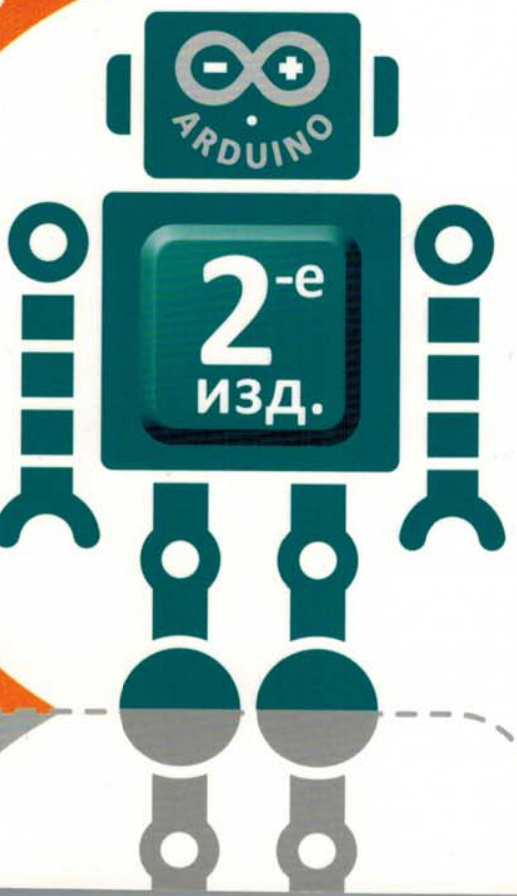
Саймон Монк

Программируем Arduino

Основы работы со скетчами

БЕСТСЕЛЛЕР AMAZON



ПИТЕР

Саймон Монк. Программируем Arduino

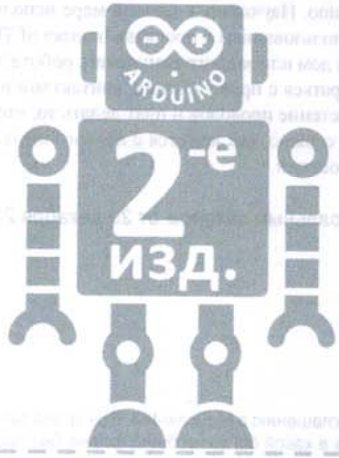
Simon Monk

Programming Arduino

Getting Started With Sketches

Саймон Монк

Программируем Arduino



Основы работы со скетчами

ПИТЕР

Санкт-Петербург · Москва · Екатеринбург · Воронеж
Нижний Новгород · Ростов-на-Дону
Самара · Минск

2017

ББК 32.973.23-018.2
УДК 004.3
М77

Монк С.

М77 Программируем Arduino: Основы работы со скетчами. 2-е изд. — СПб.: Питер, 2017. — 208 с.: ил.

ISBN 978-5-496-02562-1

Познакомьтесь с обновленной версией легендарного бестселлера Саймона Монка. Это издание представляет собой полностью обновленную книгу, основанную на Arduino 1.6.

С момента выхода первого издания многое изменилось: появились новые платы и устройства, использующие язык Arduino. Научитесь в полной мере использовать все возможности Arduino и познакомьтесь с его использованием в проектах Internet of Things.

Хотите создать умный дом или запрограммировать робота? Нет ничего проще. Саймон Монк не только поможет разобраться с проволочками, контактами и датчиками, но и покажет, как заставить все это хитросплетение проводов и плат делать то, что вам нужно.

Arduino — это не так сложно, как кажется с первого взгляда. Вы сразу будете покорены открывающимися возможностями.

12+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.23-018.2
УДК 004.3

ISBN 978-1259641633 англ.
ISBN 978-5-496-02562-1

Моим мальчикам, Стивену и Мэттью,
от очень гордого папы.

Саймон Монк

Содержание

Об авторе

Саймон Монк имеет степень бакалавра кибернетики и информатики, а также доктора наук в области программной инженерии. Со школьных лет активно увлекался электроникой и написал несколько статей для радиолюбительских журналов. Автор почти 20 книг по электронике и конструированию, бóльшая часть из которых посвящена Arduino и Raspberry Pi. Более полную информацию о его книгах можно найти по адресу: *http://simonmonk.org*. Желающие могут подписаться на него в Twitter, где он зарегистрирован как @simonmonk2.

Предисловие

Первое издание этой книги было опубликовано в ноябре 2011 года и попало в топ лучших книг по Arduino на сайте Amazon.

Во время работы над первым изданием книги текущей моделью Arduino была Arduino 2009 с версией программного обеспечения Beta018. Практически одновременно с книгой на полках магазинов появилась модель Arduino Uno с версией программного обеспечения 1.0. Это издание представляет собой полностью обновленную книгу, основанную на Arduino 1.6.

Модель Arduino Uno R3 все еще считается стандартной платой Arduino. Однако появились многие другие платы, включая официальные модели Arduino (например, Leonardo, Zero, 101, Due и Yun) и другие устройства, такие как Photon и Intel Edison, для программирования которых также используется язык Arduino.

В этом издании также описывается использование Arduino в проектах для Интернета вещей (Internet of Things, IoT) и применение разных видов устройств отображения, включая индикаторы на органических светодиодах (OLED) и жидких кристаллах (LCD).

Саймон Монк

Благодарности

Спасибо Линде, что дала мне возможность написать эту книгу, поддерживала меня и мирилась с беспорядком, который порождают мои проекты.

Большое спасибо Роберту «BobKat» Логану (Robert Logan) и многим другим вдумчивым и отзывчивым читателям, сообщившим об ошибках, обнаруженных в первом издании. Я приложил все усилия, чтобы исправить все, что не укрылось от ваших глаз.

Наконец, я хочу сказать спасибо Майклу МакКейби (Michael McCabe), Сришти Маласи (Srishti Malasi) и всем, кто причастен к изданию этой книги. Я получил удовольствие от работы с такой замечательной командой.

Вступление

Интерфейсные платы Arduino представляют недорогую и простую возможность создания проектов на основе микроконтроллеров. Обладая начальными знаниями в области электроники, вы сможете заставить свою плату Arduino делать все что угодно — от управления лампами в творческих инсталляциях до распределения мощности в солнечной энергосистеме.

Существует множество книг, описывающих проекты и демонстрирующих, как подключать внешние устройства к плате Arduino, в том числе книга «30 Arduino Projects for the Evil Genius», написанная автором. Однако в данной книге основное внимание уделяется вопросам программирования Arduino.

Эта книга расскажет, как превратить программирование для Arduino в увлекательное занятие и избежать сложностей с несовместимостью кода, которые так часто доставляют неприятности. Она шаг за шагом проведет вас через все этапы программирования на языке C, на котором пишут программы для Arduino.

Что такое Arduino

Arduino — маленькая плата микроконтроллера с разъемом USB для подключения к компьютеру и множеством контактов для соединения проводами с внешними устройствами, такими как электромоторы, реле, фотоэлементы, светодиоды, громкоговорители, микрофоны и многое другое. Она может питаться от разъема USB компьютера, от 9-вольтовой батареи или другого источника электропитания. Платой можно управлять с компьютера, точно так же ее можно запрограммировать, и после отсоединения от компьютера она будет работать автономно.

Плата имеет открытую архитектуру. То есть любой желающий может создавать свои Arduino-совместимые платы. В результате конкуренция между производителями ведет к снижению стоимости плат.

В дополнение к основным платам выпускаются платы расширения, которые можно подключать к платам Arduino.

Программное обеспечение, необходимое для программирования Arduino, также является открытым, имеются версии для Windows, Mac и Linux.

Что необходимо для чтения книги

Эта книга адресована начинающим любителям, но даже те, кто имеет опыт работы с Arduino и желает узнать больше о программировании этого микроконтроллера или получить более четкое представление об основах, найдут здесь немало полезного для себя. Основное внимание в этой книге уделяется модели Arduino Uno; однако практически все примеры кода без каких-либо изменений будут работать на всех моделях и вариантах Arduino.

От вас не требуется иметь опыт программирования или познания в радиоэлектронике, и упражнения в книге не потребуют от вас орудовать паяльником. Все, что вам нужно, — желание творить.

Если вы захотите получить от книги максимум возможного и провести некоторые из предлагаемых экспериментов, тогда вам пригодится:

- немного изолированного провода;
- недорогой мультиметр.

И то и другое можно недорого купить в ближайшем магазине радиодеталей или в интернет-магазине, таком как Adafruit или Sparkfun. И конечно же, вам понадобится плата Arduino Uno. Желающие пойти еще дальше и поэкспериментировать с дисплеями и подключением к сети должны будут купить эти платы, например в интернет-магазине. Подробности смотрите в главах 9 и 10.

Как работать с этой книгой

Эта книга организована так, чтобы помочь начинающим постепенно двигаться от простого к сложному, усваивать новые сведения, опираясь на уже полученные. Однако вы можете пропустить какие-то начальные главы или бегло пролистать их и сразу перейти к интересующей вас главе.

Книга содержит следующие главы.

- **Глава 1 «Знакомьтесь: Arduino».** Начальное знакомство с платой Arduino. Эта глава описывает возможности и разные типы плат Arduino.
- **Глава 2 «Начало».** Здесь вы проведете первые эксперименты со своей платой Arduino: установите программное обеспечение, включите плату и выгрузите на нее свой первый скетч.
- **Глава 3 «Основы языка С».** Эта глава охватывает основы языка программирования С; для начинающих она может стать также введением в программирование вообще.
- **Глава 4 «Функции».** В этой главе описываются ключевые понятия создания и использования функций в скетчах Arduino.

На протяжении всей главы будут демонстрироваться примеры исходного кода действующих скетчей.

- **Глава 5 «Массивы и строки».** Здесь вы узнаете, как определять и использовать структуры данных, более сложные, чем простые целочисленные переменные. В этой главе будет постепенно реализован проект «Morse» для демонстрации описываемых понятий.
- **Глава 6 «Ввод и вывод».** Расскажет, как управлять цифровыми и аналоговыми вводами и выводами платы Arduino в программах. Здесь очень пригодится мультиметр, с помощью которого можно будет увидеть, что происходит на контактах ввода/вывода платы Arduino.
- **Глава 7 «Стандартная библиотека Arduino».** Описывает особенности использования функций из стандартной библиотеки Arduino.
- **Глава 8 «Запись данных».** Здесь вы узнаете, как писать скетчи, способные записывать данные в электрически стираемые программируемые постоянные запоминающие устройства (ЭСППЗУ) и использовать встроенную флеш-память Arduino.
- **Глава 9 «Дисплеи».** В этой главе вы узнаете, как оборудовать плату Arduino дисплеем, и создадите простое устройство отображения сообщений через USB.
- **Глава 10 «Arduino и Интернет вещей».** Научит, как превратить плату Arduino в веб-сервер и взаимодействовать со службами в Интернете, такими как dweet и IFTTT.
- **Глава 11 «C++ и библиотеки».** Здесь вы выйдете за рамки языка С, познакомитесь с объектно-ориентированным программированием и приемами создания собственных библиотек для Arduino.

Ресурсы

В поддержку этой книги создан вспомогательный веб-сайт *www.arduinobook.com*. Там вы найдете файлы с исходным кодом для всех примеров, которые приводятся в этой книге, а также другие ресурсы, такие как список ошибок и опечаток.

1

Знакомьтесь: Arduino

Arduino — это микроконтроллерная платформа, завладевшая воображением радиолюбителей-энтузиастов. Простота использования и открытая природа делают ее превосходным выбором для всех, у кого есть желание создавать электронные устройства.

Она позволяет подключать дополнительные электронные устройства к своим контактам и управлять ими, например включать и выключать лампочки или моторчики или получать сигналы от датчиков, таких как фотоэлементы и термодатчики. Именно поэтому Arduino иногда описывают термином Physical Computing[1]. Платы Arduino можно подключать к компьютеру посредством USB (Universal Serial Bus — универсальная последовательная шина), а это означает, что их можно использовать как интерфейсные платы для управления внешней электроникой с помощью компьютера.

Эта глава знакомит с платформой Arduino, в том числе с историей ее появления, а также дает краткий обзор аппаратных средств.

Микроконтроллеры

Основой Arduino является микроконтроллер. Практически все остальное, что присутствует на плате, либо обеспечивает электропитание платы, либо служит для взаимодействия с компьютером.

Микроконтроллер — это маленький однокристальный компьютер. Он имеет все, что имели первые домашние компьютеры, и даже больше. У него есть процессор, 1–2 Кбайт оперативной памяти (ОЗУ) для хранения данных, несколько килобайт стираемой программируемой постоянной памяти (СППЗУ — Erasable Programmable Read-Only Memory, EPROM) или флеш-памяти для программ и несколько контактов для входных и выходных сигналов. Эти контакты позволяют связать микроконтроллер с другими электронными устройствами.

Входные сигналы могут быть цифровыми (с двумя состояниями: «включено» и «выключено») и аналоговыми (с изменяемым уровнем напряжения). Это позволяет подключать самые разные датчики, такие как фотоэлементы, термодатчики, акустические датчики и многие другие.

---

[1] Термин *Physical Computing* (физические вычисления) в широком смысле описывает создание физических интерактивных систем на основе программных и аппаратных компонентов, которые способны воспринимать сигналы мира и реагировать на них. — *Примеч. пер.*

Выходные сигналы также могут быть цифровыми или аналоговыми. То есть можно подавать на выходной контакт напряжение 0 или 5 В и выключать или включать светодиоды непосредственно или управлять включением питания более мощных устройств, таких как электромоторы. Аналоговые выходы позволяют плавно изменять выходное напряжение. То есть на выходной контакт можно подавать определенное напряжение и таким способом управлять скоростью вращения электромотора или яркостью лампочки, а не только включать и выключать их.

Микроконтроллер на плате — это микросхема с 28 ножками, которая вставляется в разъем в центре платы. Она содержит и память, и процессор, и всю остальную электронику поддержки входов и выходов. Выпускаются эти микросхемы компанией Atmel, ведущим производителем микроконтроллеров. Каждый производитель выпускает десятки разных микроконтроллеров, объединенных в семейства. Вообще говоря, микроконтроллеры создаются не ради забавы любителей, таких как мы с вами. Мы — лишь малая часть огромного рынка. Эти устройства предназначены для встраивания в потребительские товары, в том числе автомобили, стиральные машины, DVD-плееры, детские игрушки, кондиционеры и т. п.

Самое замечательное в платформе Arduino — то, что она сужает ошеломляющую широту выбора, стандартизировав единственный микроконтроллер, и продолжает придерживаться этого стандарта. (Это утверждение не совсем верно, как вы увидите в дальнейшем, но довольно близко к истине.)

Это означает, что, приступая к новому проекту и выбирая микроконтроллер, вам не придется взвешивать все «за» и «против».

Макетные платы

Итак, мы установили, что микроконтроллер — это в действительности единственная микросхема. Но микросхема не будет работать сама по себе, если не обеспечить ее электропитанием с определенным напряжением (микроконтроллеры весьма требовательны к электропитанию), а также средствами сопряжения с компьютером для программирования микроконтроллера. Для этой цели используются макетные платы. Плата Arduino является макетной платой микроконтроллера с открытой архитектурой. Это означает, что файлы с проектом печатного монтажа платы и рисунками общедоступны и любой желающий может использовать их для налаживания производства и продажи своих плат Arduino.

Все производители микроконтроллеров, включая компанию Atmel, которая производит микроконтроллеры ATmega328, используемые на платах Arduino, производят собственные макетные платы и программное обеспечение для программирования контроллеров. Несмотря на невысокую стоимость, эти платы обычно предназначены для использования профессиональными инженерами, а не любителями. То есть такие платы и программное обеспечение довольно сложны в использовании, и прежде, чем с их помощью можно будет создать что-то более или менее полезное, требуются большие усилия для их изучения.

Обзор платы Arduino

На рис. 1.1 изображена плата Arduino Uno. Давайте коротко пройдемся по различным ее элементам.

Электропитание

Взгляните на рис. 1.1. Ниже разъема USB находится стабилизатор напряжения 5 В. Он понижает любое входное напряжение (в диапазоне от 7 до 12 В) до 5 В и поддерживает его на постоянном уровне.

Стабилизатор напряжения довольно большой для элемента поверхностного монтажа. Большие размеры обусловлены необходимостью рассеивания тепла, выделяемого при стабилизации напряжения. Может пригодиться для управления внешними электронными компонентами.

Несмотря на наличие разъема, удобного для подключения батарей или другого источника постоянного тока, Arduino Uno может также питаться от порта USB, который, кроме того, используется для программирования Arduino.

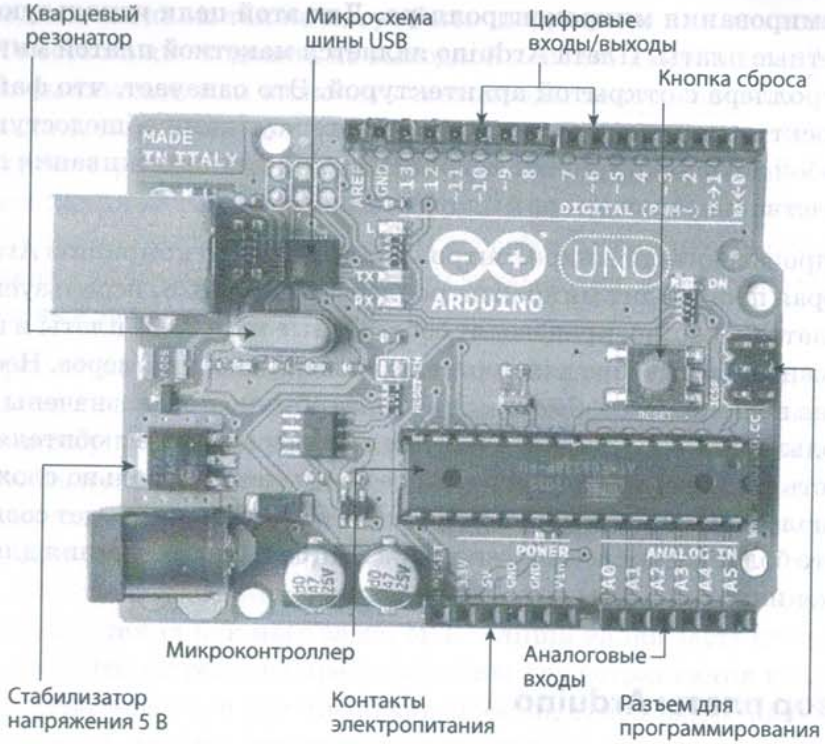


Рис. 1.1. Плата Arduino Uno

Контакты электропитания

Теперь взгляните на контакты электропитания, находящиеся внизу на рис. 1.1. Вы можете прочитать надписи рядом с контактами. Первый контакт — **Reset** (Сброс). Он служит той же цели, что и кнопка **Reset** (Сброс) на плате Arduino. По аналогии с перезагрузкой компьютера контакт **Reset** позволяет сбросить микроконтроллер в исходное состояние и заставить его выполнять программу с самого начала. Чтобы сбросить микроконтроллер с помощью этого контакта, необходимо кратковременно подать на него низкое напряжение (замкнуть на «землю»).

Остальные контакты в этой группе служат для вывода электропитания с разными уровнями напряжения (**3.3V**, **5V**, **GND** («земля») и **Vin**) согласно подписям. **GND** (ground — «земля») означает «ноль вольт». Контакты **GND** служат опорными точками, относительно которых измеряется напряжение во всех других точках на плате.

Аналоговые входы

Шесть контактов, подписанных **Analog In A0, ..., A5**, можно использовать для измерения напряжения и его анализа в скетче (так называется программа для Arduino). Отметьте, что измеряется именно напряжение, а не сила тока. Через эти контакты может течь только очень слабый ток, потому что внутренние измерительные схемы имеют очень высокое сопротивление. То есть контакты имеют высокое внутреннее сопротивление, которое позволяет протекать через них только токам небольшой силы.

Несмотря на то что эти входы обозначены как аналоговые и по умолчанию действуют как аналоговые, их можно использовать также как цифровые входы и выходы.

Цифровые входы и выходы

Теперь перейдем к верхней части рис. 1.1 и начнем движение справа налево. Итак, вверху справа находятся контакты, подписанные как **Digital 0, ..., 13**. Их можно использовать как цифровые входы или выходы. Когда эти контакты используются в качестве цифровых выходов, они действуют подобно маломощным источникам питания с напряжением 5 В, которые можно включать в скетч и выключать из него. Если включить их в скетч, на них появится напряжение 5 В, а если выключить — напряжение упадет до 0 В. Подобно контактам электропитания, их следует использовать осторожно, чтобы не превысить максимально допустимый ток. Первые два контакта, **0** и **1**, подписаны также **RX** и **TX** (от *англ.* receive — прием и transmit — передача). Эти контакты зарезервированы для организации связи и косвенно связаны с принимающим и передающим контактами разъема USB.

Цифровые контакты могут отдавать ток до 40 мА с напряжением 5 В. Этого более чем достаточно для питания светодиода, но недостаточно для непосредственного управления электромотором.

Микроконтроллер

Далее на нашем пути находится сам микроконтроллер — черная прямоугольная микросхема с 28 ножками. Она вставлена в двух-

рядный разъем, и ее легко заменить, если потребуется. На плате Arduino Uno используется микроконтроллер ATmega328. На рис. 1.2 представлена блочная диаграмма с основными характеристиками этого устройства.

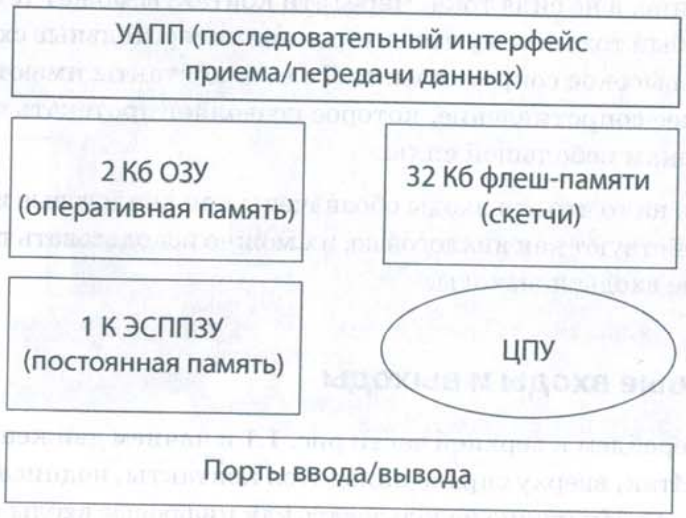


Рис. 1.2. Блочная диаграмма микроконтроллера ATmega328

Сердцем — или, точнее сказать, мозгом — устройства является центральный процессор (ЦПУ — Central Processing Unit, CPU), управляющий всем происходящим в этом устройстве. Он извлекает программные инструкции из флеш-памяти и выполняет их. Эти инструкции могут заставлять процессор читать данные из оперативной памяти (ОЗУ), изменять их и записывать обратно. Или изменять напряжение на одном или нескольких цифровых выводах.

Электрически стираемая программируемая постоянная память напоминает флеш-память своим свойством хранить информацию при выключенном питании. То есть можно выключить питание устройства, и данные в ЭСППЗУ при этом не пострадают. Разница лишь в том, что флеш-память служит для хранения программных инструкций (скетчей), а ЭСППЗУ предназначена для хранения данных, которые нельзя потерять в случае отключения питания или принудительного сброса.

Другие компоненты

Выше микроконтроллера находится маленький серебристый прямоугольник. Это кварцевый резонатор. Он генерирует 16 млн тактов в секунду, и за каждый такой такт микроконтроллер может выполнить одну операцию — сложение, вычитание или другую арифметическую операцию.

В левом верхнем углу находится кнопка **Reset** (Сброс). Нажатие на эту кнопку пошлет логический импульс на контакт **Reset** микроконтроллера и заставит его очистить оперативную память и приступить к выполнению программы с самого начала. Обратите внимание на то, что любые программы, хранящиеся на момент сброса, остаются в устройстве, потому что они хранятся в энергонезависимой флеш-памяти, то есть памяти, которая сохраняет информацию даже после выключения питания.

Возле правого края платы расположен разъем для программирования. Он обеспечивает возможность программирования микроконтроллера Arduino без использования порта USB. Так как у нас есть возможность подключиться через USB и соответствующее программное обеспечение, реализующее обмен данными с микроконтроллером, мы не будем использовать этот разъем.

В левом верхнем углу платы рядом с разъемом USB находится микросхема шины USB. Она преобразует электрические сигналы с уровнями, определяемыми стандартом USB, в сигналы с уровнями, используемыми платой Arduino.

Происхождение Arduino

Плата Arduino создавалась как учебное пособие для студентов. Затем идея получила коммерческое развитие (в 2005 году) благодаря усилиям Массимо Банци (Massimo Banzi) и Дэвида Куартилье (David Cuartielles). И с тех пор пользуется все бóльшим успехом у производителей, студентов и радиолюбителей из-за своих простоты и надежности.

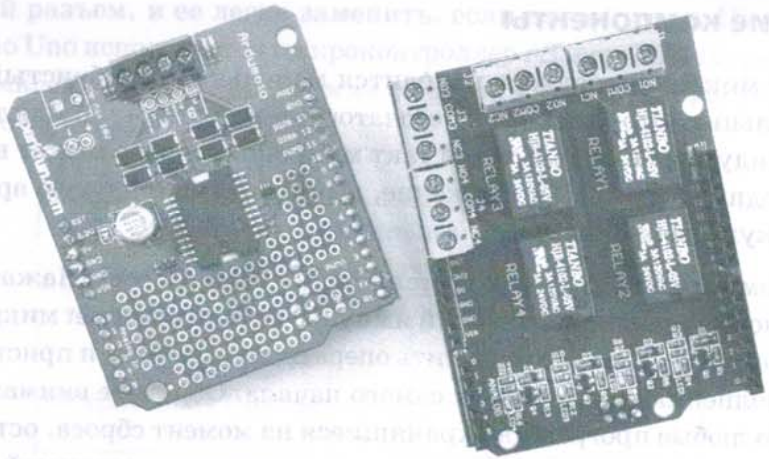


Рис. 1.3. Плата управления электромоторами и плата с блоком реле

Еще одним ключевым фактором успеха является открытость проекта Arduino — вся документация на плату распространяется бесплатно на условиях лицензии Creative Commons License. Это способствовало появлению множества недорогих альтернатив. Авторским правом защищено только название «Arduino», поэтому клонам часто дают схожие названия с окончанием «-duino», такие как Boarduino, Seeeduino (да, с тремя буквами «е») и Freeduino. В 2014 году произошел раскол между авторами проекта Arduino и основным производителем плат. В результате этого раскола за пределами США плата Arduino Uno распространяется под маркой Genuino Uno. Многие розничные продавцы продают только выпущенные официальным производителем платы, имеющие превосходное качество и привлекательную упаковку.

Еще одна причина успеха Arduino заключается в том, что производитель не ограничивается только выпуском плат с микроконтроллерами. Существует огромное число разных плат расширения, совместимых с Arduino, которые могут подключаться непосредственно к плате Arduino. Имеются платы расширения для всех мыслимых применений. Благодаря им часто удается избежать необходимости брать в руки паяльник, а просто подключать платы расширения стопкой, друг за другом. Далее перечислены наиболее популярные из таких плат:

- плата Ethernet, придающая Arduino возможности веб-сервера;
- плата управления электромоторами;
- плата расширителя USB, обеспечивающая возможность управления множеством устройств USB;
- плата с блоком реле, которые можно переключать с помощью платы Arduino.

На рис. 1.3 показана плата управления электромоторами (слева) и плата с блоком реле (справа).

Семейство Arduino

Полезно иметь некоторое представление о разнообразных платах Arduino. В этой книге в роли стандартного устройства будет использоваться плата Arduino Uno, точнее, Arduino Uno R3 (Revision 3). Это наиболее распространенная из плат Arduino, но все платы программируются на одном и том же языке и имеют практические одинаковые интерфейсы для связи с внешним миром, поэтому вы легко можете использовать другую плату.

Uno и Leonardo

Arduino Uno — лишь одна из целой серии плат Arduino. В состав этой серии входят также Diecimila (число 10 000 по-итальянски) и Duemilanove (число 2009 по-итальянски). На рис. 1.4 показана фотография модели Arduino Leonardo. Теперь вы знаете, что Arduino была изобретена в Италии.

Плата Arduino Leonardo (рис. 1.4) — еще одна популярная разновидность Arduino и в большинстве случаев может использоваться вместо Arduino Uno. Она немного дешевле модели Uno и имеет тот же набор контактов. Микросхема процессора впаяна в плату, и поэтому ее нельзя удалить (в отличие от модели Uno). Более низкая стоимость платы Leonardo обусловлена использованием процессора с собственным интерфейсом USB, благодаря чему нет необходимости в дополнительной микросхеме.

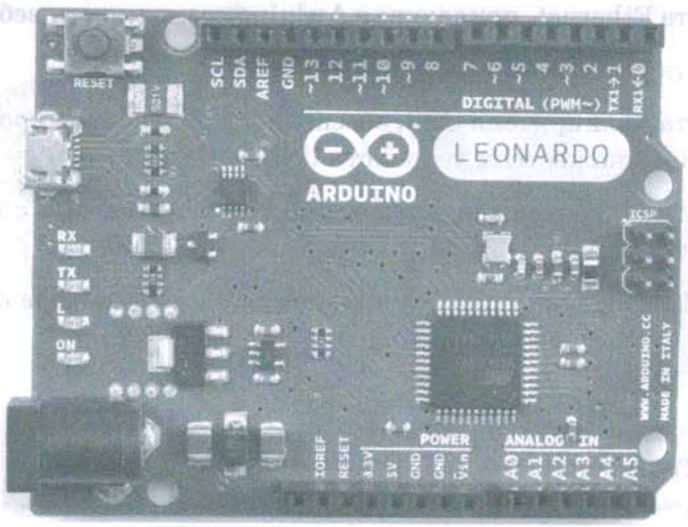


Рис. 1.4. Плата Arduino Leonardo

Mega

Плата Arduino Mega — это «мощная машина» в ряду Arduino. Она имеет множество дополнительных входных и выходных контактов, которые благоразумно расположены на одном краю платы, что обеспечивает совместимость по контактам с Arduino Uno и всеми платами расширения для Arduino. Дополнительные контакты сосредоточены вдоль одного края платы. Далее перечислены некоторые дополнительные возможности Mega:

- 54 дополнительных контакта ввода/вывода;
- 128 Кбайт флеш-памяти для хранения скетчей и постоянных данных (сравните с 32 Кбайт в Uno);
- 8 Кбайт ОЗУ и 4 Кбайт ЭСППЗУ.

Плата Arduino Due (рис. 1.5) имеет те же размеры и тот набор контактов, что и модель Mega, но в ней используется 32-разрядный процессор ARM с тактовой частотой 84 МГц. Кроме того, в отличие от большинства плат Arduino, она питается напряжением 3,3 В вместо 5 В, поэтому вместе с ней нельзя использовать некоторые платы расширения.

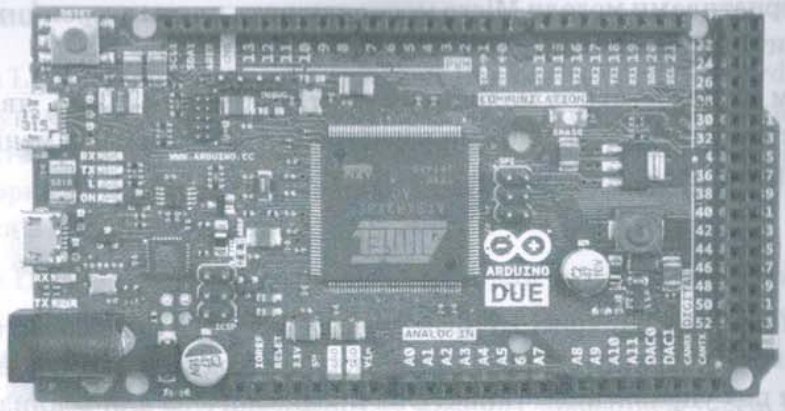


Рис. 1.5. Плата Arduino Due

Micro и другие маленькие платы Arduino

Для случаев, когда модель Uno слишком велика, был разработан целый ряд Arduino-совместимых моделей с меньшими размерами. Некоторые из них показаны на рис. 1.6.

В плате Arduino Micro используется тот же микроконтроллер, что и в модели Leonardo, но размеры самой платы значительно меньше.

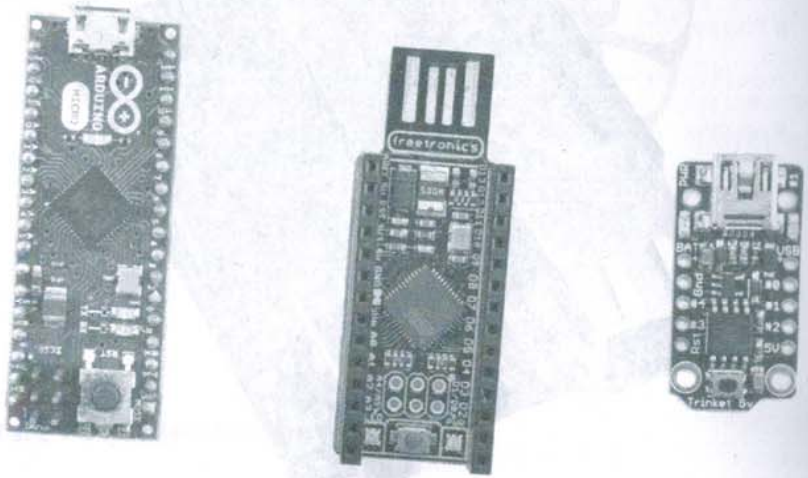


Рис. 1.6. Arduino Micro (*слева*), Freetronics LeoStick (*в центре*) и Adafruit Trinket (*справа*)

Альтернативами модели Micro могут служить платы сторонних производителей, такие как LeoStick и Adafruit Trinket.

Одним из недостатков маленьких плат, таких как Micro, является несовместимость с платами расширения для Uno из-за меньших размеров.

Yun

Плата Arduino Yun (рис. 1.7) фактически является моделью Arduino Leonardo с дополнительным миниатюрным модулем WiFi, действующим под управлением Linux. Она предназначена для реализации устройств, требующих подключения к Интернету. Взаимодействие между программным обеспечением Arduino и Linux в модели Yun реализовано с применением программного моста. Программирование платы Yun осуществляется как обычно, но ее также можно программировать из Arduino IDE без использования проводов, подключив плату к локальной сети.

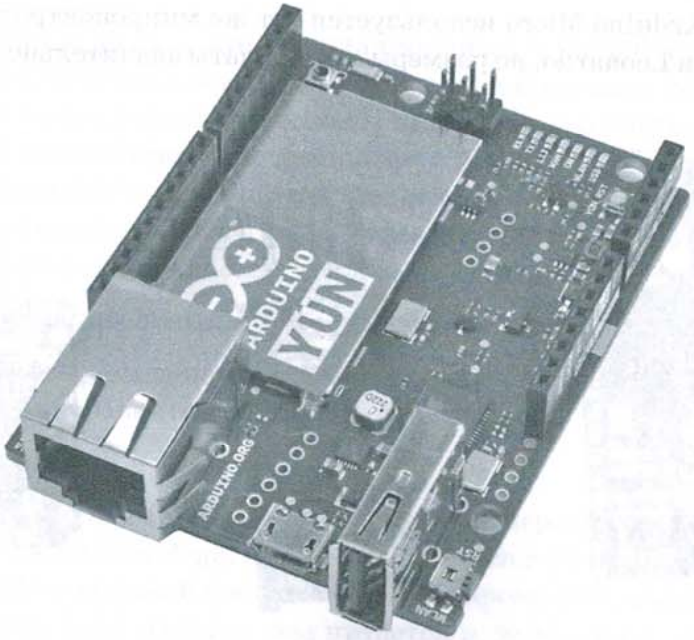


Рис. 1.7. Arduino Yun

Lilypad

Плата Lilypad (рис. 1.8) — уменьшенная в размерах плата Arduino, которую можно вшивать в элементы одежды. Такое применение вычислительных средств получило название *wearable computing* (миниатюрные устройства размещаются на теле человека или встраиваются в элементы одежды).

Плата Lilypad не имеет разъема USB — для ее программирования необходимо использовать отдельный адаптер — и выглядит весьма эффектно.

Компания Adafruit также продает модель с названием Flora, в своей основе напоминающую Lilypad.

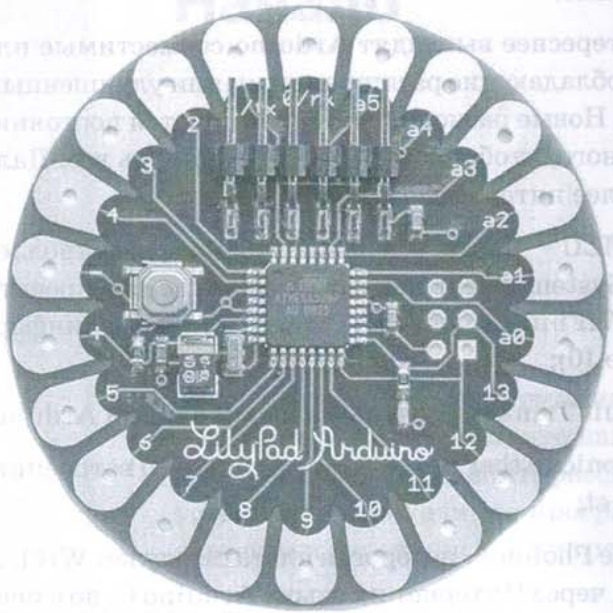


Рис. 1.8. Arduino Lilypad

Другие «официальные» платы

Выше были описаны наиболее практичные и популярные платы Arduino. Однако спектр плат Arduino постоянно обновляется, поэтому, чтобы получить самый полный и свежий их перечень, обра-

щайтесь на официальный веб-сайт Arduino по адресу: *www.arduino.cc/en/ Main/Hardware*.

Клоны и разновидности Arduino

Неофициальные платы делятся на две категории. Некоторые производители просто заимствуют архитектуру и дизайн платы Arduino и производят более дешевые аналоги. В числе таких плат можно назвать:

- Sparkfun RedBoard;
- Adafruit Metro;
- Olimexino.

Гораздо интереснее выглядят Arduino-совместимые платы второй категории, обладающие расширенными или улучшенными характеристиками. Новые разновидности появляются постоянно, и их уже слишком много, чтобы можно было упомянуть все. Далее перечислены наиболее интересные и популярные:

- Node MCU — плата, основанная на микроконтроллере ESP8266 WiFi System, очень недорогое решение для проектов, требующих WiFi-подключения (дополнительную информацию см. в главе 10);
- Adafruit Trinket — очень маленькая плата Arduino;
- Freetronics EtherTen — плата Arduino со встроенным модулем Ethernet;
- Particle Photon — недорогая плата с модулем WiFi, программируется через Интернет на языке Arduino C, но с использованием веб-версии среды разработки вместо Arduino IDE.

В заключение

Теперь, после поверхностного знакомства с аппаратурой Arduino, пришло время перейти к установке программного обеспечения Arduino.

2

Начало

Теперь, когда мы познакомились с Arduino и немного узнали о том, что нам предстоит программировать, давайте рассмотрим, как установить необходимое программное обеспечение и начать писать программный код.

Включение

Обычно на новые платы Arduino устанавливается пример программы Blink, заставляющей мигать светодиод, находящийся на плате.

Светодиод, подписанный символом L, соединен с одним из цифровых выходов. Если говорить точнее, он соединен с выходом 13. Это не означает, что выход 13 служит только для управления светодиодом, нет, его можно использовать как обычный цифровой вход или выход.

Чтобы запустить плату Arduino, достаточно подать на нее напряжение. Проще всего сделать это, подключив плату к порту USB компьютера. Для этого вам понадобится USB-кабель типа «type-A-to-type-B». Кабели данного типа обычно используются для подключения принтеров к компьютерам.

Если все в порядке, светодиод должен начать мигать. Новые платы Arduino поставляются с уже установленным скетчем Blink, чтобы покупатель мог проверить их работоспособность.

Установка программного обеспечения

Чтобы иметь возможность устанавливать новые скетчи на плату Arduino, потребуется приложить чуть больше усилий, чем потребовалось для подключения ее к порту USB, а именно установить программное обеспечение Arduino (рис. 2.1).

Полные и исчерпывающие инструкции по установке этого программного обеспечения в Windows, Linux и Mac OS можно найти на вебсайте Arduino *www.arduino.cc*.

После успешной установки программного обеспечения Arduino и в зависимости от платформы, драйверов USB у вас появится возможность выгружать программы в плату.

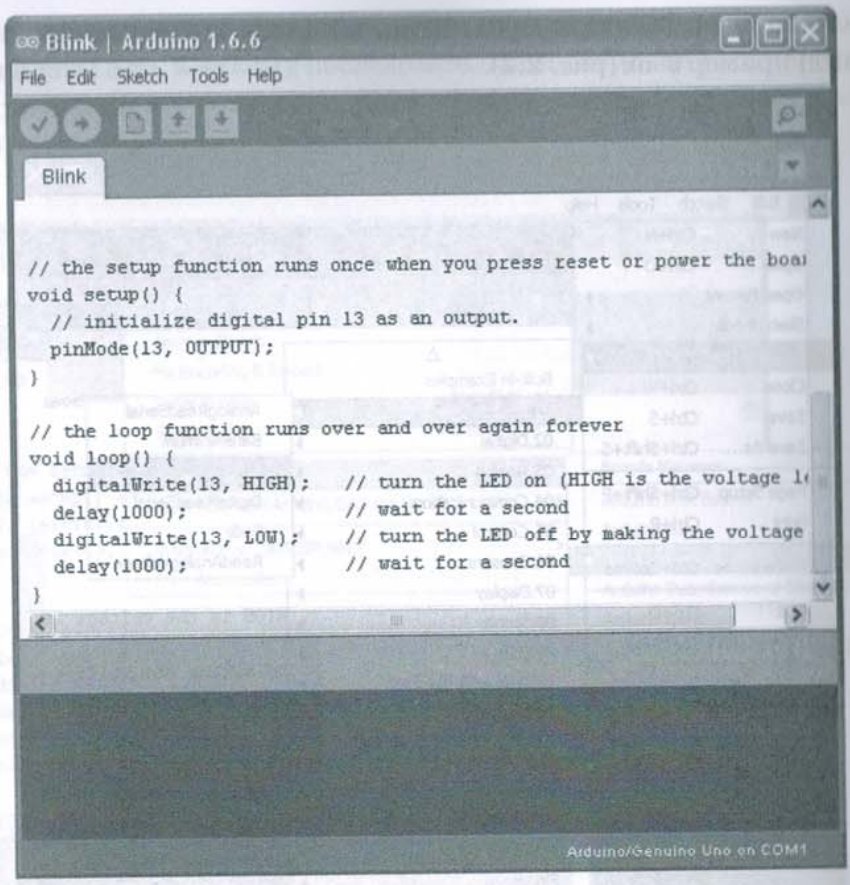


Рис. 2.1. Приложение Arduino

Выгрузка первого скетча

Мигающий светодиод можно рассматривать как Arduino-эквивалент программы *Hello world!* («Привет, мир!»), обычно используемой в других языках программирования в роли традиционной первой программы для начинающих изучать новый язык. Давайте проверим среду разработки, установив эту программу на плату Arduino, и затем немного изменим ее.

Сразу после запуска приложение Arduino открывает новый пустой скетч. К счастью, в комплекте с приложением поставляется мно-

жество интересных примеров. Итак, выберите в главном меню File (Файл) пример Blink (рис. 2.2).

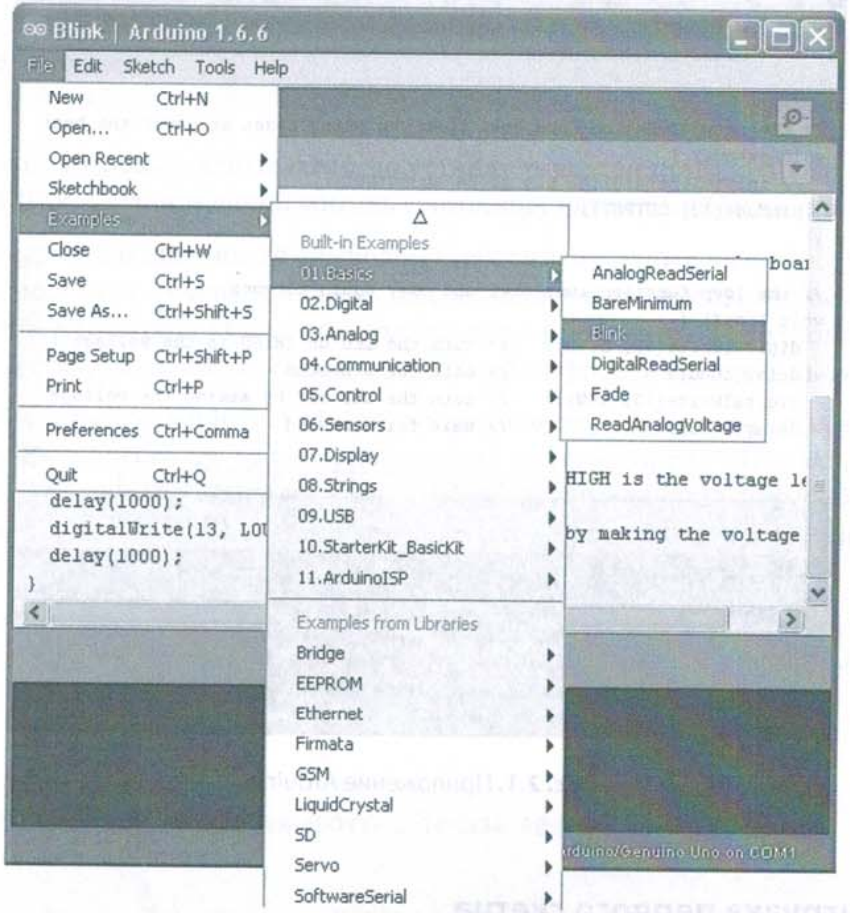


Рис. 2.2. Скетч Blink

Теперь нужно передать, или выгрузить, скетч в плату Arduino. Для этого подключите плату к компьютеру с помощью кабеля USB. При этом на плате должен загореться зеленый светодиод On. Возможно, светодиод L на плате уже мигает, потому что новые платы обычно поставляются с предустановленным скетчем Blink. Тем не менее давайте все же установим его еще раз и затем внесем в него изменения.

Прежде чем выгрузить скетч, нужно сообщить приложению Arduino тип платы и то, к какому последовательному порту она подключена. На рис. 2.3 и 2.4 показано, какие пункты в меню Tools (Инструменты) следует для этого выбрать.

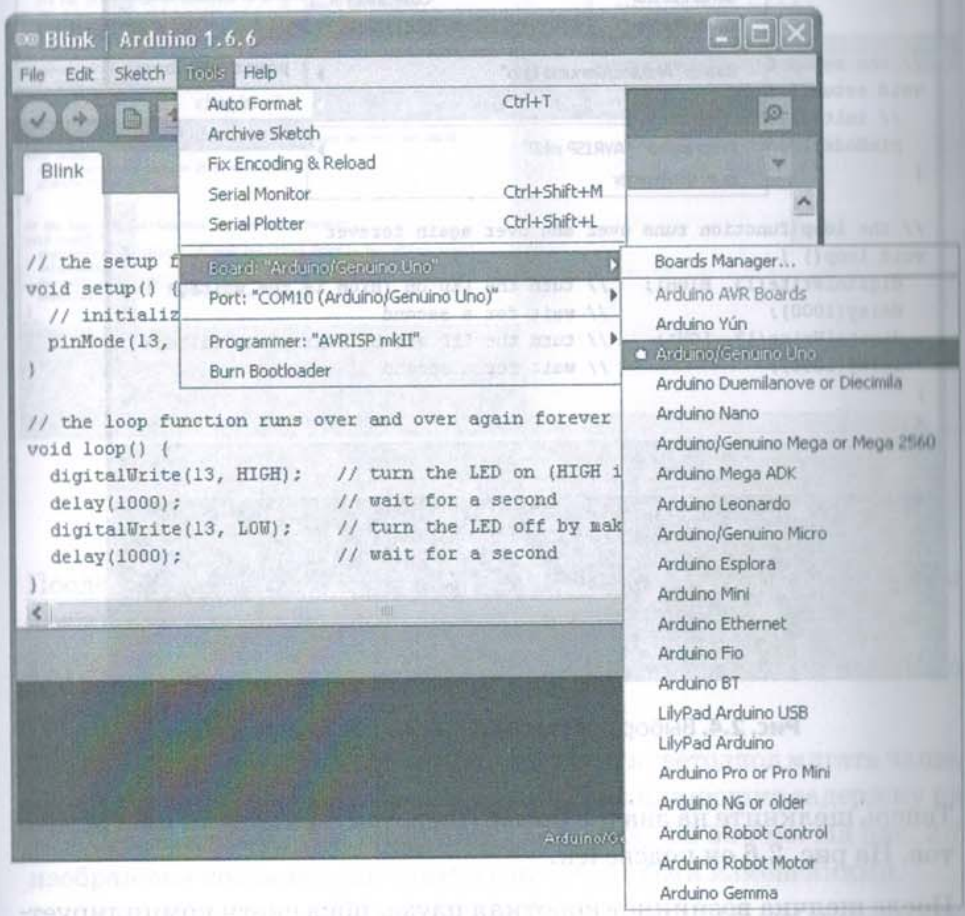


Рис. 2.3. Выбор типа платы

В Windows всегда предлагается порт с названием «COM» и следующим за ним номером. В Mac OS и Linux предлагается намного более длинный список последовательных устройств (рис. 2.5). Обычно достаточно выбрать в списке последнее устройство с именем, похожим на «/dev/cu.usbmodem621».

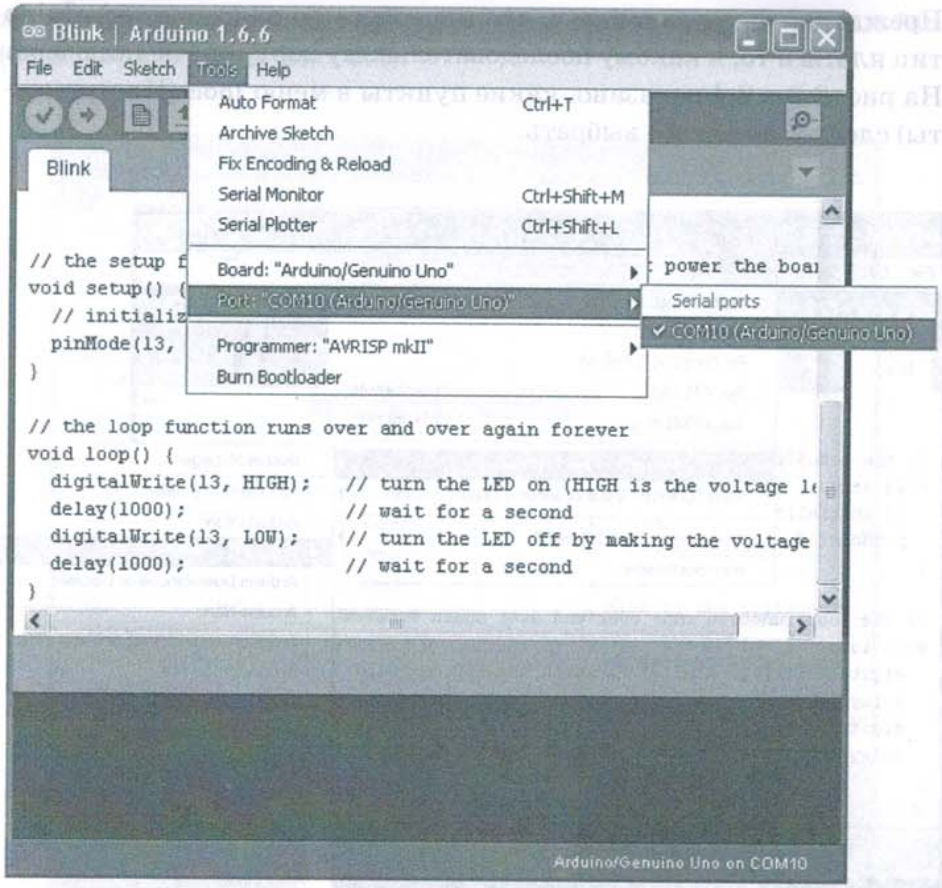


Рис. 2.4. Выбор последовательного порта (в Windows)

Теперь щелкните на значке **Upload** (Выгрузить) в панели инструментов. На рис. 2.6 он подсвечен.

После щелчка возникнет короткая пауза, пока скетч компилируется, и затем начнется передача. Во время передачи скетча светодиоды будут хаотично мигать, а по ее окончании в нижней части окна приложения Arduino должно появиться сообщение: **Done Uploading** (выгрузка завершена) с дополнительным текстом вида **Sketch uses 1033 bytes (3%) of program storage space)** (скетч занимает 1033 байт (3%) памяти для программ).

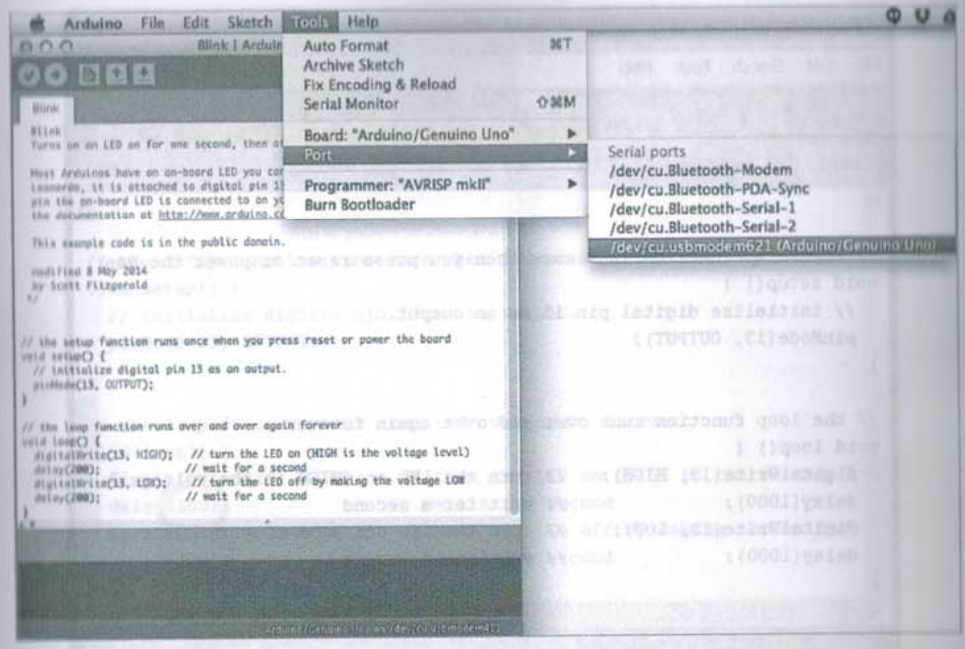


Рис. 2.5. Выбор последовательного порта в Mac

После выгрузки плата автоматически начнет выполнять скетч, и вы должны увидеть мигающий светодиод L.

Если что-то не получилось, проверьте настройки последовательного порта и типа платы.

Теперь попробуем изменить скетч и заставить светодиод мигать чаще. Для этого изменим в скетче две строки, выполняющие задержку на 1000 мс, так чтобы они выполняли задержку на 500 мс. На рис. 2.7 изображена новая версия скетча с отмеченными изменениями.

Щелкните на кнопке **Upload** (Выгрузить) еще раз. По окончании выгрузки скетча светодиод должен начать мигать в два раза чаще, чем прежде.

Поздравляем, теперь вы готовы приступить к программированию своей платы Arduino! Но прежде совершим небольшое путешествие по приложению Arduino.

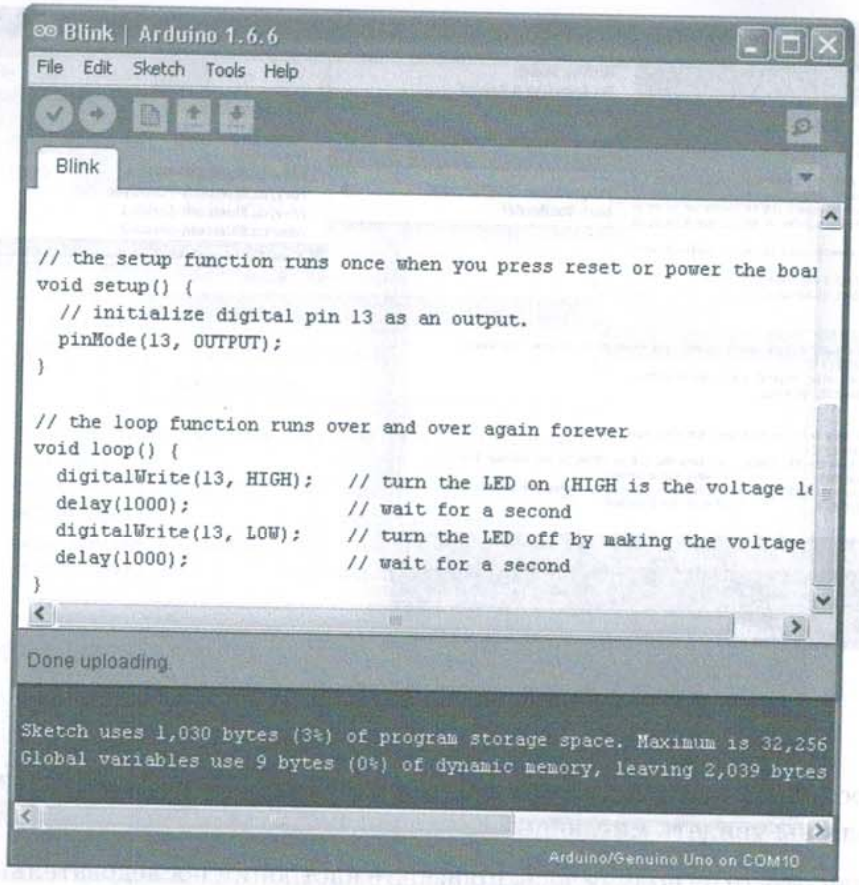


Рис. 2.6. Выгрузка скетча

Приложение Arduino

Скетчи в Arduino подобны документам в текстовом процессоре. Можно открывать их и копировать фрагменты из одного скетча в другой. В главном меню **File** (Файл) можно увидеть пункты **Open** (Открыть), **Save** (Сохранить) и **Save As** (Сохранить как). Вам редко придется пользоваться пунктом **Open** (Открыть), потому что приложение Arduino следует концепции Sketchbook (альбом), в соответствии с которой все скетчи аккуратно раскладываются по папкам. Альбом доступен из

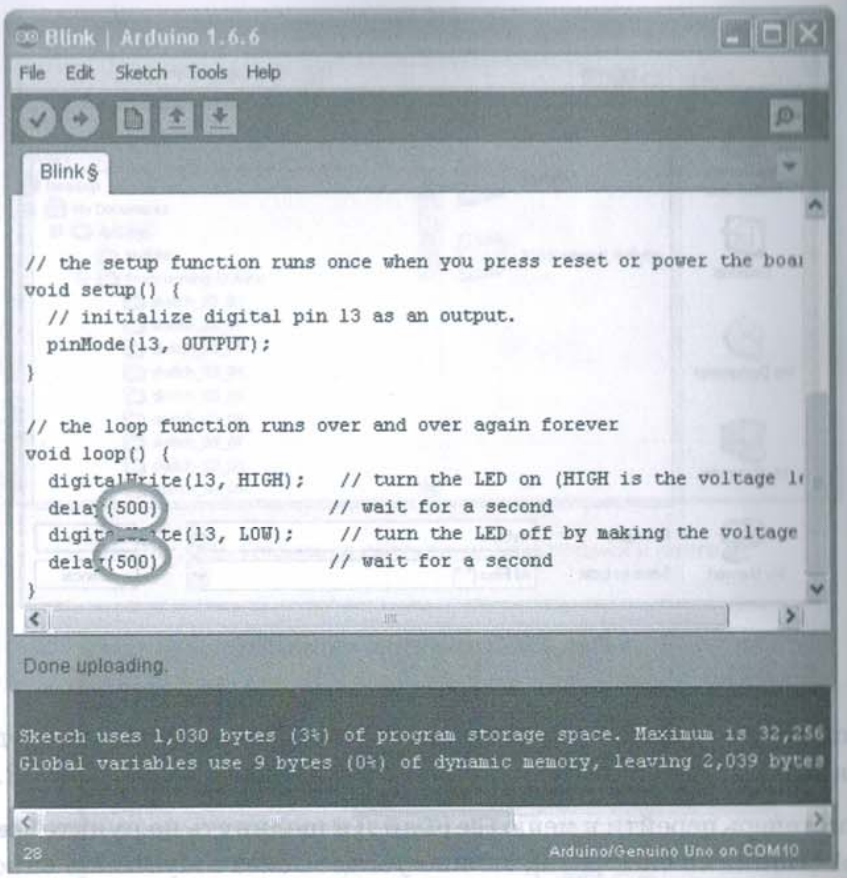


Рис. 2.7. Измененный скетч

меню **File** (Файл) в виде пункта **Sketchbook** (Папка со скетчами). Сразу после установки приложения Arduino альбом Sketchbook пуст и будет оставаться пустым, пока вы не создадите хотя бы один скетч.

Как было показано ранее, вместе с приложением Arduino поставляется коллекция примеров скетчей, которые могут очень пригодиться. Если после изменения скетча Blink попробовать сохранить его, вы получите сообщение: **Some files are marked read-only so you will need to save this sketch in a different location** (некоторые файлы помечены «только для чтения», так что повторно сохраните этот скетч в другое место).

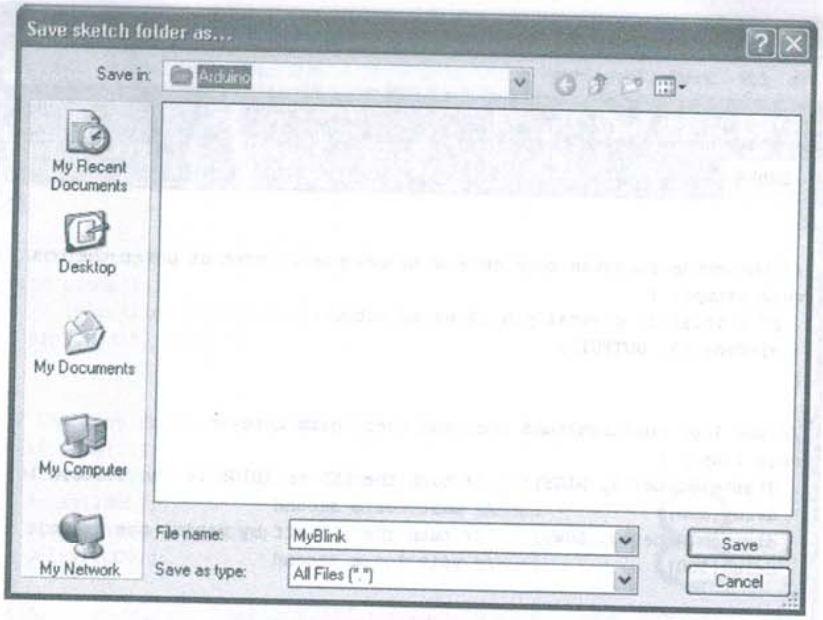


Рис. 2.8. Сохранение копии Blink

Попробуйте сделать такую попытку прямо сейчас. Оставьте предложенный путь к папке, но измените имя файла на **MyBlink** (рис. 2.8).

Если теперь перейти в меню **File** (Файл) и щелкнуть на пункте **Sketchbook** (Папка со скетчами), можно увидеть скетч MyBlink в списке. Если поискать в файловой системе компьютера, в Windows сохраненный скетч можно найти в папке **My Documents\Arduino**, а в Mac или Linux — в папке **Documents/Arduino**.

Все скетчи, используемые в этой книге, можно загрузить одним zip-архивом (**Programming_Arduino.zip**) с сайта *www.arduinobook.com*. Я предлагаю загрузить его прямо сейчас и распаковать в папку **Arduino** со скетчами. После распаковывания архива в папке **Arduino** должны появиться две папки: одна с только что сохраненным скетчем MyBlink и другая — **Programming Arduino** (рис. 2.9). Папка **Programming Arduino** содержит все скетчи, пронумерованные по главам, например **sketch_03-01** — это скетч с номером 1 из главы 3.

Эти скетчи не появятся в меню **Sketchbook**, пока вы не выйдете из приложения Arduino и не запустите его заново. Сделайте это сейчас. Тогда меню **Sketchbook** будет выглядеть, как на рис. 2.10.

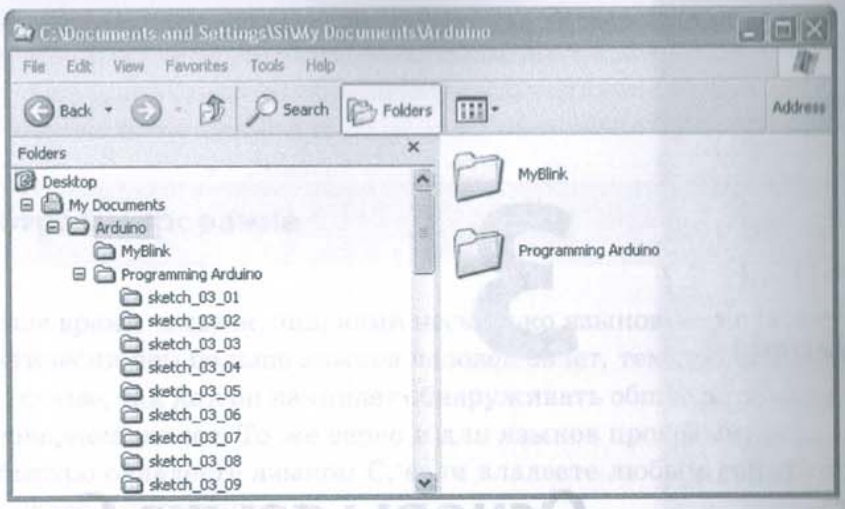


Рис. 2.9. Установка скетчей, используемых в книге

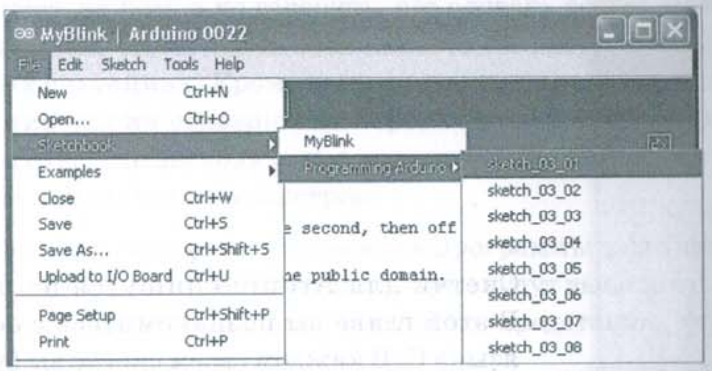


Рис. 2.10. Содержимое альбома после установки скетчей из книги

В заключение

Среда разработки настроена и готова к работе.

В следующей главе мы познакомимся с некоторыми основными принципами программирования на языке C, на котором пишутся скетчи для Arduino, и начнем писать свой код.

3

Основы языка C

Скетчи для Arduino пишутся на языке C. В этой главе вы познакомитесь с основами языка C. В каждом своем скетче вы будете использовать все, что здесь узнаете. Описываемые тут основы совершенно необходимы для успешного использования Arduino.

Программирование

В наше время человек, знающий несколько языков, — не редкость. Фактически чем больше языков человек знает, тем проще ему изучать новые, так как он начинает обнаруживать общее в грамматике и словарном запасе. То же верно и для языков программирования: вы быстро овладеете языком C, если владеете любым другим языком программирования.

Словарный запас языка программирования значительно меньше, чем разговорного, и так как язык программирования используется, чтобы писать на нем, а не говорить, его словарь всегда можно держать под рукой, чтобы иметь возможность заглянуть туда в затруднительных ситуациях. Кроме того, синтаксис и грамматика языка программирования основаны на строгих правилах, поэтому, как только вы схватите несколько простых понятий, дальнейшее изучение языка пойдет намного быстрее.

Программа, или скетч, как называются программы в Arduino, представляет собой список инструкций, которые будут выполняться в том порядке, в каком они записаны. Например, представьте, что вы написали следующее:

```
digitalWrite(13, HIGH);
delay(500);
digitalWrite(13, LOW);
```

Каждая из этих трех строк выполняет некоторую операцию. Первая устанавливает на выходе 13 высокое (HIGH) напряжение. К этому выходу подключен светодиод L на плате Arduino, то есть после выполнения этой инструкции светодиод должен загореться. Вторая просто ждет 500 мс (полсекунды), а затем третья выключает светодиод. То есть все вместе эти три строки заставят светодиод мигнуть один раз.

Кто-то в этом примере может увидеть набор знаков препинания, используемых самым странным образом, и слов, сливающихся друг с другом. Многие начинающие программисты часто впадают в уныние и говорят: «Я знаю, чего хочу, но не знаю, как это записать!» Не отчаивайтесь, я расскажу обо всем по порядку.

Для начала разберемся со знаками препинания и с особенностями формирования слов. Эти две группы являются частью того, что называют синтаксисом языка. Большинство языков программирования чрезвычайно требовательны к соблюдению синтаксиса, и одно из основных требований состоит в том, что имена программных элементов должны состоять из единственного слова. То есть они не могут содержать пробелов. Итак, **digitalWrite** — это чье-то имя. Точнее, это имя встроенной функции (позднее вы познакомитесь с функциями ближе), которая устанавливает выход платы Arduino в определенное состояние. Имена не только не могут содержать пробелов, но и чувствительны к регистру символов. То есть имя рассматриваемой функции можно написать только **digitalWrite**, но не **DigitalWrite** или **Digitalwrite**.

Функция **digitalWrite** должна знать, какой вывод требуется установить и какое состояние — **HIGH** (высокое напряжение) или **LOW** (низкое напряжение). Эти два элемента информации называют *аргументами*, которые, как говорят, *передаются* в *вызов* функции. Аргументы функции должны быть заключены в круглые скобки и отделены друг от друга запятыми.

В соответствии с соглашениями принято помещать открывающую круглую скобку сразу вслед за последней буквой в имени функции и добавлять пробел после запятой перед следующим аргументом. Но при желании внутри круглых скобок можно добавлять дополнительные пробелы.

Если функции передается единственный аргумент, ставить запятую после него не нужно. Обратите внимание на то, что каждая строка заканчивается точкой с запятой. Вместо них логичнее было бы использовать точки, потому что точка с запятой отмечает конец команды, каждая из которых немного похожа на предложение.

В следующем разделе вы больше узнаете о том, что происходит после нажатия кнопки **Upload** (выгрузить) в интегрированной среде программирования (Integrated Development Environment, IDE) Arduino. Затем вы будете готовы опробовать несколько примеров.

Особенности языка программирования

Возможно, кому-то покажется немного странным, что, добравшись до главы 3 в книге о программировании, мы никак не затрагивали особенности языка программирования. Рассматривая исходный код скетча для Arduino, можно, наверное, предположить, что он делает, но нам нужно заглянуть глубже и понять, какой путь проделывает программный код от слов на странице до реальных действий, таких как включение/выключение светодиода.

На рис. 3.1 схематически изображен весь процесс от ввода программного кода в среде Arduino IDE до запуска скетча в плате.

Когда нажимается кнопка **Upload** (Выгрузить), Arduino IDE генерирует последовательность событий, в результате которых скетч переписывается в плату Arduino и запускается. Процесс организован несколько сложнее, чем простое перемещение текста, введенного в окне редактора, в плату Arduino.

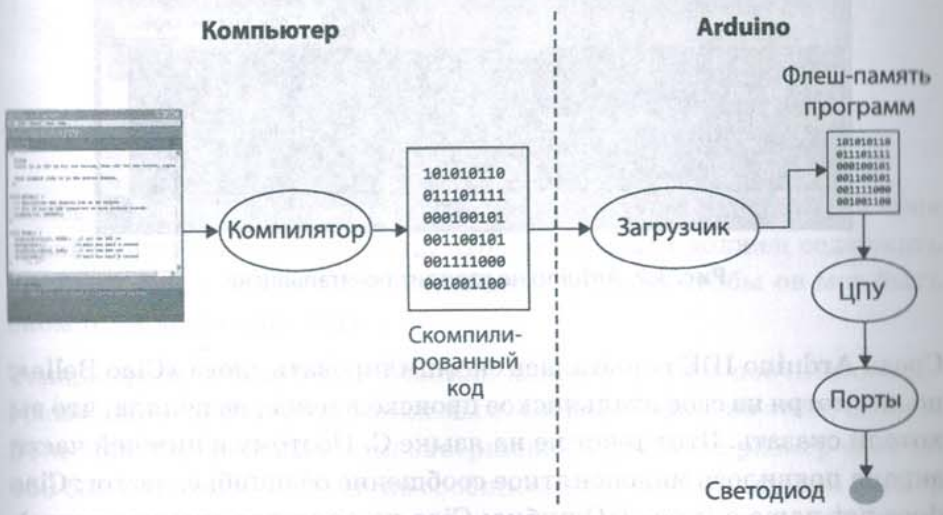


Рис. 3.1. От исходного кода до платы

Сначала выполняется преобразование текста, которое называют *компиляцией*. На этом этапе исходный код, написанный программистом, транслируется в машинный код — код на двоичном языке, который понимает Arduino. Если щелкнуть на кнопке с треугольником **Verify** (Проверить), Arduino IDE попытается скомпилировать введенный текст как программный код на языке C без передачи результата компиляции в плату Arduino. Если компиляция прошла успешно, можно утверждать, что исходный код соответствует правилам языка C. Если ввести текст `Ciao Bella` («Привет, красавица») в Arduino IDE и щелкнуть на кнопке **Verify** (Проверить), получится результат, изображенный на рис. 3.2.

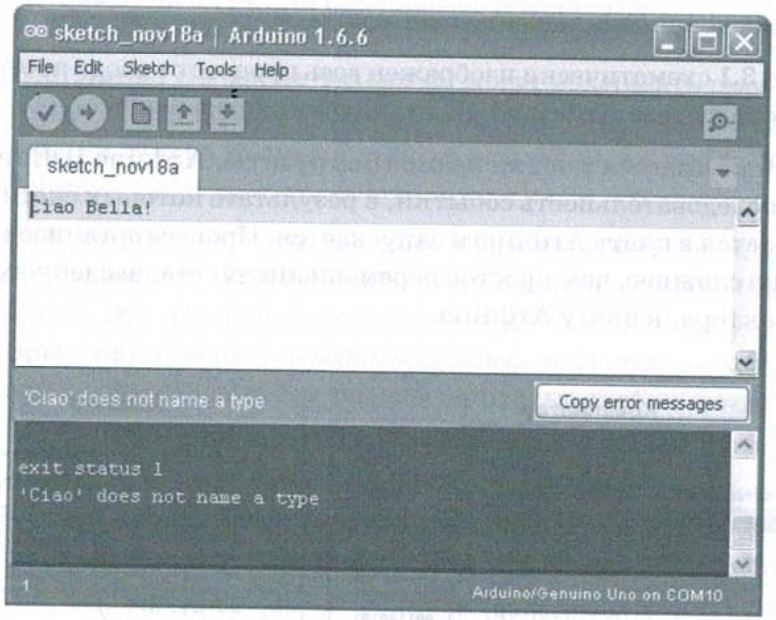


Рис. 3.2. Arduino не говорит по-итальянски

Среда Arduino IDE попыталась скомпилировать слова «Ciao Bella», но, несмотря на свое итальянское происхождение, не поняла, что вы хотели сказать. Этот текст не на языке C. Поэтому в нижней части экрана появилось малопонятное сообщение об ошибке: «error: Ciao does not name a type» («Ошибка: Ciao не является именем типа»). Фактически это означает, что в тексте обнаружена серьезная ошибка.

Давайте опробуем другой пример. Скомпилируем пустой скетч, без кода в нем (рис. 3.3). На этот раз компилятор сообщит, что в скетче отсутствует функция `setup` или `loop`. Как вы уже знаете по примеру **Blink**, который мы запускали в главе 2, каждый скетч должен содержать некоторый типовой, как его называют, код, к которому вы будете добавлять собственный. В программировании для Arduino типовой код принимает форму функций `setup` и `loop`, которые должны присутствовать в любом скетче.

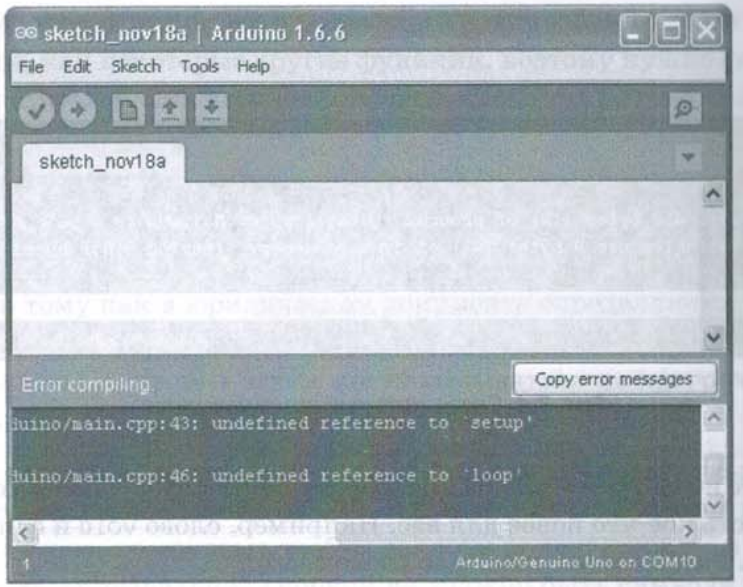


Рис. 3.3. Отсутствует функция setup или loop

Более подробно о функциях будет рассказано в дальнейшем, а пока давайте примем как данность, что любой скетч должен содержать типовой код, и просто исправим наш пример, чтобы он мог быть скомпилирован (рис. 3.4).

Теперь среда Arduino IDE оценила наши усилия и посчитала программный код вполне приемлемым. Она сообщила об этом, сказав: **Done Compiling** (компиляция завершена), и указала размер скетча: 450 байт. Arduino IDE также сообщила, что максимальный размер скетча равен 32 256 байт, то есть у нас имеется огромное пространство, в котором может разместиться гораздо более крупный скетч.

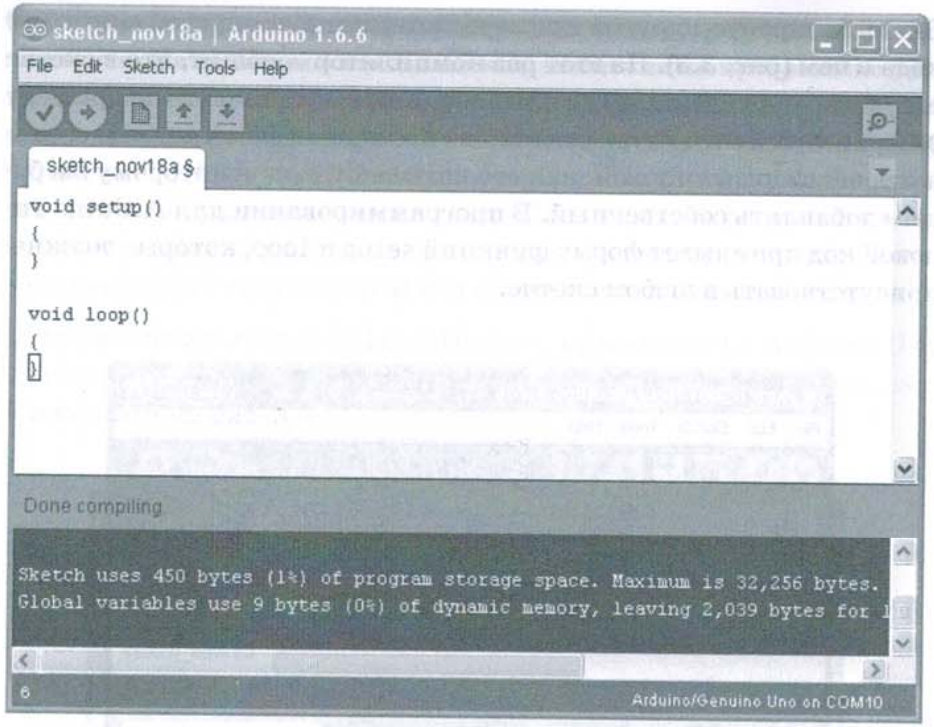


Рис. 3.4. Скетч, который компилируется

Давайте исследуем типовой код, образующий основу любого скетча. Здесь есть кое-что новое для вас. Например, слово void и фигурные скобки. Разберемся сначала со словом void.

Строка `void setup()` означает, что определяется функция с именем `setup`. В Arduino имеются предварительно определенные функции, такие как `digitalWrite` и `delay`, но вы можете или должны определить свои функции. `setup` и `loop` — это две функции, которые вы должны будете определить в каждом своем скетче.

Важно понимать, что вам не нужно вызывать функции `setup` и `loop`, как можно было бы вызвать `digitalWrite`, но вы обязаны создать эти функции, чтобы система Arduino смогла сама вызвать их. Это довольно сложная для понимания концепция, но представьте себе юридический документ.

Большинство юридических документов имеют раздел «Определения», который может выглядеть как-то так:

> Определения. Автор: лицо или группа лиц, ответственных за создание книги

Дав такое определение, юрист может дальше в документе использовать слово «автор» как сокращение для фразы «лицо или группа лиц, ответственных за создание книги» и сделать свой документ более кратким и удобочитаемым. Функции действуют подобно таким определениям. Вы определяете функцию, которую затем вы или сама система сможете использовать из любой точки в скетче.

Вернемся к слову `void`. Эти две функции (`setup` и `loop`) ничего не возвращают, как некоторые другие функции, поэтому нужно сказать, что они пустые (`void`), добавив ключевое слово `void`. Представьте себе функцию с именем `sin`, которая реализует тригонометрическую функцию с тем же именем, — такая функция должна возвращать число. Число, возвращаемое этой функцией, должно быть синусом угла, переданного функции в виде аргумента.

Подобно тому как в юридическом документе определяются термины, которые затем используются для замены групп слов, мы пишем функции на C, которые затем могут вызываться из программного кода на C.

За специальным словом `void` следуют имя функции и круглые скобки, содержащие любые параметры. В данном случае функция не имеет параметров, но круглые скобки должны присутствовать в любом случае. За закрывающей скобкой не должно быть символа «точка с запятой» — здесь мы определяем функцию, а не вызываем ее, поэтому далее должны определить, что будет происходить при вызове функции.

Описание происходящего в случае вызова функции должно быть заключено в фигурные скобки. Фигурные скобки и код между ними называют *блоком* кода, и с этим понятием мы еще встретимся далее в книге.

Обратите внимание: даже при том что вы должны определить обе функции, `setup` и `loop`, вы не обязаны добавлять какие-либо строки кода в них. Но отсутствие кода в этих функциях сделает скетч бессмысленным.

И снова Blink!

Причина, по которой в Arduino определяются две функции, setup и loop, заключается в желании отделить операции, выполняющиеся однажды, в момент запуска скетча, от операций, которые продолжают выполняться снова и снова.

Функция setup выполняется только один раз, когда происходит запуск скетча. Давайте добавим в нее код, который заставит мигнуть светодиод на плате. Добавьте строки в свой скетч, чтобы он выглядел так, как показано далее, и затем выгрузите их в плату:

```
void setup()
{
pinMode(13, OUTPUT);
  digitalWrite(13, HIGH);
}
void loop()
{
}
```

Функция setup сама вызывает две встроенные функции, pinMode и digitalWrite. Вы уже знакомы с функцией digitalWrite, но с pinMode встречаетесь впервые. Функция pinMode определяет режим работы определенного контакта — как вход или как выход. То есть включение светодиода фактически является целым процессом из двух этапов. Сначала нужно перевести контакт 13 в режим работы выхода, а потом подать на него высокий уровень напряжения (5 В).

Если запустить этот скетч, вы обнаружите, что светодиод включился и остался включенным. Это довольно скучно, поэтому давайте попробуем заставить его мигнуть — сначала включиться, а потом выключиться, но уже не в функции setup, а в функции loop.

Вызов pinMode можно оставить в функции setup, потому-то ее нужно вызвать только один раз. Скетч сохранит работоспособность, если перенести ее в функцию loop, но в этом нет необходимости, и у программистов считается хорошей привычкой делать что-то только один раз, если это что-то достаточно сделать один раз. Итак, измените скетч, чтобы он выглядел, как показано далее:


```
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(500);
  digitalWrite(13, LOW);
}
```

Запустите скетч и посмотрите, что получилось. Возможно, вы увидите не совсем то, что ожидали. Светодиод горит практически непрерывно. Хмм, почему так?

Попробуем мысленно пройтись по инструкциям в скетче.

1. Вызвать setup и перевести контакт 13 в режим работы выхода.
2. Вызвать loop и установить высокий уровень напряжения на контакте 13 (включить светодиод).
3. Выполнить задержку на полсекунды.
4. Установить низкий уровень напряжения на контакте 13 (выключить светодиод).
5. Вызвать loop снова, вернуться к шагу 2 и установить высокий уровень напряжения на контакте 13 (включить светодиод).

Проблема находится между шагами 4 и 5. Вот в чем она заключается: светодиод выключается и тут же включается снова. Между выключением и включением такой короткий промежуток времени, что создается впечатление, будто светодиод горит непрерывно.

Микроконтроллер Arduino может выполнять 16 млн операций в секунду. Это, конечно, не 16 млн команд на языке C, но все равно работа выполняется очень быстро. Итак, светодиод выключается лишь на несколько миллионных долей секунды.

Чтобы исправить проблему, нужно добавить еще одну задержку после выключения светодиода. Исправленный программный код должен выглядеть так:

```
// sketch 3-01
void setup()
{
  pinMode(13, OUTPUT);
}

void loop()
{
  digitalWrite(13, HIGH);
  delay(500);
  digitalWrite(13, LOW);
  delay(500);
}
```

Попробуйте запустить скетч — ваш светодиод должен весело замигать с частотой примерно один раз в секунду.

Возможно, вы заметили комментарий «sketch 3-01» в начале листинга. Чтобы не заставлять вас вводить программный код вручную, мы выгрузили на веб-сайт книги все скетчи, имеющие такой комментарий в начале. Вы можете загрузить их, зайдя на сайт *http://www.arduinobook.com*.

Переменные

Контакт 13 в примере Blink используется в трех местах. Если вы решите использовать другой контакт, изменения придется внести также в трех местах. Аналогично, если захотите изменить частоту мигания, которая определяется аргументом функции delay, то должны будете изменить число 500 на какое-то другое, и более чем в одном месте.

Переменные можно представлять как своего рода псевдонимы для чисел. Фактически они способны на большее, но пока будем использовать их именно в таком качестве.

Чтобы в языке C определить переменную, нужно указать ее тип. В данном случае требуется, чтобы переменные хранили целые числа, которые на языке C называются int. Иначе говоря, чтобы определить переменную с именем ledPin и значением 13, нужно записать следующий код:


```
int ledPin = 13;
```

Обратите внимание на то, что ledPin — это имя и к нему применяются те же правила, что и к именам функций. То есть имена переменных не могут содержать пробелов. В соответствии с соглашениями имена переменных принято начинать со строчной буквы, а каждое новое слово в имени — с прописной. Программисты часто называют такой стиль записи имен ухабистой (bumpy) или верблюжьей (camel) нотацией.

Давайте добавим переменные в скетч Blink, как показано далее:

```
// sketch 3-02
int ledPin = 13;
int delayPeriod = 500;

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

void loop()
{
  digitalWrite(ledPin, HIGH);
  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);
}
```

В обновленную версию скетча была введена еще одна переменная с именем delayPeriod.

Теперь везде в скетче, где потребуется сослаться на число 13, можно использовать имя переменной ledPin, а вместо числа 500 — имя переменной delayPeriod.

Если появится желание увеличить частоту мигания светодиода, достаточно будет изменить значение delayPeriod в одном месте. Попробуйте изменить его на 100 и запустить скетч.

Существует множество хитростей, которые можно применять при работе с переменными. Давайте попробуем изменить скетч так, что-

бы сразу после запуска светодиод мигал часто, а потом все реже и реже, как если бы плата Arduino «уставала» от утомительной работы. Для этого нужно организовать увеличение значения переменной delayPeriod после каждого цикла.

Измените скетч, добавив новую строку в конец функции loop, как показано далее, и запустите его. Нажмите кнопку Reset (сброс) и убедитесь, что частота мигания опять увеличилась:

```
// sketch 3-03
int ledPin = 13;
int delayPeriod = 100;

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

void loop()
{
  digitalWrite(ledPin, HIGH);
  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);
  delayPeriod = delayPeriod + 100;
```

Теперь Arduino выполняет арифметическую операцию. Каждый раз, когда вызывается функция loop, она зажигает и гасит светодиод, а потом добавляет 100 к переменной delayPeriod. Мы вскоре вернемся к арифметическим операциям, но прежде немного поэкспериментируем с платой Arduino, чтобы лучше понять, на что она способна.

Эксперименты на C

Нам нужна возможность проводить эксперименты с программным кодом на C. Один из приемов заключается в том, чтобы поместить испытуемый код на C в функцию setup, выполнить его в плате Arduino и заставить плату вернуть обратно весь вывод, произведенный программным кодом, с помощью монитора последовательного порта (Serial Monitor), как показано на рис. 3.5 и 3.6.

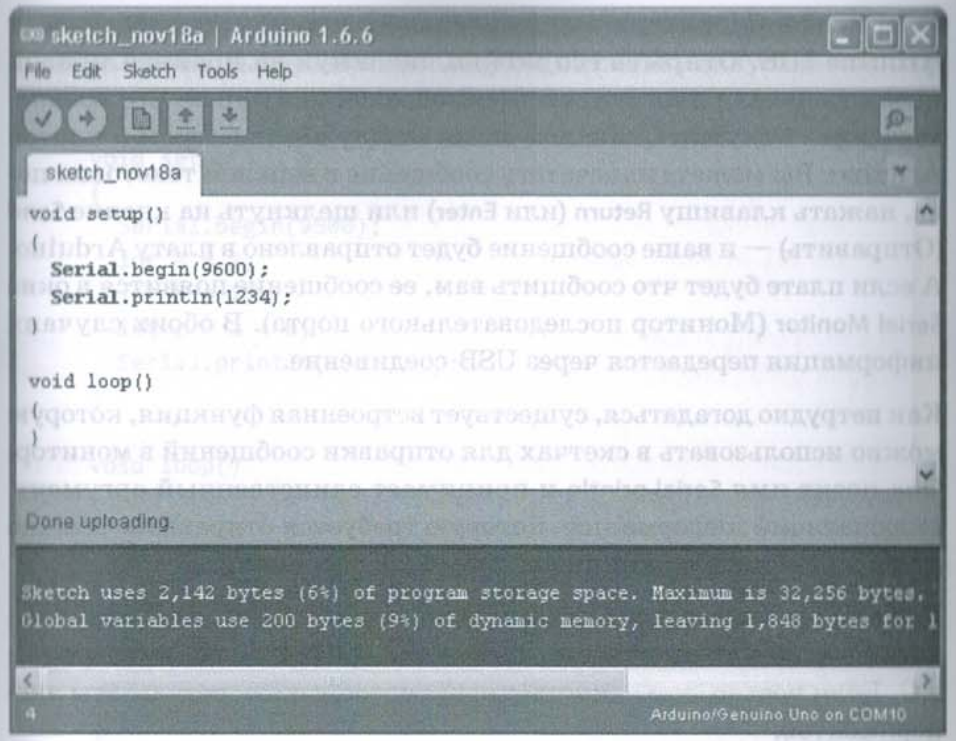


Рис. 3.5. Программный код на C в функции setup

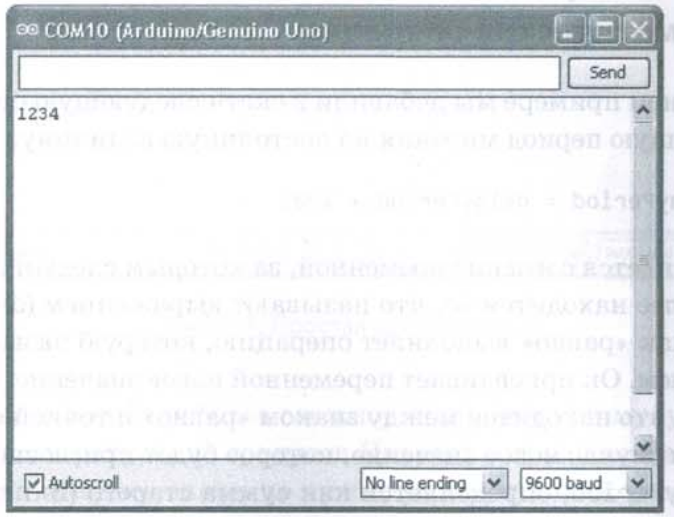


Рис. 3.6. Serial Monitor (Монитор последовательного порта)

Serial Monitor (Монитор последовательного порта) — это компонент Arduino IDE. Открыть его можно, щелкнув на крайней правой кнопке в панели инструментов (с изображением лупы). Его предназначение — служить каналом связи между компьютером и платой Arduino. Вы можете напечатать сообщение в верхнем текстовом поле, нажать клавишу **Return** (или **Enter**) или щелкнуть на кнопке **Send** (Отправить) — и ваше сообщение будет отправлено в плату Arduino. А если плате будет что сообщить вам, ее сообщение появится в окне **Serial Monitor** (Монитор последовательного порта). В обоих случаях информация передается через USB-соединение.

Как нетрудно догадаться, существует встроенная функция, которую можно использовать в скетчах для отправки сообщений в монитор. Она носит имя **Serial.println** и принимает единственный аргумент, включающий информацию, которую требуется отправить. Обычно для хранения информации используется переменная.

Мы воспользуемся этим механизмом, чтобы узнать, что можно делать с переменными и как выполняются арифметические операции в C. Если честно, это единственный способ узнать результаты экспериментов.

Числовые переменные и арифметические операции

В последнем примере мы добавили в скетч следующую строку, увеличивающую период мигания на постоянную величину:

```
delayPeriod = delayPeriod + 100;
```

Она начинается с имени переменной, за которым следует знак «равно», а далее находится то, что называют выражением (`delayPeriod + 100`). Знак «равно» выполняет операцию, которую называют присваиванием. Он присваивает переменной новое значение, определяемое тем, что находится между знаком «равно» и точкой с запятой. В данном случае новое значение, которое будет присвоено переменной `delayPeriod`, определяется как сумма старого (или текущего) значения `delayPeriod` и числа 100.

Давайте задействуем новый механизм, чтобы посмотреть, что делает плата Arduino, выгрузив в нее следующий скетч и запустив монитор:

```
// sketch 3-04
void setup()
{
  Serial.begin(9600);
  int a = 2;
  int b = 2;
  int c = a + b;
  Serial.println(c);
}

void loop()
{}
```

На рис. 3.7 показано, что должно появиться в окне монитора после запуска этого кода.

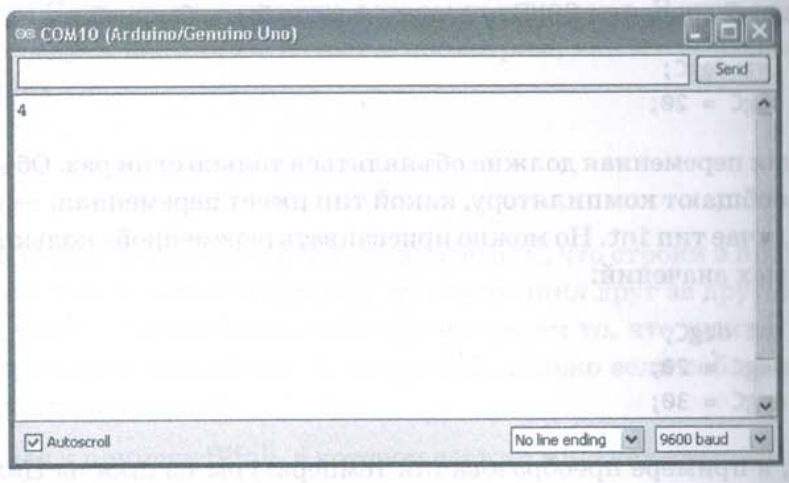


Рис. 3.7. Простая арифметика

Попробуем что-нибудь более сложное, например формулу преобразования температуры из градусов Цельсия в градусы Фаренгейта, по которой исходное значение нужно умножить на 9, разделить на 5 и прибавить 32. В скетче это будет выглядеть так:

```
// sketch 3-05
void setup()
{
  Serial.begin(9600);
  int degC = 20;
  int degF;
  degF = degC * 9 / 5 + 32;
  Serial.println(degF);
}

void loop()
{}
```

Здесь есть две особенности, на которые следует обратить внимание. Посмотрите на строку

```
int degC = 20;
```

Когда записывается такая строка, она фактически подразумевает две вещи: объявление переменной типа `int` с именем `degC` и присваивание ей значения 20. Эту строку можно было бы разбить на две:

```
int degC;
degC = 20;
```

Каждая переменная должна объявляться только один раз. Объявления сообщают компилятору, какой тип имеет переменная, — в данном случае тип `int`. Но можно присваивать переменной сколько угодно новых значений:

```
int degC;
degC = 20;
degC = 30;
```

Итак, в примере преобразования температуры из шкалы Цельсия в шкалу Фаренгейта мы объявили переменную `degC` и дали ей начальное значение 20, но в определении переменной `degF` не указали начального значения. Она получает значение в следующей строке, где выполняются вычисления по формуле, перед передачей на монитор, где его можно увидеть.

Взгляните на выражение. Здесь вы видите звездочку (*), выполняющую умножение, и слеш (/), выполняющий деление. Арифметические операции +, -, * и / выполняются с учетом правил предшествования. То есть сначала выполняются умножение и деление как имеющие более высокий приоритет, а потом сложение и вычитание. Это соответствует привычным правилам арифметики. Но иногда для большей ясности все же следует использовать круглые скобки. Например, ту же формулу можно записать так:


```
degF = ((degC * 9) / 5) + 32;
```

Выражения могут быть сколь угодно длинными и сложными и помимо обычных арифметических содержать другие, реже используемые операторы, а также разнообразные математические функции. Со всем этим мы познакомимся позже.

Команды

В языке C имеется множество встроенных команд. В этом разделе мы исследуем некоторые из них и посмотрим, как их использовать в скетчах.

if

До сих пор в своих скетчах мы предполагали, что строки в программе будут выполняться в порядке их следования друг за другом, без исключений. Но как быть, если это не совсем то, что нужно? Что, если требуется выполнить часть скетча, только если соблюдается определенное условие?

Вернемся к примеру Blink, в котором частота мигания светодиода постепенно уменьшается. В процессе выполнения этого скетча светодиод будет мигать все реже и реже, и в конце концов периоды, когда светодиод светится или погашен, могут достигнуть нескольких часов. Давайте посмотрим, как изменить эту ситуацию, чтобы по достижении некоторого предела происходил возврат к первоначальной частоте мигания.

Для этого следует воспользоваться командой `if`, как показано в следующем скетче. Попробуйте запустить его:

```
// sketch 3-06
int ledPin = 13;
int delayPeriod = 100;

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

void loop()
{
  digitalWrite(ledPin, HIGH);
  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);
  delayPeriod = delayPeriod + 100;
  if (delayPeriod > 3000)
  {
    delayPeriod = 100;
  }
}
```

Команда if немного похожа на определение функции, но это кажущееся сходство. Слово в круглых скобках не является аргументом — это то, что называют *условием*. То есть в данном случае, проверяется условие, превысило ли значение переменной delayPeriod число 3000. Если условие истинно, выполняются команды, стоящие в фигурных скобках. В данном случае переменной delayPeriod присваивается начальное значение 100.

Если условие ложно, Arduino просто перешагнет через фигурные скобки и продолжит выполнять код со следующей за ними строки. В данном случае после фигурных скобок нет никаких команд, поэтому Arduino начнет повторное выполнение функции loop с начала.

Давайте мысленно пройдемся по последовательности событий, происходящих в скетче, чтобы лучше понять происходящее. Итак, выполняются следующие действия.

1. Arduino выполнит setup и переведет контакт 13 в режим работы выхода.
2. Arduino начнет выполнение loop.

3. Светодиод включится.
4. Выполнится задержка.
5. Светодиод выключится.
6. Выполнится задержка.
7. К переменной delayPeriod прибавится число 100.
8. Если период задержки превысит 3000, он будет установлен равным 100.
9. Переход к шагу 3.

Для проверки условия мы использовали символ >, который означает «больше, чем». Это один из операторов, которые называют операторами сравнения. Эти операторы перечислены в следующей таблице:

Оператор	Значение	Пример	Результат
<	Меньше чем	9 < 10 10 < 10	Истина Ложь
>	Больше чем	10 > 10 10 > 9	Ложь Истина
<=	Меньше или равно	9 <= 10 10 <= 10	Истина Истина
>=	Больше или равно	10 >= 10 10 >= 9	Истина Истина
==	Равно	9 == 9	Истина
!=	Не равно	9 != 9	Ложь

Определить равенство двух чисел можно с помощью команды ==. Эти два знака «равно», следующих друг за другом, легко спутать с одиночным символом =, который используется для присваивания значений переменным.

Существует еще одна форма команды if, позволяющая выполнить одни действия, если условие истинно, и другие, если ложно. Мы будем использовать эту форму в некоторых практических примерах далее в книге.

for

Помимо возможности выполнять разные команды при разных условиях часто бывает нужно выполнить некоторую последовательность команд много раз. Вы уже знаете один из способов сделать это — использовать функцию loop. Как только будет выполнена последняя команда в функции loop, она тут же начнет выполняться сначала. Однако иногда требуется более полный контроль над выполнением.

Например, представьте, что нам нужно написать скетч, который должен мигнуть светодиодом 20 раз, выдержать паузу в 3 секунды и затем все начать сначала. Это можно сделать, просто повторив один и тот же код в функции loop, как показано далее:

```
// sketch 3-07
int ledPin = 13;
int delayPeriod = 100;

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

void loop()
{
  digitalWrite(ledPin, HIGH);
  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);

  digitalWrite(ledPin, HIGH);
  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);

  digitalWrite(ledPin, HIGH);
  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);
  // повторить предыдущие 4 строки еще 17 раз

  delay(3000);
}
```


Для этого потребуется ввести вручную много кода. Однако есть более удачное решение. Давайте сначала посмотрим, как можно использовать цикл for, а затем познакомимся с другим вариантом решения, основанным на применении счетчика и инструкции if.

Скетч, решающий поставленную задачу с помощью цикла for, получился намного короче и проще предыдущего примера:

```
// sketch 3-08
int ledPin = 13;
int delayPeriod = 100;

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

void loop()
{
  for (int i = 0; i < 20; i ++)
  {
    digitalWrite(ledPin, HIGH);
    delay(delayPeriod);
    digitalWrite(ledPin, LOW);
    delay(delayPeriod);
  }
  delay(3000);
}
```

Цикл for немного похож на функцию, принимающую три аргумента, но в данном случае аргументы разделены точкой с запятой вместо обычных запятых. Это просто причудливая особенность языка C. Но не бойтесь ошибиться — компилятор немедленно сообщит вам, если ему что-то не понравится.

Сразу после открывающей круглой скобки находится объявление переменной. Это объявление указывает, какая переменная будет использоваться в качестве счетчика, и присваивает ей начальное значение — в данном случае 0.

Второй элемент — условное выражение, которое должно оставаться истинным, чтобы цикл for продолжал выполняться. В данном слу-

чае цикл будет продолжать выполняться, если i меньше 20, но как только значение переменной i станет равно 20 или превысит его, выполнение цикла завершится.

Последний элемент определяет, что нужно сделать после выполнения всех команд в теле цикла. В данном случае выполняется увеличение переменной i на 1, чтобы после 20 циклов она перестала быть меньше 20 и цикл завершился.

Попробуйте ввести этот код и запустить его. Единственный способ освоить синтаксис языка и его жуткую пунктуацию — это вводить код вручную и внимательно читать сообщения компилятора, описывающие ваши ошибки. Рано или поздно все это начнет обретать для вас определенный смысл.

Один из потенциальных недостатков такого решения состоит в том, что на выполнение цикла может потребоваться довольно много времени. Для данного скетча это не является проблемой, потому что единственная его задача — мигать светодиодом. Но часто функция loop в скетче проверяет также нажатия клавиш или наличие данных, полученных по последовательному каналу. Если процессор будет занят выполнением команд в цикле for, он не сможет сделать ничего другого. Как правило, следует стараться реализовать функцию loop так, чтобы она выполнялась как можно быстрее и ее можно было запускать чаще.

Следующий скетч показывает, как добиться этого:

```
// sketch 3-09
int ledPin = 13;
int delayPeriod = 100;
int count = 0;

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

void loop()
{
  digitalWrite(ledPin, HIGH);
  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);
  count ++;
  if (count == 20)
  {
    count = 0;
    delay(3000);
  }
}
```


Обратите внимание на строку

```
count ++;
```

Это сокращенная форма записи следующей строки:

```
count = count + 1;
```

Теперь каждый вызов функции loop станет выполняться чуть дольше 200 мс, кроме каждого 20-го вызова, когда будет выполнено все то же самое плюс задержка на 3 секунды между сериями по 20 вспышек. В действительности в некоторых случаях даже такая скорость оказывается слишком медленной, а пуристы могли бы сказать, что от задержек вообще следует отказаться. Лучшее решение часто зависит от приложения.

while

Другой способ организации циклов в языке C — использовать команду while вместо for. Решить ту же задачу, что и в примере с циклом for, можно с помощью цикла while:

```
int i = 0;
while (i < 20)
{
  digitalWrite(ledPin, HIGH);
  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);
  i ++;
}
```

Выражение в круглых скобках после слова while должно быть истинным, чтобы цикл продолжал выполняться. Когда оно перестанет быть истинным, скетч продолжит выполнение с команды, стоящей за закрывающей фигурной скобкой.

Константы

Для определения постоянных значений, таких как номера контактов на плате, которые не меняются в ходе выполнения скетча, используйте ключевое слово const, которое сообщит компилятору, что переменная хранит постоянное, не изменяющееся значение.

Например, номер контакта светодиода можно было бы определить, как показано далее:

```
const int ledPin = 13
```

Ключевое слово const перед именем переменной не повлияет на работу скетча, но сэкономит немного памяти, что может оказаться важным преимуществом для больших скетчей. В любом случае считается хорошей привычкой добавлять слово const перед переменными, которые не изменяются.

В заключение

В этой главе вы начали знакомиться с языком C. Вы узнали несколько интересных способов реализации мигания светодиода и отправки результатов из платы Arduino через USB-соединение с помощью функции Serial.println. Вы также увидели, как использовать команды if и for для управления порядком выполнения других команд, и узнали немного об арифметических операциях в Arduino.

В следующей главе вы поближе познакомитесь с функциями. Там же рассказывается о других типах переменных, отличных от типа int, использовавшегося в этой главе.

4

Функции

В этой главе основное внимание будет уде
лено созданию собственных функций
а не встроенным функциям, таким ка
digitalWrite и delay, которые уже написа
ны другими программистами.

Причина того, почему нужно уметь писат
свои функции, проста. Сначала вы будете
писать простые скетчи, потом все сложне
и сложнее, и функции setup и loop буду
все больше расти в размерах, пока не ста
нут слишком большими и сложными, что
бы можно было понять, как они работают.

При разработке программного обеспечени
самая большая проблема — управлени

сложностью программного кода. Опытные программисты пишут программный код, который легко читается, прост для понимания и почти не требует пояснений.

Функции являются ключевым инструментом создания простых для понимания скетчей, которые можно изменять без лишних сложностей и риска превратить код в малопонятную мешанину команд.

Что такое функция?

Функция — это маленькая программа внутри программы. В функцию можно завернуть небольшой фрагмент кода, выполняющий нужные операции. Определяемые вами функции можно вызывать из любого места в скетче, и они могут содержать собственные переменные и списки команд. Завершив работу, функция возвращает управление в ту точку, откуда вызывалась.

К примеру, код, который включает и выключает светодиод, является отличным кандидатом на оформление в виде функции. Давайте изменим скетч, реализующий мигание светодиода сериями по 20 раз, и используем в нем созданную нами функцию с именем `flash`:

```
// sketch 4-01
const int ledPin = 13;
const int delayPeriod = 250;

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

void loop()
{
  for (int i = 0; i < 20; i ++)
  {
    flash();
  }
  delay(3000);
}

void flash()
{
  digitalWrite(ledPin, HIGH);
  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);
}
```


Все, что мы здесь сделали, — перенесли четыре строки кода, управляющие светодиодом, из тела цикла `for` в отдельную функцию с именем `flash`. Теперь вы в любой момент можете заставить светодиод мигнуть, просто вызвав новую функцию, записанную как `flash()`. Обратите внимание на пустые круглые скобки после имени функции. Они указывают, что функция не имеет параметров. Длительность задержки, как и прежде, определяется значением переменной `delayPeriod`.

Параметры

Деля скетч на функции, часто имеет смысл подумать о том, какие услуги они должны оказывать. В случае с функцией `flash` все выглядит более чем очевидно. Но давайте дадим ей параметры, которые сообщат, сколько раз должен мигнуть светодиод и насколько продолжительными должны быть задержки. Прочитайте следующий код, а потом я подробнее объясню, как работают параметры:

```
// sketch 4-02
const int ledPin = 13;
const int delayPeriod = 250;

void setup()
{
  pinMode(ledPin, OUTPUT);
}
```

```
void loop()
{
  flash(20, delayPeriod);
  delay(3000);
}

void flash(int numFlashes, int d)
{
  for (int i = 0; i < numFlashes; i ++)
  {
    digitalWrite(ledPin, HIGH);
    delay(d);
    digitalWrite(ledPin, LOW);
    delay(d);
  }
}
```

Итак, в функции loop остались всего две строки. Основную работу мы переместили в функцию flash. Отметьте, что теперь при вызове функции flash ей передаются два аргумента в круглых скобках.

В определении функции внизу скетча мы объявили типы переменных в параметрах. В данном случае оба параметра имеют тип int. Фактически мы определили новые переменные. Но эти переменные (numFlashes и d) доступны только внутри функции flash.

Это хорошая функция, потому что она включает все, что необходимо, чтобы светодиод мог мигнуть. Единственная информация, которую она получает извне, — номер контакта, к которому подключен светодиод. При желании можно было бы определить еще один параметр для передачи номера контакта, о чем стоит подумать, если к плате Arduino будут подключены несколько светодиодов.

Глобальные, локальные и статические переменные

Как отмечалось ранее, параметры функции могут использоваться только внутри этой функции. То есть вы получите сообщение об ошибке, если напишете такой код:


```
void indicate(int x)
{
  flash(x, 10);
}
x = 15;
```

Но представьте себе такой код:

```
int x = 15;

void indicate(int x)
{
  flash(x, 10);
}
```

Компилятор не обнаружит ошибки в этом коде. Однако вам придется быть очень внимательными, потому что здесь фактически используются две переменные с именем x и каждая имеет свое значение. Первая, которая объявлена в первой строке, называется *глобальной переменной*. Глобальной она называется потому, что доступна из любой точки программы, включая функции.

Однако из-за того, что в функции indicate используется переменная с тем же именем x, объявленная как параметр, в ней нельзя получить доступ к глобальной переменной x просто потому, что любая ссылка на имя x внутри функции будет интерпретироваться как ссылка на «локальную» версию x. В таких случаях говорят, что параметр x *затеняет* глобальную переменную с тем же именем. Иногда при отладке проекта это может привести к путанице.

Кроме параметров в функции можно определять переменные, которые не являются параметрами и используются только внутри функции. Такие переменные называют локальными. Например:

```
void indicate(int x)
{
  int timesToFlash = x * 2;
  flash(timesToFlash, 10);
}
```

Локальная переменная timesToFlash существует, только пока выполняется функция. Как только функция выполнит свою последнюю ко-

манду, эта переменная исчезнет. Это означает, что локальные переменные недоступны за пределами функций, в которых они определены.

Например, следующий код вызовет ошибку:

```
void indicate(int x)
{
  int timesToFlash = x * 2;
  flash(timesToFlash, 10);
}
timesToFlash = 15;
```

Опытные программисты обычно избегают глобальных переменных, так как глобальные переменные противоречат принципу инкапсуляции. Суть *инкапсуляции* состоит в том, чтобы упаковать вместе все, что имеет отношение к определенной особенности. Следовательно, функции служат на благо инкапсуляции. Проблема глобальных переменных в том, что они обычно определяются в начале скетча и доступны из любой его точки. Иногда на то есть вполне обоснованные причины. Но иногда их использование лишь результат лени, например, когда программисту лень оформить передачу параметров. В примерах, демонстрировавшихся до сих пор, нам было удобно использовать глобальную переменную ledPin. Ее легко найти в начале скетча и изменить.

Еще одна особенность локальных переменных в том, что они инициализируются каждый раз, когда вызывается функция. Особенно это заметно (и неудобно) в функции loop. Давайте попробуем заменить глобальную переменную на локальную в одном из примеров, приведенных в предыдущей главе:

```
// sketch 4-03
const int ledPin = 13;
const int delayPeriod = 250;

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

void loop()
{
  int count = 0;
```



```
  digitalWrite(ledPin, HIGH);
  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);
  count ++;
  if (count == 20)
  {
    count = 0;
    delay(3000);
  }
}
```

Скетч 4-03 основан на скетче 3-09, но пытается использовать локальную переменную вместо глобальной для подсчета числа миганий.

Этот скетч не будет работать, как ожидается, потому что при каждом вызове loop переменная count снова будет инициализирована, ее значение никогда не достигнет 20 и светодиод будет мигать без остановки. Основная причина того, почему переменная count была объявлена глобальной, состоит в том, что ее значение никогда не должно сбрасываться. Но переменная count используется только в функции loop, поэтому вполне логично было бы поместить ее туда.

К счастью, в языке C имеется механизм, позволяющий решить эту проблему. Это ключевое слово static. Если внутрь функции поместить ключевое слово static перед объявлением переменной, такая переменная будет инициализирована только при первом вызове функции. Отлично! Это именно то, что нужно в данной ситуации. Мы можем оставить переменную в функции, в которой она используется, не опасаясь, что ее значение будет сбрасываться в 0 при каждом следующем вызове. Это решение демонстрирует скетч 4-04:

```
// sketch 4-04
const int ledPin = 13;
const int delayPeriod = 250;

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

void loop()
```

```
{
  static int count = 0; digitalWrite(ledPin, HIGH);

  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);
  count ++;
  if (count == 20)
  {
    count = 0;
    delay(3000);
  }
}
```

Возвращаемые значения

Информатика как академическая дисциплина родилась на стыке математики и инженерного искусства. Эта наследственность находит отражение во многих аспектах, связанных с программированием. Слово *function* само по себе является математическим термином. В математике входное значение функции (аргумент) полностью определяет ее результат. До сих пор мы писали функции, которые принимают входные аргументы, но ничего не возвращают. Все наши функции были пустыми (void). Если функция должна возвращать какое-то значение, нужно указать его тип.

Рассмотрим функцию, получающую температуру в градусах Цельсия и возвращающую ее эквивалент в градусах Фаренгейта:

```
int centToFaren(int c)
{
  int f = c * 9 / 5 + 32;
  return f;
}
```

Теперь определение функции начинается со слова `int`, а не `void`, что показывает, что вызов функции вернет значение типа `int`. Это мог бы быть такой фрагмент кода:

```
int pleasantTemp = centToFaren(20);
```


Любая непустая функция включает инструкцию `return`. Если вы забудете добавить ее, компилятор сообщит о ее отсутствии. В функции может присутствовать более одной инструкции `return`. Такое может понадобиться, если имеется инструкция `if` с альтернативными операциями, выполняемыми при определенных условиях. Некоторые программисты осуждают использование нескольких инструкций `return` в функциях, но если функция маленькая (какими должны быть все функции), этот прием обычно не вызывает проблем.

Значение, следующее за ключевым словом `return`, может быть любым выражением, а не только единственным именем переменной. Благодаря этой особенности предыдущий пример можно сократить до

```
int centToFaren(int c)
{
  return (f = c * 9 / 5 + 32);
}
```

Если выражение состоит не только из имени единственной переменной, его следует заключить в фигурные скобки, как в предыдущем примере.

Другие типы переменных

Все переменные в примерах, приводившихся до сих пор, имели тип `int`. Безусловно, это наиболее часто используемый тип, но есть и другие, о которых вам следует знать.

float

Один из таких типов, который подошел бы для предыдущего примера преобразования температур, — тип `float`. Этот тип представляет вещественные числа, то есть числа с десятичной запятой в них, такие как 1,23. Переменные этого типа необходимы там, где целые числа не обеспечивают достаточной точности.

Взгляните на следующую формулу:

```
f = c * 9 / 5 + 32
```

Если присвоить переменной c значение 17, тогда f получит значение $17 \times 9/5 + 32$, или 62,6. Но если переменная f имеет тип int, ее значение будет усечено до 62.

Проблема становится острее, если мы не будем задумываться о порядке выполнения операций. Например, допустим, что сначала выполняется деление, как показано далее:

```
f = (c / 5) * 9 + 32
```

Согласно математическим правилам результат должен остаться равным 62,6, но если все числа будут типа int, вычисления будут выполняться так.

1. Деление 17 на 5 даст результат 3,4, который затем будет усечен до 3.
2. Умножение 3 на 9 и сложение с 32 даст в результате число 59, которое слишком далеко от 62,6.

В подобных ситуациях можно использовать вещественные числа типа float. Функция преобразования температур в следующем примере переписана с использованием вещественных чисел:

```
float centToFaren(float c)
{
  float f = c * 9.0 / 5.0 + 32.0;
  return f;
}
```

Обратите внимание на то, что ко всем числовым константам была добавлена дробная часть .0. Тем самым мы гарантировали, что компилятор будет интерпретировать их как числа типа float, а не int.

boolean

Значения типа boolean — это логические значения. Существует всего два логических значения: истина (true) и ложь (false).

В языке C слово *Boolean* записывается со строчной буквы *b*, но в обычных текстах его следует писать с прописной буквы, так как это название дано в честь английского математика Джорджа Буля, который изобрел булеву логику, играющую важную роль в информатике.

Возможно, вы не заметили этого, но мы уже сталкивались с булевыми значениями, когда знакомились с командой if. Условие в инструкции if, такое как (count==20), в действительности является выражением, возвращающим булев результат. Оператор == называется оператором равенства. Если арифметический оператор + складывает два числа и возвращает сумму, то оператор == проверяет равенство двух чисел и возвращает одно из двух значений: истина (true) или ложь (false).

Далее показано, как объявлять и использовать булевы переменные:

```
boolean tooBig = (x > 10);
if (tooBig)
{
  x = 5;
}
```

Булевыми значениями можно манипулировать с помощью булевых операторов. Подобно тому как с числами можно выполнять арифметические операции, с булевыми значениями можно выполнять логические операции. Среди логических операторов чаще других используются оператор И, который записывается как &&, и оператор ИЛИ, который записывается как ||.

На рис. 4.1 изображены таблицы истинности для операторов И и ИЛИ.

И

B \ A	Ложь	Истина
Ложь	Ложь	Ложь
Истина	Ложь	Истина

ИЛИ

B \ A	Ложь	Истина
Ложь	Ложь	Истина
Истина	Истина	Истина

Рис. 4.1. Таблицы истинности

Как показано в таблице истинности для оператора И, если оба операнда, A и B, имеют истинное (true) значение, результатом будет истинное (true) значение, в противном случае — ложное (false).

Если один из операндов оператора ИЛИ, A или B, или оба они будут иметь истинное (true) значение, результатом будет истинное (true) значение. Ложь (false) в результате возможна, только если оба операнда, A и B, имеют ложное значение.

Помимо операторов И и ИЛИ существует логический оператор НЕ, который записывается как !. Едва ли вас удивит, если я скажу, что результатом выражения «не истина» будет ложь, а результатом выражения «не ложь» — истина.

Эти операторы можно объединять в булевы выражения в инструкциях `if`, как в следующем примере:

```
if ((x > 10) && (x < 50))
```

Другие типы данных

Как вы видели, тип данных `int` и иногда `float` отлично подходит для многих ситуаций, однако в некоторых обстоятельствах могут пригодиться другие типы данных. В скетче Arduino значение типа `int` занимает 16 бит (двоичных цифр). Это позволяет представлять числа в диапазоне от –32 768 до 32 767.

Другие доступные типы данных перечислены в табл. 4.1, которую можно использовать как справочник. Некоторые из этих типов данных мы будем использовать в дальнейшем.

Важно заметить, что при превышении диапазонов значений типов данных начинают происходить странные вещи. Например, если к переменной типа `byte` со значением 255 прибавить 1, получится 0. Что еще хуже, если к переменной типа `int` со значением 32 767 прибавить 1, получится –32 768.

Пока вы полностью не освоитесь с поведением всех типов данных, я рекомендую придерживаться типа `int`, так как он хорошо подходит практически для любых ситуаций.

Таблица 4.1. Типы данных в языке C

Тип	Размер, байт	Диапазон значений	Примечания
boolean	1	Истина или ложь (1 или 0)	—
char	1	–128…+127	Используется для представления кодов символов в американской стандартной кодировке для обмена информацией (American Standard Code for Information Interchange, ASCII). Например, символ A латинского алфавита в этой кодировке имеет код 65. Отрицательные значения кодов обычно не используются
byte	1	0…255	Часто используется в операциях обмена через последовательный канал как единичный элемент данных. Подробности см. в главе 9
int	2	–32 768…+32 767	—
unsigned int	2	0…65 535	Может использоваться для увеличения точности там, где отрицательные значения не нужны. Используйте этот тип с осторожностью, так как операции над числами типа int могут производить неожиданные эффекты
long	4	–2 147 483 648… +2 147 483 647	Необходим только для представления очень больших чисел
unsigned long	4	0…4 294 967 295	См. unsigned int
float	4	–3,4028235E+38… +3,4028235E+38	—
double	4	Тот же, что для float	Обычно значения этого типа в языке C занимают 8 байт и имеют более высокую точность, чем значения типа float, при более широком диапазоне представления. Но в Arduino тип double является полным аналогом типа float

Оформление программного кода

Компилятор языка C не обращает внимания на то, как оформлен исходный код. Его вполне устроит, если вы поместите все инструкции в одну строку, разделив их точками с запятой. Однако хорошо оформленный код с ясным форматированием проще читать и сопровождать, чем исходный код, оформленный небрежно. В этом смысле чтение кода сродни чтению книги: форматирование играет важную роль.

До некоторой степени форматирование — вопрос вкуса. Никто не считает, что у него плохой вкус, поэтому представления о том, как должен выглядеть код, могут быть очень разными. Многие программисты, получая для доработки или сопровождения чужой код, начинают с того, что приводят его оформление к своему стилю.

Чтобы решить эту проблему, часто разрабатывают стандарты оформления, способствующие применению единого стиля оформления кода при создании программ.

В языке C с годами сложился фактический стандарт оформления кода, и все примеры в этой книге написаны с учетом его требований.

Отступы

В примерах скетчей, встречавшихся до сих пор, можно заметить, что некоторые строки имеют отступы слева. Например, когда определяется функция, не имеющая возвращаемого значения, ключевое слово `void` не имеет отступа слева, как и открывающая фигурная скобка в следующей строке, но весь остальной текст в фигурных скобках оформлен с отступами. Величина отступа не имеет большого значения: одни используют два пробела, другие — четыре. Для оформления отступов можно также пользоваться клавишей **Tab**. В этой книге мы используем отступы из двух пробелов.

Если в теле функции присутствует инструкция `if`, к строкам внутри фигурных скобок команды `if` можно добавить еще по два пробела слева, как в следующем примере:


```
void loop()
{
  static int count = 0;
  count ++;
  if (count == 20)
  {
    count = 0;
    delay(3000);
  }
}
```

Внутрь первой инструкции `if` можно включить еще одну инструкцию `if` с увеличенным до шести пробелов отступом.

На первый взгляд все сказанное кажется тривиальным, но стоит лишь раз столкнуться с небрежно отформатированным скетчем, и вы поймете, что это не так.

Открывающие фигурные скобки

Существуют две точки зрения на то, где должны находиться открывающие фигурные скобки в определениях функций, инструкциях `if` или циклах `for`. Одни считают, что открывающую фигурную скобку следует размещать в отдельной строке, следующей за основной командой, как это делалось до сих пор во всех примерах, другие рекомендуют размещать ее в одной строке с командой:

```
void loop() {
  static int count = 0;
  count ++;
  if (count == 20) {
    count = 0;
    delay(3000);
  }
}
```

Этот стиль часто используется для оформления программного кода на языке Java, синтаксис которого очень похож на синтаксис языка C. Я предпочитаю первый способ оформления, который, похоже, наиболее часто используется в мире Arduino.

Пробелы

Компилятор игнорирует пробелы, символы табуляции и перевода строки, интерпретируя их лишь как разделители «лексем», или слов, в скетче. Например, следующий фрагмент будет скомпилирован без проблем:

```
void loop() {static int
count=0;count++;if(
count==20){count=0;
delay(3000);}}
```

Он будет работать, но попробуйте с первой попытки прочитать его. Инструкции присваивания одни записывают так:

```
int a = 10;
```

а другие так:

```
int a=10;
```

Какой из этих двух стилей использовать, не имеет большого значения, главное — оставаться последовательным в своем выборе. Я использую первую форму.

Комментарии

Комментарий — это текст, который присутствует в скетче вместе с программным кодом, но не несет никакой функциональной нагрузки. Единственное назначение комментариев — оставить напоминание нам или другим, почему код написан именно так. Также строки комментариев могут использоваться в качестве заголовка.

Компилятор полностью игнорирует любой текст, отмеченный как комментарий. Во многие скетчи, представленные до сих пор, были включены комментарии, играющие роль заголовков.

Существуют два способа оформления комментариев:

- однострочный комментарий начинается с пары символов // и завершается в конце строки;

- многострочный комментарий начинается с пары символов /* и заканчивается парой символов */.

Следующий пример демонстрирует обе формы комментариев:

```
/* Не очень полезная функция loop.
Автор: Simon Monk
Для иллюстрации комментариев
*/
void loop() {
  static int count = 0;
  count ++; // однострочный комментарий
  if (count == 20) {
    count = 0;
    delay(3000);
  }
}
```

В этой книге я в основном использовал однострочные комментарии.

Хороший комментарий помогает понять, что происходит в скетче или как им пользоваться. Они могут пригодиться и тем, кто собирается воспользоваться вашим скетчем, и вам самим, когда вы будете просматривать скетч, к которому не прикасались несколько недель кряду.

Некоторым в курсе обучения программированию говорят, что чем больше комментариев, тем лучше. Многие опытные программисты говорят, что хорошо написанный код не требует подробных разъяснений, потому что говорит сам за себя. Используйте комментарии:

- чтобы объяснить какую-то свою хитрость или нечто неочевидное;
- чтобы описать, что должен сделать пользователь, например: // этот контакт следует соединить с транзистором, управляющим реле;
- чтобы оставить напоминание самому себе, например: // что сделать: привести в порядок эту мешанину.

В последнем пункте иллюстрируется полезный прием оформления заданий на будущее в комментариях. Программисты часто вставляют в программный код такие комментарии, чтобы напомнить са-

мим себе, что еще нужно сделать. Они всегда могут воспользоваться функцией поиска в интегрированной среде разработки (Integrated Development Environment, IDE), чтобы найти в программе все комментарии вида // что сделать.

Ниже перечислены случаи, когда не следует добавлять комментарии:

- чтобы указать на что-то очевидное, например: a = a + 1; // прибавить 1 к a.
- чтобы попытаться объяснить работу плохо написанного кода. Не делайте этого, просто сразу пишите чистый и ясный код.

В заключение

Эта глава была с небольшим уклоном в теорию. Вы должны были усвоить некоторые абстрактные понятия, касающиеся организации скетчей в функции и стилей оформления программного кода, следование которым поможет в конечном итоге сэкономить время.

В следующей главе вы начнете применять уже полученные знания и знакомиться с приемами структурирования данных и использования текстовых строк.

5

Массивы и строки

После прочтения главы 4 у вас должно сложиться представление, как структурировать свои скетчи, чтобы застраховаться от ненужных сложностей. Если и есть что-то, что любит хороший программист, то это отсутствие ненужных сложностей. А теперь обратимся к данным, используемым в скетчах.

Книга «Algorithms + Data Structures = Programs» Никлауса Вирта[1] была написана довольно давно, но все еще достаточно точно описывает суть информатики вообще и программирования в частности. Я настоятельно

[1] *Вирт Н.* Алгоритмы + структуры данных = программы. — М.: Мир, 1985.

рекомендую ее всем тем, кто страдает от постоянно преследующих их ошибок в программах. В этой книге также раскрывается мысль, что для создания хороших программ нужно обдумывать не только алгоритмы (порядок действий), но и структуры данных.

Вы уже видели циклы, инструкции if и все остальное, что составляет «алгоритмическую» сторону программирования для Arduino, теперь мы посмотрим, как правильно организовать данные.

Массивы

Массивы — это структуры, содержащие списки значений. Переменные, которые вы видели до сих пор, могли хранить только одно значение, обычно типа int. Массив, напротив, способен хранить список значений, к любому из которых можно обратиться по номеру его позиции в списке.

В языке C, как и во многих других основных языках, нумерация позиций в массиве начинается с 0, а не с 1. Это означает, что первый элемент хранится в позиции с номером 0.

Для иллюстрации использования массивов можно было бы написать скетч, который непрерывно посылает сообщение SOS азбукой Морзе с помощью светодиода на плате Arduino.

Азбука Морзе играла важную роль в обмене сообщениями в XIX и XX веках. Так как буквы в азбуке Морзе кодируются сериями из точек и тире, их можно передавать по телеграфным проводам, через радиоэфир или в виде световых сигналов. Буквы «SOS» (аббревиатура от *англ.* save our souls — спасите наши души) все еще служат международным сигналом бедствия.

Буква «S» представлена в азбуке Морзе тремя точками (или короткими вспышками), а буква «O» — тремя тире (продолжительны-

ми вспышками). Мы могли бы использовать массив целых чисел int для хранения продолжительностей каждой отдельной вспышки и затем с помощью цикла for выполнить обход элементов массива и воспроизвести вспышки с указанными в массиве продолжительностями.

Для начала посмотрим, как можно создать массив целых чисел с определенными продолжительностями:

```
int durations[] = {200, 200, 200, 500, 500, 500, 200,
                   200, 200};
```

Квадратные скобки [], стоящие после имени переменной, указывают, что она является массивом.

В данном случае содержимое массива определяется в объявлении. С этой целью используются фигурные скобки и список значений, разделенных запятыми. Не забудьте поставить точку с запятой в конце строки.

Получить доступ к конкретному элементу массива можно с помощью формы записи с квадратными скобками. То есть получить значение первого элемента массива можно так:

```
durations[0]
```

Для иллюстрации создадим массив и затем выведем все значения в монитор последовательного порта:

```
// sketch 5-01
int durations[] = {200, 200, 200, 500, 500, 500, 200,
                   200, 200};

void setup()
{
  Serial.begin(9600);
  for (int i = 0; i < 9; i++)
  {
    Serial.println(durations[i]);
  }
}

void loop() {}
```

Обратите внимание, что массивы можно объявлять с ключевым словом `const` как обычные переменные, если содержимое массива не должно изменяться во время работы скетча.

Выгрузите скетч в плату и откройте монитор последовательного порта. Если все было сделано правильно, вы увидите последовательность чисел, подобную той, что показана на рис. 5.1.

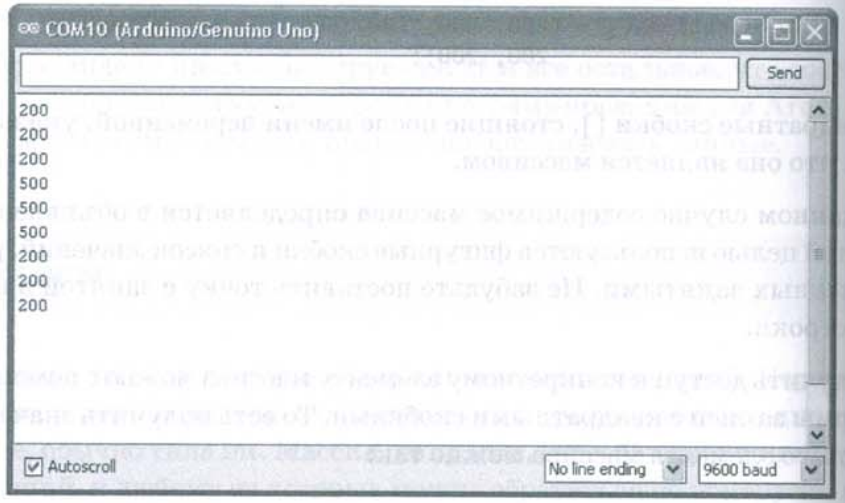


Рис. 5.1. Вывод скетча 5-01 в окне монитора последовательного порта

Все выглядит достаточно очевидным — если вам понадобится добавить в массив еще несколько значений продолжительности, достаточно будет добавить их в список в фигурных скобках и заменить число 9 в цикле `for` на новый размер массива.

При работе с массивами следует проявлять осторожность, потому что компилятор даже не попытается предотвратить попытку обратиться к элементам за пределами массива. Это объясняется тем, что в действительности переменная-массив — это указатель с адресом в памяти, как показано на рис. 5.2.

Программы размещают и обычные переменные, и массивы *в памяти*. Компьютерная память организована более жестко, чем человеческая. Проще представить память в Arduino как коллекцию ячеек. Когда вы определяете, например, массив с девятью элементами,

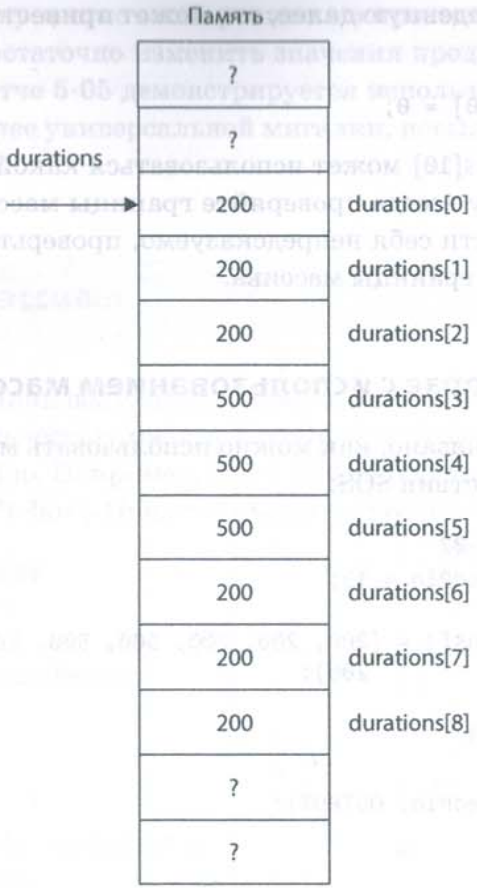


Рис. 5.2. Массивы и указатели

следующие девять доступных ячеек резервируются для хранения данных, а переменная, как говорят, указывает на первую ячейку, или элемент массива.

Вернемся к проблеме доступа к элементам за пределами массива. Если вы решите обратиться к элементу `durations[10]`, то получите целочисленное значение, но это значение может быть чем угодно. В этом нет чего-то особенно опасного, кроме того, что, случайно обратившись к элементу за пределами массива, скетч почти наверняка станет работать неправильно. Намного более серьезные неприятности могут произойти, если попытаться изменить значение за пределами массива. Например, если добавить в программу ин-

струкцию, приведенную далее, это может привести к нарушениям в работе скетча:

```
durations[10] = 0;
```

Ячейка durations[10] может использоваться какой-то другой переменной. Поэтому всегда проверяйте границы массива. Если вдруг скетч начнет вести себя непредсказуемо, проверьте, не связано ли это с выходом за границы массива.

SOS в коде Морзе с использованием массивов

В скетче 5-02 показано, как можно использовать массив, чтобы послать сигнал бедствия SOS:

```
// sketch 5-02
const int ledPin = 13;

int durations[] = {200, 200, 200, 500, 500, 500, 200, 200,
                   200};

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

void loop()
{
  for (int i = 0; i < 9; i++)
  {
    flash(durations[i]);
  }
  delay(1000);
}

void flash(int delayPeriod)
{
  digitalWrite(ledPin, HIGH);
  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);
}
```


Очевидное преимущество этого решения в простоте смены сообщения, для чего достаточно изменить значения продолжительностей в массиве. В скетче 5-05 демонстрируется использование массивов для создания более универсальной мигалки, посылающей сигналы азбукой Морзе.

Строковые массивы

В программировании под словом *строка* подразумевается не что иное, как последовательность символов. Строки дают возможность работать с текстом в Arduino. Например, скетч 5-03 каждую секунду будет отправлять текст «Hello» («Привет») в монитор последовательного порта:

```
// sketch 5-03
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.println("Hello");
  delay(1000);
}
```

Строковые литералы

Строковые литералы заключаются в двойные кавычки. Они являются литералами в том смысле, что не изменяются, как не изменяется целое число 123 в коде программы.

Как можно догадаться, переменные способны хранить строки. Кроме того, существует расширенная библиотека для работы со строками, но пока мы будем использовать стандартные строки языка C, такие как в скетче 5-03.

В языке C строковый литерал фактически является массивом значений типа char, который немного похож на тип int, то есть тоже яв-

ляется числовым, но может хранить числа в диапазоне от 0 до 127 и представляет один символ. Символом может быть буква алфавита, цифра, знак препинания или специальный символ, такой как табуляция или перевод строки. Числовые коды символов определяются американской стандартной кодировкой для обмена информацией (American Standard Code for Information Interchange, ASCII). Некоторые из наиболее часто употребляемых символов с кодами ASCII перечислены в табл. 5.1.

Таблица 5.1. Наиболее часто употребляемые символы с кодами ASCII

Символ	Код ASCII (десятичный)
a–z	97–122
A–Z	65–90
0–9	48–57
Пробел	32

Строковый литерал "Hello" фактически является массивом символов, как показано на рис. 5.3.

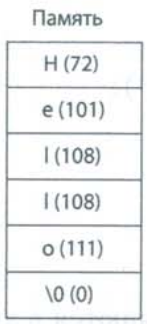


Рис. 5.3. Строковый литерал "Hello"

Обратите внимание на то, что строковые литералы оканчиваются специальным пустым (null) символом. Этот символ служит признаком конца строки.

Строковые переменные

Как можно догадаться, строковые переменные очень похожи на переменные-массивы, за исключением того, что первые поддерживают удобный метод определения начального значения:

```
char name[] = "Hello";
```

Эта инструкция определяет массив символов и инициализирует его словом «Hello», добавляя при этом заключительный пустой символ (ASCII 0), отмечающий конец строки.

Предыдущий пример наиболее хорошо соответствует вашим теперешним знаниям о массивах, но на практике чаще используется такая форма записи:

```
char *name = "Hello";
```

Она эквивалентна предыдущей, а символ * здесь служит признаком того, что переменная является указателем. Идея заключается в том, что имя указывает на первый элемент char в массиве, то есть представляет адрес в памяти, где хранится буква *H*.

Скетч 5-03 можно переписать с использованием переменной вместо строковой константы, как показано далее:

```
// sketch 5-04
char message[] = "Hello";

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  Serial.println(message);
  delay(1000);
}
```

Транслятор в азбуку Морзе

Давайте теперь объединим все, что мы узнали о массивах и строках, и напишем более сложный скетч, который будет принимать сообщения из монитора последовательного порта и передавать их азбукой Морзе через светодиод.

Соответствия между буквами и кодами в азбуке Морзе показаны в табл. 5.2.

Таблица 5.2. Азбука Морзе

Буква	Код Морзе	Буква	Код Морзе	Цифра	Код Морзе
A	.–	N	–.	0	–––––
B	–...	O	–––	1	.––––
C	–.–.	P	.––.	2	..–––
D	–..	Q	––.–	3	...––
E	.	R	.–.	4	....–
F	..–.	S	...	5	.....
G	––.	T	–	6	–....
H	....	U	..–	7	––...
I	..	V	...–	8	————
J	.–––	W	.––	9	————
K	–.–	X	–..–		
L	.–..	Y	–.––		
M	––	Z	––..		

Вот некоторые из правил азбуки Морзе: тире в три раза продолжительнее точки, интервал между тире или точками равен продолжительности точки, интервал между буквами равен продолжительности тире, а интервал между словами равен продолжительности семи точек.

В данном проекте мы не будем волноваться о знаках препинания, хотя это может стать интересным упражнением для вас — самостоятельно добавить в скетч их поддержку. Полный список соответствий

между кодами азбуки Морзе и символами можно найти по адресу: *https://ru.wikipedia.org/wiki/Азбука_Морзе*.

Данные

Мы будем конструировать этот пример шаг за шагом и начнем со структуры данных, хранящей азбуку Морзе.

Важно понимать, что эта проблема имеет множество решений. Каждый программист идет своим путем и решает ее по-своему. То есть было бы ошибкой считать: «Я никогда не додумался бы до этого». Впрочем, может быть, именно до этого вы и не додумались бы, но вполне могли найти другое, лучшее решение. Все мы думаем по-разному, и решение, которое приводится далее, просто первое, что пришло автору в голову.

Найденное представление данных является всего лишь способом выразить табл. 5.2 на языке С. Фактически данные разбиты на две таблицы: для букв и для цифр. Ниже приводится определение структуры данных для букв:

```
char* letters[] = {
  ".-", "-...", "-.-.", "-..", ".", // A-I
  "..-.", "--.", "....", "..",
  ".---", "-.-", ".-..", "--", "-.", // J-R
  "---", ".--.", "--.-", ".-.",
  "...", "-", "..-", "...-", ".--", // S-Z
  "-..-", "-.--", "--.."
};
```

Это массив строковых литералов. А так как строковые литералы являются массивами символов, данная структура фактически является массивом массивов — вполне допустимой и очень удобной структурой.

Чтобы в такой структуре найти код Морзе для буквы «А», нужно обратиться к элементу `letters[0]`, в котором хранится строка `.-`. Такое решение не особенно эффективно, потому что для представления каждого тире или каждой точки используется целый байт (8 бит) памяти, хотя вполне хватило бы одного бита. Однако я легко могу оправдать такой подход, сказав, что этот массив занимает что-то около 90 байт, тогда как у нас имеется целых 2 Кбайт. Не менее важна и простота кода.

Для цифр используется аналогичный подход:

```
char* numbers[] = {
  "-----", ".----", "..---", "...--", "....-",
  ".....", "-....", "--...", "---..", "----."};
```

Глобальные переменные и настройка

В скетче необходимо также определить пару глобальных переменных: одну для хранения длительности точки и другую для хранения номера контакта, к которому подключен светодиод:

```
const int dotDelay = 200;
const int ledPin = 13;
```

Функция `setup` очень проста — в ней нужно лишь настроить работу контакта `ledPin` в режиме выхода и активизировать последовательный порт:

```
void setup()
{
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}
```

Функция loop

Теперь можно приступать к реализации функции `loop`. Алгоритм работы этой функции описывается далее.

Если в последовательной линии есть символ для чтения:

- если это буква, вывести ее с использованием массива letters;
- если это цифра, вывести ее с использованием массива numbers;
- если это пробел, выполнить задержку, равную четырем точкам.

Вот и все. Нет необходимости заглядывать слишком далеко вперед. Этот алгоритм описывает, что требуется сделать, то есть ваши намерения, и такой стиль программирования называют *программированием по намерениям* (programming by intention). А вот как выглядит этот алгоритм на языке C:


```
void loop()
{
  char ch;
  if (Serial.available() > 0)
  {
    ch = Serial.read();
    if (ch >= 'a' && ch <= 'z')
    {
      flashSequence(letters[ch - 'a']);
    }
    else if (ch >= 'A' && ch <= 'Z')
    {
      flashSequence(letters[ch - 'A']);
    }
    else if (ch >= '0' && ch <= '9')
    {
      flashSequence(numbers[ch - '0']);
    }
    else if (ch == ' ')
    {
      delay(dotDelay * 4); // интервал между словами
    }
  }
}
```

Здесь есть несколько моментов, требующих пояснения. Взгляните на вызов `Serial.available()`. Чтобы понять его суть, нужно немного знать, как плата Arduino обменивается данными с компьютером через USB. Этот процесс изображен на рис. 5.4.

Когда компьютер посылает данные из монитора последовательного порта в плату Arduino, протокол и уровни сигналов USB преобразуются в нечто, более понятное микроконтроллеру на плате Arduino. Это преобразование выполняет специализированная микросхема, находящаяся на плате Arduino. Затем данные принимает компонент микроконтроллера, который называется универсальным асинхронным приемопередатчиком (УАПП — Universal Asynchronous Receiver/Transmitter, UART). Приемопередатчик помещает полученные данные в буфер — специальную область в памяти (128 байт), где они хранятся до тех пор, пока не будут прочитаны.

Рис. 5.4. Обмен данными по последовательной линии с платой Arduino

Прием и передача данных происходят независимо от того, что делает скетч. То есть даже когда скетч весело мигает светодиодом, данные могут продолжать прибывать в буфер и оставаться там, ожидая, пока их прочитают. Буфер чем-то похож на почтовый ящик.

Чтобы проверить, пришла ли «почта», нужно вызвать функцию `Serial.available()`. Она вернет число байтов данных, ожидающих чтения. Если в буфере нет непрочитанных сообщений, функция вернет 0. Именно поэтому инструкция `if` проверяет, превышает ли число доступных байтов значение 0, и, если условие выполняется, тут же производится чтение следующего доступного символа вызовом функции `Serial.read()`. Результат этой функции присваивается локальной переменной `ch`.

Затем выполняется следующая инструкция `if`, которая определяет, какого рода символ прочитан:

```
if (ch >= 'a' && ch <= 'z')
{
  flashSequence(letters[ch - 'a']);
}
```

На первый взгляд кажется странным применение операторов `<=` и `>=` для сравнения символов. Однако в этом нет ничего необычного, если вспомнить, что символы представлены числами (кодами ASCII).

То есть если код символа находится в диапазоне между «a» и «z» (97 и 122), можно утверждать, что от компьютера получена строчная буква. Затем вызывается функция `flashSequence`, которую мы пока не написали, и ей передается строка из точек и тире. Например, чтобы воспроизвести букву «a», нужно передать функции строковый аргумент `.-`.

Мы решили реализовать вывод символов на светодиод в виде отдельной функции. Не стоит делать этого в функции `loop`, чтобы сохранить программный код простым для чтения.

Вот как на языке C определяется, какую строку из точек и тире передать функции `flashSequence`:

```
letters[ch - 'a']
```

И снова данный код может показаться странным. Выглядит он так, как будто выполняется вычитание одного символа из другого. В действительности это вполне осмысленное действие, определяющее разность числовых значений кодов ASCII.

Вспомните, что коды азбуки Морзе мы сохранили в массиве `letters`. То есть первый элемент массива хранит строку из точек и тире, представляющую букву «a», второй элемент — строку из точек и тире, представляющую букву «b», и т. д. Нам нужно найти позицию элемента в массиве, соответствующего букве, только что прочитанной из буфера. Позиция элемента для любой строчной буквы равна разности кода буквы и кода символа *a*. Например, выражение *a* – *a* фактически будет выполнено как 97 – 97 = 0. Аналогично выражение *c* – *a* фактически будет выполнено как 99 – 97 = 2. То есть если в переменной `ch` лежится буква «c», выражение в квадратных скобках даст в результате число 2, и вы получите элемент с номером 2 в массиве, то есть `-.-..`

В этом разделе речь шла только о строчных буквах. Однако прописные буквы и цифры обрабатываются аналогично.

Функция flashSequence

Предполагалось, что у нас будет функция с именем `flashSequence`, и мы даже использовали ее, поэтому теперь нужно написать эту функцию. Мы решили, что она будет принимать строку из точек и тире

и на ее основе включать и выключать светодиод с соответствующими задержками.

Алгоритм работы функции можно представить как последовательность шагов: для каждого элемента строки из точек и тире, такого как .-.-, включить светодиод на определенное время, а затем выключить его.

Следуя концепции программирования по намерениям, сохраним код функции максимально простым.

Последовательности точек и тире для разных букв могут иметь разную длину, поэтому нужно организовать цикл по элементам строки, пока не встретится признак конца, \0. Нам также понадобится переменная-счетчик с именем i, которая на входе в функцию получает начальное значение 0 и увеличивается после обработки очередной точки или тире:

```
void flashSequence(char* sequence)
{
  int i = 0;
  while (sequence[i] != '\0')
  {
    flashDotOrDash(sequence[i]);
    i++;
  }
  delay(dotDelay * 3); // задержка между буквами
}
```

И снова фактическое включение/выключение светодиода для каждой отдельной точки или тире делегируется новой функции с именем flashDotOrDash. Наконец, после вывода всех точек и тире, соответствующих букве или цифре, нужно сделать паузу, равную длительности трех точек. Обратите внимание на пример использования комментария.

Функция flashDotOrDash

Последняя функция, которую нужно написать, — это функция, которая фактически включает и выключает светодиод. Функция будет получать в аргументе единственный символ — точку (.) или тире (-).

Эта функция просто должна включить светодиод, задержаться на длительность точки, если в аргументе получена точка, и на длительность трех точек, если в аргументе получено тире, а затем выключить светодиод. В конце она должна выдержать паузу, равную продолжительности одной точки, чтобы обеспечить интервал между вспышками:

```
void flashDotOrDash(char dotOrDash)
{
  digitalWrite(ledPin, HIGH);
  if (dotOrDash == '.')
  {
    delay(dotDelay);
  }
  else // не точка, значит, тире
  {
    delay(dotDelay * 3);
  }
  digitalWrite(ledPin, LOW);
  delay(dotDelay); // задержка между вспышками
}
```

Объединяем все вместе

Далее приводится полный листинг скетча 5-05. Выгрузите его в плату Arduino и опробуйте. Не забудьте, что для этого нужно открыть монитор последовательного порта, ввести какой-нибудь текст в поле ввода вверху и щелкнуть на кнопке **Send** (Отправить). После этого вы должны увидеть, как плата воспроизведет этот текст азбукой Морзе в виде серии вспышек:

```
// sketch 5-05
const int dotDelay = 200;
const int ledPin = 13;

char* letters[] = {
  ".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....",
  "..",   // A-I
  ".---", "-.-", ".-..", "--", "-.", "---", ".--.", "--.-",
  ".-.", // J-R
  "...", "-", "..-", "...-", ".--", "-..-", "-.--", "--.."
  // S-Z
```

```
};
char* numbers[] = {"-----", ".----", "..---", "...--",
                   "....-",
"....."  "-....", "--...", "---..", "----."};

void setup()
{
  pinMode(ledPin, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  char ch;
  if (Serial.available() > 0)
  {
    ch = Serial.read();
    if (ch >= 'a' && ch <= 'z')
    {
      flashSequence(letters[ch - 'a']);
    }
    else if (ch >= 'A' && ch <= 'Z')
    {
      flashSequence(letters[ch - 'A']);
    }
    else if (ch >= '0' && ch <= '9')
    {
      flashSequence(numbers[ch - '0']);
    }
    else if (ch == ' ')
    {
       delay(dotDelay * 4); // пауза между словами
    }
  }
}

void flashSequence(char* sequence)
{
  int i = 0;
  while (sequence[i] != '\0')
  {
```



```
    flashDotOrDash(sequence[i]);
    i++;
  }
  delay(dotDelay * 3); // пауза между буквами
}

void flashDotOrDash(char dotOrDash)
{
  digitalWrite(ledPin, HIGH);
  if (dotOrDash == '.')
  {
    delay(dotDelay);
  }
  else // не точка, значит, тире
  {
    delay(dotDelay * 3);
  }
  digitalWrite(ledPin, LOW);
  delay(dotDelay); // пауза между вспышками
}
```

Этот скетч включает функцию loop, которая вызывается автоматически и сама вызывает функцию flashSequence, которая, в свою очередь, вызывает flashDotOrDash, а та вызывает стандартные функции digitalWrite и delay из библиотеки Arduino!

Именно так должны выглядеть скетчи. Разделение операций на функции упростит создание рабочего программного кода, и вам будет проще понять его, когда вы вернетесь к нему спустя какое-то время.

В заключение

Помимо знакомства со строками и массивами вы создали довольно сложный транслятор азбуки Морзе, что, как я надеюсь, наглядно доказало важность разделения программного кода на функции.

В следующей главе вы познакомитесь с операциями ввода и вывода, под которыми подразумеваются ввод и вывод аналоговых и цифровых сигналов в Arduino.

6

Ввод и вывод

Устройства Arduino часто описывают термином «physical computing», а под этим подразумевается возможность подключения к плате Arduino дополнительных электронных устройств. Поэтому вы должны понимать, как использовать разные контакты (входы и выходы) на плате.

Выходы могут быть цифровыми, то есть выводить напряжение 0 В или 5 В, или аналоговыми, то есть выводить произвольное напряжение в диапазоне 0...5 В. Впрочем, в действительности все несколько сложнее, в чем вы вскоре убедитесь.

Аналогично входы могут быть цифровыми (например, определяющими факт нажатия кнопки) или аналоговыми (например, подключенные к фотоэлементу).

В книге, по существу описывающей приемы программирования, а не аппаратные средства, мы попробуем избежать сползания в обсуждение специфики электроники. Однако вы лучше поймете рассматриваемые здесь проблемы, если вооружитесь мультиметром и коротким куском провода в изоляции.

Желающим поближе познакомиться с электроникой я могу порекомендовать мою книгу Hacking Electronics (TAB/McGraw-Hill, 2013)[1].

Цифровые выходы

В предыдущих главах использовался светодиод, подключенный к контакту 13 на плате Arduino. Например, в главе 5 этот светодиод использовался как источник сигналов азбуки Морзе. На плате Arduino есть целая коллекция контактов для вывода цифровых сигналов.

Давайте поэкспериментируем с одним из таких контактов. Возьмем, к примеру, контакт 4 и посмотрим, как он действует, а чтобы вы могли наблюдать за происходящим, подключите свой мультиметр к плате Arduino. Как это сделать, показано на рис. 6.1. Если в комплект мультиметра входят зажимы типа «крокодил», зачистите изоляцию на концах отрезков провода и с одной стороны зафиксируйте концы проводов в зажимах, а другие их концы вставьте в гнезда на плате Arduino. Если таких зажимов у вас нет, просто накрутите концы проводов на щупы мультиметра.

Мультиметр должен быть настроен на измерение постоянного напряжения в диапазоне 0...20 В. Отрицательный провод (черный) следует подключить к «земле» (контакт GND), а положительный — к контакту D3. Провода одним концом должны быть подключены к щупам мультиметра, а другим вставлены в гнезда на плате Arduino.

---

[1] *Монк С.* Практическая электроника: иллюстрированное руководство для радиолюбителей. М.: Вильямс, 2016. — *Примеч. пер.*

Рис. 6.1. Измерение напряжения на выходе с помощью мультиметра

Загрузите скетч 6-01:

```
//sketch 6-01
const int outPin = 4;

void setup()
{
  pinMode(outPin, OUTPUT);
  Serial.begin(9600);
  Serial.println("Enter 1 or 0");
}

void loop()
{
  if (Serial.available() > 0)
  {
    char ch = Serial.read();
    if (ch == '1')
    {
      digitalWrite(outPin, HIGH);
    }
```



```
    else if (ch == '0')
    {
      digitalWrite(outPin, LOW);
    }
  }
}
```

В начале скетча можно видеть команду pinMode. Она должна использоваться для каждого контакта, с которым предполагается работать, чтобы Arduino могла настроиться на работу с электроникой, подключенной к этим контактам, и использовать их как входы или выходы:

```
pinMode(outPin, OUTPUT);
```

Как нетрудно догадаться, pinMode — это встроенная функция. Ее первый аргумент — номер контакта (значение типа int), а второй аргумент — режим работы, INPUT (вход), INPUT_PULLUP (вход с подтягивающим резистором) или OUTPUT (выход). Обратите внимание на то, что имена режимов должны записываться только прописными буквами.

Функция loop ожидает команды 1 или 0 из монитора последовательного порта. Если будет получена команда 1, она включит контакт 3, если 0 — выключит.

Выгрузите скетч в плату Arduino и откройте монитор последовательного порта, как показано на рис. 6.2.

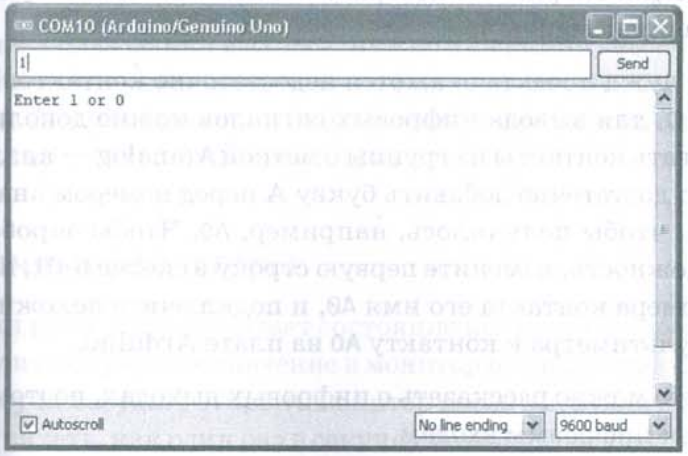


Рис. 6.2. Монитор последовательного порта

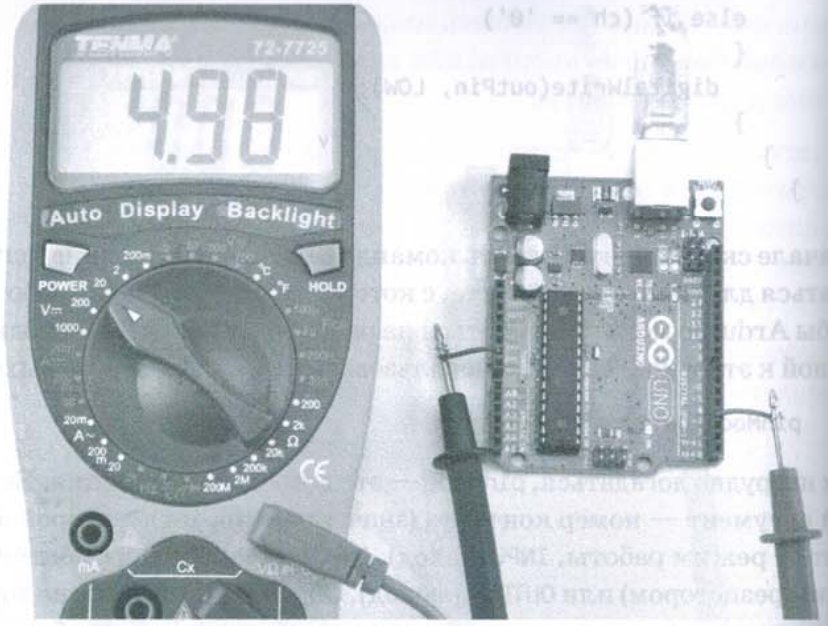


Рис. 6.3. На выход подан высокий уровень напряжения

Если мультиметр соединен с контактами на плате Arduino и включен, вы должны увидеть, как изменяется напряжение между значениями 0 и 5 В (приблизительно) при отправке команд из монитора последовательного порта, нажимая клавиши **1** и затем **Return** или **0** и затем **Return**. На рис. 6.3 изображены показания мультиметра после отправки команды **1**.

Если для нужд проекта окажется недостаточно контактов в группе с меткой **D**, для вывода цифровых сигналов можно дополнительно использовать контакты из группы с меткой **A** (analog — аналоговые). Для этого достаточно добавить букву A перед номером аналогового контакта, чтобы получилось, например, **A0**. Чтобы опробовать такую возможность, измените первую строку в скетче 6-01, подставив вместо номера контакта его имя **A0**, и подключите положительный провод мультиметра к контакту **A0** на плате Arduino.

Это все, что можно рассказать о цифровых выходах, поэтому теперь перейдем к цифровым входам.

Цифровые входы

Цифровые входы часто используются для определения момента, когда будет включен или выключен некоторый выключатель. Цифровой вход может иметь только одно из двух состояний: включено или выключено. Если напряжение на входе меньше 2,5 В (половина от 5 В), считается, что вход имеет состояние 0 (выключено), а если больше 2,5 В — состояние 1 (включено).

Отсоедините мультиметр и выгрузите в плату скетч 6-02:

```
//sketch 6-02
const int inputPin = 5;

void setup()
{
  pinMode(inputPin, INPUT);
  Serial.begin(9600);
}

void loop()
{
  int reading = digitalRead(inputPin);
  Serial.println(reading);
  delay(1000);
}
```

Как и при использовании выходов, нужно в функции `setup` сообщить плате Arduino, что некоторый контакт будет использоваться как цифровой вход. Получить состояние цифрового входа можно с помощью функции `digitalRead`. Она возвращает 0 или 1.

Нагрузочные резисторы

Скетч один раз в секунду читает состояние контакта цифрового входа и выводит полученное значение в монитор последовательного порта. Поэтому не забудьте открыть монитор после выгрузки скетча. Вы должны увидеть, как один раз в секунду появляется новое значение.

Вставьте один конец провода в гнездо D5 и зажмите между пальца ми другой его конец, как показано на рис. 6.4.

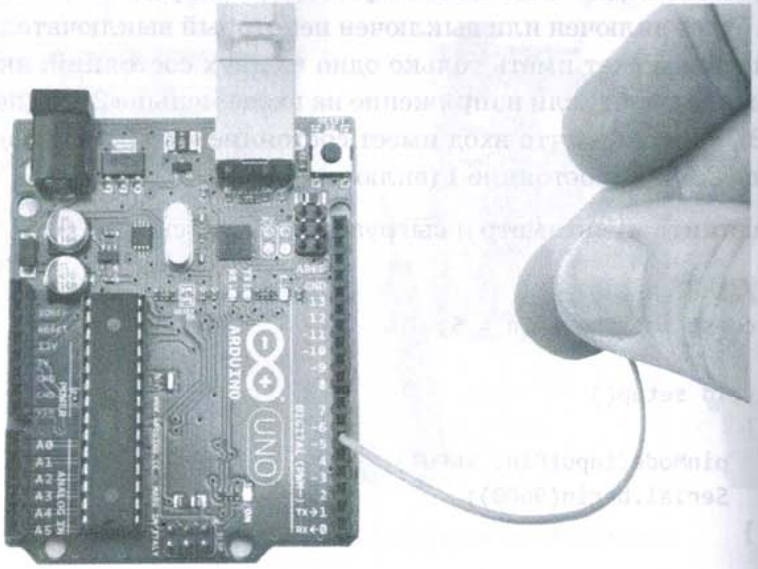


Рис. 6.4. Цифровой вход и человек как антенна

Продолжайте удерживать конец провода пальцами и следите за появлением значений в окне монитора последовательного порта. Вы должны увидеть смесь единиц и нулей. Причина такого поведе ния заключается в том, что входы на плате Arduino очень чувстви тельны. Ваше тело выступает в качестве антенны, передавая на про вод наведенное электричество.

Теперь вставьте конец провода, который зажимали между пальца ми, в гнездо с подписью +5V, как показано на рис. 6.5. На монито должны отображаться только единицы.

Теперь извлеките конец провода из гнезда +5V и вставьте его в гнез до GND. Как можно догадаться, на мониторе теперь должна отобра жаться последовательность нулей.

Обычно цифровые входы используются для подключения выклю чателей. На рис. 6.6 показано, как, на первый взгляд, должно вы глядеть такое подключение. Проблема здесь в том, что, если выклю

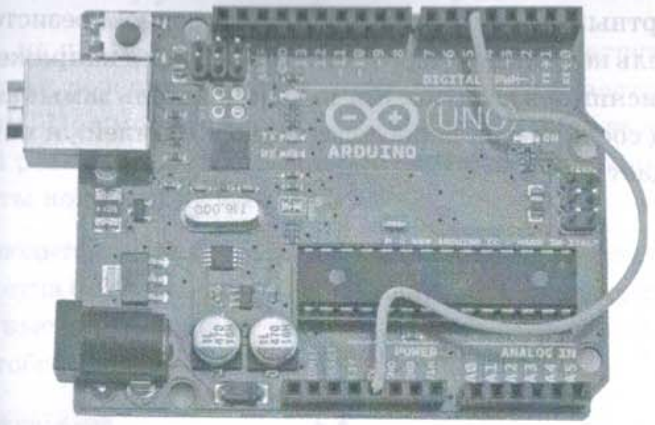


Рис. 6.5. Соединение контакта 5 с контактом +5V

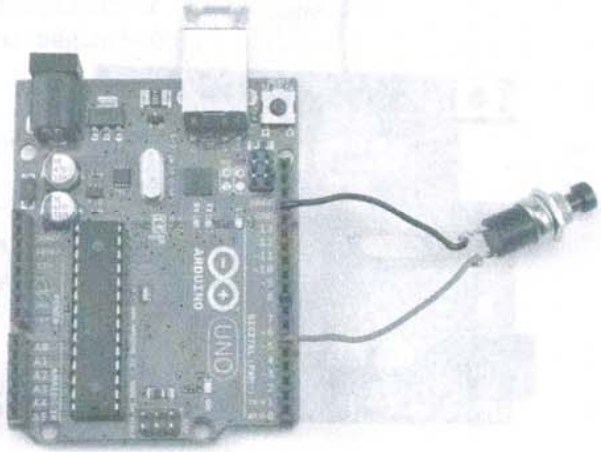


Рис. 6.6. Подключение выключателя к плате Arduino

ватель не замкнут, вход оказывается не соединен ни с чем. В этом случае говорят, что он висит в воздухе и легко может подавать ложные сигналы. Необходимо сделать поведение цифрового входа более предсказуемым, а добиться этого можно с помощью так называемого нагрузочного резистора. Далее вы узнаете, как задействовать внутренние нагрузочные резисторы, встроенные в Arduino, и избежать

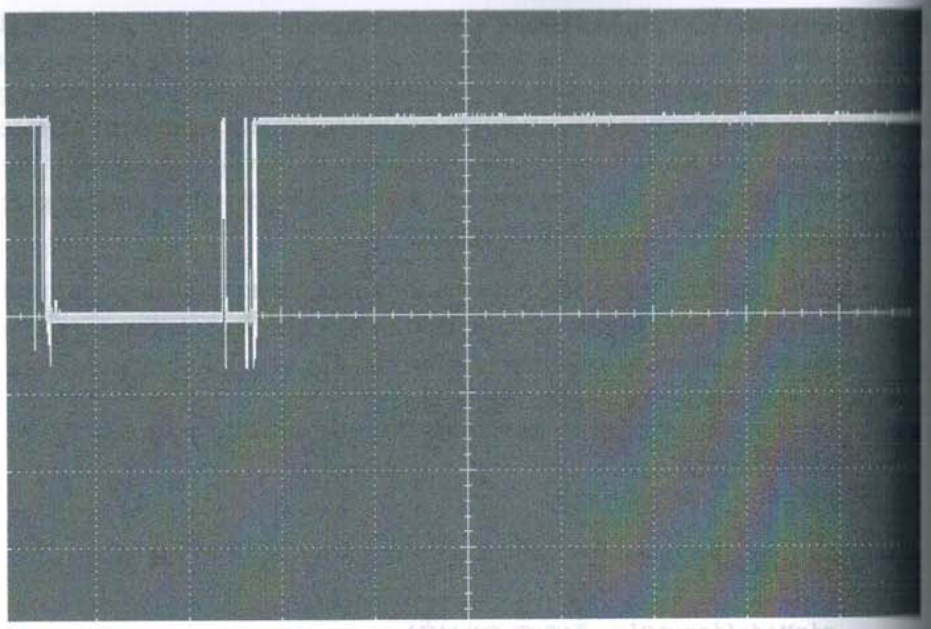


Рис. 6.8. Осциллограмма момента нажатия кнопки

Иногда дребезг контактов вообще не важен. Например, скетч 6-04 зажигает светодиод, пока кнопка нажата. На практике вам едва ли придет в голову использовать Arduino для этого, но сейчас мы находимся в царстве теории, а не практики:

```
//sketch 6-04
const int inputPin = 5;
const int ledPin = 13;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(inputPin, INPUT_PULLUP);
}

void loop()
{
  int switchOpen = digitalRead(inputPin);
  digitalWrite(ledPin, ! switchOpen);
}
```


Взгляните на функцию loop в скетче 6-04, она читает состояние цифрового входа и присваивает его переменной switchOpen. Это будет число 0, если кнопка нажата, и число 1, если нет (напомню, что на контакт подается напряжение через нагрузочный резистор, когда кнопка отжата).

Когда программа вызывает digitalWrite, чтобы включить или выключить светодиод, это значение нужно обратить. Делается это с помощью оператора НЕ !.

Если выгрузить этот скетч в плату и соединить проводом контакты 05 и GND (рис. 6.9), светодиод должен зажечься. Здесь может наблюдаться дребезг контактов, но он занимает настолько короткий промежуток времени, что, скорее всего, останется не замеченным вами, и его присутствие не имеет значения.

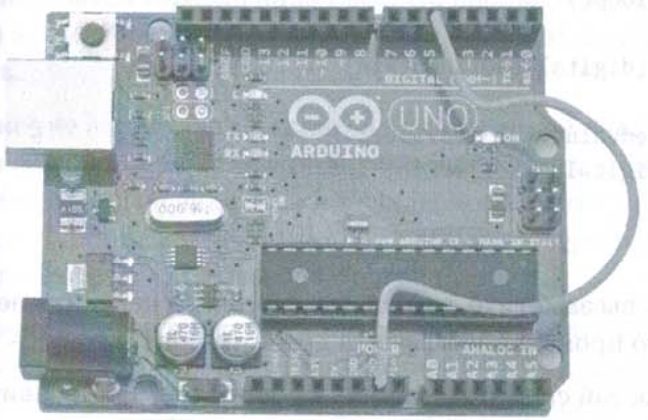


Рис. 6.9. Отрезок провода в роли кнопки

Одна из ситуаций, когда дребезг контактов может оказаться существенным, — это если скетч будет чередовать состояние «включено/выключено» светодиода при нажатии кнопки. То есть когда при первом нажатии кнопки светодиод включается, при втором — выключается. Если в этом случае будет наблюдаться эффект дребезга контактов, состояние светодиода после нажатия кнопки будет зависеть от числа импульсов, сгенерированных контактами кнопки, — четного или нечетного.

Скетч 6-05 просто переключает светодиод без попытки устранить дребезг контактов. Опробуйте его, используя в роли кнопки отрезок провода, соединяющий контакты D5 и GND (или подключите кнопку, если она у вас есть):

```
// sketch 6-05
const int inputPin = 5;
const int ledPin = 13;
int ledValue = LOW;

void setup()
{
  pinMode(inputPin, INPUT_PULLUP);
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  if (digitalRead(inputPin) == LOW)
  {
    ledValue = ! ledValue;
    digitalWrite(ledPin, ledValue);
  }
}
```

Возможно, вы заметите, что иногда светодиод переключается, а иногда нет. Это проявление эффекта дребезга контактов!

Самый простой способ устранить эту проблему — добавить задержку после определения первого момента нажатия кнопки, как показано в скетче 6-06:

```
// sketch 6-06
const const int inputPin = 5;
int ledPin = 13;
int ledValue = LOW;

void setup()
{
  pinMode(inputPin, INPUT_PULLUP);
```



```
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  if (digitalRead(inputPin) == LOW)
  {
    ledValue = ! ledValue;
    digitalWrite(ledPin, ledValue);
    delay(500);
  }
}
```

Благодаря задержке любые импульсы будут игнорироваться в течение ближайших 500 мс, а к тому времени действие эффекта дребезга контактов прекратится. Теперь вы должны заметить, что переключение выполняется гораздо надежнее. Но наблюдается интересный побочный эффект: если удерживать кнопку нажатой, светодиод будет мигать.

Если слежение за кнопкой является единственной задачей скетча, то данная задержка не является проблемой. Однако если функция loop решает и другие задачи, тогда задержка может превратиться в проблему — например, в течение этих 500 мс программа не сможет определить факт нажатия любой другой кнопки.

Так что иногда такое решение оказывается недостаточно хорошим, и требуется какое-то более сложное решение. Можно написать собственный код, устраняющий эффект дребезга контактов, но для многих это может оказаться слишком сложной задачей, к тому же нашлись замечательные парни, которые уже сделали всю работу.

Чтобы воспользоваться плодами их труда, нужно добавить библиотеку в приложение Arduino. Для этого загрузите библиотеку в виде zip-архива с сайта *http://www.arduino.cc/playground/Code/Bounce* (воспользовавшись ссылкой на репозиторий GitHub).

После загрузки файла Bounce2-master.zip его нужно установить в Arduino IDE. Для этого выберите пункт **Add .ZIP Library** (Добавить .ZIP библиотеку) в меню **Sketch** (Скетч), как показано на рис. 6.10.

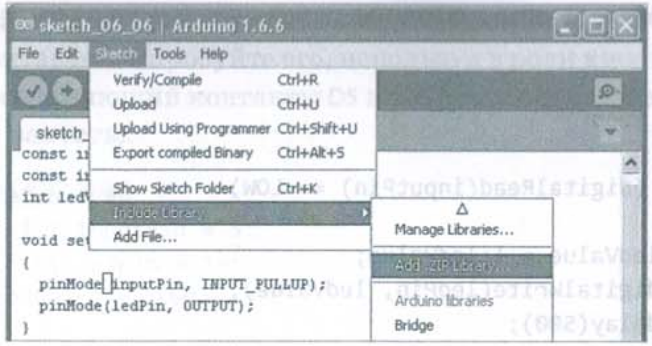


Рис. 6.10. Установка библиотеки Bounce

Скетч 6-07 показывает, как пользоваться библиотекой Bounce. Вы грузите его в плату и посмотрите, насколько надежным стало пере ключение светодиода:

```
//sketch 6-07
#include <Bounce2.h>

const int inputPin = 5;
const int ledPin = 13;

int ledValue = LOW;
Bounce bouncer = Bounce();

void setup()
{
  pinMode(inputPin, INPUT_PULLUP);
  pinMode(ledPin, OUTPUT);
  bouncer.attach(inputPin);
}

void loop()
{
  if (bouncer.update() && bouncer.read() == LOW)
  {
    ledValue = ! ledValue;
    digitalWrite(ledPin, ledValue);
  }
}
```


Пользоваться библиотекой очень просто. Для начала обратите внимание на эту строку:

```
#include <Bounce2.h>
```

Она нужна, чтобы сообщить компилятору, что скетч будет использовать библиотеку Bounce. Отметьте также строку

```
Bounce bouncer = Bounce();
```

Не будем сейчас углубляться в синтаксис этой строки — в действительности это синтаксис языка C++, а не C, и мы не будем больше сталкиваться с ним до главы 11. Пока вам достаточно знать, что эта строка настраивает объект bouncer.

В функции setup появилась новая строка, подключающая объект bouncer к контакту с номером inputPin вызовом функции attach. С этого момента, чтобы узнать состояние кнопки, вместо чтения цифрового входа следует использовать объект bouncer, который служит своего рода защитной оберткой вокруг контакта. Определение состояния выполняется в следующей строке:

```
if (bouncer.update() && bouncer.read() == LOW)
```

Функция update вернет истинное значение, если что-то изменилось в объекте bouncer, а вторая часть условия проверит, находится ли кнопка в состоянии LOW.

Аналоговые выходы

На некоторые цифровые контакты, а именно контакты с метками 3, 5, 6, 9, 10 и 11, может подаваться произвольное напряжение в диапазоне 0...5 В, а не только 0 или 5 В. Эти контакты отмечены значком ~ или надписью PWM рядом с их номерами. Аббревиатура PWM расшифровывается как Pulse Width Modulation (широтно-импульсная модуляция, ШИМ) и указывает, что выходная мощность на данном контакте регулируется. Достигается такая регулировка быстрым включением и выключением контакта.

Импульсы всегда следуют с одной и той же частотой (примерно 500 раз в секунду для всех контактов, кроме 5 и 6, для которых частота следования импульсов равна 980 раз в секунду), но длитель-

ность импульсов изменяется. Если вы решите использовать ШИМ-контакты для управления яркостью светодиода, знайте, что, когда следуют широкие импульсы, светодиод горит почти все время, а когда короткие импульсы, светодиод зажигается только на очень короткие промежутки времени. Импульсы следуют так часто, что даже если сказать наблюдателю, что светодиод мерцает, ему все равно будет казаться, что он светит ярче или тусклее.

Прежде чем проводить эксперименты со светодиодом, попробуйте поэкспериментировать с мультиметром. Подготовьте прибор для измерения напряжения между контактами GND и D3 (рис. 6.11).

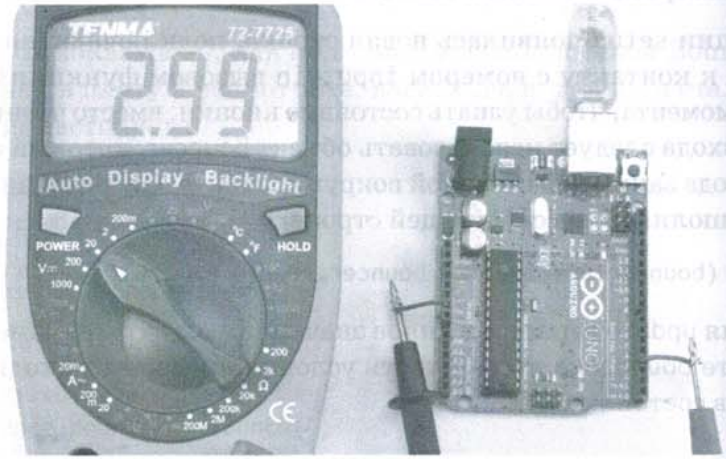


Рис. 6.11. Измерение напряжения на аналоговом выходе

Теперь выгрузите скетч 6-08 в плату и откройте монитор последовательного порта (рис. 6.12). Введите одну цифру 3 и нажмите клавишу Return. Вы должны увидеть, что вольтметр фиксирует напряжение, близкое к 3 В. Далее можете попробовать вводить любые другие числа от 0 до 5:

```
//sketch 6-08
const int outputPin = 3;

void setup()
{
  pinMode(outputPin, OUTPUT);
  Serial.begin(9600);
  Serial.println("Enter Volts 0 to 5");
}

void loop()
{
  if (Serial.available() > 0)
  {
    float volts = Serial.parseFloat();;
    int pwmValue = volts * 255.0 / 5.0;
    analogWrite(outputPin, pwmValue);
  }
}
```

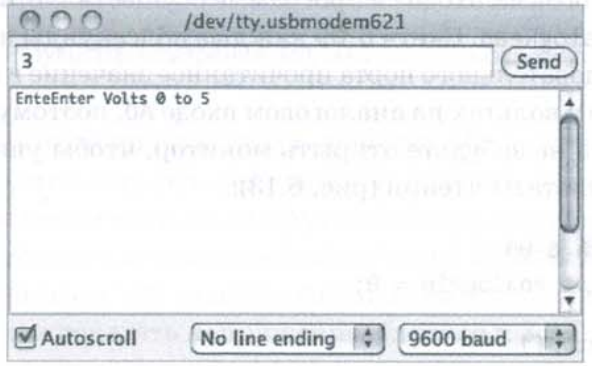



Рис. 6.12. Передача напряжения на аналоговый выход

Программа определяет значение ШИМ (PWM) для вывода как числа в диапазоне 0...255, умножая желаемое значение напряжения (от 0 до 5) на 255/5. (Полное описание принципа действия широтно-импульсной модуляции можно найти в «Википедии».)

Напряжение на аналоговый выход подается с помощью функции analogWrite, которая вместе с номером контакта требует передать ей значение от 0 до 255, определяющее напряжение, где 0 соответствует выключенному состоянию, а 255 — подаче полного напряжения. Широтно-импульсная модуляция является отличным способом управления яркостью свечения светодиода. Если попробовать управлять яркостью, изменяя напряжение, можно заметить, что пока напряжение не снизится до 2 В, ничего заметного со светодиодом не происходит, а при дальнейшем снижении напряжения светодиод быстро

теряет яркость и гаснет. Благодаря широтно-импульсной модуляции изменение светимости светодиода имеет более линейный характер.

Аналоговые входы

Цифровые входы просто дают ответ на вопрос: «Включено или выключено?» Аналоговые входы в отличие от цифровых возвращают значение в диапазоне 0...1023 в зависимости от величины напряжения, поданного на контакт аналогового входа.

Чтение аналоговых входов в программе осуществляется с помощью функции analogRead. Скетч 6-09 каждые полсекунды выводит в монитор последовательного порта прочитанное значение и фактическое напряжение в вольтах на аналоговом входе A0, поэтому при опробовании скетча не забудьте открыть монитор, чтобы увидеть, на что похожи результаты чтения (рис. 6.13):

```
//sketch 6-09
const int analogPin = 0;

void setup()
{
  Serial.begin(9600);
}

void loop()
{
  int reading = analogRead(analogPin);
  float voltage = reading / 204.6;
  Serial.print("Reading=");
  Serial.print(reading);
  Serial.print("\t\tVolts=");
  Serial.println(voltage);
  delay(500);
}
```

После запуска скетча можно заметить, что прочитанные значения время от времени немного изменяются. Как и в случае с цифровыми входами, это объясняется действием наведенных помех.

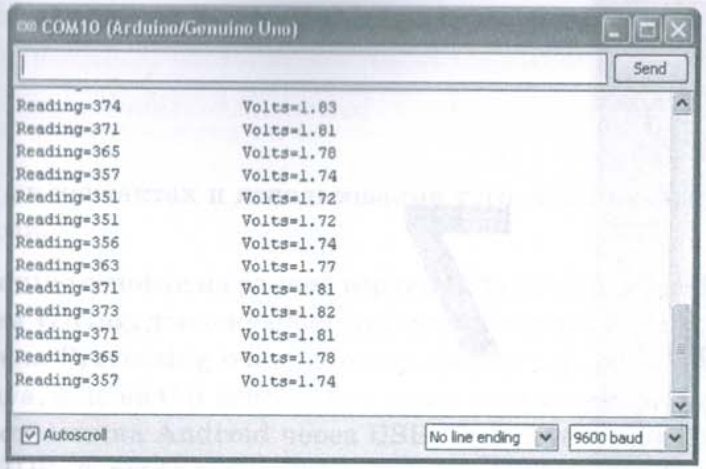


Рис. 6.13. Результаты измерения напряжения с помощью Arduino Uno

Соедините отрезком провода гнезда A0 и GND. После этого постоянно должны выводиться нулевые значения. Извлеките конец провода из гнезда GND, вставьте его в гнездо 5V, и вы должны увидеть значения, близкие к максимально возможному значению 1023. Если соединить контакт A0 с гнездом 3.3V на плате Arduino, виртуальный вольтметр Arduino должен показать напряжение, близкое к 3,3 В.

Значение 204.6 — это результат деления 1023 (максимальное значение, которое можно прочитать с аналогового входа) на 5 (максимальное напряжение). Функция Serial.print посылает сообщения в монитор последовательного порта без перехода на другую строку в отличие от функции Serial.println. Комбинация \t в сообщениях представляет символ табуляции и используется для выравнивания последовательности чисел по вертикали.

В заключение

Этой главой завершается знакомство с основами работы с сигналами в плате Arduino. В следующей главе мы рассмотрим некоторые дополнительные возможности, предоставляемые стандартной библиотекой Arduino.

7

Стандартная библиотека Arduino

Стандартная библиотека — это место, где со
браны очень полезные функции. Вы пока по
знакомились только с ядром языка C, но, кроме
этого, в скетчах вам понадобится масса вспо
могательных функций.

Вы уже встречались с некоторыми из них, та
кими как pinMode, digitalWrite и analogWrite.
Но их намного больше. Среди них есть функ
ции для математических вычислений, получе
ния случайных чисел, выполнения операций
с отдельными битами, регистрации импульсов
на входных контактах и использования того, что называется прерываниями.

Язык Arduino основан на ранней версии библиотеки, известной под названием Wiring, дополняющей другую библиотеку — Processing. Библиотека Processing очень похожа на Wiring, но написана на языке Java, а не на C и используется для соединения компьютера с устройствами на Android через USB. Фактически приложение Arduino IDE, которое вы запускаете в своем компьютере, основано на библиотеке Processing. Если у вас появится желание написать собственный интерфейс для компьютера, чтобы соединить его с платой Arduino, обратите внимание на библиотеку Processing (*www.processing.org*).

Случайные числа

Кто бы что ни говорил, но компьютеры очень предсказуемы. Однако иногда полезно внести в поведение Arduino элемент непредсказуемости. Например, у вас может появиться желание сконструировать робота, перемещающегося по случайному маршруту в комнате: двигающегося в выбранном направлении случайный промежуток времени, поворачивающего на случайный угол и повторяющего цикл снова. Или создать на основе Arduino игровой кубик, позволяющий получать случайные числа от одного до шести.

Стандартная библиотека Arduino включает необходимую для этого функцию. Она называется `random`. Функция `random` возвращает значения типа `int` и может принимать один или два аргумента. Если ей передать только один аргумент, она вернет случайное число в диапазоне от 0 до значения аргумента –1.

Версия с двумя аргументами возвращает случайные числа в диапазоне от первого аргумента (включительно) до второго минус едини-

ца. То есть вызов random(1, 10) вернет случайное число в диапазоне от 1 до 9.

Скетч 7-01 выводит в монитор последовательного порта числа от 1 до 6:

```
//sketch 7-01
void setup()
{
  Serial.begin(9600);
}

void loop()
{
  int number = random(1, 7);
  Serial.println(number);
  delay(500);
}
```

Если выгрузить скетч в плату Arduino и открыть монитор последовательного порта, можно увидеть картину, напоминающую рис. 7.1.

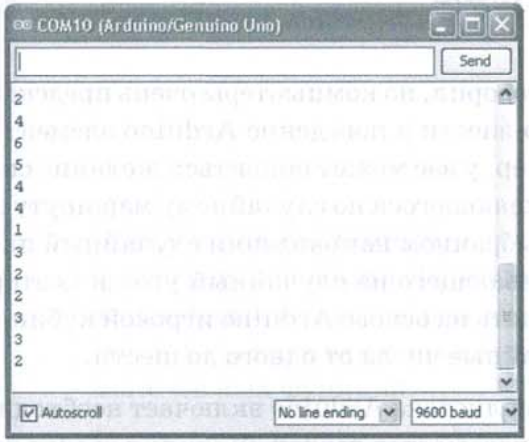


Рис. 7.1. Случайные числа

Если попробовать запустить скетч несколько раз подряд, вы, вероятно, будете удивлены тем, что каждый раз скетч генерирует одну и ту же последовательность «случайных» чисел.

В действительности эти числа не являются случайными — их часто называют *псевдослучайными*, потому что они имеют случайное распределение. То есть если запустить скетч и собрать миллион чисел, вы увидите примерно одинаковое количество единиц, двоек, троек и других чисел. Но числа не являются случайными в смысле их непредсказуемости. В действительности микроконтроллер не в состоянии генерировать по-настоящему случайные числа без вмешательства извне.

Вы можете организовать такое вмешательство, *инициализируя* генератор случайных чисел, чтобы сделать последовательность чисел менее предсказуемой. Но в этом случае можно задать лишь начальное значение для последовательности. Если задуматься, несложно прийти к выводу, что нельзя использовать random для инициализации генератора случайных чисел. На практике обычно используется трюк, основанный на том факте (обсуждавшемся в предыдущей главе), что «висящий в воздухе» аналоговый ввод подвержен внешним помехам. То есть значение, прочитанное с аналогового входа, можно использовать для инициализации генератора случайных чисел. Функция, делающая это, называется randomSeed. Скетч 7-02 демонстрирует, как добавить немного случайности в работу генератора случайных чисел:

```
//sketch 7-02
void setup()
{
  Serial.begin(9600);
  randomSeed(analogRead(0));
}

void loop()
{
  int number = random(1, 7);
  Serial.println(number);
  delay(500);
}
```

Попробуйте нажать кнопку **Reset** на плате несколько раз. Вы увидите, что каждый раз после этого генерируется иная последовательность случайных чисел.

Десятичные значения плохо подходят при работе с битами в уме. Очень сложно представить себе, какие биты установлены в десятичном числе, таком как 123. Поэтому программисты часто пользуются *шестнадцатеричной* системой счисления, то есть системой счисления с основанием 16. В этой системе счисления кроме десятичных цифр от 0 до 9 используются дополнительные шестнадцатеричные цифры от A до F. То есть каждая цифра представлена четырьмя битами. В следующей таблице приводятся соответствия между десятичными, шестнадцатеричными и двоичными представлениями чисел от 0 до 15:

Десятичное	Шестнадцатеричное	Двоичное
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Итак, в шестнадцатеричной системе счисления любое значение типа `int` можно представить числом из четырех цифр. Например, двоичное число 10001100 в шестнадцатеричной системе имеет вид 8C. В языке C предусмотрен специальный синтаксис для работы с шест

надцатеричными числами. Присвоить шестнадцатеричное число переменной типа `int` можно так:

```
int x = 0x8C;
```

Помимо шестнадцатеричной можно также использовать двоичную форму записи — с префиксом '0b'. Например, двоичное число, использованное в примере с шестнадцатеричным представлением `0x8C`, можно записать сразу в двоичной форме:

```
0b10001100
```

Стандартная библиотека Arduino включает несколько функций, позволяющих манипулировать каждым отдельным битом в переменных типа `int`. Функция `bitRead` возвращает значение конкретного бита в значении типа `int`. Так, в следующем примере переменной `bit` будет присвоено значение 0:

```
int x = 0b10001100;
int bit = bitRead(x, 0);
```

Во втором аргументе передается позиция интересующего бита, нумерация позиций начинается с 0 и продолжается до 15. Наименьший номер позиции имеет самый младший значащий бит. То есть самый правый бит имеет позицию 0, следующий за ним бит (в направлении справа налево) — позицию 1 и т. д.

Как можно догадаться, для функции `bitRead` имеется парная ей функция `bitWrite`, которая принимает три аргумента. Первый определяет исходное число, второй — позицию бита и третий — значение бита. Следующий пример изменяет значение 2 типа `int`, превращая его в 3 (в десятичном или шестнадцатеричном представлении):

```
int x = 0b10;
bitWrite(x, 0, 1);
```

Дополнительные функции ввода/вывода

В библиотеке есть несколько маленьких функций, упрощающих решение некоторых задач, связанных с вводом/выводом.

Такой способ получения случайных чисел нельзя использовать для розыгрыша лотереи. Чтобы получить их более качественную последовательность, необходима специальная аппаратура, которая опирается на случайные события, такие, например, как интенсивность космического излучения.

Математические функции

В скетчах для Arduino редко возникает необходимость в математических вычислениях сложнее элементарной арифметики. Если это потребуется, к вашим услугам имеются математические функции. В следующей таблице перечислены наиболее интересные из них.

Функция	Описание	Пример
abs	Возвращает абсолютное значение аргумента	abs(12) вернет 12 abs(-12) вернет 12
constraint	Ограничивает значения заданным диапазоном. Первый аргумент — ограничиваемое значение, второй — начало диапазона допустимых значений, третий — конец диапазона	constrain(8, 1, 10) вернет 8 constrain(11, 1, 10) вернет 10 constrain(0, 1, 10) вернет 1
map	Отображает число из одного диапазона в другой диапазон. Первый аргумент — число для отображения. Второй и третий аргументы определяют исходный диапазон, а последние два — конечный. Функция может пригодиться для отображения значений, получаемых с аналоговых входов	map(x, 0, 1023, 0, 5000)
max	Возвращает наибольший из двух аргументов	max(10, 11) вернет 11
min	Возвращает наименьший из двух аргументов	min(10, 11) вернет 10
pow	Возвращает первый аргумент, возведенный в степень, определяемую вторым аргументом	pow(2, 5) вернет 32
sqrt	Возвращает корень квадратный для аргумента	sqrt(16) вернет 4

Функция	Описание	Пример
sin, cos, tan	Тригонометрические функции. На практике используются редко	—
log	Может использоваться для вычисления температуры на основании значений, полученных от логарифмического терморезистора, например	—

Операции с битами

Бит — это единица измерения двоичной информации, он может принимать значение 0 или 1. Слово *bit* является сокращением от англ. *binary digit* — двоичная цифра. В большинстве случаев вы будете использовать переменные типа `int`, фактически состоящие из 16 бит.

Применять такие переменные для хранения логических значений истина/ложь (1 или 0) кажется расточительством. В действительности, если вы не испытываете нехватки памяти, такое расточительство является меньшим злом, чем создание сложного для понимания программного кода, выполняющего операции с битами. Но иногда бывает выгодно иметь возможность упаковывать информацию более плотно.

Каждый бит в переменной типа `int` имеет десятичное значение, и вы можете получить полное значение переменной, складывая десятичные значения, соответствующие битам, равным 1. Так, на рис. 7.2 показано двоичное представление десятичного числа 38. Ситуация немного осложняется при работе с отрицательными числами, но это происходит, только когда самый старший (самый левый) бит равен 1.

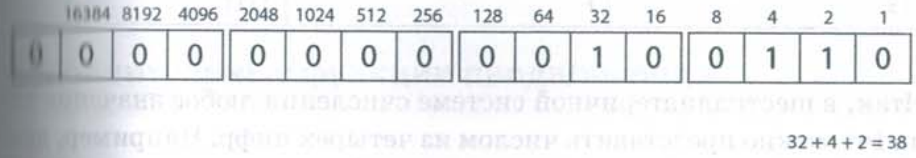


Рис. 7.2. Число типа int

Генерирование звуковых сигналов

С помощью функции tone можно сгенерировать сигнал прямоугольной формы (рис. 7.3) на одном из цифровых выходов. Часто эта возможность используется для воспроизведения звуковых сигналов с применением громкоговорителя или зуммера.

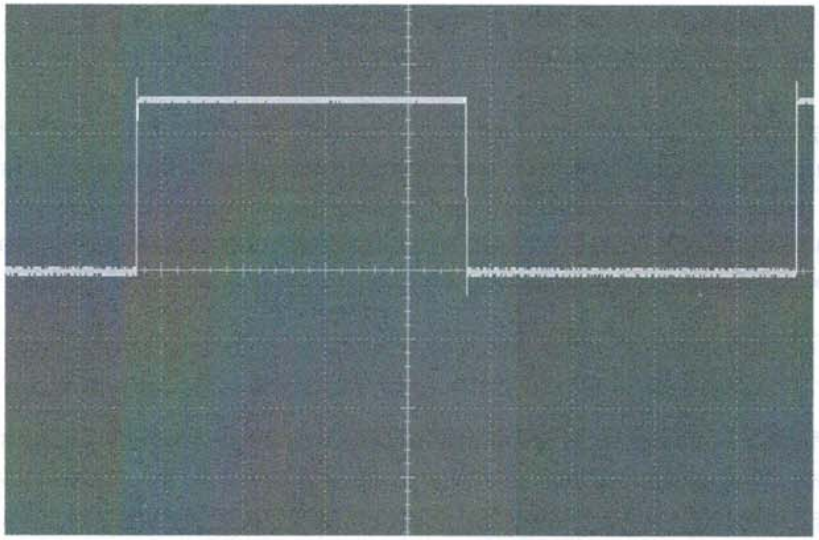


Рис. 7.3. Сигнал прямоугольной формы

Функция принимает два или три аргумента. Первый аргумент всегда определяет номер контакта, на котором должен генерироваться сигнал, второй — частоту сигнала в герцах [Гц] и третий, необязательный аргумент — продолжительность сигнала. Если продолжительность не указана, как в скетче 7-03, сигнал будет генерироваться неопределенно долго. Именно поэтому в данном примере вызов функции tone был помещен в функцию setup, а не loop:

```
//sketch 7-03
void setup()
{
  tone(4, 500);
}

void loop() {}
```


Прекратить воспроизведение сигнала можно с помощью функции noTone. Эта функция принимает единственный аргумент — номер контакта, на который выводится сигнал.

Применение сдвигового регистра

Для решения некоторых задач, например для управления большим числом светодиодов, количества выходов на плате Arduino Uno оказывается недостаточно. В таких случаях часто используют микросхему сдвигового регистра[1]. Эта микросхема читает данные по одному биту за раз и затем, когда их накапливается достаточное число, защелкивает все биты на множестве выходов (по одному на бит).

Чтобы помочь вам использовать этот прием, в библиотеке имеется вспомогательная функция shiftOut. Эта функция принимает четыре аргумента:

- номер контакта (выхода) на котором должен появиться определяемый бит;
- номер контакта, используемый в качестве тактового, он включается при передаче каждого бита;
- флаг, определяющий порядок передачи битов — начиная с младшего или со старшего значащего бита. В этом аргументе должна передаваться константа MSBFIRST или LSBFIRST;
- байт данных для передачи.

Прерывания

Программистов с опытом разработки крупных программ часто расстраивает неспособность Arduino решать сразу несколько задач. Если вы привыкли управлять большим числом потоков выполне-

[1] Очень интересную статью, демонстрирующую использование сдвигового регистра с платой Arduino, можно найти по адресу: http://arduino.ru/Tutorial/registr_74HC595. — *Примеч. пер.*

ния в своих программах, тогда вам не повезло. Некоторым удается решить эту проблему и организовать выполнение нескольких потоков, однако для задач, обычно решаемых с помощью Arduino, в этом нет большой необходимости. Наиболее близкий к применяемому на Arduino способу организации параллельного выполнения — использование прерываний.

Два контакта на плате Arduino (D2 и D3) могут генерировать соответствующие им прерывания. То есть если эти контакты действуют как входы и на них подается сигнал определенным способом, процессор Arduino отложит выполнение основной программы и выполнит функцию, связанную с прерыванием.

Скетч 7-04 заставляет мигать светодиод и при получении прерывания изменяет частоту мигания. Имитировать прерывание можно, соединяя отрезком провода контакты D2 и GND, а чтобы обеспечить постоянный высокий уровень напряжения на контакте D2 в течение остального времени, используется внутренний нагрузочный резистор:

```
//sketch 7-04
const int interruptPin = 2;
const int ledPin = 13;
int period = 500;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(interruptPin, INPUT_PULLUP);
  attachInterrupt(0, goFast, FALLING);
}

void loop()
{
  digitalWrite(ledPin, HIGH);
  delay(period);
  digitalWrite(ledPin, LOW);
  delay(period);
}
```



```
void goFast()
{
  period = 100;
}
```

Ключевой здесь является следующая строка в функции setup:

```
attachInterrupt(0, goFast, FALLING);
```

Первый аргумент определяет, какое из двух прерываний будет использоваться. Хоть это и странно, но значение 0 соответствует контакту 2, а значение 1 — контакту 3.

Следующий аргумент — имя функции, которая должна быть вызвана при получении прерывания, и, наконец, третий аргумент — это константа, одна из следующих: CHANGE (изменение), RISING (положительный перепад), FALLING (отрицательный перепад). Возможные варианты изображены на рис. 7.4.

Рис. 7.4. Типы сигналов прерывания

Если настроен режим прерывания по изменению уровня сигнала (CHANGE), прерывания будут генерироваться и по положительному (RISING, с 0 на 1), и по отрицательному (FALLING, с 1 на 0) перепаду.

Запретить прерывания можно вызовом функции noInterrupts. Она запрещает все прерывания по обоим каналам. Возобновить работу прерываний можно вызовом функции interrupts.

В заключение

В этой главе вы познакомились с некоторыми удобствами, предоставляемыми стандартной библиотекой Arduino. Они помогают экономить время и силы, и если есть что-то, что любят хорошие программисты, так это качественная работа, выполненная другими людьми.

В следующей главе мы продолжим изучение структур данных, начатое в главе 5, и посмотрим, как сохранять данные, чтобы они не терялись при выключении питания платы Arduino.

8

Запись данных

Когда скетч присваивает переменной некоторое значение, оно хранится в оперативной памяти до тех пор, пока не будет выключено питание платы Arduino. В тот момент, когда выключается питание или нажимается кнопка сброса на плате, все данные, находящиеся в оперативной памяти, теряются.

В этой главе мы рассмотрим некоторые способы сохранения данных.

Константы

Если данные, которые нужно сохранить, не изменяются, их можно восстанавливать при каждом запуске Arduino. Примером такого подхода может служить массив `letters` в примере транслятора азбуки Морзе из главы 5 (скетч 5-05).

Для определения переменной-массива требуемого размера и его заполнения в нем использовался следующий код:

```
char* letters[] = {
  ".-", "-...", "-.-.", "-..", ".",
  "..-.", "--.", "....", "..", // A-I
  ".---", "-.-", ".-..", "--", "-.",
  "---", ".--.", "--.-", ".-.", // J-R
  "...", "-", "..-", "...-", ".--",
  "-..-", "-.--", "--.." // S-Z
};
```

Возможно, вы помните — тогда мы решили, что скромных 2 КБ нам более чем достаточно, чтобы не задумываться об экономии. Однако, если оперативная память нужна для чего-то другого, лучше было бы хранить эти данные во флеш-памяти программ, размер которой на плате 32 КБ, а не в 2 КБ ОЗУ. Тем более что для этого есть все возможности — библиотечная директива `PROGMEN`, хотя она и не очень удобна в использовании.

Сохранение данных во флеш-памяти

Чтобы получить возможность сохранять данные во флеш-памяти, необходимо подключить библиотеку pgmspace, как показано далее:

```
#include <avr/pgmspace.h>
```


Эта команда сообщает компилятору, что скетч будет использовать библиотеку pgmspace. Эта библиотека содержит множество функций, написанных другими программистами, которые вы можете использовать в своих скетчах, особо не вдаваясь в детали их работы.

После подключения библиотеки появляется возможность использовать в скетче ключевое слово `PROGMEM` и функцию `pgm_read_word`. Порядок их использования демонстрируется в дальнейшем.

Эта библиотека входит в состав программного обеспечения Arduino и является официально поддерживаемой библиотекой Arduino. Существует отличная коллекция таких официальных библиотек. Кроме того, в Интернете можно найти множество неофициальных библиотек, разрабатываемых такими же людьми, как и вы, и передающих их для общего пользования. Такие неофициальные библиотеки необходимо устанавливать в окружение Arduino отдельно. Мы поближе познакомимся с этими библиотеками в главе 11, где рассказывается также, как писать собственные библиотеки.

Совместно с директивой `PROGMEM` можно использовать только специальные, `PROGMEM`-совместимые типы данных. К сожалению, среди них *нет* массивов, хранящих массивы символов переменной длины. Однако в их число входят массивы, хранящие массивы символов фиксированной длины. Полный текст программы, иллюстрирующей обсуждаемую тему, почти в точности повторяет скетч 5-05 из главы 5. Вы можете открыть скетч 8-01 у себя в IDE и отмечать описываемые далее отличия.

Во-первых, появилась новая константа `maxLen`, определяющая максимальную длину последовательности точек и тире, соответствующей одному символу, плюс один для учета нулевого символа в конце.

Структура данных для букв теперь выглядит иначе:

```
PROGMEM const char letters[26][maxLen] = {
  ".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....",
"..", // A-I
  ".---", "-.-", ".-..", "--", "-.", "---", ".--.", "--.-",
".-.", // J-R
  "...", "-", "..-", "...-", ".--", "-..-", "-.--", "--.." //
S-Z
};
```

Ключевое слово PROGMEM указывает, что структура должна храниться во флеш-памяти. В памяти этого вида можно хранить только такие константы, как эта структура; после определения содержимое структуры нельзя будет изменить, поэтому здесь использован спецификатор const. Размер массива также жестко зафиксирован — он может хранить 26 кодов букв по maxLen (минус 1) точек и тире в каждом.

Функция loop тоже немного изменилась, в сравнении со скетчем 5-05.

```
void loop()
{
  char ch;
  char sequence[maxLen];
  if (Serial.available() > 0)
  {
    ch = Serial.read();
    if (ch >= 'a' && ch <= 'z')
    {
      memcpy_P(&sequence, letters[ch - 'a'], maxLen);
      flashSequence(sequence);
    }
    else if (ch >= 'A' && ch <= 'Z')
    {
      memcpy_P(&sequence, letters[ch - 'A'], maxLen);
      flashSequence(sequence);
    }
    else if (ch >= '0' && ch <= '9')
    {
      memcpy_P(&sequence, numbers[ch - '0'], maxLen);
      flashSequence(sequence);
    }
    else if (ch == ' ')
    {
      delay(dotDelay * 4); // интервал между словами
    }
  }
}
```

Структура letters выглядит как обычный массив строк, но в действительности она хранится во флеш-памяти и доступна только с применением специальной функции memcp_P, которая копирует данные из флеш-памяти в массив символов sequence с размером maxLen.

Символ & перед именем sequence позволяет функции memcpy_P изменять данные внутри массива символов sequence.

Я не привожу здесь скетч 8-01 целиком, так как он слишком длинный, но вы можете загрузить его и убедиться, что он действует в точности так же, как и его версия, использующая ОЗУ.

Кроме того что данные должны создаваться по-особому, их и читать приходится по-особому. Программный код, извлекающий из массива код Морзе для буквы, теперь должен выглядеть так:

```
strcpy_P(buffer, (char*)pgm_read_word(&(letters[ch - 'a'])));
```

Здесь используется переменная buffer, куда копируется PROGMEM-строка, чтобы ее можно было использовать как обычный массив символов. Эта переменная должна быть объявлена как глобальная:

```
char buffer[6];
```

Такой прием подходит, только когда данные не изменяются, то есть вы не собираетесь изменять их во время выполнения скетча. В следующем разделе вы познакомитесь с *электрически стираемой программируемой постоянной памятью (ЭСППЗУ — Electrically Erasable Programmable Read-Only Memory, EEPROM)*, которая предназначена для длительного хранения данных, которые могут изменяться.

Отдельные строки, используемые, например, для вывода сообщений в окно монитора порта, можно сохранить во флеш-память, используя очень удобную функцию, имеющуюся в Arduino C. Каждую такую строку достаточно заключить в вызов F():

```
Serial.println(F("Hello World"));
```

и она будет сохранена во флеш-память, а не в ОЗУ.

ЭСППЗУ

Микроконтроллер ATMega328, являющийся сердцем Arduino Uno, имеет 1 Кбайт *электрически стираемого программируемого посто-*

янного запоминающего устройства (ЭСППЗУ). Технически ЭСППЗУ может хранить данные годами. Несмотря на свое название, эта память не является постоянной в полном смысле этого слова. В нее можно записывать данные.

Официальные команды Arduino для чтения и записи в ЭСППЗУ столь же неудобны, как и аналогичные команды при использовании директивы PROGMEM. Читать и записывать данные в ЭСППЗУ можно только по одному байту.

Скетч 8-02 читает один символ из монитора последовательного порта, записывает его в ЭСППЗУ и выводит обратно на монитор:

```
// sketch 8-02
#include <EEPROM.h>

int addr = 0;
char ch;

void setup()
{
  Serial.begin(9600);
  ch = EEPROM.read(addr);
}

void loop()
{
  if (Serial.available() > 0)
  {
    ch = Serial.read();
    EEPROM.write(0, ch);
    Serial.println(ch);
  }
  Serial.println(ch);
  delay(1000);
}
```

Чтобы опробовать этот скетч, откройте монитор последовательного порта и введите символ. Затем выключите Arduino и включите снова. Когда вы повторно откроете монитор, то увидите, что плата запомнила символ.

Функция EEPROM.write принимает два аргумента. Первый — адрес ячейки в ЭСППЗУ, который может быть числом в диапазоне 0...1023. Второй аргумент — данные для записи в эту ячейку. Записать в ячейку можно только один байт. Символ как раз и представлен одним байтом (8 бит), поэтому с символами не возникает никаких проблем, но 2-байтное (16 бит) число типа int нельзя записать напрямую.

Запись значений int в ЭСППЗУ

Чтобы записать 2-байтное значение int в ячейки 0 и 1 ЭСППЗУ, можно сделать следующее:

```
int x = 1234;
EEPROM.write(0, highByte(x));
EEPROM.write(1, lowByte(x));
```

Функции highByte и lowByte позволяют разделить значение типа int на два байта. На рис. 8.1 показано, как фактически значение int записывается в ЭСППЗУ.

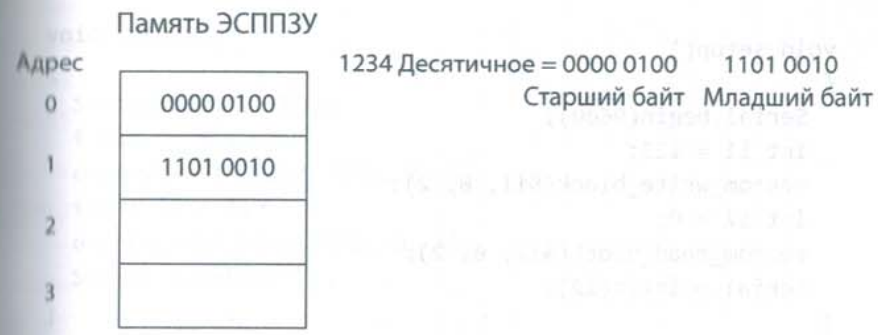


Рис. 8.1. Запись 16-битного целого числа в ЭСППЗУ

Чтобы прочитать значение int обратно из ЭСППЗУ, нужно прочитать два байта и реконструировать из них значение int:

```
byte high = EEPROM.read(0);
byte low = EEPROM.read(1);
int x = (high << 8) + low;
```

Оператор << — это оператор поразрядного сдвига, который в данном случае сдвигает 8 бит в старший байт числа int, и затем к результату добавляются 8 бит из младшего байта.

Использование библиотеки AVR EEPROM

Документированный способ записи в ЭСППЗУ отлично подходит для записи отдельных байтов. Но, как было показано на примере значений типа int, он не годится для типов данных большего размера. Еще хуже обстоит дело с числами типа float (4 байта). К счастью, есть альтернативный метод, основанный на библиотеке AVR EEPROM, которую использует сама плата Arduino. С ее помощью можно прочитать или записать данные в ЭСППЗУ всего одной командой.

В скетче 8-03 демонстрируется пример использования этой библиотеки для записи и последующего чтения числа типа int.

```
// sketch 08-03

#include <avr/eeprom.h>

void setup()
{
  Serial.begin(9600);
  int i1 = 123;
  eeprom_write_block(&i1, 0, 2);
  int i2 = 0;
  eeprom_read_block(&i2, 0, 2);
  Serial.println(i2);
}
void loop()
{
}
```

Библиотека уже входит в состав Arduino IDE, поэтому ее не нужно устанавливать отдельно. Функция, выполняющая запись в ЭСППЗУ, называется eeprom_write_block, и, как можно догадаться по ее имени, она записывает в ЭСППЗУ блок содержимого памяти. В первом

параметре она принимает ссылку на переменную. В данном случае на переменную i1, которой предварительно присваивается значение 123. Перед именем i1 добавлен символ &, потому что функция ожидает получить адрес переменной в памяти, а не ее значение. Во втором параметре принимается адрес первого байта в ЭСППЗУ, куда должен быть записан блок, и в последнем параметре – число байт для записи (2 — для числа типа int).

Чтение значения из ЭСППЗУ в ОЗУ (в переменную i2) в точности отражает процедуру записи и выполняется с теми же параметрами.

Запись значений float в ЭСППЗУ

Запись значения типа float в ЭСППЗУ с помощью библиотеки AVR EEPROM выполняется аналогично записи значения типа int, как можно видеть в скетче 8-04.

```
// sketch 08-04

#include <avr/eeprom.h>

void setup()
{
  Serial.begin(9600);
  float f1 = 1.23;
  eeprom_write_block(&f1, 0, 4);
  float f2 = 0;
  eeprom_read_block(&f2, 0, 4);
  Serial.println(f2);
}

void loop()
{
}
```

Основное отличие заключается в последнем параметре функций eeprom_write_block и eeprom_read_block — на этот раз в нем передается число 4 (4 байта), а не 2.

Запись строки в ЭСППЗУ

Запись строк в ЭСППЗУ и чтение их оттуда также лучше осуществлять с помощью библиотеки AVR EEPROM. Скетч 8-05 демонстрирует это на примере чтения и записи пароля в ЭСППЗУ. Сначала он читает пароль из ЭСППЗУ и выводит его в монитор порта, а затем предлагает ввести новый пароль (рис. 8.2). После записи нового пароля вы можете выключить плату Arduino, затем включить ее и, открыв окно монитора порта, убедиться, что новый пароль сохранился.

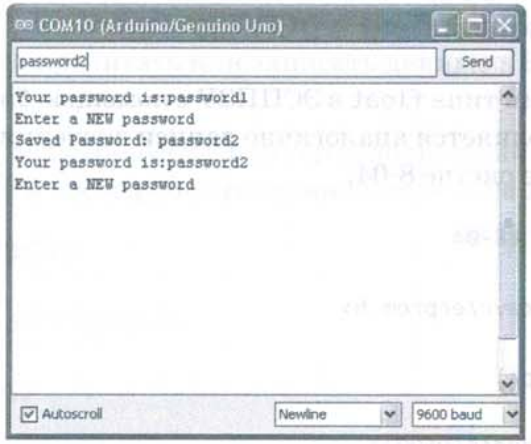


Рис. 8.2. Новый пароль сохранился

```
// sketch 08-05

#include <avr/eeprom.h>

const int maxPasswordSize = 20;
char password[maxPasswordSize];

void setup()
{
  eeprom_read_block(&password, 0, maxPasswordSize);
  Serial.begin(9600);
}
```



```
void loop()
{
  Serial.print("Your password is:");
  Serial.println(password);
  Serial.println("Enter a NEW password");
  while (!Serial.available()) {};
  int n = Serial.readBytesUntil('\n', password,
    maxPasswordSize);
  password[n] = '\0';
  eeprom_write_block(password, 0, maxPasswordSize);
  Serial.print("Saved Password: ");
  Serial.println(password);
}
```

Массив символов password имеет фиксированный размер, равный 20 символам, включая признак конца строки '\0'. Функция setup читает содержимое ЭСППЗУ в массив password, начиная с ячейки с номером 0.

Функция loop выводит необходимые сообщения и затем выполняет пустой цикл while, проверяя получение данных из монитора порта с помощью Serial.available, которая должна вернуть значение больше 0. Затем функция readBytesUntil читает символы из монитора порта, пока не встретит признак конца строки — символ '\n'. Прочитанные символы сохраняются непосредственно в массив password.

Так как заранее неизвестно, какой длины пароль будет введен, число прочитанных символов сохраняется в переменной n, и затем в элемент массива password с порядковым номером n записывается нулевой символ '\0', чтобы отметить конец строки. Наконец, новый пароль выводится в окно монитора порта, чтобы подтвердить изменение.

Очистка ЭСППЗУ

Записывая данные в ЭСППЗУ, помните, что даже выгрузка нового скетча в плату не очищает его, из-за чего в постоянной памяти могут остаться данные от предыдущего проекта. Скетч 8-06 стирает содержимое ЭСППЗУ, заполняя его нулями:

```
// sketch 8-06
#include <EEPROM.h>

void setup()
{
  Serial.begin(9600);
  Serial.println("Clearing EEPROM");
  for (int i = 0; i < 1023; i++)
  {
    EEPROM.write(i, 0);
  }
  Serial.println("EEPROM Cleared");
}

void loop()
{
}
```

Также имейте в виду, что каждая ячейка в ЭСППЗУ может выдержать только 100 000 циклов записи, после чего надежность хранения данных не гарантируется. Поэтому записывайте данные в ЭСППЗУ, только когда это действительно необходимо. Кроме того, ЭСППЗУ действует довольно медленно — запись одного байта занимает около 3 мс.

Сжатие

Иногда требуется записать в ЭСППЗУ или с помощью PROGMEM больше данных, чем может там уместиться. В таких случаях следует поискать более эффективный формат представления данных.

Сжатие диапазона

В программе могут использоваться значения, которые выглядят как значения типа int или float, первое из которых занимает 16 бит, а второе — 32 бита. Например, для представления температуры в градусах Цельсия может использоваться значение типа float, такое как

20,25. При записи этого значения в ЭСППЗУ было бы неплохо иметь возможность записать его в виде единственного байта, в этом случае можно было бы добиться двукратной экономии памяти по сравнению с записью значения float.

Одно из решений состоит в том, чтобы изменить данные перед записью. Напомню, что в одном байте можно хранить положительные числа в диапазоне 0...255. То есть если достаточно сохранить целую часть градусов Цельсия, можно просто преобразовать значение float в значение типа int и отбросить дробную часть, как показано далее:

```
int tempInt = (int)tempFloat;
```

Переменная tempFloat хранит вещественное значение. Команда (int) называется *приведением* типа и используется для преобразования значения из одного типа в другой, совместимый тип. В данном случае операция приведения типа преобразует значение типа float, например 20,25, в значение типа int 20 простым отсечением дробной части.

Если известно, что максимальная температура не превысит 60 °C, а минимальная не упадет ниже 0 °C, тогда перед записью каждое значение температуры можно умножить на 4 и преобразовать в значение типа byte. В этом случае при чтении из ЭСППЗУ прочитанное значение нужно разделить на 4, чтобы получить вещественное значение температуры с точностью до 0,25 градуса.

Скетч 8-07 сохраняет такую температуру в ЭСППЗУ, затем читает ее и выводит в монитор последовательного порта для контроля:

```
// sketch 8-07
#include <EEPROM.h>

void setup()
{
  float tempFloat = 20.75;
  byte tempByte = (int)(tempFloat * 4);
  EEPROM.write(0, tempByte);

  byte tempByte2 = EEPROM.read(0);
  float temp2 = (float)(tempByte2) / 4;
  Serial.begin(9600);
```

```
    Serial.println("\n\n\n");
    Serial.println(temp2);
  }

  void loop(){}
```

Существуют и другие приемы сжатия данных. Например, если требуется организовать запись данных, изменяющихся во времени с невысокой скоростью, отличным примером может служить изменение температуры окружающего воздуха. Тогда первое значение в последовательности можно записать в полном формате, а в виде последующих значений записывать только отклонения от первого значения. Такие отклонения будут невелики, и для их хранения потребуется меньше памяти.

В заключение

Теперь вы знаете, как сохранить данные, чтобы они не пропали после выключения питания. В следующей главе мы поговорим о дисплеях.

9

Дисплеи

В этой главе мы рассмотрим, как управлять дисплеями. На рис. 9.1 показаны два типа дисплеев, которые мы будем использовать. Первый — алфавитно-цифровой жидкокристаллический индикатор (ЖКИ). Второй — графический дисплей с размером 128 × 64 пикселей на органических светодиодах (Organic Light-Emitting Diode, OLED). Эти два типа дисплеев пользуются большой популярностью в мире Arduino.

В этой книге рассказывается о программном, а не аппаратном обеспечении, но в данной главе больший упор делается на то, как работает электроника таких дисплеев, чтобы вам было понятнее, как управлять ими.

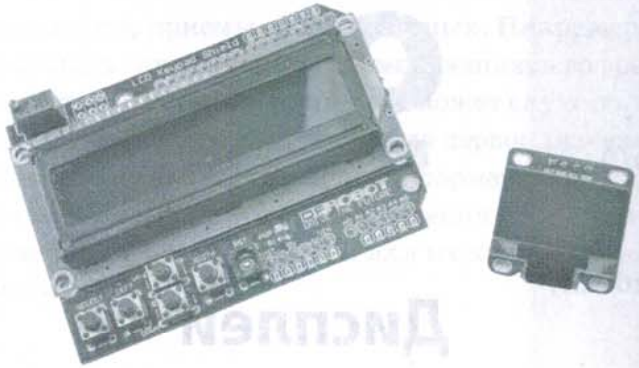


Рис. 9.1. Алфавитно-цифровой ЖК-дисплей (*слева*) и OLED-дисплей (*справа*)

Алфавитно-цифровые ЖК-дисплеи

Модуль ЖК-дисплея, о котором здесь рассказывается, сконструирован как плата расширения, которая может подключаться непосредственно к плате Arduino. Помимо самого дисплея на плате расширения имеется несколько кнопок. Существует множество разновидностей подобных плат расширения, но почти все они основаны на одной и той же микросхеме контроллера ЖК-дисплея (HD44780), поэтому далее будем рассматривать плату расширения на этом контроллере.

Я использую модель DFRobot LCD Keypad Shield for Arduino. Эта плата производится компанией DFRobot (*www.robotshop.com*), стоит относительно недорого и имеет ЖК-дисплей, отображающий 2 строки по 16 символов в каждой, а также шесть кнопок. Плата продается в собранном виде, поэтому вам не придется орудовать паяльником — ее можно просто воткнуть сверху в плату Arduino (рис. 9.2).

ЖК-дисплей занимает семь цифровых контактов на плате Arduino, посредством которых осуществляется управление им, и один аналоговый контакт для кнопок. Соответственно, данные контакты нельзя будет использовать для других целей.

Рис. 9.2. Плата ЖК-дисплея, подключенная к плате Arduino

USB-панель сообщений

Чтобы показать, насколько просто работать с дисплеем, мы создадим USB-панель сообщений. Она будет отображать сообщения, введенные в мониторе последовательного порта.

В состав среды разработки Arduino IDE входит библиотека LCD. Она существенно упрощает работу с ЖК-дисплеями. В библиотеке имеются следующие полезные функции:

- `clear` — очищает дисплей;
- `setCursor` — устанавливает текущую позицию в заданную позицию в заданной строке, куда будет выводиться последующий текст;

- print — выводит строку в текущую позицию. Исходный код примера находится в скетче 9-01:

```
// sketch 9-01 USB Message Board
#include <LiquidCrystal.h>

// lcd(RS, E, D4, D5, D6, D7)
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);
int numRows = 2;
int numCols = 16;

void setup()
{
  Serial.begin(9600);
  lcd.begin(numRows, numCols);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Arduino");
  lcd.setCursor(0,1);
  lcd.print("Rules");
}

void loop()
{
  if (Serial.available() > 0)
  {
    char ch = Serial.read();
    if (ch == '#')
    {
      lcd.clear();
    }
    else if (ch == '/')
    {
      // новая строка
      lcd.setCursor(0, 1);
    }
    else
    {
      lcd.write(ch);
    }
  }
}
```


Как и при использовании любых библиотек Arduino, сначала нужно подключить библиотеку, чтобы известить компилятор о том, какая библиотека будет использоваться.

Следующая строка определяет, какие контакты на плате Arduino будут задействованы платой расширения. Если вы используете другую плату расширения, возможно, она задействует другие контакты, поэтому обязательно загляните в документацию к плате.

В данном случае для управления дисплеем используются следующие шесть контактов: D4, D5, D6, D7, D8 и D9. Назначение каждого из них описывается в табл. 9.1.

Таблица 9.1. Назначение контактов платы ЖК-дисплея

Параметр функции LCD()	Контакт на плате Arduino	Назначение
RS	8	Регистр выбора (Register Select, RS). Устанавливается в значение 1 или 0 в зависимости от того, какие данные посылает плата Arduino — символы или инструкции. Инструкции, например, могут заставить курсор мигать
E	9	Готов (Enable). Переключается, чтобы сообщить контроллеру ЖК-дисплея, что данные на следующих четырех контактах готовы к чтению
Data 4	4	Эти четыре контакта используются для передачи данных. Контроллер ЖК-дисплея может принимать 8- или 4-битные данные. Данная плата расширения принимает 4-битные данные, в данном случае используются биты 4–7 вместо 0–7
Data 5	5	
Data 6	6	
Data 7	7	

Функция setup в этом примере проста и понятна. Она инициализирует обмен с монитором последовательного порта, чтобы дать возможность передавать команды, и инициализирует библиотеку LCD размерами используемого дисплея. Также она выводит сообщение «Arduino Rules» («Arduino рулит») в две строки, устанавливая курсор в начало первой строки перед выводом «Arduino», а затем в начало второй строки перед выводом «Rules».

Бóльшая часть операций выполняется в функции loop, которая проверяет поступление очередного символа от монитора последовательного порта. Скетч обрабатывает символы по одному.

Кроме обычных символов, которые скетч просто отображает, поддерживается также пара специальных символов. Если получен символ #, скетч очищает дисплей, а если получен символ /, он перемещает текущую позицию вывода в начало второй строки. Все остальные символы просто выводятся в текущую позицию вызовом функции write. Функция write действует подобно функции print, но выводит единственный символ, а не строку.

Использование дисплея

Опробуйте скетч 9-01, выгрузив его в плату, а затем подключив к ней плату расширения. Имейте в виду, что перед подключением или отключением плат расширения всегда необходимо снимать напряжение питания с платы Arduino.

Откройте монитор последовательного порта и попробуйте ввести текст, как показано на рис. 9.3.

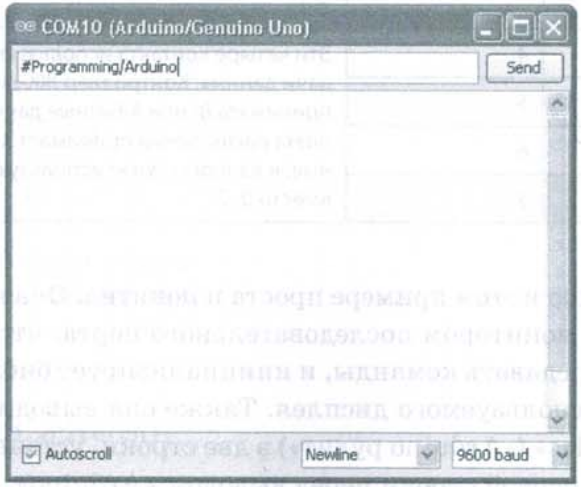


Рис. 9.3. Отправка команд дисплею

Другие функции из библиотеки LCD

Помимо функций, использовавшихся в примере, приведенном ранее, в библиотеке LCD имеется еще множество функций:

- home действует подобно вызову setCursor(0,0) — перемещает курсор в начало первой строки;
- cursor отображает курсор;
- noCursor скрывает курсор;
- blink заставляет курсор мигать;
- noBlink останавливает мигание курсора;
- noDisplay гасит дисплей, не удаляя содержимого;
- display включает дисплей после вызова noDisplay;
- scrollDisplayLeft смещает текст на дисплее на один символ влево;
- scrollDisplayRight смещает текст на дисплее на один символ вправо;
- autoscroll включает режим автопрокрутки, в котором новые символы добавляются в позицию курсора, а имеющийся на дисплее текст смещается в направлении, которое было определено вызовом функции leftToRight или rightToLeft;
- noAutoscroll выключает режим автопрокрутки.

Графические OLED-дисплеи

Дисплеи на органических светодиодах (Organic Light-Emitting Diode, OLED) отличаются высокой яркостью и четкостью и в настоящее время быстро вытесняют ЖК-дисплеи в бытовых приборах. Описываемый здесь OLED-дисплей подключается к Arduino посредством шины I2C и управляется микросхемой SD1306. Его можно приобрести во многих интернет-магазинах, таких как eBay или Adafruit.

Ищите дисплей с 4-контактным разъемом, так как он проще всего в использовании.

На рис. 9.4 изображена плата Arduino Uno с подключенным 0,96-дюймовым OLED-дисплеем. Эти дисплеи имеют разрешение 128 × 64 пикселей и светятся одним цветом, в данном случае синим.

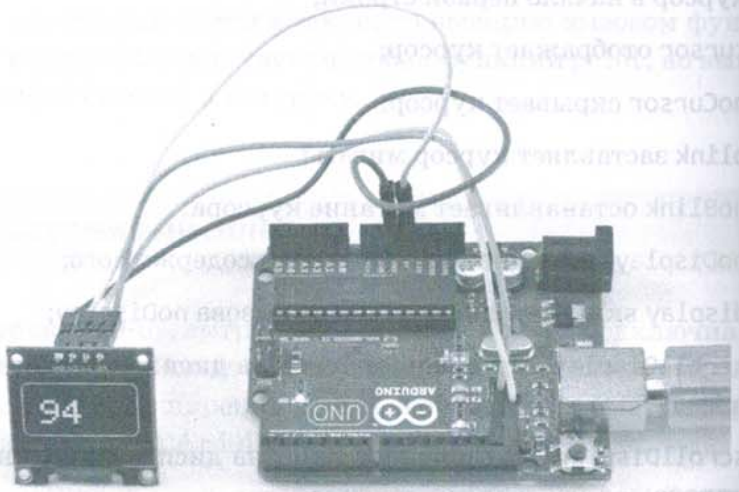


Рис. 9.4. Плата Arduino Uno с подключенным OLED-дисплеем

Подключение OLED-дисплея

Для подключения OLED-дисплея к плате Arduino необходим 4-контактный разъем с проводами. Такой разъем можно купить почти везде, например в интернет-магазине Adafruit (код товара: 825). Для подключения вам придется выполнить следующие соединения:

- контакт **GND** дисплея соединить с контактом **GND** на плате Arduino;
- контакт **VCC** дисплея соединить с контактом **5V** на плате Arduino;
- контакт **SCL** дисплея соединить с контактом **SCL** на плате Arduino. Этот контакт на плате Arduino Uno не подписан, но вы найдете его, руководствуясь рис. 9.5;

- контакт **SDA** дисплея соединить с контактом **SDA** на плате Arduino Arduino (также руководствуйтесь рис. 9.5).

I2C (произносится как I квадрат C) — это стандарт организации последовательной шины, широко используемой для подключения датчиков и дисплеев к микроконтроллерам, в том числе и к Arduino. Помимо контактов **GND** и питания шина использует контакты **SDA** (линия передачи данных) и **SCK** (линия тактовых импульсов) для взаимодействия с микроконтроллером и передачи данных по одному биту.

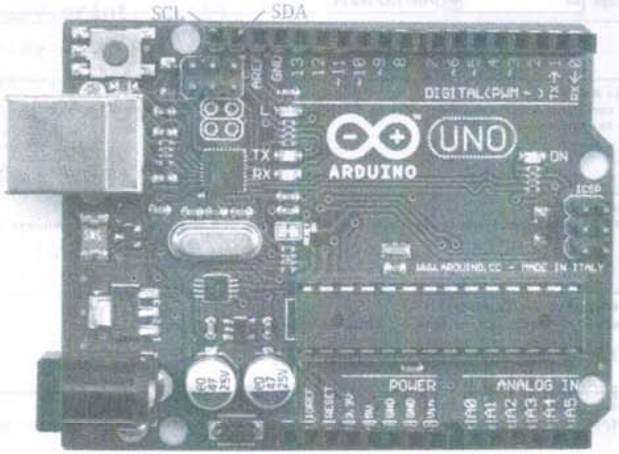


Рис. 9.5. Контакты SCL и SDA на плате Arduino Uno

Программное обеспечение

Скетч 9-02 считает секунды до 9999, затем сбрасывает показания в 0 и начинает счет сначала. Прежде чем выгрузить скетч в плату Arduino, определите I2C-адрес своего дисплея. Это шестнадцатеричное число, которое написано на обратной стороне OLED-дисплея. Многие недорогие OLED-дисплеи, продаваемые на eBay, используют адрес 0x3c.

Кроме того, чтобы успешно скомпилировать скетч, необходимо установить несколько библиотек. Выполнить установку можно с помощью

диспетчера библиотек в Arduino IDE. Откройте окно диспетчера библиотек, выбрав пункт меню **Sketch ▸ Include Library ▸ Manage Libraries...** (Скетч ▸ Подключить библиотеку ▸ Управлять библиотеками...). В открывшемся окне найдите пункт с библиотекой «Adafruit GFX Library» и щелкните на кнопке **Install** (Установка), как показано на рис. 9.6. Точно так же установите библиотеку «Adafruit SSD1306». Библиотеки SPI и Wire, также необходимые скетчу, уже входят в состав Arduino IDE.

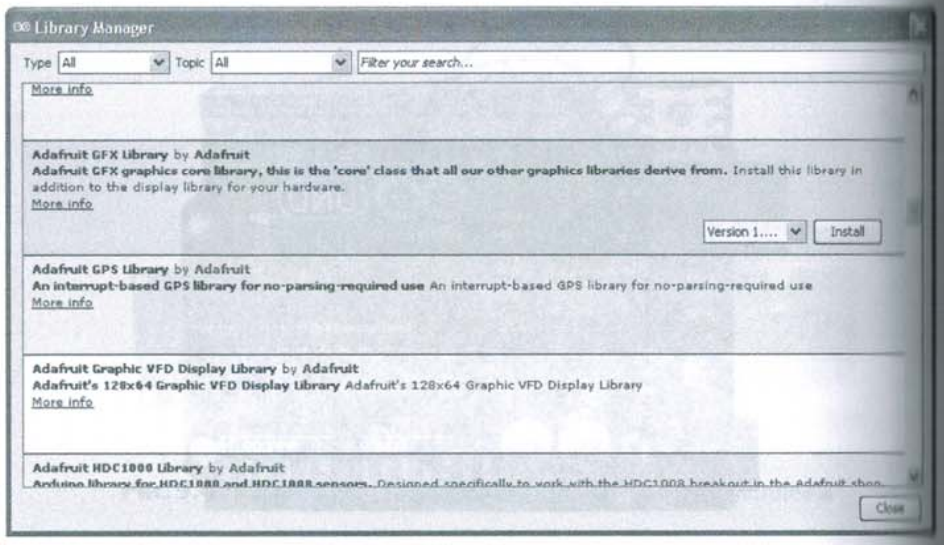


Рис. 9.6. Установка библиотек Adafruit

```
// sketch 09-02

#include <SPI.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

Adafruit_SSD1306 display(4); // выберите неиспользуемый
                             контакт

void setup()
{
```



```
  display.begin(SSD1306_SWITCHCAPVCC, 0x3c); // адрес может
потребоваться изменить
  display.setTextSize(4);
  display.setTextColor(WHITE);
}

void loop()
{
  static int count = 0;
  display.clearDisplay();
  display.drawRoundRect(0, 0, 127, 63, 8, WHITE);
  display.setCursor(20,20);
  display.print(count);
  display.display();
  count ++;
  if (count > 9999)
  {
    count = 0;
  }
  delay(1000);
}
```

В самом начале скетч импортирует необходимые библиотеки и инициализирует переменную `display`. Параметр, используемый при инициализации, определяет дополнительный контакт «сброса», который имеют отдельные OLED-дисплеи (включая производимые компанией Adafruit). Если ваш дисплей не имеет контакта сброса, укажите в качестве значения параметра номер любого неиспользуемого контакта. В данном случае я выбрал контакт 4.

Функция `setup` инициализирует дисплей, передавая I2C-адрес 0x3c во втором параметре, который вам, возможно, придется изменить. Затем она устанавливает крупный размер шрифта (4) и белый цвет текста (цвет можно выбрать любой, кроме черного).

Функция `loop` очищает дисплей, рисует прямоугольник с закругленными углами, задает позицию курсора и затем выводит значение счетчика `count`. В действительности изображение выводится на дисплей только после выполнения команды `display.display()`. Затем переменная `count` увеличивается на единицу и выполняется задержка на одну секунду.

Библиотека Adafruit GFX включаает большое разнообразие процедур рисования для вывода изображений на графический дисплей. Описание этой библиотеки можно найти по адресу: *https://learn.adafruit.com/adafruit-gfx-graphics-library*.

В заключение

Как видите, работать с платами расширения совсем не сложно, особенно когда основную работу выполняет библиотека.

В следующей главе мы подключим Arduino к компьютерной сети и Интернету.

10

Arduino и Интернет вещей

Интернет вещей (Internet of Things, IoT) — концепция вычислительной сети физических предметов («вещей»), оснащенных встроенными технологиями для взаимодействия друг с другом или с внешней средой. С каждым годом появляется все больше и больше устройств, способных подключаться к Интернету. В данном случае под устройствами подразумеваются вовсе не компьютеры, а самые обычные бытовые приборы и переносные устройства. К ним относятся все виды средств автоматизации быта, от интеллектуальных бытовых электроприборов и устройств управ-

ления освещением до охранных систем и даже кормушек для домаш них питомцев, допускающих возможность управления через Интер нет, а также масса менее практичных, но весьма интересных проектов.

Платы Arduino широко применяются в IoT-проектах, но для их ре ализации необходимо использовать специализированные модели Arduino или платы расширения, добавляющие сетевые возможно сти, которые позволят определять состояние «вещей» и управлять ими посредством локальной сети и Интернета. Подключение к сети может быть проводным (Ethernet) или беспроводным (WiFi).

В этой главе мы исследуем возможности использования Arduino с платой расширения Ethernet (или комбинированного устройства, такого как EtherTen от компании *Freetronics*), а также набирающих популярность плат WiFi на микроконтроллере ESP8266, которые программируются на языке Arduino C и отличаются необычайно низкой ценой, от $5 и ниже (рис. 10.1).

Вы узнаете, как с использованием обоих типов плат запустить ло кальный веб-сервер и посылать веб-запросы службам в Интерне те. В заключение кратко будут описаны некоторые альтернативные устройства, такие как Particle Photon и Arduino Yun.

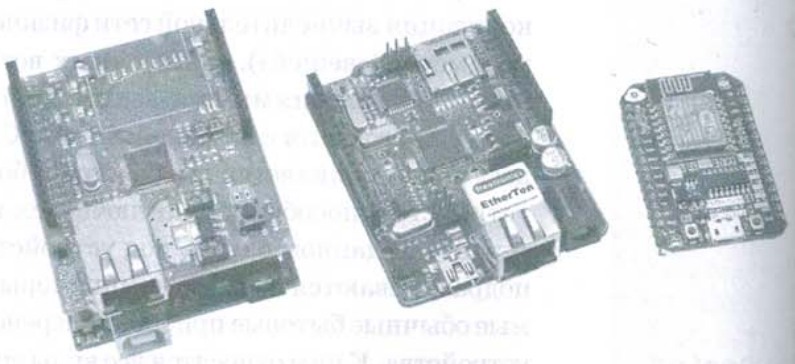


Рис. 10.1. *Слева направо*: Arduino Uno + плата Ethernet, EtherTen и Node MCU ESP8266 boardst

Обмен данными с веб-серверами

Прежде чем рассматривать, как в Arduino организован обмен данными между браузером и веб-сервером, потратим немного времени на знакомство с протоколом передачи гипертекста (HyperText Transfer Protocol, HTTP) и языком разметки гипертекста (HyperText Markup Language, HTML).

HTTP

Протокол передачи гипертекста определяет порядок взаимодействий веб-браузеров с веб-сервером.

Когда вы пытаетесь открыть веб-страницу в веб-браузере, он посылает запрос серверу, на котором находится страница, выражая свое желание получить ее. Запрашиваемая страница может содержать простую разметку HTML.

Веб-сервер всегда прослушивает сеть, ожидая получения таких запросов, и, когда запрос поступает, он обрабатывает его. В простейшем случае обработка заключается всего лишь в отправке обратно страницы HTML, которая была указана в скетче Arduino.

HTML

Язык разметки гипертекста определяет правила форматирования обычного текста, чтобы он выглядел привлекательнее или заметнее при отображении в окне браузера. Например, далее приводится разметка HTML, которую браузер отображает так, как показано на рис. 10.2:

```
<html>
<body>
<h1>Programming Arduino</h1>
<p>A book about programming Arduino</p>
</body>
</html>
```

Разметка HTML состоит из тегов. Эти теги имеют начало и конец и обычно содержат другие теги. Начало тега отмечается как: < имя

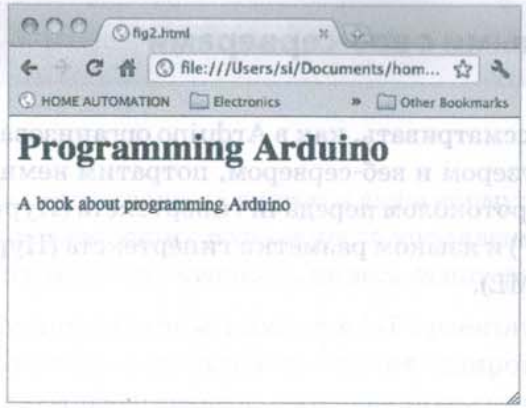


Рис. 10.2. Пример отображения разметки HTML

тега и > (например, <html>). Конец тега выглядит аналогично, с той лишь разницей, что за угловой скобкой < следует слеш /. В предыдущем примере внешним является тег <html>, он содержит тег <body>. Все веб-страницы должны начинаться с этих тегов и, как можно видеть в конце примера, завершаться соответствующими конечными тегами. Обратите внимание на то, что теги должны завершаться в правильном порядке, то есть тег body должен закрываться раньше тега html.

Теперь перейдем к самому интересному в середине, к тегам h1 и p. Эти теги управляют отображением текста.

Тег h1 сообщает, что его содержимое является заголовком (header) первого уровня. Текст в этом теге отображается крупным жирным шрифтом. Тег p — это тег абзаца (paragraph), а все его содержимое отображается как единый абзац.

Я показал лишь вершину айсберга HTML. Более детальное описание языка HTML можно найти на многих интернет-ресурсах.

Arduino Uno как веб-сервер

Для начала рассмотрим скетч, превращающий Arduino с платой расширения Ethernet в маленький веб-сервер. Это определенно не ферма серверов Google, но позволит отправлять веб-запросы плате Arduino

и просматривать результаты в браузере. Альтернативой комбинации из двух плат — Arduino Uno и Ethernet — может служить плата EtherTen от компании *Freetronics*, которая объединяет Uno и сетевой интерфейс на одной плате и совместима с Uno платой расширения.

Прежде чем выгрузить скетч 10-01, в него необходимо внести пару изменений. В начале скетча можно найти такую строку:

```
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
```

Эта строка определяет MAC-адрес (Media Access Control — управление доступом к среде), который должен быть уникальным среди всех устройств, подключенных к сети.

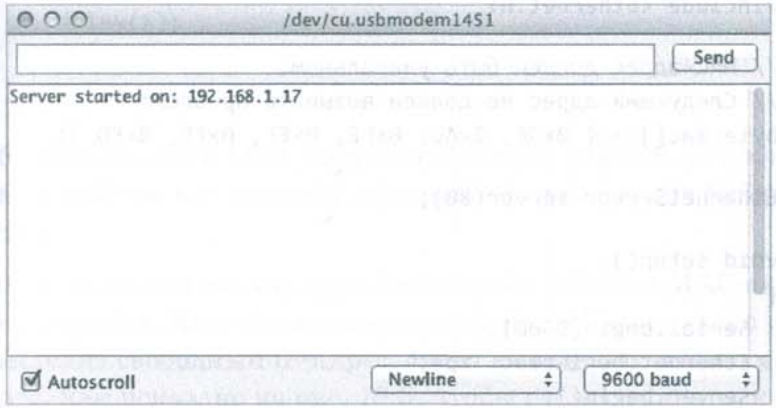


Рис. 10.3. Определение IP-адреса платы Arduino

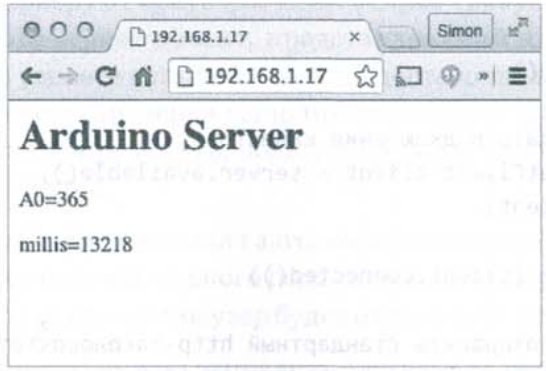


Рис. 10.4. Пример простого веб-сервера на Arduino

Подключите плату Arduino к компьютеру кабелем USB и выгрузите скетч. Теперь можно отсоединить кабель USB, подключить источник питания к плате Arduino и кабель сети — к разъему Ethernet. Откройте окно монитора порта в Arduino IDE, и вы увидите строки, отображенные на рис. 10.3 и сообщающие IP-адрес, который был выделен плате Arduino вашей сетью. Введите этот IP-адрес в адресной строке браузера на вашем компьютере, и увидите веб-страницу, напоминающую изображенную на рис. 10.4.

Далее приводится листинг со скетчем 10-01:

```
// sketch 10-01 Пример простого сервера
#include <SPI.h>
#include <Ethernet.h>

// MAC-адрес должен быть уникальным.
// Следующий адрес не должен вызывать проблем:
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

EthernetServer server(80);

void setup()
{
  Serial.begin(9600);
  Ethernet.begin(mac, ip);
  server.begin();
  Serial.print("Server started on: ");
  Serial.println(Ethernet.localIP());

}

void loop()
{
  // ожидать подключения клиентов
  EthernetClient client = server.available();
  if (client)
  {
    while (client.connected())
    {
      // отправить стандартный http-заголовок ответа
      client.println("HTTP/1.1 200 OK");
      client.println("Content-Type: text/html");
      client.println();
      // отправить тело ответа
      client.println("<html><body>");
      client.println("<h1>Arduino Server</h1>");
      client.print("<p>A0=");
      client.print(analogRead(0));
      client.println("</p>");
      client.print("<p>millis=");
      client.print(millis());
      client.println("</p>");
      client.println("</body></html>");
      client.stop();
    }
    delay(1);
  }
}
```


Подобно библиотеке LCD, обсуждавшейся в главе 9, все хлопоты по взаимодействию с платой расширения берет на себя библиотека Ethernet.

Функция `setup` инициализирует библиотеку Ethernet MAC-адресом, заданным ранее. Как только соединение будет установлено и ваша сеть выделит свободный IP-адрес, скетч выведет его в окно монитора порта, как показано на рис. 10.3, чтобы вы могли знать, куда направить свой браузер.

Функция `loop` отвечает за обслуживание любых запросов, поступающих веб-серверу от браузера. Если запрос требует ответа, вызов `server.available` вернет объект, представляющий клиента. Что такое объект, вы узнаете в главе 11. А пока достаточно знать, что, если клиент существует (проверка выполняется первой инструкцией `if`), есть возможность определить, подключен ли он к веб-серверу, вызовом `client.connected`.

Следующие три строки выводят заголовок возвращаемого ответа. Он просто сообщает браузеру, какого рода содержимое ему предстоит отобразить. В данном случае браузер будет отображать разметку HTML.

Вслед за заголовком нужно отправить браузеру тело ответа с разметкой HTML. Разметка включает обычные теги `<html>` и `<body>`, а также

тег заголовка <h1> и два тега <p>, где отображаются значение на аналоговом входе A0 и значение, возвращаемое функцией millis, — число миллисекунд, прошедших с момента последнего сброса платы Arduino.

Наконец, вызов client.stop сообщает браузеру, что передача сообщения закончена. После этого браузер отобразит страницу.

В последнем разделе этой главы мы повторим этот пример, но с использованием платы расширения WiFi ESP8266 вместо Ethernet.

Управление Arduino по сети

Второй пример демонстрирует использование платы расширения Ethernet для управления контактами с D3 по D7 на плате Arduino через веб-форму.

В отличие от примера простого веб-сервера нам нужно найти способ передачи настроек контактов в плату Arduino.

Сделать это можно с помощью метода передачи данных, являющегося частью стандарта HTTP. Чтобы задействовать этот метод, нужно встроить механизм передачи в разметку HTML, которую плата Arduino будет возвращать браузеру, а тот — отображать ее в виде формы. Эта форма (рис. 10.5) содержит переключатели состояний On (Включено) и Off (Выключено) контактов и кнопку Update (Изменить), щелчок на которой отправляет настройки контактов в плату Arduino.

Когда выполняется щелчок на кнопке Update (Изменить), плате Arduino посылается второй запрос. Он выглядит так же, как первый, за исключением содержащихся в нем параметров запроса со значениями состояния контактов.

По своей сути параметры запроса похожи на параметры функции. Параметры функции позволяют передать в функцию некоторую информацию, например число вспышек светодиода, а параметры запроса дают возможность передать в плату Arduino данные для обработки. Когда скетч в плате Arduino примет веб-запрос, он извлечет новые значения состояний контактов из параметров и изменит фактические состояния.

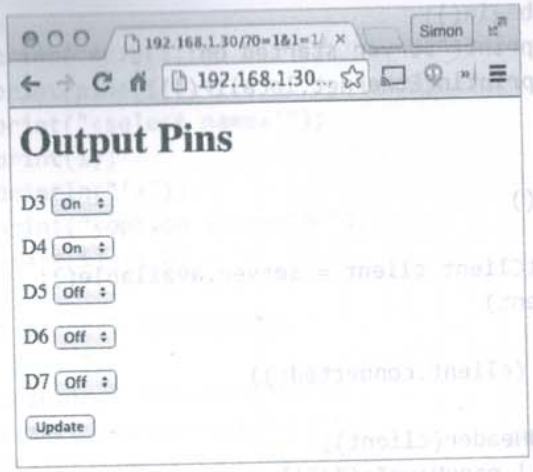


Рис. 10.5. Форма отправки сообщения

Исходный код скетча приводится далее:

```
// sketch 10-02 Установка состояний контактов через Интернет
#include <SPI.h>
#include <Ethernet.h>

// MAC-адрес должен быть уникальным.
// Следующий адрес не должен вызывать проблем:
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };

EthernetServer server(80);

int numPins = 5;
int pins[] = {3, 4, 5, 6, 7};
int pinState[] = {0, 0, 0, 0, 0};
char line1[100];

void setup()
{
  for (int i = 0; i < numPins; i++)
  {
    pinMode(pins[i], OUTPUT);
  }
  Serial.begin(9600);
  Ethernet.begin(mac, ip);
```

```
  server.begin();
  Serial.print("Server started on: ");
  Serial.println(Ethernet.localIP());

}

void loop()
{
  EthernetClient client = server.available();
  if (client)
  {
    while (client.connected())
    {
      readHeader(client);
      if (! pageNameIs("/"))
      {
        client.stop();
        return;
      }
      client.println("HTTP/1.1 200 OK");
      client.println("Content-Type: text/html");
      client.println();

      // отправить тело ответа
      client.println("<html><body>");
      client.println("<h1>Output Pins</h1>");
      client.println("<form method='GET'>");
      setValuesFromParams();
      setPinStates();
      for (int i = 0; i < numPins; i++)
      {
        writeHTMLforPin(client, i);
      }
      client.println("<input type='submit' value='Update'/>");
      client.println("</form>");
      client.println("</body></html>");

      client.stop();
    }
  }
}

void writeHTMLforPin(Client client, int i)
```



```
{
  client.print("<p>D");
  client.print(pins[i]);
  client.print("<select name='");
  client.print(i);
  client.println("'>");
  client.print("<option value='0'");
  if (pinState[i] == 0)
  {
    client.print(" selected");
  }
  client.println(">Off</option>");
  client.print("<option value='1'");
  if (pinState[i] == 1)
  {
    client.print(" selected");
  }
  client.println(">On</option>");
  client.println("</select></p>");
}

void setPinStates()
{
  for (int i = 0; i < numPins; i++)
  {
    digitalWrite(pins[i], pinState[i]);
  }
}

void setValuesFromParams()
{
  for (int i = 0; i < numPins; i++)
  {
    pinState[i] = valueOfParam(i + '0');
  }
}

void readHeader(EthernetClient client)
{
  // прочитать первую строку заголовка
  char ch;
  int i = 0;
```

```
  while (ch != '\n')
  {
    if (client.available())
    {
      ch = client.read();
      line1[i] = ch;
      i ++;
    }
  }
  line1[i] = '\0';
  Serial.println(line1);
}

boolean pageNameIs(char* name)
{
  // имя страницы начинается с позиции 4
  // заканчивается пробелом
  int i = 4;
  char ch = line1[i];
  while (ch != ' ' && ch != '\n' && ch != '?')
  {
    if (name[i-4] != line1[i])
    {
      return false;
    }
    i++;
    ch = line1[i];
  }
  return true;
}

int valueOfParam(char param)
{
  for (int i = 0; i < strlen(line1); i++)
  {
    if (line1[i] == param && line1[i+1] == '=')
    {
      return (line1[i+2] - '0');
    }
  }
  return 0;
}
```


Для управления состоянием контактов скетч использует два массива. Первый, pins, просто определяет список контактов, управляемых скетчем. Массив pinState хранит состояние каждого контакта: 0 или 1.

Чтобы извлечь полученную от браузера информацию, описывающую новые состояния контактов, необходимо прочитать заголовок запроса. Фактически вся необходимая информация содержится в первой строке заголовка. Для хранения первой строки в скетче используется массив символов line1.

После того как пользователь щелкнет на кнопке Update (Изменить) и отправит форму плате Arduino, строка URL страницы будет выглядеть примерно так:

```
http://192.168.1.17/?0=1&1=1&2=0&3=0&4=0
```

Список параметров запроса начинается за символом ?, и внутри списка параметры отделяются друг от друга символом &. Взгляните на первый параметр (0=1), он означает, что первый контакт в массиве (pins[0]) должен получить значение 1. Если взглянуть на первую строку в заголовке, можно увидеть те же параметры запроса:

```
GET /?0=1&1=1&2=0&3=0&4=0 HTTP/1.1
```

Перед списком параметров присутствует текст GET/. Он указывает, какая страница была запрошена браузером. В данном случае символ / соответствует начальной странице.

В функции loop скетч вызывает функцию readHeader, чтобы прочитать первую строку заголовка. Затем вызовом функции pageNameIs проверяется, была ли запрошена начальная страница /.

После этого скетч генерирует заголовок и отправляет форму HTML для отображения в браузере. Прежде чем сгенерировать разметку HTML, описывающую состояние каждого конкретного контакта, скетч вызывает функцию setValuesFromParams, чтобы прочитать параметры запроса и установить соответствующие значения в массиве pinStates. Затем этот массив используется для установки состояний контактов перед функцией writeHTMLforPin, которая вызывается для каждого контакта. Эта функция поэтапно генерирует раскрывающийся список, гарантируя выбор соответствующих значений с помощью инструкций if.

Функции `readHeader`, `pageNameIs` и `valueOfParam` — это вспомогательные универсальные функции, которым можно найти применение в других скетчах.

Чтобы убедиться в том, что контакты действительно включаются и выключаются, можно воспользоваться мультиметром, как было показано в главе 6. Желающие могут попробовать подключить светодиоды или реле к управляемым контактам, чтобы получить более наглядный эффект.

Веб-сервер Node MCU

Микроконтроллер ESP8266 — это целая WiFi-система в одной микросхеме. То есть одна-единственная микросхема обладает теми же возможностями, что и плата Arduino Uno с подключенной платой расширения WiFi. Платы на этой микросхеме имеют несколько универсальных входов/выходов и один аналоговый вход и могут программироваться из Arduino IDE, как самые обычные платы Arduino.

На рис. 10.6 представлены наиболее популярные модели плат на микроконтроллере ESP8266.

Модель ESP01 отличается очень маленькими размерами, имеет только два входа/выхода и может программироваться только по последовательной линии, с помощью специального программатора. Другая модель — Node MCU (справа на рис. 10.6) — имеет полноценный интерфейс USB для программирования, а также приличный набор входов/выходов. Начинать знакомство с этими платами лучше с модели Node MCU.

Микроконтроллер ESP8266 имеет около 36 КБ памяти для данных (что намного больше, чем в Arduino Uno с ее 2 КБ). В нем отсутствует встроенная флеш-память, но она имеется на самих платах. Например, на плате Node MCU имеется отдельная микросхема флеш-памяти для хранения программ объемом 4 МБ.

Ни одна из плат на микроконтроллере ESP8266 не является официальной платой Arduino; в действительности плата Node MCU поставляется со своим встроенным программным обеспечением на языке

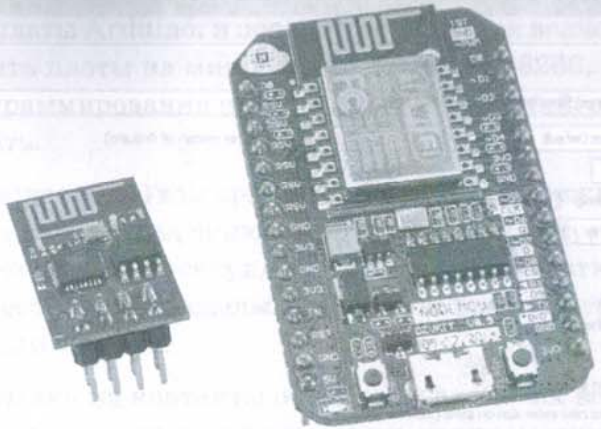


Рис. 10.6. Платы на микроконтроллере ESP8266: ESP01 (слева) и Node MCU (справа)

Lua. Однако, благодаря усилиям Sandeep Mistry, это встроенное программное обеспечение можно заменять и программировать плату, как любую другую плату Arduino. Все платы на микроконтроллере ESP8266 поддерживают проводное подключение к последовательному порту Arduino и таким образом используются как интерфейс WiFi. Но, учитывая, что эти платы можно использовать взамен Arduino, есть смысл реализовать весь их потенциал.

Перед использованием платы Node MCU необходимо обновить Arduino IDE, добавив в нее поддержку нового типа плат. Далее приводятся инструкции по обновлению, но они предполагают, что у вас установлена версия Arduino IDE 1.6 или выше.

Итак, запустите Arduino IDE и откройте окно **Preferences** (Настройки), выбрав одноименный пункт в меню **File** (Файл). Введите в поле **Additional Board Manager URLs** (Дополнительные ссылки для Менеджера плат) адрес *http://arduino.esp8266.com/stable/package_esp8266com_index.json* (рис. 10.7).

Откройте окно **Boards Manager** (Менеджер плат), выбрав одноименный пункт в меню **Tools ▸ Boards** (Инструменты ▸ Плата). Прокрутите список вниз до конца и щелкните на кнопке **Install** (Установка) в разделе с заголовком «esp8266 by ESP8266 Community».

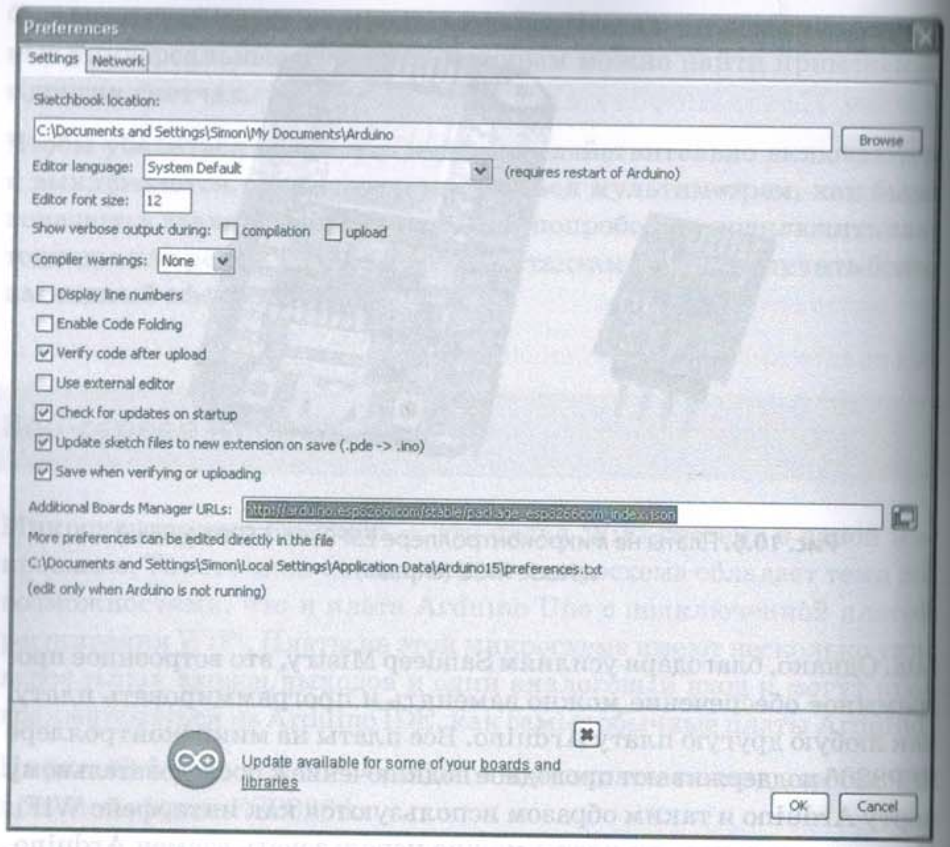


Рис. 10.7. Добавление ссылки в окне Board Manager (Менеджер плат) для загрузки поддержки плат на микроконтроллере ESP8288

Закройте окно **Boards Manager** (Менеджер плат) и загляните в список доступных плат — здесь вы должны увидеть несколько новых моделей на основе микроконтроллера ESP8266, в частности: «Node MCU0.9» и «NodeMCU 1.0». Когда вы приобретете плату на микроконтроллере ESP8266, вы сможете выбрать один из этих двух типов.

Чтобы получить возможность программировать платы Node MCU, необходимо установить драйвер для поддержки микросхемы связи между USB и последовательной линией. Эта микросхема отличается от аналогичной, что установлена на плате Arduino Uno, чем и объясняется такое требование. Загрузите драйвер для своей платформы по адресу *https://github.com/nodemcu/nodemcu-devkit/tree/master/Drivers* и запустите программу установки.

В приложении Arduino IDE выберите тип платы и порт, как для любой другой платы Arduino, и после этого появится возможность программировать платы на микроконтроллере ESP8266. Однако процедура программирования имеет несколько отличий, о которых вы должны знать.

- Иногда (не всегда) вам придется нажимать и удерживать кнопку **Flash** на плате перед включением питания и отпускать ее только через несколько секунд после включения платы. Эту процедуру необходимо проделывать, если во время выгрузки скетча возникли ошибки.
- Для ссылки на контакты от **D0** до **D8** в скетчах всегда следует использовать константы с именами, начинающимися с символа **D**, например: `pinMode(D0, OUTPUT)`. В скетчах для официальных плат Arduino символ «D» можно опускать.
- Модель Node MCU имеет встроенный светодиод, подобный светодиоду **L** на плате Arduino Uno, но он подключен к контакту **D0**, а не 13, поэтому, если пожелаете опробовать скетч Blink, вам потребуется изменить его, задействовав контакт **D0**.
- В скетчах для официальных плат Arduino не имеет значения, в каком порядке определяются функции. В скетчах для платы Node MCU функции должны определяться выше места первого их вызова.

Библиотека поддержки Node MCU немного отличается от стандартной библиотеки Arduino Ethernet. Скетч 10-03 повторяет реализацию простого веб-сервера, показанного на рис. 10.4, для Node MCU и WiFi.

```
// sketch 10-03. Простой веб-сервер на плате Node MCU
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

const char* ssid = "my-network-name";
const char* password = "my_password";

ESP8266WebServer server(80);
```

```
void handleRoot()
{
  String message = "<html><body>\n";
  message += "<h1>Arduino Server</h1>\n";
  message += "<p>A0=";
  message += analogRead(A0);
  message += "</p>";
  message += "<p>millis=";
  message += millis();
  message += "</p>";
  message += "</html></body>\n";
  server.send(200, "text/html", message);
}

void connectToWiFi()
{
  Serial.print("\n\nConnecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void setup()
{
  Serial.begin(115200);
  connectToWiFi();
  server.on("/", handleRoot);
  server.begin();
  Serial.println("HTTP server started");
}

void loop()
{
  server.handleClient();
}
```


Скетч для Node MCU имеет несколько иную организацию, чем скетч 10-01 для Arduino с платой расширения Ethernet. Здесь вместо MAC-адреса определены константы ssid (имя беспроводной сети) и password. Переменная, представляющая объект сервера, в этом скетче имеет тип ESP8266WebServer вместо EthernetServer.

Функция handleRoot демонстрирует интересную особенность библиотеки ESP8266 — возможность определять обработчики для отдельных страниц, обслуживаемых сервером. Так, если заглянуть в функцию setup, можно заметить в ней команду server.on("/", handleRoot). Она сообщает серверу, что для обработки запроса к корневой странице (/) должна вызываться функция handleRoot, которая сгенерирует необходимую разметку HTML и вернет ее клиенту.

Для конструирования разметки HTML, строку за строкой, эта функция использует класс String. Имейте в виду, что в некоторых задачах класс String может потреблять значительные объемы памяти, поэтому, если во время выполнения кода вдруг возникают неожиданные ошибки, они могут быть вызваны классом String. Этот класс отлично подходит для конструирования небольших строк, таких как в данном примере, но для сборки более длинных строк лучше все-таки использовать символьный буфер, как показано в скетче 10-04.

Создание WiFi-соединения, а также вывод IP-адреса в окно монитора порта реализует функция connectToWiFi. Обратите внимание, что в этом скетче выбрана скорость обмена по последовательной линии 115200 бод, поэтому ту же скорость необходимо выбрать в раскрывающемся списке, находящемся справа внизу в окне монитора порта.

Функция setup просто вызывает connectToWiFi, устанавливает обработчик корневой страницы и запускает сервер. Функция loop вызывает метод handleClient сервера, который ожидает поступления входящих запросов и обслуживает их.

Управление Node MCU по сети

Скетч 10-02, реализующий управление платой Arduino по сети, можно адаптировать для работы с платой Node MCU. Результат такой адаптации приведен в скетче 10-04. Чтобы опробовать его, под-

ключите светодиоды к соответствующим контактам или измеряйте напряжение мультиметром.

```
// sketch 10-04 Управление Node MCU по сети

#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>

const char* ssid = "my-network-name";
const char* password = "my_password";

int numPins = 5;
char* pinNames[] = {"D5", "D6", "D7", "D8", "D9"};
int pins[] = {D5, D6, D7, D8, D9};
int pinState[] = {0, 0, 0, 0, 0};
ESP8266WebServer server(80);

void setPinStates()
{
  for (int i = 0; i < numPins; i++)
  {
    digitalWrite(pins[i], pinState[i]);
  }
}

void setValuesFromParams()
{
  for (int i = 0; i < numPins; i++)
  {
    pinState[i] = server.arg(i).toInt();
  }
}

void connectToWiFi()
{
  Serial.print("\n\nConnecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
```



```
    Serial.print(".");
  }
  Serial.println("\nWiFi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void handleRoot()
{
  char buff[1000];
  Serial.println("Got a Request");
  setValuesFromParams();
  setPinStates();

  strcat(buff, "<html><body>\n");
  strcat(buff, "<h1>Output Pins</h1>\n");
  strcat(buff, "<form method='GET'>\n");
  for (int i = 0; i < numPins; i++)
  {
    strcat(buff, "<p>");
    strcat(buff, pinNames[i]);
    strcat(buff, " <select name='");
    char indexStr[10];
    sprintf(indexStr, "%d", i);
    strcat(buff, indexStr);
    strcat(buff, "'><option value='0'");
    if (pinState[i] == 0)
    {
      strcat(buff, " selected");
    }
    strcat(buff, ">Off</option>");
    strcat(buff, "<option value='1'");
    if (pinState[i] == 1)
    {
      strcat(buff, " selected");
    }
    strcat(buff, ">On</option></select></p>\n");
  }
  strcat(buff, "<input type='submit' value='Update'/>");
  strcat(buff, "</form></html></body>\n");
  server.send(200, "text/html", buff);
}
```

```
void setup()
{
  for (int i = 0; i < numPins; i++)
  {
    pinMode(pins[i], OUTPUT);
  }
  Serial.begin(115200);
  connectToWiFi();
  server.on("/", handleRoot);
  server.begin();
  Serial.println("HTTP server started");
}

void loop()
{
  server.handleClient();
}
```

Этот скетч организован практически так же, как его аналог для Arduino (скетч 10-02). Однако он использует другой набор контактов и содержит массив строк с их названиями для использования в веб-странице. Вспомогательные функции `pageNameIs` и `valueOfParam`, написанные ранее для проверки имени страницы, к которой осуществляется доступ, и определения значений параметров запроса, оказались не нужны в версии для ESP8266, так как для обработки страниц библиотека использует механизм `server.on` и дает прямой доступ к значениям параметров с использованием `server.arg(i)`, где `i` — индекс параметра. Функция `arg` позволяет также извлекать параметры по именам. Подробнее об этом рассказывается в документации, доступной по адресу: *http://links2004.github.io/Arduino/d3/d58/class_e_s_p8266_web_server.html*.

В данном примере функция `handleRoot` конструирует намного более длинную строку с разметкой HTML для отправки клиенту. Он иллюстрирует иной подход и вместо класса `String` использует строковый буфер (`buff`) и функцию `strcat` для объединения последовательности более мелких строк в одну большую строку (завершающуюся символом `\0`). Этот прием хорош, пока дело не доходит до добавления чисел, например номеров контактов. Для решения этой задачи используется второй строковый буфер (`indexStr`). Число записывается

в этот буфер с помощью функции `sprintf`, которая «впечатывает» заданное число в буфер, и затем содержимое буфера добавляется в `buff`.

Все эти операции, связанные с конструированием буфера, не нужны при использовании библиотеки Arduino Ethernet library, потому что она позволяет посылать ответ браузеру построчно, не собирая полное сообщение перед его отправкой.

Вызов веб-служб

Все примеры, приведенные выше в этой главе, демонстрировали превращение платы Arduino или Node MCU в веб-сервер. А как быть, если понадобится, чтобы плата действовала подобно веб-браузеру и посылала запросы веб-серверам?

Например, можно было бы заставить плату читать температуру воздуха через регулярные интервалы времени и посылать ее веб-службе. В следующих двух разделах вы узнаете, как с помощью обеих плат — Arduino и Node MCU — посылать запросы популярной веб-службе автоматизации IFTTT (If This Then That — сделай это, потом то). Когда служба получает запрос, она выполняет указанное действие, в данном случае — отправляет электронное письмо со значением, прочитанным с контакта **A0**.

Независимо от используемой вами платы — Arduino или Node MCU — необходимо сначала создать учетную запись на сайте *ifttt.com*, а затем новый «рецепт». Служба IFTTT имеет очень простой и интуитивно понятный интерфейс. Так, после щелчка на кнопке для создания нового рецепта щелкните на слове «this» в заголовке «if this then that». Вам будет предложено выбрать запускаемый канал. Введите «maker» в поле поиска, чтобы найти канал **Maker** (соответствующая ему пиктограмма содержит многоцветную большую букву «M»). Выберите канал **Maker** и затем триггер «Receive a web request» (получение веб-запроса). Далее вам будет предложено ввести имя события (рис. 10.8).

Введите в поле **Event Name** (Имя события) текст «arduino_spoke» и щелкните на кнопке **Create Trigger** (Создать триггер).

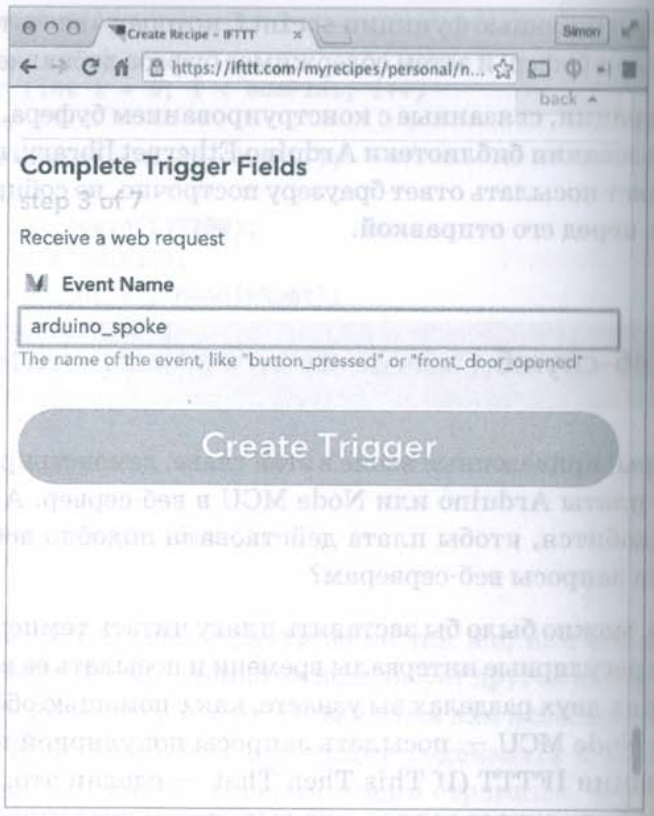


Рис. 10.8. Определение триггера

Следующий шаг — определение действия. То есть вам нужно указать, что должно произойти, когда будет запущен рецепт. В данном случае нам нужно, чтобы служба IFTTT послала электронное письмо, но вы можете выбрать другое действие и заставить IFTTT, например, послать сообщение в Twitter, обновить статус в Facebook или сделать что-то еще (на выбор предоставляется большое количество разных действий). После того как вы добьетесь исправной работы от простого рецепта, можете поэкспериментировать с другими действиями. Щелкните на слове «that» в заголовке «if this then that» и найдите канал «e-mail». Выберите единственное доступное действие: «Send me an e-mail» (послать мне электронное письмо). В ответ на это откроется форма, как показано на рис. 10.9.

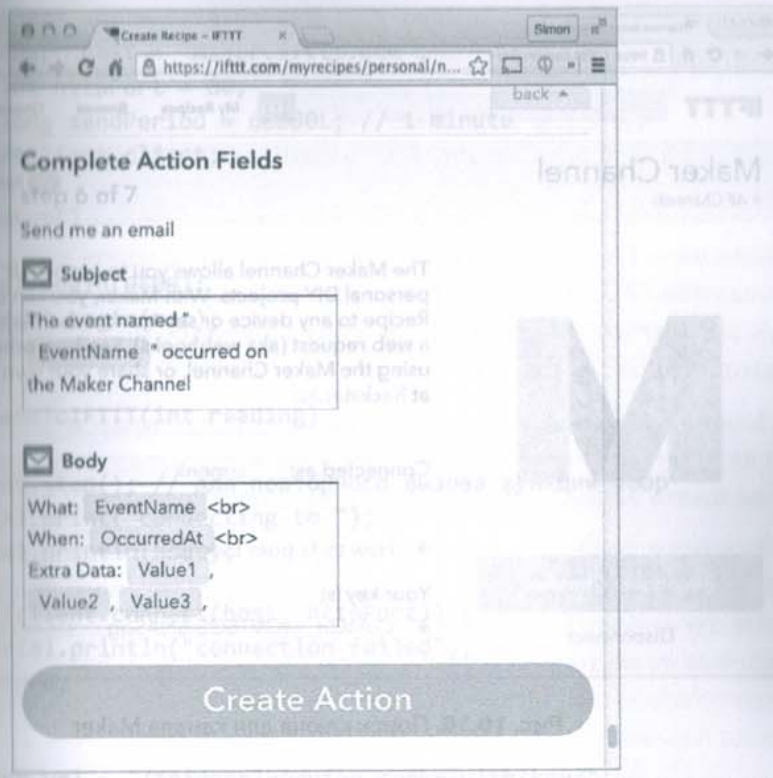


Рис. 10.9. Заполнение формы с описанием действия

В полях формы можно оставить значения, предложенные по умолчанию, и просто щелкнуть на кнопке **Create Action** (Создать действие), а можно изменить тему и текст письма, которое должно отправляться вам. Позднее вы сможете вернуться и изменить действие. В заключение щелкните на кнопке **Create Recipe** (Создать рецепт), и после этого новый рецепт будет ждать сообщения от вашей платы Arduino или Node MCU.

Далее необходимо сделать еще один важный заключительный шаг — получить ключ для запуска вашего события, чтобы только вы могли пользоваться им. Для этого щелкните на ссылке «Channels» (Каналы) вверху на домашней странице *ifttt.com*, найдите и выберите канал **Maker**. После этого должна открыться страница с описанием, как показано на рис. 10.10.

Рис. 10.10. Поиск ключа для канала Maker

Наиболее важная информация здесь находится вслед за подписью «Your key is:» («Ваш ключ:»). Вам нужно будет вставить его в свой скетч.

Arduino Uno и служба IFTTT

Начнем со скетча для плат Arduino и Ethernet, посылающего сообщения службе IFTTT (скетч 10-05).

```
// sketch 10-05 IFTTT
#include <SPI.h>
#include <Ethernet.h>

// MAC-адрес должен быть уникальным.
// Следующий адрес не должен вызывать проблем:
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
```



```
const char* key = "c1AsQq_qsQVTSO5-6NASqg";
const char* host = "maker.ifttt.com";
const int httpPort = 80;
const long sendPeriod = 60000L; // 1 minute
EthernetClient client;

void setup()
{
  Serial.begin(9600);
  Ethernet.begin(mac);
}

void sendToIFTTT(int reading)
{
  client.stop(); // для повторного вызова функции loop
  Serial.print("connecting to ");
  Serial.println(host);

  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }

  String url = "/trigger/arduino_spoke/with/key/";
  url += key;
  url += "?value1=" + String(reading);

  String req = String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n";
  Serial.println(req);
  client.print(req);
}

void loop()
{
  static long lastReadingTime = 0;
  long now = millis();
  if (now > lastReadingTime + sendPeriod)
  {
    int reading = analogRead(A0);
    sendToIFTTT(reading);
    lastReadingTime = now;
```

```
    }
    if (client.available())
    {
      Serial.write(client.read());
    }
  }
```

Измените константу key, подставив в нее свой ключ для канала Maker в службе IFTTT, который вы определили ранее (рис. 10.10). Константа sendPeriod определяет интервал времени между отправкой веб-запросов службе IFTTT.

Переменная client ссылается на экземпляр класса EthernetClient, а не EthernetServer, как в предыдущих примерах, потому что на этот раз плата Arduino действует в роли веб-клиента.

Функция loop вызывает функцию sendToIFTTT один раз в минуту и передает ей значение, прочитанное с аналогового входа A0. Затем она проверяет получение какого-либо ответа от сервера и выводит его в окно монитора порта. Строго говоря, в этом нет необходимости, но это позволяет получить ценную обратную связь и сразу заметить, если что-то пойдет не так.

Функция sendToIFTTT сначала вызывает метод client.stop(), чтобы завершить отправку предыдущего запроса перед подключением к серверу, и конструирует строку URL с адресом сервера (String url), включающую значение аргумента reading в виде единственного параметра запроса. Строка URL включает также ключ и имя события arduino_spoke, которое было задано при создании канала на сайте IFTTT.

Node MCU ESP8266 и служба IFTTT

Ниже приводится вариант скетча для платы Node MCU, посылающего запрос службе IFTTT.

```
// sketch 10-06

#include <ESP8266WiFi.h>

const char* ssid = "my-network-name";
```



```
const char* password = "my_password";
const char* key = "c1AsQq_qsQVTSO5-6NASqg";
const char* host = "maker.ifttt.com";
const int httpPort = 80;
const long sendPeriod = 10000L; // 1 минута

WiFiClient client;

void connectToWiFi()
{
  Serial.print("\n\nConnecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void sendToIFTTT(int reading)
{
  Serial.print("connecting to ");
  Serial.println(host);

  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }

  String url = "/trigger/arduino_spoke/with/key/";
  url += key;
  url += "?value1=" + String(reading);

  String req = String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n";
  Serial.println(req);
  client.print(req);
}
```

```
void setup()
{
  Serial.begin(115200);
  connectToWiFi();
}

void loop()
{
  static long lastReadingTime = 0;
  long now = millis();
  if (now > lastReadingTime + sendPeriod)
  {
    int reading = analogRead(A0);
    sendToIFTTT(reading);
    lastReadingTime = now;
  }
  if (client.available())
  {
    Serial.write(client.read());
  }
}
```

Кроме кода, осуществляющего подключение к сети WiFi, и использования класса `WiFiClient` вместо `EthernetClient` в остальном этот скетч идентичен предыдущему.

Другие средства реализации Интернета вещей

Платы Arduino Yun и Particle Photon (рис. 10.11) — альтернативные платформы реализации проектов для Интернета вещей.

Arduino Yun

Плата Arduino Yun — официальная плата Arduino для Интернета вещей. Это полнофункциональная плата, включающая отдельный модуль WiFi, разъем для подключения кабеля Ethernet и разъем USB. Фактически ее можно рассматривать как комбинацию Arduino

Uno с модулем WiFi на ESP8266-подобном микроконтроллере. Взаимодействие между программным обеспечением, управляющим Arduino, и модулем WiFi осуществляется с помощью программного «моста».

Чтобы использовать плату Yun, ее нужно подключить к сети WiFi. После этого появится возможность программировать ее из Arduino IDE без проводного подключения к компьютеру — через WiFi-соединение.

Плата также имеет разъем USB, но все перечисленные возможности увеличивают стоимость платы и делают ее слишком дорогой для многих IoT-проектов, особенно если учесть наличие в продаже плат на микроконтроллере ESP8266.

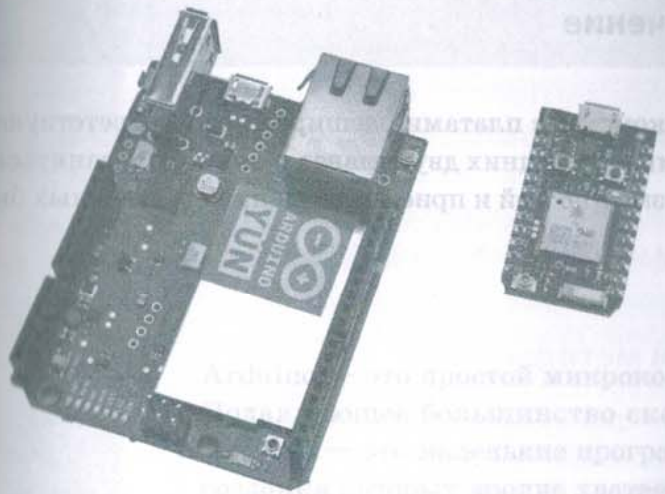


Рис. 10.11. Arduino Yun (слева) и Particle Photon (справа)

Particle Photon

Плата Photon и ее предшественница, Spark Core, — это сторонняя Arduino-подобная платформа для IoT-проектов. Она содержит только модуль WiFi и поддерживает возможность программирования без проводов с использованием веб-версии среды разработки, которая не является версией Arduino IDE, но выглядит очень похоже.

Photon программируется на языке Arduino C с некоторыми расширениями, обеспечивающими прозрачную связь с IoT-фреймворком Particle.io, избавляющим программиста от многих сложностей, связанных с программированием сетевых взаимодействий, с которыми мы столкнулись раньше в этой главе.

Photon стоит немного дороже плат на микроконтроллере ESP8266, но наличие встроенного IoT-фреймворка и возможность программирования по беспроводной сети делают ее отличным выбором для IoT-проектов, действующих в сетях WiFi.

Желающие познакомиться с платой Photon поближе могут приобрести мою книгу «Getting Started with the Photon» (Maker Media, 2015).

В заключение

После знакомства с платами расширения и соответствующими библиотеками в последних двух главах самое время заняться исследованием возможностей и приемов создания собственных библиотек.

11

C++ и библиотеки

Arduino — это простой микроконтроллер. Подавляющее большинство скетчей для Arduino — это маленькие программы, для создания которых вполне хватает возможностей языка программирования C. Однако в действительности языком программирования для Arduino является C++, а не C. C++ — это расширенная версия языка C, в которую добавлена поддержка *объектно-ориентированного* стиля программирования.

Объектно-ориентированное программирование

Это всего лишь маленькая книжка, поэтому здесь не удастся глубоко охватить язык программирования C++. Однако можно описать его базовые возможности и познакомиться с основами объектно-ориентированного программирования. Но главная цель — помочь вам увеличить *инкапсуляцию* ваших программ. Инкапсуляция подразумевает объединение взаимосвязанных компонентов вместе, что увеличивает привлекательность C++ для создания библиотек, таких как те, что были использованы в предыдущих главах для работы с платами расширения ЖК-дисплея и Ethernet.

Существует множество превосходных книг, посвященных описанию языка C++ и объектно-ориентированного программирования. Поищите их в верхних строчках рейтингов по этой теме на понравившихся вам сайтах книжных интернет-магазинов.

Классы и методы

В объектно-ориентированном программировании используется понятие *классов* как средства инкапсуляции. Класс напоминает раздел программы, включающий переменные, их называют *переменными членами*, и *методы*, которые очень похожи на функции, но применяются к классу. Эти функции могут быть *общедоступными*, то есть могут вызываться другими классами, или *закрытыми*, которые вызываются только другими методами внутри того же класса.

Обычно скетч для Arduino хранится в одном файле, однако при программировании на языке C++ программы часто хранятся более чем в одном файле. Как правило, каждый класс содержится в двух файлах: объявление класса — в заголовочном файле с расширением **.h**, а реализация — в файле с расширением **.cpp**.

Пример встроенной библиотеки

В двух предыдущих главах использовались встроенные библиотеки, поэтому сначала давайте рассмотрим подробнее, как они устроены.

Вернемся к скетчу 9-01 (откройте его в Arduino IDE). Здесь можно видеть команду `include`, подключающую файл **LiquidCrystal.h**:

```
#include <LiquidCrystal.h>
```

Это заголовочный файл с объявлением класса `LiquidCrystal`. Этот файл сообщает скетчу все, что он должен знать, чтобы использовать библиотеку. Этот файл можно найти в папке установки Arduino в подкаталоге **libraries/LiquidCrystal**. Откройте его в текстовом редакторе, чтобы заглянуть внутрь. Если вы пользуетесь Mac OS, щелкните правой кнопкой мыши на самом приложении Arduino и выберите в контекстном меню пункт **Show Package Contents** (Показать содержимое пакета). Затем перейдите в каталог **Contents/Resources/Java/libraries/LiquidCrystal**.

В файле **LiquidCrystal.h** вы увидите много программного кода, потому что это очень большой библиотечный класс. Фактическая реализация самого класса, все его потаенные механизмы, обеспечивающие отображение сообщений, находятся в файле **LiquidCrystal.cpp**.

В следующем разделе приводится пример простой библиотеки, который должен помочь вам понять основные принципы создания библиотек.

Создание библиотек

Многим может показаться, что создание библиотеки для Arduino под силу только искушенным ветеранам Arduino. В действительности же в создании библиотек нет ничего сложного. Например, можно превратить в библиотеку функцию `flash` из главы 4, которая заставляет светодиод мигать определенное число раз.

Чтобы создать необходимые для этого файлы на языке C++, вам понадобится текстовый редактор, такой как TextPad для Windows или TextMate для Mac.

Заголовочный файл

Начнем с создания папки для всех библиотечных файлов. Эту папку нужно создать в каталоге **libraries**, в папке документов Arduino. В Windows папка **libraries** должна находиться в каталоге **My Documents\Arduino**. В Mac OS ее можно найти в домашнем каталоге, в папке **Documents/Arduino/**, а в Linux — в папке **sketchbook**, в домашнем каталоге. Если вы не найдете папки **libraries** в указанном месте, создайте ее.

Все библиотеки, которые вы напишете сами, а также все неофициальные библиотеки, полученные со стороны, должны устанавливаться в папку **libraries**.

Назовем вновь созданную папку **Flasher**. Откройте текстовый редактор и введите следующий текст:

```
// Библиотека LED Flashing
#include "Arduino.h"

class Flasher
{
  public:
    Flasher(int pin, int duration);
    void flash(int times);
  private:
    int _pin;
    int _d;
};
```

Сохраните файл в папке **Flasher**, дав ему имя **Flasher.h**. Это заголовочный файл с объявлением библиотечного класса. Здесь объявляется структура класса. Как видите, он делится на общедоступную (`public`) и закрытую (`private`) части.

В общедоступной части объявляются компоненты, напоминающие начала двух функций. Они называются методами и отличаются от функций только своей связью с классом. Они могут использоваться только как часть класса. В отличие от функций методы нельзя использовать в отрыве от класса.

Имя первого метода, `Flasher`, начинается с прописной буквы — этот прием не рекомендуется использовать для именования обычных функций. Его имя совпадает с именем класса. Такие методы называют *конструкторами*. Он может использоваться только с целью создания нового объекта, экземпляра класса `Flasher`, для использования в скетче.

Например, можно вставить в скетч строку

```
Flasher slowFlasher(13, 500);
```

Она создаст новый объект `Flasher` с именем `slowFlasher`, способный зажечь светодиод, подключенный к контакту **D13**, на 500 мс.

Второй метод в классе называется `flash`. Этот метод принимает единственный аргумент — число вспышек. Так как этот метод связан с классом, то, чтобы вызвать его, необходимо сослаться на объект, созданный ранее:

```
slowFlasher.flash(10);
```

В результате этого вызова 10 раз мигнет светодиод, указанный в вызове конструктора `Flasher`.

В закрытом (`private`) разделе класса находятся объявления двух переменных: одной, с именем `_pin`, для хранения номера контакта и другой, с именем `_d`, — длительности вспышки. Каждый созданный объект класса `Flasher` будет иметь эти две переменные, благодаря чему каждый объект будет хранить номер своего контакта и свою продолжительность.

Эти переменные называют переменными-членами, чтобы подчеркнуть их принадлежность к классу. Их имена, начинающиеся с символа подчеркивания, многим покажутся необычными, однако это всего лишь распространенное соглашение, а не обязательное требование. В соответствии с еще одним распространенным соглашением принято начинать имена переменных-членов с маленькой буквы (от *англ.* member — член).

Файл реализации

В заголовочном файле находится только объявление класса. Собственно реализацию класса принято помещать в отдельный файл. Этот файл называется файлом реализации и имеет расширение .cpp.

Итак, создайте новый файл, содержимое которого приводится далее, и сохраните его с именем Flasher.cpp в папке Flasher:

```
#include "Flasher.h"

Flasher::Flasher(int pin, int duration)
{
  pinMode(pin, OUTPUT);
  _pin = pin;
  _d = duration / 2;
}

void Flasher::flash(int times)
{
  for (int i = 0; i < times; i++)
  {
    digitalWrite(_pin, HIGH);
    delay(_d);
    digitalWrite(_pin, LOW);
    delay(_d);
  }
}
```

Здесь многие увидят незнакомый пока синтаксис. Перед именами обоих методов присутствует префикс Flasher::. Он указывает, что метод принадлежит классу Flasher.

Метод конструктора (Flasher) просто присваивает значения аргументов соответствующим закрытым переменным-членам. Параметр duration делится на два перед присваиванием переменной-члену _d. Это сделано потому, что функция delay вызывается дважды, к тому же кажется логичнее, если параметр duration (продолжительность) будет определять общую продолжительность — продолжительность вспышки плюс интервал между вспышками.

Функция flash отвечает за всю фактическую работу: она выполняет заданное число итераций и в каждой из них включает и выключает светодиод с соответствующей задержкой.

Завершение создания библиотеки

Вы видели все основные этапы создания библиотеки. Теперь эту библиотеку можно развернуть и использовать в деле. Однако нужно сделать еще два шага, чтобы завершить создание библиотеки.

1. Определить ключевые слова, используемые в библиотеке, чтобы среда разработки Arduino IDE могла выделять их цветом в окне редактора.
2. Добавить несколько примеров использования библиотеки.

Ключевые слова

Чтобы определить ключевые слова, нужно создать файл keywords.txt в папке Flasher и добавить в него всего две строки:

```
Flasher   KEYWORD1
flash     KEYWORD2
```

По сути, это таблица с двумя графами в текстовом виде. В левой графе перечисляются ключевые слова, а в правой определяются их типы. Для имен классов следует указывать тип KEYWORD1, а для имен методов — KEYWORD2. Не важно, сколько пробелов или знаков табуляции будет отделять графы друг от друга, но каждое ключевое слово должно находиться строго в начале новой строки.

Примеры

Второе, что мы как добропорядочные разработчики для Arduino должны сделать, — включить в состав библиотеки папку с примерами. В данном случае библиотека настолько проста, что достаточно будет одного примера.

Примеры должны находиться в папке **examples**, внутри папки Flasher. Пример — это обычный скетч для Arduino, поэтому его можно создать в среде разработки Arduino IDE. Но перед этим нужно закрыть, а затем повторно открыть среду Arduino IDE, чтобы она прочитала информацию о новой библиотеке.

После перезапуска Arduino IDE выберите в главном меню пункт **File ▸ New** (Файл ▸ Создать), чтобы создать новое окно со скетчем. За-

тем в меню **Sketch ▸ Import Library** (Эскиз ▸ Подключить библиотеку) выберите библиотеку Flasher, как показано на рис. 11.1.

Библиотеки над разделительной линией в подменю — это официальные библиотеки, а ниже — неофициальные. Если ничего непредвиденного не произойдет, вы должны увидеть в списке и библиотеку Flasher.

Если этой библиотеки нет в списке, весьма вероятно, это связано с тем, что папка **Flasher** находится не в папке **libraries**, поэтому проверьте еще раз местоположение библиотеки.

Введите в окне редактора скетча следующий текст:

```
#include <Flasher.h>

const int ledPin = 13;
const int slowDuration = 300;
const int fastDuration = 100;

Flasher slowFlasher(ledPin, slowDuration);
Flasher fastFlasher(ledPin, fastDuration);

void setup(){}

void loop()
{
  slowFlasher.flash(5);
  delay(1000);
  fastFlasher.flash(10);
  delay(2000);
}
```

Среда разработки Arduino IDE не позволит сохранить скетч примера непосредственно в папке **libraries**, поэтому сохраните его где-нибудь в другом месте под именем **Simple_Flasher_Example**, а затем скопируйте всю папку **Simple_Flasher_Example** с только что сохраненным скетчем в папку **examples** библиотеки.

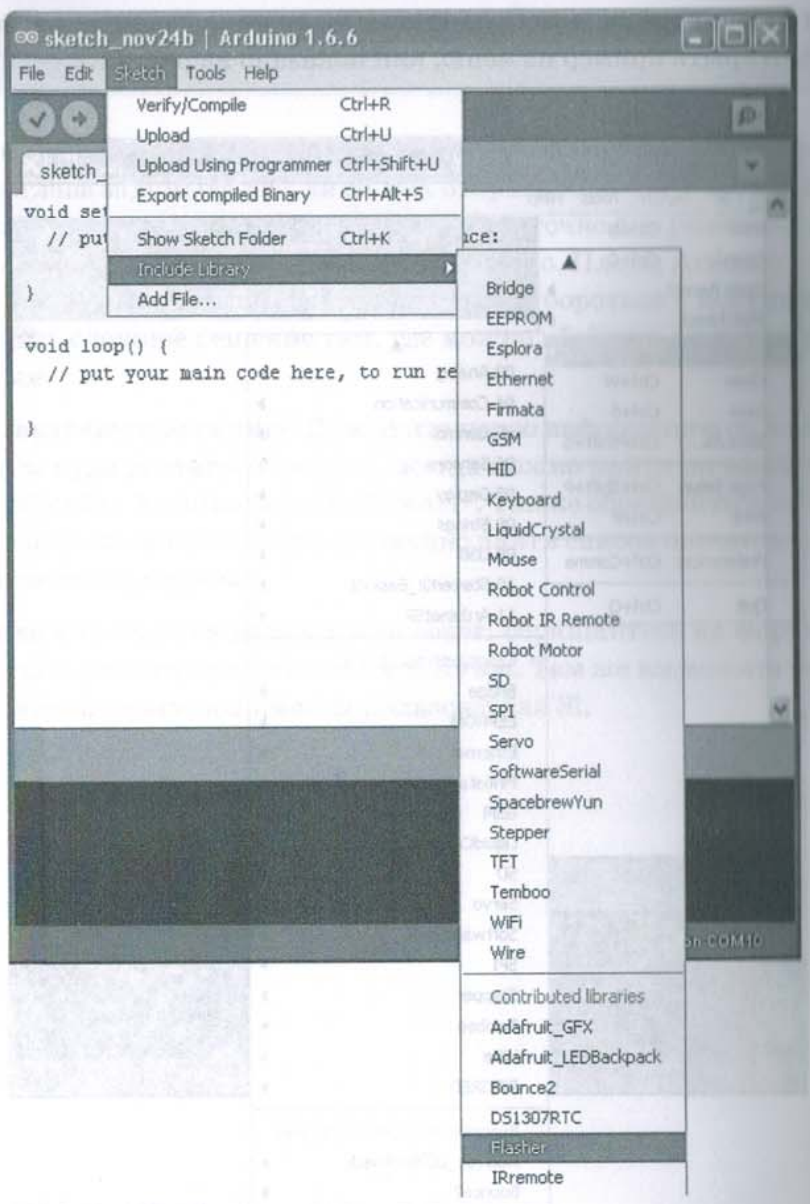


Рис. 11.1. Импортирование библиотеки

Если теперь перезапустить Arduino IDE, должна появиться возможность открыть пример из меню, как показано на рис. 11.2.

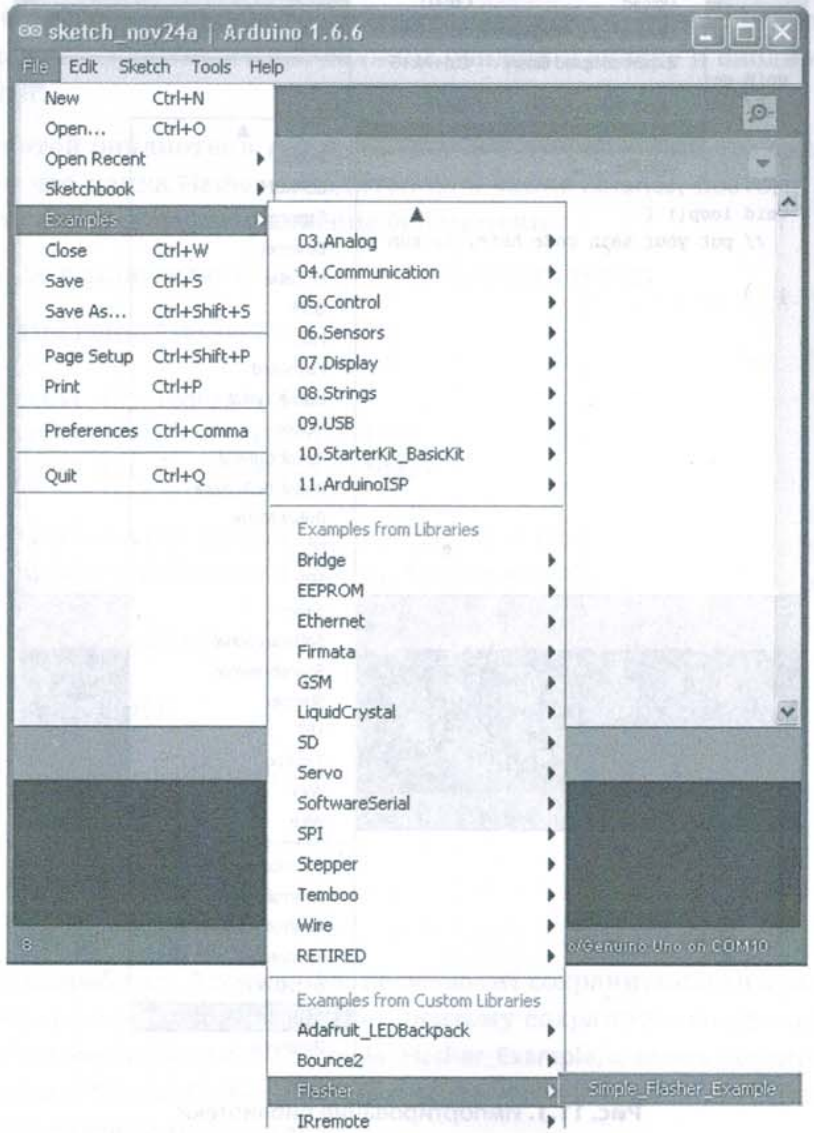


Рис. 11.2. Открытие скетча примера

В заключение

Можно еще многое рассказать о языке C++ и о разработке библиотек, но эта глава задумывалась лишь как отправная точка. Сведений, которые приводятся здесь, должно быть достаточно для решения самого широкого круга задач с помощью Arduino. Платы Arduino — это маленькие устройства, и часто приходится бороться с искушением написать сложное решение там, где можно обойтись простым и понятным.

Эта глава завершает книгу. Дополнительную информацию об Arduino и о том, куда двигаться дальше, всегда можно найти на официальном веб-сайте Arduino: *www.arduino.cc*[1]. Также обращайтесь на сайт книги *www.arduinobook.com*, где можно найти список опечаток и другие полезные ресурсы.

Если вам требуются помощь или совет, обращайтесь на форум сообщества Arduino *www.arduino.com/forum*. Там же вы можете найти автора этой книги под именем пользователя Si.

[1] Существует великолепный сайт на русском языке *http://arduino.ru/*. — *Прим. пер.*

С. Монк

Программируем Arduino: Основы работы со скетчами

2-е издание

Перевел на русский *А. Киселев*

Заведующая редакцией	*Ю. Сергиенко*
Ведущий редактор	*Н. Римицан*
Художник	*С. Маликова*
Корректор	*Н. Витько*
Верстка	*Н. Лукьянова*

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), 3, литер А, пом. 7Н.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12.000 — Книги печатные профессиональные, технические и научные.

Подписано в печать 13.10.16. Формат 70×100/16. Бумага писчая. Усл. п. л. 16,770. Тираж 1000. Заказ 6961.

Отпечатано в АО «Первая Образцовая типография». Филиал «Чеховский Печатный Двор».
142300, Московская область, г. Чехов, ул. Полиграфистов, 1.
Сайт: www.chpd.ru. E-mail: sales@chpd.ru
Телефон: (499) 270-73-59

Программируем Arduino

2-е издание

Познакомьтесь с обновленной версией легендарного бестселлера Саймона Монка. Это издание представляет собой полностью обновленную книгу, основанную на Arduino 1.6.

С момента выхода первого издания многое изменилось: появились новые платы и устройства, использующие язык Arduino. Научитесь в полной мере использовать все возможности Arduino и познакомьтесь с его использованием в проектах Internet of Things. Хотите создать умный дом или запрограммировать робота? Нет ничего проще. Саймон Монк не только поможет разобраться с проволочками, контактами и датчиками, но и покажет, как заставить все это хитросплетение проводов и плат делать то, что вам нужно.

Arduino — это не так сложно, как кажется с первого взгляда. Вы сразу будете покорены открывающимися возможностями.

ПИТЕР®

Заказ книг:
Санкт-Петербург
тел.: (812) 703-73-74,
postbook@piter.com

www.piter.com — каталог книг и интернет-магазин

#питер25

ISBN: 978-5-496-02562-1