWrite(13, LOW); // Выключение светодиода
   delay(val); // Остановка выполнения программы
               // на некоторое время
}
```

Теперь попробуем код, приведенный в примере 5-4. Прежде чем это сделать, нам необходимо изменить схему. Подключите светодиод к контакту 9, как показано на рисунке 5-4. Поскольку на макетной плате уже стоит довольно много элементов, вам потребуется найти место, где светодиод, провода и резистор не будут мешать работе фоторезистора.

Пример 5-4. Яркость светодиода задается значением, считанным с аналогового входа

```
#define LED 9 // Контакт для светодиода (LED)
int val = 0; // переменная, используемая для хранения
             // значения, поступающего от датчика
void setup() {
   pinMode(LED, OUTPUT); // Светодиод действует
                         // как OUTPUT (ВЫХОД)
   // Примечание. Аналоговые контакты
   // автоматически устанавливаются в качестве входов
}
void loop() {
   val = analogRead(0); // Считывание значения с
                        // датчика
   analogWrite(LED, val/4); // Включение светодиода при
                            // яркости, задаваемой
                            // датчиком
   delay(10); // Остановка выполнения программы
              // на некоторое время
}
```

Примечание

Мы задаем яркость, деля значение val на 4, так как функция `analogRead()` возвращает число от 0 до 1023, а функция `analogWrite()` принимает значение максимум равное 255.

Пробуем другие аналоговые датчики

В схеме из предыдущего раздела можно использовать множество других резистивных датчиков, например терморезистор, представляющий собой устройство, сопротивление которого изменяется в зависимости от температуры. А я уже продемонстрировал вам, как в соответствии с законом Ома изменение сопротивления влияет на изменение напряжения, на которое реагирует Arduino.

Если вы работаете с терморезистором, следует учитывать, что между считываемым значением и реальной измеряемой температурой нет прямой зависимости. Если требуются точные показания, следует соотнести числа, приходящие от аналогового входа, с одновременно проводимыми измерениями при помощи термометра. Сопоставив эти значения в таблице, можно разработать способ калибровки значений, полученных с аналогового входа, по значениям реальной температуры.

До сих пор мы использовали светодиод в качестве выходного устройства, но как сделать так, чтобы мы могли понимать, что именно Arduino считывает из датчика? Мы не можем заставить плату, весело подмигивая, передавать нам информацию азбукой Морзе. Вернее, можем, но существует более простой способ считывать эти значения. Для этого мы заставим Arduino взаимодействовать с компьютером через последовательный порт.

Связь по последовательному интерфейсу

В самом начале я говорил, что на плате Arduino имеется USB-порт, который используется интегрированной средой разработки для выгрузки кода в процессор. Однако написанные вами скетчи могут использовать данный интерфейс как для получения команд из компьютера, так и для передачи данных обратно в компьютер. Для этой цели мы собираемся применить так называемый объект Serial (объект — это комплекс возможностей, предназначенных для удобства людей, пишущих скетчи).

Данный объект содержит весь программный код, который необходим для организации передачи и приема данных. Мы будем использовать последнюю схему с фоторезистором и передадим считанные значения обратно в компьютер. Для этого наберите приведенный ниже код (его можно также загрузить с веб-сайта www.makezine.com/getstartedarduino):

Пример 5-5

```
#define SENSOR 0 // Выбор входного контакта для
                 // резистора датчика
int val = 0; // переменная для хранения значения,
             // поступающего от датчика
void setup() {
  Serial.begin(9600); // Открытие последовательного порта
                      // для передачи данных
                      // обратно в компьютер
                      // на скорости 9600 бит/сек
}
void loop() {
    val = analogRead(SENSOR); // Считывание значения
                              // из датчика
    Serial.println(val); // передача значения
                         // в последовательный порт
    delay(100); // Ожидание в течение 100 мсек между
                // отправками данных
}
```

После загрузки этого скетча в Arduino на экране компьютера в интегрированной среде разработки Arduino нажмите кнопку **Serial Monitor**. В нижней части окна будут видны меняющиеся числа. Теперь любая программа, которая способна считывать данные из последовательного порта, может общаться с Arduino.

Многие языки программирования позволяют писать фрагменты кода, адресованные последовательному порту. Прекрасным дополнением к Arduino является язык Processing (www.processing.org), идеологически и структурно весьма схожий с интегрированной средой разработки.

Управление значительными нагрузками (электродвигателями, лампочками и другими устройствами)

К выходным контактам Arduino можно подключать устройства, потребляющие ток до 20 миллиампер. Этого достаточно для светодиода, однако если туда подсоединить даже небольшой маломощный электромотор, появляется опасность выхода из строя процессора. Для управления большими нагрузками (двигателями, лампами накаливания) требуется некий внешний компонент, который способен включать и выключать подобные устройства и к тому же управляться с помощью Arduino. Одно из таких устройств называется полевым МОП-транзистором. Не обращайте внимания на странное имя, для вас это не более чем электронный переключатель, которым можно управлять, подавая напряжение на один из трех его выводов, который называется затвор. По назначению этот элемент схож с домашним настенным выключателем света. Посудите сами: мы включаем и выключаем свет, нажимая на клавишу пальцем, а в случае МОП-транзистора палец заменяет контакт платы Arduino, с которого подается напряжение на затвор.

Примечание

Аббревиатура «МОП» расшифровывается как «металл-оксид-полупроводник». Это особый тип транзистора, в котором электрический ток протекает через полупроводник, когда на затвор подано напряжение. Затвор изолирован от остальных «внутренностей» транзистора, что упрощает его согласование с Arduino. Подобные транзисторы идеально подходят для включения и выключения больших нагрузок на высоких частотах.

На рис. 5-7 показано, как можно использовать полевой МОП-транзистор (IRF520) для включения и выключения небольшого двигателя вентилятора. Можно заметить, что питание на этот двигатель подается от 9-вольтового разъема на плате Arduino. Это еще одно достоинство полевого МОП-транзистора: он позволяет управлять устройствами, которые запитываются отдельно от Arduino. В связи с тем, что полевой МОП-транзистор подключен к контакту 9, вы можете использовать функцию `analogWrite()` для управления скоростью электродвигателя методом широтно-импульсной модуляции.

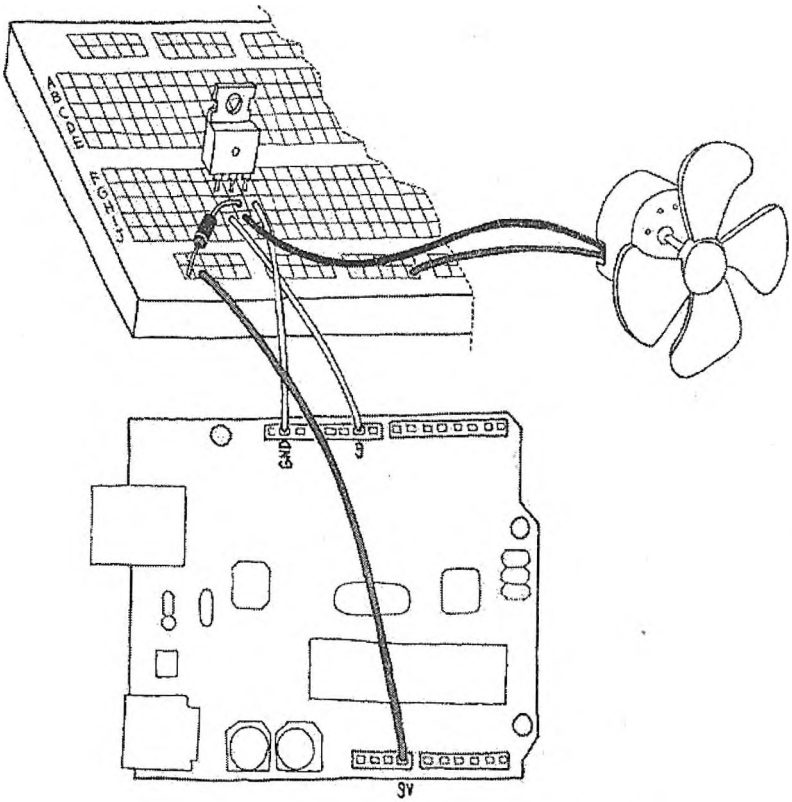
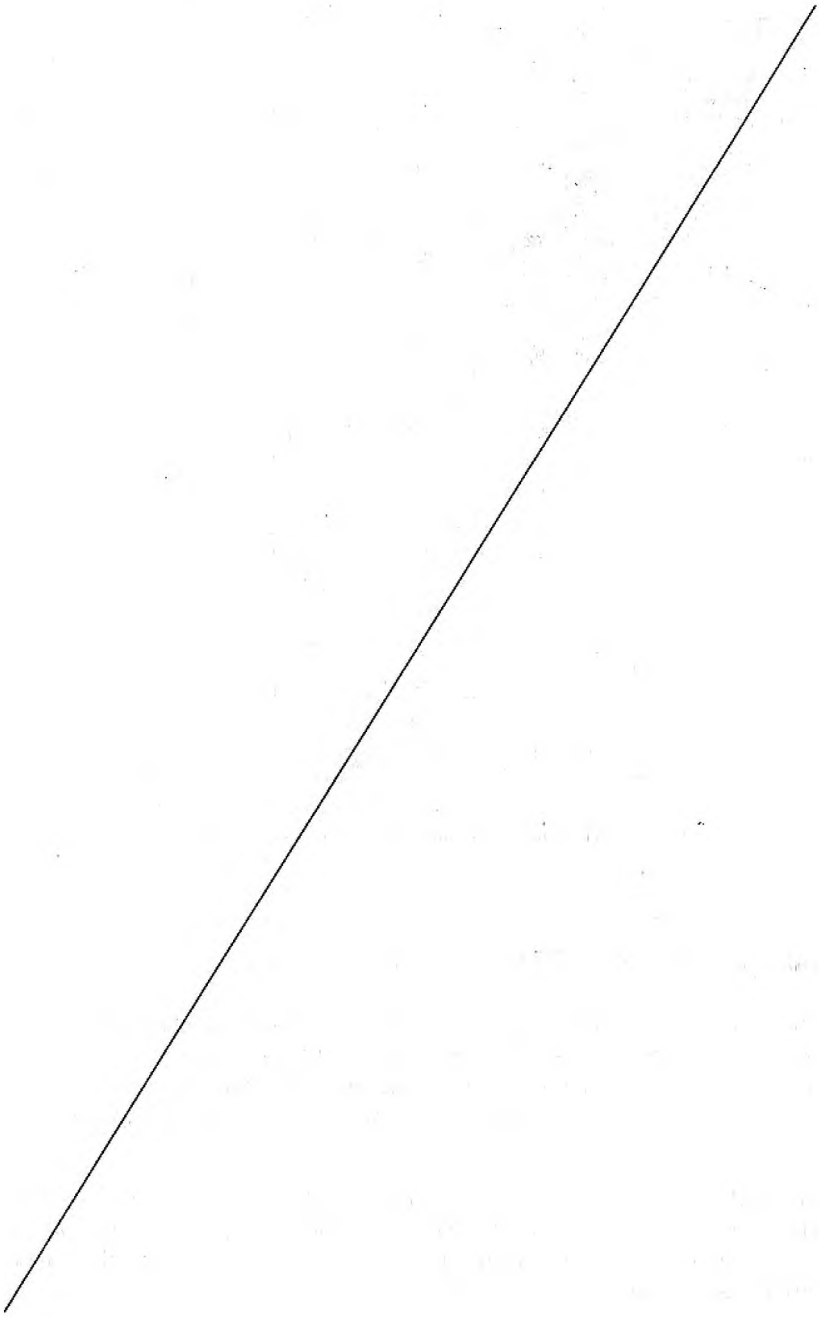


Рис. 5-7. Подключение электромотора к плате Arduino

Комплексные датчики

Комплексными датчиками считаются устройства, которые генерируют информацию, требующую для обработки не только функцию `digitalRead()` и `analogRead()`, но и другие средства. Такие датчики обычно представляют собой небольшие схемы на основе маломощного микроконтроллера.

Среди доступных комплексных датчиков можно отметить ультразвуковые дальномеры, инфракрасные локаторы и акселерометры. Примеры их применения можно найти на веб-сайте в разделе Tutorials (www.arduino.cc/en/Tutorial/HomePage).



Глава 6. Там, за облаками

В предыдущих главах излагались основы Arduino и фундаментальные функциональные блоки, которыми могут манипулировать пользователи.

Позволю себе напомнить основные термины «алфавита» Arduino.

Цифровой выход

Цифровой выход использовался нами для управления светодиодом, но в сочетании с дополнительной цепью он может применяться для управления двигателями, генерирования звуков и во многих других приложениях.

Аналоговый выход

Данный выход позволяет не только включать и выключать светодиод, но и управлять его яркостью. С помощью аналогового выхода можно управлять скоростью вращения двигателя.

Цифровой вход

Цифровой вход предоставляет возможность считывания состояния простых датчиков, таких как кнопки или качающиеся переключатели.

Аналоговый вход

Способен считывать информацию с датчиков, состояние которых плавно и непрерывно меняется в широких пределах.

Последовательный интерфейс

Данный интерфейс позволяет Arduino взаимодействовать с компьютером, то есть обмениваться данными или просто следить, что происходит с программой, которая выполняется на плате.

В этой главе я хочу продемонстрировать вам, как, используя знания, полученные в предыдущих главах, собрать работоспособную модель. Я постарался показать здесь, что каждый уже рассмотренный нами простой пример может быть использован в качестве функционального блока более сложного проекта.

Именно в этой главе я пытаюсь раскрыться как дизайнер. Применив технологии XXI века, мы повторим здесь классический светильник, созданный моим любимым итальянским дизайнером Джо Коломбо. Предмет, который мы собираемся создать, навеян лампой «Атон», разработанной в 1964 году.

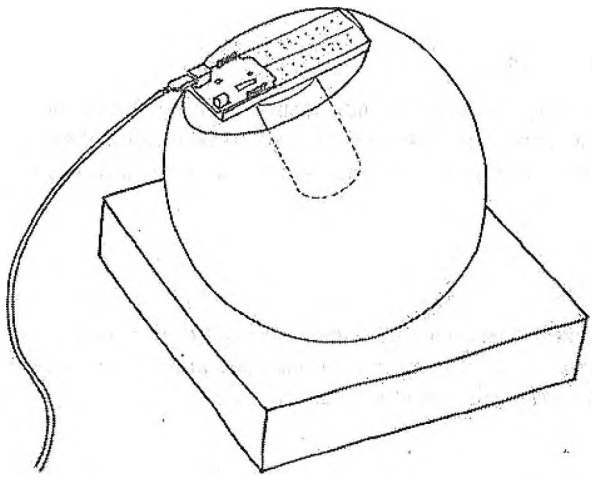


Рис. 6-1. Готовая лампа

Как становится понятно из рисунка 6-1, лампа представляет собой шар, расположенный на основании с большим отверстием, предназначенным для того, чтобы шар не скатывался со стола. Такая конструкция позволяет ориентировать лампу в любом положении.

С функциональной точки зрения мы создадим устройство, которое будет подключаться к Интернету, получать текущий список статей в блоге Make (blog.makezine.com) и подсчитывать, сколько раз там упоминаются слова peace (мир), love (любовь) и Arduino. С помощью полученных значений мы сгенерируем цвет, излучаемый лампой. У самого светильника будет кнопка включения и выключения, а также светочувствительный датчик для автоматического включения.

Планирование

Давайте обдумаем, что мы хотим в конечном счете получить, а также посмотрим, что нам для этого понадобится. Прежде всего нам нужна плата Arduino, чтобы можно было подключиться к Интернету. Поскольку на плате Arduino есть только USB-порт, мы не можем напрямую подключить ее к телефонной розетке или к оптоволоконному кабелю для выхода в Интернет, поэтому нам необходимо продумать, как получить доступ к всемирной сети. Обычно в этом случае запускают соответствующее приложение на компьютере, которое устанавливает подключение к Интернету, обрабатывает данные и передает Arduino уже отфильтрованную информацию.

Плата Arduino — это всего-навсего простейший компьютер с небольшой памятью. Она не способна обрабатывать большие файлы, а ведь когда мы подключаемся к RSS-рассылке, то получаем весьма «тяжелые» XML-файлы, для которых требуется гораздо больше памяти, чем у Arduino. Но выход есть: используя язык Processing, мы реализуем прокси-агент для упрощения XML.

Язык Processing

Processing — это язык, из которого вырос проект Arduino. Мы любим этот язык и используем его для обучения новичков программированию, а также для составления замечательных программ. Processing и Arduino можно назвать идеальным сочетанием. Еще одно достоинство этого языка состоит в том, что он является продуктом с открытым исходным кодом и выполняется на всех основных платформах (Mac, Linux и Windows). Он позволяет создавать автономные приложения, которые выполняются на этих платформах. Сообщество пользователей языка Processing известно своей активностью и склонностью к взаимопомощи, здесь вы можете найти тысячи готовых программ.

Прокси-агент выполняет за нас следующую работу: загружает RSS-рассылку с веб-сайта makezine.com и извлекает все слова из итогового XML-файла. Затем подсчитывается количество слов peace, love и Arduino, встречающихся во всем тексте. По этим трем значениям рассчитывается значение цвета, которое отправляется в Arduino. Плата передает обратно данные по интенсивности света, измеренной датчиком. Полученная информация отображается на экране компьютера.

Что касается аппаратной начинки, то мы объединим пример с кнопкой, пример со светочувствительным элементом, управление светодиодом с помощью широтно-импульсной модуляции (только помножим все это на три) и последовательный интерфейс.

Так как Arduino является простым устройством, нам потребуется зако-

дировать цвет наиболее простым способом. Мы воспользуемся стандартным представлением цвета в HTML: знак решетки (#), за которым следует шесть шестнадцатеричных цифр.

Шестнадцатеричное счисление удобно, так как каждое 8-разрядное число представляется двумя символами. В десятичной системе для этого потребовалось бы от одного до трех символов. Программный код будет довольно прост: мы ждем, пока не увидим знак #, затем считываем шесть последующих знаков в буфер (переменная, используемая в качестве области временного хранения данных). Наконец, мы преобразуем каждую группу из двух символов в байт, соответствующий яркости одного из трех светодиодов.

Программный код

Мы составим две программы: одну на языке Processing и еще одну для Arduino. Здесь приводится фрагмент на языке Processing, который можно загрузить с веб-страницы www.makezine.com/getstartedarduino.

Пример 6-1. Подключенная в сеть лампа Arduino; фрагменты программного кода, разработанные под влиянием информации в блоге Тода Е. Курта todbot.com)

```
import processing.serial.*;

String feed = "http://blog.makezine.com/index.xml";

int interval = 10; // Извлечение рассылки каждые 60 сек;
int lastTime; // Последний раз извлечения содержимого

int love = 0;
int peace = 0;
int arduino = 0;

int light = 0; // Уровень света, измеренный лампой

Serial port;
color c;
String cs;
```

```
String buffer = ""; // Накапливание символов, поступающих из
PFont font;

void setup() {
    size(640,480);
    frameRate(10); // Быстрые обновления не требуются

    font = loadFont("HelveticaNeue-Bold-32.vlw");
    fill(255);
    textFont(font, 32);
    // ВАЖНОЕ ПРИМЕЧАНИЕ:
    // Первый последовательный порт, извлекаемый с помощью
    // Serial.list()должен быть портом Arduino. Если нет,
    // удалить следующую строку комментария, убрав перед
    // ней символы, и повторно выполнив программу, чтобы
    // увидеть список последовательных портов. Затем,
    // изменить 0 между [ и ] на номер порта,
    // к которому подключена плата Arduino.
    //println(Serial.list());
    String arduinoPort = Serial.list()[0];
    port = new Serial(this, arduinoPort, 9600);
                                        // Подключение

    lastTime = 0;
    fetchData();
}

void draw() {
    background( c );
    int n = (interval - ((millis()-lastTime)/1000));

    // Определение цвета на основе 3 значений
    c = color(peace, love, arduino);
    cs = "#" + hex(c,6); // Подготовка строки к отправке в

    text("Arduino Networked Lamp", 10,40);
    text("Чтение рассылки:", 10, 100);
    text(feed, 10, 140);
```

```
  text("Следующее обновление через "+ n + " секунд",10,450);
  text("peace" ,10,200);
  text(" " + peace, 130, 200);
  rect(200,172, peace, 28);

  text("love ",10,240);
  text(" " + love, 130, 240);
  rect(200,212, love, 28);

  text("arduino ",10,280);
  text(" " + arduino, 130, 280);
  rect(200,252, arduino, 28);

  // Запись строкового значения цвета на экране
  text("sending", 10, 340);
  text(cs, 200,340);
  text("light level", 10, 380);
  rect(200, 352,light/10.23,28); // Преобразование 1023 в

  if (n <= 0) {
     fetchData();
     lastTime = millis();
}

port.write(cs); // Передача данных в Arduino

if (port.available() > 0) { // Проверка, есть ли данные,
                            // ожидающие
   int inByte = port.read(); // считывание одного байта
   if (inByte != 10) { // Если байт не является
                       // разделителем строки (newline)
      buffer = buffer + char(inByte); // просто добавляет
                                      // его в
   }
   else {

        // Достигнут разделитель строки (newline),
        // давайте обработаем данные
        if (buffer.length() > 1) { // Убедитесь,
                                   // что достаточно
```

```
            // Отсекается последний символ, это символ
                возврата каретки
            // (возврат каретки — символ в конце
            // строки текста)
            buffer = buffer.substring(0,buffer.length() -1);

            // преобразование буфера из строкового значения
            // в целое число
            light = int(buffer);

            // очистка буфера для  следующего цикла чтения
            buffer = "";

            // Вероятно мы запаздываем, принимая показания
            // из Arduino. Так что, давайте очистим
            // незавершенные входящие показания датчика,
            // чтобы следующее считывание было новейшим.
            port.clear();
            }
        }
    }
}

void fetchData() {
    // эти строки используются для анализа
    // поступающих данных
    String data;
    String chunk;

    // обнуление счетчиков
    love = 0;
    peace = 0;
    arduino = 0;
    try {
        URL url = new URL(feed); // Объект для представления
    // подготовка соединения
    URLConnection conn = url.openConnection();
    conn.connect(); // теперь подключаемся к веб-сайту
```

```
    // Это в некоторой степени виртуальные слесарные
    // работы, поскольку мы соединяем данные,
    // поступающие из соединения в буферизованный
   // считыватель, который считывает данные по строке за раз.
    BufferedReader in = new BufferedReader
        (new InputStreamReader(conn.getInputStream()));
    // считывает каждую строку из рассылки
    while ((data = in.readLine()) != null) {
       StringTokenizer st =
          new StringTokenizer(data,"\"<>,.()[] ");
          // останов
       while (st.hasMoreTokens()) {
          // каждая порция данных преобразуется
           // в нижний регистр
          chunk= st.nextToken().toLowerCase() ;
          if (chunk.indexOf("love") >= 0 )
              // найдено слово "love" (любовь)?
          love++; // increment love by 1
          if (chunk.indexOf("peace") >= 0)
              // найдено слово "peace" (мир)?
          peace++; // increment peace by 1
          if (chunk.indexOf("arduino") >= 0)
             // найдено слово "arduino"?
          arduino++; // increment arduino by 1
       }
    }
    // Устанавливает 64 в качестве максимального количества
    // ссылок, которые учитываются
if (peace > 64) peace = 64;
   if (love > 64) love = 64;
    if (arduino > 64) arduino = 64;
    peace = peace * 4; // Умножается на 4, чтобы макс.
    love = love * 4;    // что оказывается удобно
                        // при построении
    arduino = arduino * 4; // цвета, состоящего из 4 байтов
```

```
}
catch (Exception ex) { // Если была ошибка, остановить
    ex.printStackTrace();
    System.out.println("ERROR: "+ex.getMessage());
    }
}
```

Прежде чем скетч на языке Processing станет выполняться корректно, необходимо сделать две вещи. Во-первых, необходимо дать команду среде Processing на генерирование шрифта, который мы собираемся использовать для скетча. Чтобы сделать это, наберите и сохраните приведенный выше фрагмент. Затем, не закрывая его, выберите меню **Tools** в среде Processing, далее выберите пункт **Create Font**. Выберите шрифт с именем **HelveticaNeue-Bold**, **32** в качестве размера шрифта и нажмите кнопку **OK**.

Во-вторых, нам потребуется подтвердить, что для взаимодействия с платой Arduino в скетче используется правильный последовательный порт. Прежде чем убедиться в этом, необходимо собрать схему и загрузить скетч для Arduino. В большинстве случаев приведенный выше фрагмент на языке Processing выполняется без проблем. Тем не менее, если вы видите, что на плате Arduino ничего не происходит и на экране не отображается никакой информации от фоторезистора, найдите комментарий `ВАЖНОЕ ПРИМЕЧАНИЕ` в скетче на языке Processing и следуйте изложенным там инструкциям.

Ниже приводится скетч для Arduino (он доступен на веб-странице www.makezine.com/getstartedarduino):

Пример 6-2. Включенная в сеть лампа Arduino

```
#define SENSOR 0
#define R_LED 9
#define G_LED 10
#define B_LED 11
#define BUTTON 12

int val = 0; // переменная для хранения значения,
             // поступающего из

int btn = LOW;
int old_btn = LOW;
int state = 0;
char buffer[7] ;
int pointer = 0;
byte inByte = 0;

byte r = 0;
byte g = 0;
byte b = 0;

void setup() {
   Serial.begin(9600); // Открывается последовательный
                       // порт
   pinMode(BUTTON, INPUT);
}

void loop() {
   val = analogRead(SENSOR); // Считывание значения
                             // из датчика
   Serial.println(val); // Вывод значения
                        // в последовательный порт

if (Serial.available() >0) {
// Считывание поступающего байта:

inByte = Serial.read();

// Если маркер найден, следующие 6 символов являются цветом
if (inByte == '#') {
while (pointer < 6) { // Накапливание 6 символов
buffer[pointer] = Serial.read(); // Хранение в буфере
pointer++; // Перемещение указателя вперед на 1
}

// Теперь имеются 3 числа, сохраненных
// как шестнадцатеричные числа
// Требуется декодировать их в 3 байта r, g и b
```

```
r = hex2dec(buffer[1]) + hex2dec(buffer[0]) * 16;
g = hex2dec(buffer[3]) + hex2dec(buffer[2]) * 16;
b = hex2dec(buffer[5]) + hex2dec(buffer[4]) * 16;

pointer = 0; // Указатель сбрасывается в 0,
             // чтобы можно было повторно использовать
}
}

btn = digitalRead(BUTTON); // Считывается и сохраняется
                           // входное значение

// Проверяется, был ли переход
if ((btn == HIGH) && (old_btn == LOW)){
   state = 1 - state;
}

   old_btn = btn; // теперь значение val — старое,
                  // давайте сохраним его

   if (state == 1) { // Если лампочка включена

   analogWrite(R_LED, r); // включаются светодиоды
   analogWrite(G_LED, g); // с цветом
   analogWrite(B_LED, b); // переданным компьютером
} else {

   analogWrite(R_LED, 0); // иначе выключаются
   analogWrite(G_LED, 0);
   analogWrite(B_LED, 0);
}

   delay(100); // Ожидание в течение 100 мсек
               // между передачами
}

int hex2dec(byte c) { // Преобразование одного
                      // шестнадцатеричного символа в
   if (c >= '0' && c <= '9') {
      return c - '0';
   } else if (c >= 'A' && c <= 'F') {
     return c - 'A' + 10;
   }
}
```

Сборка схемы

На рис. 6-2 показано, как собрать схему. Номинал всех резисторов равен 10 кОм, хотя резисторы, подсоединенные к светодиодам, могут быть и меньшего номинала.

Если помните, светодиоды обладают полярностью (пример с широтно-импульсной модуляцией, приведенный в главе 5). В данной схеме длинный вывод (положительный) должен располагаться справа, а короткий вывод (отрицательный) — слева. У большинства светодиодов имеется плоская грань со стороны отрицательного контакта.

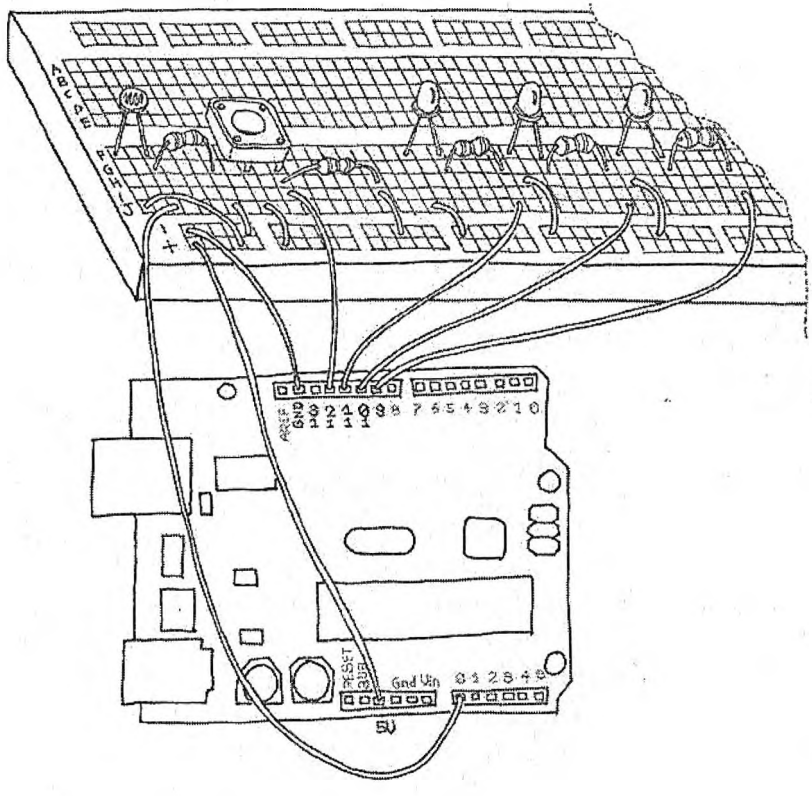


Рис. 6-2. Схема «Подключенная в сеть лампа Arduino»

Соберите показанную на рисунке схему, используя один красный, один зеленый и один синий светодиод. Далее загрузите скетчи в среду Arduino и в среду Processing, затем выполните программы и посмотрите, что вышло. Если возникли какие-то проблемы, обратитесь за необходимыми разъяснениями к главе 7, «Поиск и устранение неполадок».

Теперь мы завершим наше конструирование, поместив макетную плату в стеклянный шар. Самый простой и дешевый способ сделать это — купить настольную лампу FADO в магазине IKEA. Уверяю вас, это совсем недорого!

Вместо трех отдельных светодиодов можно приобрести один RGB-светодиод (рекомендуется www.amperka.ru/RGB-LED), из которого торчат четыре контакта. Он подключается практически так же, как те светодиоды, которые показаны на рис. 6-2, за одним исключением: к земляной шине будет подсоединен только один вывод. Его ни с чем не спутаешь, он тут самый длинный. Три вывода покороче надо подключить на плате Arduino к контактам 9, 10 и 11 (с резистором между выводами и контактами, так же как и в случае, когда используются отдельно красный, зеленый и синий светодиоды).

Порядок сборки схемы

Распакуйте лампу и удалите кабель, который подведен снизу, он вам больше не понадобится.

Закрепите плату Arduino на макетной плате, а макетную плату зафиксируйте термоклеем на плафоне.

Припаяйте длинные провода к RGB-светодиоду и приклейте его там, где обычно находится электрическая лампочка. Подсоедините провода, выходящие из светодиода, к контактам макетной платы, туда, где он был подключен до того, как вы его удалили. Не забывайте, что 4-выводной RGB-светодиод имеет только один вывод для заземления.

Найдите красивую толстую доску с отверстием, которая может послужить подставкой для сферы, или просто срежьте верх картонной коробки, куда была упакована лампа, приблизительно на 5 см и проделайте в ней отверстие такого диаметра, чтобы коробка превратилась в надежную подставку. Укрепите коробку изнутри — нанесите термоклей вдоль всех изгибов, чтобы сделать основание более устойчивым.

Поместите шар на подставку, протяните через нее USB-кабель и подсоедините его к компьютеру.

Запустите программу на языке Processing, нажмите кнопку Вкл./Выкл. и посмотрите, как будет «оживать» лампа.

В качестве упражнения попробуйте добавить программный код, который реализует включение лампы, когда в комнате становится темно.

Другие возможные усовершенствования:

- Добавьте качающиеся датчики, чтобы можно было включать и выключать лампу, вращая ее в разные стороны.
- Добавьте небольшой ПИК-датчик, чтобы лампа включалась и выключалась в зависимости от того, стоит возле нее кто-нибудь или нет.
- Разработайте различные режимы ручного управления интенсивностью и цветом, чтобы можно было воспроизводить всю палитру.

Поразмышляйте над возможными модификациями проекта, экспериментируйте и наслаждайтесь результатами.

Глава 7. Поиск и устранение неполадок

В процессе работы с Arduino может наступить момент, когда ничего не работает, а значит, надо найти причину проблемы и устранить ее. Поиск и устранение неполадок оборудования, а также отладка программ — это старые как мир занятия, для которых существует несколько простых правил. Тем не менее результата удается добиться не сразу, а изрядно повозившись.

Чем дольше вы работаете с электронными устройствами и Arduino, тем больше вы приобретаете знаний и опыта, которые в конечном счете делают процесс поиска и устранения неполадок менее болезненным. Не пасуйте перед проблемами, которые встретятся вам, — как правило, они решаются гораздо проще, чем кажется на первый взгляд.

В связи с тем, что любой проект на основе Arduino включает как оборудование, так и программное обеспечение, существует несколько «контрольных точек», которые первым делом надо проверить, если что-то работает не должным образом. В процессе поиска дефекта следует действовать по трем направлениям:

Осмысление

Попытайтесь как можно глубже разобраться в том, как работают используемые вами элементы и каковы их функции в проекте. Это позволит вам найти способы проверки каждого компонента в отдельности.

Упрощение и сегментация

Древние римляне говаривали: *divide et impera*, то есть: разделяй и властвуй. Попробуйте на основе собственного понимания работы устройства мысленно разбить проект на составляющие и определить, где начинается и где заканчивается область действия каждого компонента.

Исключение и определенность

В процессе исследования причины неисправности проверяйте каждый компонент по отдельности, чтобы быть абсолютно уверенным, что он работоспособен. Постепенно у вас сформируется твердое представление о том, какие узлы проекта выполняют свои функции корректно и какие вызывают сомнение.

Отладка (debugging) — термин, обозначающий процесс поиска и устранения ошибок в программном коде. По легенде, это словечко придумал Грейс Хоппер в 1940-х годах, когда компьютеры были большей частью электромеханическими и один из них перестал работать из-за того, что в него проникли насекомые.

Большинство современных «жучков» уже не являются физическими объектами. Они по большей части виртуальны и невидимы, поэтому для их выявления требуется иногда длинный и утомительный процесс поиска.

Тестирование платы

Если не работает самый первый пример «Мигающий светодиод», не унывайте. Давайте разберемся, что надо делать в этом случае.

Прежде чем начать проклинать свое творение, необходимо убедиться, что у вас все в порядке, примерно так, как это делают пилоты, выполняя перед полетом стандартные тесты, чтобы быть уверенными в том, что самолет не только взлетит, но и приземлится.

Соедините плату Arduino с USB-портом компьютера.

- Убедитесь, что компьютер включен (да, это звучит глупо, но случается и такое). Если зеленая лампочка PWR горит, значит, на плату подается питание с компьютера. Если светодиод кажется очень тусклым, очевидно, появились какие-то проблемы с питанием. Попробуйте другой USB-кабель, осмотрите USB-порт компьютера, а также USB-разъем платы Arduino, чтобы убедиться в отсутствии повреждений. Если все попытки идентифицировать неисправность закончились неудачей, попробуйте воспользоваться другим USB-портом на компьютере или другим компьютером.
- В новой версии платы Arduino при включении желтый светодиод с маркировкой «L» начнет раздражающе мигать. Не пугайтесь, это работает тестовая программа, загруженная на заводе.

- Если на старую плату Arduino (Extreme, NG или Diecimila) питание подается от внешнего источника, проверьте, чтобы источник питания был подключен к сети и чтобы переключатель SV1 соединял два контакта, находящихся ближе всего к разъему внешнего источника питания.

Примечание

Когда возникают проблемы с другими программами и необходимо убедиться в правильном функционировании платы, откройте первый пример «Мигающий светодиод» в интегрированной среде разработки и загрузите программный код в Arduino. При этом находящийся на плате светодиод должен ритмично мигать.

Если все эти шаги успешно выполнены, можете быть уверенным в правильной работе платы Arduino.

Тестирование схемы, собранной на макетной плате

Подключите Arduino к макетной плате, соединив контакты 5 В и земли. Если зеленый светодиод PWR погаснет, немедленно отсоедините провода. Это означает, что в вашей схеме имеется дефект — где-то «короткое замыкание». Когда оно возникает, плата начинает потреблять слишком большой ток, и питание отключается, чтобы защитить компьютер.

Примечание

Если вас беспокоит возможность повреждения компьютера, помните, что на многих компьютерах обычно установлена хорошая защита от перегрузки по току, которая срабатывает практически мгновенно. Кроме того, плата Arduino оснащена собственным предохранителем — устройством защиты от перегрузки, которое сбрасывается в исходное состояние, когда неполадка устранена.

Если вас продолжает преследовать навязчивая мысль о возможном повреждении компьютера, всегда можно подключить плату Arduino через USB-разветвитель с автономным питанием. В этом случае, если все пойдет по наихудшему сценарию, неприятности случатся с USB-разветвителем, а не с компьютером.

Если произошло короткое замыкание, необходимо выполнить процедуру «упрощения и сегментации». Для этого надо «прощупать» каждый датчик, подсоединяя их к схеме по одному.

Первое, с чего всегда следует начинать, — это источник питания, то есть проводники, соединяющие контакты 5 В и GND с шинами монтажной платы. Убедитесь, что на каждую цепь схемы подается надлежащее питание.

При устранении неисправностей правило номер один — пошаговое выполнение процедуры и внесение одного изменения за раз. Это пра-

вило вбил в мою юную голову школьный учитель и мой первый работодатель Маурицио Пирола. Каждый раз, когда я что-нибудь отлаживал и все шло не так как надо (поверьте мне, подобное происходит довольно часто), в моей памяти «всплывала» его голова, говорящая: «одно изменение за раз... одно изменение за раз», и это мне очень помогало, когда я занимался ремонтом. Данное правило позволяет точно знать, что именно привело к устранению проблемы. Очень легко запутаться, какое изменение в действительности решило проблему, вот почему важно вносить изменения по одному за раз.

Опыт, приобретенный в процессе отладки, «генерирует» в вашей памяти «базу знаний» по дефектам и способам их исправлений. Однако прежде, чем вы это поймете, вы превратитесь в эксперта. Это серьезно повысит ваш рейтинг в глазах окружающих, и как только какой-нибудь чайник воскликнет «не работает!», вы небрежно бросите на схему проницательный взгляд и мгновенно выдадите рецепт лечения проблемы.

Изолирование проблем

Еще одно важное правило заключается в поиске надежного способа воспроизведения проблемы. Если ваша схема ведет себя странным образом от случая к случаю, постарайтесь проявить упорство в плане точного определения момента возникновения проблемы. Такая процедура позволит тщательно обдумать все возможные причины.

Очень полезно попытаться объяснить кому-то, что именно происходит с вашим устройством. Это действительно отличный способ поиска решения — по возможности максимально точное описание проблемы. Попробуйте найти какого-нибудь внимательного слушателя и начните ему рассказывать о том, что произошло, во многих случаях решение всплывает во время попытки составления точной формулировки. Брайан В. Керниган и Роб Пайк в книге *Practice of Programming* рассказывали об одном университете, где рядом со справочным столом держали плюшевого медведя. Студенты, столкнувшиеся с загадочными ошибками, должны были сначала объяснить свои проблемы медведю, и только потом им разрешалось обратиться к консультанту.

Проблемы с интегрированной средой разработки

В некоторых случаях могут возникать проблемы с интегрированной средой разработки Arduino, особенно в операционной системе Windows.

Если выводится сообщение об ошибке, когда вы дважды щелкаете значок Arduino, и если при этом ничего не происходит, попробуйте в качестве альтернативного способа запуска Arduino дважды щелкнуть по файлу *run.bat*.

Пользователи Windows могут также столкнуться с тем, что операционная система назначает Arduino порт COM10 или с еще большим номером. В этом случае обычно можно заставить Windows назначить Arduino порт с меньшим номером. Для этого откройте **Диспетчер устройств**, нажав кнопку **Пуск**, щелкните правой кнопкой мыши **Компьютер** (Vista) или **Мой компьютер** (Windows XP) и выберите **Свойства**. В Windows XP щелкните по пункту **Оборудование** и выберите **Диспетчер устройств**. В операционной системе Windows Vista щелкните по пункту **Диспетчер устройств** (он отображается в списке задач в левой части окна).

Посмотрите наличие устройств с последовательным интерфейсом в списке под заголовком **Порты (COM и LPT)**. Найдите неиспользуемое устройство с последовательным интерфейсом, которому назначен COM9 или порт с меньшим номером. Щелкните по нему правой кнопкой мыши и выберите в контекстном меню пункт **Свойства**. Затем выберите вкладку **Параметры порта** и щелкните по кнопке **Дополнительно**. Установите номер порта COM10 или выше, нажмите кнопку **ОК**, а потом еще раз нажмите кнопку **ОК**, чтобы закрыть диалоговое окно **Свойства**.

Теперь проделайте то же самое с Arduino, внеся только одно изменение: назначьте ему номер порта COM9 или ниже, словом, тот, который вы только что освободили.

Если все это не помогло или ваша проблема здесь не описана, посетите страницу поиска и устранения неполадок Arduino по адресу www.arduino.cc/en/Guide/Troubleshooting.

Помощь в Интернете

Если дело совсем плохо, не тратьте время на попытки самостоятельно решить проблему, обратитесь за помощью. В сообществе пользователей Arduino вы всегда найдете поддержку, если, разумеется, сможете грамотно или хотя бы членораздельно описать свою проблему.

Возьмите за правило вырезать и вставлять фрагменты кода и другие относящиеся к делу тексты в поисковую систему и смотреть, обсуждает ли кто-нибудь аналогичные вопросы. Например, когда интегрированная среда разработки Arduino выдает неприятное сообщение об ошибке, скопируйте его в поисковую систему Google и посмотрите, что из этого

выйдет. Сделайте то же самое с фрагментами программного кода, над которым вы работаете, или просто вставьте в строку поиска имя конкретной функции. Оглянитесь вокруг: все уже давно изобретено и хранится на какой-нибудь странице в Интернете.

Дальнейшие исследования начните с веб-сайта www.arduino.cc. Ознакомьтесь с часто задаваемыми вопросами (www.arduino.cc/en/Main/FAQ), затем перейдите на игровую площадку (www.arduino.cc/playground), где есть свободно редактируемая доска объявлений, куда любой пользователь может вносить изменения, дабы внести свой посильный вклад в разработку документации. Это одна из лучших составляющих философии открытого программного кода. Люди участвуют в подготовке документации и делятся собственными разработками на основе Arduino. Поэтому, прежде чем начать проект, поищите информацию на игровой площадке, вы почти наверняка найдете там фрагменты программного кода или электрические схемы, которые станут составляющими вашего проекта.

Если не удается найти ответ таким путем, поищите необходимую информацию на форуме (http://arduino.cc/forum/). Если и это не поможет, оставьте на форуме вопрос. Выберите для своей проблемы подходящую доску объявлений (вопросы по программному и аппаратному обеспечению разведены, кроме того, предусмотрены формы на пяти разных языках). Разместите в Интернете как можно больше информации, касающейся вашей проблемы, где среди прочего обязательно укажите следующее:

- Какой платой Arduino вы пользуетесь?
- В какой операционной системе работает интегрированная среда разработки Arduino?
- Дайте общее описание того, что вы пытаетесь делать. Разместите ссылки на спецификации используемых деталей.

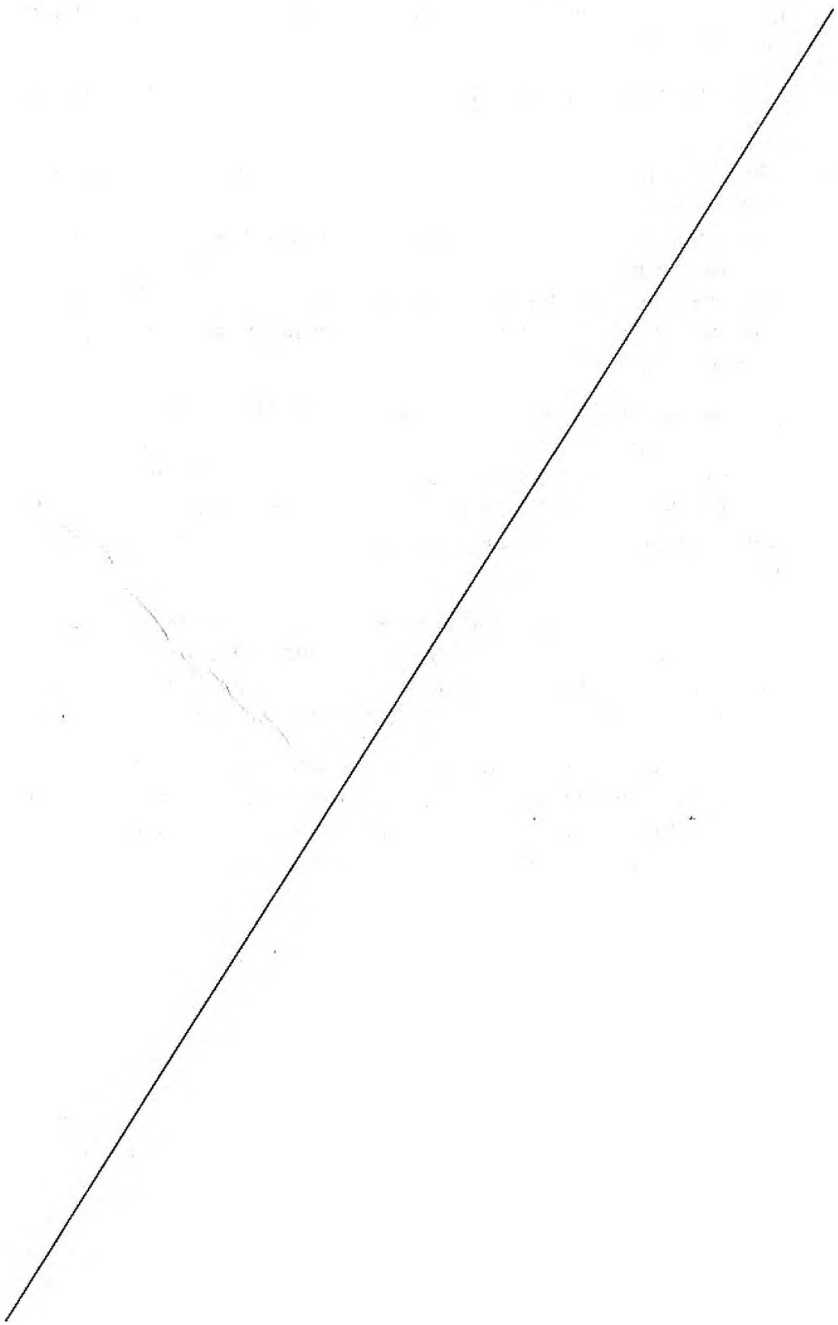
Число ответов будет зависеть от того, насколько хорошо вы сформулировали свой вопрос.

Ваши шансы на успех возрастут, если исключить следующие ошибки (эти правила годятся для любых интернетовских форумов, а не только для форума Arduino):

- Набор сообщений ЗАГЛАВНЫМИ БУКВАМИ. Это раздражает аудиторию примерно так же, как прогулка по научной лаборатории с огромной надписью «ЧАЙНИК» на лбу. В интернет-сообществах на-

бор текста заглавными буквами принимают за разговор на повышенных тонах.

- Размещение одного и того же сообщения на нескольких ветвях форума.
- «Проталкивание» сообщения путем размещения сопроводительных комментариев, содержащих вопрос типа «Ау, неужели никто может ответить?» или, что хуже, текст «Сижу на мели». Если вы не получили ответа, внимательно прочитайте свое сообщение, оставленное на форуме. Внятно ли вы сформулировали тему? Изложена ли проблема понятным языком? Были ли вы тактичны? Всегда будьте деликатны!
- Размещение сообщений типа «Я хочу построить космический аппарат, используя Arduino. Как это сделать?». Выходит, вы хотите, чтобы кто-то сделал вашу работу за вас. Такой подход никому не интересен. Лучше объясните, что вы хотите создать, а затем задайте конкретный вопрос, относящийся к какой-то определенной части проекта. Тогда ответ будет.
- Разновидность предыдущего пункта: если ответ на вопрос явно требует оплаты. Помните, люди будут счастливы ответить на конкретные вопросы, но если вы просите сделать всю вашу работу за вас, причем бесплатно, ответ, скорее всего, будет малоприятным.
- Размещение сообщений, подозрительно смахивающих на вопросы по студенческим курсовым работам. Имейте в виду, преподаватели любят бродить по форумам, а затем с большим удовольствием наказывают таких студентов.



Приложение А. Макетная плата

Отладка схемы подразумевает внесение множества изменений до тех пор, пока она не заработает так, как надо. Этот оперативный итеративный процесс в чем-то сходен с рисованием эскизов. Конструкция растет и изменяется прямо в ваших руках по мере опробования разных комбинаций. Для получения наилучших результатов имеет смысл использовать такую систему, которая позволяет менять компоненты и соединения между ними максимально быстрым, наиболее практичным и наименее разрушительным способом. Все эти требования, разумеется, исключают из рассмотрения пайку, во-первых, потому, что это дело небыстрое и трудоемкое, а во-вторых, потому, что при пайке компоненты подвергаются серьезным термическим нагрузкам.

Всем изложенным выше требованиям отвечает практичное устройство, которое называется беспаечной макетной платой. Как видно из рис. А-1, это небольшая пластмассовая конструкция с большим количеством отверстий, в каждом из которых имеется подпружиненный контакт. Если вставить ножку какого-нибудь компонента в одно из отверстий, он соединится со всеми отверстиями в этом же вертикальном столбце. Все отверстия располагаются на одинаковых расстояниях друг от друга.

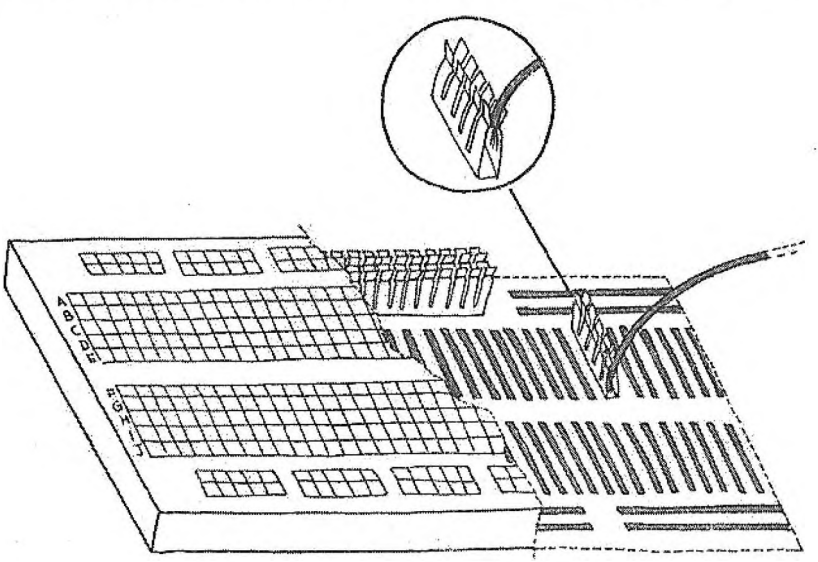


Рис. А-1. Беспаечная макетная плата

Так как большинство компонентов имеет ножки (профессионалы называют их «выводами») со стандартным шагом между ними, микросхемы, имеющие много ножек, прекрасно устанавливаются на плате. Не все контакты на макетной плате одинаковы, есть и различия. Верхний и нижний ряды, окрашенные в красный и синий (черный) цвета и промаркированные знаками «+» и «–», соединены горизонтально и используются для подачи питания на плату. Таким образом, если нам надо запитать какой-то элемент, достаточно установить между ним и этими шинами перемычки (короткие кусочки монтажного провода, которые используются для соединения двух точек в схеме). Наконец, последнее, что нам требуется знать. В середине макетной платы расположен большой зазор, ширина которого равна размеру небольшой микросхемы. Все вертикальные ряды контактов-отверстий в этом месте прерываются таким образом: если мы вставим сюда интегральную микросхему, ее выводы не будут контачить друг с другом. Ловко, не правда ли?

Приложение Б. Маркировка резисторов и конденсаторов

Чтобы работать с электронными компонентами, надо для начала научиться их идентифицировать, что может стать трудной задачей для начинающего. Большинство резисторов, которые можно найти в магазине, имеют цилиндрический корпус с двумя ножками и странными цветными полосками. Дело в том, что когда начали выпускать первые коммерческие резисторы, еще не изобрели способ печати таких маленьких символов, которые умещались бы на их корпусах. Тогда умные инженеры решили, что величину сопротивления можно представить разноцветными колечками.

Сегодняшние конструкторы-любители должны разбираться в этих обозначениях. Ключ к пониманию прост: обычно на корпусе четыре колечка, а их цвет соответствует определенной цифре. Одно из колец обычно золотого цвета, оно говорит о классе точности того резистора, на который нанесено. Чтобы прочитать полосатый код в нужном порядке, держите резистор так, чтобы золотая (или серебряная) полоска находилась справа. Смотрим на цвета и сопоставляем их с цифрами. В следующей таблице показана связь между цветами и их числовыми значениями.

Цвет	Значение
Черный	0
Коричневый	1
Красный	2
Оранжевый	3
Желтый	4
Зеленый	5
Синий	6
Фиолетовый	7
Серый	8
Белый	9
Серебряный	10%
Золотой	5%

Например, коричневое, черное, оранжевое и золотое кольца означают 103±5%. Просто, не правда ли? Не совсем, поскольку третье кольцо в действительности означает не цифру, а число нулей. Поэтому 1 0 3 — это 1 0, а потом еще три нуля. Таким образом, у нас выходит 10 000 ом ± 5%. В электронике тысячи ом называют кило омами, а миллионы ом называют мегаомами. В результате наш резистор на 10 000 ом получит сокращенное обозначение 10к, а резистор на 10 000 000 ом мы запишем как 10М.

Поскольку инженеры любят все оптимизировать, на некоторых принципиальных схемах вы встретите обозначения, похожие на следующее: 47к7. На самом деле это резистор на 4,7 килоом, или на 4700 ом.

С конденсаторами дело обстоит несколько проще: номинал бочкообразных электролитических конденсаторов можно прочесть на их корпусах. Значение емкости измеряется в фарадах (Ф), но емкость большинства конденсаторов, которые вам встретятся, будет измеряться в микрофарадах (мкФ). То есть маркировка 100 мкФ означает емкость 100 микрофарад.

На большинстве дискообразных керамических конденсаторов вы не найдете значения емкости, только трехразрядный числовой код, показывающий количество пикофарад (пФ). Одна мкФ равна 1 000 000 пФ. Так же как и у резисторов, третья цифра здесь используется для определения количества нулей, которые надо поместить после первых двух цифр. Правда, есть и существенное отличие: цифры от 0 до 5 соответствуют количеству нулей, цифры 6 и 7 не используются, а цифры 8 и 9 обрабатываются так: если видите 8, умножайте число, образуемое первыми двумя разрядами, на 0,01, а если видите 9, умножайте это число на 0,1.

Таким образом, конденсатор с маркировкой 104 имеет емкость 100 000 пФ, или 0,1 мкФ. Конденсатор с маркировкой 229 имеет емкость 2,2 пФ.

Приложение В. Краткий справочник по Arduino

Здесь приводится краткое описание стандартных команд, поддерживаемых языком Arduino.

Более подробные сведения смотрите на веб-странице: arduino.cc/en/Reference/HomePage.

Структура

Скетч Arduino состоит из двух частей:

```
void setup()
```

Это фрагмент программы, куда помещается код инициализации — блок команд, устанавливающий плату в состояние, необходимое для запуска основного цикла программы.

```
void loop()
```

Этот фрагмент является основным блоком программы. Он состоит из набора команд, которые многократно повторяются до тех пор, пока не будет выключено питание платы.

Специальные символы

Язык Arduino содержит ряд символов, предназначенных для разделения строк, комментариев и блоков.

; (точка с запятой)

Каждая команда (строка кода) завершается точкой с запятой. Данный синтаксис позволяет произвольно форматировать код. Можно поместить две команды в одну строку, если разделить их точкой с запятой. (Правда, это сделает код более трудным для восприятия.)

Пример:

```
delay(100);
```

{} (фигурные скобки)

Эти символы используются для определения блоков кода. Например, если вы пишете команды, входящие в цикл loop (), вам необходимо использовать фигурные скобки в начале и в конце этого блока.

Пример:

```
void loop()   {
   Serial.println("bye");
}
```

Комментарии

Это фрагменты текста, игнорируемые процессором Arduino, но чрезвычайно полезные для напоминания себе (или другим) о том, для чего предназначен данный фрагмент кода.

Существует два стиля комментариев в Arduino:

```
// одиночная строка: этот текст игнорируется
// до конца строки
/* многострочный:
   здесь можно написать
   целую поэму
/*
```

Константы

Arduino содержит набор предопределенных ключевых слов, каждому из которых соответствует свое значение. Например, `HIGH` и `LOW` используются, когда необходимо включить или выключить вывод платы Arduino. Константы `INPUT` (ВХОД) и `OUTPUT` (ВЫХОД) используются для установки конкретного вывода в качестве входа или выхода.

`TRUE` (ИСТИНА) и `FALSE` (ЛОЖЬ) в точности соответствуют их названиям: истинность или ложность условия или выражения.

Переменные

Переменные являются именованными областями памяти платы Arduino, где хранятся данные, которые можно использовать и обрабатывать в пользовательской программе. Переменные могут менять значение столько раз, сколько необходимо.

Так как Arduino представляет собой очень простой процессор, то при объявлении переменной необходимо указать ее тип. Это позволяет сообщить процессору о размере переменной, которую нужно хранить.

Далее перечисляются доступные типы данных:

boolean (логический тип)

Могут принимать два значения: True (Истина) и False (Ложь).

char (символьный тип)

Содержит один символ, например «A». Как и любой другой компьютер, Arduino хранит символы в виде чисел, даже если это текст. Когда символы используются для хранения чисел, они могут принимать значения от –128 до 127.

Примечание

Существует два набора символов, применяемых в компьютерах: ASCII и UNICODE. ASCII — это набор из 127 символов, который использовался среди прочего для передачи информации между последовательными терминальными устройствами и вычислительными машинами, работающими по принципу разделения времени, такими как мэйнфреймы (большие ЭВМ) и мини-ЭВМ.

UNICODE представляет собой достаточно объемный набор, используемый в программном обеспечении современных компьютеров для представления символов в широком диапазоне языков. ASCII по-прежнему удобен для обмена короткими фрагментами информации на европейских языках, в которых используются латинский алфавит, арабские цифры и стандартные символы пунктуации.

---

byte (байт)

Представляет число от 0 до 255. Так же как и символ, он занимает только один байт памяти.

int (целое число)

Использует 2 байта памяти для представления целых чисел от –32 768 до 32 767. Это самый распространенный тип данных, используемых в Arduino.

unsigned int (целое число без знака)

Так же как int, использует 2 байта; префикс unsigned означает, что этот тип переменных не позволяет хранить отрицательные числа, поэтому его значения находятся в диапазоне от 0 до 65 535.

long (длинное целое число)

Этот тип в два раза больше, чем int, и содержит числа в диапазоне от –2 147 483 648 до 2 147 483 647.

unsigned long (длинное целое число без знака)

Версия long без знака позволяет записывать числа от 0 до 4 294 967 295.

float (число с плавающей запятой)

Данный тип переменных позволяет хранить числа с десятичной дробью. Данный тип занимает 4 байта ценной памяти, и функции, которые обрабатывают переменные этого типа, также требуют немало пространства. Поэтому старайтесь как можно реже использовать числа с плавающей запятой.

double (число двойной точности)

Число с плавающей запятой двойной точности с максимальным значением 1,7976931348623157 × 10308. Круто!

string (строка)

Набор ASCII-символов для хранения текстовой информации (строку можно использовать для отправки сообщения через последовательный порт или для отображения на жидкокристаллическом дисплее). Для хранения переменной этого типа нужно по одному байту на каждый символ в строке плюс знак Null, чтобы сообщить Arduino об окончании строки. Следующие выражения эквивалентны друг другу:

```
Char string1[] = "Arduino"; // 7 знаков + один знак null
Char string2[8] = "Arduino"; // То же, что и выражение
                             // вверху
```

array (массив)

Список переменных, доступ к которым можно организовать посредством индекса. Массив используется для построения таблиц значений, к которым предоставляется простой доступ. Например, если вы хотите хранить разные уровни яркости, используемые для плавного управления светодиодом, можно создать шесть переменных с именами light01, light02 и так далее. Однако было бы удобнее использовать простой массив, подобный следующему:

```
int light[6] = {0, 20, 50, 75, 100}
```

Слово array (массив) в действительности не используется в объявлении переменной: его роль выполняют символы [ ] и { }.

Управляющие конструкции

Arduino содержит ключевые слова для управления логической последовательностью выполнения команд в программе.

if...else

Данная конструкция позволяет принимать решения. За выражением `if` должен следовать вопрос, задаваемый в виде выражения, заключенного в круглые скобки. Если выражение справедливо, выполняется программный код, следующий за этим выражением. Если выражение ложно, будет выполняться блок программного кода, следующий за `else`. Допускается использовать только `if` без предоставления выражения `else`.

Пример:

```
if (val == 1) {
  digitalWrite(LED,HIGH);
}
```

for

Позволяет повторять блок программного кода указанное число раз.

Пример:

```
for (int i = 0; i < 10; i++) {
   Serial.print("ыао");
}
```

switch case

Оператор if похож на развилку дорог для вашей программы, а `switch case` напоминает большую транспортную развязку. Этот оператор позволяет программе выполняться по разнообразным направлениям в зависимости от значения переменной. Данный оператор очень удобен, поскольку заменяет длинные списки условных операторов `if`.

Пример:

```
switch (sensorValue) {
   case 23:
     digitalWrite(13,HIGH);
     break;
   case 46:
     digitalWrite(12,HIGH);
     break;
  default: // если совпадений нет, выполняется этот блок
     digitalWrite(12,LOW);
     digitalWrite(13,LOW);
}
```

while

Аналогично оператору `if`, данным оператором выполняется блок программного кода при определенном условии.

Пример:

```
// Светодиод мигает, пока значение датчика меньше 512
sensorValue = analogRead(1);
while (sensorValue < 512) {
   digitalWrite(13,HIGH);
   delay(100);
   digitalWrite(13,HIGH);
   delay(100);
   sensorValue = analogRead(1);
}
```

do...while

Практически такой же, как и оператор `while`, за исключением того, что программный код выполняется непосредственно до оценки условия. Данная конструкция используется, когда нужно, чтобы код внутри блока выполнялся хотя бы раз до проверки условия.

Пример:

```
do {
  digitalWrite(13,HIGH);
  delay(100);
  digitalWrite(13,HIGH);
  delay(100);
  sensorValue = analogRead(1);
} while (sensorValue < 512);
```

break (прервать)

Данный элемент позволяет выйти из цикла и продолжить выполнение кода, располагающегося после цикла. Он также используется для разделения блоков оператора switch case.

Пример:

```
// Светодиод мигает, пока значение датчика меньше 512
do {
   // Выход из цикла, если нажата кнопка
   if (digitalRead(7) == HIGH)
     break;
   digitalWrite(13,HIGH);
   delay(100);
   digitalWrite(13,HIGH);
   delay(100);
   sensorValue = analogRead(1);
} while (sensorValue < 512);
```

continue (продолжить)

Внутри цикла оператор continue позволяет пропустить оставшийся за ним код и вызывает повторную проверку условия.

Пример:

```
for (light = 0; light < 255; light++)
{
 // пропуск интенсивностей между 140 и 200
 if ((x > 140) && (x < 200))
   continue;
 analogWrite(PWMpin, light);
 delay(10);
}
```

return (вернуть)

Останавливается выполнение функции и происходит возврат из нее. Этот оператор можно также использовать для возврата значения из функции.

Например, если имеется функция `computeTemperature()` и требуется вернуть результат во фрагмент кода, вызвавший эту функцию, можно было бы написать следующий код:

```
int computeTemperature() {
    int temperature = 0;
    temperature = (analogRead(0) + 45) / 100;
    return temperature;
}
```

Арифметика и формулы

Arduino может применяться для выполнения сложных вычислений, использующих специальный синтаксис. Для операций сложения и вычитания используются символы + и –, умножение задается знаком *, а деление — знаком /.

Существует дополнительный оператор, называемый `modulo (%)`, который возвращает остаток целочисленного деления. Можно использовать столько уровней скобок, сколько необходимо, чтобы сгруппировать выражения. В отличие от того, что вы изучали в школе, квадратные и фигурные скобки зарезервированы для других целей (индексов массивов и блоков кода).

Примеры:

```
a = 2 + 2;
light = ((12 * sensorValue) - 5 ) / 2;
remainder = 3 % 2; // возвращает 2,
                   // так как 3 / 2 дает остаток
```

Операторы сравнения

Когда вы задаете условия или проверки для операторов `if`, `while` и `for`, можно использовать следующие знаки операций:

```
== равно
!= не равно
< меньше, чем
> больше, чем
<= меньше или равно
>= больше или равно
```

Логические операторы

Данные операторы используются, когда требуется объединить несколько условий. Например, если вы хотите проверить, находится ли значение сигнала, поступающего от датчика, между 5 и 10, можно было бы написать такой код:

```
if ((sensor => 5) && (sensor <=10))
```

Существует три логических оператора: И, представляемое с помощью символов &&; ИЛИ, представляемое с помощью символа ||; и наконец, НЕ, представляемое с помощью символа !.

Составные операторы

Эти специальные операторы применяются для сокращения программного кода при некоторых весьма распространенных операциях, таких как приращение значения.

Например, чтобы увеличить value на 1, можно было бы написать следующий код:

```
value = value +1;
```

При помощи составного оператора это действие записывается короче:

```
value++;
```

инкремент и декремент (— и ++)

Эти операторы увеличивают или уменьшают значение на 1. Однако будьте внимательны. Если записать i++, выполняется приращение i на 1, затем вычисляется значение i + 1; при записи ++i сначала вычисляется значение i, затем выполняется приращение на 1. То же справедливо и для оператора —.

+=, -=, *= и /=

Эти операторы позволяют сократить некоторые выражения. Следующие два выражения эквивалентны друг другу:

```
a = a + 5;
a += 5;
```

Функции для входов и выходов

Язык Arduino содержит функции для обработки входных и выходных сигналов. Некоторые из них вы уже видели в примерах, приведенных в книге.

pinMode(pin, mode)

Изменяет конфигурацию цифрового вывода для работы в качестве входа или выхода.

Пример:

```
pinMode(7,INPUT); // устанавливает вывод 7 в качестве входа
```

digitalWrite(pin, mode)

Включает или выключает цифровой вывод. Прежде чем будет действовать функция `digitalWrite`, выводы должны быть в явном виде установлены в качестве выхода с помощью функции `pinMode`.

Пример:

```
digitalWrite(8,HIGH); // включает цифровой вывод 8
```

int digitalRead(pin)

Считывает состояние входного контакта, возвращает значение HIGH, если на выводе определяется некоторое напряжение, или LOW, если напряжения там нет.

Пример:

```
val = digitalRead(7); // Считывание напряжения
                      // на выводе 7 в переменную val
```

int analogRead(pin)

Считывает напряжение на аналоговом входе и возвращает число от 0 до 1023, которое представляет напряжения в диапазоне от 0 до 5 В.

Пример:

```
val = analogRead(0); // Считывание значения
                     // аналогового входа 0
                     // в переменную val
```

analogWrite(pin, value)

Изменяет частоту широтно-импульсной модуляции на одном из выводов с маркировкой «PWM». Аргумент pin (вывод) может принимать значения 11, 10, 9, 6, 5, 3. Аргумент value (значение) может быть числом от 0 до 255, которое соответствует диапазону выходного напряжения от 0 до 5 В.

Пример:

```
analogWrite(9,128); // Уменьшение яркость свечения
                    // светодиода на выводе 9 до 50%
```

shiftOut(dataPin, clockPin, bitOrder, value)

Отправляет данные в сдвиговые регистры — внутренние механизмы микроконтроллера, которые используются для расширения числа цифровых выходов. Данным протоколом используется один вывод для данных и один для тактовых импульсов. Аргумент bitOrder показывает порядок представления байтов (самый младший или самый старший), и аргумент value является передаваемым байтом.

Пример:

```
shiftOut(dataPin, clockPin, LSBFIRST, 255);
```

функция pulseIn(pin, value) в формате длинного целого числа без знака

Измеряет длительность импульса, поступающего на один из цифровых входов. Данная функция полезна, например, для анализа сигналов некоторых инфракрасных датчиков или акселерометров, которые на выходе выдают импульсы меняющейся длительности.

Пример:

```
time = pulseIn(7,HIGH); // Измеряет время, в течение
  // которого уровень следующего импульса остается высоким
```

Функции времени

Язык Arduino содержит функции для измерения истекшего времени, а также для приостановки выполнения программы.

функция millis() в формате длинного целого числа без знака

Возвращает число миллисекунд, прошедших с момента запуска программы.

Пример:

```
duration = millis()-lastTime; // Вычисляет время,
                 // прошедшее с момента последнего запуска
```

delay(мс)

Приостанавливает выполнение программы на указанное количество миллисекунд.

Пример:

```
delay(500); // Останавливает выполнение программы
            // на половину секунды
```

delayMicroseconds(мкс)

Приостанавливает выполнение программы на указанное количество микросекунд.

Пример:

```
delayMicroseconds(1000); // Ожидание в течение
                         // 1 миллисекунды
```

МАТЕМАТИЧЕСКИЕ ФУНКЦИИ

В языке Arduino имеется много распространенных математических и тригонометрических функций.

min(x, y)

Возвращает меньшее из значений *x* и *y*.

Пример:

```
val = min(10,20); // Теперь val равно 10
```

max(x, y)

Возвращает большее из значений *x* и *y*.

Пример:

```
val = max(10,20); // Теперь val равно 20
```

abs(x)

Возвращает абсолютное значение *x*, то есть отрицательные значения превращаются в положительные. Если x равно 5, функция возвратит 5, но если *x* равно –5, функция по-прежнему возвратит 5.

Пример:

```
val = abs(-5); // Теперь val равно 5
```

constrain(x, a, b)

Возвращает значение *x*, ограниченное *a* и *b*. Если x меньше, чем *a*, функцией возвращается *a*, и если *x* больше, чем *b*, функцией возвращается *b*.

Пример:

```
val = constrain(analogRead(0), 0, 255);
                                  // Отклоняет значения
```

map(значение, изНизкий, изВысокий, вНизкий, вВысокий)

Отображает значение в диапазоне *изНизкий — изВысокий* в диапазон *вНизкий — вВысокий*. Очень удобна для обработки сигналов аналоговых датчиков.

Пример:

```
val = map(analogRead(0),0,1023,100, 200);
        // отображает значение аналогового сигнала
        // от 0 до 1023 в значение между 100 и 200
```

Функция pow(основание, экспонента) двойной точности

Возвращает результат возведения числа (основания) в степень (экспонента).

Пример:

```
double x = pow(y, 32); // Устанавливает x равным
            // значению y, возведенному в 32-ю степень
```

Функция sqrt(x) двойной точности

Возвращает квадратный корень числа.

Пример:

```
a (двойной точности) = sqrt(1138);
                    // приблизительно 33,73425674438
```

Функция sin(радианы) двойной точности

Возвращает синус угла, заданного в радианах.

Пример:

```
sine (двойной точности) = sin(2);
                    // приблизительно 0,90929737091
```

Функция cos(радианы) двойной точности

Возвращает косинус угла, заданного в радианах.

Пример:

```
cosine (двойной точности) = cos(2);
                    // приблизительно -0,41614685058
```

Функция tan(радианы) двойной точности

Возвращает тангенс угла, заданного в радианах.

Пример:

```
tangent (двойной точности) = tan(2);
                          // приблизительно -2,18503975868
```

Функции случайных чисел

Если вам необходимо сгенерировать случайные числа, можно воспользоваться генератором псевдослучайных чисел в Arduino.

randomSeed(начальное число)

Сбрасывает в исходное состояние генератор псевдослучайных чисел. Хотя распределение чисел, возвращаемых функцией `random()`, по существу, случайное, их последовательность вполне предсказуема, поэтому следует сбросить генератор в какое-нибудь случайное значение. Если на плате имеется неподсоединенный аналоговый вывод, он собирает шум из окружающей среды (радиоволны, космические лучи, электромагнитные помехи от сотовых телефонов, люминесцентного излучения и т. д.).

Пример:

```
randomSeed(analogRead(5)); // генерирует случайное
     // число, используя шум от аналогового входа 5
```

Функции random(max) и random(min, max) в формате длинного целого числа

Возвращает псевдослучайное длинное целое число в диапазоне между min и max — 1. Если аргумент min не указан, нижняя граница равна 0.

Пример:

```
randnum (длинное целое число) = random(0, 100);
                          // Число между 0 и 100
randnum (длинное целое число) = random(11);
                          // Число между 0 и 11
```

Последовательный интерфейс

Как описывалось в главе 5, в Arduino предусмотрена возможность взаимодействия с устройствами через USB-порт по протоколу последовательного интерфейса. Ниже приводятся функции последовательного интерфейса.

Serial.begin(speed)

Подготавливает плату Arduino к началу отправки или получения последовательных данных. Обычно для связи с интегрированной средой разработки используется скорость передачи 9600 бит/сек, но доступны и другие скорости, обычно не превышающие значения 115 200 бит/сек.

Пример:

```
Serial.begin(9600);
```

Serial.print(данные) Serial.print(данные, кодировка)

Отправляет данные в последовательный порт. По желанию можно задать кодировку. Если кодировка не указывается, данные обрабатываются как обычный текст.

Примеры:

```
Serial.print(75); // Печать "75"
Serial.print(75, DEC); // То же, что и вверху.
Serial.print(75, HEX); // "4B" (75 в шестнадцатеричном виде)
Serial.print(75, OCT); // "113" (75 в восьмеричном виде)
Serial.print(75, BIN); // "1001011" (75 в двоичном виде)
Serial.print(75, BYTE); // "K" (необработанный байт
                        // соответствует 75
                        // в наборе ASCII-символов)
```

Serial.println(данные) Serial.println(данные, кодировка)

Подобна функции `Serial.print()` за исключением того, что добавляются символы возврата каретки и перевода строки (\r\n), как если бы вы набирали данные, а затем нажимали клавиши Return (Возврат) или Enter (Ввод).

Примеры:

```
Serial.println(75); // Печать "75\r\n"
Serial.println(75, DEC); // То же, что и вверху.
Serial.println(75, HEX); // "4B\r\n"
Serial.println(75, OCT); // "113\r\n"
Serial.println(75, BIN); // "1001011\r\n"
Serial.println(75, BYTE); // "K\r\n"
```

Функция Serial.available() целочисленного типа

Возвращает количество непрочитанных байтов, доступных на последовательном порте для чтения с помощью функции `read()`. После того как все доступное прочитано с помощью функции `read()`, функция `Serial.available()` будет возвращать 0 до тех пор, пока в последовательный порт не поступят новые данные.

Пример:

```
int count = Serial.available();
```

Функция Serial.read() целочисленного типа

Выбирает один байт из поступающих последовательных данных.

Пример:

```
int data = Serial.read();
```

Serial.flush()

Поскольку данные могут поступать через последовательный порт быстрее, чем их может обрабатывать ваша программа, Arduino хранит их в буфере. Если необходимо очистить буфер и предоставить возможность его заполнения новыми данными, используйте функцию `flush()`.

Пример:

```
Serial.flush();
```

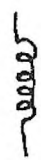
Приложение Г.
Чтение принципиальных схем

До сих пор для того, чтобы наглядно описать процесс сборки наших схем, мы пользовались очень подробными иллюстрациями. Понятно, что это не самый быстрый способ — рисовать целую картину для каждого любого эксперимента, который вы хотите оформить документально.

Аналогичные вопросы рано или поздно возникают в каждой дисциплине. К примеру, композитор, сочинив чудесную песню, сохраняет ее на бумаге, пользуясь нотной записью.

Инженеры, люди практичные, разработали быстрый и удобный способ фиксации сути электронной схемы, чтобы иметь возможность оформить ее документально и позднее повторно собрать ее или передать схему кому-то еще.

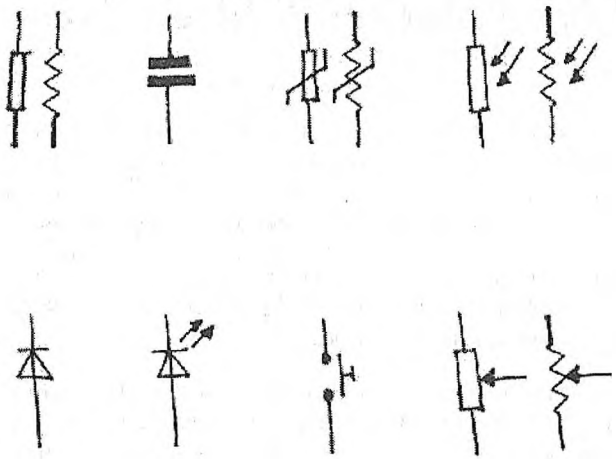
В электронике принципиальные схемы позволяют описать электронные устройства таким образом, чтобы они были понятны для остальных членов сообщества. Каждый компонент представляется символом, который является в некотором роде абстрактным отражением формы компонента или его назначения. Например, конденсатор изготавливается из двух металлических пластин, разделенных воздухом или пластмассой. Поэтому его обозначение выглядит так:



Индуктивность изготавливается путем навивания медной проволоки на катушку цилиндрической формы. Соответственно, обозначение индуктивности выглядит следующим образом:

Соединения между компонентами обычно представляют собой кусочки проволоки или дорожки на печатной плате, поэтому выглядят на принципиальной схеме они как обычные линии. Место соединения двух проводников указывается большой точкой:

Это все, что требуется для понимания базовых схем. Далее приводится более полный перечень символов и их значений:



Вам могут встретиться изменения в этих символах (например, здесь показаны оба варианта обозначения резистора). Более подробный список обозначений, применяемых в электронике, смотрите в Интернете по адресу en.wikipedia.org/wiki/Electronic_symbol. По соглашению принципиальные схемы рисуют слева направо. Например, схема радиоприемника начиналась с антенны слева, затем следовал бы тракт прохождения сигнала к громкоговорителю (справа).

Следующая принципиальная схема описывает пример с кнопкой, который я приводил в этой книге: